AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

95/1109

Université Henri Poincaré – Nancy I

Département de Formation Doctorale en Informatique — École Doctorale IAE + M

SC N 95/182 B

Déduction avec Sortes Ordonnées et Égalités

THÈSE

présentée et soutenue publiquement le 3 octobre 1995

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy I

(Spécialité Informatique)

par

Claus Hintermeier

Composition du jury

Président :	Hubert Comon
Rapporteurs :	Adam Cichon
	Peter D. Mosses
	Fernando Orejas
Examinateurs :	Hélène Kirchner
	Claude Kirchner

Centre de Recherche en Informatique de Nancy
INRIA–Lorraine

Résumé

Cette thèse est dédiée au développement de langages de spécification formels, de leur sémantique opérationnelle sous forme de systèmes de réécriture décorés et des outils de preuves associés. Nous définissons les logiques R^n et G^n, qui sont des logiques à clauses de Horn égalitaires avec prédicat d'appartenance, adaptées à la spécification et à la programmation polymorphique d'ordre supérieur avec sous-typage dynamique et paramétrique, des fonctions strictes et partielles et une sémantique initiale basée sur la théorie des ensembles. Puis, nous introduisons les termes décorés, qui représentent une structure de données bien adaptée aux spécifications et programmes équationnels typés. Ils permettent d'intégrer le raisonnement égalitaire, les fonctions partielles et le typage dynamique de termes.

Mots-clés: spécifications formelles, logique équationelle, théorie ensembliste, sortes ordonnées, sous-typage dynamique et paramétré, fonctions partielles strictes, réécriture, contraintes, complétion, calculs de superposition, prouveurs de théorèmes automatiques.

Abstract

This thesis is dedicated to the development of formal specification languages, their operational semantics in form of typed term rewriting systems and associated proof tools. We define R^n- and G^n- logic, which are equational Horn clause logics with a membership predicate and which are adequate for specification and polymorphic higher-order programming with dynamic, parametric subtyping, strict partial functions and an initial semantics based on set theory. Then, we introduce decorated terms, which represent a data structure which is well-adapted for typed equational specifications and programs. They allow to integrate equational reasoning, partial functions and dynamic typing of terms.

Remerciements

Merci à tous.

À Pauline.

Table des matières

Résumé i

Abstract ii

Introduction

1 Motivation et Contexte Général 1
2 Présentation Détaillée de la Thèse 6
2.1 Les Types Abstraits de Données et les Sortes Ordonnées 6
2.2 La Réécriture 9
2.3 Les logiques R^n et G^n 10
2.4 Superposition Basique pour les Logiques R^n et G^n 15
2.5 Une Théorie Equationelle pour les Termes Décorés 17
2.6 Une Logique Equationnelle à Clauses de Horn avec Termes Décorés 21
3 Domaines et Travaux Reliés 26
3.1 La Théorie des Ensembles 26
3.2 La Théorie des Types 28
3.3 Sous-Typage et Types Dynamiques dans la Programmation Fonctionnelle 29
3.4 Sous-Typage et Structures/Classes dans la Programmation Logique . . . 33
3.5 Méthodes Formelles de Développement 35
4 Plan 36
5 Publications et Présentation du Contenu 38

Partie I Foundations **39**

2 Logical Foundations **41**
1 G-Algebra 44
1.1 Formulas and Presentations 45
1.2 Deduction 46

2 $\mathcal{OSE}$-Logics 46
2.1 Syntax and Semantics 47
2.2 Substitutions, Non-Empty Sorts and Deduction 51
2.3 Initial Model 55
3 R^n-Logics 57
3.1 Models and Axioms 58
3.2 Initial Model Construction 62
3.3 Discussion 66
3.4 Relation To Other Work 69
4 G^n-Logics 70
4.1 Models and Axioms 70
4.2 Deduction in G^n-logics 71
4.3 The Initial Model 75
4.4 Discussion 79
4.5 Conclusion from R^n- and G^n-logics 79

3 Term Rewriting **81**
1 Introduction 81
2 Term Rewriting Relations and Properties 83
2.1 Unsorted Terms and Rewriting 83
2.2 Order-Sorted Rewriting with Syntactic Sorts 85
2.3 Conditional Term Rewriting 86
2.4 Strongly Deterministic Term Rewriting Systems 87
3 The Formal Expressiveness of Term Rewriting 89
3.1 Innermost by Order-Sorted Rewriting 89
3.2 The Expressiveness of SDTRSs 94
3.3 Outline of a Transformation for SDTRSs 95

4 Clausal Theorem Proving with Equality **99**
1 Introduction 99
1.1 Syntax and Semantics for Clausal Theorem Provers with Constraints . . . 102
1.2 General Notions and Lemmas 103
1.3 Basic Superposition with Equational Constraint 106
2 Reduction of R^n-/G^n-to $\mathcal{OSE}$-Logics 110
2.1 Eliminating the Set Theoretic Inference Rules of R^n-Logics 110
2.2 Eliminating the Set Theoretic Inference Rules of G^n-Logics 111
2.3 An EX - Elimination Theorem 111

2.4 From Predicates to Propositional Functions 115
3 Clausal Theorem Proving for R^n-and G^n-Logic 116
3.1 A Saturation Calculus for R^n-Logics 116
3.2 A Saturation Calculus for G^n-Logics 118
3.3 Adding Assertions 124

Partie II Operationalisation **131**

5 Unconditional Decorated Term Algebra **139**
1 Sort Membership 139
1.1 Associating a Partial Ordering over Sorts 139
1.2 Assumptions Concerning the Sort Membership 141
1.3 Eliminating cycles 142
1.4 Non-empty sorts 144
1.5 Sort completion 145
2 Decorated Terms 147
2.1 The Term Structure 147
2.2 Subsumption for Decorated Terms 150
2.3 Substitutions 150
3 A Restricted Version of Semantical Order-Sorted Matching 152
3.1 Generalities 152
3.2 The Algorithm 153
4 Strict Decorated Unification in Sort Inheriting Presentations 156
4.1 Generalities 157
4.2 The Algorithm 159
5 Subterm Conservative Solutions 165

6 Rewriting Decorated Terms **167**
1 Decorated Term Rewriting Systems 167
1.1 Decorated Equalities 167
1.2 Decorated Rewriting 168
1.3 Decoration Rewriting 168
2 Elementary Properties of Rewriting 169
3 Converting Presentations to Decorated TRSs 173
4 Decorated Orderings 175
5 A Birkhoff Theorem for G-Algebra 175

7 Decorated Completion in Sort Inheriting Presentations **181**
1 Confluence and Critical Pairs 182
1.1 $\mathcal{T}$-Confluence 182
1.2 Definition and Properties of Critical Pairs 184
1.3 Overlapping Computations 186
2 Transformation Rules for Order-Sorted Completion 191
3 Proof Reduction and Reflection 197
4 Fairness 200
5 Changing the Subsort Relation 201

8 Checking Sort Inheritance **203**
1 Testing Sort Inheritance On D-Closed Sets 204
2 Sort Inheritance on Valid Decorated Terms 208
2.1 Subterm Conservation and Typing Proofs 209
2.2 Flat and Linear Term Declarations 210
2.3 Flat Term Declarations 213
2.4 General Term Declarations 214
3 Comparison 231
4 Changing the Membership Relation 232

9 Related Work, Discussion and Conclusion **235**
1 Syntactic Sort Approaches 235
1.1 Sort Inheritance vs. Regularity 235
1.2 Retracts are Superfluous 237
1.3 Subsumption of Sort-Decreasing Rules Approach 237
1.4 The Signature Extension Approach 238
1.5 Other Syntactic Sort Approaches 241
2 Semantic Sort Approaches 241
2.1 The Tree Automata Approach 241
2.2 The T-contact Method 246
2.3 Other Semantic Sort Approaches 248
3 Conclusion from Decorated Completion 248

10 A Decorated Superposition Calculus **251**
1 Subsets, Sort Inheritance and Decorated Terms 253
2 Decorated Clause Transformation 255
3 The Initial Presentation, or How to Test Sort Inheritance 257
4 Decorated Unification and Saturation 260

5 Decorated Rewriting 275
6 Completeness for the SI-test 285
7 Looking for Standard Elements in Sets 286
8 Examples 288
9 Conclusion from Decorated Clause Theorem Proving with Constraints 294

11 Conclusion **297**

A Proof Transformations **299**
1 Transformation Rules for Completeness of $\mathcal{OSC}$ Completion 300
2 Transformation Rules for MSS Rewriting 301

B Testing SI for Flat Term Declarations **303**

C Calculi, Transformations and Symbols **313**
1 List of Calculi and Formula Transformations 313
1.1 Logic Calculi 313
1.2 Superposition Calculi 313
1.3 Formula Transformations 314
2 List of Used Symbols 314
2.1 Model/Proof Theoretic Relations 314
2.2 Orderings 314
2.3 Arrows 314
2.4 Equation Symbols 315
2.5 Matching/Unification Equations 315
2.6 Constraints 315

Index **317**

Bibliography **321**

Table des figures

1.1 La Structure de la Thèse 5
1.2 La Structure des algèbres G 9
1.3 La Structure de la Logique R^n 13
1.4 Le Développement de la Théorie des Ensembles au Début du Siècle 27
1.5 Le Graphe de Dépendance de cette Thèse 37

2.1 **GA** : Deduction rules for G-algebra 47
2.2 **OSL** : Horn Clause Deduction Rules for $\mathcal{OSE}$-logics 52
2.3 **NES** : Inference Rules To Decide Non-Emptiness of Sorts 52
2.4 The Sort Structure of R^n-signatures 59
2.5 R^n-logic deduction rules to be added to **OSL** 62
2.6 **ET** : The **ETL** Calculus 67
2.7 **UAA** : Unified Algebra Axioms 68
2.8 **GNL** : Horn Clause Deduction Rules for G^n-logics 72
2.9 Inference Rule For a Sufficient Criterion of Non-Emptiness of Sorts in G^n-logics . 75

3.1 **MATCH** : Matching using Transformation Rules 84

4.1 **Dag – Unify** : A constraint solver for unsorted unification 107
4.2 A constraint solver for order-sorted unification 118

5.1 Two non-regular but sort inheriting sort structures 142
5.2 Rule to search syntactically empty sorts 144
5.3 Rules for Strict, Syntactic, Decorated Matching 153
5.4 Transformation Rules of $\mathbf{UNIF_d}$ 159
5.5 Failure Rules of $\mathbf{UNIF_d}$ 160
5.6 sketch of the termination proof of $\mathbf{UNIF_d}$ 161
5.7 Variable elimination rule for completeness proof 164

6.1 Extraction of decorated/decoration rewrite rules from the Presentation 173

7.1 The Newman Lemma for Decorated Rewriting 183
7.2 $\mathcal{OSC}$ The completion rules for decorated terms. 192
7.3 Test for Equivalence of $\leq_S^{syn}$ and $\leq_S^{sem}$. 201

8.1 Sort Inheritance Test Rule. 203
8.2 Bottom-up Layer Rewriting. 215
8.3 The Proof Cases for Extended Type Propagation 220
8.4 No Overlap Case 222

8.5 Variable Overlap Case 223
8.6 Critical Overlap Case 225
8.7 $\mathcal{MSSC}$: Specific Completion Rules for Maximally Subterm Sharing Rewriting. . 227
8.8 A Sample MSSC Strategy 228

10.1 Clause (Set) Transformations in this Chapter 260
10.2 **DU** : A constraint solver for decorated unification 263
10.3 Rules for Decorated Matching with Subset Constraints 277

B.1 No Overlap Case 306
B.2 Critical Overlap Case 308
B.3 Variable Overlap Case, Simple Part 310

Introduction

1 Motivation et Contexte Général

Cette thèse se situe dans le cadre général des techniques formelles de développement de logiciels, en utilisant des types de données basés sur la théorie des ensembles. Un problème clé pour la spécification et la programmation est la conception d'outils de preuve établissant un lien entre spécifications et programmes ou plus précisément entre les théories sous-jacentes. Le contexte général dans lequel nous plaçons notre travail est développé dans ce qui suit. Nous abordons ensuite les paradigmes de langages de spécification et de programmation que nous intégrons dans ce travail, et formulons les thèses défendues dans ce document.

Un contexte de développement de logiciels sûrs. Le développement de logiciels dans l'industrie des années 1990 se fait dans la plupart des cas dans des cycles de spécification (mise en œuvre d'un cahier de charges ou d'un plan de développement), codage (programmation) et test (non-exhaustif). Afin de donner des garanties de correction du logiciel, la recherche en informatique vise à produire des techniques de vérification formelle des programmes par rapport à leur spécifications. Ces techniques remplacent ou complétent en général la phase de test.

Cet effort est nécessaire, en particulier dans les domaines d'application critiques pour la sécurité, comme les auto-pilotes pour l'aviation ou la technologie des centrales nucléaires. On peut constater aussi un effort au niveau de la standardisation internationale afin de donner des normes de sécurité aux logiciels dans le cadre de la série ISO 9000. Cela nous fait croire qu'il pourrait y avoir, dans un futur proche, des labels attachés aux logiciels estimant leur sûreté d'utilisation. De tels labels pourraient être exigés dans certains domaines d'application comme c'est déjà le cas, par exemple, pour l'équipement d'aviation.

A partir d'une spécification `spec` et d'un programme `prog`, les problèmes typiques que pose la vérification de logiciels sont les preuves des assertions suivantes :

- `prog` termine.
- `prog` est non-ambiguë.
- `prog` satisfait `spec`.

Nous ne nous attacherons pas dans cette thèse au dernier point qui implique en général des preuves de théorèmes par récurrence traitées par exemple dans le thèse de A. Bouhoula [Bouhoula, 1994].

Cette thèse contribue au premier et au deuxième problèmes dans le sens où elle donne la possibilité de produire des programmes terminants et non-ambigus, qui sont écrits dans une logique équationnelle de clauses de Horn avec un prédicat d'appartenance. Les méthodes de

construction sont basées sur la technique de complétion de Knuth-Bendix : elle produit des programmes dans lesquels les opérations sont définies par des équations (conditionnelles) orientées. L'évaluation d'une expression donne toujours le même résultat après un nombre fini d'étapes.

Paradigmes de langages de spécification et de programmation. La procédure originale de Knuth et Bendix [Knuth et Bendix, 1970] est décrite pour les théories équationnelles du premier ordre, puis, l'idée a été étendue pour des théories clausales du premier ordre. La contribution principale de cette thèse consiste en l'extension de cette procédure à des théories admettant un typage expressif proche des langages de programmation impératifs. Ceci nécessite que les aspects comme le polymorphisme, la paramétrisation et l'héritage par sous-typage soient traités d'une manière appropriée.

Le concept de **Polymorphisme** signifie qu'une fonction peut être surchargée, dans le sens où son axiomatisation est donnée pour plusieurs types d'arguments, comme le montre l'exemple suivant :

Exemple 1.1.1 *Supposons que nous avons les types* `Tas`, `Pile` *et* `Liste`. *L'opérateur* `taille`, *qui détermine le nombre d'éléments dans des structures de ces différents types, rend un nombre naturel (de type* `Nat`*). Il est typiquement surchargé, c'est-à-dire qu'il a les profils suivants :*

```
op  taille : Tas -> Nat
op  taille : Pile -> Nat
op  taille : Liste -> Nat
```

Remarquons que `Tas`, `Pile` *et* `Liste` *sont typiquement supposés de ne pas avoir d'élément commun.*

La **Paramétrisation** permet d'élargir le concept de types qui deviennent des termes et offrent la possibilité de schématiser certaines axiomatisations de fonctions. Conjointement avec le polymorphisme, ceci nous donne une base syntaxique pour distinguer entre des différentes sortes de listes, comme dans l'exemple suivant :

Exemple 1.1.2 *Supposons que* `X` *est un paramètre de type, et* `Liste` *est un constructeur de types. Soit* `1` *une constante de type* `Nat`, `tt` *de type* `Bool` *et* `cons` *un constructeur binaire de listes. Nous pouvons alors donner les jugements de typage suivants :*

```
ax  nil ∈ Liste(X)
ax  cons(1,nil) ∈ Liste(Nat)
ax  cons(tt,nil) ∈ Liste(Bool)
ax  cons(cons(1,nil),nil) ∈ Liste(Liste(Nat))
```

L'héritage est un principe lié au *sous-typage*, qui permet de réutiliser des fonctions définies sur un type plus général pour des types plus spécifiques. Utilisé avec le polymorphisme paramétré ce principe donne beaucoup d'expressivité aux langages de programmation, mais est difficile à traiter par les systèmes de typage. Il y a plusieurs possibilités pour restreindre le sous-typage sur les types paramétrés, de telle sorte que les trois paradigmes mentionnés soient compatibles. Continuons avec l'exemple ci-dessus :

Exemple 1.1.3 *Supposons que* `Nat` *soit un sous-type de* `Ent`, *ce qui s'écrit* `Nat <Ent`, *et que* `Liste` *soit un type paramétré pour les listes d'éléments du type donné comme argument. Supposons que* `Liste` *soit monotone par rapport aux inclusions de type.*

Une fonction `somme` *qui donne la somme de tous les éléments d'une liste de nombres entiers (type* `Ent`*) est aussi bien applicable aux listes de nombres naturels (type* `Nat`*).*

L'inférence de type statique, c'est-à-dire indépendante des équations, en présence de sous-typage, est souvent en conflit avec le raisonnement équationnel, car le type d'un terme, déterminé par l'inférence de type statique, peut être différent de celui du résultat après l'application d'une équation. Donc, nous choisissons *l'inférence de types dynamique*, ce qui veut dire dans notre contexte, que le typage dépend du raisonnement équationnel. L'arithmétique est riche de tels exemples :

Exemple 1.1.4 *Soient* `Nat` *le type des nombres naturels,* `Ent` *le type de tous les nombres entiers relatifs,* `carré` *la fonction qui calcule le carré d'un nombre entier,* `fois` *la multiplication sur les nombres entiers relatifs.*

La fonction `carré(x)` *devrait être du type* `Nat` *pour tous les nombres entiers* `x`*, mais la définition usuelle* `carré(x) = fois(x,x)` *le rend égal à un terme du type* `Ent`*. Avec le typage dynamique, nous pouvons déduire que le terme* `fois(x,x)` *est de type* `Nat`*, car il est égal à un terme de ce type.*

Le **typage dynamique** nous donne la possibilité de travailler de manière consistante avec une sémantique basée sur la théorie des ensembles pour les types : des termes égaux appartiennent aux mêmes types, comme pour l'appartenance aux ensembles.

Mais ceci n'est pas le seul obstacle à franchir. Dans le domaine de la programmation, nous avons typiquement des fonctions qui sont indéfinies pour certaines valeurs, ce qu'on appelle des **fonctions partielles**. À nouveau, l'arithmétique nous donne un exemple :

Exemple 1.1.5 *Soient* `Ent` *le type des nombres entiers relatifs,* `div` *l'opération de division associée et* `1,0` *les nombres zéro et un standards. Alors,* `div(1,0)` *n'est pas défini, c'est-à-dire* `div` *est une fonction partielle dans* `Ent`*.*

Il est donc très intéressant de considérer des fonctions partielles dans notre cadre formel. Néanmoins, nous ne considérons pas la théorie entière des fonctions partielles dans le domaine de la programmation, car ceci inclut aussi des fonctions non-strictes (qui permettent d'ignorer des arguments indéfinis). Un exemple typique de fonction non-stricte est la fonction à trois arguments si-alors-sinon. Au lieu des fonctions non-strictes nous utilisons des égalités conditionnelles, comme l'illustre l'exemple suivant :

Exemple 1.1.6 *Soient* `x` *un nombre entier relatif standard (type* `Ent`*),* `racine` *la fonction qui donne la racine carrée d'un entier relatif positif, c'est-à-dire le plus grand nombre naturel dont le carré est plus petit ou égal à l'argument.* `racine` *est indéfinie sur les entiers négatifs. Soit* $-$ *l'opération unaire standard et* $<, \leq$ *les ordres standards sur les nombres entiers. Si nous voulons définir l'opération* `f` *prenant un argument* `x` *comme* `racine(x)` *si* `x` $\geq$ `0` *et* `racine(-x)` *si* `x` $<$ `0`*, alors on pourrait penser à donner les équations suivantes, où* `si-alors-sinon` *est une fonction stricte,* `vv` *représente «vrai» et* `ff` *«faux» :*

`si vv alors x sinon y` $= x$,
`si ff alors x sinon y` $= y$,
`f(x)` $=$ (`si (x` $\geq$ `0) alors racine(x) sinon racine(`$-x$`)`).

Malheureusement, cette définition ne nous donne pas le résultat souhaité, car l'opération `si-alors-sinon` *est stricte. Donc, si* `x` $\geq$ `0`*, on obtient le terme* `si vv alors racine(x) sinon racine(-x)` *qui n'est pas égal à* `racine(x)`*, car la première équation pour* `si-alors-sinon` *n'est pas applicable. Il faut remarquer que* `racine(-x)` *n'est pas défini et donc on ne peut pas instancier* `y` *avec ce terme.*

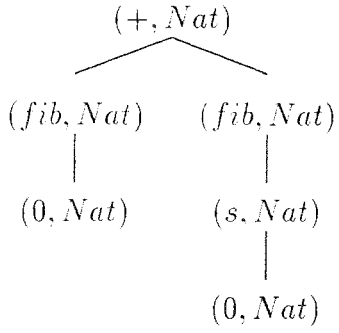
Mais nous pouvons prendre comme alternative une définition conditionnelle, où «si» fait partie du langage formel:

$$\texttt{f(x)} = \texttt{racine(x)} \quad si\ \texttt{x} \geq \texttt{0}$$
$$\texttt{f(x)} = \texttt{racine(}-\texttt{x)} \quad si\ \texttt{x} < \texttt{0}$$

Nos contributions. Dans cette thèse, nous considérons plusieurs théories différentes: premièrement, un cadre équationnel, c'est-à-dire sans conditions, mais avec (sous-)typage sur un nombre fini de constantes de type et des fonctions polymorphiques et partielles. Ceci est le cadre des algèbres G, développé par A. Mégrelis dans [Mégrelis, 1990]. Afin d'intégrer des types paramétrés et des formules avec conditions, nous passons ensuite à une théorie de clauses de Horn avec un prédicat d'appartenance, appelée logique R^n dans le cas où toutes les fonctions sont totales, et logique G^n au cas où il y a des fonctions partielles. Ces deux théories sont basées sur une logique de clauses de Horn équationnelle ordo-sortée, appelée $\mathcal{OSE}$-logique, à cause des sortes ordonnées utilisées dans le système de types. Les deux logiques, R^n et G^n, admettent des équations sur des individus et des ensembles, en même temps que des déclarations d'appartenance. Toutes les théories mentionnées ont des modèles initiaux, qui donnent une sémantique à n'importe quel ensemble de formules syntaxiquement valides. Cette propriété, utilisée dans l'approche des types abstraits de données (ADTs), ainsi que les exemples donnés ci-dessus, nous conduisent à formuler la thèse suivante:

Thèse 1 *Les logiques R^n et G^n sont adaptées à la spécification et à la programmation polymorphique d'ordre supérieur avec du sous-typage dynamique et paramétrique, des fonctions partielles et une sémantique claire basée sur la théorie des ensembles.*

Des études plus approfondies menées dans la partie principale de cette thèse nous conduisent à la conclusion qu'il y a une différence significative entre le codage des types comme conditions d'appartenance et l'utilisation de variables typées, comme cela a déjà été observé dans le domaine de la déduction automatique (voir l'exemple de «Schubert's Steamroller» [Walther, 1985]). Cet effet est même amplifié par une représentation de termes étendue contenant de l'information sur le typage (appelée *décorations*) localement dans chaque nœud d'un terme. Cette représentation est proposée et étudiée en détail. Le mécanisme effectuant le typage, c'est-à-dire qui ajoute de l'information aux décorations, devient une partie du calcul au même niveau que le raisonnement équationnel. Donc, un terme $fib(0) + fib(s(0))$, où fib représente la fonction de Fibonacci, peut être étendue de la manière suivante:



Une décoration D d'un terme t est écrit $t^{:D}$ dans ce qui suit. Dans l'exemple ci-dessus, nous écrivons:

$$+(fib(0^{:\{Nat\}})^{:\{Nat\}}, fib(s(0^{:\{Nat\}})^{:\{Nat\}})^{:\{Nat\}})^{:\{Nat\}}$$

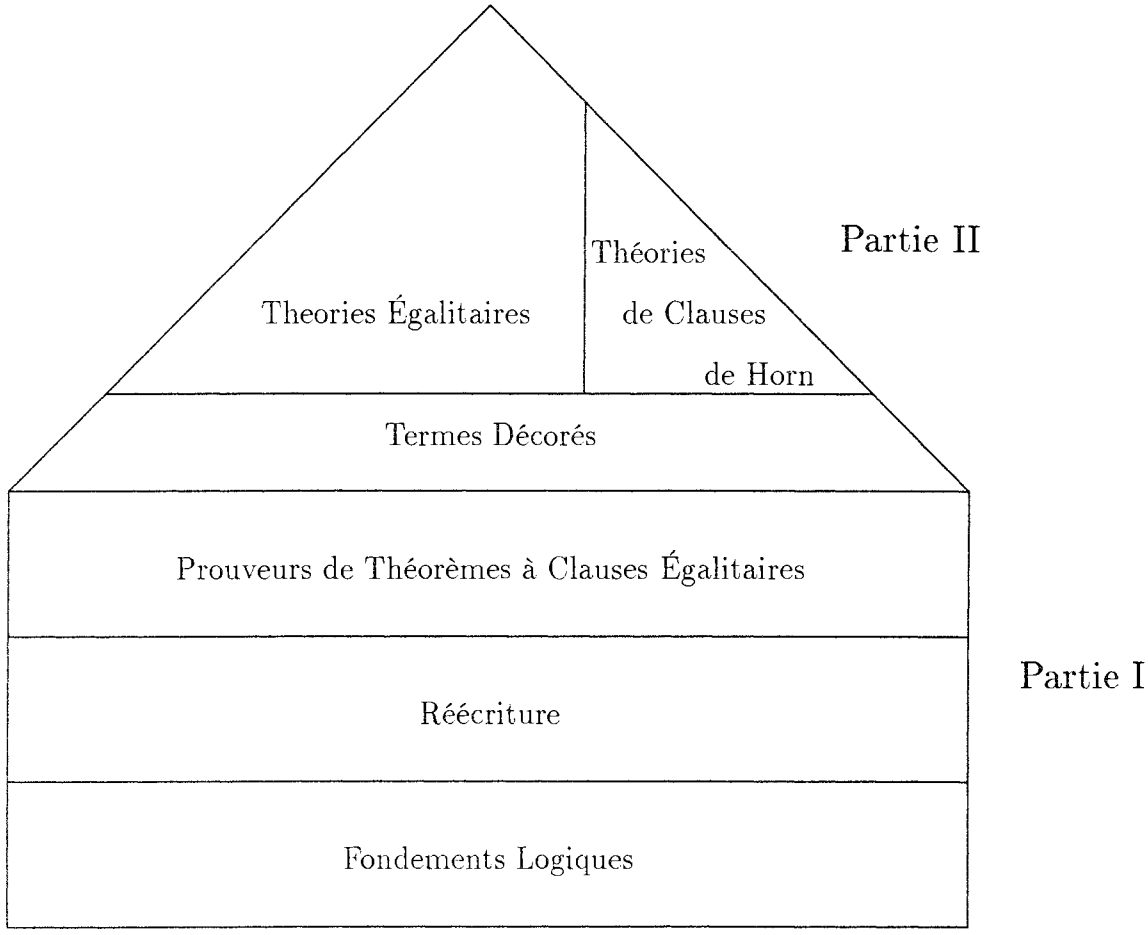


FIG. 1.1 – *La Structure de la Thèse*

Les conditions de typage insatisfiables sont détectées par le test d'une propriété appelée *héritage de sortes* (sort inheritance), qui stipule qu'un terme ne peut appartenir simultanément à tous les éléments d'un ensemble fini de types que s'il existe un type inclus dans tous ces éléments. En plus, cette propriété garantit la décidabilité de l'unification de variables typées. Au contraire, une approche naïve sans variable typée mène à un ensemble infini de buts à résoudre lorsque l'on cherche à tester si la propriété est satisfaite. En résumé, nous pouvons formuler la deuxième thèse principale défendue dans ce document :

Thèse 2 *Les termes décorés représentent une structure de données bien adaptées pour les spécifications et programmes équationnels typés. Ils permettent d'intégrer le raisonnement égalitaire, les fonctions partielles et le typage dynamique de termes.*

Conformément aux deux thèses formulées ci-dessus, le document est structuré en deux parties (voir figure 1.1): premièrement, nous travaillons sur les fondements théoriques, incluant les bases logiques et des techniques générales de déduction comme la réécriture et la superposition basique. Ensuite, nous abordons les aspects opérationnels, en construisant des programmes exécutables sous forme de systèmes de réécriture décorés.

2 Présentation Détaillée de la Thèse

Dans cette section nous rappelons les concepts de base utilisés, avant de présenter les résultats obtenus. Nous présentons succinctement les types abstraits de données basés sur des logiques équationnelles ordo-sortées, puis l'algèbre G, introduite dans la thèse d'A. Mégrelis [Mégrelis, 1990], qui est sous-jacente à nos études concernant les théories algébriques des termes décorés, et qui nous a inspiré pour la conception de la logique G^n. Ensuite, les notions de base de la réécriture sont évoquées. Ce survol inclut la complétion de théories égalitaires comme procédure standard menant à la non-ambiguïté du résultat d'une fonction définie par un ensemble d'équations.

Puis, nous passons en revue les principaux résultats de cette thèse. Nous introduisons les logiques R^n et G^n et donnons les théorèmes principaux concernant un calcul réfutationnellement complet de superposition basique pour ces théories. Ensuite, nous présentons nos résultats concernant l'utilisation de termes décorés dans des théories équationnelles, en utilisant la sémantique de l'algèbre G, avec le but de réaliser une complétion équationnelle qui nous permet de construire des programmes non-ambigus et terminants. Finalement, nous montrons des résultats pour un calcul similaire, mais avec la sémantique de la G^1-logique.

2.1 Les Types Abstraits de Données et les Sortes Ordonnées

Basées sur le principe de remplacement d'égal par égal et la théorie des modèles très riche de l'algèbre, la notion des types abstraits de données à été introduite au milieu des années soixante-dix [Thatcher et Wagner, 1976; Goguen *et al.*, 1978] comme une approche à la spécification formelle. Le point fort des types abstraits de données est l'existence de modèles initiaux, dont nous rappelons les principales propriétés.

Les sortes ordonnées sont un moyen de structurer le domaine de modèles. Ceci permet de donner des axiomes plus concis, par exemple dans le domaine de l'arithmétique, et la possibilité de réutilisation de spécifications et programmes, comme par l'héritage multiple dans les approches orientées objets. Une bonne partie de cette thèse est basée sur une algèbre à sortes ordonnées, l'algèbre G développée par A. Megrélis, qui sera présentée brièvement après les sortes ordonnées en général.

Modèles Initiaux

Une exigence clé pour les types abstraits de données est l'existence d'un modèle initial dans la catégorie de tous les modèles d'une spécification. Dans le cas de modèles de Herbrand, ceci correspond à l'existence de modèles minimaux. Cette propriété est garantie par les théories de clauses de Horn équationnelles et nous permet d'imposer cette classe de modèles comme sémantique standard, contournant ainsi l'utilisation de la négation, en postulant que deux termes qui ne peuvent pas être prouvés égaux, doivent être inégaux. Ce principe est illustré par l'exemple suivant :

Exemple 1.2.1 *Considérons* `0` *et* `s` *comme le zéro et la fonction de succession sur les nombres naturels. Le modèle initial pour ces deux déclarations est* $\{\mathtt{s}^n(\mathtt{0}) \mid n > 0\}$, *c'est-à-dire les nombres naturels.*

En ajoutant `plus` *comme opérateur et les équations suivantes :*

$$\mathtt{plus(s(x),y)} = \mathtt{s(plus(x,y))},$$
$$\mathtt{plus(0,x)} = \mathtt{x},$$

on ne change pas le domaine du modèle initial, à condition de choisir les bons représentants des classes d'équivalence. L'opérateur `plus` *est alors interprété comme l'addition des nombres naturels.*

Cet exemple montre très bien la puissance de la sémantique initiale: il n'y a pas besoin d'ajouter des formules contenant des négations afin de distinguer entre `0` et `s(x)`. Cet absence de négation, qui nous permet de raisonner d'une manière efficace par rapport aux théories incluant la négation, est, avec la théorie des modèles correspondante, la raison pour laquelle nous adoptons la sémantique initiale. Pourtant, quand nous arrivons aux types et aux fonctions d'ordre supérieur, nous obtenons un comportement moins agréable de la sémantique initiale:

Exemple 1.2.2 *Considérons les formules* `a ∈ B` *et* `a ∈ C`. *En utilisant la sémantique initiale standard, on peut dériver* `B = C`.

Il faut remarquer que ceci n'est pas voulu lorsque `B,C` sont des types non inclus l'un dans l'autre. Quand un programmeur donne les mêmes éléments pour deux types incomparables, alors soit c'est par erreur, soit son intention est soit d'ajouter d'autres éléments en réutilisant la même spécification dans un autre contexte. Donc, il ne devrait pas être possible d'utiliser `B = C` comme théorème.

De façon similaire, dans le contexte de fonctions d'ordre supérieur, nous pourrions avoir deux fonctions avec la même fonctionnalité que nous voulons distinguer. Cette distinction est nécessaire par exemple pour donner une sémantique aux pointeurs de fonction dans des langages impératifs comme C dans le contexte de la vérification de programmes écrits dans ce langage.

Ceci nous amène à utiliser des modèles non-standards pour les spécifications d'ordre supérieur. L'idée sous-jacente est d'admettre des individus ou des ensembles non représentés par des termes dans le domaine d'interprétation des formules. Ceci implique que `B = C` n'est plus vrai dans tous les modèles de `a ∈ B` et `a ∈ C`, car il peut y avoir d'autres éléments que `a` dans `B` ainsi que d'autres dans `C`. Cette notion de modèle est une adaptation de celle de l'algèbre universelle dans le contexte de l'ordre supérieur.

Sortes Ordonnées

Au lieu de parler de types, nous parlons de sortes comme dans la logique traditionnelle. L'utilisation différente de *type* et *sorte* semble être ancré dans le fait que les sortes ont une sémantique basée sur des ensembles, alors que les types sont utilisés principalement comme un moyen pour exprimer des restrictions syntaxiques, bien que la théorie des modèles des λ-calculs typés ait très tôt commencé à interpréter les types dans des cadres formels basés sur la théorie des ensembles. Selon Russell [Russell, 1908], une variable représentant des individus est de type 0; une fonction propositionnelle prenant seulement des variables de type 0 est de type 1; une variable représentant une telle fonction propositionnelle est donc aussi de type 1 et ainsi de suite. Finalement, l'ordre d'une formule, appelé aussi type logique, est défini comme le plus grand des types de toutes les variables dans la formule.

La logique équationnelle ordo-sortée à été définie par A. Oberschelp [Oberschelp, 1962], comme un moyen naturel de s'exprimer dans le domaine des mathématiques. Après la logique multi-sortée initiée par Herbrand [Herbrand, 1930; Herbrand, 1971] et complétée par Schmidt [Schmidt, 1938], où le domaine d'un modèle est divisé en des ensembles disjoints, appelés sortes, les théories ordo-sortées se distinguent par le fait d'admettre un ordre sur ces ensembles. Intuitivement, ceci donne plus d'expressivité, bien qu'on puisse facilement prouver que formellement, on n'en a pas gagné, car toute présentation basée sur la logique ordo-sortée peut-être

transformée dans une présentation équivalente sans sortes [Oberschelp, 1962] (les sortes sont traduites dans des prédicats monadiques).

Dans les années quatre-vingt, les formalismes ordo-sortés ont surgi de nouveau dans le domaine du raisonnement automatique (voir par exemple [Walther, 1983; Schmidt-Schauß, 1987]) et dans les langages de spécification et de programmation ([Goguen et Meseguer, 1992; Kirchner *et al.*, 1988a; Aït-Kaci et Nasr, 1985]). Dans le domaine du raisonnement automatique, le résultat principal est la réduction de l'espace de recherche, grâce à l'unification avec variables sortées [Walther, 1985]. Au niveau des langages, l'avantage principal est l'analogie avec la tradition mathématique – pensons aux nombres naturels, entiers etc. – ainsi qu'une concision et donc une lisibilité augmentées.

Un exemple classique pour une théorie ordo-sortée est l'arithmétique, où on distingue entre des nombres naturels, entiers, rationnels, réels et ainsi de suite, avec des inclusions ensemblistes classiques. La distinction entre les types de données correspondants dans les langages impératifs menait aussitôt à des systèmes de type sophistiqués avec des conversions automatiques d'une représentation dans l'autre. Mais il n'y a pas de raison pour restreindre les inclusions de sortes aux types prédéfinis. La nature même des types abstraits de données suppose qu'on puisse définir de nouveaux types non-prédéfinis.

L'algèbre G

Une grande partie de cette thèse utilise comme sémantique l'algèbre G : une algèbre ordo-sortée avec des fonctions partielles strictes. Une des propriétés principales de l'algèbre G est que les déclarations de fonctions et de termes, qui font partie de la signature (statique) dans les approches classiques, deviennent des formules et sont impliquées dans les preuves aussi bien que les équations. En algèbre G, nous distinguons surtout entre deux types d'unités syntaxiques : les termes construits sur une signature graduée, écrits s, t, u, v, avec éventuellement des indices ou des primes, et un ensemble fini de sortes, écrites en lettres majuscules $A, B, C \ldots$. Ce schéma très simple est illustré par la figure 1.2, où : est le symbole de la relation d'appartenance utilisé par A. Mégrelis.

La différence principale par rapport aux approches purement syntaxiques, comme [Goguen et Meseguer, 1992], consiste donc en la notion de typage. À la place du typage syntaxique, où le fait qu'un terme t appartienne à une sorte A, écrit $t : A$, dépend seulement des déclarations de typage dans la signature, ce jugement dépend dans notre approche (comme dans [Watson et Dick, 1989; Comon, 1992; With, 1992a; Werner, 1993]) aussi des égalités. Ceci correspond donc à l'intuition que des termes égaux appartiennent aux mêmes sortes et est reflété par la «règle de sortes sémantiques» dans le système de déduction de l'algèbre G [Mégrelis, 1990]:

$$\boxed{t : A, t = s \vdash s : A}$$

Les fonctions strictes et partielles sont traitées dans le système de déduction par l'intermédiaire d'un prédicat d'existence EX, interprété comme l'appartenance au domaine de l'interprétation. Intuitivement, les règles de déduction correspondantes disent que tous les sous-termes de théorèmes doivent être bien définis, c'est-à-dire exister comme objet dans le domaine et les applications des fonctions dans un tel théorème doivent être définies.

Nous proposons pour plus de détails concernant les algèbres G, de voir la section 2.1 ou consulter la thèse d'Aristide Mégrelis [Mégrelis, 1990].

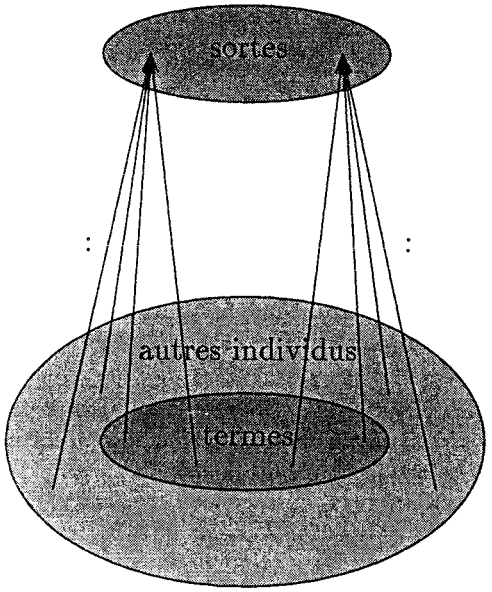


FIG. 1.2 – *La Structure des algèbres G*

2.2 La Réécriture

La réécriture représente le choix naturel pour donner une sémantique opérationnelle pour les théories égalitaires. La β-réduction du λ-calcul ou bien le langage OBJ en sont des exemples. En présence de sortes ordonnées, comme en OBJ-3, la réécriture présente des difficultés spécifiques, qui sont la motivation principale de notre travail au début de la deuxième partie de la thèse.

L'idée de la réécriture consiste en l'application orientée des équations et correspond au principe du remplacement d'égal par égal sans retour en arrière. Évidemment, la congruence sur les termes définie par l'égalité des formes irréductibles (en supposant la terminaison) peut ne pas correspondre avec l'identité dans tous les modèles d'une spécification, comme l'illustre l'exemple suivant :

Exemple 1.2.3 *Soient $a = b$ et $a = c$ les seules égalités dans la spécification. L'orientation des deux égalités de gauche à droite n'admet plus de prouver $b = c$. En plus, a peut être* évalué *en deux résultats différents : b ou c.*

Ceci montre la nécessité d'une propriété supplémentaire à satisfaire par un système de réécriture, c'est-à-dire un ensemble d'équations orientées : la confluence. Un système de réécriture est dit confluent, si deux suites de réductions aléatoires, commençant par le même terme, peuvent être rejointes par l'ajout (optionnel) d'étapes de réduction à leur fin. Ceci nous garantit alors la non-ambiguïté du résultat si toutes les séquences de réduction terminent.

La réécriture joue un rôle essentiel en tant que lien entre les théories équationnelles et leur opérationalisation, car elle permet de décider l'égalité dans une théorie équationnelle représentée par un système de réécriture confluent et terminant. Du point de vue de la programmation, elle nous garantit qu'une implantation par réécriture des fonctions spécifiées correspond exactement à la sémantique de la théorie équationnelle. Pour un survol sur la réécriture voir [Dershowitz et Jouannaud, 1990b].

Etant donnée une théorie équationnelle, il est bien sûr intéressant de savoir comment on peut lui associer un système de réécriture correspondant. Une possibilité est d'orienter les égalités

immédiatement, c'est-à-dire de les utiliser dans une direction spécifique seulement, et d'essayer de prouver ensuite un critère appelé «critère de paires critiques». Il garantit en principe que toutes les ambiguïtés, résultant de deux étapes différentes de réduction simultanées possibles pour le même terme, peuvent être résolues avec le même système d'équations orientées. En réécriture, cette propriété est appelé *confluence locale* et le théorème qui prouve qu'un ensemble de superpositions est suffisant pour garantir la confluence locale est appelé *lemme de paires critiques*. Si un système de réécriture est localement confluent et terminant, alors il est non-ambigu pour la réduction de termes, c'est-à-dire appliquer les règles du système autant que possible mais dans n'importe quel ordre mène à un résultat unique. Ceci est la propriété de confluence déjà mentionnée. L'exemple suivant sert à illustrer ces notions :

Exemple 1.2.4 *Rappelons les deux règles de réécriture de l'exemple 1.2.3. Une paire critique est l'équation de deux termes résultant de la superposition des deux membres gauches des règles et l'application respective des règles correspondantes. Pour les règles mentionnées, ceci nous donne* `b = c`. *Ni* `b`, *ni* `c` *ne peuvent plus être réduits. Donc, le système des deux règles de l'exemple 1.2.3 n'est pas confluent.*

Néanmoins, l'orientation de l'équation `b = c` *dans une direction précise et l'ajout de la règle de réécriture résultante aux deux règles initiales donne un système confluent, c'est-à-dire, le résultat devient non-ambigu pour n'importe quelle séquence de réductions.*

Ce que nous avons réalisé dans l'exemple précédent illustre la complétion de Knuth et Bendix : nous avons ajouté des paires critiques injoignables au système. Remarquons que ceci ne change pas la théorie équationnelle, car l'égalité ajoutée peut toujours être prouvée en utilisant les deux règles superposées. Donc, la complétion de Knuth-Bendix nous donne la possibilité de construire des programmes non-ambigus et terminants à partir de spécifications équationnelles, sans changer leur sémantique. La complétion devient plus compliquée en présence d'équations conditionnelles.

Pourtant, pendant les cinq dernières années, des techniques puissantes traitant le cas plus général de théories équationnelles utilisant des théories clausales du premier ordre ont été développées, surtout dans le contexte de la théorie des preuves, afin d'obtenir des prouveurs de théorèmes efficaces. L'idée principale est de construire un système de réécriture qui soit confluent sur les termes sans variables, appelés termes *clos*, par des techniques de superposition. Au lieu de calculer seulement des paires critiques, la superposition est aussi utilisée afin de résoudre partiellement les conditions dans les clauses. Quand toutes les instances des clauses diminuant la mesure de complexité (utilisée pour prouver la terminaison) sont calculées, alors un modèle peut être construit par induction sur l'ordre de terminaison et donc, l'ensemble de ces clauses est consistant, s'il ne contient pas la clause vide. Cette technique est appelée *saturation*.

Notre travail consiste en l'extension des résultats principaux de la théorie de la réécriture pour des cadres formels typés avec fonctions strictes et partielles, où les types sont interprétés comme ensembles. La relation d'appartenance $x \in A$ est donc considérée comme fonction caractéristique $A(x)$ pour laquelle nous devons établir la terminaison et la confluence comme pour toutes les autres fonctions. Ceci est le point de vue que nous adoptons dans la première partie de la thèse. Un autre raffinement est possible en utilisant des termes décorés où les décorations mémorisent des formules d'appartenance.

2.3 Les logiques R^n et G^n

Le développement des logiques R^n et G^n a pour but principal de donner un cadre formel simple, basé sur la théorie des ensembles, qui admet un typage expressif, des fonctions d'ordre

supérieur et des modèles initiaux. Notre volonté d'utiliser des interprétations venant de la théorie des ensembles s'explique par la simplicité de la théorie des ensembles et le fait qu'elles sont intuitivement attractives pour la spécification formelle de logiciels. Les fonctions et les types d'ordre supérieur sont des concepts souvent utilisés que nous voulons manier d'une façon uniforme. Nous voulons aussi donner un système de déduction concis et suffisamment simple, permettant l'utilisation de la réécriture.

Les techniques de spécification algébrique modélisent des types comme ensembles et les sous-types comme sous-ensembles, appelés respectivement sortes et sous-sortes. Dans les cadres formels algébriques habituels, les expressions de sortes sont généralement restreintes à des constantes, les fonctions sont du premier ordre, et des assertions de typage sont statiques et sans conditions. De cette approche, nous voulons garder la sémantique initiale, qui nous donne un modèle unique à un isomorphisme près, et des techniques de réécriture comme sémantique opérationnelle.

Définir des graphes de fonction comme ensembles de paires argument/valeur est une technique classique pour donner une sémantique aux fonctions, venant de la théorie des ensembles. Mais quand les fonctions doivent être applicables à elles-mêmes, comme dans le λ-calcul non-typé, les interprétations naïves, basées sur la théorie simple de types, ne sont pas assez puissantes pour permettre la définition de graphes de fonction. Notre théorie inclura des références aux ensembles (c'est-à-dire des noms), pour traiter ce problème.

La logique R^n est une logique de clauses de Horn avec une relation d'appartenance $\in$. Le paramètre n, qui est un nombre naturel ≥ 0, donne une borne sur la profondeur d'imbrication pour les ensembles utilisés dans les interprétations. Par analogie avec [Whitehead et Russell, 1925], nous affectons des ordres $i \in [0..n]$ aux variables et termes, de telle sorte qu'un terme d'ordre 0 soit interprété comme valeur individuelle, et un terme d'ordre 1 ou plus grand soit un ensemble. De plus, les formules sont restreintes à des formules stratifiées : $t \in t'$ est une formule valide seulement si t est d'un ordre plus petit que t', et $t = t'$ est une égalité admissible seulement si toutes ses instances préservent l'ordre (membre gauche et membre droit sont de même ordre). Ceci empêche le paradoxe de Russell. De plus, tous les ensembles, représentés par des termes, doivent être non vides, afin d'éviter la négation dans ce fragment de clauses de Horn.

La différence avec la théorie des types de Russell [Whitehead et Russell, 1925] est la prise en compte de modèles non-standards. Dans notre cas, ces modèles non-standards sont des modèles qui ne sont pas générés par des termes. Ceci s'accompagne de l'extension de la signature par des fonctions de choix, qui sont déterministes dans notre approche. Afin de distinguer des ensembles qui ne sont pas déclarés égaux, nous ajoutons l'axiome de choix pour des ensembles non vides et l'extensionalité non-standard. L'utilisation essentielle des choix dans notre cadre formel est la possibilité d'exprimer qu'il peut y avoir d'autres objets, que ceux représentés par des termes dans un modèle particulier. Donc, étant donné un ensemble d'individus, nous définissons les ensembles, en même temps que les choix, comme suit : un choix d'ordre 0 est un terme représentant un individu (peut-être non-standard). Un ensemble d'ordre 1 est un ensemble de choix d'ordre 0 et d'individus. Un choix d'ordre $k \in [1..n-1]$ est un terme représentant un ensemble d'ordre k. Un ensemble d'ordre $k \in [2..n]$ est un ensemble de choix et d'ensembles d'ordre $k-1$.

L'idée sous-jacente pour la structure de sortes associée à la logique R^n est donc de commencer avec la théorie de types de Russell jusqu'à l'ordre n, qui est multi-sortée. Supposons que $\{s_0, \ldots, s_n\}$ est l'ensemble de sortes. Le symbole de sorte s_0 est donc la sorte des individus qui sont représentés par des termes, et si $i \in [1..n]$, alors s_i est la sorte des termes représentant des ensembles d'ensembles d' ... (i fois) d'individus. Par conséquence, les ensembles dans s_i sont appelés les ensembles d'ordre i. Maintenant, nous pouvons ajouter des super-sortes $s'_0, \ldots, s'_n$ respectivement pour $s_0 \ldots, s_n$, tel que s'_0 est la sorte de tous les individus, pas nécessairement

représentés par un terme, et s'_i pour $i \in [1..n]$ est la sorte des ensembles d'ensembles d' ... (i fois) d'individus, pas forcément représentés par un terme. Ceci nous permet, par exemple, de raisonner sur des nombres réels comme individus, bien qu'il soit impossible de les représenter tous sous forme de termes générés à partir d'un ensemble fini d'opérateurs. Les fonctions de choix sont alors définies de s_i à s'_{i-1}, pour $i \in [1..n]$. Appelons cette théorie intermédiaire la logique R^n simple. Afin de comparer ce concept avec celui de l'algèbre G, nous avons illustré le système de types dans la figure 1.3.

Par analogie, nous pouvons définir la logique G^n simple, qui diffère simplement par le fait que les fonctions ne sont pas forcément totales : si f est définie comme fonction, par exemple, des individus dans les individus, ceci n'implique pas que f soit complètement définie sur tous les individus. Par contre, si $f(t)$ est défini pour un terme individuel t, alors $f(t)$ doit être un individu. La logique G^n semble plus proche de l'intuition humaine pour la spécification de logiciels que la logique R^n. Mais elle est aussi un peu plus complexe, car il faut savoir traiter l'existence d'un terme t en utilisant un prédicat supplémentaire $EX\ t$.

La différence entre les logiques R^n/G^n simples et complètes consiste en références aux ensembles, qui sont des individus. Comme dernière étape de notre construction, nous ajoutons des sortes pour des références à des objets, sous condition que l'objet référencé soit représenté par des termes dans une des sortes $s_0, \ldots, s_n$. Intuitivement les références sont des noms pour les objets qu'elles réfèrent. Il y a une correspondance bijective entre références et objets référés. Les références dans notre cadre sont surtout utiles pour la construction de graphes de fonction. Elles permettent de définir un graphe comme un ensemble de paires composées de références aux arguments et du résultat, même si l'on n'admet que des fonctions qui préservent l'ordre.

Les présentations en logique R^n ou G^n (simple ou complète) ont une signature et des axiomes sous forme de clauses de Horn, incluant des formules d'appartenance conditionnelles et des égalités conditionnelles, qui permettent (entre autres) de traiter des types paramétrés et dépendants. Les fonctions de choix nous donnent la possibilité de formuler l'extensionalité non-standard comme règle de déduction sur des clauses de Horn sans conditions inductives. Ceci est en plus la clé pour atteindre la complétude de la déduction pour la logique R^n utilisant un calcul de clauses de Horn simple, de premier ordre, ordo-sorté et équationnel. Une classe de modèles est définie pour les deux cas (logique R^n et G^n) et la correction et la complétude des calculs associés par rapport à la classe de modèles correspondante est prouvée dans le chapitre 2. Avant d'illustrer ces deux logiques par des exemples, résumons les résultats principaux, prouvés dans les sections mentionnées (attention : la première composante des nombres indiquant les théorèmes est le numéro du chapitre) :

Résultat 1 *Il existe des systèmes de déduction corrects et complets pour les logiques R^n et G^n. (voir théorèmes 2.3.6, 2.3.12, 2.4.5 et 2.4.14).*

Toute présentation R^n ou G^n possède un modèle minimal dans la catégorie des modèles associée (voir théorèmes 2.3.13 et 2.4.15).

Un exemple simple pour l'utilisation de la logique R^n et de la logique G^n est l'expression de la division par deux sans reste pour les nombres naturels. La version non-opérationnelle est donnée par $\{(a(x) = y \Leftarrow 2 * y = x)\}$; l'alternative opérationnelle par $\{(a(0) = 0), (a(1) = 0), (a(x) = a(x-2) + 1 \Leftarrow x \geq 2)\}$. Ceci illustre l'ambivalence des logiques R^n et G^n comme langage de spécification formelle et langage de programmation. Un exemple plus compliqué est celui des listes polymorphiques ordonnées, qui est donné ci-dessous afin d'illustrer les types dépendant des fonctions.

Exemple 1.2.5 *Soient* `elem`, `ens` *and* `ensens` *respectivement les types d'individus, d'ensembles*

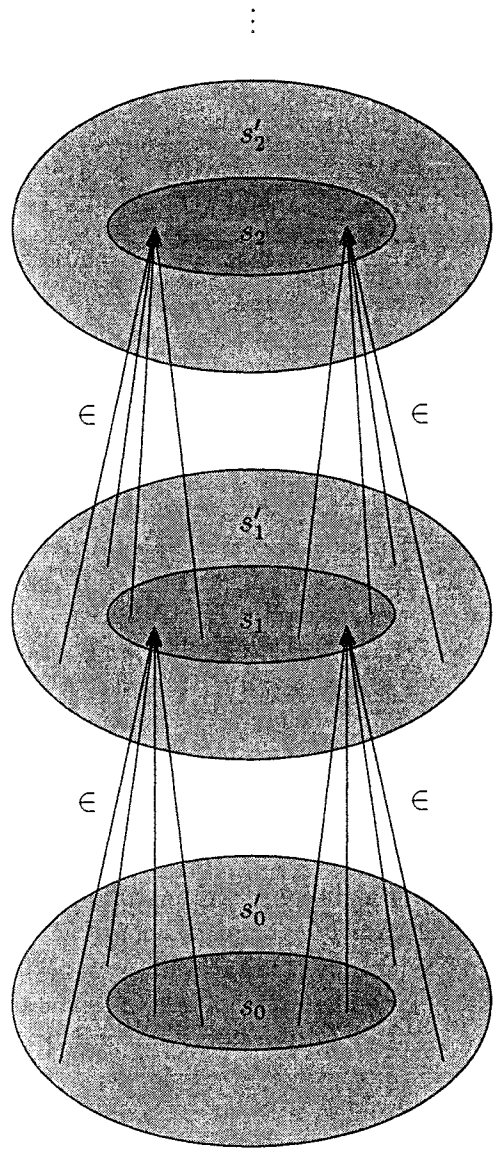


FIG. 1.3 – *La Structure de la Logique* R^n

d'individus et d'ensembles d'ensembles d'individus. Soit la signature Σ *telle que les opérateurs* `oliste` *(listes ordonnées),* `ocreer`, `ocons` *(constructeurs de listes),* `insert` *(insertion d'un élément) et* `tordres` *(ordres totaux d'éléments) y soient contenus et aient les profils suivants:*

```
oliste : ens ens -> ens,    ocreer : ens -> ens,
ocons : elem elem -> elem,  tordres : ens -> ensens,
insert : elem elem -> elem
```

De plus, soient `x,y,l` *des variables de type* `elem`, *et* `s,ord` *des variables de type* `ens`. *Soit* ϕ *la conjonction* (`ord` $\in$ `tordres(s)` $\wedge$ `x` $\in$ `s` $\wedge$ `y` $\in$ `s` $\wedge$ `l` $\in$ `oliste(s, ord)`) *et* ϕ' *soit* $\phi \wedge$ `ocons(y,l)` $\in$ `oliste(s,ord)`. *Alors les axiomes sont les suivants:*

```
          ocreer(ord) ∈ oliste(s,ord)   ⇐ ord ∈ tordres(s),
    ocons(x,ocreer(ord)) ∈ oliste(s,ord)  ⇐ ord ∈ tordres(s) ∧ x ∈ s,
     ocons(x,ocons(y,l)) ∈ oliste(s,ord)  ⇐ paire(x,y) ∈ ord ∧ φ',
            insert(x,l) ∈ oliste(s,ord)   ⇐ φ,
insert(x,ocons(y,l)) = ocons(x,ocons(y,l))  ⇐ paire(x,y) ∈ ord ∧ φ',
insert(x,ocons(y,l)) = ocons(y,insert(x,l))  ⇐ paire(y,x) ∈ ord ∧ φ
```

La logique G^n permet par exemple de coder l'intersection $\cap$ des ensembles. Evidemment, l'hypothèse que tous les ensembles dans les modèles considérés sont non vides, et la totalité des fonctions restreindrait considérablement nos modèles : chaque intersection $a \cap b$ devrait être non-vide, si $\cap$ n'est pas une opération partielle.

Exemple 1.2.6 *Dans la logique* G^n *nous pouvons définir l'intersection (partielle)* $\cap$ *d'une façon propre comme suit:*

```
(x ∈ s ⇐ x ∈ s ∩ s'), (x ∈ s' ⇐ x ∈ s ∩ s'),
        (x ∈ s ∩ s' ⇐ x ∈ s ∧ x ∈ s')
```

Maintenant, on peut donner des constantes `nat,ent,liste` *de type* `ens` *et* `0` *de type* `elem`, *ensemble avec les axiomes* `0` $\in$ `nat`, `0` $\in$ `ent`, *impliquant que* `ent` $\cap$ `nat` *soit défini, mais pas* `ent` $\cap$ `liste`.

Les logiques R^n et G^n offrent la possibilité d'exprimer et manipuler des fonctions en utilisant des graphes. Si `paire` et `args` sont respectivement des constructeurs de paires argument-valeur et de tuples d'arguments, alors le graphe `f :->set` d'une fonction `f` peut être défini comme `paire(args(`$\vec{x}$`), f(`$\vec{x}$`))` $\in$ `f`, tant que `paire(args(`$\vec{x}$`), f(`$\vec{x}$`))` est un individu. Ceci est même possible dans la logique R^n et G^n simple, parce que l'on peut définir `paire` et `args` comme fonctions qui retournent un individu pour toute combinaison de sortes d'arguments. Dans ce cas, les arguments peuvent être des ensembles, et donc `paire` et `args` jouent un rôle similaire aux fonctions de choix. Une alternative est d'utiliser des références : Si `#` est l'opérateur retournant des références à son argument, alors `paire(args(#`$\vec{x}$`), #f(`$\vec{x}$`))` $\in$ `f` définit des graphes de fonction sans utiliser des fonctions qui ressemblent aux fonctions de choix, parce que tous les arguments sont maintenant des individus.

Nos logiques permettent aussi de définir des fonctions auto-applicables avec une sémantique basée sur la théorie des ensembles. Supposons que l'on veuille définir des restrictions de domaines `restreint(f,s)` pour des fonctions, souvent écrit $f|_s$. Maintenant, notre système de types permet de dire `restreint(restreint, fs)`, où `fs` est un ensemble de graphes de fonction, défini dans la même logique. Ceci est possible, car nous pouvons définir le graphe de `restreint` comme simple ensemble d'individus, exactement comme ci-dessus.

2.4 Superposition Basique pour les Logiques R^n et G^n

Comme cela a déjà été mentionné dans la section concernant la réécriture, il existe une technique récemment développée pour les preuves par réfutation, donnant des systèmes de réécriture confluents sur des termes clos. Cette technique s'appelle *saturation* d'ensembles de clauses de Horn par superposition. Puisque les logiques R^n et G^n sont des théories basées sur des clauses de Horn, cette technique peut facilement être reconnue comme une bonne procédure pour construire des programmes à partir de présentations dans ces langages formels. Par conséquence, une première étape vers d'une méthode de construction est de considérer une extension de la saturation pour des théories de clauses de Horn, incluant un prédicat d'appartenance prédéfini. La saturation de clauses avec contraintes, utilisant la superposition basique, comme proposée par R. Nieuwenhuis et A. Rubio dans [Nieuwenhuis et Rubio, 1992a] et [Nieuwenhuis et Rubio, 1992b], est une technique de preuve très puissante. Néanmoins, ce calcul peut-être optimisé dans le contexte de preuves de typage, qui sont très uniformes et fréquemment répétées.

Afin de s'approcher de la technique de saturation classique, nous prouvons premièrement que les règles de déduction supplémentaires (l'extensionalité, l'axiome du choix, la bijectivité des opérateurs de référence et de déréférence et, pour la logique G^n, l'axiome d'existence), qui sont nécessaires pour les logiques R^n et G^n par rapport à la logique équationnelle ordo-sortée de clauses de Horn (abrégée $\mathcal{OSE}$-logique, voir section 2.2) peuvent être remplacées par des clauses de Horn sans changer le modèle initial (voir section 4.2.1, théorèmes 4.2.1 et 4.2.2): Soient **RNL** et **GNL** respectivement les systèmes de déduction pour les logiques R^n et G^n. Soit **P** un ensemble de clauses de Horn dans ces logiques. Les règles de déduction, provenant de la théorie des ensembles, peuvent être remplacées dans ces deux systèmes par des clauses de Horn, d'une telle façon que pour tout atome clos L, nous pouvons prouver L à partir de **P** en utilisant respectivement **RNL** ou **GNL**, si et seulement si nous pouvons prouver L à partir de $\mathbf{P}^+$ sans utiliser ces règles de déduction, où $\mathbf{P}^+$ est **P** avec les clauses de Horn représentant ces règles.

La prochaine étape est l'élimination de la relation d'existence supplémentaire, qui distingue la logique R^n de la logique G^n. Heureusement, ceci est possible par une transformation de clauses de Horn (voir section 4.2.3, théorème 4.2.4): Soient **P** un ensemble de clauses de Horn G^n, **GNL** comme ci-dessus, et **OSGL** notre système de déduction pour la $\mathcal{OSE}$-logique avec une règle de réflexivité changée (voir section 4.2.3). Il existe une transformation Ξ d'ensembles de clauses de Horn G^n-, qui élimine le prédicat EX, tel que pour tout atome clos L, nous pouvons prouver L à partir de **P** en utilisant **GNL** si et seulement si nous pouvons prouver $\Xi(L)$ à partir de $\Xi(\mathbf{P})$ avec **OSGL** et sans prédicat d'existence.

L'idée principale est de remplacer EX par une formule d'appartenance qui dit que l'argument appartient au domaine de tout modèle. Maintenant, la seule différence entre la déduction pour la logique R^n et pour la logique G^n est la règle de réflexivité, qui repose sur les deux concepts différents de fonction.

Exemple 1.2.7 *Soient* `0,1,div` *zéro, un et division dans le modèle standard des nombres naturels. Par conséquent,* `div(1,0) = div(1,0)` *est un théorème de la logique* R^n*, à cause de la restriction des interprétations aux fonctions totales, mais ceci n'est pas un théorème dans la logique* G^n*, car* `div(1,0)` *n'est pas défini.*

Alors que l'extension de la saturation par superposition basique de la logique de clauses de Horn classique à la logique R^n consiste simplement à remplacer le résolveur de contraintes par un résolveur ordo-sorté, les changements sont plus nombreux pour la logique G^n. Néanmoins, nous prouvons l'existence de procédures de saturation pour les deux, logique R^n et logique G^n,

dans les sections 4.3.1 et 4.3.2 (théorèmes 4.3.1 et 4.3.8):

Résultat 2 *Il existe une procédure de saturation par superposition basique, qui est réfutationnellement complète, respectivement pour la logique R^n et pour la logique G^n.*

Ce résultat nous sert comme une référence standard pour la comparaison avec un calcul décoré que nous présentons dans la suite. Afin de donner une base formelle pour la réduction d'un calcul de superposition décorée à un des calculs du dernier résultat, nous ajoutons encore une composante syntaxique aux clauses à contraintes. Pour ce but, nous étendrons les clauses contraintes (écrites $L \Leftarrow \Gamma \ [\![T]\!]$ dans ce qui suit) par des assertions $[\![S]\!]$ qui contiennent des théorèmes de typage dans le contexte de Γ et $[\![T]\!]$. Ces théorèmes sont ajoutés par des règles d'inférence supplémentaires, dont nous prouvons qu'elles représentent une extension conservatrice du calcul.

Les clauses à contraintes et assertions résultantes sont appelées *clauses casées*, car elles sont entourées d'une assertion $[\![S]\!]$ et d'une contrainte $[\![T]\!]$, ce qui donne la syntaxe suivante: $[\![S]\!]\ L \Leftarrow \Gamma\ [\![T]\!]$. Il faut remarquer, que nous avons inversé la direction de la flèche d'implication par rapport à [Nieuwenhuis et Rubio, 1992a], afin de rendre la lecture d'une clause casée plus intuitive: S et L sont une conséquence de Γ et T. Le résultat prouvé dans la section 4.3.3 (théorème 4.3.13) est le suivant:

Résultat 3 *Le calcul de saturation par superposition basique pour la logique G^n peut être étendu aux clauses casées.*

L'exemple suivant illustre ce résultat:

Exemple 1.2.8 *Soit* $\mathbf{F} = \{f, a\}$ *avec* $f \succ a$ *et* $\mathbf{P}$ *l'ensemble de clauses suivants:*

$$\left\{ \begin{array}{l} a \in A, \\ f(x) = x \Leftarrow x \in A, \\ \Leftarrow f(a) \in A \end{array} \right\}$$

Nous commençons la saturation avec une assertion pour la dernière clause, qui peut être déduite en utilisant la première clause. Le résultat est:

$$[\![a \in A]\!]\ f(a) \in A$$

Maintenant, nous pouvons superposer la deuxième clause de $\mathbf{P}$ *dans celle que nous venons d'obtenir, ce qui donne:*

$$[\![a \in A]\!] \Leftarrow x \in A\ [\![x =^? a]\!]$$

Maintenant on sait que le but $x \in A$ est satisfait pour toute instantiation de x satisfaisant la contrainte $[\![x =^? a]\!]$, car l'atome correspondant est déjà présent dans l'assertion. Donc, on peut immédiatement déduire la clause vide, ce qui prouve l'inconsistance de $\mathbf{P}$.

Le mécanisme d'assertions est en effet restreint pour éviter des circularités dans le raisonnement, qui feraient que le calcul devient incorrect. Par exemple, il est interdit de déduire $[\![p]\!] \Leftarrow p$ de $\Leftarrow p$, car sinon on aurait automatiquement la clause vide à partir de tout but. La règle exacte restreint les assertions pouvant être ajoutées à des conséquences (dérivables avec $\mathbf{P}$) de littéraux strictement plus petits (par rapport à $\succ$) apparaissant dans la condition de la clause.

2.5 Une Théorie Equationelle pour les Termes Décorés

Comme nous l'avons souligné dans le début de l'introduction, il nous semble nécessaire de traiter des fonctions partielles, des types et du polymorphisme, en particulier dans le contexte de spécifications algébriques (voir [Ehrig et Mahr, 1985; Bidoit *et al.*, 1990]), mais aussi en calcul formel (voir [Jenks et Sutor, 1992; Miola, 1993]). Les deux dernières décennies ont vu l'apparition de beaucoup de cadres formels intégrant la notion de sorte, sous-sorte, égalité et de polymorphisme dans le contexte du premier ordre et de l'ordre supérieur (voir [Curien et Ghelli, 1991]). Dans la partie de cette thèse qui traite des termes décorés dans les théories équationnelles (chapitres 5 à 9), nous considérons le problème du calcul efficace dans les algèbres à sortes ordonnées avec des conditions moins restrictives pour les présentations admises.

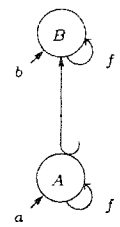
Dans la plupart des études de calculs à sortes ordonnées, une logique est définie, et la notion correspondante de modèle est introduite, ainsi que les résultats montrant les correspondances entre la théorie de modèles et la théorie de preuves. Mentionnons, sans que cette liste soit exhaustive, l'introduction de l'algèbre ordo-sortée dans [Goguen, 1978], entièrement développée dans [Goguen et Meseguer, 1992], la sémantique ordo-sortée introduite par [Smolka *et al.*, 1989], et les déclarations de termes, reprises dans [Schmidt-Schauß, 1987]. Tous ces travaux utilisent les expressions de la forme $A \leq B$ (déclaration d'inclusion de sortes) et $f : A\ B \to C$ (déclaration d'un opérateur) dans un sens purement syntaxique. Elles sont utilisées afin de définir les termes bien-formés, mais ne sont pas directement intégrées dans le processus de déduction. Ceci cause en un grand nombre de problèmes afin d'obtenir un théorème de complétude ou de décider de la confluence locale (voir [Jouannaud *et al.*, 1992; Waldmann, 1992]).

Les Problèmes

La situation suivante est souvent rencontrée: soient A une sous-sorte de la sorte B, f un opérateur défini uniquement sur A et retournant un résultat dans A, a une constante dans A et b une constante dans B. Si l'égalité $a = b$ est donnée, alors il est problématique de déduire $f(a) = f(b)$, car $f(b)$ n'est pas un terme bien formé (en utilisant les déclarations ci-dessus seulement comme restriction syntaxique). De ce point de vue, l'égalité n'est plus une congruence. Ce problème a été surmonté de plusieurs façons : par exemple dans [Goguen et Meseguer, 1992], où seulement des termes bien-formés sont admis dans les égalités à prouver, ou bien en considérant le concept de sortes dynamiques (voir [Watson et Dick, 1989]).

Un autre problème de l'approche considérant les inclusions de sortes et les déclarations d'opérateurs comme outils purement syntaxiques, est que les techniques standards de la réécriture se comportent d'une façon inattendue. La spécification écrite ci-dessous dans une syntaxe proche d'OBJ, et donnée par [Smolka *et al.*, 1989], représente une spécification avec un système de réécriture qui n'a pas de paire critique mais qui n'est pas localement confluent.

sorte	$A\ B$
sous-sorte	$A \leq B$
op	$f : A \to A$
op	$f : B \to B$
op	$a : \to A$
op	$b : \to B$
règle	$a \to b$
règle	$f(x : A) \to x$



Il n'y a pas de paire critique (standard) entre les deux règles de réécriture, mais le terme

$f(a)$ peut être réécrit en a en utilisant la deuxième règle, puis en b avec la première. Une alternative est de réécrire $f(a)$ en $f(b)$ en utilisant la première règle. Les deux termes b et $f(b)$ sont irréductibles. Dans ce cas, le processus de réécriture n'est pas assez intelligent pour découvrir le fait que b est de sorte A, car $a : A$ et $a \to b$. Ce fait permettrait de faire converger b et $f(b)$ par l'application de $f(x : A) \to x$. Le problème rencontré par cette première approche pour le calcul ordo-sorté, est que les termes peuvent être syntaxiquement mal formés, bien qu'ils soient sémantiquement corrects. Une façon simple de surmonter ce problème est de restreindre l'approche aux règles qui font décroître la sorte du terme réécrit, appelées règles à sortes décroissantes, comme dans [Kirchner *et al.*, 1988b]. Une solution opérationnelle décrite dans [Goguen *et al.*, 1985; Jouannaud *et al.*, 1992] est d'ajouter des *rétractions* (en anglais : *retracts*) au moment de l'exécution, mais cette proposition ne résout pas le problème en toute généralité.

Il existe déjà des propositions pour des procédures de complétion pour les spécifications algébriques ordo-sortées. Par exemple, dans [Gnaedig *et al.*, 1990], le cadre formel de l'algèbre ordo-sortée de [Goguen *et al.*, 1985; Kirchner *et al.*, 1988b] est considéré. Un autre cadre formel pour le calcul ordo-sorté est développé dans [Smolka *et al.*, 1989] (pour une comparaison des des deux approches, voir [Waldmann, 1992]). Toutes ces approches ont besoin de l'hypothèse supplémentaire de règles à sortes décroissantes, afin de prouver une version adéquate du lemme de paires critiques. Dans [Ganzinger, 1991b], une traduction de spécifications ordo-sortées dans des spécifications multi-sortées est proposée comme solution opérationnelle pour la complétion des théories ordo-sortées, qui permet d'éviter la restriction aux règles à sortes décroissantes, mais qui doit traiter des équations conditionnelles.

Des travaux plus récents en déduction avec contraintes (voir [Kirchner *et al.*, 1990; Comon, 1992]) nous mènent à considérer ce dernier problème comme une instance plus générale du phénomène de l'interaction entre contraintes et formules. Les règles de l'exemple ci-dessus peuvent par exemple être exprimées comme suit :

$$\begin{array}{ll} a & \to b \\ f(x) & \to x \quad \| \quad x \in A \end{array}$$

Dans la deuxième formule la contrainte $x \in A$ traduit l'information de typage $(x : A)$.

Des restrictions, telles que la régularité, la cohérence et la décroissance de sortes, peuvent être comprises comme une façon de résoudre l'interaction entre les contraintes de sortes et les formules. Mais ceci n'est pas tout à fait satisfaisant, et des solutions alternatives visant à contourner le problème esquissé dans l'exemple ci-dessus ont été recherchées. Une première direction est de modifier l'unification et les opérations de filtrage, afin qu'elles soient plus puissantes. Ceci est l'approche de H. Comon, utilisant en particulier l'unification monadique de deuxième ordre (voir [Comon, 1992]). Nous proposons dans cette thèse une autre approche, qui est basée sur l'algèbre G (section 1.2.1) et qui admet des sortes, des sous-sortes, des égalités et des fonctions partielles dans un cadre formel du premier ordre. En particulier, le problème de la signification de $f(a) = f(b)$ dans l'exemple ci-dessus est surmonté, car il est possible d'inférer de la présentation que b, qui est égal à a, est aussi de la sorte A. Dans un tel contexte, on peut aussi écrire, par exemple, que $i * i : \mathbf{R}$, où i est le nombre complexe bien connu, et des termes comme $pop(pop(push(a, push(a, P))))$ s'évaluent d'une façon naturelle en P, sans utiliser de rétractions. En effet, pour l'algèbre G, le typage devient une preuve et le problème est d'automatiser ces preuves et d'atteindre un calcul efficace dans ce cadre formel.

Notre Approche

Partant des motivations et des idées exprimées ci-dessus, notre approche dans ce travail est, premièrement, d'introduire aussi peu de restrictions sur les présentations ordo-sortées que possible. Deuxièmement, nous essayons de comprendre pourquoi quelques-unes de ces restrictions sont nécessaires, afin de réaliser un processus de déduction complet, et troisièmement, nous proposons des restrictions supplémentaires, qui pourraient être introduites afin d'obtenir une meilleure efficacité du calcul ordo-sorté.

En algèbre G, les formules peuvent être des égalités $(t = t')$ ou des déclarations de termes (aussi appelées formules d'appartenance), et le processus de déduction incorpore l'interaction entre ces deux sortes de formules. Mais cette généralité mène à l'indécidabilité du typage, précisément à cause de l'interaction entre le calcul, concernant respectivement sortes et égalités, et parce que l'on admet des déclarations non-triviales de termes. Ceci a deux conséquences immédiates : filtrage et unification sont indécidables en algèbre G. Donc, il est impossible d'appliquer les techniques usuelles pour simuler la logique équationnelle par la logique de réécriture (voir [Meseguer, 1992]) après la complétion, car la réécriture (utilisant le filtrage), ainsi que que la superposition de règles (utilisant l'unification), sont indécidables.

Afin de surmonter le problème d'indécidabilité, nous proposons une sémantique opérationnelle pour les règles de déduction de algèbre G (données dans la figure 2.1) basée sur une représentation adéquate de termes, appelée *termes décorés*, et sur une traduction des formules de la présentation par des règles de réécriture.

L'aspect dynamique de l'information de sortes et son interaction avec l'égalité est pris en compte par la notion de termes décorés. Les décorations sont des ensembles de sortes, qui enregistrent les sortes actuellement prouvées pour un terme. Elles sont distribuées dans le terme et agissent localement, comme contraintes de sortes, pendant la déduction. Manier l'information de sortes localement dans chaque nœud d'un terme impose une définition adéquate du filtrage et de l'unification. Le filtrage décidable sur les termes décorés permet de définir la réécriture avec deux notions appropriées de règles de réécriture. Le premier type de règles correspond aux axiomes égalitaires de la présentation, transformés en *règles décorées*, qui réécrivent des termes et enrichissent éventuellement leurs décorations. Le second type de règles correspond aux déclarations des termes, qui sont transformées en *règles de décoration* conditionnelles, qui enrichissent les décorations sans modifier la structure des termes.

En utilisant ces outils, nous revenons au problème de semi-décision de formules égalitaires $(t = t')$ et de formules d'appartenance $(t : A)$ en algèbre G. Ceci est résolu par une généralisation de la complétion bien connue de [Knuth et Bendix, 1970], qui s'applique aux termes décorés, en utilisant des versions décidables du filtrage et de l'unification mentionnées ci-dessus. Ainsi, bien sûr, nous transférons tous les problèmes au niveau de la complétion.

Le processus de complétion que nous proposons, est basé sur l'hypothèse que l'information dans la présentation est modularisée en trois parties. La première (i) contient toutes les égalités $(t = t')$, la deuxième (ii) contient les déclarations de termes $(t : A)$, et la dernière (iii) définit les inclusions de sortes $(A \leq B)$. Les deux premières sont traitées par des règles de réécriture et sont alors modifiées et enrichies pendant la complétion. Par contre, la dernière est considérée comme information stable pendant *toute* la complétion. En particulier, les filtres et unificateurs sont calculés en utilisant, comme d'habitude, la structure de termes mais aussi l'information de sortes, donnée dans les décorations et la structure de sortes. Comme le filtrage et l'unification utilisent seulement l'information de sortes présente dans le terme décoré au moment du filtrage ou de l'unification, ils sont en général corrects mais pas complets. Afin d'obtenir la complétude pour le calcul de paires critiques, impliqué dans la complétion, il est nécessaire que l'information

de sortes dans (iii) satisfasse la propriété suivante : si un terme t dans la présentation peut être prouvé appartenir aux sortes $A_1, \ldots, A_n$, alors ces sortes ont une sous-sorte commune. Cette propriété de complétude de la partie (iii) de la présentation est appelée *héritage de sortes*, et nous supposons qu'elle est satisfaite pendant toute la complétion.

La procédure de complétion présuppose la propriété d'héritage de sortes de la présentation, nécessaire pour la complétude du calcul de paires critiques. Si la complétion termine, alors l'ensemble de règles de réécriture résultant offre une possibilité de prouver non seulement des théorèmes équationnels de la forme $(t = t')$, mais aussi des théorèmes de typage de la forme $(t : A)$, et des théorèmes existentiels $(EX\ t)$ (voir théorème 7.4.3):

Résultat 4 *Soit $\mathcal{P}$ une présentation en algèbre G, qui satisfait la propriété d'héritage de sortes. Soit la dérivation de complétion sans échec et équitable (toute paire critique de la présentation limite $\mathcal{P}_\infty$ obtenue par complétion est confluente).*

Alors $\mathcal{P}_\infty$ est terminant et possède la propriété de Church-Rosser. Donc, tout théorème équationnel, d'appartenance ou existentiel peut être décidé par la réécriture de termes décorés.

Ensuite, nous décrivons une procédure, qui permet de tester l'héritage de sortes. Quand la propriété d'héritage de sortes n'est pas satisfaite, ceci est détecté par un échec de la procédure. Un contre-exemple qui peut être exploité afin d'enrichir la structure de sortes est alors synthétisé.

Résultat 5 *Il existe un test correct pour l'héritage de sortes basé sur la superposition de règles de décoration. Pour nos résultats de complétude, nous distinguons trois cas, en fonction de la forme des règles de décoration : plates et linéaires, plates ou sans restrictions.*

Dans le premier cas, le test peut être ajourné jusqu'à la fin, si nous restreignons les règles décorées utilisées pour la simplification de règles de décoration, aux règles qui préservent les décorations. Dans le deuxième cas, une précomplétion avec une relation de réécriture différente (et donc des paires critiques différentes) doit être appliquée en premier. Le test peut-être ajourné de nouveau, si la simplification par des règles décorées est évitée. Dans le troisième cas, la précomplétion devient plus complexe, car les paires critiques sont plus générales, et le test doit être appliqué pendant la précomplétion avec une version restreinte de la simplification par des règles de décoration, sans simplification par des règles décorées, et avec une stratégie particulière.

Il faut souligner, que dans cette approche, l'information de typage a deux parties : la partie statique ne contient que la relation de sous-sortes. La partie dynamique couvre les déclarations de termes sous forme de règles de réécriture qui sont traitées en-dehors de l'unification, et qui donnent automatiquement un algorithme de typage dans la présentation complétée.

Comme une conséquence de cette approche, nous obtenons un calcul ordo-sorté assez général, qui peut être implanté d'une manière efficace, à cause de la mémorisation de l'information de typage dans les termes et le comportement statique du filtrage et de l'unification. En sus, la notion de rétraction n'est plus nécessaire pour traiter des termes syntaxiquement malformés.

Un Exemple Simple

Appliquer cette approche à l'exemple précédent nous donne les étapes de calcul suivantes. La présentation est premièrement traduite dans des formules de l'algèbre G (voir deuxième colonne ci-dessous) :

sort	$A\ B$	$x :: A,\ y :: B$
subsort	$A \leq B$	$x : B$
op	$f : A \to A$	$f(x) : A$
op	$f : B \to B$	$f(y) : B$
op	$a : \to A$	$a : A$
op	$b : \to B$	$b : B$
rule	$a \to b$	$a = b$
rule	$\forall x : A,\ f(x) \to x$	$f(x) = x.$

Puis les formules de la première colonne sont traduites en formules à termes décorés dans la deuxième colonne :

$f(x) : A$	$f(x^{:\{A\}})^{:s}$	$\to$	$f(x^{:\{A\}})^{:s \cup \{A\}}$ if $\{A\} \not\subseteq s$
$f(y) : B$	$f(y^{:\{B\}})^{:s}$	$\to$	$f(y^{:\{B\}})^{:s \cup \{B\}}$ if $\{B\} \not\subseteq s$
$a : A$	$a^{:s}$	$\to$	$a^{:s \cup \{A\}}$ if $\{A\} \not\subseteq s$
$b : B$	$b^{:s}$	$\to$	$b^{:s \cup \{B\}}$ if $\{B\} \not\subseteq s.$
$a = b$	$a^{:\emptyset}$	$=$	$b^{:\emptyset}$
$f(x) = x$	$f(x^{:\{A\}})^{:\emptyset}$	$=$	$x^{:\{A\}}$

Cette présentation est saturée, en utilisant le processus de complétion, en la présentation suivante :

$$
\begin{aligned}
f(x^{:\{A\}})^{:s} &\to f(x^{:\{A\}})^{:s \cup \{A\}} \text{ if } \{A\} \not\subseteq s \\
f(y^{:\{B\}})^{:s} &\to f(y^{:\{B\}})^{:s \cup \{B\}} \text{ if } \{B\} \not\subseteq s \\
a^{:s} &\to a^{:s \cup \{A\}} \text{ if } \{A\} \not\subseteq s \\
b^{:s} &\to b^{:s \cup \{A,B\}} \text{ if } \{A, B\} \not\subseteq s \\
a^{:\{A\}} &\to b^{:\{A,B\}} \\
f(x^{:\{A\}})^{:\{A\}} &\to x^{:\{A\}}
\end{aligned}
$$

Dans cette présentation saturée, le terme $f(a^{:\emptyset})^{:\emptyset}$ est d'abord décoré (en utilisant des règles de décoration) en $f(a^{:\{A\}})^{:\{A\}}$. Puis, il est réécrit en utilisant les deux dernières règles de décoration en $b^{:\{A,B\}}$.

Il faut remarquer que dans ce nouveau cadre formel, les restrictions de régularité, de cohérence et de décroissance de sortes ne sont plus nécessaires pour obtenir les résultats mentionnés.

2.6 Une Logique Equationnelle à Clauses de Horn avec Termes Décorés

Des observations plus profondes, concernant l'algèbre G, révèlent que l'on aimerait bien parfois surmonter la restriction aux fonctions partielles strictes, en utilisant des égalités conditionnelles, bien que l'algèbre G soit déjà assez puissante par rapport aux spécifications équationnelles ordo-sortées classiques. En plus, nous voudrions bien ajouter au concept de calcul décoré le sous-typage paramétré. Cet objectif peut être atteint en passant de l'algèbre G à un fragment de la logique G^1. On convient que la sorte s_0 contient les termes d'individus et s_1 les termes d'ensembles. On restreint les fonctions à valeur dans s_1 à être totales et à ne pas prendre d'argument dans s_0. On n'autorise pas d'égalité sur s_1, mais on peut spécifier une relation d'inclusion sur les termes de s_1 à condition qu'elle n'interfère pas avec le reste de la présentation.

Le lecteur pourrait s'interroger à ce stade sur la restriction de n à 1. La réponse est pragmatique : cette restriction nous permet de combiner un calcul décoré avec des résolveurs de contraintes existants pour les inclusions d'ensembles (voir [Smolka, 1989; Hill et Topor, 1990;

Fuh et Mishra, 1988]). Le fait de se restreindre à un fragment de G^1 ne veut pas dire qu'il n'y ait pas de solution au problème de déduction automatique, ou de la réécriture, pour le cas général de la logique G^n : la solution dans ce cas consiste à utiliser l'approche *conditionnelle* de la section 4.3.2. Par contre, si on sort de ce fragment, le concept de termes décorés rend le calcul de superpositions trop complexe. Ceci est dû au problème d'unification de variables qui nécessite la résolution de contraintes d'inclusion sur les ensembles. Des recherches pour réduire les hypothèses restrictives semblent être intéressantes, mais demandent encore beaucoup d'effort. Néanmoins, s'il est possible de résoudre les contraintes d'inclusion sur les ensembles, le calcul proposé dans le chapitre 10 pourra être facilement étendu, car il est paramétré par le résolveur de contraintes.

Pour le fragment de la logique G^1 défini plus haut, l'étape suivante est de donner une sémantique aux clauses contraintes, basées sur des termes décorés, en les transformant en clauses casées, de sorte que les décorations correspondent aux assertions. Il faut remarquer, qu'il y a une grande différence entre cette sémantique pour les décorations et celle utilisée pour la complétion de présentations en algèbre G. Cette dernière est complètement syntaxique, et les décorations sont donc une partie du terme, au même titre que les symboles de fonction. Ceci mène à un grand nombre de paires critiques avec des règles de décoration. Donc, donner une sémantique aux décorations par l'intermédiaire d'assertions, semble avantageux et, en effet, nous avons trouvé des exemples, où, grâce à cette sémantique, le calcul se comporte mieux.

On définit une transformation Ψ de clauses décorées en clauses casées, illustrée dans l'exemple suivant :

Exemple 1.2.9 *Soit* $(\Leftarrow f(x^{:\{A\}})^{:\{B\}} = x^{:\{A\}} \; [\![x^{:\{A\}} =^? a^{:\{A\}}]\!])$ *une clause décorée. La clause casée correspondante est la suivante :*

$$[\![f(x) \in B, a \in A]\!] \Leftarrow f(x) = x, x \in A \; [\![x =^? a]\!]$$

Donc, chaque décoration d'un terme non-variable dans la clause décorée (contrainte inclue) mène à une assertion et chaque variable décorée à une formule d'appartenance dans la condition.

En s'inspirant de la transformation des clauses casées en clauses décorées définie par l'inverse de Ψ, nous arrivons à un nouveau calcul de saturation décoré. Par analogie avec le cas algébrique, nous utilisons la propriété d'héritage de sortes pour restreindre les superpositions calculées dans le calcul décoré. Néanmoins, le test de l'héritage de sortes est plus complexe dans ce cas à cause de la présence d'un ensemble de sortes potentiellement infini. Dans le calcul de clauses casées cette propriété d'héritage de sortes est axiomatisée par un ensemble infini potentiellement (dénombrable) de *buts* (i.e. clauses de Horn sans tête), de telle sorte que la présentation devient consistante, si et seulement si elle satisfait la propriété d'héritage de sortes.

Soient $\mathbf{P}$ la présentation G^1 appartenant au fragment considéré, et $\mathbf{P}'$ la même présentation, augmentée par un ensemble de clauses de Horn axiomatisant l'héritage de sortes. Afin de prouver la complétude réfutationnelle du calcul de saturation décoré, par rapport au calcul à clauses casées pour les logiques G^n appliqué à $\mathbf{P}'$, nous procédons en trois étapes :

1. Nous prouvons la correction des règles de déduction décorées par rapport à $\mathbf{P}'$, en traduisant toute clause décorée avec Ψ (voir lemme 10.4.10).

2. Puis, nous prouvons que la saturation dans le calcul décoré implique la saturation dans le calcul à clauses casées après la transformation avec Ψ (voir lemme 10.4.11).

3. Finalement, nous établissons une correspondance entre un test sur la présentation décorée saturée et la consistance de $\mathbf{P}'$ dans le calcul casé qui nous garantit la propriété d'héritage des sortes. Ce test nécessite que la présentation décorée saturée soit finie.

Il faut remarquer que cette formulation des théorèmes est simplifiée, car elle ne décrit pas comment se fait le transfert dans le résolveur de contraintes d'une partie du calcul sur les inclusions des ensembles réalisé au niveau des clauses casées. Les détails, concernant ce sujet peuvent être trouvés dans la formulation des lemmes et théorèmes du chapitre 10. Nous pouvons déduire en suivant le schéma de preuve donné ci-dessus :

Résultat 6 *Le calcul de saturation décoré incluant le test d'héritage de sortes, est réfutationnellement complet sous les restrictions mentionnées en logique G^1. De plus, si l'ensemble de clauses saturé satisfait une restriction supplémentaire syntaxique concernant les occurrences de variables, alors nous pouvons construire un système de réécriture terminant et confluent sur les termes d'individus clos, complet pour la déduction de formules équationnelles, existentielles et d'appartenance, dans lesquelles tout terme d'individus est clos.*

Ceci nous donne une technique de construction de programmes, incluant le sous-typage paramétrique. Il faut remarquer, que le résultat admet des termes d'ensemble non-clos, tel que $liste(x)$. Ceci correspond aux types polymorphiques dans des langages de programmation fonctionnelle. Nous terminons cette section avec deux exemples. Le premier illustre l'utilisation de cette procédure avec des règles de typage conditionnelles, comme dans le cas des contraintes de sortes.

Exemple 1.2.10 *Soit* $\mathbf{P}$ *la présentation suivante, décrivant des arbres binaires AVL (Adelson-Velskii, Landis) [Knuth, 1981], dont la propriété principale est que la différence de profondeur entre le sous-arbre gauche et celle du sous-arbre droit est au maximum* 1. *Le trait sépare les fonctions auxiliaires sur les nombres naturels des fonctions sur les arbres.*

$$
\left\{\begin{array}{l}
Nat, \\
0 \in Nat, \\
s(x) \in Nat \Leftarrow x \in Nat, \\
diff(x,0) = x \Leftarrow x \in Nat, \\
diff(0,x) = x \Leftarrow x \in Nat, \\
diff(s(x),s(y)) = diff(x,y) \Leftarrow x \in Nat, y \in Nat, \\
0 \leq x \Leftarrow x \in Nat, \\
s(x) \leq s(y) \Leftarrow x \leq y, x \in Nat, y \in Nat, \\
0 + x = x \Leftarrow x \in Nat, \\
s(x) + y = s(x + y) \Leftarrow x \in Nat, y \in Nat, \\
max(0,x) = x \Leftarrow x \in Nat, \\
max(x,0) = x \Leftarrow x \in Nat, \\
max(s(x),s(y)) = s(max(x,y)) \Leftarrow x \in Nat, y \in Nat, \\
\hline
AVL \subseteq Arbre, \\
arbre\_vide \in AVL, \\
arbre(x,y) \in Arbre \Leftarrow x \in Arbre, y \in Arbre, \\
arbre(x,y) \in AVL \Leftarrow x \in AVL, y \in AVL, diff(prof(x),prof(y)) \leq s(0), \\
prof(arbre\_vide) = 0, \\
prof(arbre(x,y)) = max(prof(x),prof(y)) + s(0) \Leftarrow x \in Arbre, y \in Arbre
\end{array}\right.
$$

Toute variable dans **P** *est d'ordre* 0. *Cette présentation est en effet saturée, comme on peut facilement le vérifier avec la procédure de saturation non-décorée. Par contre, on peut observer que typer un arbre AVL non-vide entièrement d'une manière naïve nous fait prouver plusieurs fois la même chose pour les sous-termes (une fois pour les typer eux-mêmes, une fois pour chaque nœud au-dessus). La version décorée évite de refaire ce travail en réutilisant les preuves de typage :*

$$
\left\{
\begin{array}{l}
0^{:s} \rightarrow 0^{:s\cup\{Nat\}}, \\
s(x^{:\{Nat\}})^{:s} \rightarrow s(x^{:\{Nat\}})^{:s\cup\{Nat\}}, \\
diff(x^{:\{Nat\}}, 0) = x^{:\{Nat\}}, \\
diff(0, x^{:\{Nat\}}) = x^{:\{Nat\}}, \\
diff(s(x^{:\{Nat\}}), s(y^{:\{Nat\}})) = diff(x^{:\{Nat\}}, y^{:\{Nat\}}), \\
0 \leq x^{:\{Nat\}}, \\
s(x^{:\{Nat\}}) \leq s(y^{:\{Nat\}}) \Leftarrow x^{:\{Nat\}} \leq y^{:\{Nat\}}, \\
0 + x^{:\{Nat\}} = x^{:\{Nat\}}, \\
s(x^{:\{Nat\}}) + y^{:\{Nat\}} = s(x^{:\{Nat\}} + y^{:\{Nat\}}), \\
max(0, x^{:\{Nat\}}) = x^{:\{Nat\}}, \\
max(x^{:\{Nat\}}, 0) = x^{:\{Nat\}}, \\
max(s(x^{:\{Nat\}}), s(y^{:\{Nat\}})) = s(max(x^{:\{Nat\}}, y^{:\{Nat\}})), \\
\hline
arbre\_vide^{:s} \rightarrow arbre\_vide^{:s\cup\{AVL\}}, \\
arbre(x^{:\{Arbre\}}, y^{:\{Arbre\}})^{:s} \rightarrow arbre(x^{:\{Arbre\}}, y^{:\{Arbre\}})^{:s\cup\{Arbre\}}, \\
arbre(x^{:\{AVL\}}, y^{:\{AVL\}})^{:s} \rightarrow arbre(x^{:\{AVL\}}, y^{:\{AVL\}})^{:s\cup\{AVL\}} \\
\qquad\qquad \Leftarrow diff(prof(x^{:\{AVL\}}), prof(y^{:\{AVL\}})) \leq s(0), \\
prof(arbre\_vide) = 0, \\
prof(arbre(x^{:\{Arbre\}}, y^{:\{Arbre\}})) = max(prof(x^{:\{Arbre\}}), prof(y^{:\{Arbre\}})) + s(0)
\end{array}
\right\}
$$

Pour typer entièrement le terme $arbre(arbre(arbre\_vide, arbre\_vide), arbre\_vide)$, *il faut typer le sous-terme* $arbre(arbre\_vide, arbre\_vide)$ *deux fois dans la première version et une seule fois dans la version décorée. Le filtrage réutilise les renseignements dans les décorations des sous-termes, si on type avec une stratégie ascendante, car au moment d'arriver au nœud de tête, le terme a la forme suivante :*

$$arbre(arbre(arbre\_vide^{:\{AVL\}}, arbre\_vide^{:\{AVL\}})^{:\{AVL\}}, arbre\_vide^{:\{AVL\}})$$

Donc, on peut appliquer la règle de décoration pour les arbres AVL non vides. En effet, on n'a que la profondeur à recalculer, car x et y sont instanciés par des termes avec le terme d'ensembles AVL dans leur décoration de tête. On obtient alors :

$$arbre(arbre(arbre\_vide^{:\{AVL\}}, arbre\_vide^{:\{AVL\}})^{:\{AVL\}}, arbre\_vide^{:\{AVL\}})^{:\{AVL\}}$$

Une analyse statique des types pourrait donner des résultats similaires sur cet exemple précis. Mais elle est insuffisante en général quand on combine surcharge des opérateurs, sous-typage et égalités.

De nouveau, nous avons simplifié l'exemple pour améliorer la lisibilité : pour obtenir le résultat il faut en fait prouver que les ensembles Nat et AVL sont non-vides. Ceci nous permet d'éliminer les conditions d'appartenance des variables dans les conditions des clauses. Le deuxième

exemple montre l'application de la procédure avec des types polymorphiques. On reprend le cas des arbres AVL avec données dans les nœuds, mais pas dans les feuilles :

Exemple 1.2.11 *Soit* **P** *la présentation suivante (où la partie des fonctions auxiliaires est identique avec celle de l'exemple précédent, les variables X, Y en majuscules sont des variables d'ensembles) :*

$$\left\{\begin{array}{l} \vdots \\ \hline AVL(X) \subseteq Arbre(X), \\ AVL(X) \subseteq AVL(Y) \Leftarrow X \subseteq Y, \\ Arbre(X) \subseteq Arbre(Y) \Leftarrow X \subseteq Y, \\ arbre\_vide \in AVL(X), \\ arbre(x,y,z) \in Arbre(X) \Leftarrow x \in Arbre(X), y \in Arbre(X), z \in X, \\ arbre(x,y,z) \in AVL(X) \Leftarrow x \in AVL(X), y \in AVL(X), \\ \qquad\qquad z \in X, diff(prof(x), prof(y)) \leq s(0), \\ prof(arbre\_vide) = 0, \\ prof(arbre(x,y,z)) = max(prof(x), prof(y)) + s(0) \\ \qquad\qquad \Leftarrow x \in Arbre(X), y \in Arbre(X), z \in X \end{array}\right\}$$

La deuxième et la troisième clause expriment la monotonie des opérateurs AVL et Arbre. **P** *est transformé dans l'ensemble de clauses décorées suivant :*

$$\left\{\begin{array}{l} \vdots \\ \hline arbre\_vide^{:s} \rightarrow arbre\_vide^{:s \cup \{AVL(X)\}}, \\ arbre(x^{:\{Arbre(X)\}}, y^{:\{Arbre(X)\}}, z^{:\{X\}})^{:s} \rightarrow \\ \qquad arbre(x^{:\{Arbre(X)\}}, y^{:\{Arbre(X)\}}, z^{:\{X\}})^{:s \cup \{Arbre(X)\}}, \\ arbre(x^{:\{AVL(X)\}}, y^{:\{AVL(X)\}}, z^{:\{X\}})^{:s} \rightarrow arbre(x^{:\{AVL(X)\}}, y^{:\{AVL(X)\}}, z^{:\{X\}})^{:s \cup \{AVL(X)\}} \\ \qquad \Leftarrow diff(prof(x^{:\{AVL(X)\}}), prof(y^{:\{AVL(X)\}})) \leq s(0), \\ prof(arbre\_vide(x^{:\{X\}})) = 0, \\ prof(arbre(x^{:\{Arbre(X)\}}, y^{:\{Arbre(X)\}}, z^{:\{X\}})) = \\ \qquad max(prof(x^{:\{Arbre(X)\}}), prof(y^{:\{Arbre(X)\}})) + s(0) \end{array}\right\}$$

Donc, nous pouvons décorer le terme $arbre(arbre\_vide, arbre\_vide, 0)$ de la manière suivante en utilisant des contraintes de sous-ensembles $X \subseteq^? Y$ introduites par le filtrage :

$$\begin{array}{l} arbre(arbre\_vide, arbre\_vide, 0) \\ \rightarrow arbre(arbre\_vide, arbre\_vide, 0^{:\{Nat\}}) \\ \rightarrow arbre(arbre\_vide^{:\{AVL(X')\}}, arbre\_vide, 0^{:\{Nat\}}) \\ \rightarrow arbre(arbre\_vide^{:\{AVL(X')\}}, arbre\_vide^{:\{AVL(X'')\}}, 0^{:\{Nat\}}) \\ \rightarrow arbre(arbre\_vide^{:\{AVL(X')\}}, arbre\_vide^{:\{AVL(X'')\}}, 0^{:\{Nat\}})^{:\{AVL(X''')\}} \\ \qquad\qquad [\![X' \subseteq^? X''', X'' \subseteq^? X''', Nat \subseteq X''']\!] \end{array}$$

Les variables X' et X'' sont des variables universellement quantifiées au niveau des termes décorés, c'est-à-dire l'ensemble $AVL(X')$ correspond à un type fonctionnel polymorphique de la forme $\forall X'.AVL(X')$. La variable X''' par contre est existentiellement quantifiée. La contrainte $[\![X' \subseteq^? X''', X'' \subseteq^? X''', Nat \subseteq X''']\!]$ résulte de la décomposition de la contrainte initiale :

$$[\![AVL(X') \subseteq^? AVL(X'''), AVL(X'') \subseteq^? AVL(X'''), Nat \subseteq X''']\!]$$

Elle est satisfiable pour $X''' = \Omega_0$, l'ensemble universel des individus, ou pour $X''' = Nat$.

Le passage aux clauses décorées dans le dernier exemple n'arrive pas à optimiser le calcul de profondeur au moment de typage. Pour cela il faudrait pouvoir utiliser des termes d'ensembles du style $AVL(max(x, y) + s(0))$ dans des formules d'appartenance.

Ou bien on considère que le terme $max(x, y) + s(0)$ est un terme d'ensembles, mais on ne peut pas donner des axiomes pour max et $+$ dans notre fragment de G^1. Ou bien on considère ce terme comme un terme d'individus, mais on sort encore du fragment, car des termes d'individus ne sont pas admis comme sous-terme d'un terme d'ensembles.

Le problème sous-jacent est l'interaction des axiomes sur les termes d'individus avec les contraintes d'ensemble. Une solution possible semble être l'utilisation de calculs de superposition pour des relations transitives.

3 Domaines et Travaux Reliés

Dans cette section nous abordons quelques domaines voisins et travaux reliés. Nous commençons avec les domaines proches de nos bases théoriques, notamment la théorie des ensembles et la théorie des types, afin de situer les logiques R^n et G^n, qui sont non-standards, par rapport aux approches «classiques». Nous nous sommes décidés à omettre l'abstraction dans notre approche.

Ensuite, nous passons aux travaux reliés. Comme les approches utilisant la réécriture ont déjà été mentionnées (et sont discutées abondamment dans le chapitre 9), nous nous restreignons ici à la comparaison avec le typage dynamique dans la programmation fonctionnelle/logique et l'héritage multiple des langages de programmation impérative.

3.1 La Théorie des Ensembles

La théorie des ensembles a été fondée par G. Cantor, qui à été le premier à prouver l'existence du continuum à la fin du siècle dernier et qui a donc préparé les fondements de la théorie des ordinaux et des cardinaux. Cependant, peu de temps après, les premiers paradoxes ont été découverts pour sa théorie. Notamment l'existence d'un ordinal correspondant à l'ensemble (bien-fondé) de tous les ordinaux a été prouvé paradoxal par Burtali-Forti.

Afin d'éviter ces paradoxes, des propositions différentes ont été faites au début du 20ième siècle, concernant l'exclusion de paradoxes, mais aucune des théories résultantes n'a été prouvée consistante pour l'instant. Les mieux connues sont celles de Zermelo et Fraenkel, Russell et von Neumann, Bernays et Gödel. Le développement chronologique est illustré dans la figure 1.4. Pour un survol sur ces théories on peut consulter l'annexe historique de [Quine, 1969].

La différence principale est la forme des prédicats p admis dans le schéma de compréhension $\{x : p(x)\}$ utilisés pour la construction des ensembles. En les résumant en quelques mots, Zermelo et Fraenkel admettaient de nouveaux ensembles seulement comme sous-ensembles d'ensembles déjà construits, c'est-à-dire $p(x)$ est de la forme «$x \in A$ et $p'(x)$», tel que A est un ensemble. Russell, et plus tard Quine, donnaient un système de types évitant la construction de «sophismes de cercle vicieux», tel que $\{x : x \notin x\}$, aussi connu sous le nom de *paradoxe de Russell*. Von Neumann, Bernays et Gödel proposaient un nombre fini d'axiomes distinguant entre grands et petits ensembles, et des restrictions correspondantes sur les axiomes admettant la construction de nouveaux grands/petits ensembles.

Notre travail concernant la théorie des modèles est basé sur la théorie des types de Russell, qui est étendue en fonction des modèles considérés. Il faut mentionner que les approches basées sur la théorie des ensembles étaient profondément critiquées par l'école intuitionniste, fondée par Brouwer, qui prônait le constructivisme des résultats prouvés plutôt que l'absence de paradoxes comme critère d'acceptation pour l'existence d'objets mathématiques et la validité de preuves.

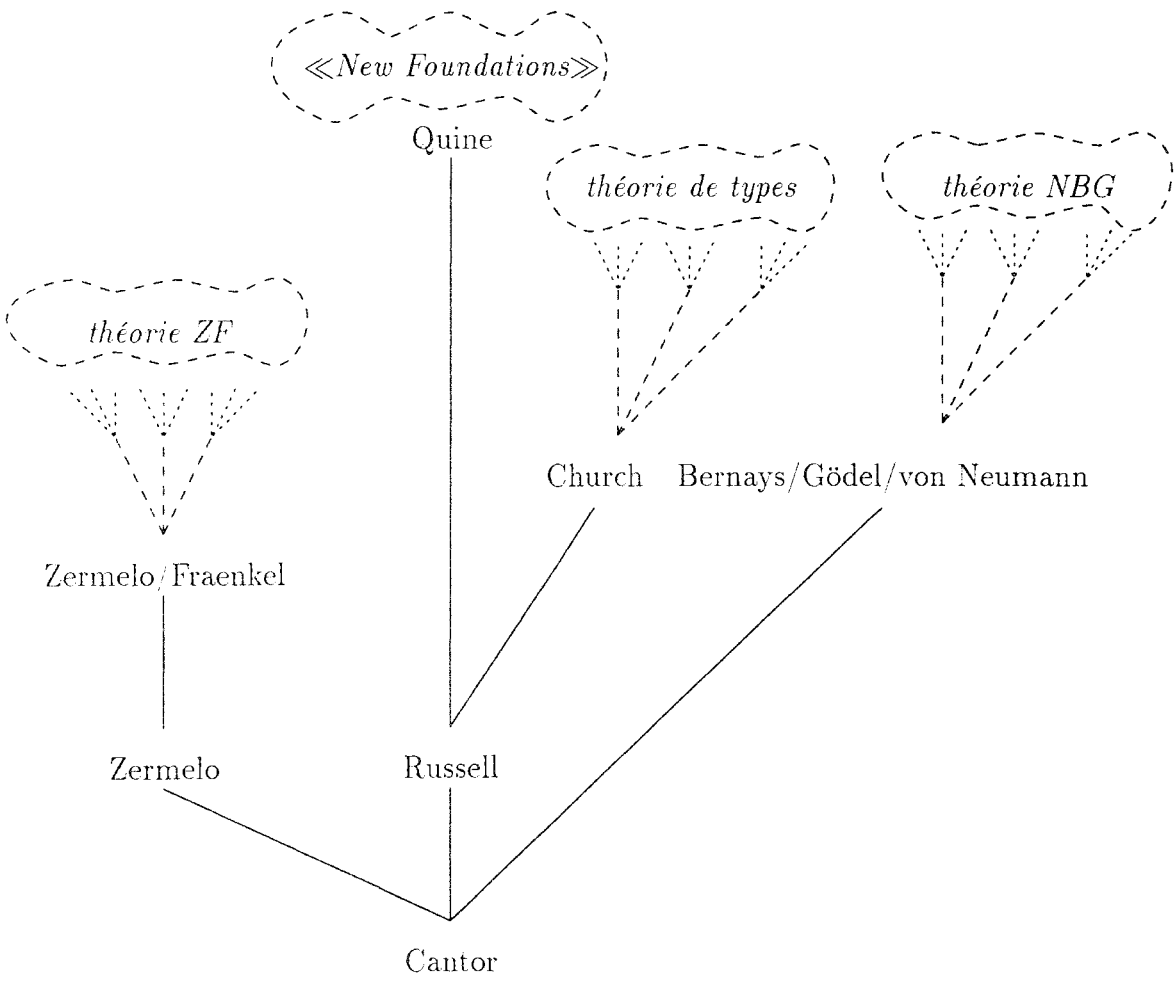


FIG. 1.4 – *Le Développement de la Théorie des Ensembles au Début du Siècle*

L'école des mathématiciens intuitionnistes n'est pas mentionnée dans le schéma de la figure 1.4, car les exigences supplémentaires sont relativement orthogonales aux différentes approches de la théorie des ensembles. Néanmoins, il y a aussi la notion de théorie des ensembles constructive, qui demande pour le schéma de construction $\{x : p(x)\}$ que p soit décidable. En outre, l'égalité des objets satisfaisant p doit être décidable et il faut savoir construire un élément pour tout ensemble nouveau. Notre travail sur les prouveurs de théorèmes automatiques dans cette thèse donne la possibilité de trouver des algorithmes pour les fonctions caractéristiques d'ensembles et de décider l'égalité pour les éléments de ces ensembles. Ceci à la condition que nos techniques de complétion rendent un programme exécutable sous forme d'un système de réécriture décoré terminant et non-ambigu, car nous savons dans ce cas, comment décider si un terme appartient ou non à un ensemble, en utilisant des techniques de réécriture.

3.2 La Théorie des Types

Dans la théorie des types, les deux dernières décennies ont vu beaucoup de résultats intéressants. Par exemple, les résultats concernant l'interprétation des types comme propositions par l'intermédiaire des isomorphismes de Curry-Howard, dans le cas du λ-calcul simplement typé où le système de type est analogue à la logique propositionnelle minimale; ou du système F, lorsque les types correspondent à des formules du calcul propositionnel de deuxième ordre. Dans les deux théories, le typage correspond à une preuve de la formule correspondante dans un système de déduction naturelle et le terme typé code la preuve. Un aspect de l'analogie des types et des formules est que la terminaison du λ-calcul peut être prouvée par un théorème d'élimination de coupure, inspiré par le travail de Gentzen concernant la logique du premier ordre.

La théorie des types joue un rôle important dans les fondements des mathématiques. Dans ses applications à la programmation surgissent un certain nombre de questions spécifiques :

La Théorie des Modèles. La théorie des modèles sous-jacente de ces calculs est souvent très sophistiquée. Ceci est un inconvénient, car les spécifications, aussi bien que les programmes, devraient être intuitivement clairs et compréhensibles.

Typage Statique et Dynamique. Le typage est en général statique, c'est-à-dire l'évaluation d'un terme ne change pas l'appartenance aux types. Il y a des exemples naturels de l'arithmétique où ceci n'est pas le cas (cf. Exemple 1.1.4). Les extensions de λ-calculs typés avec des types dynamiques seront discutées dans une prochaine sous-section.

Représentations Différentes. La vérification de programmes vis-à-vis de spécifications indépendantes pose souvent le problème de comparer deux types de données intuitivement équivalents, mais structurellement différents. De façon analogue, des fonctions écrites par deux personnes différentes, faisant intuitivement le même travail, pourraient ne pas être représentées par le même λ-terme, suite à un codage différent des données. Par exemple, le codage de l'addition de nombres entiers est différente en fonction du choix entre entiers de Curry et entiers de Church.

Terminaison Universelle. Un système de typage restreignant les fonctions acceptables à des fonctions terminantes est trop strict en général, car il doit toujours y avoir des fonctions calculables qui ne sont pas typables dans ces systèmes. (Sinon le système de types déciderait le problème d'arrêt.)

Des Espaces de Fonctions Complets. La quantification sur des espaces de fonctions entiers (comme celui des fonctions continues dans des topologies de Scott [Scott, 1976] dans le cas du λ-calcul non-typé) peut-être un obstacle pour la preuve de terminaison. Des travaux récents (par exemple [Lysne et Piris, 1995]) ont montré les difficultés à surmonter, si on veut étendre des techniques de preuve de terminaison aux calculs avec λ-abstraction.

Récursion. Quelques théories des types n'admettent pas les définitions récursives de fonctions pour des raisons de prédicativité, ce qui est un très lourd handicap pour la programmation.

Cette thèse essaie de poursuivre une autre approche, où les problèmes mentionnés ci-dessus sont contournés. Nous utilisons des *modèles* facilement compréhensibles de la *théorie des ensembles*. Le *typage dynamique* est adopté pour la spécification et la programmation. Toutes les formules ont à satisfaire des conditions de typage statique simple. Notre travail prouve l'existence d'un modèle initial. Des *représentations différentes* pour les données et la distinction entre spécifications et implantations sont bien développées pour ce genre de modèles. Ceci est très bien adapté à l'utilisation de prouveurs de théorèmes par récurrence automatisés, qui aident à prouver des théorèmes de correspondance. La *terminaison* ne restreint a priori pas la classe des spécifications, mais la construction de programmes inclut la preuve de terminaison, où l'ordre bien-fondé utilisé n'est pas fixé une fois pour toute. Nous avons évité la λ-abstraction afin de permettre des preuves de terminaison plus simples que dans le cas d'un espace de fonctions complet, même si on utilise des fonctions d'ordre supérieur. Les modèles initiaux nous donne également une sémantique naturelle pour la restriction des *espaces de fonction* au fragment actuellement défini dans une présentation. Finalement, les définitions *récursives* sont admises et la prédicativité est assurée, car nous prouvons la terminaison des fonctions construites.

3.3 Sous-Typage et Types Dynamiques dans la Programmation Fonctionnelle

Dans cette section, nous discutons le système de types $F_{\leq}$ de Curien et Ghelli [Curien et Ghelli, 1990][1] qui est une extension du système F de Girard par le sous-typage, et les types dynamiques proposés par Abadi, Cardelli, Curien, Plotkin et Rémy. Nous terminons en revenant sur les logiques R^n et G^n afin d'exhiber des types qu'on ne trouve pas dans la programmation fonctionnelle.

Le système $F_{\leq}$ de Curien et Ghelli est un λ-calcul avec les types et expressions suivantes :

$$A ::= t \mid Top \mid A \to A \mid \forall t \leq A.A$$
$$a ::= x \mid top \mid \lambda x : A.a \mid a(a) \mid \Lambda t \leq A.a \mid a(A)$$

Le langage contient donc des types d'ordre supérieur (ou de fonction) $A \to B$ et des types polymorphiques $\forall t \leq A.B$. De plus, il contient l'abstraction ($\Lambda t \leq A.a$) et l'application ($a(A)$) de types, aussi bien que abstraction et application de termes ($\lambda x : A.a$ et $a(a)$). Comme l'abstraction de types est bornée ($t \leq A$), il faut en plus du système F des jugements de sous-typage $A \leq A$, qui ont été prouvé indécidable par Pierce[2]. Le type Top inclut tous les types et top est une constante représentant l'élément unique de Top.

Système $F_{\leq}$ inclut trois types de théorèmes (ou jugements) : typage $a : A$, sous-typage $A \leq A$ et égalité $(a = a) : A$ (dans le type A). La sémantique est proche de la sémantique surchargée algébrique, comme celle utilisée pour l'algèbre ordo-sortée par Goguen et Meseguer [Goguen et Meseguer, 1992], c'est-à-dire chaque terme peut avoir des dénotations différentes dans des types différents. Dans le système $F_{\leq}$, les égalités sont héritées des sous-types, par exemple $(a = b) : A$ et $A \leq B$ impliquent $(a = b) : B$. Par contre, $(a = b) : B$ et $A \leq B$ n'impliquent pas $(a = b) : A$. Ceci permet de définir $(x = top) : Top$ pour tout x.

Dans ce qui suit, nous discutons les possibilités de spécifier les types et expressions de $F_{\leq}$ dans les logiques R^n et G^n. Puis, nous prenons les résultats de la discussion comme base pour montrer

1. Nous aurions pu choisir aussi bien le système $F_{<:}$ de Cardelli et al. [Cardelli *et al.*, 1991], qui est très similaire à système $F_{\leq}$. Ce choix est arbitraire et ne change pas les arguments utilisés.

2. [Vorobyov, 1994] contient une extension conservatrice décidable du système de sous-typage de $F_{\leq}$.

les différences principales qui se trouvent dans l'abstraction, le typage en général, l'héritage en particulier et l'opérationalisation.

Nous pouvons reconnaître beaucoup de similitudes avec les logiques R^n et G^n : Top correspond avec l'ensemble universel Ω, qu'on peut définir par $x'^0 \in \Omega$ ou $x'^0 \in \Omega \Leftarrow EX\ x'^0$ dans le cas de la logique G^n, où x'^0 est de la sorte s'_0 de tous les individus, non-standards inclus. Les types d'ordre supérieur $A \rightarrow A$ peuvent être spécifiés par l'introduction de l'opérateur $\rightarrow$ avec le profil $s_1\ s_1 \rightarrow s_1$ et les clauses suivantes, où x^0, y^0 sont de sorte s_0, X^1, Y^1, Z^1, V^1 sont des variables d'ensemble (de sorte s_1) représentant des types et U^1 est une variable d'ensemble (de sorte s_1 aussi) représentant un graphe de fonction :

$$y^0 \in Y^1 \quad \Leftarrow < x^0, y^0 > \in U^1, ref(U^1) \in (X^1 \rightarrow Y^1), x^0 \in X^1$$
$$(X^1 \rightarrow Y^1) \subseteq (Z^1 \rightarrow V^1) \quad \Leftarrow Z^1 \subseteq X^1, Y^1 \subseteq V^1$$

La première clause décrit les conséquences de l'application d'une fonction et la deuxième dit que les types de fonction sont antimonotones dans le premier argument et monotones dans le deuxième. Une fonction f (avec le graphe F^1) peut alors être définie de type $A^1 \rightarrow B^1$ en disant $ref(F^1) \in (A^1 \rightarrow B^1)$. Le type $\forall t \leq A.A$ correspond à une condition supplémentaire $t \subseteq A$ (l'inclusion d'ensembles $\subseteq$ peut être définie dans les logiques R^n et G^n). L'application $a(a)$ ou $a(A)$ ne change pas, sauf si on veut utiliser les graphes. A ce moment, il faut traduire $F^1(x^0)$ en y^0 avec la condition correspondante $< x^0, y^0 > \in F^1$. Il n'y a pas de possibilité de spécifier la constante *top* et les abstractions de variables de type et de termes.

Nous pouvons voir dans la définition des types d'ordre supérieur $X^1 \rightarrow Y^1$ que la nature du typage dynamique dans notre approche diffère du typage contreignant des λ-calculs typés, comme $F_{\leq}$: dans ces derniers, un λ-terme n'est considéré que s'il est syntaxiquement typable, ce qui reste invariant par application. Le typage syntaxique permet (au moins dans le cas sans sous-typage) d'éliminer des erreurs syntaxiques. Par contre, avec le typage dynamique utilisé dans les logiques R^n et G^n, nous acceptons tous les termes comme admissible et essayons de faire des preuves de typage par des preuves égalitaires. Le typage dynamique correspond au paradigme des langages de programmation orientés objets où il influence le flux de contrôle d'une manière qui ne peut pas être détectée au moment de la compilation. Le typage optimal devrait réunir les deux paradigmes afin de combiner leurs avantages.

Nous pouvons aussi localiser une différence principale dans le traitement de l'héritage. Rappelons qu'il n'existe qu'une seule classe d'équivalence dans Top et tout terme est égal à *top* dans Top. Par contre, il est possible que a ne soit pas égal à b dans un sous-type de Top. Ceci n'est pas possible dans nos logiques, car les termes ont une équivalence structurelle, indépendamment de leur type. Donc, déclarer $a = b$ dans Ω n'est pas restreint à Ω, mais globalement respecté. Par conséquence, les classes d'équivalence diminuent en montant dans la hiérarchie des ensembles (par inclusion) dans les logiques R^n et G^n. Le système $F_{\leq}$ offre donc en plus une façon de redéfinir des opérations d'une manière plus forte dans des super-types. Nous pensons que le traitement différent du même objet ou un comportement différent d'une fonction dans des types différents augmente la confusion. Si une opération d'un super-type n'est pas la même que sur un sous-type, alors il nous semble plus sage de la nommer différemment. Si deux données représentent deux choses différentes, alors il faut marquer ceci dans la structure de l'objet afin d'éviter des confusions. Ce sont les raisons principales motivant notre choix d'une sémantique non-surchargée à la Smolka et al. [Smolka *et al.*, 1989] et Mégrelis [Mégrelis, 1990], bien que ce comportement diffère de la plupart des langages orientés-objets.

Les travaux concernant la sémantique opérationnelle dans le système $F_{\leq}$ montrent certaines difficultés, notamment pour la stabilité par contexte. Ceci est dû à l'antimonotonie des constructeurs de types $\forall t \leq A.A$ and $A \rightarrow A$ dans la première expression A. Bien que nous admettions

l'antimonotonie dans les logiques R^n et G^n, nous avons remarqué qu'une restriction à la monotonie simplifie beaucoup les travaux techniques au niveau des preuves. Nous proposons donc d'utiliser à la place de l'antimonotonie des restrictions de fonction explicites comme proposé dans la thèse de Z. Qian [Qian, 1991], afin d'arriver au même effet, comme illustré dans l'exemple suivant :

Exemple 1.3.1 *Soit $f : A \to B$ une fonction prenant des arguments de type A et retournant un résultat dans B, c une constante de type C, qui est un sous-type de A.*

L'idée de l'antimonotonie de $\to$ dans le premier argument vient de l'utilisation de variables d'ordre supérieur. Si x est une variable de type $C \to B$, alors $x(c)$ est bien-typé (de type B). De plus, nous pouvons utiliser f comme instance de x sans perdre ce résultat (appelé «type safety» en anglais). Donc, il est raisonnable d'admettre le remplacement de x par f.

Néanmoins, le calcul de sous-typage peut devenir très complexe si nous admettons l'antimonotonie dans le premier argument. Par exemple, nous ne pouvons plus utiliser des ordres de terminaison monotones comme les ordres récursifs sur les chemins afin de prouver la terminaison d'une procédure utilisée pour tester le sous-typage.

Par contre, il est possible de définir des restrictions de domaine explicites en donnant les axiomes suivants dans les logiques R^n et G^n, où x^0, y^0 sont des variables de sorte s_0, F^1 est une variable d'ensemble représentant un graphe de fonction et X^1, Y^1 et Z^1 sont des variables d'ensemble représentant des types :

$$\begin{aligned}
y^0 \in Z^1 &\Leftarrow < x^0, y^0 > \in F^1, ref(F^1) \in (X^1 \to Y^1), \\
&\quad x^0 \in X^1 \\
(X^1 \to Y^1) \subseteq (X^1 \to Z^1) &\Leftarrow Y^1 \subseteq Z^1 \\
< x^0, y^0 > \in restreint(F^1, X^1) &\Leftarrow < x^0, y^0 > \in F^1, x^0 \in X^1 \\
ref(restreint(F^1, Y^1)) \in ((X^1 \cap Y^1) \to Z^1) &\Leftarrow F^1 \in (X^1 \to Z^1)
\end{aligned}$$

Le premier axiome décrit de nouveau l'appartenance du résultat à l'ensemble indiqué dans le type, le deuxième est la monotonie de $\to$ dans le deuxième argument, le troisième définit le graphe restreint d'une fonction et le dernier le nouveau type d'un graphe de fonction restreint. Remarquons que $(X^1 \cap Y^1) = X^1$ si $X^1 \subseteq Y^1$. Donc, il est possible d'utiliser $restreint(f, C)$ comme instantiation de x dans l'exemple ci-dessus ($x(c)$ serait remplacé par y avec $< c, y > \in x$ comme condition).

Comme conclusion de cette brève discussion, nous pouvons dire que $F_{\leq}$ a été conçu comme langage de programmation fonctionnelle (si on utilise une version décidable du sous-typage) et non comme langage de spécification avec possibilité d'opérationalisation comme ceci est le cas pour les logiques R^n et G^n. Ceci explique les nombreuses différences dans les détails, venant d'une sémantique adaptée respectivement à la programmation et à la spécification.

Nous arrivons maintenant aux types dynamiques proposés pour les langages de programmation fonctionnelle. Dans le domaine de la programmation fonctionnelle le typage est en général statique. Néanmoins, il est reconnu que cette technique n'est pas suffisante pour la programmation orientée objet avec sous-typage, qui nécessite la résolution de liens à l'exécution (*run-time linking*). Les dernières années ont montré l'apparition de plusieurs approches visant à combiner le typage statique et dynamique, afin de surmonter ce problème. Nous présentons dans la suite comme exemple les travaux d'Abadi, Cardelli, Pierce, Plotkin et Rémy [Abadi *et al.*, 1991; Abadi *et al.*, 1994], car ils montrent très bien les différences avec notre approche.

Le langage de programmation fonctionnelle proposé dans [Abadi *et al.*, 1991] est un langage à typage syntaxique avec un type distingué appelé `Dynamic`. Les valeurs de `Dynamic` sont des paires de valeur et type, appelées `dynamic`. L'opération principale qui vient avec les `dynamics` est le `typecase`, qui permet de distinguer les différents cas possibles pour le type contenu dans une paire `dynamic`. L'exemple suivant repris de [Abadi *et al.*, 1991] illustre ce principe.

Exemple 1.3.2 *La fonction suivante prend un* `dynamic x` *et retourne* `x+1` *si* `x` *contient une valeur* `i` *de type* `Nat` *et* `0` *sinon:*

```
λx:Dynamic.
     typecase x of
          (i:Nat)    i+1
          else       0
     end
```

Le type (statique) de la fonction est donc `Dynamic -> Nat`.

Il est aussi possible de définir des fonctions qui retournent des objets de type `Dynamic`. Pour cela, il suffit de retourner le terme `dynamic v:T`, où `v` est une valeur de type `T`. Le symbole de fonction `dynamic` est en effet le constructeur pour les paires de valeur et de type. Le terme `dynamic v:T` est admissible si `v` appartient au type `T`, ce qui peut être vérifié syntaxiquement, c'est-à-dire au moment de la compilation. Le dynamisme du typage vient du fait qu'on peut utiliser le type dans un `dynamic` pour choisir le bon flux de contrôle avec la structure de contrôle `typecase`.

Dans [Abadi *et al.*, 1994], les auteurs présentent une extension de cette technique aux langages polymorphiques, notamment un cadre avec polymorphisme explicite comme le système F [Girard, 1972] et un cadre avec polymorphisme implicite comme utilisé dans les langages de la famille ML [Milner et Tofte, 1991; Weis *et al.*, 1989] avec des restrictions au niveau des variables de type utilisées dans les cas du `typecase`, c'est-à-dire `(i:Nat)` dans l'exemple ci-dessus.

L'élégance de cette approche est donc dans la combinaison des deux paradigmes de typage. Les auteurs voulaient que le typage statique, qui permet l'omission d'information de typage et augmente l'efficacité, prenne le rôle principal et que le typage dynamique, qui est nettement plus coûteux dans l'implantation, ne soit utilisé qu'aux endroits qui la nécessitent.

En comparant avec notre approche, nous pouvons constater que Abadi et al. insistent beaucoup sur le maintien du typage statique, tandis que nous regardons cet aspect comme optimisation à posteriori pour notre travail. En effet, un terme décoré ressemble à un `dynamic`, car il combine valeur et types. La différence est que notre cadre formel inclut le sous-typage. Donc, le type associé à un terme n'est pas unique. Dans le langage AMBER [Cardelli, 1986], un prédécesseur du langage décrit dans [Abadi *et al.*, 1991], sous-typage et types dynamiques sont inclus, mais avec des équivalences structurelles entre types : une structure est un sous-type d'une autre structure, si les sélecteurs de champs de la première contiennent ceux de la deuxième et les types des champs correspondants respectent le sous-typage. Ce principe rappelle aussi les logiques de traits («feature logics»), qui sont discutées dans la section 1.3.4.

Il existe encore une autre possiblité d'interpréter les `dynamics`. Dans notre travail basé sur l'algèbre G, nous évitons de quantifier sur des types, afin de rester dans une logique de premier ordre. Dans les logiques R^n et G^n, nous admettons des variables d'ensembles (et donc de types). Il est donc possible de spécifier des `dynamics` comme termes dans ces logiques.

Pour terminer cette comparaison avec les langages de programmation fonctionnelle, nous revenons aux logiques R^n et G^n, afin de montrer une dernière grande différence avec notre travail.

Dans notre approche, les ensembles peuvent être utilisés comme tout autre objet. Comme nous pouvons définir des graphes de fonction comme ensemble d'individus dans les logiques R^n et G^n, il est possible de spécifier des ensembles d'individus dépendant d'une fonction (de son graphe) et d'un individu. Un exemple pratique pour un tel ensemble est celui des listes ordonnées bornées, où l'ensemble dépend de l'ordre utilisé et de la longueur maximale d'une liste. L'exploitation de ces possibilités au niveau de la réécriture a malheureusement dépassé le cadre de cette thèse. Il constitue néanmoins une direction prometteuse.

3.4 Sous-Typage et Structures/Classes dans la Programmation Logique

Dans cette section, nous discutons les relations de notre travail avec les langages de programmation logique avec sous-typage et avec structures de données («records») employées dans les langages orientés-objet avec héritage multiple. Nous esquissons le codage de telles structures dans les logiques R^n et G^n. Afin de limiter cette discussion délibérément, nous nous concentrons sur l'approche de Hanus pour le typage polymorphique, les travaux de Weidenbach pour le typage dynamique défini plus haut, et sur les logiques de traits («feature logics») de Smolka et al. pour les structures des langages orientés-objets.

Dans le domaine des langages de programmation logiques avec sous-typage, on peut citer outre les travaux historiques de la section 1.2.1.0, des travaux plus récents visant à intégrer les types polymorphiques et le sous-typage dans les systèmes PROLOG, comme par exemple dans [Hanus, 1989]. La différence principale avec nos travaux est la restriction au typage syntaxique et l'absence d'égalités dans les clauses. Comme Abadi et al., Hanus garde un typage syntaxique et propose un calcul de sous-typage algébrique, où le fait que A soit un sous-type de B est codé par $B(A) = A$. La théorie des types est donc une théorie algébrique séparée de la présentation logique. Mis à part cette différence, nous avons adopté la même approche concernant la séparation du calcul d'inclusion de types.

Une deuxième approche de la programmation logique avec sous-typage est celle de Weidenbach [Weidenbach, 1991], qui est principalement un calcul de résolution avec prédicat d'appartenance. La différence avec nos logiques est l'absence d'égalités et la considération de clauses générales (pas restreintes aux clauses de Horn positives), mais le typage est dynamique. Les clauses avec une conclusion d'appartenance sont pris en compte au niveau de l'unification, qui travaille à la fois sur des termes et des conditions exprimant que l'intersection des types des variables unifiées est non-vide. Les deux approches mentionnées ciblent clairement la résolution et non la déduction équationnelle comme la réécriture. L'application des procédures de saturation au fragment de clauses de Horn de ces calculs devrait donner des systèmes de premier ordre formant un fragment de notre calcul.

Une autre approche est représentée par les logiques de traits («feature logics»), qui sont des logiques ordo-sortées où les termes sont principalement des structures, comme les objets dans les langages de programmation orientés-objet. Chaque champ dans une structure a un sélecteur, qui permet d'accéder à son contenu. Le type d'un objet est, informellement, la structure du terme avec les sélecteurs mais sans le contenu des champs. Le sous-typage provient de la subsomption des champs aux positions correspondantes : si l'ensemble des sélecteurs d'un type est contenu dans l'ensemble des sélecteurs à la même position dans un autre type, alors le deuxième type est un sous-type du premier. Par exemple, le type d'un point à deux dimensions pourrait être $[x, y]$, où x, y sont des sélecteurs. C'est un sous-type de $[x]$ et $[couleur, [x, y]]$ est un sous-type de $[x, y]$. La logique que nous venons de décrire est due à Aït-Kaci et al. [Ait-Kaci *et al.*, 1994]. Elle a aussi été étendue aux arbres rationnels, qui sont des graphes récurrents, mais cette extension nous semble orthogonale.

Le sous-typage est donc souvent accompagné des structures de la programmation orientée-objet. Une façon naïve d'intégrer des structures dans notre travail est d'utiliser des tuples comme dans [Cardelli, 1992]. Donc, nous pouvons spécifier en logique R^n/G^n que le terme $objet(membres)$ est une structure, où $membres$ est de la forme suivante :

$$< champs(sel_1, valeur_1), < champs(sel_2, valeur_2), < \ldots, champs(sel_n, valeur_n) \ldots >>>,$$

avec les sélecteurs sel_1 jusqu'à sel_n. Ceci peut être spécifié par les clauses suivantes, où sel, x, y, m, xt, yt et mt sont des variables (nous avons omis les types afin de simplifier l'énoncé) :

$$\begin{aligned} champs(sel, x) &\in sel, \\ < x, y > &\in xt \quad \Leftarrow x \in xt \\ < champs(sel, x), y > &\in yt \quad \Leftarrow y \in yt, \\ objet(m) &\in struct(mt) \quad \Leftarrow m \in mt \end{aligned}$$

Le typage spécifié par les ensembles $struct(mt)$ correspond à celui de la logique des traits et peut être réalisé en utilisant la réécriture décorée dans la logique G^1. Afin de déclarer quelles structures existent, il suffit soit de donner un individu dans l'intersection de tous les sélecteurs, soit d'inclure une formule existentielle dans la présentation.

Avant de continuer, il faut remarquer que l'intersection $\cap$ et l'inclusion $\subseteq$ d'ensembles peuvent être spécifiés dans les logiques R^n et G^n. Donc, l'objet suivant :

$$\begin{aligned} objet(\quad & < champs(sel_1, valeur_1), \\ & < champs(sel_2, valeur_2), \\ & < \ldots, champs(sel_n, valeur_n) \ldots >>>) \end{aligned}$$

appartient à $struct(\cap_{i\in[1..m]} sel_i)$. L'accès aux composantes peut-être réalisé avec l'opération '.' en notation infixée. Donc, nous pouvons donner les équations suivantes :

$$\begin{aligned} & objet(membres).sel = membres.sel \quad \Leftarrow membres \in sel, \\ & < champs(sel, val), y > .sel = val, \\ & < x, y > .sel = y.sel \qquad\qquad \Leftarrow y \in sel \end{aligned}$$

Il faut remarquer que les formules dans les présentations doivent être syntaxiquement restreintes afin qu'un sélecteur dans un objet n'apparaisse qu'une seule fois. Sinon, les fonctions d'accès sont ambiguës. Il y a des possibilités d'étendre notre spécification avec des noms pour des structures (classes) et les sélecteurs correspondants. Le type d'une structure prend la forme $struct(nom, \cap_{i\in[1..m]} sel_i)$:

$$objet(membres) \in struct(nom, sels) \Leftarrow membres \in sels$$

Par exemple, $struct(chaussure, couleur \cap pointure)$ correspond avec la déclaration d'une structure «chaussure» avec les champs «couleur» et «pointure». En utilisant la spécifiction ci-dessus, nous pouvons prouver le théorème suivant :

$$< champs(couleur, noir), champs(pointure, 43) > \in struct(chaussure, couleur \cap pointure)$$

Utiliser des noms permet de distinguer deux structures avec les mêmes sélecteurs, comme $struct(point, x \cap y)$ et $struct(paire, x \cap y)$, qu'on ne veut pas confondre car ils se comportent différemment.[3]

3. Ceci est par exemple la sémantique définie par Stroustrup pour les classes en C++.

Notons que la possibilité de manier des graphes de fonctions permet de spécifier aussi facilement des *fonctions membres*, c'est-à-dire des graphes de fonction comme valeur d'un champ. L'héritage multiple de plusieurs classes de base $struct(nom\_de\_base_i, sels\_de\_base_i)$, $i \in [1..m]$, au profit d'une structure $struct(nom, sels)$ peut être réalisé en déclarant $nom \subseteq nom\_de\_base_i$, sous condition que *struct* est monotone dans les deux arguments. Ceci peut être spécifié par :

$$struct(x_1, y_1) \subseteq struct(x_2, y_2) \Leftarrow x_1 \subseteq x_2, y_1 \subseteq y_2$$

Donc, tout objet ayant des sélecteurs couvrant les $sels\_de\_base_i$ pour $i \in [1..m]$ appartiennent à l'ensemble $struct(nom, \cap_{i\in[1..m]} sels\_de\_base_i)$. Un raffinement dans le style de [Cardelli, 1992] peut être fait en remplaçant des sélecteurs sel_i par $sel\_et\_type(sel_i, type_i)$, tel que *sel_et_type* est monotone dans le deuxième argument (spécifié par $sel\_et\_type(x, y_1) \subseteq sel\_et\_type(x, y_2) \Leftarrow y_1 \subseteq y_2$). Avec ce changement, on associe un type avec un sélecteur, comme pour les structures dans les langages impératifs, afin d'éviter des confusions entre structures anonymes avec les mêmes sélecteurs. La monotonie de *sel_et_type* permet en plus d'exploiter le sous-typage pour le raffinement de champs dans des sous-types.

En résumé, nous pouvons dire que les logiques R^n et G^n ne disposent pas d'un mécanisme prédéfini pour le traitement de structures, mais que nous pouvons assez facilement définir des extensions dans ce sens, en analogie avec [Cardelli, 1992], où le système $F_{<:}$ (une extension de système F avec sous-typage [Cardelli *et al.*, 1991]) est utilisé comme base pour une telle définition.

3.5 Méthodes Formelles de Développement

Les méthodes formelles de spécification et de développement d'applications industrielles ont suscité un intérêt croissant durant les cinq dernières années. Elles représentent les méthodologies technologiques clés pour la sécurité, la modularisation, la réutilisation et les aspects légaux des logiciels. Une grande variété de formalismes et d'outils associés est déjà utilisée dans le milieu académique (par exemple OBJ, COQ, ALF, LARCH) et industriel (par exemple Z, SDL, VDM, LOTOS, B).

Actuellement, il y a deux techniques principales pour obtenir des programmes réellement opérationnels à partir de spécifications. Des logiques équationnelles de premier ordre (cf. OBJ, LARCH) peuvent être transformées par une procédure de complétion qui oriente les équations, ce qui en fait des systèmes de réécriture exécutables. Dans les approches d'ordre supérieur basées sur la théorie des types (cf. COQ, ALF), la spécification d'une fonction est représentée comme un type, par l'isomorphisme de Curry-Howard, pour lequel un témoin d'habitation est recherché par une application pas-à-pas de règles de déduction. À partir du témoin, un λ-terme exécutable peut être extrait. L'approche de complétion peut être vue comme une sorte de compilation sûre, car la non-ambiguïté et la terminaison sont prouvées au même moment, tandis que l'approche basée sur la théorie des types est une approche transformationnelle, qui nécessite le raffinement des spécifications.

Dans l'industrie, en général, les spécifications ne sont pas transformées, car elles sont principalement utilisées comme langage précis pour exprimer les exigences d'un client vis-à-vis du producteur. Ou bien, elles donnent une documentation du programme claire et propre (Z, VDM, SDL, LOTOS). Une exception notable est la méthode B, qui offre des techniques de raffinement de spécifications menant en fin de compte à une implantation. Le succès de B semble provenir de l'intuition humaine apparemment élevée pour les modèles basés sur la théorie des ensembles. Nous concluons de cette situation qu'une méthode de construction de programmes à partir de

spécifications avec une sémantique basée sur la théorie des ensembles pourrait aider à rapprocher les méthodes académiques et industrielles.

Un des résultats de cette thèse est un système formel basé sur une logique équationnelle de clauses de Horn étendue par une relation d'appartenance. Ce cadre permet du typage expressif, comparable à la théorie des types de Martin-Löf et du sous-typage comme en système $F_{\leq}$ de Cardelli et Wegner. En sus, il permet d'obtenir des théories d'ordre supérieur par l'utilisation des ensembles comme graphes de fonction. La nature de cette logique nous laisse espérer un compromis réaliste entre les méthodes de spécification basées sur la théorie des ensembles, comme par exemple Z ou B, qui manquent d'un aspect opérationnel, et de systèmes de type d'ordre supérieur basés sur la théorie des types, comme Coq ou Alf, qui sont plus opérationnels. Mais ces derniers sont pour l'instant très abstraits et nous semblent moins intuitifs pour les raisons données dans la section 1.3.2.

4 Plan

Cette thèse est divisée en deux parties, contenant respectivement les fondements théoriques et l'étude de structures de termes décorés. Dans la première partie, nous commençons par des résultats de base (chapitre 2) pour les différentes théories utilisées par la suite, à savoir l'algèbre G, l'$\mathcal{OSE}$-logique, et les logiques R^n et G^n. Puis, nous présentons les techniques de déduction automatique, telles que la réécriture classique (chapitre 3) et les calculs de saturation, des transformations de systèmes de déduction et, finalement, la construction de nouvelles procédures de saturation, adaptées aux logiques R^n et G^n, qui sont aussi étendues par des assertions (chapitre 4).

La deuxième partie est entièrement dédiée à l'étude de calculs permettant la construction de programmes sous forme de systèmes de réécriture de termes décorés. Nous commençons le chapitre 5 par le développement d'une théorie syntaxique de termes décorés, incluant les notions de filtrage et unification stricts. A partir de ces notions, nous définissons dans le chapitre 6 des égalités décorées et deux relations de réécriture, associées à la déduction équationnelle et la relation d'appartenance. Dans le même chapitre, nous expliquons aussi comment on peut obtenir un ensemble de règles qui correspondent à une présentation donnée en algèbre G.

Une procédure de complétion de ces présentations est définie dans le chapitre 7. Nous prouvons, que cette procédure est correcte, et qu'elle établit la confluence du système de réécriture final en cas de succès, si la présentation initiale satisfait la propriété d'héritage de sortes, et si une condition d'équité est respectée pour l'application des règles de complétion. Dans le chapitre 8, nous montrons, comment l'héritage de sortes peut être testé pendant, ou même après, la complétion, en fonction de la forme des règles d'appartenance, appelées règles de décoration, car l'information est mémorisée dans les décorations d'un terme. Le chapitre 9 contient des comparaisons avec d'autres approches aux problèmes de la complétion ordo-sortée.

Dans le chapitre 10 nous développons un calcul de saturation, basé sur des termes décorés, avec une preuve de la complétude réfutationnelle du calcul, en supposant l'héritage de sortes. En même temps, un résultat de complétude pour le test d'héritage de sortes est donnée. Nous finissons ce chapitre avec quelques exemples, illustrant le comportement du calcul entier, qui nous mène à la conclusion, contenant une liste de sujets de recherche futurs. La partie II et donc la thèse elle-même, se termine avec des remarques sur les résultats obtenus. Les dépendances entre les différents chapitres de la thèse sont illustrées par la figure 1.5.

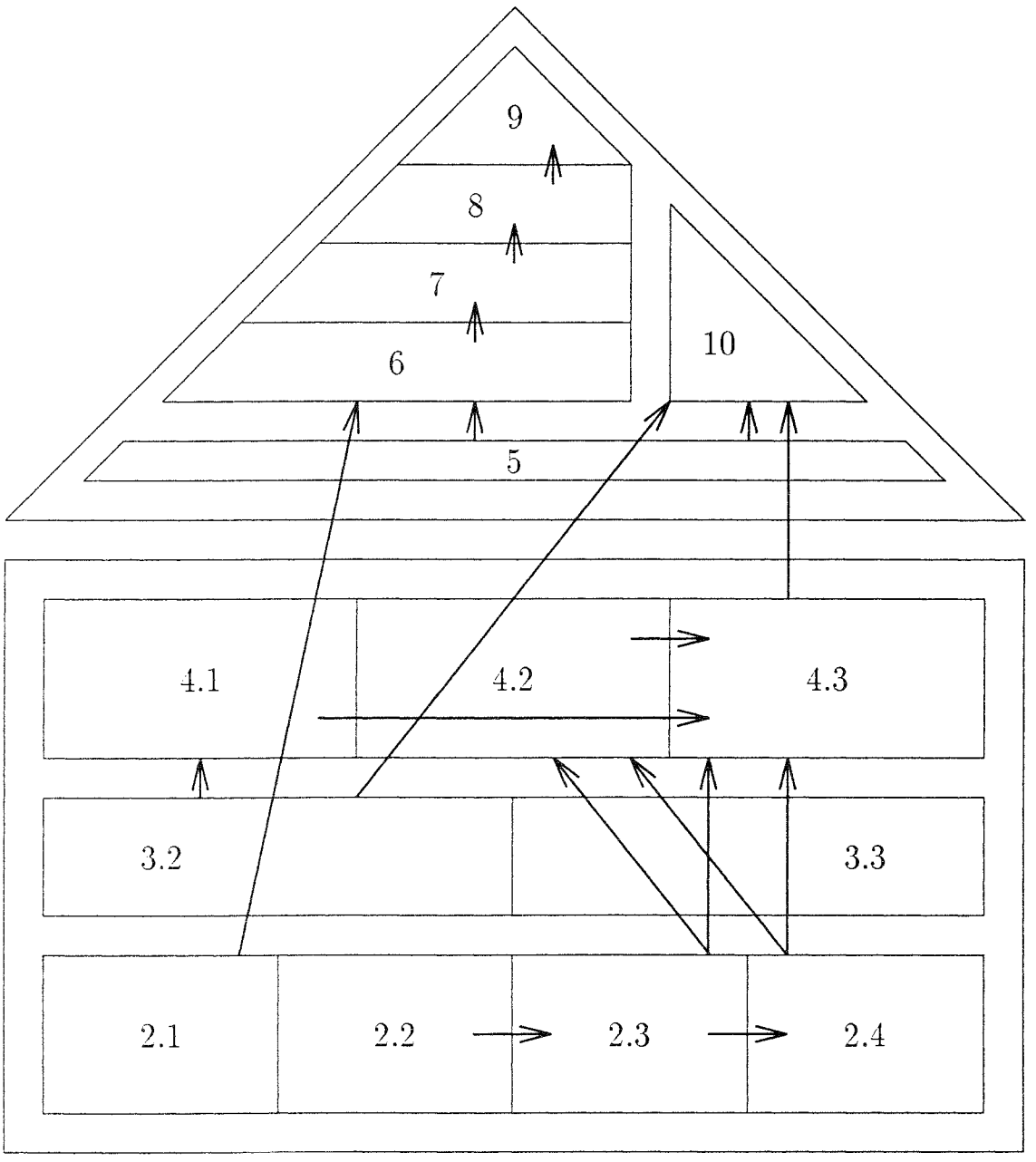


FIG. 1.5 – *Le Graphe de Dépendance de cette Thèse*

5 Publications et Présentation du Contenu

Les résultats des sections 2.3 et 2.4, seront présentés au «Second International Workshop on Higher-Order Algebra, Logic and Term Rewriting» (septembre 1995, Paderborn, Allemagne). Des versions préliminaires ont été présentées au «7th COMPASS Workshop» (février 1995, Sintra, Portugal) et au «6th Nordic Workshop on Programming Theory» [Engberg *et al.*, 1994] (octobre 1994, Aarhus, Danemark).

La section 3.3 est inspirée par l'article [Hintermeier, 1994], contenant une transformation de systèmes de réécriture conditionnels décroissants en systèmes de réécriture ordo-sortés, qui préserve la confluence et la terminaison. Cet article a été publiée dans les actes du «4th International Workshop on Conditional Term Rewriting Systems» (juillet 1994, Jerusalem, Israel).

Une version simplifiée du calcul de saturation basique pour la logique G^n dans la section 4.3 a été présentée au «9th Workshop on Unification» (avril 1995, Sitges, Espagne). Une version de conférence sera soumise dans un futur proche.

Le contenu des chapitres 5 à 7 a été présenté [Hintermeier *et al.*, 1994] au «21st International Colloquium on Automata, Languages and Programming» (juillet 1994, Jerusalem, Israel). Une version préliminaire a été présentée au «7th International Workshop on Unification» (juin 1993, Boston, Etats Unis). Une version quasi-identique des algorithmes de filtrage et d'unification du chapitre 5 a été présentée au «9th Workshop on Specification of Abstract Data Types joint with the 4th COMPASS Workshop» (octobre 1992, Caldes de Malavella, Espagne) [Hintermeier *et al.*, 1992].

Le chapitre 8 à été présenté [Hintermeier *et al.*, 1995a] au «10th Workshop on Specification of Abstract Data Types, joint with the 6th COMPASS Workshop» (juin 1994, Santa Margherita Ligure, Italie). Les chapitres 5 à 9 sont le sujet d'une soumission du numéro spécial «Order-Sorted Equational Reasoning» du «Journal of Symbolic Computation».

Le chapitre 10 n'est pas encore publié, mais nous comptons le soumettre à une conférence.

Part I

Foundations

Chapitre 2

Logical Foundations

Introduction

In this chapter we present four different equational logics, which give the common formal language for the specifications and programs considered in this thesis. For each logic a model notion is given first, which plays a central role in the development of a deduction system. Hence, we start with a brief survey on model theory explaining the role of models from a historic point of view, before we come to Abstract Data Types, motivating initial models as our main goal to prove for each of the considered logical frameworks, and finally to an outline of this chapter.

Model Theory

Models are associated with a formal language and a set of formulas built with symbols from this language. Simply expressed, a model is an interpretation of these symbols, s.t. all formulas in the set hold. Typically models are defined in a structural way, i.e. by induction on the structure of formulas, and a formula holds if it *evaluates* (using an interpretation) to true, but there are also model definitions violating both restrictions, e.g. non-structural models for λ-calculi and multi-valued logics. Also the above phrasing is a bit simplified in the sense that it ignores (possibly present) variables completely. However, it may give already an idea of what a model actually is.

Model theory goes back to the beginning of the 20th century (Löwenheim/Skolem) and won still rising interest in the thirties when Gödel showed the incompleteness of formal systems like the one used by Russell/Whitehead in the Principia Mathematica. The result is simply known as *Gödel's Incompleteness Theorem* for Peano Arithmetic. But incomplete with respect to what ? Completeness may be shown w.r.t. another system, as this is usual with classical logics, where sound and complete deduction systems are well-known. But at some point one has to show completeness for one system. Models actually gave exactly the standard w.r.t. which (in)completeness and also soundness has to be measured:

- A deduction system D is *sound* for a model class K, if for every set S of formulas and all formulas F in the formal language, if F can be derived from S using D, then all models satisfying S also satisfy F.

- D is *complete* for a model class K, if for every set S of formulas and all formulas F in the formal language, F can be derived from S using D if all models satisfying S also satisfy F.

One way to prove soundness is to show that D preserves falsity in K. More formally, assuming D is rule based, then for every inference of a formula *F2* from a formula *F1*, if *F2* is satisfied in a model M of K, then *F1* must be satisfied in M, too. Completeness, however, needs typically more sophisticated arguments like the existence of a finite derivation that may be shown using proof theoretic arguments. In some theories, it is possible to use more abstract, model theoretic arguments, like minimality of the domain of the model. The completeness proof is then simpler. A classical example is equational Horn clause logic. The notions of soundness and completeness thus give one motivation for model theory.

Before model theory came up, logic was typically investigated for the foundations of arithmetic. There, it is natural to ask for consistency only, since the concept of a number is not present as an object itself in the nature, but it is more a concept in our mind. For an essay to define the natural numbers in a consistent way and a long list of errors in previous attempts see [Frege, 1884]. However, with such a definition intuitively in mind, this also gives an informal reference model, called the standard or intended model. Remark that this notion only came up later, of course, with model theory and was not used in the last century. Note also that for real number arithmetic, it is not possible to restrict the domain of such a model to a finitely generated formal language, because of cardinality arguments. Instead, one has to presuppose the existence of more than denumerably many constants. This leads to other model notions. Clearly, the choice of the model class also has great impact on the design of a deduction system.

The next question arising naturally is what completeness w.r.t. some model class intuitively means for computer science. Assume the semantics of a programming language PL is given using some formal language FL and a model class K. Then, completeness of a deduction system w.r.t. FL and K means that we can prove all properties expressible in FL of programs in PL by deduction. This makes completeness a very interesting property for computer science provided FL and K are well-chosen.

Abstract Data Types

In the beginning of the seventies, the ADJ group postulated the principle of abstract data types, which are modular definitions of term based data structures using conditional equations. Consider e.g. the following definition of an abstract data type of lists generated by `cons` and `nil` over some type `E` for list elements, together with an operation `delete` discarding an element from a list, assuming `=/=` is a predicate deciding syntactical difference in `E`:

Example 2.0.1

```
spec LIST  is
           op nil : -> LIST
           op cons : E LIST -> LIST
           op delete : E LIST -> LIST
           var x,y : E
           var l : LIST
           ax delete(x, nil) = nil
           ax delete(x, cons(x, l)) = delete(x, l)
           ax delete(x, cons(y, l)) = cons(y, delete(x, l))
                if x =/= y
endspec
```

The semantics for such an abstract data type (ADT) is the class of initial models in the category of all Σ-models, where Σ is the signature used to define the specification. Initial

models of a given set of formulas are unique up to isomorphism, i.e. up to renaming. Hence, we speak just of *the* initial model in the following. Note that this model coincides with the quotient of the minimal Herbrand model $\mathcal{H}$ by the congruence generated by the equality predicate in $\mathcal{H}$.

An immediate consequence is that any sound and complete deduction system for classical first-order equational Horn logics is complete for the set of valid ground formulas. However, due to the restriction to initial models, inductive theorems have to be considered in order to be complete for all theorems, non-ground ones included. In a deduction system, such an inference rule, often called ω-rule, typically has the following shape:

$$\frac{\textbf{for all } t \ : \ C\{x \mapsto t\}}{C}$$

The automatisation of such rules is an interesting research topic on its own and goes beyond this thesis. However, recall that already the deduction system without an ω-rule is powerful enough to construct the initial model in the case of first-order, equational Horn logics, since this model contains only ground formulas. Therefore, sound and complete deduction systems for all models, not only initial ones, can be seen as a first step necessary to approach the initial model theory, since the latter is a conservative extension of the complete theory by an additional (ω-) deduction rule.

In Example 2.0.1 one may observe that `LIST` actually depends on `E`, since the axioms of `delete` depend on the inequality predicate for `E`. Hence, `LIST` should be a parametric ADT, usually written `LIST[E]` if parametricity is resolved at compile-time, which is then called *genericity*, or `LIST(E)` if parametricity is resolved at run-time, then called dynamic typing in the context of object-oriented programming languages. The usual semantics for the first is based on push-out diagrams in category theory. For the second, we may give a set theoretic semantics with one of the theories introduced towards the end of this chapter. Note that the first is clearly more efficient, at least at run-time, but nevertheless the second gets more and more popular due to the use of object-oriented philosophies in programming.

Remark that ADT semantics is close to the intuition that one might have from type definitions in imperative programming languages like PASCAL, C and especially MODULA: new families of data objects are created by type definitions together with record/variant (PASCAL, MODULA) or structure/union (C) definitions. Something missing, however, is function types, due to the first-order nature of ADTs, and pointers, which are closely related to subterm sharing.

Outline of the Chapter

In the following sections, we will briefly introduce the following theories and show as basic result, if not done yet, the existence of an initial model for any set of formulas in the associated formal language:

- G-algebra of A. Megrelis [Mégrelis, 1990], which is order-sorted algebra with strict partial functions.
- $\mathcal{OSE}$-logic, which is the Horn clause fragment of first-order, order-sorted, equational logic with total functions, similar to [Oberschelp, 1962]. It serves as a basis for the following two logics.
- R^n-logic, which is $\mathcal{OSE}$-logic with predefined membership predicate and a sort structure derived from Russell's theory of types in set theory, s.t. sets are restricted to a nesting depth of n.

- G^n-logic, which is like R^n-logics except that functions may be partial.

Remark that R^n-/G^n-logics are in fact higher-order in that one quantifies over sets of arbitrary nesting depth up to n, which is the original meaning of order given by Russell in 1908 [Russell, 1908]. What nowadays is usually called higher-order logics stems from Church's reformulation [Church, 1940] of Russell's simple theory of types using function spaces instead of sets. Both were at that time really close, since functions usually had a set theoretic semantics as a graph, which is a set of argument/value pairs.

The difference between R^n-/G^n-logic and the two theories above is the non-standard interpretation of sets, i.e. sets in this framework may contain other elements than those declared to be elements, which is closer to the intuition in programming, as we show in the corresponding sections. This is comparable to Henkin-structures for higher-order predicate logic. These structures are often called first-order because of their similar behaviour on the deduction level. Giving an initial semantics to R^n-/G^n-logics makes the two closer again and results in a more primitive but also easier behaviour on the higher-order level.

The construction of an initial model requires the definition of a deduction system first, i.e. there is not a really clean separation between model and proof theory in these cases. However, we do not need to reason about the form of proofs here.

Required notions. In this first part, the reader is supposed to have basic knowledge in algebra, first-order logic and set theory.

Partial functions from A to B, written $A \rightharpoonup B$, are mappings defined over some subset of A, for which the result is in B. $f : A \rightharpoonup B$ is called *total* if f is defined over all elements of A. We then write $f : A \to B$. The *domain* $\mathcal{D}om(f)$ of an n-ary function f is the set of tuples of arguments for which the function is defined. The restriction of a function f to a set X is the function $f_{|X}$, s.t. $f_{|X}(x) = f(x)$ if $x \in \mathcal{D}om(f) \cap X$.

We refer to [Dershowitz et Jouannaud, 1990b], in particular for the definition of the various *orderings* used in this paper. Specifically, lexicographic orderings are written as a tuple $(\leq_1, \ldots, \leq_n)$ of the orderings $\leq_1, \ldots, \leq_n$. These basic notions should be sufficient to understand the definitions in the following.

1 *G*-Algebra

Introduction

In this section we briefly define formulas and presentations in G-algebra. A main feature of this framework w.r.t. other order-sorted algebras is that term declarations, usually seen as part of the (static) signature in classical approaches, become G-algebra formulas and are involved in proofs at the same level as equalities. Also the syntax changes slightly w.r.t. older approaches. What was typically written:

```
sort      Stack
subsort   NeStack < Stack
op        empty_stack :  -> Stack
op        push :  E Stack -> NeStack
```

now becomes:

```
sort      Stack
subsort   NeStack ≤ Stack
var       x::E
var       y::Stack
op        empty_stack :  Stack
op        push(x,y) :  NeStack
```

Here, `push(x,y)` is a term declaration corresponding to the rank of `push` in the old syntax. Additionally to the translation of the old syntax, this allows us now to express facts like `i*i : Int` in complex arithmetic.

Furthermore, functions are considered by default to be strict and partial. This feature is handled by a well-definedness predicate indicating when a value has to be defined. So, e.g. $1/0$ is syntactically acceptable but well-definedness cannot be proved in the standard axiomatisation of division $/$, whereas, e.g., $0/1$ can be proven to be well-defined and equal in the standard way to 0.

More details on G-algebras can be found in [Mégrelis, 1990]. We just give below the formal definitions needed in the following and the sound and complete deduction system from there.

1.1 Formulas and Presentations

Let $\mathcal{S}$ be a finite set of sort symbols, containing always the universal sort symbol Ω. For $\mathcal{X}$ a set of variables, '::' is a binary relation associating to any variable a unique sort in $\mathcal{S}$, denoted $sort(x)$. Then $\mathcal{X}$ is called a $\mathcal{S}$-sorted set of variables. Let also $\mathcal{F}$ be a finite set of function symbols with an arity function $ar()$, associating a natural number to each element of $\mathcal{F}$. Then $\Sigma = (\mathcal{S}, \mathcal{F})$ is called a *signature*. A $(\Sigma, \mathcal{X})$-*term* is either a variable $x \in \mathcal{X}$, or of the form $f(t_1, \ldots, t_n)$ with $f \in \mathcal{F}$, $ar(f) = n$ and $t_1, \ldots, t_n$ being $(\Sigma, \mathcal{X})$-terms. The set of all $(\Sigma, \mathcal{X})$-terms is denoted by $\mathcal{T}(\Sigma, \mathcal{X})$, the ground $(\Sigma, \mathcal{X})$-terms by $\mathcal{T}(\Sigma)$. Notice that except the arity condition, there is no requirement of well-sortedness: the fact that a term is well-sorted will result from an (arbitrarily complicated) proof and is thus a semantical fact rather than a syntactical one. A formula in a G-algebra can be :

- $(EX\ t)$, an existence formula for a term t.
- $(t : A)$, a membership formula for t to be in a sort A, also called variable declaration if $t \in \mathcal{X}$,
- $(t = t')$, an equality of two terms t and t'.

All formulas are implicitly universally quantified over all variables occurring in the formula. In the G-algebra framework, a set of formulas built on a signature Σ containing additionally variable definitions $x :: A$ indicating the sort A of all variables occurring in the formulas is called Σ-*presentation*. Remark that $x :: A$ and $x : B$ correspond with the subsort declaration $A \leq B$. A pair $(\Sigma, \mathcal{P})$ of a signature Σ and a Σ-presentation is called *specification*.

For every signature Σ, a Σ-*algebra* $\mathcal{A}$ is defined by its domain $|\mathcal{A}|$ and by interpretations for each symbol in $\mathcal{S}$ and $\mathcal{F}$:

1. $\forall A \in \mathcal{S}$, the interpretation of A, $A^{\mathcal{A}}$ is a non-empty subset of $|\mathcal{A}|$, and $\Omega^{\mathcal{A}} = |\mathcal{A}|$,
2. $\forall f \in \mathcal{F}$, the interpretation of f, $f^{\mathcal{A}}$ is a partial function $f^{\mathcal{A}} : |\mathcal{A}|^{ar(f)} \rightarrow |\mathcal{A}|$.

A *variable assignment* α of variables in a set $\mathcal{V} \subseteq \mathcal{X}$ in a Σ-algebra $\mathcal{A}$ is a function that assigns to every $x \in \mathcal{V}$, $(x :: A) \in \mathcal{P}$, an element in $A^{\mathcal{A}}$. Hence, given a Σ-algebra, where $\equiv$ is the identity, formulas can be interpreted in the following way:

1. $\mathcal{A} \models_G (EX\ t)$ if for all assignments α of the variables in t, $\alpha(t) \in \Omega^{\mathcal{A}}$
2. $\mathcal{A} \models_G (t : A)$ if for all assignments α of the variables in t, $\alpha(t) \in A^{\mathcal{A}}$
3. $\mathcal{A} \models_G (t = t')$ if for all assignments α of the variables in t and t', $\alpha(t) \equiv \alpha(t')$

Remark that $x :: A$ and $x : B$ imply that A has to be interpreted as subset of B. This is usually written $A \leq B$. A Σ-algebra $\mathcal{A}$ is a *model* of a Σ-presentation $\mathcal{P}$ if for each ϕ in $\mathcal{P}$ being not a variable definition, $\mathcal{A} \models_G \phi$. $\mathcal{P} \models_G \phi$ if ϕ is true in all models of $\mathcal{P}$. Note that if a function is not defined for some argument, the result of this function application is not in the interpretation of any sort and two undefined results of function applications are never equal.

1.2 Deduction

The system of deduction rules **GA** for G-algebras is shown in Figure 2.1. When a formula ϕ can be deduced from the formulas of a presentation $\mathcal{P}$ using these deduction rules, this is denoted $\mathcal{P} \vdash_{GA} \phi$. The substitutions σ mentioned in the deduction system are supposed to be *conform* with the current presentation which means that, for all $(x \mapsto t) \in \sigma$, we have $\mathcal{P} \vdash_{GA} (t : A)$ if $(x :: A) \in \mathcal{P}$. These rules are proved to be sound and complete in [Mégrelis, 1990] i.e. for any presentation $\mathcal{P}$ and formula ϕ:

$$\mathcal{P} \models_G \phi \Leftrightarrow \mathcal{P} \vdash_{GA} \phi.$$

Notice that membership in Ω and existence formulas are equivalent statements and thus this set of rules can be simplified if needed. We denote the set $\{\phi \mid \mathcal{P} \vdash_{GA} \phi\}$, also called the theory of $\mathcal{P}$, by $Th(\mathcal{P})$.

The Σ-term algebra $\mathcal{T}_\Sigma$ for a Σ-presentation is a Σ-algebra $\mathcal{A}$, s.t. the interpretation function $.^{\mathcal{A}}$ is the identity for each term $t \in \mathcal{T}(\Sigma)$ with $\mathcal{P} \vdash EX\ t$. Given a Σ-presentation $\mathcal{P}$, the congruence $\equiv$ is defined as $\{(s,t) \in \mathcal{T}(\Sigma) \times \mathcal{T}(\Sigma) \mid \mathcal{P} \vdash_{GA} s = t\}$. The quotient term algebra $\mathcal{T}_{\Sigma/\equiv}$, is a Σ-algebra which is proved initial in the class of Σ-algebras satisfying the presentation $\mathcal{P}$ (see [Mégrelis, 1990]), provided all sorts contain at least one ground term in $\mathcal{T}_\Sigma$.

2 $\mathcal{OSE}$-Logics

Introduction

Order-sorted equational logics were first defined by Oberschelp [Oberschelp, 1962], in order to give a natural way of expression for mathematical purposes. Starting with many-sorted logics, as outlined by Herbrand [Herbrand, 1971] and completed by Schmidt [Schmidt, 1938], where the domain of a model is divided into subsets, called sorts, order-sorted theories allow additionally to specify an inclusion order on these sets. This results in intuitively more expressiveness, although one can easily prove that formally no gain could be achieved, since any order-sorted logic presentation can be transformed into an unsorted one [Oberschelp, 1962], where sorts are translated into monadic predicates.

Globality	$EX\ t$	$\Longrightarrow$	$t : \Omega$
VariableMembership	$x :: A$	$\Longrightarrow$	$x : A$
ExSubterm	$EX\ t[u]$	$\Longrightarrow$	$EX\ u$
ExMembership	$t : A$	$\Longrightarrow$	$EX\ t$
ExEquality	$t = t'$	$\Longrightarrow$	$EX\ t$
ExReplacement	$EX\ t[u], u = v$	$\Longrightarrow$	$EX\ t[v]$
MeReplacement	$t[u] : A, u = v$	$\Longrightarrow$	$t[v] : A$
EqReplacement	$t[u] = w, u = v$	$\Longrightarrow$	$t[v] = w$
ExSubstitutivity	$EX\ t$	$\Longrightarrow$	$EX\ \sigma(t)$
MeSubstitutivity	$t : A$	$\Longrightarrow$	$\sigma(t) : A$
EqSubstitutivity	$t = t'$	$\Longrightarrow$	$\sigma(t) = \sigma(t')$
Reflexivity	$EX\ t$	$\Longrightarrow$	$t = t$
Symmetry	$u = v$	$\Longrightarrow$	$v = u$
Transitivity	$u = v, v = w$	$\Longrightarrow$	$u = w$

Figure 2.1: **GA** : Deduction rules for G-algebra

In the eighties, order-sorted formalisms had a come-back in automated theorem proving (cf. e.g. [Walther, 1983; Schmidt-Schauß, 1987]) as well as in form of specification and programming languages ([Goguen et Meseguer, 1992; Kirchner *et al.*, 1988a; Aït-Kaci et Nasr, 1985]). In theorem provers, the main achievement was a reduction in search space, due to sorted variable unification. On the language level, the principal advantages are analogy to mathematical tradition – think of natural numbers and integers or reals – and enhanced readability.

In this section, we work out order-sorted equational Horn logics, a fragment of the one defined in [Oberschelp, 1962], in order to prove some basic results that can be reused for meta-logical investigations in the sense of Russell's theory of types [Whitehead et Russell, 1925]. Instead of using the original definitions of [Oberschelp, 1962], we preferred for convenience to take the syntax of the polymorphic order-sorted Horn clause calculus presented in [Hanus, 1991].

2.1 Syntax and Semantics

Signatures, Interpretations and Homomorphisms

A *signature* Σ is a triple $(\mathbf{S}, \leq_S, \mathbf{F}, \mathbf{R})$, where $\mathbf{S}$ stands for a finite set of *sort symbols* with associated partial order $\leq_S$, $\mathbf{F}$ is a set of *function symbols* and $\mathbf{R}$ is a set of *predicate symbols* including $=$, s.t. $\mathbf{S}$, $\mathbf{F}$, and $\mathbf{R}$ are finite and disjoint. For any $f \in \mathbf{F}$, we suppose a (possibly empty) set $ranks(f)$ of ranks $(f : s_1, \ldots s_n \rightarrow s)$ to be given, s.t. $s_1, \ldots, s_n, s \in \mathbf{S}$. The ranks of predicate symbols p, written $ranks(p)$, are similarly assumed to be given as $(p : s_1, \ldots, s_n)$, s.t. $s_1, \ldots, s_n \in \mathbf{S}$ if $p \in \mathbf{R}$. For $=$ the arities are predefined as $(= : s\ s)$ for all $s \in \mathbf{S}$. We also write $ar(f) = n$ and $ar(p) = n$ for short, if f and p take n arguments, respectively. Since the set of ranks $ranks(f)$ and $ranks(p)$ may be empty, we require at least the existence of the number of arguments $ar(f)$ or $ar(p)$. These are just conventions in order to simplify definitions.

Given an $\mathbf{S}$-sorted family of variables $\mathbf{X} = \{\mathbf{X}_s\}_{s \in \mathbf{S}}$, the set of *terms* of sort $s \in \mathbf{S}$ is the minimal set $\mathbf{T}_s(\mathbf{X})$ satisfying:

$$\mathbf{T}_s(\mathbf{X}) = \bigcup_{s' \leq_S s} \mathbf{T}_{s'}(\mathbf{X}) \cup \mathbf{X}_s \cup \{f(t_1, \ldots, t_n) \mid \begin{array}{l}(f : s_1, \ldots, s_n \rightarrow s) \in ranks(f) \\ \text{and } \forall i \in [1..n], t_i \in \mathbf{T}_{s_i}(\mathbf{X})\}\end{array}$$

The set of *extended terms* $\mathbf{ET(X)}$ is the set of unsorted terms over $\mathbf{F}$ and $\mathbf{X}$, i.e. those respecting the arities only (cf. [Waldmann, 1989a]). Instead of $\exists s \in \mathbf{S}, x \in \mathbf{X}_s$ we also write $x \in \mathbf{X}$ for short. $\mathbf{T(X)}$ simply stands for $\bigcup_{s \in \mathbf{S}} \mathbf{T}_s(\mathbf{X})$ and $\mathbf{T}$ stands for the set of ground terms in $\mathbf{T(X)}$. If $x \in \mathbf{X}_s$, we may also write $sort(x)$ instead of s.

Let $sorts(t) = \{s \mid t \in \mathbf{T}_s(\mathbf{X})\}$. If all terms t in $\mathbf{T(X)}$ have a least element[4] w.r.t. $\leq_S$ in $sorts(t)$, then the signature Σ is called *regular*. It is well-known that regularity is decidable.

Definition 2.2.1 *A* Σ-interpretation $\mathbf{A}$ *is a pair* $(C^{\mathbf{A}}, .^{\mathbf{A}})$ *of a* carrier $C^{\mathbf{A}}$ *and an interpretation function* $.^{\mathbf{A}}$ *defined over all sort, function and predicate symbols, s.t.:*

- $\forall s \in \mathbf{S}, s^{\mathbf{A}} \subseteq C^{\mathbf{A}}$, $s^{\mathbf{A}} \neq \emptyset$,
- $\forall s, s' \in \mathbf{S}$, $s^{\mathbf{A}} \subseteq s'^{\mathbf{A}}$ *if* $s \leq_S s'$,
- $\forall f \in \mathbf{F}, ar(f) = n, f^{\mathbf{A}} : (C^{\mathbf{A}})^n \rightharpoonup C^{\mathbf{A}}$ *is a partial function, s.t.:*

  $(s_1^{\mathbf{A}}, \ldots, s_n^{\mathbf{A}}) \subseteq \mathcal{D}om(f^{\mathbf{A}})$ and $f^{\mathbf{A}}(s_1^{\mathbf{A}}, \ldots, s_n^{\mathbf{A}}) \subseteq s^{\mathbf{A}}$ if $(f : s_1, \ldots, s_n \rightarrow s) \in ranks(f)$,

- $\forall p \in \mathbf{R} \setminus \{=\}, ar(p) = n, p^{\mathbf{A}} \subseteq (C^{\mathbf{A}})^n$.
- $=^{\mathbf{A}}$ *is the diagonal on* $C^{\mathbf{A}}$.

The class of all Σ*-interpretation is written* $Interp(\Sigma)$. *Let* $\mathbf{A}$ *be a* Σ*-interpretation and* $\mathbf{X}$ *a* $\mathbf{S}$*-sorted variable set. An* $(\mathbf{A}, \mathbf{X})$-variable assignment *is a function* $\delta : \mathbf{X} \rightarrow C^{\mathbf{A}}$, *s.t.* $\delta(x) \in s^{\mathbf{A}}$ *if* $x \in \mathbf{X}_s$.

Remark that no undefined value is supposed to be contained in some $s^{\mathbf{A}}$, i.e. $f^{\mathbf{A}}$ has to be defined for all values in $(s_1^{\mathbf{A}} \times \ldots \times s_n^{\mathbf{A}})$ when we write $f^{\mathbf{A}}(s_1^{\mathbf{A}}, \ldots, s_n^{\mathbf{A}}) \subseteq s^{\mathbf{A}}$. Note furthermore that the ranks of predicates are not used in this definition. It does not play any role for interpretations, but for the the definition of the class of formulas we consider. It may as well be used to restrict models, s.t. predicates have to be false outside of the interpretation of their ranks, but this has undesirable side-effects with overloading. Hence, we do not consider this idea. An ordering over Σ-interpretations is given by Σ-homomorphisms, defined as follows:

Definition 2.2.2 *Let* $\mathbf{A}, \mathbf{B}$ *be* Σ*-interpretations. A* Σ-homomorphisms $h : \mathbf{A} \rightarrow \mathbf{B}$ *is a function satisfying:*

- $h(C^{\mathbf{A}}) \subseteq C^{\mathbf{B}}$,
- $\forall s \in \mathbf{S}. h(s^{\mathcal{A}}) \subseteq s^{\mathbf{B}}$,
- $\forall f \in \mathbf{F}, h(\mathcal{D}om(f^{\mathcal{A}})) \subseteq \mathcal{D}om(f^{\mathcal{B}})$,

  $$h(f^{\mathbf{A}}(\xi_1, \ldots, \xi_n)) = f^{\mathbf{B}}(h(\xi_1), \ldots, h(\xi_n)) \text{ if } (\xi_1, \ldots, \xi_n) \in \mathcal{D}om(f^{\mathbf{A}}) \text{ and}$$

- $\forall p \in \mathbf{R}, h(p^{\mathbf{A}}) \subseteq p^{\mathbf{B}}$.

[4] In other words: $sorts(t)$ must contain a greatest lower bound.

As usual, one can easily verify that the composition of two Σ-homomorphisms gives again a Σ-homomorphism. Moreover, composition is associative and there is a unit, the identity homomorphism $id_{\mathbf{A}}$, for each Σ-interpretation $\mathbf{A}$. Hence, Σ-interpretations and -homomorphisms form a category named $Cat(\Sigma)$. Term interpretations for Σ are needed for the construction of the initial model in $Cat(\Sigma)$.

Definition 2.2.3 *Let* $\mathbf{V}$ *be an arbitrary* $\mathbf{S}$*-sorted variable set* $\{\mathbf{V}_s \mid s \in \mathbf{S}\}$. *The* (Σ, V)-term interpretation $\mathbf{T}_{(\Sigma,\mathbf{V})}$ *is the interpretation* T, *defined as follows:*

- $C^T = \bigcup_{s \in \mathbf{S}} s^T$,
- $\forall s \in \mathbf{S}, s^T = \bigcup_{s' \leq_{\mathcal{S}} s} s'^T \cup \mathbf{V}_s \cup \{f(t_1, \ldots, t_n) \mid (f : s_1, \ldots, s_n \to s) \in ranks(f)$ and $\forall i \in [1..n], t_i \in {s_i}^T\}$,

  s.t. s^T *is minimal,*

- $\forall f \in \mathbf{F}$, *if* $(f : s_1, \ldots, s_n \to s) \in ranks(f)$, *then:*
  - $Dom(f^T) = s_1^T \times \ldots \times s_n^T$ *and*
  - $f^T(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$, *if* $\bigwedge_{i \in [1..n]} t_i \in {s_i}^T$,

  and
- $\forall p \in \mathbf{R}$, *if* $(p : s_1, \ldots, s_n) \in ranks(p)$, *then:*
  - $Dom(p^T) = s_1^T \times \ldots \times s_n^T$ *and*
  - $p^T = \emptyset$ *if* p *is different from* $=$, *which is defined as diagonal on* C^T.

Then, the Σ-ground term interpretation $\mathbf{T}_\Sigma$ *is the* $(\Sigma, \emptyset)$ *- term interpretation, the* free Σ-term interpretation $\mathbf{T}_\Sigma(\mathbf{X})$ *is the* $(\Sigma, \mathbf{X})$*-term interpretation.*

A first result is the extendibility of Σ-interpretations to homomorphisms from $\mathbf{T}_\Sigma$ to $\mathbf{A}$.

Lemma 2.2.4 *Let* $\mathbf{A}$ *be a* Σ*-interpretation.*
Then, there is a unique Σ*-homomorphism* $h : \mathbf{T}_\Sigma \to \mathbf{A}$.

Proof: Let h be defined for all $f \in \mathbf{F}$ by $h(f(t_1, \ldots, t_n)) = f^{\mathbf{A}}(h(t_1), \ldots, h(t_n))$ whenever $f(t_1, \ldots, t_n) \in \mathbf{T}(\mathbf{X})$.

Clearly, well-typed terms are mapped to objects in $C^{\mathbf{A}}$, i.e. $h(C^{\mathbf{T}_\Sigma}) \subseteq C^{\mathbf{A}}$. Furthermore, $\forall f \in \mathbf{F}. h(f(t_1, \ldots, t_n)) = f^{\mathbf{A}}(h(t_1), \ldots, h(t_n))$ if $(t_1, \ldots, t_n) \in Dom(f^{(\mathbf{T}_\Sigma)})$ by definition of h and $\forall p \in \mathbf{R}, h(p^{(\mathbf{T}_\Sigma)}) \subseteq p^{\mathbf{A}}$, since $p^{(\mathbf{T}_\Sigma)} = \emptyset$.

We show that h fulfils the last remaining condition for Σ-homomorphisms, i.e. $\forall t \in s^{(\mathbf{T}_\Sigma)}, h(t) \in s^{\mathbf{A}}$, by structural induction on the term t. Let $t = f(t_1, \ldots, t_n) \in s^{(\mathbf{T}_\Sigma)}$. Then there are $s_1, \ldots s_n, s \in \mathbf{S}$, s.t. $(f : s_1, \ldots, s_n \to s) \in ranks(f)$ and $\bigwedge_{i \in [1..n]} t_i \in {s_i}^{(\mathbf{T}_\Sigma)}$. By the induction hypothesis, we get $h(t_i) \in s_i^{\mathbf{A}}$ and therefore $f^{\mathbf{A}}(h(t_1), \ldots, h(t_n)) \in s^{\mathbf{A}}$, since $f^{\mathbf{A}}$ must map $(s_1^{\mathbf{A}}, \ldots, s_n^{\mathbf{A}})$ into $s^{\mathbf{A}}$.

Suppose there is another Σ-homomorphism $g : \mathbf{T}_\Sigma \to \mathbf{A}$. Then, $h = g$ can be shown by structural induction on the argument $t \in C^{(\mathbf{T}_\Sigma)}$: Let $t = f(t_1, \ldots, t_n)$. Hence, we know that $g(f(t_1, \ldots, t_n)) = f^{\mathbf{A}}(g(t_1), \ldots, g(t_n)) = f^{\mathbf{A}}(h(t_1), \ldots, h(t_n)) = h(f(t_1, \ldots, t_n))$, since g and h are both Σ-homomorphisms and the induction hypothesis providing $g(t_i) = h(t_i)$ for all $i \in [1..n]$. □

We come to the following conclusion:

Corollary 2.2.5 $\mathbf{T}_\Sigma$ *is initial in the category of all* Σ*-interpretations.*

In order to get unique semantics for formula evaluation, we prove the following:

Lemma 2.2.6 *Let* $\mathbf{A}$ *be a* Σ*-interpretation and* δ *a* $(\mathbf{A}, \mathbf{X})$*-variable assignment. Then, there is a unique* Σ*-homomorphism* $h : \mathbf{T}_\Sigma(\mathbf{X}) \to \mathbf{A}$ *with* $\forall x \in \mathbf{X}, h(x) = \delta(x)$.

Proof: Let h be defined as follows:

- $\forall x \in \mathbf{X}, h(x) = \delta(x)$ and
- $\forall f \in \mathbf{F}, h(f(t_1, \ldots, t_n)) = f^{\mathbf{A}}(h(t_1), \ldots, h(t_n))$ if $f(t_1, \ldots, t_n) \in \mathbf{T}(\mathbf{X})$.

Clearly, well-typed terms are mapped to objects in $C^{\mathbf{A}}$, i.e. $h(\mathbf{T}(\mathbf{X})) \subseteq C^{\mathbf{A}}$. Furthermore, $\forall f \in \mathbf{F}, h(f(t_1, \ldots, t_n)) = f^{\mathbf{A}}(h(t_1), \ldots, h(t_n))$ if $(t_1, \ldots, t_n) \in Dom(f^{\mathbf{T}_\Sigma(\mathbf{X})})$ by definition of h and $\forall p \in \mathbf{R}, h(p^{\mathbf{T}_\Sigma(\mathbf{X})}) \subseteq p^{\mathbf{A}}$, since $p^{\mathbf{T}_\Sigma(\mathbf{X})} = \emptyset$.

We show that h fulfils the last remaining condition for Σ-homomorphisms, i.e. $\forall t \in \mathbf{T}_s(\mathbf{X}), h(t) \in s^{\mathbf{A}}$, by structural induction on the term t:

- If $t = x \in \mathbf{X}$, then $x \in \mathbf{X}_s$ for some $s \in \mathbf{S}$. Therefore, $h(x) = \delta(x) \in s^{\mathbf{A}}$ by definition of a $(\mathbf{A}, \mathbf{X})$-variable assignment.
- If $t = f(t_1, \ldots, t_n) \in \mathbf{T}_s(\mathbf{X})$, then there are $s_1, \ldots s_n, s \in \mathbf{S}$, s.t. $(f : s_1, \ldots, s_n \to s) \in ranks(f)$ and $\bigwedge_{i \in [1..n]} t_i \in \mathbf{T}_{s_i}(\mathbf{X})$. By the induction hypothesis, we get $h(t_i) \in s_i^{\mathbf{A}}$ and therefore $f^{\mathbf{A}}(h(t_1), \ldots, h(t_n)) \in s^{\mathbf{A}}$, since $f^{\mathbf{A}}$ must map $(s_1^{\mathbf{A}}, \ldots, s_n^{\mathbf{A}})$ into $s^{\mathbf{A}}$.

Suppose there is another Σ-homomorphism $g : \mathbf{T}_\Sigma(\mathbf{X}) \to \mathbf{A}$ with $\forall x \in \mathbf{X}, g(x) = \delta(x)$. Then, $h = g$ can be shown by structural induction on the argument $t \in \mathbf{T}_\Sigma(\mathbf{X})$:

- $t = x \in \mathbf{X}$, i.e. $g(x) = \delta(x) = h(x)$ by requirement.
- $t = f(t_1, \ldots, t_n)$, i.e. since g and h are Σ-homomorphisms and the induction hypothesis providing $g(t_i) = h(t_i)$ for all $i \in [1..n]$, we get:

$$g(f(t_1, \ldots, t_n)) = f^{\mathbf{A}}(g(t_1), \ldots, g(t_n)) = f^{\mathbf{A}}(h(t_1), \ldots, h(t_n)) = h(f(t_1, \ldots, t_n))$$

□

Therefore, we use in the following the same symbol δ for the variable assignment and its homomorphic extension.

Equational Horn Clauses and Models

In the following, we assume that $\mathbf{R}$ contains a special relation symbol '=' with $(s, s) \in ranks(=)$ for all $s \in \mathbf{S}$. For all $p \in \mathbf{R}$, we say that $p(t_1, \ldots t_n)$ is a $(\Sigma, \mathbf{X})$-*atom*, if $(s_1, \ldots, s_n) \in ranks(p)$ and $\forall i \in [1..n], t_i \in \mathbf{T}_{s_i}(\mathbf{X})$. A $(\Sigma, \mathbf{X})$-*goal* is a conjunction $L_1 \wedge \ldots \wedge L_n$ of $(\Sigma, \mathbf{X})$-atoms L_i, $i \in [1..n]$, and a $(\Sigma, \mathbf{X})$-*clause* is of the form $L \leftarrow G$, where L is $(\Sigma, \mathbf{X})$-atom and G is a $(\Sigma, \mathbf{X})$-goal. The conjunction operator $\wedge$ is supposed to be associative and commutative. A set of Σ-clauses $\mathbf{P}$ is called a (Σ-)*presentation*.

Given a Σ-interpretation $\mathbf{A}$ and a variable assignment $\delta : \mathbf{X} \to \mathbf{A}$, we write:

- $\mathbf{A}, \delta \models_{OSE} p(t_1, \ldots t_n)$, if $p(t_1, \ldots t_n)$ is a $(\Sigma, \mathbf{X})$-atom and $(\delta(t_1), \ldots, \delta(t_n)) \in p^{\mathbf{A}}$.

- $\mathbf{A}, \delta \models_{OSE} G_1 \wedge G_2$, if G_1, G_2 are $(\Sigma, \mathbf{X})$-goals, $\mathbf{A}, \delta \models_{OSE} G_1$ and $\mathbf{A}, \delta \models_{OSE} G_2$.
- $\mathbf{A}, \delta \models_{OSE} L \leftarrow G$, if $L \leftarrow G$ is a $(\Sigma, \mathbf{X})$-clause and $\mathbf{A}, \delta \models_{OSE} L$ whenever $\mathbf{A}, \delta \models_{OSE} G$.
- $\mathbf{A}, \delta \models_{OSE} \{C_1, \ldots, C_n\}$, if C_i is a $(\Sigma, \mathbf{X})$-clause for $i \in [1..n]$ and $\mathbf{A}, \delta \models_{OSE} C_i$ for all $i \in [1..n]$.

Furthermore, $\mathbf{A}, \mathbf{X} \models_{OSE} F$, stands for $\mathbf{A}, \delta \models_{OSE} F$ for all δ being a $(\mathbf{A}, \mathbf{X})$-variable assignment, where F is a $(\Sigma, \mathbf{X})$-atom, -goal or -clause.

If $\mathbf{A}, \mathbf{X} \models_{OSE} L$, then we say that L is *valid* in $\mathbf{A}$. Given a set of $(\Sigma, \mathbf{X})$-clauses $\mathbf{P}$, also called presentation, $\mathbf{A}$ is called a *model* of $\mathbf{P}$ if all clauses in $\mathbf{P}$ are valid in $\mathbf{A}$ and $=$ is interpreted as the diagonal in $C^{\mathbf{A}} \times C^{\mathbf{A}}$. $Mod(\mathbf{P})$ stands for the set of all models of $\mathbf{P}$. A $(\Sigma, \mathbf{X})$-goal G is *valid* in $\mathbf{P}$ w.r.t. $\mathbf{X}$, written $(\Sigma, \mathbf{P}, \mathbf{X}) \models_{OSE} G$, if $\mathbf{A}, \mathbf{X} \models_{OSE} G$ for all $\mathbf{A} \in Mod(\mathbf{P})$. Let $CMod(\mathbf{P})$ stand for the full subcategory of $Cat(\Sigma)$ with $Mod(\mathbf{P})$ as set of objects.

2.2 Substitutions, Non-Empty Sorts and Deduction

Before we formulate the deduction rules, we need to define substitutions.

Definition 2.2.7 *A Σ-substitution σ is a Σ-homomorphism σ from $\mathbf{T}(\mathbf{X})$ into $\mathbf{T}(\mathbf{X})$, s.t. its domain $Dom(\sigma) = \{x \mid \sigma(x) \neq x\}$ is finite. The set of all such substitutions is denoted by $SUBST_\Sigma(\mathbf{X})$. The identity substitution on $\mathbf{X}$ is written $id_{\mathbf{X}}$, i.e. $Dom(id_{\mathbf{X}}) = \emptyset$.*

Remark that the notion of Σ-homomorphism guarantees the well-typedness of the variable image. Clearly, Lemma 2.2.4 implies that it is sufficient to represent a substitution σ by the variable mapping for all $x \in Dom(\sigma)$, since the extension on all terms in $\mathbf{T}(\mathbf{X})$ is uniquely determined. An object (term, atom, clause, goal etc.) o is called *instance* of another object o', on condition that all variables of o, o' are from $\mathbf{X}$, if there exists a substitution $\sigma \in SUBST_\Sigma(\mathbf{X})$, s.t. $\sigma(o') = o$.

From the definition of substitutions and models, we get the following Lemma, which is very useful for the initiality proof of the quotient term model.

Lemma 2.2.8 *Let $\mathbf{A}$ be a Σ-interpretation, G a $(\Sigma, \mathbf{X})$-goal, $\sigma \in SUBST_\Sigma(\mathbf{X})$ and δ be a variable assignment for $\mathbf{X}$ in $\mathbf{A}$.*
Then $\mathbf{A}, \delta \models_{OSE} \sigma(G)$ iff $\mathbf{A}, \delta \circ \sigma \models_{OSE} G$.

Proof: Clearly, $\delta' = \delta \circ \sigma$ is a $(\mathbf{A}, \mathbf{X})$-variable assignment. Let $p(\ldots, t, \ldots) \in G$. Then:

$$\begin{aligned} \mathbf{A}, \delta \models_{OSE} \sigma(p(\ldots, t, \ldots)) &\text{ iff } \mathbf{A}, \delta \models_{OSE} p(\ldots, \sigma(t), \ldots) \\ &\text{iff } (\ldots, \delta(\sigma(t)), \ldots) \in p^{\mathbf{A}} \\ &\text{iff } (\ldots, \delta'(t), \ldots) \in p^{\mathbf{A}} \\ &\text{iff } \mathbf{A}, \delta' \models_{OSE} p(\ldots, t, \ldots) \end{aligned}$$

□

Now we can formulate an equational Horn clause calculus for our logic. The deduction rules can be found in Figure 2.2. We write $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L$ if $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow \emptyset$ can be deduced using these rules. Furthermore, we write $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} \bigwedge_{i \in [1..n]} L_i$ if $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L_i$ for all $i \in [1..n]$.

In order to prove the soundness of the rules in Figure 2.2, we need an additional restriction, which guarantees that all models interpret all sorts as non-empty sets.

Definition 2.2.9 *A signature $\Sigma = (\mathbf{S}, \mathbf{F}, \mathbf{R})$ has* non-empty sorts, *if every Σ-interpretation interprets every sort $s \in \mathbf{S}$ as a non-empty set.*

Reflexivity	$\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} x = x}$ if $x \in \mathbf{X}$
Axioms	$\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow G}$ if $(L \Leftarrow G) \in \mathbf{P}$ is a $(\Sigma, \mathbf{X})$–clause
Substitutivity	$\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow G}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} \sigma(L) \Leftarrow \sigma(G)}$ if $\sigma \in SUBST_{\Sigma}(\mathbf{X})$
Cut	$\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow G \wedge L',\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L' \Leftarrow G'}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow G \wedge G'}$
Paramodulation	$\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L[s] \Leftarrow G,\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} (s = t) \Leftarrow G'}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L[t] \Leftarrow G \wedge G'}$

Figure 2.2: **OSL** : Horn Clause Deduction Rules for $\mathcal{OSE}$-logics

NESubsort	$\dfrac{(A, N \cup \{s\})}{(A, N \cup \{s, s'\})}$ if $s \leq_{\mathbf{S}} s'$ and $s' \notin N \cup \{s\}$
Nonempty	$\dfrac{(A \cup \{(f : s_1, \ldots, s_n \to s)\}, N \cup \{s_1, \ldots, s_n\})}{(A \cup \{(f : s_1, \ldots, s_n \to s)\}, N \cup \{s_1, \ldots, s_n, s\})}$ if $s \notin N \cup \{s_1, \ldots, s_n\}$

Figure 2.3: **NES** : Inference Rules To Decide Non-Emptiness of Sorts

Non-emptiness of sorts can easily be decided by the rules **NES** shown in Figure 2.3.

Lemma 2.2.10 *The rules* **NESubsort** *and* **Nonempty** *can only be applied a finite number of times starting with the pair* $(A, \emptyset)$*, where* $A = \bigcup_{f \in \mathbf{F}} ranks(f)$*.*

Proof: Immediate consequence of the finiteness of $\mathbf{S}$. □

Theorem 2.2.11 *Let* $\Sigma = (\mathbf{S}, \leq_{\mathbf{S}}, \mathbf{F}, \mathbf{R})$ *be a signature and* $A = \bigcup_{f \in \mathbf{F}} ranks(f)$*.*
Then $(A, \emptyset) \overset{*}{\vdash}_{NES} (A, \mathbf{S})$ *iff* Σ *has non-empty sorts.*

Proof: $\Rightarrow$: Direct consequence of the definition of Σ-models. $\Leftarrow$: Consider $\mathbf{T}_{\Sigma}$, i.e. take a ground term of minimal depth for each sort and prove that we need maximally as many steps as this term is deep to prove that the sort is non-empty. Since the set of sorts is finite, we get the completeness. □

Clearly, we get:

Corollary 2.2.12 *Given a signature* Σ*, it is decidable whether* Σ *has non-empty sorts or not.*

Remark that in order to decide unification, which is used with more practical calculi, the signature has to fulfill a *regularity* property saying that for all terms $t \in \mathbf{T}(\mathbf{X})$, the set of sorts t belongs to must contain a greatest lower bound w.r.t. $\leq_{\mathbf{S}}$. For the moment, we will use our axiomatisation only for the construction of the initial model and can therefore ignore this (practically very important) detail.

Theorem 2.2.13 *Let* $\mathbf{P}$ *be set of* $(\Sigma, \mathbf{X})$*-clauses with non-empty sorts. The rules in Figure 2.2 are sound, i.e. if* $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L$ *then* $(\Sigma, \mathbf{P}, \mathbf{X}) \models_{OSE} L$*, assuming* L *be a* $(\Sigma, \mathbf{X})$*-atom.*

Proof: Let $\mathbf{A}$ be a model for $(\Sigma, \mathbf{P})$. The proof is an induction on the length of the derivation, i.e. assuming the validity of the premiss(es) of a rule, we show the one of the conclusion. This clearly covers all clauses occurring in the inference $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L$.

- **Axioms** : Since $\mathbf{A}$ is a model of $\mathbf{P}$, we have $\mathbf{A}, \mathbf{X} \models_{OSE} L \Leftarrow G$ and therefore $\mathbf{A}, \delta \models_{OSE} L \Leftarrow G$ for all $(\mathbf{A}, \mathbf{X})$-variable assignments δ.
- **Substitutivity** : Let $\sigma \in SUBST_{\Sigma}(\mathbf{X})$. If δ' is a $(\mathbf{A}, \mathbf{X})$-variable assignment, then $\delta = \delta' \circ \sigma$ is a $(\mathbf{A}, \mathbf{X})$-assignment, and due to Lemma 2.2.8, we get $\mathbf{A}, \delta \models_{OSE} G$ whenever $\mathbf{A}, \delta' \models_{OSE} \sigma(G)$. Consequently, $\mathbf{A}, \delta \models_{OSE} L$ and once more by Lemma 2.2.8, we get $\mathbf{A}, \delta' \models_{OSE} \sigma(L)$, whenever $\mathbf{A}, \delta' \models_{OSE} \sigma(G)$. Hence, $\mathbf{A}, \delta' \models_{OSE} \sigma(L) \Leftarrow \sigma(G)$.
- **Cut** : If $G \wedge G' = \emptyset$, then we know that $\delta(L')$ holds for all $(\mathbf{A}, \mathbf{X})$-variable assignments δ. Remark that the non-emptiness condition guarantees the existence of such a δ. Therefore, $\delta(L)$ must also hold in $\mathbf{A}$, because of $\mathbf{A} \models_{OSE} L \Leftarrow L'$.

  Suppose $G \wedge G' \neq \emptyset$ and there is a $(\mathbf{A}, \mathbf{X})$-variable assignment δ with $\mathbf{A}, \delta \models_{OSE} G \wedge G'$. If δ does not exist, then the rule is trivially sound. Otherwise, we know that $\mathbf{A}, \delta \models_{OSE} L' \Leftarrow G'$, giving $\mathbf{A}, \delta \models_{OSE} L'$ and consequently $\mathbf{A}, \delta \models_{OSE} G \wedge L'$, which now implies $\mathbf{A}, \delta \models_{OSE} L$. Therefore, $\mathbf{A}, \delta \models_{OSE} L \Leftarrow G \wedge G'$.
- **Reflexivity** : Since all sorts are non-empty and all $(\mathbf{A}, \mathbf{X})$-variable assignments have to map variables x of sort s_m, $m \in [1..n]$, into elements of $s_m^{\mathbf{A}}$ and the interpretation of $=$ in any model $\mathbf{A}$ is the diagonal of $C^{\mathbf{A}} \times C^{\mathbf{A}}$, we get $\mathbf{A}, \delta \models_{OSE} x = x$ for all $(\mathbf{A}, \mathbf{X})$-variable assignments δ.
- **Paramodulation** : Since we use non-overloaded semantics (see [Smolka *et al.*, 1989; Waldmann, 1989a]), we are sure for all $(\mathbf{A}, \mathbf{X})$-variable assignments δ which satisfy $\mathbf{A}, \delta \models_{OSE} G'$, that $\delta(s)$ and $\delta(t)$ are identical in $\mathbf{A}$. Furthermore, if $\mathbf{A}, \delta \models_{OSE} G$, then also $\mathbf{A}, \delta \models_{OSE} L[s]$. Consequently, if δ satisfies $\mathbf{A}, \delta \models_{OSE} G \wedge G'$, we also know that $\mathbf{A}, \delta \models_{OSE} L[t]$, since for all models $\mathbf{B}$ and $(\mathbf{B}, \mathbf{X})$-variable assignments δ', we have $\mathbf{B}, \delta' \models_{OSE} L[s]$ iff $\mathbf{B}, \delta' \models_{OSE} L[t]$.

□

Let us close the section with some technical, in the following very useful lemmas. The first one intuitively means that the typability of the clauses in $\mathbf{P}$ implies that any term in the head literal of a provable clause is provably equal to a typable term in the context of the body of the clause.

Lemma 2.2.14 *If $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow G$, then for any subterm $u \in \mathbf{ET}(\mathbf{X})$ of L, there exists a $v \in \mathbf{T}(\mathbf{X})$, s.t. $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = v \Leftarrow G'$ for some goal G' containing only literals from G, written $G' \subseteq G$.*

Proof: The proof goes by induction on the length $n > 0$ of the derivation $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L \Leftarrow G$. We distinguish the different cases for the last step:

- **Reflexivity** : $x \in \mathbf{X} \subseteq \mathbf{T}(\mathbf{X})$ and the proof can be obtained by **Reflexivity** itself.
- **Axioms** : All axioms are $(\Sigma, \mathbf{X})$-clauses, i.e. all subterms are by definition in $\mathbf{T}(\mathbf{X})$ and the corresponding proofs can be obtained by **Reflexivity** and **Substitutivity**.

- **Substitutivity** : By induction hypothesis we know proofs $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = v \Leftarrow G'$ with $v \in \mathbf{T}(\mathbf{X})$ for all subterms u of L. Hence, $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} \sigma(u) = \sigma(v) \Leftarrow G'$ can be obtained by applying **Substitutivity** on this proof and $\sigma(v) \in \mathbf{T}(\mathbf{X})$ since $v \in \mathbf{T}(\mathbf{X})$ and $\sigma \in SUBST_\Sigma(\mathbf{X})$.
- **Cut** : L already appears in the premiss and since we know that $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = v \Leftarrow G''$ exists, s.t. $v \in \mathbf{T}(\mathbf{X})$ and $G'' \subseteq G \wedge L'$, for all u being subterms of L. If $L' \notin G''$, then we get the lemma by induction hypothesis. Otherwise, we can apply **Cut** to $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = v \Leftarrow G''$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L' \Leftarrow G'$, giving us $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = v \Leftarrow G''' \wedge G'$, where G''' is G'' without L'. Clearly, $G''' \subseteq G$, since $G'' \subseteq G \wedge L'$, and hence the lemma is shown.
- **Paramodulation** : We know the lemma for all subterms u of t and $L[s]$, respectively, by induction hypothesis. Remains the case where u contains the newly inserted t in $L[t]$. In this case, there is a $u' \equiv u'[s]$ with the property by induction hypothesis and $u \equiv u'[t]$. Hence, we can construct a proof of $u = v'$ by adding one more step of paramodulation to the one of $u' = v'$.

□

Definition 2.2.15 *The* sortal behavior *of an term $t \in \mathbf{T}(\mathbf{X})$ is the set:*

$$SB(t) = \{(\sigma, s) \mid \sigma(t) \in \mathbf{T}_s(\mathbf{X})\}$$

Let $s, t \in \mathbf{T}(\mathbf{X})$. An equation $s = t$ is sort preserving *if $SB(s) = SB(t)$. An $\mathcal{OSE}$-Σ-presentation $\mathbf{P}$ is called* sort preserving, *if all equations $s = t$ occurring in heads of clauses in $\mathbf{P}$ are sort preserving.*

Lemma 2.2.16 *Let $s, t, u[s]_\omega \in \mathbf{T}(\mathbf{X})$ and $s = t$ be a sort preserving equation. Then, $SB(u[s]_\omega) = SB(u[t]_\omega)$.*

Proof: By induction on the length of ω. If ω is the top occurrence, then the lemma follows from the sort preservation by $s = t$. Otherwise, let $u[s] = f(t_1, \ldots, t_k[s]_{\omega'}, \ldots, t_n)$, i.e. $\omega = k.\omega'$. By the induction hypothesis, we know that $SB(t_k[s]_{\omega'}) = SB(t_k[t]_{\omega'})$ and therefore $\sigma(u[s]) \in \mathbf{T}_s(\mathbf{X})$ iff $\sigma(u[t]) \in \mathbf{T}_s(\mathbf{X})$ - remember that all function declarations are flat and linear. Hence, $SB(u[s]_\omega) = SB(u[t]_\omega)$. □

Lemma 2.2.17 *Let $\mathbf{P}$ be a sort preserving presentation. If $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = v$, then $u = v$ is sort preserving.*

Proof: Actually, we prove that whenever $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} s' = t' \Leftarrow G$, then $s' = t'$ is sort preserving and $s', t' \in \mathbf{T}(\mathbf{X})$. The proof goes by induction on the derivation length of $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} s' = t' \Leftarrow G$. We distinguish the different cases for the last step:

- **Reflexivity** : Trivially sort preserving and in $\mathbf{T}(\mathbf{X})$.
- **Axioms** : $\mathbf{P}$ is sort preserving and contains only $(\Sigma, \mathbf{X})$-clauses.
- **Substitutivity** : By the definition of sort preservation and the well-known fact that instantiating terms in $\mathbf{T}(\mathbf{X})$ by substitutions in $SUBST_\Sigma(\mathbf{X})$ can only yield terms in $\mathbf{T}(\mathbf{X})$.
- **Cut** : L already appears in the premiss and the lemma follows therefore by the induction hypothesis.

- **Paramodulation** : We know that the length of the proofs $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} s = t \Leftarrow G'$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} L[s] \Leftarrow G$, respectively, are strictly smaller than the one of $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OS} L[t] \Leftarrow G \wedge G'$. Therefore, $s = t$ is sort preserving and $s, t, u[s] \in \mathbf{T}(\mathbf{X})$ for all subterms $u[s]$ in $L[s]$ by the induction hypothesis. Now, Lemma 2.2.16 gives the result for $L[t]$.

□

2.3 Initial Model

This section contains the (classical) construction of the free/ground deductive term interpretation by building the quotient, the completeness proof of **OSL** and the proof of initiality for the ground deductive term interpretation in $CMod(\mathbf{P})$.

Given a $\mathbf{S}$-sorted variable set $\mathbf{V}$, we define the $\mathbf{V}$-*deductive term interpretation* $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{V})$ to be T, s.t.:

- $C^T = \bigcup_{s \in \mathbf{S}} s^T$,
- $\forall t, [t]$ is a representative of $\{s \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} s = t\}$ if this set is non-empty, undefined otherwise.
- $\forall s \in \mathbf{S}, s^T = \bigcup_{s' \leq_{\mathbf{S}} s} s'^T \cup \{[x] \mid x \in \mathbf{V}_s\} \cup \{[f(t_1, \ldots, t_n)] \mid (f : s_1, \ldots, s_n \to s) \in ranks(f)$ and $\forall i \in [1..n], [t_i] \in {s_i}^T\}$, s.t. s^T is minimal w.r.t. $\subseteq$,
- $\forall f \in \mathbf{F}$, if $(f : s_1, \ldots, s_n \to s) \in ranks(f)$, then:
  - $Dom(f^T) = s_1^T \times \ldots \times s_n^T$ and
  - $f^T([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$, if $\forall i \in [1..n], [t_i] \in {s_i}^T$,

  and
- $\forall p \in \mathbf{R}$, if $(p : s_1, \ldots, s_n) \in ranks(p)$, then:
  - $Dom(p^T) = s_1^T \times \ldots \times s_n^T$ and
  - $p^T = \{([t_1], \ldots, [t_n]) \in (C^T)^n \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} p(t_1, \ldots, t_n)\}$.

Remark that since we started with $(\Sigma, \mathbf{X})$-clauses, all terms s in deduced atoms must by Lemma 2.2.14 have a typeable term in its equivalence class w.r.t. $=$, i.e. the union of all defined $[t]$, s.t. t is well-typed, covers all such terms. Let us state this more formally a last corollary:

Corollary 2.2.18 *For all subterms $t \in \mathbf{ET}$ in an atom A, s.t. $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} A$, there is a u with $[u] = [t]$ and $u \in \mathbf{T}(\mathbf{X})$.*

Proof: By Lemma 2.2.14, there is a $u \in \mathbf{T}(\mathbf{X})$ with $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{OSE} u = t$, i.e. $[u] = [t] \neq \emptyset$. □

This is important, since paramodulation may yield terms outside of $\mathbf{T}(\mathbf{X})$, but those terms must still be meaningful, provided the premiss of the clause where the problematic term occurs is fulfilled. See also the soundness proof of **Paramodulation** for details

Let the *(ground) deductive term interpretation* $\mathbf{T}_{\Sigma,\mathbf{P}}$ be $\mathbf{T}_{\Sigma,\mathbf{P}}(\emptyset)$. Let furthermore the *free deductive term interpretation* be $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$. Before we go on, let us prove the quasi-equivalence of variable assignments in $\mathbf{T}_\Sigma$ and $\mathbf{T}_{\Sigma,\mathbf{P}}$.

Lemma 2.2.19 *Let all sorts be non-empty. For all* $(\mathbf{T}_{\Sigma,\mathbf{P}},\mathbf{X})$- *or* $(\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\mathbf{X})$*-variable assignments* δ*, there exists a* $(\mathbf{T}_\Sigma,\mathbf{X})$*-variable assignment* δ' *with* $[\delta'(x)] = \delta(x)$*.*

Proof: We know that $\delta(x)$ is of the form $[t]$ with $(\Sigma,\mathbf{P},\mathbf{X}) \vdash_{OSE} t = t$, i.e. by Lemma 2.2.14, we get the existence of some $u \in \mathbf{T}(\mathbf{X})$ with $(\Sigma,\mathbf{P},\mathbf{X}) \vdash_{OSE} u = t$. Hence, take $\delta'(x) = u$ in the case $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$ or $\delta'(x) = \sigma(u)$ for some ground Σ-substitution σ, which must exist since all sorts are non-empty, in the case of $\mathbf{T}_{\Sigma,\mathbf{P}}$. □

Lemma 2.2.20 *Let* Σ *be a signature with non-empty sorts and* $\mathbf{P}$ *be a* Σ*-presentation. Then* $\mathbf{T}_{\Sigma,\mathbf{P}}$ *and* $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$ *are models for* $(\Sigma,\mathbf{P})$*.*

Proof: Clearly, $\mathbf{T}_{\Sigma,\mathbf{P}}$ and $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$ are Σ-interpretations. We first prove that all clauses in $\mathbf{P}$ are valid in $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$. Suppose $L \Leftarrow G$ is in $\mathbf{P}$ and δ is a $(\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\mathbf{X})$-variable assignment, s.t. $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\delta \models_{OSE} G$. By Lemma 2.2.19, there is a $(\mathbf{T}_\Sigma(\mathbf{X}))$-variable assignment δ', s.t. $[\delta'(x)] = \delta(x)$ for all $x \in \mathbf{X}$, i.e. $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}) \models_{OSE} \delta'(G)$ and therefore $(\Sigma,\mathbf{P},\mathbf{X}) \vdash_{OSE} \delta'(G)$ by Lemma 2.2.8 (use δ' as σ and δ be defined by $\delta(x) = [x]$ for all $x \in \mathbf{X}$) and the definition of $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$.

Then we can use the **Substitutivity** rule followed immediately by $|G|$-times **Cut**, in order to get $(\Sigma,\mathbf{P},\mathbf{X}) \vdash_{OSE} \delta'(L)$. Hence, $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\delta \models_{OSE} L$ by the definition of $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$.

The case of $\mathbf{T}_{\Sigma,\mathbf{P}}$ is similar, except that we can take any δ for an application of Lemma 2.2.8, since $\delta'(G)$ is ground, and we need to apply once more **Substitutivity** after **Cut**, in order to bind eventually free variables x in $\delta'(L)$ to the ground terms $\delta'(x)$. Finally, we have to check that $=$ is interpreted as the diagonal on the cartesian product of the carrier. But this is clear by **Reflexivity**, **Substitutivity** and **Paramodulation**, which imply that $=$ is interpreted as congruence[5] on all t in $\mathbf{T}(\mathbf{X})$. □

We get almost immediately the completeness of our calculus.

Theorem 2.2.21 *Let all sorts be non-empty,* $\mathbf{P}$ *be a* Σ*-presentation, and* L *a* $(\Sigma,\mathbf{X})$*-atom with* $(\Sigma,\mathbf{P},\mathbf{X}) \models_{OSE} L$*. Then* $(\Sigma,\mathbf{P},\mathbf{X}) \vdash_{OSE} L$*.*

Proof: Since $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$ is a model for $(\Sigma,\mathbf{P})$ (see Lemma 2.2.20), we have $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\mathbf{X} \models_{OSE} L$. Therefore, for all $(\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\mathbf{X})$-variable assignments δ, $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X}),\delta \models_{OSE} L$, especially for $\delta(x) = [x]$, i.e. $(\Sigma,\mathbf{P},\mathbf{X}) \vdash_{OSE} L$ by definition of $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$. □

Using the ground deductive term interpretation, we get an initial model.

Theorem 2.2.22 *Let all sorts be non-empty and* $\mathbf{P}$ *be a* Σ*-presentation. Then* $\mathbf{T}_{\Sigma,\mathbf{P}}$ *is initial in* $CMod(\mathbf{P})$*.*

[5] Remark that symmetry is a consequence of reflexivity and paramodulation.

Proof: Let $\mathbf{A}$ be an arbitrary model for $(\Sigma, \mathbf{P})$ and h be defined by $\forall f \in \mathbf{F}, h([f(t_1, \ldots, t_n)]) = f^{\mathbf{A}}(h([t_1]), \ldots, h([t_n]))$ if $f(t_1, \ldots, t_n) \in \mathbf{T}$. Let T be the shorthand for $\mathbf{T}_{\Sigma,\mathbf{P}}$ in this proof.

Clearly, well-typed terms are mapped to objects in $C^{\mathbf{A}}$, i.e. $h(C^T) \subseteq C^{\mathbf{A}}$. Furthermore, $\forall f \in \mathbf{F}, h(f^T([t_1], \ldots, [t_n])) = f^{\mathbf{A}}(h(t_1^T), \ldots, h(t_n^T))$ if $(t_1^T, \ldots, t_n^T) \in \mathcal{D}om(f^T)$ by definition of h and $\forall p \in \mathbf{R}, h(p^T) \subseteq p^{\mathbf{A}}$, since the deduction rules are sound and we can find a $(\mathbf{T}_\Sigma, \mathbf{X})$-variable assignment δ' whenever we can find a $(T, \mathbf{X})$ assignment δ, s.t. $\delta(x) = [\delta'(x)]$ for all $x \in \mathbf{X}$ and $T, \delta \models_{OSE} \phi$ whenever $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash \delta'(\phi)$ and vice versa.

We show that h fulfils the last remaining condition for Σ-homomorphisms, i.e. $\forall [t] \in s^T, h([t]) \in s^{\mathbf{A}}$, by structural induction on the term t. Let $t = f(t_1, \ldots, t_n)$ and $[t] \in s^T$. Then there are $s_1, \ldots s_n, s \in \mathbf{S}$, s.t. $(f : s_1, \ldots, s_n \to s) \in ranks(f)$ and $\bigwedge_{i \in [1..n]} [t_i] \in s_i{}^T$. By the induction hypothesis, we get $h([t_i]) \in s_i^{\mathbf{A}}$ and therefore $f^{\mathbf{A}}(h([t_1]), \ldots, h([t_n])) \in s^{\mathbf{A}}$, since $f^{\mathbf{A}}$ must map $(s_1^{\mathbf{A}}, \ldots, s_n^{\mathbf{A}})$ into $s^{\mathbf{A}}$.

Suppose there is another Σ-homomorphism $g : T \to \mathbf{A}$. Then, $h = g$ can be shown by structural induction on the argument $[t] \in C^T$: Let $t = f(t_1, \ldots, t_n)$. Hence:

$$\begin{aligned} g([f(t_1, \ldots, t_n)]) &= f^{\mathbf{A}}(g([t_1]), \ldots, g([t_n])) \\ &= f^{\mathbf{A}}(h([t_1]), \ldots, h([t_n])) \\ &= h(f([t_1], \ldots, [t_n])), \end{aligned}$$

since g and h are Σ-homomorphisms and the induction hypothesis, which provides $g([t_i]) = h([t_i])$ for all $i \in [1..n]$.

□

3 R^n-Logics

Introduction

The idea underlying R^n-logic is to use $\mathcal{OSE}$-logic as a set theory with predefined membership predicate $\in$, such that terms may represent individuals or sets of nesting depth up to n. The sets may be used for different purposes: they can represent sorts as in G-algebras, or they can be used to define function graphs, which may be passed to other functions as higher-order arguments. In the former case, parameterised sorts correspond with set valued functions, which may take individuals or sets as arguments. So it is easy to write down a specification of bounded stacks:

Example 2.3.1 *Assume that* $\mathtt{s}^n(\mathtt{0})$ *represents the natural number* n *(from the set* `nat`*),* `indiv` *is the set of all individuals and* `set` *is the set of all sets of individuals. Then the following specification defines the parameterised set* `bounded_stack` *of bounded stacks, taking two arguments: the set of elements in the stack and the bound on the depth. This is done using an additional parameterised set* `bstack` *whose first two arguments are identical with the one of* `bounded_stack` *and whose third argument is the actual depth, i.e. all elements of* `bstack(E,b,n)` *are bounded stacks of elements in* `E` *and depth* `n` *which is lower or equal to* `b`.

```
spec BSTACK  is
             op bounded_stack : set indiv -> set
             op bstack : set indiv indiv -> set
             op create : set indiv -> indiv
             op push : indiv indiv -> indiv
             op pop : indiv -> indiv
             op top : indiv -> indiv
             var b,n : indiv
             var E : set
             var x,y : indiv
             ax x ∈ bounded_stack(E,b)
                  if x ∈ bstack(E,b,n)
             ax create(E,b) ∈ bstack(E,b,0)
             ax push(x, y) ∈ bstack(E,b,s(n))
                  if x ∈ E and b,n ∈ nat
                  and y ∈ bstack(E,b,n) and n < b
             ax pop(push(x, y)) = y
                  if x ∈ E and b,n ∈ nat
                  and y ∈ bstack(E,b,n) and n < b
             ax top(push(x, y)) = x
                  if x ∈ E and b,n ∈ nat
                  and y ∈ bstack(E,b,n) and n < b
endspec
```

Remark that this pseudo-language translates `if` and `and` into implication and conjunction. The design of a truly practical specification language with more syntactic sugar and based on R^n-logics is being investigated.

The use of higher-order features depends on function graphs, which are specified by:

$$\langle \mathtt{x}, \mathtt{f(x)} \rangle \in \mathtt{f},$$

where $\langle .,. \rangle$ is a tuple constructor. The resulting set of pairs may then be passed to functions like `E` was given to `create` in the last example. Whenever the function should be evaluated for an argument `x` then, it is sufficient to place a condition $\langle$ `x,y` $\rangle \in$ `f` in the body of the clause where the result is needed.

We start with the definition of R^n-signatures and models, which are a non-trivial subset of $\mathcal{OSE}$-models. Then, we give a set of deduction rules for R^n-logics, which is comfortably small in our opinion. It consists in four additional rules added to any sound and complete deduction system for $\mathcal{OSE}$-logics. Before we discuss R^n-logics and compare them to other related frameworks (ETL and unified algebras), the initial models are proved to exist for any set of R^n-formulas.

3.1 Models and Axioms

We start with the definition of R^n-signatures and models, followed by an axiomatisation based on $\mathcal{OSE}$-logics. One might use this axiomatisation then for proving theorems in R^n-logics using order-sorted resolution and paramodulation [Loveland, 1978; Walther, 1983]. R^n-Σ-signatures are a subclass of Σ-signatures, containing predefined symbols and having a fixed sort structure. The sort structure is also illustrated in Figure 2.4.

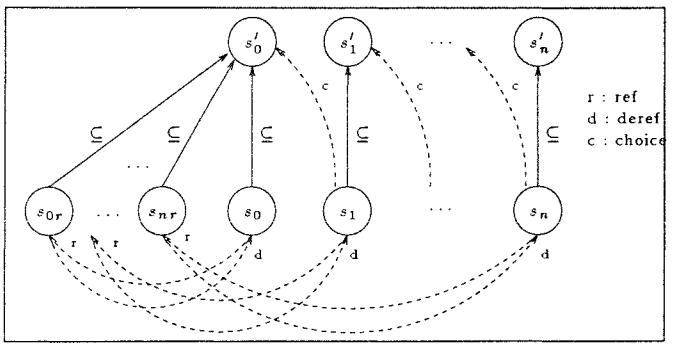


Figure 2.4: The Sort Structure of R^n-signatures

The underlying idea for the sort structure is to start with Russell's theory of types up to order n, which is basically many-sorted. Assume $\{s_0, \ldots, s_n\}$ is the set of sorts. Then, s_0 stand for the sort of individuals represented by terms and if $i \in [1..n]$, then s_i is the sort for terms representing sets of sets of etc. (i times) of individuals. Therefore, sets in s_i are called sets of *order* i.

Now, we add supersorts $s'_0, \ldots, s'_n$ for $s_0 \ldots, s_n$, respectively, such that s'_0 stands for all individuals not necessarily represented by a term, s'_i, $i \in [1..n]$, for all sets of sets of etc. (i times) of individuals, also not necessarily represented by a term. This allows us for example to reason about real numbers as individuals although it is impossible to represent all of them as terms.

As a last step, we add sorts $s_{0r}, \ldots, s_{nr}$ for references to objects (i.e. individuals, sets of individuals etc.) represented by terms in one of the sorts $s_0, \ldots, s_n$. The intuition for references is that they are names for the objects they refer to. Hence, references are in bijection with these objects.

Definition 2.3.2 *Let n be a natural number ≥ 0. An R^n-*signature Σ *is an $\mathcal{OSE}$-signature $(\mathbf{S}, \leq_\mathbf{S}, \mathbf{F}, \mathbf{R})$, s.t. $\mathbf{S} = \{s_0, \ldots, s_n\} \cup \{s_{0r}, \ldots, s_{nr}\} \cup \{s'_0, \ldots, s'_n\}$, and $\leq_\mathbf{S}$ is minimal, s.t. for all $i \in [0..n]$, $s_i \leq_\mathbf{S} s'_i$ and $s_{ir} \leq_\mathbf{S} s'_0$, and if $n > 0$, $\mathbf{F}$ contains the following functions:*

- *choose with $ranks(choose) = \{(choose : s_i \to s'_{i-1}) \mid i \in [1..n]\}$,*
- *ref with $ranks(ref) = \{(ref : s_i \to s_{ir}) \mid i \in [0..n]\}$,*
- *deref with $ranks(deref) = \{(deref : s_{ir} \to s_i) \mid i \in [0..n]\}$,*

All other functions f have a fixed number of arguments and ranks of the form $(f : s''_1, \ldots, s''_q \to s'') \in ranks(f)$, where $s'', s''_i \in \{s_0, \ldots, s_n, s_{0r}, \ldots, s_{nr}\}$ for all $i \in [1..q]$. If $n > 0$, $\mathbf{R}$ contains the relation $\in$ with $ranks(\in) = \{(\in : s'_{i-1}\ s_i)\}$ for $i \in [1..n]$. Furthermore, Σ is regular.

*An R^n-(Σ-)*presentation $\mathbf{P}$ *is a Σ-presentation, s.t. all variables of sorts $s'_0, \ldots, s'_n$ in $\mathbf{P}$ occur on the left of a membership relation '$\in$', choose does only appear as top operator of a left argument of $\in$ and all equations occurring in the head of some clause in $\mathbf{P}$ are sort preserving.*

First of all, remark the nature of the terms in the different sorts: The lower sorts s_m should contain only terms without choice function symbol in it, the ones in s'_m those of s_{m+1} with one *choose* symbol on the top. In addition, s_{mr} contains only terms of s_m with *ref* additionally

on its top, whenever $m \in [0..n]$. Remark also that regularity implies that for all $i, j \in [1..n]$, $\mathbf{T}_{s_0} \cap \mathbf{T}_{s_j} = \emptyset$ and $i \neq j$ implies $\mathbf{T}_{s_i} \cap \mathbf{T}_{s_j} = \emptyset$. We give an example of such R^n-signatures.

Example 2.3.3 *Consider the following cases for n:*

- $n = 0$ *: Let* $\mathcal{F}$ *and* $\mathcal{R}$ *be arbitrary sets of unsorted function symbols* f *and relation symbols* p*, respectively, with associated arity* $ar(f)$ *and* $ar(p)$*. Then, let* $\Sigma = (\mathbf{S}, \leq_\mathbf{S}, \mathbf{F}, \mathbf{R})$ *with* $\mathbf{S} = \{s_0, s_0', s_{0r}\}$, $\leq_\mathbf{S} = \{(s_{0r}, s_0'), (s_0, s_0')\}$, $\mathbf{F} = \mathcal{F}$ *and* $\mathbf{R} = \mathcal{R}$ *with* $ranks(f) = \{(f : s_0, \ldots, s_0 \rightarrow s_0)\}$ *for all* $f \in \mathbf{F}$, $ranks(p) = \{(p : s_0, \ldots, s_0)\}$, *s.t.* $ar(f)$ *and* $ar(p)$ *remain unchanged. Clearly, any such* Σ *is an* R^0*-signature, if* $\mathbf{R}$ *contains* $=$ *and we add the ranks* $(= : s_0, s_0)$, $(= : s_0', s_0')$ *and* $(= : s_{0r}, s_{0r})$.

- $n = 1$ *: Let* $\Sigma = (\mathbf{S}, \leq_\mathbf{S}, \mathbf{F}, \mathbf{R})$ *with* $\mathbf{S} = \{s_0, s_0', s_{0r}, s_1, s_1', s_{1r}\}$, $\leq_\mathbf{S} = \{(s_0, s_0'), (s_{0r}, s_0), (s_{1r}, s_0)\}$, $\mathbf{F} = \{succ, nat, 0, choose, ref, deref\}$ *with rank* $ranks(s) = \{(f : s_0 \rightarrow s_0)\}$, $ranks(nat) = \{(nat : \rightarrow s_1)\}$ *and* $ranks(0) = \{(0 : \rightarrow s_0)\}$, *plus the ones of the predefined function symbols. Finally, let* $\mathbf{R} = \{=, \in\}$ *with* $ranks(\in) = \{(\in : s_0', s_1)\}$ *and* $ranks(=) = \{(= : s_0, s_0), (= : s_1, s_1), (= : s_{0r}, s_{0r}), (= : s_{1r}, s_{1r}), (= : s_0', s_0'), (= : s_1', s_1')\}$. *Then,* Σ *is an* R^1*-signature. However, if e.g.* $ranks(succ) = \{(succ : s_0 \rightarrow s_0), (succ : s_0 \rightarrow s_1)\}$, *then* Σ *would not be an* R^1*-signature, since it is no more regular (cf.* $sorts(s(0))$*). Furthermore, if* $x^0 \in \mathbf{X}_{s_0}$, *then* $\mathbf{P} = \{0 \in nat \Leftarrow, succ(x^0) \in nat \Leftarrow x^0 \in nat\}$ *is an* R^1*-*Σ*-presentation.*

The signature restrictions are relatively strong, because of the clear separation of sets of different order. In fact, the sort preservation of equations results in static typing like in many-sorted logics. The choice functions are introduced in order to get a characteristic element for each set represented by a term. They also play a role in constructing the initial model.

Some difficulty arises with the treatment of these choice functions, which classically have a non-deterministic behavior, but which we like to keep deterministic for technical reasons ($\mathcal{OSE}$-logics are not prepared to handle non-deterministic functions). However, the restrictions on R^n-presentations make it impossible to define the result of a choice. Hence, the non-determinism has moved to the model level. Remark that these choice functions correspond with Hilbert's ϵ operator also used in HOL (cf. [Gordon, 1993]).

References are particularly useful when we try to see functions as sets, since they allow to use arguments of arbitrary type for functions over individuals. It may be seen as somewhat more technical solution to the domain equation $D = [D \rightarrow D]$. From a pure set theoretic point of view, there may be objections to the use of references as elements different from sets. However, similar to individuals, we regard them as *a priori* given objects (sometimes called *ur-elements*), just as terms are. To give an appropriate intuition: just think of references as purely syntactic, finite objects (just like terms), although the set they represent might be infinite.

Before we start with the definition of R^n-models, we have to rule some technicalities concerning choices. It is in the following very useful to treat choices different from other objects, like individuals or sets. The essential use of choices in our framework is the possibility to express that there may be other objects than those represented by terms in a particular model. Consequently, we may represent a choice in a set of sets of individuals by a single term. This is a difference to Russell's original theory of types, where a choice in a set of sets of individuals would have to be a set of terms representing individuals.

Hence, given a set of individuals, we define sets together with choices as follows: A choice of order 1 is a term representing an individual. A set of order 1 is a set of choices of order 1

and individuals. A choice of order $k \in [2..n]$ is a term representing a set of order $k-1$. A set of order $k \in [2..n]$ is a set of sets and choices of order $k-1$.

Definition 2.3.4 *Let Σ be an R^n-signature, $\mathbf{X}$ a set of $\mathbf{S}$-sorted variables and $\mathbf{P}$ an R^n-presentation.*

*An R^n-*model $\mathbf{A}$ *of $\mathbf{P}$ is a Σ-model of $\mathbf{P}$, where:*

- *$C^{\mathbf{A}}$ contains a non-empty set of individuals $s_0'^{\mathbf{A}}$ in the sense of [Whitehead et Russell, 1925], s.t.*
- *s_0 is interpreted as the set of individuals represented by a term,*
- *for $i \in [1..n]$, $s_i^{\mathbf{A}}$ contains only non-empty sets of order i represented by a term,*
- *for $i \in [0..n]$, $s_{ir}^{\mathbf{A}}$ is a set of symbolic names contained in $s_0'^{\mathbf{A}}$,*
- *for $i \in [1..n]$, $s_i'^{\mathbf{A}}$ is the set of all sets of order i, plus all choices of order $i+1$ whenever $i < n$,*
- *$choose^{\mathbf{A}}$ gives a unique element from any set in $C^{\mathbf{A}}$ given as argument (choice function) and*
- *ref is a bijective function over $s_i^{\mathbf{A}}$, $i \in [0..n]$, taking a set of order i and returning its unique name in $s_{ir}^{\mathbf{A}}$,*
- *$deref$ is its inverse function on $s_{ir}^{\mathbf{A}}$, and*
- *$\in$ is interpreted as the membership relation on all sets in $C^{\mathbf{A}}$.*

We adapt our notation from $\mathcal{OSE}$-logics, s.t. $\mathbf{P} \models_{R^n} \phi$ means that ϕ is true in all R^n-models of $\mathbf{P}$. Let R^n-$Cat(\Sigma)$ be the category of R^n-logics with Σ-homomorphisms.

Notation: Let for $\mathbf{V} \subseteq \mathbf{X}$ and $k \in [0..n]$:

- $T^k(\mathbf{V})$, $T^{kr}(\mathbf{V})$ and $T'^k(\mathbf{V})$ be the shorthand for $\mathbf{T}_{s_k}(\mathbf{V})$, $\mathbf{T}_{s_{kr}}(\mathbf{V})$ and $\mathbf{T}_{s'_k}(\mathbf{V})$, respectively,
- T^k, T^{kr} and T'^k be the shorthand for $\mathbf{T}_{s_k}$, $\mathbf{T}_{s_{kr}}$ and $\mathbf{T}_{s'_k}$, respectively,
- u^k, v^k, t^k etc. stand for terms in $T^k(\mathbf{X})$,
- u^{kr}, v^{kr}, t^{kr} etc. stand for terms in $T^{kr}(\mathbf{X})$ and
- u'^k, v'^k, t'^k etc. stand for terms in $T'^k(\mathbf{X})$,
- x^k, y^k, z^k, x'^k, y'^k, z'^k and x^{kr}, y^{kr}, z^{kr} stand for variables in $\mathbf{X}_{s_k}$, $\mathbf{X}_{s'_k}$ and $\mathbf{X}_{s_{kr}}$, respectively.

Figure 2.5 now shows the set of deduction rules to be added to those of $\mathcal{OSE}$-logics, if $n > 0$, in order to perform deduction for R^n-logics.

Note first of all that these are deduction rule schemes, since there is one rule for each $m \in [1..n]$ and $k \in [0..n]$. Remark that **Choice** is nothing else than the axiom of choice and

Choice $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} choose(x^m) \in x^m}$ if $m \in [1..n]$

Ref $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} ref(deref(x^{kr})) = x^{kr}}$ if $k \in [0..n]$

Deref $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} deref(ref(x^k)) = x^k}$ if $k \in [0..n]$

Ext $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} choose(u^m) \in t^m,\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} choose(t^m) \in u^m}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} u^m = t^m}$ if $m \in [1..n]$

Figure 2.5: R^n-logic deduction rules to be added to **OSL**

Ext is a Horn Clause version of the axiom of extensionality, which can also be given as hereditary Harrop formula:

$$\textbf{ClassicExt} \quad (y^m = z^m) \Leftarrow (\forall x'^{m-1}, x'^{m-1} \in y^m \Leftarrow x'^{m-1} \in z^m) \wedge (\forall x'^{m-1}, x'^{m-1} \in z^m \Leftarrow x'^{m-1} \in y^m)$$

The intuition behind **Ext** is the freeness of *choose*, the choice function, which we keep deterministic but without equational axioms by requiring that it does not occur in the head of a clause in an R^n-presentation. Let **RNL** be a shorthand for **OSL** plus **Choice**, **Ref**, **Deref** and **Ext**.

Remark that **Ext** cannot be formulated as a Horn clause of the following form:

$$y^m = z^m \Leftarrow choose(y^m) \in z^m \wedge choose(z^m) \in y^m.$$

This is simply not a valid clause, i.e. satisfied by all R^n-logic models, since it does not hold for any fixed interpretation of *choose*, as the following example shows:

Example 2.3.5 *Let $n = 1$ and $\mathbf{P} = \{a \in A, b \in B, a \in C, b \in C\}$. Now, let $\mathbf{A}$ be the model interpreting A as $\{a\}$. B as $\{b\}$ and C as $\{a, b\}$. $\mathbf{A}$ trivially satisfies* **ClassicExt**, *since the premiss never gets true. However, there is no way to give an interpretation to choose, s.t. $choose^{\mathbf{A}}(C^{\mathbf{A}})$ is different to $choose^{\mathbf{A}}(A^{\mathbf{A}})$ and $choose^{\mathbf{A}}(B^{\mathbf{A}})$. But neither $A^{\mathbf{A}} = C^{\mathbf{A}}$ nor $B^{\mathbf{A}} = C^{\mathbf{A}}$, i.e. the Horn clause version of* **Ext** *cannot be valid in this R^n-model of* **P**.

However, we may quantify over all possible interpretations, since *choose* is free, and therefore **Ext** is sound for deduction. The soundness of the other deduction rules being obvious by the definition of R^n-models, we can state immediately:

Theorem 2.3.6 *The axioms of Figure 2.5 are sound w.r.t. deduction in R^n-models.*

3.2 Initial Model Construction

Let $\vdash_{RNL}$ be the deduction relation generated by the rules $\mathcal{OSL}$ and those of figure 2.5. We start with the definition of our candidate for an initial model. The idea behind the consruction is that individuals and choices are interpreted by a representative of their equivalence class, whereas all other terms are interpreted as the set of representatives of equivalence classes of their provable elements. Sorts function and relation symbols are then interpreted as usual.

Definition 2.3.7 *Let* $\mathbf{P}$ *be an* R^n*-presentation and* $\mathbf{V} \subseteq \mathbf{X}$. *The interpretation* $\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})$ *is I defined as follows:*

- $\forall k \in [0..n], t'^k \in T^0(\mathbf{V}) \cup (T'^k(\mathbf{V}) \setminus T^k(\mathbf{V}))$,
  $[t'^k]$ is a representative of $\{u'^k \in T'^k(\mathbf{V}) \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} u'^k = t'^k\}$
  (Remark that this includes the cases $t^0 \in T^0$ *and* $t^{kr} \in T^{kr}(\mathbf{V})$ *for* $k \in [0..n]$.*)*

- $\forall m \in [1..n], \forall t^m \in T^m(\mathbf{V})$:
  $[t^m] = \{[u'^m] \mid u'^m \in T'^m(\mathbf{V})$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} u'^m \in t^m\}$

- $\forall s \in \mathbf{S}$, $s^I = \bigcup_{s'' \leq_s} \mathbf{V}_{s''} \cup \{[f(t_1, \ldots, t_q)] \mid (f : s''_1, \ldots, s''_q \to s'') \in ranks(f)$ and $\forall i \in [1..q], [t_i] \in s''^I_i$ and $s'' \leq_{\mathbf{S}} s\}$,
  s.t. s^I *is minimal w.r.t.* $\subseteq$,

- $C^I = \bigcup_{s \in \mathbf{S}} s^I$,

- $\forall m \in [0..n], \forall f \in \mathbf{F}$, *if* $(f : s''_1, \ldots, s''_q \to s'') \in ranks(f)$, *then:*
  - $Dom(f^I) = s''^I_1 \times \ldots \times s''^I_q$ *and*
  - $f^I([t_1], \ldots, [t_q]) = [f(t_1, \ldots, t_q)]$, *if* $\bigwedge_{i \in [1..q]} [t_i] \in s''^I_i$,

- $\forall p \in \mathbf{R}$:
  - $Dom(p^I) = (C^I)^q$ *and*
  - $p^I = \{([t_1], \ldots, [t_q]) \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} p(t_1, \ldots, t_q)$ and $\forall i \in [1..n], [t_i] \in C^I\}$.

Let from now on $\mathbf{P}$ be a fixed R^n-presentation. Instead of $\mathbf{I}_{R,\mathbf{P}}(\emptyset)$ and $\mathbf{I}_{R,\mathbf{P}}(\mathbf{X})$, we will write $\mathbf{I}$ and $\mathbf{I}_{R,X}$ in what follows. We start with some more technical results about the latter construction. The first of the following lemmas (2.3.8) shows that the result of choice functions applied to some set can only be proven equal to the result of applying the choice function to an equivalent set and that only sets of the same order (or individuals) can be proved equal.

Lemma 2.3.8 *Let* $\mathbf{P}$ *be an* R^n*-presentation.*

$$\text{If } (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} u = v \text{ then } (u \in \mathbf{T}_s(\mathbf{X}) \text{ iff } v \in \mathbf{T}_s(\mathbf{X}))$$

Proof: Remark that all equations in heads of clauses of $\mathbf{P}$ have to be sort-preserving by definition. The deduction rules in **OSL** preserve this property and so do those added for R^n-logics: The only rule introducing a new equation is **Ext**. But this equation is of the form $y^m = z^m$, where $y^m, z^m \in T^m(\mathbf{X})$ and $m \in [1..n]$. This is trivially a sort-preserving equation, since none of the sorts $s_1, \ldots, s_n$ has a subsort. □

The next lemma guarantees the soundness and completeness of $\in^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}$ and $=^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}$.

Lemma 2.3.9 *Let* $\mathbf{V} \subseteq \mathbf{X}$ *and consider* $[.]$ *as defined for* $\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})$.
For all $k \in [0..n], v'^k, t'^k \in T'^k(\mathbf{V}) \setminus T^k(\mathbf{V})$:

$$(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v'^k = t'^k \text{ iff } [v'^k] = [t'^k] \text{ and } [t'^k] \in C^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}. \quad (1)$$

For all $k \in [0..n], v^k, t^k \in T^k(\mathbf{V})$*:*

$$(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v^k = t^k \text{ iff } [v^k] = [t^k] \text{ and } [t^k] \in C^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}. \quad (2)$$

And for all $k \in [1..n], v'^{k-1} \in T'^{k-1}(\mathbf{V}), t^k \in T^k(\mathbf{V})$*:*

$$(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v'^{k-1} \in t^k \text{ iff } [v'^{k-1}] \in [t^k] \text{ and } [t^k] \in C^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}. \quad (3)$$

Proof: The left-to-right direction is trivial by definition and the reflexivity/paramodulation rules of $\mathcal{OSE}$-logics. From right-to-left, remark first of all that if $[v'^k] = [t'^k]$, then by definition $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{\mathcal{OS}} v'^k = t'^k$, i.e. (1) holds for $k \in [0..n]$.

There remain (2) and (3), which are proved by simultaneous induction on k. The base case is (2) for $k = 0$. Again the left-to-right case is trivial and the right-to-left direction is a consequence of the definition of equivalence classes.

Now, assume $k > 0$. We start to show (3) for k assuming (2) for $k - 1$.

If $v'^{k-1} \in T'^{k-1}(\mathbf{V}) \setminus T^{k-1}(\mathbf{V})$ and $[v'^{k-1}] \in [t^k]$, then by definition of $[t^k]$, there is a u'^{k-1} with $[u'^{k-1}] = [v'^{k-1}]$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} u'^{k-1} \in t^k$. Now, $[u'^{k-1}] = [v'^{k-1}]$ implies by (1) that $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} u'^{k-1} = v'^{k-1}$. Hence, **Paramodulation** gives us $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v'^{k-1} \in t^k$.

If $v^{k-1} \in T^{k-1}(\mathbf{V})$ and $[v^{k-1}] \in [t^k]$, then the same argument holds if we use the induction hypothesis for (2) with $k - 1$ instead of (1). Hence, (3) holds for k.

Finally, we show (2) for k assuming (3) for $k - 1$. The left-to-right case is still trivial by **Paramodulation**. For proving the right-to-left direction let $v^k, t^k \in T^k(\mathbf{V})$ be such that $[v^k] = [t^k]$. By the axiom of choice and (3) for $k - 1$ we know that $[choose(t^k)] \in [t^k]$ and $[choose(v^k)] \in [v^k]$. Since $[v^k] = [t^k]$, we get $[choose(v^k)] \in [t^k]$ and $[choose(t^k)] \in [v^k]$. Hence, (3) for $k - 1$ also gives $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} choose(v^k) \in t^k, choose(t^k) \in v^k$. By the axiom of extensionality we get $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v^k = t^k$, i.e. (2) holds for k. □

The last one of the three lemmas gives us the existence of a representative ground term t' of sort s for each object $[t] \in s^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}$, s.t. $[t'] = [t]$. This is useful in the initiality proof. Note that we use $\equiv$ for syntactic term and formula identity, in order to distinguish it from the standard identity sign $=$, already used as relation in $\mathcal{OSE}$-logics.

Lemma 2.3.10 *Let* $\mathbf{V} \subseteq \mathbf{X}$*, t be an extended term (i.e. a not necessarily well-sorted term respecting function ranks) and* $[t] \in s^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}$ *for some* $s \in \mathbf{S}$*. Then,* $t \in \mathbf{T}_s(\mathbf{V})$*.*

Proof: Let us start this proof with a reformulation of the definition of the sort interpretations, s.t. a traditional induction can be done:
Let $\mathfrak{S}_i$ for $i \geq 0$ stand for a $|\mathbf{S}|$-tuple indexed by sort names in $\mathbf{S}$ and π_s is the projection function giving the (unique) member of the tuple corresponding with $s \in \mathbf{S}$. Now, let $\mathfrak{S}_0$ be s.t. $\pi_s(\mathfrak{S}_0) = \mathbf{V}_{\leq s} = \cup_{s' \leq_{\mathbf{S}} s} \mathbf{V}_{s'}$ and $\mathfrak{S}_{i+1} = U(\mathfrak{S}_i)$, where $U = \langle U_s \rangle_{s \in \mathbf{S}}$ defined as follows:

$$U_s(\mathfrak{S}) = \pi_s(\mathfrak{S}) \cup \{[f(t_1, \ldots, t_q)] \mid \begin{array}{l}(f : s''_1, \ldots, s''_q \to s'') \in ranks(f) \text{ and} \\ \forall i \in [1..q], [t_i] \in \pi_{s''_i}(\mathfrak{S}) \text{ and} \\ s'' \leq_{\mathbf{S}} s\}.\end{array}$$

Hence, by construction, $\pi_s(\mathfrak{S}_\infty) = s^{\mathbf{I}_R}$.

Let for all $s \in \mathbf{S}$, s'''_k stand for $\phi_s(\mathfrak{S}_k)$. We prove by induction on k, that for all $[t] \in s'''_k$, $t \in \mathbf{T}_s$. If $k = 0$, then this is trivial since $s'''_k = \mathbf{V}_{\leq s}$. Otherwise, consider $s'''_k = U_s(\mathfrak{S}_{k-1})$. Either $[t] \in \pi_s(\mathfrak{S}_{k-1})$, i.e. the lemma follows by the induction hypothesis, or $t = f(t_1, \ldots, t_q)$ and there are $t'_i \in \mathbf{T}_{s''_i}$, s.t. $[t_i] = [t'_i]$, for $i \in [1..q]$. In the latter case, the induction hypothesis guarantees us that $t_i \in \mathbf{T}_{s''_i}$. Therefore, $t \in \mathbf{T}_s$ by definition of $\mathbf{T}_s$. □

Theorem 2.3.11 *Let* $\mathbf{V} \subseteq \mathbf{X}$. *If* $\mathbf{T}_{s_m} \neq \emptyset$ *for all* $m \in [0..n]$, *then* $\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})$ *is an* R^n*-model of* $\mathbf{P}$.

Proof: Let I stand for $\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})$ in the following. The following statements are easy to verify: All terms $t^0 \in T^0(\mathbf{V})$ are interpreted as terms in s_0^I, the set of individuals represented by a term. All terms $t^m \in T^m(\mathbf{V})$, for $m \in [1..n]$, are interpreted as sets of elements of s'^I_{m-1}, i.e. choices of order m and sets of order $m-1$ if $m > 1$ or individuals if $m = 0$. All $t^{mr} \in T^{mr}(\mathbf{V})$ for $m \in [0..n]$ and all $t'^0 \in T'^0(\mathbf{V}) \setminus (T^0(\mathbf{V}) \cup_{m \in [0..n]} T^{mr}(\mathbf{V}))$ are terms, i.e. finite, symbolic names. Finally, those in $T'^m(\mathbf{V}) \setminus T^m(\mathbf{V})$ for $m \in [1..n-1]$ are interpreted as terms representing choices of order $m+1$.

The non-emptiness of all $\mathbf{T}_{s_m}$ guarantees that $\mathbf{T}_s(\mathbf{V})$ is non-empty for all $s \in \mathbf{S}$ by **Choice**, *ranks*(*ref*) and the subsort relation. **Choice** also guarantees that $(t^m)^I$ is non-empty for all $t^m \in T^m(\mathbf{V})$, $m \in [1..n]$. The subsort relation is respected by definition.

By definition of s^I for $s \in \mathbf{S}$, $s'^I \subseteq s^I$, if $s' \leq_{\mathbf{S}} s$. Furthermore, $ref(t^m)^I = [ref(t^m)]$ is a unique term giving a name to any $[t^m] \in s^I_m$, $t^m \in T^m(\mathbf{V})$, i.e. ref^I is total on s^I_m by Lemma 2.3.10. Analogously, $deref^I$ is total on s^I_{mr}. Now, $deref^I([ref(t^m)]) = [t^m]$ and $ref^I(deref^I([t^{mr}])) = [t^{mr}]$ by **Ref**, **Deref** and Lemma 2.3.9, i.e. for all $m \in [0..n]$, ref^I is bijective on s^I_m and $deref^I$ is its inverse function on s^I_{mr}..

Functions $f \in \mathbf{F}$ are interpreted, s.t. they fulfil the rank conditions, since if f has the rank $(f : s''_1, \ldots, s''_q \to s'')$ and $([t_1], \ldots, [t_q]) \in s''^I_1 \times \ldots \times s''^I_q$, then we know by definition of s''^I, that $[f(t_1, \ldots, t_q)] \in s''^I$. Furthermore, Lemma 2.3.10 implies, that $f(t_1, \ldots, t_q) \in \mathbf{T}_{s''}(\mathbf{V})$, i.e. $[f(t_1, \ldots, t_q)]$ is defined.

The choice function $choose^I$ gives $choose^I([t]) = [choose(t)]$ for any $[t] \in s^I_m$ and $m \in [1..n]$. From $[t] \in s^I_m$, we know by Lemma 2.3.10 that $t \in T^m(\mathbf{V})$, i.e. $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} choose(t) \in t$ by **Choice** and **Substitutivity**. Consequently, $[choose(t)] \in [t]$ by definition of $[t]$, and $[choose(t)]$ is unique, since it is a unique representative of its $=$-equivalence class.

By Lemma 2.3.9, we know that $\in$ and $=$ are interpreted as needed for R^n-interpretations. Hence, $\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})$ is an R^n-interpretation.

In order to prove that it is an R^n-model of $\mathbf{P}$, we can proceed as in the corresponding Lemma for $\mathcal{OSE}$-logics (cf. Section 2.2), except that Lemma 2.3.10 has to be used in this case. □

It remains to prove completeness of deduction and initiality of $\mathbf{I}_R$.

Theorem 2.3.12 *Let* $\mathbf{T}_{s_m}(\mathbf{X}) \neq \emptyset$ *for all* $m \in [0..n]$, $\mathbf{P}$ *be an* R^n*-*Σ*-presentation, and* L *a* $(\Sigma, \mathbf{X})$*-atom with* $(\Sigma, \mathbf{P}, \mathbf{X}) \models_{R^n} L$. *Then* $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} L$.

Proof: Completely analogous to the corresponding Theorem of $\mathcal{OSE}$-logics, where $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$ has to be replaced by $\mathbf{I}_{R,X}$ and by using Lemma 2.3.11 instead of the corresponding one for $\mathcal{OSE}$-logics. □

Theorem 2.3.13 *Let* $\mathbf{T}_{s_m}(\mathbf{X}) \neq \emptyset$ *for all* $m \in [0..n]$ *and* $\mathbf{P}$ *be an* R^n*-*Σ*-presentation. For all* R^n*-models* $\mathbf{A}$ *of* $\mathbf{P}$*, there is a unique* Σ*-homomorphism* $h : \mathbf{I}_R \to \mathbf{A}$.

Proof: Let I be the shorthand for $\mathbf{I}_R$ and h be defined as follows:

$$\forall k \in [0..n], \forall t'^k \in T'^k, h([t'^k]) = (t'^k)^{\mathbf{A}}$$

First of all notice that h is a function, because of Lemma 2.3.9 and the fact that $\mathbf{A}$ is an R^n-model of $\mathbf{P}$, i.e. has to satisfy all deducible equalities.

Let us prove $h(s^I) \subseteq s^{\mathbf{A}}$ for all $s \in \mathbf{S}$. Suppose $[t] \in s^I$ for some $s \in \mathbf{S}$. Then $t \in \mathbf{T}_s$, by Lemma 2.3.10 and $[t]$ is mapped to an element in $s^{\mathbf{A}}$, which can be shown by structural induction on t: If t is a constant c in s, then $\mathbf{A}$ has to map c to some element of $s^{\mathbf{A}}$, i.e. $h([c]) \in s^{\mathbf{A}}$. Else, t must be of the form $f(t_1, \ldots, t_q)$ and $h([t_i]) = t_i^{\mathbf{A}} \in s''^{\mathbf{A}}_i$ for all $i \in [1..q]$, whenever $t_i \in \mathbf{T}_{s''_i}$. Hence, $h([f(t_1, \ldots, t_q)]) = f^{\mathbf{A}}(t_1^{\mathbf{A}}, \ldots, t_n^{\mathbf{A}}) \in s^{\mathbf{A}}$, since t is in $\mathbf{T}_s$ and $\mathbf{A}$ has to respect function ranks. This implies also immediately $h(C^I) \subseteq C^{\mathbf{A}}$.

The property $\forall f \in \mathbf{F}, h((f(t_1, \ldots, t_q))^I) = f^{\mathbf{A}}(h(t_1^I), \ldots, h(t_q^I))$, if $(t_1^I, \ldots, t_n^I) \in \mathcal{D}om(f^I)$ (i.e. $[f(t_1, \ldots, t_n)]$ is defined) is a direct consequence of the definition. Remains to show that $h(p^I) \subseteq p^{\mathbf{A}}$. Assume $([t_1], \ldots, [t_q]) \in p^I$, i.e. $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} p(u_1, \ldots, u_q)$ and $[u_i] = [t_i]$ for $i \in [1..q]$ by definition. Hence, $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{\mathcal{RNL}} u_i = t_i$ by Lemma 2.3.9. Therefore, we get $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} p(t_1, \ldots, t_q)$ by paramodulation and consequently we know that $(h([t_1]), \ldots, h([t_q])) = (t_1^{\mathbf{A}}, \ldots, t_q^{\mathbf{A}}) \in p^{\mathbf{A}}$, since $\mathbf{A}$ is an R^n-model of $\mathbf{P}$.

The proof of uniqueness is an induction on the size n of t. Assume $h' : \mathbf{I}_R \to \mathbf{A}$ is another homomorphism. In the case of a constant c, $h([c]) = c^{\mathbf{A}} = h'(c)$ is obvious. If t is of the form $f(t_1, \ldots, t_n)$, we get the result as usual by the induction hypothesis:

$$\begin{aligned} h([f(t_1, \ldots, t_n)]) &= f^{\mathbf{A}}(h([t_1]), \ldots, h([t_n])) \\ &= f^{\mathbf{A}}(h'([t_1]), \ldots, h'([t_n])) \\ &= h'([f(t_1, \ldots, t_n)]) \end{aligned}$$

Note that the $[t_i]$, $i \in [1..q]$, have to be defined by the definition of $[f(t_1, \ldots, t_n)]$. □

3.3 Discussion

Remark that R^n-logics are powerful enough to define (total) set intersection and inclusion, but not singletons nor set union:

Example 2.3.14 *Let* $m \in [1..n], k < m$*. Set intersection can be defined as follows:*

$$x'^{m-1} \in inter(s^m, t^m) \Leftrightarrow x'^{m-1} \in s^m \wedge x'^{m-1} \in t^m.$$

Remark that from right to left we obtain two clauses. Note furthermore that intersection has to be total, which may be very inconvenient in practice, since the empty set is prohibited.

The intuition for inclusion is the same as for **Ext***:*

$$\begin{aligned} x^m \subseteq y^m &\Leftarrow choose(x^m) \in y^m, \\ z'^{m-1} \in y^m &\Leftarrow z'^{m-1} \in x^m \wedge x^m \subseteq y^m. \end{aligned}$$

Remark that $choose(x^m) \in y^m \Leftarrow x^m \subseteq y^m$ *is not a well-defined clause in* R^n*-presentations, since choose appears in the head of the clause, in contradiction to Definition 2.3.2. Inclusion may lead to an alternative definition of extensionality:*

$$x^m = y^m \Leftrightarrow x^m \subseteq y^m \wedge y^m \subseteq x^m.$$

Tautology	$\dfrac{}{P \vdash_{ET} L \Leftarrow L}$
Monotonicity	$\dfrac{E \vdash_{ET} L \Leftarrow \Gamma}{E \vdash_{ET} L \Leftarrow \Gamma, L'}$
Reflexivity	$\dfrac{}{E \vdash_{ET} t = t}$
Symmetry	$\dfrac{E \vdash_{ET} t = t'}{E \vdash_{ET} t' = t}$
Transitivity	$\dfrac{E \vdash_{ET} (t = t'), (t' = t'')}{E \vdash_{ET} t = t''}$
Substitution	$\dfrac{E \vdash_{ET} t = t'}{E \vdash_{ET} \sigma(t) = \sigma(t')}$
Replacement	$\dfrac{E \vdash_{ET} (t_1 = t'_1), \ldots, (t'_m = t'_m)}{E \vdash_{ET} f(t_1, \ldots, t_m) = f(t'_1, \ldots, t'_m)}$
ModusPonens	$\dfrac{E \vdash_{ET} (L \Leftarrow L', \Gamma), (L' \Leftarrow \Gamma)}{E \vdash_{ET} L \Leftarrow \Gamma}$
TypingEquals	$\dfrac{E \vdash_{ET} (t = t'), (t : A)}{E \vdash_{ET} t' : A}$
EquatingTypes	$\dfrac{E \vdash_{ET} (A = A'), (t : A)}{E \vdash_{ET} t : A'}$

Figure 2.6: **ET** : The **ETL** Calculus

One might try to define singletons $sgl(t^m)$ by:

$$x'^m = t^m \Leftrightarrow x'^m \in sgl(t^m).$$

But this contradicts the sort preservation for the equation in the head of the clause.

Union would be:

$$x^k \in union(s^m, t^m) \Leftrightarrow x^k \in s^m \vee x^k \in t^m.$$

This does unfortunately not result in a Horn clause from left to right. However, we may define a weak union that covers the exact one:

$$\begin{gathered} s^m \subseteq union(s^m, t^m) \Leftarrow, \\ t^m \subseteq union(s^m, t^m) \Leftarrow, \\ union(s^m, t^m) \subseteq u^m \Leftarrow s^m \subseteq u^m \wedge t^m \subseteq u^m. \end{gathered}$$

Singletons might be introduced by special functions, corresponding sorts and axioms saying that the choice in a singleton is deterministic.

In order to compare R^n-logics, which are rather simple with only four additional deduction rules to $\mathcal{OSE}$-logics, with the previously-developed frameworks of equational type logic (ETL) and unified algebras, let us try to code them in our framework and see what happens. ETL [Manca *et al.*, 1990] is in fact a fragment of R^n-logics, as we claim in the following theorem. The corresponding ET calculus from [Manca *et al.*, 1990] is shown in Figure 2.6. An equational type presentation is a triple $\langle \Omega, V, E \rangle$, s.t. Ω is a set of function symbols (with associated arity), V is a set of unsorted variables and E is a set of Ω-Horn clauses using only $=$ and $:$ as binary operators.

Theorem 2.3.15 *For any equational type presentation $\langle \Omega, V, E \rangle$, there is an R^n-logical program* **P**, *s.t.* $\mathbf{P} \models_{R^n} \phi$ *iff* $E \models_{ETL} \phi$ *for all Ω-atoms ϕ.*

$$
\begin{array}{ll}
x = y \Leftarrow x \leq y \wedge y \leq x, & x \leq z \Leftarrow x \leq y \wedge y \leq z, \\
x \leq x \Leftarrow, & \mathtt{nothing} \leq x \Leftarrow, \\
x \mid y \leq z \Leftarrow x \leq z \wedge y \leq z, & x \leq x \mid y \Leftarrow, \\
y \leq x \mid y \Leftarrow, & x\&y \leq y \Leftarrow, \\
z \leq x\&y \Leftarrow z \leq x \wedge z \leq y, & x\&y \leq x \Leftarrow \\
x\&(y \mid z) = (x\&y) \mid (x\&z) \Leftarrow, & x \mid (y\&z) = (x \mid y)\&(x \mid z) \Leftarrow, \\
x : y \Leftarrow x : x \wedge x \leq y, & \\
x : x \Leftarrow x : y, & x \leq y \Leftarrow x : y, \\
\multicolumn{2}{c}{f(\ldots, x_i, \ldots) \leq f(\ldots, x_i', \ldots) \Leftarrow x_i \leq x_i'}
\end{array}
$$

Figure 2.7: **UAA** : Unified Algebra Axioms

Proof: Let $n = 0$ and $\Sigma = (\mathbf{S}, \leq_{\mathbf{S}}, \mathbf{F}, \mathbf{R})$ with $\mathbf{S} = \{s_0, s_0', s_{0r}\}$, $\leq_{\mathbf{S}} = \{(s_{0r}, s_0')\}$ and:

- $\mathbf{R} = \{=, \in, :\}$ with $ranks(=) = \{(s_0, s_0), (s_0', s_0'), (s_{0r}, s_{0r})\}$, $ranks(:) = \{(s_0, s_0)\}, ranks(\in) = \emptyset$, and
- $\mathbf{F} = \Omega \cup \{ref, deref, choose\}$, s.t. all $f \in \Omega$ with arity n have $ranks(f) = \{(f : s_0, \ldots, s_0 \rightarrow s_0)\}$.

Let furthermore $V \subset \mathbf{X}_{s_0}$. Since $n = 0$, there is no rank for $\in$ and it can therefore not be used in $(\Sigma, \mathbf{X})$-clauses, i.e. also not in R^n-presentations. Now, the rules of ET calculus are clearly contained in $\mathcal{OSE}$-logic deduction rules. It is also easy to recognise that the additional deduction rules for R^n-logics do not interfere with Ω-atoms – only **Ref** and **Deref** can be used, since $n = 0$. The remaining five rules of $\mathcal{OSE}$-deduction are obviously equivalent (w.r.t. deduction of Ω-atoms) to those of the ET calculus. On the model level, there is only one difference concerning the carrier, which might be extended by additional elements in R^n-logics. However, if we restrict the models to those interpreting s_0 equal to C, then both model classes are equivalent. Remark that using $\in$ for : does not have the desired effect, since $\in$ has more properties (like the axiom of choice and extensionality). □

Concerning unified algebras, consider their axioms from [Mosses, 1989] in Figure 2.7. In the beginning, everything looks very close to R^n-logics. However, there is already one fact saying that $\mathtt{nothing} \leq x$, i.e. in the semantics of unified algebras that **nothing** represents the bottom of the lattice. Using set interpretations, this results in interpreting this constant as empty set, which is not allowed for R^n-interpretations.

Later on, we can find three clauses containing the :-relation. In [Mosses, 1989], we can find the definition of power algebras, which seem to be very close to R^n-logics and which define the :-relation as $\subseteq$, restricted to singleton arguments on the left. As we already saw, singletons are unfortunately not completely axiomatisable in R^n-logics, due to restrictions concerning the use of variables of sort $s_{mc}, m \in [1..n]$.

Another way might be the following: The :-relation is mapped to $\in$, providing a way to change levels, i.e. $\leq$ becomes set inclusion, & (total) set intersection and | is a weak form of set union, giving a set that covers union, but that is not necessarily minimal. Clearly, the three axioms in **UAA** concerning the :-relation must then be dropped, since they contradict the rank of $\in$. However, our type system provides us with similar information.

3.4 Relation To Other Work

As already mentioned, simple R^n-logics stem from naive set theory in that orders are added in such a way that valid formulas cannot lead to Russell's paradox. They can also be seen as fragment of Z [Spivey, 1988]. These observations make us optimistic with respect to the use of (simple or full) R^n-logics in the specification domain.

From an algebraic point of view, simple R^n-logics compare best with many- and order-sorted algebras [Goguen et Meseguer, 1992], already implemented via rewriting, for instance in OBJ-3 [Kirchner *et al.*, 1988a]. However, simple R^n-logics provide arbitrary terms as sorts and therefore surpass these approaches by far in expressivity. Polymorphic order-sorted algebras can be seen as a fragment of simple R^1-logics. Simple R^n-logics are also very close to power algebras, i.e. unified algebras with set interpretations [Mosses, 1989]. A slight extension of simple R^n-logics by explicit singletons provides initial models for a fragment of power algebras (although weakening unions and excluding the empty set). One may say that R^n-logics unify ETL [Manca *et al.*, 1990] and unified algebras, while preserving models from total G-algebras [Mégrelis, 1990], which are actually a fragment of G^1-logics.

The untyped, simply-typed and second-order polymorphically-typed λ-calculi have already been successfully specified [Hintermeier *et al.*, 1995b] in full R^3-logics. The resulting set-theoretic models provide function graphs containing reference pairs. References can be seen as analogous to the projection functions used in λ-calculus models like P_ω [Scott, 1976]. There, continuous functions over P_ω, the domain of finite sets of natural numbers, are converted into elements of P_ω and back, in order to obtain the reflexivity of P_ω as domain. Reflexivity of the domain then allows giving a consistent semantics to the untyped λ-calculus, where functions are self-applicable.

Universal algebra with higher-order types was developed by Meinke [Meinke, 1992], based on general models introduced by Henkin [Henkin, 1950]. Meinke proves the existence of initial, extensional models in equational theories following Henkin's observation that many-sorted first-order predicate logic deduction is complete for finite type theory with respect to general models. Our results differ from those in [Meinke, 1992] in that our formal basis is set-theoretic rather than purely algebraic, and a uniform treatment of typing and higher-order functions is presented, going beyond Church's simple types used by Meinke's approach.

Polymorphically-typed λ-calculi like $F_\omega^\leq$ have shown that types provide interesting features for functional programming. Martin-Löf type theory and AUTOMATH deal with types as with values, provide a large amount of uniformity and hence formal simplicity. Constructive type theory illustrates that typing can integrate program development and verification. Many of these features can also be recognized in R^n-/G^n-logics. Extensions of Martin-Löf type theories with names, i.e. references in our framework, building so-called universes have also been investigated. Names are currently used in several domains, including functional programming [Odersky, 1994] and concurrency theory [Milner, 1993], mainly for information hiding and avoiding name conflicts, in contrast to our work, where references are used globally for semantical and type theoretic reasons.

In the theorem proving domain, R^n-logics can be related to HOL [Gordon, 1993] and ISABELLE [Paulson, 1989], both based on type theory and providing means to reason in set theory (mainly ZF). HOL moreover has its semantics defined over set theory and provides choice functions. Automated reasoning in von Neumann/Bernays/Gödel axiomatic set theory was introduced by Boyer et al. [Boyer *et al.*, 1986], cf. also [Quaife, 1992]. The latter work contains a comparison of choice functions with Skolem functions, related to our use of choices to formulate extensionality. However, none of these works provide initial semantics or other model-theoretic

results concerning specifications.

4 G^n-Logics

Introduction

Analogously to (simple) R^n-logics, we can define (simple) G^n-logics, except that functions in G^n-logics are not necessarily total. Hence, if f is declared as a function e.g. from individuals to individuals in G^n-logics, then this does not imply that f is completely defined over all individuals. However, if f is defined for some individual ξ, then $f(\xi)$ has to be an individual. G^n-logics seem to be more straightforward to use for program specifications than R^n-logics. However, they are also a bit more complex since we have to handle definedness of a term t using an additional predicate $EX\ t$.

The G^n-logic framework shows its usefulness e.g. to code set intersections. Obviously, our assumption that all sets in the models have to be non-empty together with totality of functions would restrict our models a lot. In G^n-logics we can define non-empty (partial) intersection $\cap$ in a clean way as follows:

$$\mathtt{x} \in \mathtt{s} \Leftarrow \mathtt{x} \in \mathtt{s} \cap \mathtt{s'}, \mathtt{x} \in \mathtt{s'} \Leftarrow \mathtt{x} \in \mathtt{s} \cap \mathtt{s'},$$
$$\mathtt{x} \in \mathtt{s} \cap \mathtt{s'} \Leftarrow \mathtt{x} \in \mathtt{s} \wedge \mathtt{x} \in \mathtt{s'}$$

Now, we can give constants `nat,int,list` of type `set` and `0` of type `elem`, together with the axioms $\mathtt{0} \in \mathtt{nat}$, $\mathtt{0} \in \mathtt{int}$, implying that $\mathtt{int} \cap \mathtt{nat}$ is defined in all models of these axioms, whereas $\mathtt{int} \cap \mathtt{list}$ is not defined for all these models.

G^n-logics were inspired by G-algebras [Mégrelis, 1990], but can also be seen as closely related to Scott's logic of partial equality [Scott, 1977; Fourman et Scott, 1977], where strictness, as reflected by a well-definedness axiom in what follows, leads to a simplified definition of equality (see also [Poigné, 1988]) in our case, in that we may use one equality relation only instead of two (strict and unstrict).

4.1 Models and Axioms

G^n-signatures are just extensions of R^n-signatures.

Definition 2.4.1 *A* G^n-signature *is an* R^n*-signature* $\Sigma = (\mathbf{S}, \leq_{\mathbf{S}}, \mathbf{F}, \mathbf{R})$, *s.t.* $\mathbf{R}$ *contains the relations* $\doteq$ *and* EX *with* $ranks(\doteq) = \{(\doteq : s'_i s'_i) \mid i \in [0..n]\}$ *and* $ranks(EX) = \{(EX : s'_i) \mid i \in [0..n]\}$.

A G^n*-(*Σ*-)*presentation *is an* R^n*-*Σ*-presentation, s.t.* Σ *is a* G^n*-signature and* $\mathbf{P}$ *does not contain any occurrence of* $=$*, the* $\mathcal{OSE}$*-logics identity relation.*

The difference between R^n-and G^n-interpretations seems small at first glance, but is more fundamental, since functions may now be partial and not necessarily total on their ranks any more:

Definition 2.4.2 *Let* Σ *be a* G^n*-signature,* $\mathbf{X}$ *a set of* $\mathbf{S}$*-sorted variables and* $\mathbf{P}$ *a set of* $(\Sigma, \mathbf{X})$*-clauses.*

A G^n-Σ-interpretation $\mathbf{A}$ *is a pair* $(C^{\mathbf{A}}, .^{\mathbf{A}})$ *of a* carrier $C^{\mathbf{A}}$ *and an interpretation function* $.^{\mathbf{A}}$ *defined over all sort, function and predicate symbols, s.t. the part concerning sorts satisfies:*

- $\forall s \in \mathbf{S}, s^{\mathbf{A}} \subseteq C^{\mathbf{A}}$, $s^{\mathbf{A}} \neq \emptyset$,

- $\forall s, s' \in \mathbf{S}$, $s^{\mathbf{A}} \subseteq s'^{\mathbf{A}}$ *if* $s \leq_S s'$,
- $s'^{\mathbf{A}}_0$ *is a non-empty set of individuals,*
- *for* $i \in [1..n]$, $s^{\mathbf{A}}_i$ *contains only non-empty sets* $t^{\mathbf{A}}$ *of elements of* $s'^{\mathbf{A}}_{i-1}$, *s.t.* $t \in \mathbf{T}$,
- *for* $i \in [0..n]$, $s^{\mathbf{A}}_{ir}$ *is a set of symbolic names contained in* $s'^{\mathbf{A}}_0$,
- *for* $i \in [1..n]$, $s'^{\mathbf{A}}_i$ *is the set of all sets of elements in* $s'^{\mathbf{A}}_{i-1}$,

And the part concerning function symbols satisfies:

- $\forall f \in \mathbf{F}$, *if* $(f : s_1, \ldots, s_q \to s) \in ranks(f)$, *then* $f^{\mathbf{A}} : (s^{\mathbf{A}}_1 \times \ldots \times s^{\mathbf{A}}_q) \rightharpoonup s^{\mathbf{A}}$ *is a partial function,*
- $choose^{\mathbf{A}}$ *gives a unique element from any set in* $C^{\mathbf{A}}$ *given as argument (choice function), and*
- $ref^{\mathbf{A}}$ *is a bijective function over* $s^{\mathbf{A}}_i$ *taking an element of* $s^{\mathbf{A}}_i$, $i \in [0..n]$ *and returning its unique name in* $s^{\mathbf{A}}_{ir}$, $deref^{\mathbf{A}}$ *is its inverse function over* $s^{\mathbf{A}}_{ir}$.

Finally, the part concerning relations satisfies:

- $\forall p \in \mathbf{R}$, *if* $ar(p) = n$, *then* $p^{\mathbf{A}} \subseteq (C^{\mathbf{A}})^n$.
- $\in$ *is interpreted as the membership relation on sets in* $C^{\mathbf{A}}$,
- $\doteq$ *is interpreted as diagonal on* $C^{\mathbf{A}} \times C^{\mathbf{A}}$ *and*
- EX *is interpreted as membership in* $C^{\mathbf{A}}$.

The class of all G^n-Σ-*interpretations is written* G^n-$Interp(\Sigma)$. *Let* $\mathbf{A}$ *be a* G^n-Σ-*interpretation. An* $(\mathbf{A}, \mathbf{X})$-variable assignment *is a function* $\delta : \mathbf{X} \to C^{\mathbf{A}}$, *s.t.* $\delta(x) \in s^{\mathbf{A}}$ *if* $x \in \mathbf{X}_s$.

Let $\mathbf{A}, \mathbf{B}$ *be* G^n-Σ-*interpretations. A* G^n-Σ-homomorphism *is a function* $h : \mathbf{A} \to \mathbf{B}$ *satisfying the four conditions of Definition 2.2.2 of* Σ-*homomorphisms.*

Remark first of all that there is no more totality obligation for function symbols. However, a function f with $(f : s_1, \ldots, s_q \to s) \in ranks(f)$ has to respect the range sort $s^{\mathbf{A}}$ whenever it is defined over $(s^{\mathbf{A}}_1 \times \ldots \times s^{\mathbf{A}}_q)$. Now, G^n-formulas have to determine the domain of functions. Similarly to R^n-logics, the satisfaction relation $\models$ has changed and we write $\models_{G^n}$ instead of $\models$. Models and validity of Σ-goals can then be defined in exactly the same way as in Section 2.2.1.0, using G^n-Σ-interpretations instead of Σ-interpretations. The redefinition of homomorphisms was necessary, since G^n-Σ-interpretations are not necessarily Σ-interpretations, due to the weaker conditions on function domains.

4.2 Deduction in G^n-logics

The less restrictive definition of term interpretations makes it impossible to use $\mathcal{OSE}$-deduction directly and unchanged. E.g., **Reflexivity** is no more sound. Instead, we have to ask for well-definedness of the interpretation of a term before deducing reflexivity. This is also the reason why we renamed $=$ into $\doteq$. Figure 2.8 shows the new set of deduction rules **GNL** incorporating well-definedness formulas and therefore more suitable for partial function handling. Remind that m ranges over $1..n$ and k over $0..n$.

Choice $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} choose(x^m) \in x^m}$ if $m \in [1..n]$

Ref $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} ref(deref(x^{kr})) \doteq x^{kr}}$ if $k \in [0..n]$

Deref $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} deref(ref(x^k)) \doteq x^k}$ if $k \in [0..n]$

Ext $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} choose(t^m) \in u^m,\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} choose(u^m) \in t^m}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} t^m \doteq u^m}$ if $m \in [1..n]$

WellDef $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} \Phi[t]}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ t}$ if $\Phi[t]$ *is a* $(\Sigma, \mathbf{X})$-atom containing t

PartialReflex $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ t}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} t \doteq t}$

Axioms $\dfrac{}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L \Leftarrow G}$ if $(L \Leftarrow G) \in \mathbf{P}$ is a $(\Sigma, \mathbf{X})$-clause

SubstConform $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L \Leftarrow G}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} \sigma(L) \Leftarrow \sigma(G)}$ if $\sigma \in \mathbf{P}\text{-}SUBST_\Sigma(\mathbf{X})$

Cut $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L \Leftarrow G \wedge L',\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L' \Leftarrow G'}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L \Leftarrow G \wedge G'}$

Paramodulation $\dfrac{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L[s] \Leftarrow G,\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} (s \doteq t) \Leftarrow G'}{(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L[t] \Leftarrow G \wedge G'}$

Figure 2.8: **GNL** : Horn Clause Deduction Rules for G^n-logics

As far as the confusion of the axioms with the old ones for $\mathcal{OSE}$-logics is harmless, as is the case for **Axioms**, **Cut** and **Paramodulation**, we avoid to change their names for the sake of simplicity. The new axiom **WellDef** deduces the well-definedness of a term in every model, if it appears in a theorem of **P**, **PartialReflex** is the restriction of **Reflexivity** to well-defined terms and **SubstConform** is the restriction of **Substitutivity** to so-called **P**-conform substitution, defined as follows:

Definition 2.4.3 *Let* $\sigma \in SUBST_\Sigma(\mathbf{X})$ *and* **P** *be a* G^n*-presentation. Then* σ *is* **P**-conform *if:*

$$\forall x \in Dom(\sigma)\ :\ (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ \sigma(x)$$

The set of all **P**-*conform substitutions is denoted by* $\mathbf{P}\text{-}SUBST_\Sigma(\mathbf{X})$.

P-conform substitutions have the nice property that they behave like Σ-substitutions. Do not worry about the use of $\models_{G^n}$ instead of $\vdash_{G^n}$ in the following lemma, the latter will imply the former when we have shown soundness of **GNL**.

Lemma 2.4.4 *Let*

- $\mathbf{P}$ *be a* G^n*-presentation,*
- $\sigma \in SUBST_\Sigma(\mathbf{X})$*, s.t.*
- $\forall x \in \mathcal{D}om(\sigma) : \mathbf{P} \models_{G^n} EX\ \sigma(x)$*,*
- $\mathbf{A}$ *be a* G^n*-model of* $\mathbf{P}$ *and*
- δ *a* $(\mathbf{A}, \mathbf{X})$*-variable assignment.*

Then, $\delta \circ \sigma$ *is a* $(\mathbf{A}, \mathbf{X})$*-variable assignment.*

Proof: We have to prove that $\delta \circ \sigma(x) \in sort(x)^{\mathbf{A}}$ for all $x \in \mathbf{X}$. If $x \notin \mathcal{D}om(\sigma)$, then this is implied by δ being a $(\mathbf{A}, \mathbf{X})$-variable assignment. Otherwise, we know that $\delta(\sigma(x)) \in C^{\mathbf{A}}$, since $\mathbf{P} \models_{G^n} EX\ \sigma(x)$. Hence, $\delta(\sigma(x)) \in sort(x)^{\mathbf{A}}$, since $\sigma(x) \in \mathbf{T}_{sort(x)}(\mathbf{X})$ and the range condition for G^n-interpretations of function symbols. □

One could define some kind of G-logics as a starting point for G^n-logics, i.e. a slightly changed version of $\mathcal{OSE}$-logics, including **WellDef**, **PartialReflex** and **SubstConform** instead of **Reflexivity** and **Substitutivity** in its meta-logical axiomatisation. However, we started with $\mathcal{OSE}$-logics as basis for R^n-logics, because we wanted to provide a basis for comparisons and introduce smoothly the new components, keeping the changes as small as possible. Now that we know of R^n-logics, this should be no more necessary and we can give immediately a new axiomatic system, containing only slight changes w.r.t. the rules in Figure 2.2. As usual, we have to start with the proof of soundness for G^n-deduction. We write $\equiv$ for the meta-logical identity relation.

Theorem 2.4.5 *Let* $\mathbf{P}$ *be a* G^n*-presentation. Then, all deduction rules in* **GNL** *are sound w.r.t.* G^n*-models.*

Proof: We have to show that if each of the clauses in the premiss of a clause is satisfied by each G^n-model, so is the conclusion. The proof is by induction on the deduction tree. Remark that the proof of the condition for σ in **SubstConform**, i.e. that $\forall x \in \mathcal{D}om(\sigma), (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ \sigma(x)$, has to be taken as a part of the deduction tree.

- **Choice** : Let $\mathbf{A}$ be an arbitrary G^n-model of $\mathbf{P}$. Since all elements of $s_m^{\mathbf{A}}$ for $m \in [1..n]$ have to be non-empty sets of grade m, the result of applying the choice function *choose* must return an element of $\delta(x^m)$ for all $(\mathbf{X}, \mathbf{A})$-variable assignments δ.
- **Ref** : Let $\mathbf{A}$ be as in the last case. All elements of $s_{mr}^{\mathbf{A}}$ for $m \in [0..n]$ correspond by definition of G^n-models one-to-one with elements of $s_m^{\mathbf{A}}$. Furthermore, $deref^{\mathbf{A}}$ has to be the inverse function of $ref^{\mathbf{A}}$, the bijective function associating a unique name in $s_{mr}^{\mathbf{A}}$ to each element of $s_m^{\mathbf{A}}$, i.e. $(ref \circ deref)_{|s_{mr}^{\mathbf{A}}} \equiv id_{s_{mr}^{\mathbf{A}}}$, the identity function on $s_{mr}^{\mathbf{A}}$, i.e. $ref^{\mathbf{A}}(deref^{\mathbf{A}}(\delta(x^{mr}))) \equiv \delta(x^{mr})$ for all $(\Sigma, \mathbf{X})$-variable assignments δ.
- **Deref** : Analogous to **Ref**, knowing that $(deref \circ ref)_{|s_m^{\mathbf{A}}} \equiv id_{s_m^{\mathbf{A}}}$, the identity function on $s_m^{\mathbf{A}}$.
- **Ext** : If we know for all G^n-models $\mathbf{A}$ of $\mathbf{P}$ and all $(\mathbf{X}, \mathbf{A})$-variable assignments δ, that $choose^{\mathbf{A}}(\delta(t^m)) = \delta(choose(t^m)) \in \delta(u^m)$ and $choose^{\mathbf{A}}(\delta(u^m)) = \delta(choose(u^m)) \in \delta(t^m)$, i.e. that any choice in $\delta(t^m)$ is in u^m and vice versa, then clearly $\delta(u^m) = \delta(t^m)$.

- **WellDef** : If $\Phi[t]$ is satisfied by every G^n-model $\mathbf{A}$ of $\mathbf{P}$, then $t^{\mathbf{A}}$ has to be defined for each such $\mathbf{A}$, since satisfaction of an atom $p(t_1, \ldots, t_q)$ in $\mathbf{A}$ is equivalent to $(t_1^{\mathbf{A}}, \ldots, t_q^{\mathbf{A}}) \in p^{\mathbf{A}} \subseteq (C^{\mathbf{A}})^q$, i.e. $t^{\mathbf{A}}$ is in $C^{\mathbf{A}}$.
- **PartialReflex** : If $t^{\mathbf{A}} \in C^{\mathbf{A}}$ for all G^n-models $\mathbf{A}$ of $\mathbf{P}$, then $(t^{\mathbf{A}}, t^{\mathbf{A}})$ is element of the diagonal in $C^{\mathbf{A}} \times C^{\mathbf{A}}$ in all such $\mathbf{A}$.
- **Axioms** : Trivial.
- **SubstConform** : By Lemma 2.4.4 and the induction hypothesis, we know that $\delta \circ \sigma$ is a $(\mathbf{A}, \mathbf{X})$-variable assignment whenever δ is. Furthermore $\mathbf{A}, \delta \models_{G^n} L$ if $\mathbf{A}, \delta \models_{G^n} G$ for all G^n-models $\mathbf{A}$ of $\mathbf{P}$ and $(\mathbf{A}, \mathbf{X})$-variable assignments δ, i.e. we can take $\delta' = \delta \circ \sigma$ in order to verify that $\mathbf{A}, \delta \models_{G^n} \sigma(L)$ if $\mathbf{A}, \delta \models_{G^n} \sigma(G)$: This follows from $\mathbf{A}, \delta' \models_{G^n} L$ if $\mathbf{A}, \delta' \models_{G^n} G$
- **Cut. Paramodulation** : Analogous to $\mathcal{OSE}$-logics (cf. Theorem 2.2.13 in Section 2.2).

□

Now, that we know soundness, we get the following analogon to Lemma 2.2.8 in Section 2.2:

Lemma 2.4.6 *Let Σ be a G^n-signature, $\mathbf{P}$ be a G^n-presentation, $\mathbf{A}$ a G^n-interpretation, G a $(\Sigma, \mathbf{X})$-goal, $\sigma \in \mathbf{P}\text{-}SUBST_\Sigma(\mathbf{X})$ and δ be a variable assignment for $\mathbf{X}$ in $\mathbf{A}$.*

Then $\mathbf{A}, \delta \models_{G^n} \sigma(G)$ iff $\mathbf{A}, \delta \circ \sigma \models_{G^n} G$.

Proof: The proof is very similar to the one of Lemma 2.2.8 in Section 2.2. We know by Lemma 2.4.4 and the soundness of **WellDef** that $\delta' = \delta \circ \sigma$ is a $(\mathbf{A}, \mathbf{X})$-variable assignment. Let $p(\ldots, t, \ldots) \in G$. Then:

$$\begin{aligned} \mathbf{A}, \delta \models_{G^n} \sigma(p(\ldots, t, \ldots)) &\text{ iff } \mathbf{A}, \delta \models_{G^n} p(\ldots, \sigma(t), \ldots) \\ &\text{iff } (\ldots, \delta(\sigma(t)), \ldots) \in p^{\mathbf{A}} \\ &\text{iff } (\ldots, \delta'(t), \ldots) \in p^{\mathbf{A}} \\ &\text{iff } \mathbf{A}, \delta' \models_{G^n} p(\ldots, t, \ldots) \end{aligned}$$

□

A first remark for this section concerns constants as partial functions. One might think that a constant c, that is declared in the signature, must always denote something, i.e. be mapped to an element of $C^{\mathbf{A}}$ for every G^n-Σ-model. However, similar to G-algebras, this does not hold in our framework. Only if we can deduce $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ c$, it has a denotation in every model of $\mathbf{P}$. The reason for this is hidden in the definition of constants as partial functions, where we follow [Mégrelis, 1990].

Remark also that we avoided the problem of empty carrier sets by requiring every sort in $\mathbf{S}$ to be interpreted as non-empty set in the definition of G^n-models. So, e.g. $\mathbf{P} = \{f(x^0) = x^0\}$ does not have the trivial model interpreting s_0 as $\emptyset$. This simplifies the proofs in this section. However, if one wants to check that each sort has at least one denoting term in it, which is necessary for the construction of an initial model, we can give a sufficient criterion. Figure 2.9 shows the only rule for such a test.

Similar to Theorem 2.2.11, we get the following result:

Lemma 2.4.7 *Let $\Sigma = (\mathbf{S}, \leq_{\mathbf{S}}, \mathbf{F}, \mathbf{R})$ and $\mathbf{P}$ a G^n-Σ-presentation.*

If $(\emptyset) \vdash_{G\_Nonempty} (\mathbf{S})$, then for all $s \in \mathbf{S}$, there is a $t_s \in \mathbf{T}_s$ with $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{WellDef} EX\ t_s$ and $s^{\mathbf{A}}$ is non-empty for every G^n-model $\mathbf{A}$ of $\mathbf{P}$.

G_Nonempty	$\dfrac{(N)}{(N \cup \{s_m, s'_m, s_{mr}\})}$	if $\exists t^m \in \{t \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{WellDef} EX\ t\}$ and $\forall x \in \mathcal{V}ar(t^m)$, $sort(x) \in N$ and $s_m \notin N$

Figure 2.9: Inference Rule For a Sufficient Criterion of Non-Emptiness of Sorts in G^n-logics

Proof: By induction on the number n of derivation steps, we prove that if N is the current sort set after n steps, then all $s \in N$ fulfil $s^{\mathbf{A}} \neq \emptyset$ for all G^n-models $\mathbf{A}$ of $\mathbf{P}$. If $n = 0$, then this holds trivially, since $N = \emptyset$. Otherwise, we know that $\mathbf{A} \models_{G^n} EX\ t^m$ by the soundness of **WellDef**. By the condition on the variables of t^m in **G_Nonempty** and the induction hypothesis, we know that there are ground terms for any sort of a variable occurring in t^m. Let σ be the Σ-substitution replacing the variables in t^m by these terms. Then, $\sigma(t^m) \in T^m$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ \sigma(t)$ using the **SubstConform** rule, i.e. by the soundness of **GNL**, that $\sigma(t)^{\mathbf{A}} \in C'^{\mathbf{A}}$.

Now, for every $t \in \mathbf{T}_{s'}$ with $t^{\mathbf{A}} \in C'^{\mathbf{A}}$, we can prove by structural induction on t, that $t^{\mathbf{A}} \in s'^{\mathbf{A}}$. Remark that $t^{\mathbf{A}} \in C'^{\mathbf{A}}$ implies also $u^{\mathbf{A}} \in C'^{\mathbf{A}}$ for all subterms u of t. Let t be of the form $f(t_1, \ldots, t_q)$ with $(f : s'_1, \ldots, s'_q \to s) \in ranks(f)$ for some $s \leq_{\mathbf{S}} s'$ and $\forall i \in [1..q]$, $t_i \in \mathbf{T}_{s'_i}$, since $t \in \mathbf{T}_{s'}$ and therefore $t_i^{\mathbf{A}} \in s'^{\mathbf{A}}_i$ by the induction hypothesis. Consequently, $f(t_1, \ldots, t_q)^{\mathbf{A}} \in s^{\mathbf{A}}$, since $\mathbf{A}$ has to fulfil the ranks of function symbols, whenever the function is defined for the given arguments.

Hence, $(t^m)^{\mathbf{A}} \in s^{\mathbf{A}}_m$ and the ranks of *ref* and *choose* give the non-emptiness of $s'^{\mathbf{A}}_m$ and $s^{\mathbf{A}}_{mr}$ in the same way. □

An easy way to make the test succeed is to add 'universal' constants Ω_m of sort s_{m+1} for each $m \in [0..n-1]$, together with the formulas $x^m \in \Omega_m$, i.e. semantically Ω_0 stands for all individuals and Ω_m with $m > 0$ for all sets of order m. This technique was already employed in G-algebra [Mégrelis, 1990].

4.3 The Initial Model

In difference to R^n-logics, we do not need to interpret all terms in $\mathbf{T}$, the set of all Σ-ground terms. Instead, we just take the set of all terms for which we surely know that they have to denote an element in all models. Hence, the semantics of satisfaction in G^n-logics, in particular the definition of $\mathbf{A}, \delta \models_{G^n} p(t_1, \ldots t_q)$, which allows us infer that $(\delta(t_1), \ldots, \delta(t_q)) \in (C'^{\mathbf{A}})^q$, is now the determining factor for well-definedness, and no more syntactic typability, as this was the case for R^n-logics (and $\mathcal{OSE}$-logics, of course).

Remark that most of the changes in the following are replacements of $=$ by $\doteq$. The biggest change can be found in the interpretation of sorts and functions, since the domain is now typically much smaller.

Definition 2.4.8 *Let* $\mathbf{V} \subseteq \mathbf{X}$. *The interpretation* $\mathbf{I}_{G,\mathbf{P}}(\mathbf{V})$ *is* I *as follows:*

- $\forall k \in [0..n], t'^k \in T^0(\mathbf{V}) \cup (T'^k(\mathbf{V}) \setminus T^k(\mathbf{V}))$,
  $[t'^k]$ is a representative of $\{u'^k \in T'^k(\mathbf{V}) \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} u'^k \doteq t'^k\}$ *if* $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ t'^k$,
  (Remark that this includes the cases $t^0 \in T^0$ *and* $t^{kr} \in T^{kr}(\mathbf{V})$ *for* $k \in [0..n]$.*)*
- $\forall m \in [1..n], \forall t^m \in T^m(\mathbf{V})$:
  $[t^m] = \{[u'^m] \mid u'^m \in T'^m(\mathbf{V})$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{\mathcal{RNL}} u'^m \in t^m\}$,

if $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ t^m$,

- $\forall s \in \mathbf{S},\ s^I = \bigcup_{s' \leq_{\mathbf{S}} s} \mathbf{V}_{s'} \cup \{[f(t_1, \ldots, t_q)] \mid$ $(f : s'_1, \ldots, s'_q \to s') \in ranks(f)$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ f(t_1, \ldots, t_q)$ and $\forall i \in [1..q],\ [t_i] \in s'^I_i$ and $s' \leq_{\mathbf{S}} s\}$, s.t. s^I is minimal,
- $C^I = \bigcup_{s \in \mathbf{S}} s^I$.
- $\forall m \in [0..n], \forall f \in \mathbf{F}$, *if* $(f : s''_1, \ldots, s''_q \to s'') \in ranks(f)$, *then:*
  - $Dom(f^I) = \{([t_1], \ldots, [t_q]) \in (s'^I_1 \times \ldots \times s'^I_q) \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ f(t_1, \ldots, t_q)\}$ *and*
  - $f^I([t_1], \ldots, [t_q]) = [f(t_1, \ldots, t_q)]$, *if* $([t_1], \ldots, [t_q]) \in Dom(f^I)$,
- $\forall p \in \mathbf{R}$:
  - $Dom(p^I) = (C^I)^q$ *and*
  - $p^I = \{([t_1], \ldots, [t_q]) \mid (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} p(t_1, \ldots, t_q)$ and $\forall i \in [1..n], [t_i] \in C^I\}$.

As in R^n-logics, we may write $\mathbf{I}_G$ and $\mathbf{I}_{G,X}$ instead of $\mathbf{I}_{G,\mathbf{P}}(\emptyset)$ and $\mathbf{I}_{G,\mathbf{P}}(\mathbf{X})$, respectively. In order to prove that $\mathbf{I}_G$ is a G^n-model, we proceed step-wise. First, we prove that the term interpretations are well-defined. Therefore, we need something like Lemma 2.3.8 for the typability of terms in deduced equations. But this is obvious, since:

Lemma 2.4.9 *Let*

- $\Sigma = (\mathbf{S}, \leq_{\mathbf{S}}, \mathbf{F}, \mathbf{R})$ *be a* G^n*-signature and*
- Σ' *be the* R^n*-signature* $(\mathbf{S}, \leq_{\mathbf{S}}, \mathbf{F}, \mathbf{R} \setminus \{EX, \doteq\})$.
- $\mathbf{P}$ *be a* G^n*-presentation and* $\mathbf{P}'$ *be obtained from* $\mathbf{P}$ *by deleting all clauses containing* EX *in its head, removing all literals of the form* $EX\ u$ *in the remaining clauses' bodies and finally replacing all* $\doteq$ *by* $=$.

Then, $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} t \doteq t'$ *implies* $(\Sigma', \mathbf{P}, \mathbf{X}) \vdash_{\mathcal{RNL}} t = t'$.

Proof: First of all remark that $=$ is by definition in $\mathbf{R}$. Clearly, all **PartialReflex** steps can be replaced by **Reflexivity** steps and **SubstConform** by **Substitutivity**. Hence, any subproof involving EX can be pruned.

Now, we still have to prune **Cut** steps into literals of the form $EX\ u$. The so resulting derivation tree is a valid R^n-deduction using $\mathbf{P}$ and $\vdash_{\mathcal{RNL}}$, if we replace all used clauses by their reduced ones, i.e. without EX and with $=$ in place of $\doteq$. □

An informal interpretation of this last result is simply '$\doteq^{\mathbf{I}_G(\mathbf{V})}$ for $\mathbf{P}$ is included in $=^{\mathbf{I}_R(\mathbf{V})}$ for $\mathbf{P}'$'. We get almost immediately:

Corollary 2.4.10
Let $\mathbf{P}$ *be a* G^n*-presentation.*

$$\text{If } (\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} u \doteq v \text{ then } (u \in \mathbf{T}_s(\mathbf{X}) \text{ iff } v \in \mathbf{T}_s(\mathbf{X}))$$

Proof: Assuming the contrary would result in a contradiction to Lemma 2.4.9 and 2.3.8. □

Lemma 2.4.11 *Let* $\mathbf{V} \subseteq \mathbf{X}$ *and consider* $[.]$ *as defined for* $\mathbf{I}_{G,\mathbf{P}}(\mathbf{V})$.
For all $k \in [0..n], v'^k, t'^k \in T'^k(\mathbf{V}) \setminus T^k(\mathbf{V})$:

$$(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v'^k \doteq t'^k \text{ iff } [v'^k] = [t'^k] \text{ and } [t'^k] \in C^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}. \quad (1)$$

For all $k \in [0..n], v^k, t^k \in T^k(\mathbf{V})$:

$$(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v^k \doteq t^k \text{ iff } [v^k] = [t^k] \text{ and } [t^k] \in C^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}. \quad (2)$$

And for all $k \in [1..n], v'^{k-1} \in T'^{k-1}(\mathbf{V}), t^k \in T^k(\mathbf{V})$:

$$(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{RNL} v'^{k-1} \in t^k \text{ iff } [v'^{k-1}] \in [t^k] \text{ and } [t^k] \in C^{\mathbf{I}_{R,\mathbf{P}}(\mathbf{V})}. \quad (3)$$

Proof: Literally the same as for Lemma 2.3.9, except that references to Lemma 2.3.8 have to be replaced by references to Corollary 2.4.10 and $=$ has to replaced by $\doteq$. □

Remains the analogon of Lemma 2.3.10 in order to prove that $\mathbf{I}_G(\mathbf{V})$ is a model of $\mathbf{P}$.

Lemma 2.4.12 *Let* $\mathbf{V} \subseteq \mathbf{X}$, t *be an extended term (i.e. a not necessarily well-sorted term respecting function arities) and* $[t] \in s^{\mathbf{I}_{G,\mathbf{P}}(\mathbf{V})}$ *for some* $s \in \mathbf{S}$. *Then,* $t \in \mathbf{T}_s(\mathbf{V})$.

Proof: The proof is the translated version of Lemma 2.3.10, where we have to replace the reference to Lemma 2.3.9 by one on Lemma 2.4.11 and the operator U has to be adapted to the new definition, which allows for the same conclusions. □

Now, we can easily prove the main goal of this section:

Theorem 2.4.13 *Let* $\mathbf{V} \subseteq \mathbf{X}$. *If* $\mathbf{T}_{s'_n} \neq \emptyset$ *and* $\mathbf{T}_{s_m} \neq \emptyset$ *for all* $m \in [0..n]$, *then* $\mathbf{I}_G(\mathbf{V})$ *is a* G^n*-model of* $\mathbf{P}$.

Proof: Let I stand for $\mathbf{I}_G(\mathbf{V})$ in the following. By our preconditions we get the non-emptiness of all s^I for $s \in \mathbf{S}$. Once again, the following statements are easy to verify: All denoting terms $t^0 \in T^0(\mathbf{V})$ are interpreted as terms in s_0^I, the set of individuals represented by a term. All denoting terms $t^m \in T^m(\mathbf{V})$, for $m \in [1..n]$, are interpreted as sets of elements of s'^I_{m-1}, i.e. choices of order m and sets of order $m-1$ if $m > 1$ or individuals if $m = 0$. All denoting $t^{mr} \in T^{mr}(\mathbf{V})$ for $m \in [0..n]$ and all denoting $t'^0 \in T'^0(\mathbf{V}) \setminus (T^0(\mathbf{V}) \cup_{m \in [0..n]} T^{mr}(\mathbf{V}))$ are interpreted as terms, i.e. finite, symbolic names. Finally, those denoting terms in $T'^m(\mathbf{V}) \setminus T^m(\mathbf{V})$ for $m \in [1..n-1]$ are interpreted as terms representing choices of order $m+1$.

As in Theorem 2.3.11, we get that $s'^I \subseteq s^I$, if $s' \leq_{\mathbf{S}} s$, as well as the totality of ref^I and $deref^I$ on s_m^I and s_{mr}^I, respectively, for $m \in [0..n]$. Only the references to Lemmas 2.3.10 and 2.3.9 have to be replaced by ones to Lemmas 2.4.12 and 2.4.11, respectively.

Concerning function ranks, we know that $f^I([t_1], \ldots, [t_q]) = [f(t_1, \ldots, t_q)] \in s^I$ whenever $([t_1], \ldots, [t_q]) \in Dom(f^I)$, i.e. $(f : s''_1, \ldots, s''_q \to s'') \in ranks(f)$, $([t_1], \ldots, [t_q]) \in \times_{i \in [1..q]} s''^I_i$, $s'' \leq_{\mathbf{S}} s$ and $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ f(t_1, \ldots, t_q)$. By definition of s^I, we get $[f(t_1, \ldots, t_q)] \in s^I$.

The choice function $choose^I$ gives $choose^I([t]) = [choose(t)]$ for any $[t] \in s^I_m$ and $m \in [1..n]$. From $[t] \in s^I_m$, we know by definition of s^I_m that $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} EX\ t$ and by Lemma 2.4.12 that $t \in T^m$, i.e. $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} choose(t) \in t \wedge EX\ choose(t)$ by **Choice**, **SubstConform** and **WellDef**. Consequently, $[choose(t)] \in [t]$ by definition of $[t]$, and $[choose(t)]$ is unique, since it is a unique representative of its $\doteq$-equivalence class.

By definition, $p^I \subseteq (C^I)^q$ and Lemma 2.4.11 guarantees us that $\in$ and $\doteq$ are interpreted correctly. Remark that **Paramodulation** and **PartialReflex** guarantee that $\doteq$ is a congruence. The correct interpretation of EX follows from the definition of the sorts and $C^I = \cup_{s \in \mathbf{S}} s^I$.

In order to prove that I is an G^n-model of $\mathbf{P}$, we can proceed once more as in the corresponding Lemma for $\mathcal{OSE}$-logics (cf. Section 2.2), except that Lemma 2.4.12 and Lemma 2.4.6 has to be used in this case. □

Remains to prove completeness of deduction and initiality.

Theorem 2.4.14 *Let $\mathbf{T}_{s_m}(\mathbf{X}) \neq \emptyset$ for all $m \in [0..n]$, $(\Sigma, \mathbf{P})$ be a G^n-presentation, and L a $(\Sigma, \mathbf{X})$-atom with $(\Sigma, \mathbf{P}, \mathbf{X}) \models_{G^n} L$. Then $(\Sigma, \mathbf{P}, \mathbf{X}) \vdash_{G^n} L$.*

Proof: As for R^n-logics, the proof is completely analogous to the corresponding theorem in Section 2.2. Just replace $\mathbf{T}_{\Sigma,\mathbf{P}}(\mathbf{X})$ by $\mathbf{I}_{G,X}$ and use Lemma 2.4.13 instead of the corresponding one for OSE-logics. □

Theorem 2.4.15 *Let $\mathbf{T}_{s_m}(\mathbf{X}) \neq \emptyset$ for all $m \in [0..n]$ and $(\Sigma, \mathbf{P})$ be a G^n-presentation. For all G^n-models $\mathbf{A}$ of $\mathbf{P}$, there is a unique G^n-Σ-homomorphism $h : \mathbf{I}_G \to \mathbf{A}$.*

Proof: The proof is the literal translation of the one of Theorem 2.3.13, where the references to Lemmas 2.3.9 and 2.3.10 have to be replaced by ones to Lemmas 2.4.11 and 2.4.12, respectively, except for the part concerning $h(s^{\mathbf{I}_G}) \subseteq s^{\mathbf{A}}$ for $s \in \mathbf{S}$. There, the new definitions for function interpretations result in slight changes. We give therefore this part explicitly in the following:

Let I stand for $\mathbf{I}_G$ in the sequel. Suppose $[t] \in s^I$ for some $s \in \mathbf{S}$. If $t \in \mathbf{T}_s$, then $[t]$ is mapped to an element in $s^{\mathbf{A}}$, which can be shown by structural induction on t: If t is a constant c in s, then $\mathbf{A} \models_{G^n} EX\ c$, because of the definition of $[c]$ and the soundness of **GNL**. Hence, $c^{\mathbf{A}} \in C^I$ and $\mathbf{A}$ has to map c to some element of $s^{\mathbf{A}}$, because of $c \in \mathbf{T}_s$ and hence $(c : \to s'') \in ranks(c)$ for some $s'' \leq_{bS} s$, i.e. $h([c]) = c^{\mathbf{A}} \in s^{\mathbf{A}}$. Else, t must be of the form $f(t_1, \ldots, t_q)$, $\mathbf{A} \models_{G^n} EX\ f(t_1, \ldots, t_q)$ and $h([t_i]) = t_i^{\mathbf{A}} \in s''^{\mathbf{A}}_i$ for all $i \in [1..q]$, whenever $t_i \in \mathbf{T}_{s''_i}$. Hence, $h([f(t_1, \ldots, t_q)]) = f^{\mathbf{A}}(t_1^{\mathbf{A}}, \ldots, t_n^{\mathbf{A}}) \in s^{\mathbf{A}}$, since t is in $\mathbf{T}_s$ and $\mathbf{A}$ has to respect function ranks.

Otherwise, if t is not in some $\mathbf{T}_s$, then by Lemma 2.4.12 there must be some $[t'] = [t]$ with $t' \in \mathbf{T}_s$. Hence, $h([t]) \in s^{\mathbf{A}}$. This implies also immediately $h(C^I) \subseteq C^{\mathbf{A}}$. □

One may have the impression that G^n-logics are a restricted version of R^n-logics. Quite on the contrary, we come to the following result:

Theorem 2.4.16 *Let Σ be an R^n-logical signature and $\mathbf{P}$ be a Σ-R^n-presentation. Then, there is a G^n-signature Σ' and a Σ'-G^n-presentation $\mathbf{P}'$, s.t. $\mathbf{P} \models_{R^n} \phi$ iff $\mathbf{P}' \models_{G^n} \phi'$ for all Σ-atoms ϕ, where ϕ' is ϕ with $=$ replaced by $\doteq$.*

Proof: Just add $EX\ f(x_1, \ldots, x_q)$ to $\mathbf{P}$ in order to obtain $\mathbf{P}'$, for all $(f : s''_1, \ldots, s''_n \to s) \in ranks(f)$, $f \in \mathbf{F}$, s.t. $x_i \in \mathbf{X}_{s''_i}$ for $i \in [1..q]$. □

4.4 Discussion

In order to illustrate G^n-logics we give a simple axiomatisation of the restriction of the binary logarithm function `ln` to natural numbers, i.e. the function is defined whenever both the argument and the result are natural numbers. Clearly, this is a partial function, since `ln` is not defined on zero nor on all numbers that are not a power of two. In order to simplify the task, we represent numbers in binary format. Assume therefore the natural number $\sum_{i=0}^{i=n} 2^{b_i}$ be represented by $b_0.b_1.\ \ldots\ .b_n$, where b_i is either `L` for one or `0` for zero and the dot operator is just for concatenation, i.e.. associative as usual (cf. last axiom). We omit parentheses around binary numbers therefore for simplicity. Assuming `s` to stand for the successor function on binary numbers, we get:

`s(0)` $\doteq$ `L`	`ln(0.b)` $\doteq$ `s(n)`	$\Leftarrow$	`ln(b)` $\doteq$ `n`
`s(0.b)` $\doteq$ `L.b`	`ln(L.b)` $\doteq$ `0`	$\Leftarrow$	`b` $\in$ `zeros`
`s(L)` $\doteq$ `0.L`	`ln(L)` $\doteq$ `0`		
`s(L.b)` $\doteq$ `0.s(b)`	`b.b'` $\in$ `zeros`	$\Leftarrow$	`b` $\in$ `zeros` $\wedge$ `b'` $\in$ `zeros`
`(b1.b2).b3` $\doteq$ `b1.(b2.b3)`	`0` $\in$ `zeros`		

The set `zeros` contains only sequences of zeros. Note the difference between saying:

$$\texttt{ln(0.b)} \doteq \texttt{s(n)} \Leftarrow \texttt{ln(b)} \doteq \texttt{n}$$

and saying:

$$\texttt{ln(0.b)} \doteq \texttt{s(ln(b))}$$

The latter axiom says that `ln` is defined on all binary numbers starting with `0`, implying furthermore that `ln(0.L.0.L)` is equal to `s(ln(L.0.L))`, which is a non-sense theorem, because none of the two terms are defined. Remark that the former axiom circumvents this trap by the semantics of Horn clauses, allowing for no consequence if the premise is not true in all models. Hence, `ln(0.L.0.L)` is not equal to anything, since `ln(L.0.L)` is neither, because `0.L` $\notin$ `zeros`, since `L` $\notin$ `zeros`. The latter also illutrates a way to compute effectively with this specification.

4.5 Conclusion from R^n- and G^n-logics

We developed R^n-logics as simple derivative from $\mathcal{OSE}$-logics and G^n-logics as generalization from R^n-logics. Examples show in our opinion, that both logics offer a large amount of expressiveness, while preserving set interpretations. However, the type theory used, although it is already a liberalized version of the rather strict simple theory of types, might be weakened in that sets of clauses may be given without explicit type assignment for variables as in [Whitehead et Russell, 1925]. If furthermore all arguments of a function and its result are restricted to be of the same type, then types might be reconstructible up to a certain point and for the remaining freedom of type assignments for variables, one might consider clauses as schematizations. Further extensions might concern the sets considered: We only used sets of finite order. Further expressiveness may be gained by using limit ordinals.

However, all these extensions may not be necessary due to the reference/dereference mechanism added to our logics. This mechanism offers a simple way of dealing with sets/functions as symbolic objects without using their particular properties. So a first-order function f, seen as set of pairs of individuals, is a constant `f` that can be passed to another function g. But this function g, seen as set of pairs, does not become a set of sets of pairs of individuals: it still remains only a set of such pairs. Consequently, g is a first-order function and not second-order

as one might think. People being used to program with imperative languages might recognize function pointers behind these references.

A question arising naturally in this context is: What is the difference between using function references and functions as arguments, as e.g. in the λ-calculus. The answer is rather simple: We may take function graphs as arguments and return such graphs as result. Hence, there is no real difference. This idea in mind, we showed in [Hintermeier *et al.*, 1995b] that it is possible to code the untyped λ-calculus in a set theoretic setting without problems – a subject which might be the basis for further investigations and comparisons of our framework with other λ-calculi.

Chapitre 3

Term Rewriting

1 Introduction

Introduction

In this chapter, we give a brief introduction to term rewriting based on universal and order-sorted algebra. This gives us the basic notions and techniques used in the sequel of this thesis. We start with a brief development of term rewriting out of general algebra, followed by some remarks on the expressiveness of term rewriting and an overview over this chapter.

From general algebra to Term Rewriting. The development of general algebra goes back to G. Birkhoff [Birkhoff, 1935] and its origins can be traced back to [Whitehead, 1898]. The only relation symbol used in general algebra is equality and the inference rules are the well-known reflexivity, symmetry, transitivity, stability by context and substitution. All equations are furthermore universally closed. Hence, deduction for general algebra is relatively simple.

However, doing mechanic deduction using these axioms results in an inefficient procedure. If one wants to know if two terms t and t' are equal, then one usually knows intuitively in which directions equations are to be used. Hence, symmetry can be recognised as the main culprit for the observed inefficiency and attaching a direction for each equation seems to be the obvious solution. But, orienting equations alone may result in incompleteness of deduction. Remember Example 1.2.3. Hence, more sophisticated notions have to be introduced to describe such situations and sufficient tests are to be introduced to detect incompleteness of deduction.

Deduction using equations in one direction only, replacing subterms that are an instantiation of a left hand side of an oriented equation by the instantiated right hand side is called term rewriting, the corresponding relation generated by oriented equations through stability by context and substitution is called term rewriting relation. Hence, the basic step of deduction is now a term rewriting step inside a member of an equation, which is a macro combination of transitivity, stability by context and substitution. Soundness is therefore guaranteed. Completeness for term rewriting means that two terms are equal in all models[6] of the equational theory if they can be rewritten to the same term. A theorem stating soundness and completeness of term rewriting using equations in any direction is called Birkhoff Theorem in this thesis, in analogy to the completeness results obtained for general algebra in [Birkhoff, 1935].

However, using equations only in one direction, it is not obvious that completeness can be

[6] Remark that the model theory of general algebra the main origin of ADTs as discussed in the introduction to Chapter 2.

obtained, as already mentioned. This is only guaranteed if the term rewriting relation generated by the oriented equations through closure by substitutivity and context is confluent, i.e. if a term can be rewritten (maybe in several steps) in two different ways, then the two obtained terms can be joined again by term rewriting. Confluence is also called the Church-Rosser property. There are a lot more abstract properties for term rewriting relations (cf. e.g. [Huet, 1980]).

Still, even with confluence, the equational theory may remain undecidable. The most well-known example for this may be the untyped λ-calculus, but also Turing machines can be specified in a confluent way, even with only one rule [Dauchet, 1989]. Hence, other restrictions have to be imposed to make the equational theory decidable. The most common solution adopted is well-foundedness of the term rewriting relation. Hence, together with confluence, we can be sure that two terms rewrite to the same irreducible, i.e. no more rewritable, term if and only if they are equal in all models of the equational theory. Consequently, confluent and terminating rewrite relations give a decision procedure for equality in this theory. The unique irreducible term in an equivalence class is therefore called normal forms. A more instructive and complete overview over term rewriting theory can be found in [Dershowitz et Jouannaud, 1990a].

Expressiveness of term rewriting. As already mentioned, term rewriting may be used to simulate any Turing machine and hence this computation technique is Turing complete. For the inverse direction people use typically Church's thesis saying that everything that is intuitively computable may be computed by a λ-term, which is provably equivalent to computation with Turing machines. Although it seems clear that term rewriting as described above is embedded in rewriting of λ-terms, we think that this indirect 'proof' is not very instructive and does not teach us very much about the untuitive expressiveness of term rewriting systems.

One way to evaluate the expressiveness of term rewriting relations is to classify the algebras that can be described by finite and denumerably infinite sets of unconditional and conditional equations, using finite and infinite sets of terms, allowing for auxiliary operations or not, respectively. This is the approach taken by Bergstra and Tucker in [Bergstra et Tucker, 1987]. This is relevant for the task of specification.

In this chapter, we take the point of view of computing and try to investigate the intuitive expressiveness of term rewriting relation in the presence of confluence and termination, i.e. computability of irreducible terms. To do this, we try to express the innermost strategy of evaluation into typing restrictions and to transform constructively extensions of the unconditional term rewriting relation by conditions including some cases of extra-variables back into the unconditional relation, s.t. confluence and termination are preserved. The structure and readability of these transformation are investigated and lead to an intuitive judgement for their expressiveness. In particular, one may observe that the use of ordered sorts simplifies this task. As a by-product, we argue that these transformations can serve in the future to promote the reuse of proofs concerning unconditional term rewriting for proofs concerning innermost or conditional term rewriting.

Overview. We start in Section 3.2 with notions from term rewriting theory. First, we briefly introduce the basic notions concerning term rewriting theory. We start with unsorted matching, unification and term rewriting systems (TRSs), including conditional term rewriting systems (CTRSs), and go then over to order-sorted TRSs and finally strongly deterministic TRSs from [Avenhaus et Loría-Sáenz, 1994], for short SDTRSs, a class of CTRSs with extra-variables, which is particularly interesting for a reduction of typed equational specifications to conditional ones. In Section 3.3, we discuss the formal expressivity of term rewriting systems.

For SDTRSs we suggest a constructive, proof theoretic transformation into unconditional TRSs in case of ground confluence, as extension of our result for decreasing, ground-confluent CTRSs without extra-variables in [Hintermeier, 1994], from which we show a part allowing to specify the innermost strategy for TRSs.

2 Term Rewriting Relations and Properties

2.1 Unsorted Terms and Rewriting

Terms and Positions

Notations concerning classical *terms, occurrences, replacements, substitutions* and generality on terms and substitutions are consistent with [Dershowitz et Jouannaud, 1990b; Smolka *et al.*, 1989]. In particular we write $|t|$ for the number of function symbols occurring in a term t, $subterm\_set(t)$ for all subterms of t, $\mathcal{VO}cc(t)$ for the set of all variable occurrences and $\mathcal{NVO}cc(t)$ for the non-variable ones in t. Let $g, d, l, r, s, t, u, g_1, d_1, l_1, r_1, s_1, t_1, u_1, \ldots$ stand for terms in the following. All notions concerning *unsorted matching, unification and term rewriting* are consistent with those in [Jouannaud et Kirchner, 1991]. Notions on *completion, equational proofs and proof reductions* are consistent with [Bachmair, 1991].

The notion of *replacement* will also be used in an extended way for sets of *incomparable occurrences* and *multiple replacements*. Let O, O' be such sets. Therefore, $t[u]_O$ stands for t with u at all occurrences $\omega \in O$ and $t[u]_O[v]_{O'}$ for $(t[u]_O)[v]_{O'}$. Furthermore, the occurrence orderings extend pairwise to such occurrence sets, i.e. O is incomparable with O', written $O \asymp O'$, if all occurrences in O are incomparable with all occurrences in O'. O is smaller than O', written $O \leq O'$, if $\forall \omega \in O, \exists \omega' \in O', \omega \leq \omega'$, etc. Remark that in general, two arbitrary given O, O' might be neither incomparable nor greater/smaller. However, this is guaranteed for singleton occurrence sets. Additionally, the concatenation of two occurrence sets O and O' denotes the set of occurrences $\{\omega.\omega' \mid \omega \in O \text{ and } \omega' \in O'\}$. Finally, if $t_{|\omega} = t_{|\omega'}$ for all $\omega, \omega' \in O$, then $t_{|O}$ stands for $t_{|\omega}$, $\omega \in O$.

Let $Var(.)$ be the function returning the set of variables occuring in the syntactic object(s), such as terms, formulas etc., given as argument(s). For a substitution σ, the domain is defined as $\mathcal{D}om(\sigma) = \{x \mid \sigma(x) \neq x\}$ (remark that this is not as usual function domains), $\mathcal{R}an(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} Var(\sigma(x))$ is the set of variables in the image of the variables in the domain of the substitution, and $\mathcal{I}m(\sigma)$ stands for the set $\{t \mid \exists x \in \mathcal{D}om(\sigma) : \sigma(x) = t\}$. Moreover, $terms(\rho)$ stands for the terms occurring in an arbitrary structure ρ and $subterm\_set(\rho)$ is $terms(\rho)$ together with all subterms of terms in $terms(\rho)$. We briefly recall the most basic definitions - for simplicity we omit the used signature Σ, which is usually given as prefix.

Matching and Unification

A *matching problem* is a finite conjunction $\mathcal{M}$ of matching equations $\bigwedge_{i \in [1..n]} s_i \leq^? t_i$. If $(\bigcup_{j \in [1..n]} Var(s_j)) \cap (\bigcup_{j \in [1..n]} Var(t_j)) = \emptyset$, then $\mathcal{M}$ is called *variable disjoint*. A *matcher* of $\mathcal{M}$ is a substitution σ with $\mathcal{D}om(\sigma) \subseteq \bigcup_{i \in [1..n]} Var(s_i)$, s.t. for all $i \in [1..n]$, $\sigma(s_i) = t_i$. Such syntactic matchers are unique.

Analogously, a unification problem $\mathcal{U}$ is a conjunction of matching equations $\bigwedge_{i \in [1..n]} s_i =^? t_i$. A *unifier* of $\mathcal{U}$ is an idempotent substitution σ, s.t. for all $i \in [1..n]$, $\sigma(s_i) = \sigma(t_i)$. Unifiers are ordered by the substitution subsumption ordering $\leq$ which takes a set of variables V as parameter, written $\leq^V$. A unifier σ is *more general* than another unifier τ on V if there is substitution ρ, s.t. $\rho(\sigma(x)) = \tau(x)$ for all $x \in V$. For the above defined syntactic unifiers,

Delete	$\mathcal{M} \wedge t \leq^? t$
	$\Longrightarrow \mathcal{M}$
Decompose	$\mathcal{M} \wedge f(s_1, \ldots, s_n) \leq^? f(t_1, \ldots, t_n)$
	$\Longrightarrow \mathcal{M} \wedge s_1 \leq^? t_1 \wedge \ldots \wedge s_n \leq^? t_n$
Conflict	$\mathcal{M} \wedge f(s_1, \ldots, s_n) \leq^? g(t_1, \ldots, t_m)$
	$\Longrightarrow \mathbb{F}$
	if $f \neq g$
Merging	$\mathcal{M} \wedge x \leq^? t_1 \wedge x \leq^? t_2$
	$\Longrightarrow \mathcal{M} \wedge x \leq^? t_1$
	if $x \in \mathcal{X}$ and $t_1 = t_2$
MergingClash	$\mathcal{M} \wedge x \leq^? t_1 \wedge x \leq^? t_2$
	$\Longrightarrow \mathbb{F}$
	if $x \in \mathcal{X}$ and $t_1 \neq t_2$
SymbolClash	$\mathcal{M} \wedge f(t_1, \ldots, t_n) \leq^? x$
	$\Longrightarrow \mathbb{F}$
	if $x \in \mathcal{X}$

Figure 3.1: **MATCH** : Matching using Transformation Rules

there is a most general one w.r.t. $\leq^{\mathcal{V}ar(\mathcal{U})}$ up to the equality induced by this ordering, which is essentially variable renaming. [Jouannaud et Kirchner, 1990; Siekmann, 1989] are surveys on unification containing also syntactic unification algorithms.

The matching and unification procedures presented in this thesis are described as set of *transformation rules* working over terms representing the problem, in the style of [Jouannaud et Kirchner, 1990]. A transformation rule is *sound* if the set of solutions of the resulting problem is subsumed by the one of the initial problem. If the set of solutions of the resulting problem subsumes the solutions of the initial problem, then the transformation rule is *complete*. The transformation rules have the following form:

$$t \Longrightarrow t' \text{ if } condition$$

where t and t' are terms representing problems and *condition* is a boolean expression that controls application of the rule, namely the rule is applicable to a problem s if and only if there is a substitution σ with $\sigma(t) = s$ and $\sigma(condition)$ is satisfied. Such a procedure tries to apply its rules deterministically, i.e. without backtracking, to the current problem as long as possible, replacing the current problem by $\sigma(t')$. Consequently, the procedure terminates if no more rule is applicable and in this case we call the procedure an *algorithm*. Figure 3.1 shows a syntactic matching algorithm in this style.

Unsorted Term Rewriting

An (unconditional) *term rewrite rule* has the form $l \to r$, where l and r are terms with $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(r)$. A *term rewrite system* $\mathcal{R}$ is a set of such rewrite rules. A term t *rewrites* to a term t' in $\mathcal{R}$, if there is a rule $(l \to r)$ in $\mathcal{R}$ and an occurrence $\omega \in \mathcal{O}cc(t)$, s.t. there is a matcher σ of $l \leq^? t|_\omega$ and t' is of the form $t[\sigma(r)]_\omega$. This is written $t \rightarrowtail_{\mathcal{R}}^{l \to r, \omega, \sigma} t'$, where $l \to r$, ω and σ may occur in any order since they can be distinguished by their syntax.

A binary relation R on terms is a *reduction relation* if R is stable by context and substitution, i.e. tRt' implies $\sigma(t)R\sigma(t')$ and $f(\ldots t\ldots)Rf(\ldots t'\ldots)$ for any substitution σ and any function symbol f in the current signature Σ. $\bigcup_{n\geq 0} R^n$, where R^0 is the identity, is written R^* and if R is well-founded, i.e. there is a well-founded ordering $\succ$ on terms containing R, then $R^!$ is defined as $\{(t,t') \in R^* \mid t' \text{ is minimal w.r.t. } \succ\}$. A reduction relation R is *locally (ground) confluent* if for all (ground) terms t, $t_1R^{-1}tRt_2$ implies (t_1, t_2 are ground and) $t_1R^*(R^{-1})^*t_2$. R is (ground) confluent if for all (ground) terms t, $t_1(R^{-1})^*tR^*t_2$ implies (t_1, t_2 are ground and) $t_1R^*(R^{-1})^*t_2$. The *Diamond Lemma* ensures that all terminating, locally confluent (ground) reduction relations are also (ground) confluent.

A term rewrite system is *orthogonal* if the left hand-sides are linear and whenever they overlap, then this is on top and the corresponding instantiated right hand sides are identical. These systems have the nice property to be always confluent, even without termination.

2.2 Order-Sorted Rewriting with Syntactic Sorts

For order-sorted algebras (OS-algebras or OSA for short) one may distinguish between overloaded and non-overloaded semantics [Goguen et Meseguer, 1992; Smolka *et al.*, 1989; Waldmann, 1989a]. We opt for *non-overloaded semantics* in the following. This is in fact a special case of $\mathcal{OSE}$-logics as developed in Section 2.2: an order-sorted algebra with non-overloaded semantics is an $\mathcal{OSE}$-presentation with only the equality symbol as relation.

Now, matchers and unifiers are defined in exactly the same way as in the unsorted case, except that they are only defined on matching/unification equations over well-formed terms. Matchers are still unique for any matching problem, but for the existence of a most general unifier for a set of unification equations over terms in $\mathcal{T}(\Sigma,\mathcal{X})$, we need additional restrictions, as the following example shows:

Example 3.2.1 *Let $\Sigma = (\{f,a\},\{A,B\})$ with ranks $f : A \to A$, $f : B \to B$, $a : \to A$ and $a : \to B$, the quasi-ordering $\leq$ being the empty relation. Let furthermore $x \in \mathcal{X}_A$ and $y \in \mathcal{X}_B$. Then $f(x)$ and $f(y)$ are well-formed and the unification equation $f(x) =^? f(y)$ has an infinite set of incomparable solutions of the form $\{x \mapsto f^n(a), y \mapsto f^n(a)\}$.*

A sufficient condition for the existence of a most general unifier for any unification problem is *regularity*, saying that if a term t is well-formed of sort s_1 and also well-formed of sort s_2, then there is a unique, minimal (w.r.t. $\leq$) sort $s \in \mathcal{S}$ with $s \leq s_1, s_2$, s.t. t is also well-formed of sort s. A finite set of unifiers may be guaranteed if unicity and minimality of s are given up. Another approach is to generalise this problem completely to unification in second-order monadic predicate logic [Comon, 1991; Comon, 1992], where tree automata can be used to decide intersection of sorts, which is hidden behind the unification of sorted variables. The classical definition of a term rewriting relation in order-sorted algebras is restricted to terms in $\mathcal{T}(\Sigma,\mathcal{X})$:

An (unconditional) *OS-term rewrite rule* has the form $l \to r$, where l and r are terms in $\mathcal{T}(\Sigma,\mathcal{X})$ with $Var(r) \subseteq Var(r)$. It is called *sort-decreasing* if for all Σ-substitutions σ, $\sigma(r)$ is well-formed of sort s whenever $\sigma(l)$ is. A *rewrite system* $\mathcal{R}$ is a set of such rewrite rules. A term $t \in \mathcal{T}(\Sigma,\mathcal{X})$ *rewrites* to a term t' in $\mathcal{R}$, if there is a rule $(l \to r)$ in $\mathcal{R}$ and an occurrence $\omega \in \mathcal{O}cc(t)$, s.t. there is a matcher σ of $l \leq^? t|_\omega$ and t' is of the form $t[\sigma(r)]_\omega$. This is written $t \xrightarrow[\mathcal{R}]{l\to r,\omega,\sigma} t'$, where as before $l \to r$, ω and σ may occur in any order since they can be distinguished by their syntax. Remark that $t \in \mathcal{T}(\Sigma,\mathcal{X})$ and $t \xrightarrow[\mathcal{R}]{l\to r,\omega,\sigma} t'$ do not imply $t' \in \mathcal{T}(\Sigma,\mathcal{X})$:

Example 3.2.2 *Let $\Sigma = (\{f,a,b\},\{A,B\})$ with $A \leq B$ and ranks $f : A \to A$, $f : B \to B$,*

$a : \rightarrow A$ and $b : \rightarrow B$. Let furthermore $\mathcal{R}$ be the following system of OS-term rewriting rules:

$$\left\{ \begin{array}{l} a \rightarrow b \\ f(x : A) \rightarrow x \end{array} \right\}$$

Then $f(a) \rightarrowtail^{a \rightarrow b, 1, \{x \rightarrow a\}} f(b)$, where $f(a)$ is well-formed of sort A, but $f(b)$ is not well-formed of any sort.

However, when all rules in $\mathcal{R}$ are sort-decreasing, which is decidable, then $\mathcal{T}(\Sigma, \mathcal{X})$ is closed under rewriting in $\mathcal{R}$ [Kirchner *et al.*, 1988b].

2.3 Conditional Term Rewriting

In general, *conditional rewrite rule* is a Horn clause of the shape $l \rightarrow r \Leftarrow \Gamma$, where $l \rightarrow r$ is an oriented equation and Γ is a conjunction of equations, called the condition. *Normal conditional rewrite rules* (cf. [Dershowitz et Okada, 1990]), are of the form:

$$g \rightarrow d \Leftarrow \bigwedge_{i \in I} s_i \rightarrow^* s'_i,$$

s.t. $|I| < \infty$ and $\bigcup_{i \in I} \mathcal{V}ar(s_i) \cup \mathcal{V}ar(d) \subseteq \mathcal{V}ar(g)$. The rewrite relation is the smallest relation $\rightarrow_{\mathcal{R}}$, s.t. $t \rightarrow_{\mathcal{R}} t'$ iff $\exists (g \rightarrow d \Leftarrow \bigwedge_{i \in I} s_i \rightarrow^* s'_i) \in \mathcal{R}, \omega \in Occ(t)$, a substitution σ, s.t. $\sigma(g) = t_{|\omega}$, $t' = t[\sigma(d)]_\omega$ and $\forall i \in I : \sigma(s_i) \stackrel{*}{\rightarrow}_{\mathcal{R}} \sigma(s'_i)$.

As we see from this definition, a conditional term rewriting rule $(l \rightarrow r) \Leftarrow \Gamma$ is often restricted to the case where all variables in the right-hand side r and the condition Γ are also in l. Weakening this restriction leads us to the following classification of CTRSs from [Middeldorp et Hamoen, 1994]:

Type	Requirement
1	$\mathcal{V}ar(r) \cup \mathcal{V}ar(\Gamma) \subseteq \mathcal{V}ar(l)$
2	$\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$
3	$\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(\Gamma)$
4	no restrictions

We may recognise normal CTRSs as a special case of 1-CTRSs, in which all variables occuring in the clause are predetermined by a matcher for the left-hand side. 2-CTRSs are Horn clauses with rewrite rules as head, 3-CTRSs are a generalisation where computation of the right hand side's instantiation may depend on intermediate computations in the condition and 4-CTRSs are actually Horn clauses.

In the following, we will show interest in a particular case of type 3 CTRS, investigated in more detail in [Avenhaus et Loría-Sáenz, 1994], where the instantiations of variables occuring in the condition but not in the left-hand side can all be determined step-by-step. These systems were already used by H. Ganzinger [Ganzinger, 1989; Ganzinger, 1991b] in order to realize the completion of order-sorted term-rewriting systems including non-sort-decreasing rules by many-sorted conditional completion. The idea there was to perform the evaluation of the right-hand side of a rule that is not sort-decreasing as part of the computations in the condition. A variable with the sort of the left-hand side was then tried to be instantiated by result of this evaluation. If the normal form r' of instantiated right-hand side was of lower or equal sort w.r.t. the left hand side, then the result of the rule application was r', otherwise the condition was not satisfied and

thus the rule was not applicable. Hence, rules were only applied if they were sort-decreasing. Let us illustrate this principle:

Example 3.2.3 *Recall the definition of* `square` *and* `times` *of Example 1.1.4. Clearly, the equation* `square(x) = times(x,x)` *is not sort-decreasing if oriented left-to-right. The solution suggested in [Ganzinger, 1991b] is to replace the rule by:*

$$(\mathtt{square(x)} \to \mathtt{z}) \Leftarrow \mathtt{times(x,x)} \to^{!} \mathtt{r_{Int-Nat}(z)},$$

where $\mathtt{r_{Int-Nat}}$ *is the injection function of* `Nat` *into* `Int` *and* `z` *is of sort* `Nat`. *Remark that both equations are now sort-preserving if we discard the subsort-ordering.*

We think that the technique is interesting, but in practice the injection functions disturb the reading of the equations. A better way would be the consideration of order-sorted conditional rewriting. One more thing to mention is the difference to (quasi-)reductive CTRSs [Ganzinger, 1991b; Jouannaud et Waldmann, 1986], since these variables are in general not predetermined and therefore existentially quantified. This would, in general, force us to use narrowing in order to test satisfiability of a condition. A sufficient restriction that allows us to circumvent narrowing is the use of strongly deterministic [Avenhaus et Loría-Sáenz, 1994] conditions, where one side is by definition irreducible.

2.4 Strongly Deterministic Term Rewriting Systems

In this section, we present a class of type 3 CTRSs which is particularly interesting for us because it may occur as the result of a program construction using the techniques described in the sequel of this thesis. The intuition for the definition of quasi-reductive, deterministic term rewriting systems (DTRS) below from [Ganzinger, 1991b; Avenhaus et Loría-Sáenz, 1994] is that equations in conditions may be evaluated by reducing one side only and finally matching the normal form of this side with the other side, which gives the instantiation for all extra-variables in the latter side. However, there is still some non-determinism left that avoids a critical pair lemma, due to reducibility of the non-reduced side. Remark that in general DTRSs variable overlaps and top self-overlaps of rewrite rules may lead to divergence. This is the motivation for restricting to strongly deterministic DTRSs.

Actually, we considered a slight weakening of the definitions restricting the orientation of equations in the conditions to those equations defining extra variables.

Definition 3.2.4 *An* unconditional oriented equation *is an oriented pair of terms, written* $l \stackrel{*}{\to} r$. *An* partially oriented condition *is a finite conjunction of unconditional equations, possibly oriented, of the form* $u_1 \stackrel{*}{\to} u'_1 \wedge \ldots \wedge u_m \stackrel{*}{\to} u'_m \wedge u_{m+1} = u'_{m+1} \wedge \ldots, u_{m'} = u'_{m'}$. *It is* operational *for a set* V_0 *of variables if:*

$$\begin{array}{l}\text{For } i \in [1..m], \mathcal{V}ar(u_i) \subseteq V_0 \cup \bigcup_{j\in[1..i-1]} \mathcal{V}ar(u_j, u'_j) \\ \text{and } \bigcup_{i\in[m+1..m']} \mathcal{V}ar(u_i, u'_i) \subseteq V_0 \cup \bigcup_{i\in[1..m]} \mathcal{V}ar(u_i, u'_i).\end{array}$$

A deterministic rule *is a clause* $(l \to r) \Leftarrow \Gamma$, *s.t.* Γ *is an operational partially oriented condition for* $\mathcal{V}ar(l)$ *and* $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \bigcup_{i\in[1..m]} \mathcal{V}ar(u_i, u'_i)$ *and* l *is not a variable. The set of* extra variables *of this rule is* $\mathcal{EV}ar((l \to r) \Leftarrow \Gamma) = \bigcup_{i\in[1..m]} \mathcal{V}ar(u_i, u'_i) \setminus \mathcal{V}ar(l)$. *A* deterministic term rewrite system *(DTRS) is a finite set* R *of deterministic rules.*

A DTRS R *is* quasi-reductive *(w.r.t. a reduction ordering* $\succ$*) if for every substitution* σ *and every rule* $(l \to r) \Leftarrow \bigwedge_{i\in[1..m]}(u_i \stackrel{*}{\to} u'_i) \wedge \bigwedge_{i\in[m+1..m']}(u_i = u'_i)$ *in* R:

1. $\sigma(u_j) \succeq \sigma(u'_j)$ *for* $1 \leq j \leq i$ *implies* $\sigma(l) \succ \sigma(u_{i+1})$

2. $\sigma(u_j) \succeq \sigma(u'_j)$ *for* $1 \leq j \leq m$ *implies* $\sigma(l) \succ \sigma(r), \sigma(u_i), \sigma(u'_i)$ *for* $i \in [m+1..m']$.

A term v *is* strongly irreducible *if* $\sigma(v)$ *is irreducible for every irreducible substitution* σ. *R is called* strongly deterministic *if for every rule* $(l \to r) \Leftarrow \bigwedge_{i\in[1..m]}(u_i \xrightarrow{*} u'_i) \wedge \bigwedge_{i\in[m+1..m']}(u_i = u'_i)$ *in R, each* u'_i *for* $i \in [1..m]$ *is strongly irreducible w.r.t. R. A DTRS R is called* strongly deterministic term rewrite system *(SDTRS), if all its rules are strongly deterministic.*

Clearly, strong determinism is undecidable in general. However, there are sufficient conditions that are easily decidable.

Definition 3.2.5 *Let R be a DTRS. A term u is called* absolutely irreducible *w.r.t. R if for all non-variable occurrences p of u and each rule $(l \to r) \Leftarrow \Gamma$, l and $u_{|_p}$ do not overlap. R is* absolutely deterministic *if for every rule $(l \to r) \Leftarrow \bigwedge_{i\in[1..m]}(u_i \xrightarrow{*} u'_i) \wedge \bigwedge_{i\in[m+1..m']}(u_i = u'_i)$ in R each u'_i for $i \in [1..m]$ is absolutely irreducible w.r.t. R.*

Absolute determinism implies strong determinism [Avenhaus et Loría-Sáenz, 1994]. Now, we can define a term rewriting relation giving an operational semantics for quasi-reductive SDTRS:

Definition 3.2.6 *Let the set P of decorated clauses be a quasi-reductive SDTRS, t be a Σ-ground term and $C = ((l \to r) \Leftarrow \bigwedge_{i\in[1..m]})(u_i \xrightarrow{*} u'_i) \wedge \bigwedge_{i\in[m+1..m']}(u_i = u'_i))$ be a rule from P.*

Then, t rewrites *to t' by C at position $p \in \mathcal{O}cc(t)$, written $t \rightarrowtail_P^{p,\sigma,C} t'$, if σ is a matcher of $t_{|_p}$ against l, s.t.:*

- $t' = t[\sigma_{m+1}(r)]_p$, *where*
- $\sigma_1 = \sigma$,
- *for* $i \in [1..m]$, $\sigma_i(u_i) \rightarrowtail_P^* u''_i$ *and if tau is a matcher of* $u''_i \geq^? \sigma_i(u'_i)$, *then* σ_{i+1} *is* $\tau \circ \sigma_i$, *and*
- *for* $i \in [m+1..m']$, $\sigma_{m+1}(u_i) \rightarrowtail_P^* . \leftarrowtail_P^* \sigma_{m+1}(u'_i)$.

Let us finish this section with an illustrating example for a strongly deterministic term rewriting system realising quick-sort (cf. [Avenhaus et Loría-Sáenz, 1994]):

Example 3.2.7 *Let tt be a constant for truth, ff for falsehood, natural numbers be constructed via s (successor) and 0 (zero), as usual, lists of natural numbers be constructed via nil and cons and pair be a constructor for pairs of lists of natural numbers. Then, a naive untyped version of quick-sort for lists of natural numbers can be formulated through SDTRSs as follows:*

$$
\begin{array}{lll}
qsort(nil) & \rightarrow nil & \\
qsort(cons(x,y)) & \rightarrow z & \Leftarrow split(y,x) = pair(y_1,y_2), \\
 & & \quad qsort(y_1) = z_1, \\
 & & \quad qsort(y_2) = z_2, \\
 & & \quad append(z_1, cons(x,z_2)) = z \\
split(nil,x) & \rightarrow nil & \\
split(cons(x_1,y),x_2) & \rightarrow pair(cons(x_1,y_1),y_2) & \Leftarrow leq(x_1,x_2) = tt, \\
 & & \quad split(y,x_2) = pair(y_1,y_2) \\
split(cons(x_1,y),x_2) & \rightarrow pair(y_1,cons(x_1,y_2)) & \Leftarrow leq(x_1,x_2) = ff, \\
 & & \quad split(y,x_2) = pair(y_1,y_2) \\
append(nil,y) & \rightarrow y & \\
append(cons(x,y_1),y_2) & \rightarrow cons(x,append(y_1,y_2)) & \\
leq(0,x) & \rightarrow tt & \\
leq(s(x),0) & \rightarrow ff & \\
leq(s(x),s(y)) & \rightarrow leq(x,y) &
\end{array}
$$

3 The Formal Expressiveness of Term Rewriting

In this section we illustrate how order-sorted TRSs can be used to restrict evaluation strategies and to transform CTRSs and SDTRSs into TRSs. First, we show how the evaluation strategy for an unconditional, ground confluent, terminating term rewrite system $\mathcal{R}$ may be implicitely restricted to innermost evaluation. This is in fact a simplified version of the transformation of ground confluent, decreasing CTRSs from [Hintermeier, 1994]. Finally, we discuss the expressiveness of SDTRSs and outline how ground confluent SDTRSs may be transformed in a similar way as ground confluent, decreasing CTRSs. These transformations have two purposes: they allow us to reuse proofs from the rich unconditional term rewriting theory for the conditional theory, e.g. for completeness proofs of narrowing procedures, and they show in a constructive way that the mentioned CTRSs and SDTRSs do not give more computational power than unconditional TRSs, independently from Church's Thesis, although they allow us to express things more liberally.

3.1 Innermost by Order-Sorted Rewriting

In order to show how to restrict evaluation strategies to innermost strategies by using ordered sorts, we first copy the signature and then define matching by OS-term rewriting. The basic idea of the construction is to use sorting information to distinguish between irreducible terms and reducible terms. Irreducible terms have a label *irr* attached to each function symbol occurring in it, so that an *irr*-label on top means that the whole term is in normal form. Hence, using variables that can be instantiated by irreducible terms only allows us to restrict the evaluation of a function only if all its arguments are irreducible.

The Rules to be Transformed

Let $\mathcal{R}$ be a finite, ground-confluent and terminating TRS (for short *canonical* TRS). For each $f \in \mathcal{F}, ar(f) = n$, let $\mathcal{R}_f$ denote the set of rules defining f, i.e. all rules in $\mathcal{R}$ with f as top symbol on the left hand side of the rule. Let $\mathcal{R}_f$ be of the following form:

$$\mathcal{R}_f = \left\{ \begin{array}{ll} \phi_f^1 : & f(t_1^1, \ldots t_n^1) \rightarrow r_1, \\ & \vdots \\ \phi_f^m : & f(t_1^m, \ldots t_n^m) \rightarrow r_m \end{array} \right\}$$

Remark that the ϕ_f^i, $i \in [1..m]$ are labels used to distinguish rules. A rewrite derivation respects *lexicographic rule priority* if whenever ϕ_f^i is applied at occurrence ω of some term t in the derivation, then there is no $j < i$, s.t. ϕ_f^j could be applied at t in ω instead. In order to simplify our formulations in the following, we assume that $\mathcal{F}$ can be partitioned into sets $\mathcal{F}_n$ of function symbols of arity n. Functions with different arities are not allowed, which is not an essential restriction, since it can always be reached by renaming of function symbols.

Constructing Σ' from Σ

Let $\Sigma = \mathcal{F}$ be the unsorted signature of a term rewriting system. We show now how to construct an OS-signature Σ' with the mentioned features. First we define Σ' to be $(\mathcal{S}', \mathcal{F}')$ with $\mathcal{F}'$ containing $\mathcal{F} \cup \mathcal{F}^{irr} \cup \mathcal{F}^{vars}$, where $\mathcal{F}^{vars} = \{d_x \mid \exists (l \rightarrow r) \in \mathcal{R} \ : \ x \in Var(l, r)\}$, $\mathcal{F}^{irr}$ is a *mirror* function set defined as $\{f^{irr} \mid f \in \mathcal{F}\}$, s.t. f and f^{irr} have the same arity. Additionally, $\mathcal{F}'$ contains some operations defined later on, for which we assume that their names do not occur in $\mathcal{F} \cup \mathcal{F}^{irr} \cup \mathcal{F}^{vars}$. Let $\leq_{\mathcal{S}'}$ denote the sort ordering. $\mathcal{S}'$ contains the following ordered sorts:

$$IrrTerms \leq Terms,$$
$$Vars \leq Terms$$

Let $\mathcal{X}'$ be a set of $\mathcal{S}'$-sorted variables and $(x :: A)$ be the shorthand for 'x is of sort A'. In order to internalise the matching algorithm needed to simulate innermost rewriting, we represent any variable x contained in a rule of $\mathcal{R}$ by a constant d_x of sort $Vars$ in $\mathcal{F}'$. All functions f with $ar(()f) = n$ in $\mathcal{F}$ lead to the function declarations $f \ : \ Terms^n \rightarrow Terms$ and $f^{irr} \ : \ IrrTerms^n \rightarrow IrrTerms$. $\mathcal{T}_{Terms}(\mathcal{X}')$ be $\mathcal{T}_{IrrTerms}(\mathcal{X}')$ be a shorthand for $(\Sigma', \mathcal{X}')$-terms of the mentioned sorts.

Rule Transformation

If we want to apply the ith equation ϕ_f^i of a function f to a term $f(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ represent the current arguments of the functions, which are ground terms, we need to match $f(t_1^i, \ldots, t_n^i)$ against $f(x_1, \ldots, x_n)$. In order to mimic this with unconditional, left-linear rewrite rules, we need to compare possible variable images, since the pattern $f(t_1^i, \ldots, t_n^i)$ may be non-linear. Unfortunately, it is impossible to decide the difference of two variables x and y without negation, i.e. we need to replace them by constants d_x and d_y, respectively. Distinguishing variables is necessary to do the merging part of matching and to apply substitutions to variables, both using rewrite rules.

Furthermore, since we want a rule only to be applied if all arguments are irreducible, we need to attach at any node in the pattern the label irr. This motivates the definition of the following two term transformations, used later on to give the transformation of $\mathcal{R}_f$ for each $f \in \mathcal{F}$, $x \in \mathcal{X}'$:

$$\begin{array}{rcl|rcl} (.)^{irr} : \mathcal{T}_{Terms}(\mathcal{X}') \rightarrow \mathcal{T}_{IrrTerms}(\mathcal{X}') & & & (.)^{cl} : \mathcal{T}_{Terms}(\mathcal{X}') \rightarrow \mathcal{T}_{Terms}(\mathcal{X}') & & \\ f(t_1, \ldots, t_n)^{irr} & = & f^{irr}(t_1^{irr}, \ldots, t_n^{irr}) & f(t_1, \ldots, t_n)^{cl} & = & f^{irr}(t_1^{cl}, \ldots, t_n^{cl}) \\ x^{irr} & = & x :: IrrTerms & x^{cl} & = & d_x \end{array}$$

Let *solve* and *match* be operations representing the matching algorithm described further below and $\leq$ be a matching equation constructor. Now we can represent the matching problem, which was presented in a simplified form using *matchpb* in the introduction, as the following term:

$$\sigma'_i = match(solve(f(t_1^i, \ldots, t_n^i)^{cl} \leq f^{irr}(x_1, \ldots, x_n), nil))$$

Remember that terms labelled by irr, like those produced by cl, are irreducible. Let τ_d be the substitution replacing all variables $x \in \mathcal{V}ar(\mathcal{R})$ by the corresponding constants d_x. Now, the set of rules $\mathcal{R}_f$ can be replaced by the following rule, assuming $x_i :: IrrTerms$ for $i \in [1..n]$:

$$\begin{aligned} \phi'_f : f(x_1, \ldots, x_n) \rightarrow \quad & try\ \sigma'_1\ then\ \tau_d(r_1) \\ & else\ try \\ & \vdots \\ & else\ try\ \sigma'_m\ then\ \tau_d(r_m) \\ & else\ f^{irr}(x_1, \ldots, x_n), \end{aligned}$$

where *try-then-else* results in the instantiated right hand side if there is a match, otherwise in the fourth argument. Let $\mathcal{R}'_{\mathcal{F}}$ denote the set $\{\phi'_f \mid f \in \mathcal{F}\}$. Remark that the size of $\mathcal{R}'_{\mathcal{F}}$ is linear w.r.t. the size of $\mathcal{R}$.

The difference between computing with the original set of rules and this set of unconditional rules relies in the sorting restriction of the variables, i.e. the function f can only be evaluated if the arguments are of sort $IrrTerms$. Therefore, the sort $IrrTerms$ has to contain exactly all t^{irr} for each t in the original signature Σ. Consequently, we add for any operation symbol g with $ar(()g) = n \geq 0$ in Σ which does not occur on the top of a left hand side in $\mathcal{R}$, the following rules, where $x_1, \ldots, x_n :: IrrTerms$:

$$g(x_1, \ldots, x_n) \rightarrow g^{irr}(x_1, \ldots, x_n)$$

Auxiliary Functions

In this section, we define formally the *if_then_else* construct and some boolean valued operations for deciding syntactic equality and inequality. They are given in order to prove the existence of orthogonal rules for these functions. Let the sort *Bool* be in $\mathcal{S}'$, $x :: Bool$ and consider the following operator declarations:

$$\begin{aligned} F, T : & \rightarrow Bool \\ \wedge : Bool \times Bool & \rightarrow Bool \\ \vee : Bool \times Bool & \rightarrow Bool \\ == : Terms \times Terms & \rightarrow Bool \\ = / = : Terms \times Terms & \rightarrow Bool \\ if\_then\_else : Bool \times Terms \times Terms & \rightarrow Terms \end{aligned}$$

We start with boolean conjunction and disjunction:

$$\begin{aligned} & x \wedge T \rightarrow x,\ x \wedge F \rightarrow F,\ T \wedge x \rightarrow x,\ F \wedge x \rightarrow F, \\ & x \vee T \rightarrow T,\ x \vee F \rightarrow x,\ T \vee x \rightarrow T,\ F \vee x \rightarrow x. \end{aligned}$$

Next we define syntactic equality == used for the translation of the conditions c_i. Let $x :: Bool$ and $x_i, y_i :: Terms$ for $i \in [1..max(m,n)]$. Assuming $c, d \in \mathcal{F}'_0 \cup \{T, F\}$, $f, g \in \mathcal{F}'_{>0}$,

s.t. $c \neq d$, $f \neq g$, we can use the following axioms:

$$\begin{aligned}
&==(c,c) && \rightarrow T,\\
&==(c,d) && \rightarrow F,\\
&==(f(x_1,\ldots,x_n),f(y_1,\ldots,y_n)) && \rightarrow (==(x_1,y_1)\wedge(\ldots\wedge==(x_n,y_n))\ldots),\\
&==(f(x_1,\ldots,x_n),g(y_1,\ldots,y_m)) && \rightarrow F.
\end{aligned}$$

Remark that these rules are schemas for all combinations of c,d and f,g, respectively. The definition of syntactic inequality $=/=$ is dual. Note that the number of rules needed for $==$ and $=/=$ is a quadratic function over the number of function symbols in $\mathcal{F}$. The *if_then_else* mixfix function is defined as usual:

$$\mathit{if}\ T\ \mathit{then}\ x\ \mathit{else}\ y \rightarrow x\quad \mathit{if}\ F\ \mathit{then}\ x\ \mathit{else}\ y \rightarrow y$$

Matching by Term Rewriting

The complete unconditional term rewriting system $\mathcal{R}'$ for $\mathcal{R}$ is the set of all rules in $\mathcal{R}'_{\mathcal{F}}$ plus those described in this and the last paragraph. Remark that the size of $\mathcal{R}'$ depends linearly on the size of $\mathcal{R}$ and quadratically on $\mathcal{F}$. Our matching algorithm is derived from the set of transformation rules (in the style of [Jouannaud et Kirchner, 1991]) shown in Figure 3.1.

After normalisation with this set of rules, we obtain $\mathbb{F}$ if there is no solution. Otherwise, the normal form obtained is of the form $\bigwedge_{j\in[1..m]} x_j \leq u_j$, if $\sigma = \bigcup_{j\in[1..m]}\{x_j \mapsto u_j\}$ is a matcher, the empty conjunction being defined as $\mathbb{T}$.

Let $\mathit{Presolved}, \mathit{MatchPb}, \mathit{Presuccess}, \mathit{Failure} \in \mathcal{S}'$, s.t.:

$$\mathit{Presuccess}, \mathit{Failure} \leq_{\mathcal{S}'} \mathit{Presolved} \leq_{\mathcal{S}'} \mathit{MatchPb}.$$

We define the profile of the matching operations as follows:

$$\begin{aligned}
\mathit{match} &: \mathit{MatchPb} && \rightarrow \mathit{Presolved}\\
\mathit{merge} &: \mathit{Presolved} && \rightarrow \mathit{Presolved}\\
\mathit{solve} &: \mathit{MatchPb}\times\mathit{MatchPb} && \rightarrow \mathit{MatchPb}\\
\mathit{solve} &: \mathit{Presolved}\times\mathit{Presolved} && \rightarrow \mathit{Presolved}\\
\mathit{solve} &: \mathit{Presuccess}\times\mathit{Presuccess} && \rightarrow \mathit{Presuccess}\\
\leq &: \mathit{Terms}\times\mathit{IrrTerms} && \rightarrow \mathit{MatchPb}\\
\leq &: \mathit{Vars}\times\mathit{IrrTerms} && \rightarrow \mathit{Presuccess}\\
\mathit{nil} &: && \rightarrow \mathit{Presuccess}\\
\mathit{none} &: && \rightarrow \mathit{Failure}
\end{aligned}$$

We start with the axiomatization of *match*, *merge* and *solve*. Here we need to know the maximal number of variables in a rule figuring in $\mathcal{R}$, denoted by V_{max}, in order to have an upper bound for the number of variables in a matcher. We can assume V_{max} to be given, since the number of rules is finite and the rules have a finite number of variables. Assume $z :: \mathit{MatchPb}$. Informally, *match* can be given by $\mathit{match}(z) = \mathit{none}$ if $\mathit{Sol}(z) = \{\sigma\}$. Otherwise, if $\mathit{Sol}(z)$ is the singleton $\{\{x_1 \mapsto w_1, \ldots, x_k \mapsto w_k\}\}$, let $\mathit{match}(z)$ be $\mathit{solve}(d_{x_1} \leq w_1, \mathit{solve}(\ldots \mathit{solve}(d_{x_k} \leq w_k, \mathit{nil})\ldots))$.

The matching algorithm itself is a two step process. First, an arbitrary matching problem constructed with *solve*, $\leq$ and *nil* is transformed into a first presolved form, s.t. all decompositions and clash detections are performed. These are performed by *solve* itself. Let us assume $x, x_1, \ldots x_n :: \mathit{Terms}$, $y, y_1, \ldots y_{max(m,n)} :: \mathit{IrrTerms}$, $z :: \mathit{MatchPb}$ and $c^{irr}, d^{irr} \in \mathcal{F}_0^{irr}$ s.t. $c^{irr} \neq d^{irr}$ and $f^{irr}, g^{irr} \in \mathcal{F}_{\geq 0}^{irr}$, s.t. $f^{irr} \neq g^{irr}$ or $ar(()f^{irr}) = n \neq m = ar(()g^{irr})$:

$$
\begin{array}{lll}
solve(c^{irr} \leq c^{irr}, z) & \rightarrow & z, \\
solve(c^{irr} \leq d^{irr}, z) & \rightarrow & none, \\
solve(x \leq y, none) & \rightarrow & none, \\
solve(f^{irr}(x_1, \ldots, x_n) \leq g^{irr}(y_1, \ldots, y_m), z) & \rightarrow & none, \\
solve(f^{irr}(x_1, \ldots, x_n) \leq f^{irr}(y_1, \ldots, y_n), z) & \rightarrow & \\
\qquad solve(x_1 \leq y_1, solve(\ldots solve(x_n \leq y_n, z) \ldots)). &&
\end{array}
$$

Remark that the right hand sides of the matching equations for *solve* can be assumed to be ground terms, since we are only interested in ground confluence of $\mathcal{R}'$. Therefore, *solve* always returns a matching problem of sort *Presolved*. The second step consists in the merge steps necessary to decide if there is an ambiguity in the image for a variable.

Consequently, *match* and *merge* become, assuming $v_1, v_2 :: Vars$, $w_1, w_2 :: IrrTerms$, $z :: Presolved$, $z_1 :: Presuccess$ and $z_2 :: Failure$:

$$
\begin{array}{l}
match(z_1) \rightarrow merge^{V_{max}}(z_1), \; match(z_2) \rightarrow z_2, \\
merge(solve(z, nil)) \rightarrow solve(z, nil), \quad merge(none) \rightarrow none, \\
merge(solve(v_1 \leq w_1, solve(v_2 \leq w_2, z))) \rightarrow \\
\quad if(v_1 == v_2 \wedge w_1 == w_2) \; then \; merge(solve(v_1 \leq w_1, z)) \\
\quad else \; if \; (v_1 = / = v_2) \; then \; solve(v_2 \leq w_2, merge(solve(v_1 \leq w_1, z))) \\
\quad else \; none.
\end{array}
$$

Here, $merge^{V_{max}}(z)$ stands for the V_{max} times application of *merge* on z. Therefore, we need to overload *if_then_else* to $Bool \times Presolved^2 \rightarrow Presolved$. Remark that as long as *merge* traverses the list, it is of sort $MatchPb$. Only after *merge* has disappeared, the result is of sort *Presuccess* or *Failure*. This definition of *merge* is also the reason why variables need to be replaced by constants: We need to decide the difference $v_1 = / = v_2$, which is not possible with 'real' variables.

How to Apply Substitutions

In *apply*, the first argument represents the substitution calculated by *match* and the second argument is a term containing constants instead of variables. Assuming $f \in \mathcal{F}$, $x :: Presuccess$, $y_1, \ldots y_n, y :: Terms$, $w :: IrrTerms$ and $v_1, v_2 :: Vars$, we define:

$$
\begin{array}{ll}
apply : Presuccess \times Terms \rightarrow Terms & \\
apply(nil, y) & \rightarrow y, \\
apply(x, f(y_1, \ldots, y_n)) & \rightarrow f(apply(x, y_1), \ldots, apply(x, y_n)), \\
apply(x, f^{irr}(y_1, \ldots, y_n)) & \rightarrow f^{irr}(apply(x, y_1), \ldots, apply(x, y_n)), \\
apply(solve(v_1 \leq w, x), v_2) & \rightarrow if \; (v_1 == v_2) \; then \; w \; else \; apply(x, v_2).
\end{array}
$$

Therefore, *apply*'s first axiom represents the application of the empty substitution, the second to fourth stand for the homomorphism property of substitutions and the last provides recursive search of a variable replacement in the term representation of a substitution.

Trying Alternatives

The axiomatisation of *try-then-else* is done as follows: If there is a matcher, then the second argument, instantiated by the matcher, is returned. In any other case, the third argument becomes the result of the operation. Hence, we get the following operator profile:

$$try\_then\_else\ :\\ MatchPb \times Terms \times Terms \rightarrow Terms$$

Let $x_1 :: Presuccess$, $x_2 :: Failure$ and $z_1, z_2 :: Terms$. Then the axioms are as follows:

$$\begin{aligned} try\ x_1\ then\ z_1\ else\ z_2 &\rightarrow apply(x_1, z_1)\\ try\ x_2\ then\ z_1\ else\ z_2 &\rightarrow z_2, \end{aligned}$$

Properties

One can now prove [Hintermeier, 1994] that the constructed term rewriting system is orthogonal, terminating, ground confluent and that for all ground terms t in the original signature Σ:

$$(t \xrightarrow{!}_{\mathcal{R}} t' \text{ iff } t \xrightarrow{!}_{\mathcal{R}'} t'^{irr})$$

Furthermore, any evaluation of a term in $\mathcal{R}'$ corresponds with a bottom-up evaluation in $\mathcal{R}$. The proofs in [Hintermeier, 1994] were actually done for decreasing, ground-confluent CTRSs, where the result is more interesting. Nevertheless, one may notice that the transformation does what is usually done when implementing TRSs in a deterministic way.

3.2 The Expressiveness of SDTRSs

The presence of extra-variables may seem to extend the framework of ordinary unconditional term rewriting systems, but at least formally, this is clearly not true. If we restrict ourselves to ground-confluent SDTRSs, then it is clear from the adequacy of unconditional term rewriting systems that SDTRSs cannot express more computable functions. However, it may be easier to express a function in form of a confluent SDTRS instead of a simple TRS due to the increased syntactic liberty. So we have to distinguish two points of view here: First, every SDTRS may be reduced to an ordinary TRS provided it is ground-confluent. Second, from a programmer's point of view it is more interesting to have the right to express yourself using SDTRSs or something even more general, provided it results eventually in a ground-confluent term rewriting system.

For the existence of such a term rewriting system, we already collected a couple of arguments in [Hintermeier, 1994], that we briefly summarize here in the following. One way to get a transformation from CTRSs to corresponding TRSs is the construction of a Turing machine, which can be coded as a single left-linear unconditional rewrite rule [Dauchet, 1989]. However, working with such Turing machine encodings is of course not practical. Another way is the compilation of the CTRS, as proposed, e.g., by S. Kaplan [Kaplan, 1987], who transforms CTRSs into LISP code. Slight changings in the code generation, eliminating LISP operations involving static memory (*setq*) and other built-ins (like the equality test *eq*), result in a program that can be interpreted in combinatory logics. The latter having only left-linear, unconditional axioms, we get the wanted result.

Nevertheless, the correctness of the compiled code is only guaranteed under innermost evaluation [Kaplan, 1987]. This is an additional explicit restriction, which we would like to avoid, as well as the code size incrementation by a factor asymptotically proportional with $n * log(n)$ (supposing n be the size of the original code), which is inherent to the translation into combinatory logics. Note that the code size also depends on the signature, since the size of the replacement of the LISP built-in *eq* grows asymptotically like m^2, if the considered signature contains m function symbols.

Similarly, the target code for the CTRS compilation may be the one of an abstract machine, like in numerous implementations integrating functional and logic programming paradigms (e.g. [Bosco *et al.*, 1989]), often extending the Warren abstract machine principles. This is also impractical, since abstract machines typically use static memory and we do not know of any interpreter for the corresponding code in form of an unconditional term rewriting system.

For orthogonal, i.e. left-linear and non-overlapping (not necessarily terminating) CTRSs, an effective transformation was given in [Bergstra et Klop, 1986]. In the case of a *easily safely transformable* CTRS, a non-linear generalisation of orthogonal CTRSs for rules containing at most one condition, a similar transformation can be found in [Giovannetti et Moisi, 1987]. H. Aida, J. Goguen and J. Meseguer proposed a transformation of non-linear canonical CTRSs in [Aida *et al.*, 1990] using additional arguments and explicit strategy restrictions, but without detailed proofs. [Hintermeier, 1994] contains a transformation of arbitrary ground-confluent, decreasing CTRSs into canonical TRSs.

In the functional-logic programming domain, [Dershowitz et Plaisted, 1988] and [Dershowitz et Okada, 1990] discuss the use of transformations of conditional into unconditional equational theories. In [Josephson et Dershowitz, 1989], a transformation of conditional equations into unconditional ones is given in the context of narrowing, i.e. it is possible to backtrack when a condition is not satisfied, in contrast to the work presented in this paper. [Sivakumar, 1989] contains a transformation for the purpose of handling non-decreasing equations and for the detection of infeasible conditions during completion of CTRSs.

A transformation of first-order logics into discriminator varieties, i.e. algebras containing an *if-then-else* operation, is proposed in [Burris, 1992], based on work by McKenzie and Ackermann. Burris and Jeong (unpublished) also proved linear reducibility of first-order logic to unconditional equational logics. Further (unpublished) transformations were used by Chtourou/Rusinowitch and Arts/Zantema, respectively, in the termination domain, however, without preserving equivalences.

3.3 Outline of a Transformation for SDTRSs

Recall the implicit realisation of the innermost strategy using OS-term rewrite systems in Section 3.3.1. We now try to extend it for SDTRSs. First, we have to add an additional sort $FrozenTerms$ with $Terms \leq FrozenTerms$ as in [Hintermeier, 1994], which contains terms that are not reducible by any equation, but get reducible after instantiation with a substitution through a term rewriting function. This prevents us from evaluating subterms of the right hand side or the condition before we can be sure that they are strictly smaller (w.r.t. the termination ordering used to show the structural conditions for SDTRSs) than the instantiated left hand side. In any other case, we would risk non-termination.

So we have to add another mirror set of function symbols with another different label: fz for *freeze*. Let $\mathcal{F}'$ be as in Section 3.3.1, but contain additionally $\mathcal{F}^{fz} = \{f^{fz} \mid f \in \mathcal{F}\}$ with associated ranks of the form $f^{fz} : FrozenTerms^n \to FrozenTerms$ for any $f \in \mathcal{F}_n$. Now let $\mathcal{R}_f$ denote the set of rules defining f, i.e. all rules in $\mathcal{R}$ with f as top symbol on the left hand side of the rule. Let $\mathcal{R}_f$ be of the following form:

$$\mathcal{R}_f = \left\{ \begin{array}{lll} \phi_f^1 : & f(t_1^1, \ldots t_n^1) & \to r_1 \quad \text{if } c_1, \\ & \vdots & \\ \phi_f^m : & f(t_1^m, \ldots t_n^m) & \to r_m \quad \text{if } c_m \end{array} \right\}$$

For simplicity, we assume that the c_i are of the form $\bigwedge_{j \in [1..n_i]}(s_j \xrightarrow{*} s'_j)$, since any condition

of the form $s_j = s'_j$ can be transformed into an equivalent one of the form $s_j \xrightarrow{*} z \wedge s'_j \xrightarrow{*} z$, where z is a new variable.

The already mentioned freezing of the right hand side r_i of ϕ^i_f, i.e. attaching labels fz in term nodes and replacing variables by their corresponding constants, is performed by the following term transformation:

$$\begin{array}{rcl}(.)^{fz} : \mathcal{T}_{Terms}(\mathcal{X}') \rightarrow \mathcal{T}_{FrozenTerms}(\mathcal{X}') \\ f(t_1, \ldots, t_n)^{fz} & = & f^{fz}(t_1^{fz}, \ldots, t_n^{fz}) \\ x^{fz} & = & d_x\end{array}$$

The condition c_i of ϕ^i_f is represented by the term $c'_i = (s'^{cl}_1 \leq s_1^{fz})\ next\ (\ldots\ next\ (s'^{cl}_n \leq s_n^{fz}\ next\ nil)\ldots))$, assuming the empty sequence, i.e. in case that ϕ^i_f is unconditional, to be nil. This implies that $\leq$ has to be overloaded with the following declaration, additionally to the ones given in Section 3.3.1:

$$\leq : FrozenTerms \times FrozenTerms \rightarrow FrozenMatchPb$$

Remark that in order to unfreeze terms by applying substitutions, we need another axiom scheme for *apply* from Section 3.3.1:

$$apply(x, f^{fz}(y_1, \ldots, y_n)) = f(apply(x, y_1), \ldots, apply(x, y_n))$$

Furthermore, the rank of *apply* now has to be $apply : Presuccess \times FrozenTerms \rightarrow Terms$, where the second argument sort has changed from $Terms$ to $FrozenTerms$. Observe that this axiom discards the fz-label.

Now, the set of rules $\mathcal{R}_f$ can be replaced by the following rule, assuming $x_i :: IrrTerms$ for $i \in [1..n]$:

$$\begin{array}{rl}\phi'_f : f(x_1, \ldots, x_n) \rightarrow & try\ \sigma'_1\ cond\ c'_1\ then\ r_1^{fz} \\ & else\ try \\ & \vdots \\ & else\ try\ \sigma'_m\ cond\ c'_m\ then\ r_m^{fz} \\ & else\ f^{irr}(x_1, \ldots, x_n),\end{array}$$

where *try-cond-then-else* results in the instantiated right hand side if there is a match and the instantiated condition evaluates to T, otherwise in the fourth argument. This definition prevents r_i from being instantiated and reduced, if the corresponding rule in $\mathcal{R}$ is not applicable. Due to the definition of SDTRSs, $\sigma(r_i)$ may be greater than $\sigma(\phi_i)$ for any σ, s.t. $\sigma(c_i)$ reduces to F, i.e. reduction of $\sigma(r_i)$ may cause non-termination in this case. Let $\mathcal{R}'_{\mathcal{F}}$ denote the set $\{\phi'_f \mid f \in \mathcal{F}\}$. Remark that the size of $\mathcal{R}'_{\mathcal{F}}$ is linear w.r.t. the size of $\mathcal{R}$.

The difference between computing with the original set of rules and this set of unconditional rules originates in the point when the conditions are evaluated. The semantics of conditional rules suppose a rule to be applied only if its condition is satisfied. Here, in $\mathcal{R}'$, a function will only be evaluated if its arguments are irreducible, i.e. we force a bottom-up strategy. Therefore, the sort $IrrTerms$ has to contain exactly all t^{irr} for each t in the original signature Σ. Consequently, we add for any operation symbol g with $arity(g) = n \geq 0$ in Σ which does not occur on the top of a left hand side in $\mathcal{R}$, the following rules, where $x_1, \ldots, x_n :: IrrTerms$:

$$g(x_1, \ldots, x_n) \rightarrow g^{irr}(x_1, \ldots, x_n)$$

So far, this transformation is similar to [Hintermeier, 1994], except that the conjunction of conditions has now a different constructor *next* and the conditions itself are no more syntactical

equations but matching equations. Let *next* have the rank $next : MatchPb \times MatchPbList \rightarrow MatchPbList$, where $MatchPbList$ is a new sort standing intuitively for a list of matching problems with $MatchPb \leq MatchPbList$, i.e. as a consequence of subsorting, the empty match problem *nil* is also of sort $MatchPbList$. The auxiliary functions may be reused entirely as they are given in Section 3.3.1. What has to be changed is the axiomatisation of $try - cond - then - else$, which differs from both the version in Section 3.3.1, since there were no conditions, and [Hintermeier, 1994], where conditions were syntactical equalities instead of matching equations.

The axiomatisation of *try-cond-then-else* is done by primitive recursion on the list of frozen matching problems in the condition. Remark that applying a substitution to a member of the condition results in a term of the form $t \leq t'$, where t is of sort $Terms$ and t' of sort $IrrTerms$. Only after normalisation of t, this term gets of sort $IrrTerms$, too, and the matching equation is actually evaluated.

If the unfrozen matching problem has a solution, i.e. it results in a term of sort $Presuccess$, then we can try to merge the obtained matcher with the current one in the first argument of $if - then - else$, which might result in a term of sort $Failure$, if the current substitution has a different variable image than the one computed by the unfrozen matching problem. In case of failure we can return the last argument of *try-cond-then-else*, since the condition of this alternative is not satisfied. Otherwise, i.e. when the merging was successful (result of sort $Presuccess$), we can perform a recursion by defrosting the next matching equation in the condition, until there is none left. Then, we may return the third argument, i.e. the right hand side of the corresponding alternative. Hence, we get the following operator profiles:

$$\begin{aligned} &try\_cond\_then\_else : \\ &MatchPb \times MatchPb \times FrozenTerms \times Terms \rightarrow Terms \end{aligned}$$

Let $x, x_1 :: Presuccess$, $x_2 :: Failure$, $y :: MatchPbList$, $y'_1, y'_2 :: FrozenTerms$, $z_1 :: FrozenTerms$ and $z_2 :: Terms$. Then the axioms are as follows:

$$\begin{aligned} &try\ x_1\ cond\ nil\ then\ z_1\ else\ z_2 && \rightarrow\ apply(x, z_1), \\ &try\ x_1\ cond\ ((y'_1 \leq y'_2)\ next\ y)\ then\ z_1\ else\ z_2 && \rightarrow \\ &\qquad try\ match(solve(apply(x_1, y'_1) \leq apply(x_1, y'_2), x_1))\ cond\ y\ then\ z_1\ else\ z_2, \\ &try\ x_2\ cond\ y\ then\ z_1\ else\ z_2 && \rightarrow\ z_2. \end{aligned}$$

Now, our experience with such transformations makes us conjecture the following:

Conjecture 3.3.1 *The constructed term rewriting system is orthogonal, terminating, ground confluent and for all ground terms t in the original signature Σ:*

$$(t \xrightarrow{!}_{\mathcal{R}} t' \text{ iff } t \xrightarrow{!}_{\mathcal{R}'} t'^{irr})$$

Furthermore, any evaluation of a term in $\mathcal{R}'$ corresponds with a bottom-up evaluation in $\mathcal{R}$.

Chapitre 4

Clausal Theorem Proving with Equality

1 Introduction

In this chapter, we develop theorem provers based on sequent calculi and paramodulation using rewrite techniques and constraints. We therefore start with an overview over the origins of these provers in proof theory, before we come to a brief overview over similar provers and finally an overview over the contents of the chapter.

Proof theory as origin of clausal theorem proving.

> *Mathematics is just a collection of proofs.*
> (G. Takeuti, Introduction of [Takeuti, 1975])

This is the standpoint underlying proof theory. Hence, following this point of view, formalising proofs and investigating the structure of proofs should help to find properties of mathematical theories. The origins of proof theory go back to the beginning of the century. D. Hilbert and his famous program [Sieg, 1988] are the roots of proof theory, but one might also see axiomatic set theory [Jech, 1978], especially Principia Mathematica [Whitehead et Russell, 1925], as a first step to proof theory, in that these were the first global attempts to formalise proofs in analysis. In extremis one can say that geometry as developed by the ancient Greeks is the real origin. D. Hilbert's approach was further developed by G. Gentzen, who was the first to prove consistency of Peano Arithmetic and whose techniques led to a branch of Gentzen-type proof theory.

Looking at the most well-known results of Gentzen's, we can also observe that proof theory is mainly concerned with cut-elimination and consistency proofs. The former are useful to prove interpolation theorems allowing to restrict the form of proofs. The latter are interesting for the foundations of mathematics, in that they ensure that an axiomatic system is non-trivial. Remark that the Cut-Elimination Theorem for Gentzen's systems **LK** and **LJ** also provides a consistency proof through the observation that proofs after Cut-elimination only use subformulas of the theorem to be proved. This makes it impossible to derive the empty sequent which implies the consistency of the inference system.

Analogously to proof theory, we are interested in the form of proofs, but not with the goal to show consistency of our deduction systems, which is often trivial, but to find decidability results for certain fragments of logics, which are sometimes also by-products in proof theory. Term

rewriting theory as outlined in Section 3.2.1 can be seen as a (largely independent) branch of proof theory. Many of the classical results and notions, like the Diamond Lemma and confluence, deal in fact with the form of proofs. Moreover, we get in term rewriting theory the result that if the term rewriting relation is confluent and well-founded, then term rewriting gives a decision procedure for equational theorems in universal algebra.

During the last decade, many results were achieved in the area of term rewriting, going beyond universal algebra in that conditions or types were added and last but not least, to sequent calculi for full classical first-order logic, i.e. **LK**, where consistency is a consequence of a model construction in the form of an unconditional term rewriting system generating a congruence, which actually is the model. Due to our wish to have initial models, if a theory is consistent, we restrict ourselves to a Horn clause fragment of **LJ**, Gentzen's first-order intuitionistic logic, where sequents have maximally one formula in the succedent. In our framework, this formula, as well as all formulas in the antecedent of a sequent, has to be a positive equational literal. Since the semantics for such sequents is similar to the semantics of implication, as defined in the last chapter for $\mathcal{OSE}$-, R^n-and G^n-logics, we use directly Horn clauses with the implication arrow $\Leftarrow$ instead of the usually purely syntactical sequent symbol.

The calculi developed in the following use apart from term rewriting techniques also a constraint mechanism, which separates in a clean way deduction rules from side-conditions restricting the validity of a sequent. The satisfiability of the constraints used in our calculi is well-known and therefore distinguishing between deduction rules and constraint solving rules results in extracting a computational part from a semi-computational calculus. This modularisation allows us in one case to reuse the deduction part of a calculus completely by simply exchanging the constraint solver. Furthermore, when constraint solution is more costly than deduction rule applications, then the satisfiability of a constraint may be post-poned and tested only after several induction steps, when more information is available.

Clausal theorem proving with equality. The domain of Clausal Theorem Proving with Equality is closely related to Knuth-Bendix completion for conditional equations [Kounalis et Rusinowitch, 1988; Ganzinger, 1987; Ganzinger, 1991a] and paramodulation calculi, which go back to [Robinson et Wos, 1969] and for which we have given examples for the restricted case of Horn clause theories in Chapter 2. The basic problem for clausal theorem proving with equality through paramodulation is that the set of derived clauses gets easily intractable, due to the use of equations in any orientation. Rewriting Theory seems therefore to be the right tool to prune the search space considerably by using only those instances of equations that decrease a simplification or more general a reduction ordering. To illustrate this, consider the following paramodulation rule assuming all formulas to be ground and non-equational atoms $p(\vec{t})$ in clauses to be replaced by equations $p(\vec{t}) \simeq tt$, where tt is a new constant standing intuitively for truth:

$$\textbf{Para}\quad \frac{s \simeq t, \Delta \Leftarrow \Gamma,\ u[s] \simeq v, \Pi \Leftarrow \Lambda}{u[t] \simeq v, \Pi, \Delta \Leftarrow \Lambda, \Gamma}$$

Now, there are several possibilities to restrict this inference rules (cf. [Bachmair et Ganzinger, 1994a]), by using different combinations of the following conditions, assuming $\succ$ to be a reduction ordering on ground terms and formulas:

(1) $s \succ t$,

(2) $s \simeq t$ is strictly maximal in $s \simeq t, \Gamma, \Delta$ w.r.t. $\succ$,

(3) $u[s] \simeq v$ is strictly maximal in $u[s] \simeq v, \Lambda, \Pi$ w.r.t. $\succ$,

(4) $u[s] \succ v$,

(5) s does not occur in Γ.

A combination of (1),(2) and (3) is an *ordered paramodulation*, together with (4) it is a *superposition* and if (1) through (5) hold, then it is a *strict superposition* inference. A combination of (1),(3) and (4) is a *weak superposition*. Ordered paramodulation and weak superposition were proven to be refutationally complete in [Hsiang et Rusinowitch, 1991] and [Rusinowitch, 1988], respectively. Strict superposition is known to be incomplete [Bachmair et Ganzinger, 1990].

In [Bachmair et Ganzinger, 1991; Bachmair et Ganzinger, 1994a], from where adopted this part of the introduction, a superposition calculus with selection functions on negative literals and a corresponding redundancy notion are introduced. The main difference to the calculi presented in [Hsiang et Rusinowitch, 1991] and [Rusinowitch, 1988] is that superpositions are also performed into the antecedent of a clause. However, these additional superpositions are also only necessary if the literal into which superposition is performed is maximal in the corresponding clause. This is intuitively nothing else than a restricted kind of narrowing during Knuth-Bendix completion of conditional equational theories. Indeed, unfailing completion [Hsiang et Rusinowitch, 1987] procedures were developed independently for the unconditional case in order to get refutationally complete theorem provers.

A further improvement is now to use common restrictions of narrowing procedures in the context of superposition calculi, in particular the basic strategy [Hullot, 1980]. Basicness means that occurrences of subterms used as variable images for a superposition step are no more subject to further paramodulation steps. Refutational completeness for basic superposition or paramodulation, depending on the authors, was proven independently by Nieuwenhuis/Rubio and Bachmair/Ganzinger/Lynch/Snyder in [Nieuwenhuis et Rubio, 1992a] and [Bachmair *et al.*, 1992].

Overview. We start with a brief overview over the basic definitions and results together with the original calculus of Nieuwenhuis/Rubio [Nieuwenhuis et Rubio, 1992b]. The framework of R. Nieuwenhuis and A. Rubio uses equational constraints in the style of [Kirchner *et al.*, 1990] and is therefore easy to extend by complementary constraint solvers, e.g. for ordering constraints [Nieuwenhuis et Rubio, 1992b], and to extend to syntactical sort structures, like for R^n-/G^n-logics. This and the very well-structured proofs in [Nieuwenhuis et Rubio, 1992a; Nieuwenhuis et Rubio, 1992b] are the reason why we focus in Section 4.1 on this calculus, which is furthermore implemented in the SATURATE system [Nivela et Nieuwenhuis, 1993]. In analogy, we call superposition calculi having as goal the saturation of a set of clauses up to redundancy saturation calculi.

The application goal for this saturation technique is theorem proving for R^n-/G^n-logics. For this goal we adapt our deduction systems for R^n-/G^n-logic to be closer to the one of $\mathcal{OSE}$-logic in Section 4.2. The advantage of this is that $\mathcal{OSE}$-logic is close to unsorted classical equational first-order logic as used by Nieuwenhuis/Rubio. In any case, R^n or G^n, the axiom of choice/extensionality and the bijectivity of *ref*/*deref* are added as clause. In the case of extensionality, the semantic changes slightly. Section 4.2.1 addresses this problem and shows that the set of provable ground atoms does not change, i.e. the change of deduction system is correct under initial semantics.

In the case of R^n-logics, we can now use the presentation directly, after adding the additional axioms for R^n-logics as Horn clauses. For G^n-logics, we encounter an additional problem in the form of existential formulas. These formulas are strict in the sense that from the existence

of a term $t[u]$ we must be able to infer the one of the subterm u. Section 4.2.3 contains an elimination theorem for existential formulas in any G^n-presentation by transforming it into a G^{n+1}-presentation, s.t. the initial model remains unchanged, but the set of theorems for the transformed set of Horn clauses becomes bigger.

Next, in the R^n- and in the G^n- case, we replace membership formulas $s \in t$ by equations of the form $s : t = tt$, where the colon and the tt operator are new (i.e. we implicitly work with an extended system), in order to meet the formalism of Nieuwenhuis and Rubio. Remark that tt is a constant that is by definition always defined in the case of G^n-logics, i.e. $tt \simeq tt$ is added as a fact to the current set of clauses.

Both saturation calculi are presented in Section 4.3, where we eventually extend the results to a calculus with hidden information on the clause level, used to store membership information in a way close to decorations. This calculus mainly serves as a basis for reduction of a decorated superposition calculus to one with ordinary terms.

One result of this thesis, comparable to the ones of [Nieuwenhuis et Rubio, 1992a] or [Bachmair *et al.*, 1992], is to provide a saturation procedure to produce a clause set that is complete w.r.t. ground equations in R^n-/G^n-logics. This motivates the use of saturation for completion.

1.1 Syntax and Semantics for Clausal Theorem Provers with Constraints

Syntax

According to Section 3.2.1, a rewrite rule replacing t by s as $t \to s$, instead of $t \Rightarrow s$ as in [Nieuwenhuis et Rubio, 1992a], and the *congruence generated* by a ground rewrite relation R is denoted by R^*. A *multiset* will be represented by a list with the AC-constructor ',' for concatenation. We use the classical set operations of union $\cup$ and the subset relation $\subseteq$ in a straightforward way for multisets. Additionally, we use the *comprehension notation* $(e \in M \mid P(e))$ also for multisets, i.e. for the multiset of all e from the multiset M satisfying $P(e)$, preserving the number of occurrences of e if $P(e)$.

Let Σ be a regular order-sorted signature and $\succ$ be a given simplification ordering that is total on Σ-ground terms. Substitutions are written in prefix (most often for terms as arguments) or postfix (for atoms and clauses) notation. The *constraints* considered in this section are purely equational constraints, which are multisets of equalities of the form $t =^? t'$, where t, t' are Σ-terms. A conjunction T of the form $t_1 =^? t'_1, \ldots, t_k =^? t'_k$ is satisfiable iff there exists a Σ-ground unifier σ of T, i.e. $\sigma(t_i) = \sigma(t'_i)$ for $i \in [1..k]$, denoted by $T\sigma \equiv true$.

Atomic formulas are represented by L in order to distinguish them from sort constants A as used sometimes in the G-algebra framework (cf. Section 2.1). All in all, we use A, B for ground set terms from R^n-/G^n-logics, u, p for occurrences (or positions), C for clauses, $L, L', L'', \ldots$ for atoms, Γ for multisets of atoms, $\mathbf{P}, \mathbf{P}', P_0, P_1, \ldots$ for sets of clauses, $\vdash_K$ for a step of *inference* in the calculus K and $\models_{Log}$ for the *satisfaction/entailment* relation in the logic *Log*.

An atom in our framework is an equation, which is a multiset (s, t) denoted by $s \simeq t$ or $t \simeq s$, where s, t are Σ-terms. Remark that any logical atom $p(t_1, \ldots, t_n)$ can be transformed into an equation of the form $p(t_1, \ldots, t_n) \simeq tt$ using the new constant tt. This is just a simplification of the framework as in [Nieuwenhuis et Rubio, 1992b]. A *formula* in the considered calculus is a Horn clause (and not a sequent) of the form $L \Leftarrow \Gamma$, where L is an equation, called *conclusion*, and Γ is a multiset of equations, called *premise*. This is in contrast to [Nieuwenhuis et Rubio, 1992a] and [Bachmair *et al.*, 1992], where Gentzen-style sequents were used and the corresponding notions for premise and conclusion are *antecedent* and *succedent*.

Semantics

The semantics is the one of [Nieuwenhuis et Rubio, 1992a]. Let this logic be called HCL for Horn clause logic. An *interpretation* I is a congruence on ground terms. It satisfies a ground clause $L \Leftarrow \Gamma$, denoted $I \models_{HCL} L \Leftarrow \Gamma$, if $I \not\supseteq \Gamma$ or else $I \cap \Gamma \neq \emptyset$. An arbitrary constrained clause $C \; [\![T]\!]$ is satisfied by I if all its ground instances are satisfied by I, where a *ground instance!clause* of a constrained clause $C \; [\![T]\!]$ is a ground instance $C\sigma$ of C, s.t. σ satisfies T. I satisfies a set of clauses, if it satisfies every clause of S. A clause C is entailed by a set of clauses S if every model of S also satisfies C. Remark that this represents a restriction to reachable (also called term generated) models.

Hence, this includes initial semantics for Horn clauses, since if a set of ground Horn clauses is satisfied by all reachable models, then it is also satisfied by the initial model (being a reachable model[7]). The other direction (from the initial model to all reachable models) is trivial by initiality.

1.2 General Notions and Lemmas

The following definitions stem from [Nieuwenhuis et Rubio, 1992b] and are only slightly changed in order to allow for a parameterisation for the used satisfaction relation. Let tt be the smallest symbol in the given simplification ordering $\succ$ and $\succ_{mul}$ ($\succ_{mul^n}$) denote its (n-fold) multiset extension. Recall that $\succ$ has to be total on ground terms.

The *multiset expression of an equation* $t \simeq t'$ in a clause $L \Leftarrow \Gamma$ is $\{\{t,t\},\{t',t'\}\}$ if $t \simeq t'$ belongs to Γ and $\{\{t\},\{t'\}\}$ if $t \simeq t'$ is L. The ordering $\succ_E$ on ground equations in clauses is by definition the ordering $\succ_{mul^2}$ on their multiset expressions. The ordering $\succ_C$ on ground clauses is by definition the ordering $\succ_{mul^3}$ on the multisets containing the multiset expressions of their equations. A ground equation is called maximal (resp. strictly maximal) in a ground clause C if $e \succeq_E e'$ (resp. $e \succ_E e'$), for every other equation e' in C.

As usual, we assume that the initial set of axioms contains only clauses $C \; [\![T]\!]$ with empty constraints, sometimes written $C \; [\![true]\!]$. We continue with the definitions of [Nieuwenhuis et Rubio, 1992b], indicating the dependencies on the satisfaction relation $\models_{Log}$ of a logic Log with associated inference system $\mathcal{K}$ by (Log-) and ($\mathcal{K}$-) prefixes, respectively:

Definition 4.1.1 *A ground substitution σ is* irreducible *w.r.t. a set of ground rewrite rules R if $\sigma(x)$ is irreducible w.r.t. R for every variable x in the domain of σ. A normal form of σ w.r.t. R is a substitution σ' with the same domain as σ and s.t. $\sigma'(x)$ is a normal form w.r.t. R of $\sigma(x)$, for every variable x in the domain of σ.*

Definition 4.1.2 *Let D be a ground instance $t \simeq s \Leftarrow \Gamma$ of a clause $C \; [\![T]\!]$ in a set S. Then D (Log-)*generates *the ground rewriting rule $t \to s$ if the following conditions hold:*

1. $R_D^* \not\models_{Log} D$
2. *$t \simeq s$ is strictly maximal w.r.t. $\succ_E$ in D with $t \succ s$*
3. *t is irreducible by R_D*
4. *σ is irreducible by R_D*

[7] At least in the categories of models of the logics considered here.

where R_D is the set of rules generated by ground instances smaller than D w.r.t. $\succ_C$ of clauses in S. R_S is the set of ground rewrite rules generated by all ground instances of clauses in S.

Definition 4.1.3 *Let S be a set of constrained clauses and let R be a set of ground rewrite rules. The set $irred_R(S)$ is defined as follows:*

$$irred_R(S) = \{C\sigma \ : \ C \ [\![T]\!] \in S, T\sigma = true, \sigma \ ground, \sigma \ irreducible \ w.r.t. \ R\}$$

Definition 4.1.4 *Let $P_0, P_1, \ldots$ be a sequence of sets of constrained clauses.*

1. *The set P_∞ of* persistent *clauses in the sequence is $\cup_j(\cap_{k \geq j} P_k)$.*

2. *A clause $C \ [\![T]\!]$ is (Log-)*redundant *in a set P_j if for every ground instance $C\sigma$ of it with σ irreducible w.r.t. R_{P_∞}, there exist instances D_i in $irred_{R_{P_\infty}}(P_j)$, for all $i \in [1..m]$, s.t. $C\sigma \succ_C D_i$ and:*

$$R_{P_\infty} \cup \{D_1, \ldots, D_m\} \models_{Log} C\sigma$$

Definition 4.1.5 *A (Log-)*theorem proving derivation *is a sequence $P_0, P_1, \ldots$ of sets of constrained clauses, s.t.:*

$$\begin{array}{ll} P_i = P_{i-1} \cup \{C \ [\![T]\!]\}, & \text{where } P_{i-1} \models_{Log} C \ [\![T]\!], \text{ or} \\ P_i = P_{i-1} \setminus \{C \ [\![T]\!]\}, & \text{if } C \ [\![T]\!] \text{ is redundant in } P_{i-1}. \end{array}$$

An ($\mathcal{K}$-)inference is (*Log*-)redundant in a set P_j of a (*Log*-)theorem proving derivation, if in every R_{P_∞}-irreducible instance of it, the conclusion can be deduced from irreducible instances of clauses P_j smaller than the maximal premise.

Definition 4.1.6 *Let $P_0, P_1, \ldots$ be a (Log-)theorem proving derivation and let π be an ($\mathcal{K}$-) inference with premises $C_1 \ [\![T_1]\!], \ldots, C_m \ [\![T_m]\!]$, and with conclusion $C \ [\![T]\!]$.*

1. *Every ($\mathcal{K}$-)inference with premises $C_1\sigma_1, \ldots, C_m\sigma_m$ and conclusion $C\sigma$ for some ground substitution σ with $T\sigma \equiv true$ is a* ground instance *$\pi\sigma$ of π.*

2. *The ($\mathcal{K}$-)inference π is (Log-)*redundant *in a set of constrained clauses P if for every ground instance $\pi\sigma$ of π with σ irreducible w.r.t. R_{P_∞}, there exist instances D_i in the set $irred_{R_{P_\infty}}(P)$, $i \in [1..k]$, s.t. $max(C_1\sigma, \ldots, C_m\sigma) \succ_C D_i$ and $R_{P_\infty} \cup \{D_1, \ldots, D_k\} \models_{Log} C\sigma$.*

3. *A set P is (Log-)*saturated *if every ($\mathcal{L}$-)inference π with premises in P is (Log-)redundant in P.*

Definition 4.1.7 *A (Log-)theorem proving derivation is (Log-)*fair *if every ($\mathcal{K}$-)inference with premises in P_∞ is (Log-)redundant in some P_j w.r.t. R_{P_∞}.*

We prove some easy consequences of the definitions (cf. [Nieuwenhuis et Rubio, 1992b], except the first one):

Lemma 4.1.8 *Let $l \to r$ be a ground rewrite rule (Log-)generated by C in P. Then l is irreducible by all clauses in R_P other than C.*

Proof: By definition of generation by C, l has to be R_C-irreducible. Let $l' \to r'$ be a clause generated by C' in P, s.t. $l' \to r'$ reduces l. Then, $l' \succeq l$, since l' is maximal in C' and assuming $l \succ l'$ thus automatically leads to $C \succ C'$, in contradiction to the definition of a generated ground rewrite rule. If $l' \succ l$, then l is irreducible by $l' \to r'$, since $\succ$ is a simplification ordering and therefore all subterms of l must be strictly smaller than l' w.r.t. $\succ$, i.e. they cannot match. Remains the case where l' is l. But then l' is $R_{C'}$-reducible by $l \to r$, since $C' \succ C$, in contradiction to the definition of generation by C'. □

We get as immediate conclusion:

Corollary 4.1.9 *If $(s\theta \to s'\theta)$ is (Log-)generated by an instance of a clause C of the form $(s \simeq s') \Leftarrow \Gamma$ in P, then $\theta(x)$ is R_P-irreducible for all $x \in \mathcal{V}ar(s, s', \Gamma)$.*

Proof: By definition of generation, we know that $\theta(x)$ is R_C-irreducible for all $x \in \mathcal{D}om(\theta)$. By the last lemma we know that $s\theta$ is R_P-irreducible.

Assume that $\theta(x)$ for $x \in \mathcal{V}ar(s', \Gamma)$ is reducible by a rule generated by some clause instance $C'\sigma \succ C\theta$. But then, similar to the last lemma, the left hand side l of the rule $l \to r$ generated by $C'\sigma$ cannot be greater or equal to $s\theta$, since otherwise $\theta(x) \succeq s\theta$ and therefore either $s'\theta \succeq s\theta$ (if $x \in \mathcal{V}ar(s')$) or similarly $\Gamma\theta \succ_C (s\theta \simeq s'\theta)$ (if $x \in \mathcal{V}ar(\Gamma)$ – recall that terms in premisses 'count twice'), in contradiction to the definition of *Log*-generation. Therefore $C\theta \succ_C C'\sigma$ contradicting the well-foundedness of $\succ_C$. □

Lemma 4.1.10 *Let $P_0, P_1, \ldots$ be a (Log-)theorem proving derivation. Then for every set P_j and instance C in $irred_{R_{P_\infty}}(P_j)$, there are instances D_i for $i \in [1..m]$ in $irred_{R_{P_\infty}}(P_\infty)$, s.t. $C \succeq_C D_i$ and:*

$$R_{P_\infty} \cup \{D_1, \ldots, D_m\} \models_{Log} C$$

Proof: In order to do a proof by contradiction, assume that the clause C is minimal w.r.t. $\succ_C$ in $irred_{R_{P_\infty}}(P_j)$ for all $j \geq 0$, s.t. there are no such instances D_i in $irred_{R_{P_\infty}}(P_\infty)$. The corresponding clause $C' \llbracket T \rrbracket$ is not persistent, since otherwise C were in $irred_{R_{P_\infty}}(P_\infty)$. Hence, $C' \llbracket T \rrbracket$ is *Log*-redundant in some P_k with $k \geq j$, i.e. there exist instances $D'_1, \ldots, D'_{m'}$ in $irred_{R_{P_\infty}}(P_k)$ s.t. $R_{P_\infty} \cup \{D'_1, \ldots, D'_{m'}\} \models_{Log} C$ with $C \succ_C D'_j$ for $j \in [1..m']$. Since C was minimal we can apply the Lemma to any D'_j, and therefore the Lemma holds for C. □

Lemma 4.1.11 *Let $P_0, P_1, \ldots$ be a (Log-)theorem proving derivation.*
If an ($\mathcal{K}$-)inference is Log-redundant in some P_j, then it also is in P_∞.

Proof: Let π be an ($\mathcal{K}$-)inference with premises $C_1 \llbracket T_1 \rrbracket, \ldots, C_k \llbracket T_k \rrbracket$ and with conclusion $C \llbracket T \rrbracket$, s.t. π is (*Log*-)redundant in P_j. By the definition of a (*Log*-) redundant inference, for every ground instance $\pi\sigma$ of π with σ irreducible w.r.t. R_{P_∞}, there exist instances D_i in $irred_{R_{P_\infty}}(P_j)$, for $i \in [1..m]$, s.t. $max(C_1\sigma, \ldots, C_k\sigma) \succ_C D_i$ and furthermore $R_{P_\infty} \cup \{D_1, \ldots, D_m\} \models_{Log} C\sigma$. By Lemma 4.1.10, each of the instances D_i can be deduced from R_{P_∞} and other instances $\{D'_1, \ldots, D'_{m'}\}$ in $irred_{R_{P_\infty}}(P_\infty)$ s.t. $\forall i \in [1..m] \exists j \in [1..m'], D_i \succeq_C D'_j$. This implies that π is also redundant in P_∞. □

Lemma 4.1.12 *If $P_0, P_1, \ldots$ is a (Log-)fair theorem proving derivation, then P_∞ is (Log-) saturated.*

Proof: By (*Log*-)fairness, every ($\mathcal{K}$-)inference π with premisses in P_∞ is redundant in some P_j. By Lemma 4.1.11, π is also redundant in P_∞, i.e. P_∞ is complete. □

The models construction lemmas based on the last lemmas depend on the inference system $\mathcal{K}$. We therefore have to redo them in every particular case for *Log* and $\mathcal{K}$.

1.3 Basic Superposition with Equational Constraint

We present in this section the basic superposition algorithm from [Nieuwenhuis et Rubio, 1992b], but specialised for Horn clauses, i.e. without factorisation. Proofs are given extensively, since they will serve as basis for an extension to R^n-/G^n-logics in the following sections. This calculus is called $\mathcal{NR}$ in the following. These rules together with equational constraints, are suitable for deduction in HCL.

1. *strict basic superposition into conclusion* :

$$\frac{s \simeq s' \Leftarrow \Gamma' \; [\![T']\!] \qquad t \simeq t' \Leftarrow \Gamma \; [\![T]\!]}{t[s']_u \simeq t' \Leftarrow \Gamma, \Gamma' \; [\![T, T', t|_u =^? s]\!]}$$

where $t|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t|_u =^? s]\!]$ has a ground solution σ, s.t.:

(a) $t\sigma \succ t'\sigma, s\sigma \succ s'\sigma$ and $t\sigma \simeq t'\sigma \succ_E s\sigma \simeq s'\sigma$,

(b) $s\sigma \simeq s'\sigma$ is strictly maximal in $s\sigma \simeq s'\sigma \Leftarrow \Gamma'\sigma$ and

(c) $t\sigma \simeq t'\sigma$ is strictly maximal in $t\sigma \simeq t'\sigma \Leftarrow \Gamma$.

2. *strict basic superposition into premise* :

$$\frac{s \simeq s' \Leftarrow \Gamma' \; [\![T']\!] \qquad L \Leftarrow \Gamma, t \simeq t' \; [\![T]\!]}{L \Leftarrow t[s']_u \simeq t', \Gamma, \Gamma' \; [\![T, T', t|_u =^? s]\!]}$$

where $t|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t|_u =^? s]\!]$ has a ground solution σ, s.t.:

(a) $t\sigma \succ t'\sigma$ and $s\sigma \succ s'\sigma$,

(b) $s\sigma \simeq s'\sigma$ is strictly maximal in $s\sigma \simeq s'\sigma \Leftarrow \Gamma'\sigma$ and

(c) $t\sigma \simeq t'\sigma$ is maximal in $L\sigma \Leftarrow \Gamma\sigma, t\sigma \simeq t'\sigma$.

3. *equality resolution* :

$$\frac{L \Leftarrow t \simeq t', \Gamma' \; [\![T']\!]}{L \Leftarrow \Gamma' \; [\![T', t =^? t']\!]}$$

where $[\![T', t =^? t']\!]$ has a ground solution σ, s.t. $t\sigma \simeq t'\sigma$ is maximal in $L\sigma \Leftarrow t\sigma \simeq t'\sigma, \Gamma'\sigma$.

The corresponding equational constraint solver (from [Jouannaud et Kirchner, 1991]) is an unsorted unification algorithm shown in Figure 4.1, which is well-known to be sound, complete and terminating [Dershowitz et Jouannaud, 1990a].

Remark that an alternative possibility is the use of ordering constraints as in [Nieuwenhuis et Rubio, 1992b], based on the solver in [Comon, 1990]. The use of such a constraint solver

Delete	$\dfrac{[\![x =^? x, T]\!]}{[\![T]\!]}$
Decompose	$\dfrac{[\![f(t_1,\ldots,t_m) =^? f(t'_1,\ldots,t'_m), T]\!]}{[\![t_1 =^? t'_1,\ldots,t_m =^? t'_m, T]\!]}$
Conflict	$\dfrac{[\![f(t_1,\ldots,t_m) =^? g(t'_1,\ldots,t'_{m'}), T]\!]}{\bot}$ if $f \neq g$ or $m \neq m'$.
Coalesce	$\dfrac{[\![x =^? y, T]\!]}{[\![x =^? y, T\{x \mapsto y\}]\!]}$ if $x, y \in \mathcal{V}ar(T)$
Check*	$\dfrac{[\![x_1 =^? s_1[x_1],\ldots,x_n =^? s_n[x_1], T]\!]}{\bot}$
Merge	$\dfrac{[\![x =^? t', x =^? t'', T]\!]}{[\![x =^? t', t' =^? t'', T]\!]}$ if $0 < \lvert t'\rvert \leq \lvert t''\rvert$.

Figure 4.1: **Dag – Unify** : A constraint solver for unsorted unification

is expensive but results in general in less generated clauses. We think that the use of ordering constraints in this case is an orthogonal problem and that the solvers in [Comon, 1990] and [Nieuwenhuis et Rubio, 1992b] may be easily adapted for our use if needed. We may state the following (cf. [Nieuwenhuis et Rubio, 1992a]):

Theorem 4.1.13 *The inference system $\mathcal{NR}$ is sound for HCL-deduction from any set of unsorted Horn clauses.*

Proof: Obvious. □

Recall the general definitions and lemmas from Section 4.1.2 before we start with the model construction.

Lemma 4.1.14 *Let $P_0, P_1, \ldots$ be a fair HCL-theorem proving derivation, s.t. P_∞ does not contain the empty clause. Then $R^*_{P_\infty} \models_{HCL} irred_{R_{P_\infty}}(P_\infty)$.*

Proof: In order to derive a contradiction, assume that $C\sigma \in irred_{R_{P_\infty}}(P_\infty)$ is a minimal (w.r.t. $\succ_C$) instance of a clause $C\ [\![T]\!]$ in P_∞, s.t. $R^*_{P_\infty} \not\models_{HCL} C\sigma$. We distinguish the different cases for $C\sigma$:

1. $C\sigma$ is of the form $(t\sigma \simeq t'\sigma) \Leftarrow \Gamma$, s.t. $t\sigma \simeq t'\sigma$ is maximal in $C\sigma$:
   (a) If $t\sigma$ is $t'\sigma$, then $C\sigma$ is redundant, since it is subsumed by reflexivity.
   (b) If $t\sigma \succ t'\sigma$, then $C\sigma$ has not HCL-generated $t\sigma \rightarrow t'\sigma$, since this would be in contradiction to $R^*_{P_\infty} \not\models_{HCL} C\sigma$. This must be due to non-satisfaction of condition 3 in definition 4.1.2. Hence, $t\sigma$ must be reducible in $R_{C\sigma}$, e.g. with a rule $s\theta \rightarrow s'\theta$, generated by a clause $C'\theta$ strictly smaller than $C\sigma$. Let C' be of

the form $(s \simeq s') \Leftarrow \Gamma'$ in P_∞ and $t\sigma|_u = s\theta$. So we can do an inference π using *strict basic superposition into conclusion*:

$$\frac{(s \simeq s') \Leftarrow \Gamma' \; [\![T']\!] \; (t \simeq t') \Leftarrow \Gamma \; [\![T]\!]}{(t[s']_u \simeq t') \Leftarrow \Gamma, \Gamma' [\![T, T', t|_u =^? s]\!]}$$

The conclusion of π has a ground instance D, which is of the form $(t\sigma[s'\theta]_u \simeq t'\sigma) \Leftarrow \Gamma\sigma, \Gamma'\theta$, s.t. $R^*_{P_\infty} \not\models_{HCL} D$ – otherwise we would have $R^*_{P_\infty} \models_{HCL} C\sigma$, since $R^*_{P_\infty} \models_{HCL} C'\theta$. Moreover, D is an instance of this conclusion by an R_{P_∞}-irreducible ground substitution (remember that σ is R_{P_∞}-irreducible and θ is $R_{C\theta}$-irreducible, implying R_{P_∞}-irreducibility by the definition of generation – cf. Corollary 4.1.9). Since P_∞ is saturated, π must be HCL-redundant in P_∞. Hence, by definition of HCL-redundancy, we know that there exist instances $D_1, \ldots, D_m$ in $irred_{R_{P_\infty}}(P_\infty)$, s.t. $R_{P_\infty} \cup \{D_1, \ldots, D_m\} \models_{HCL} D$ and $C\sigma \succ_C D_i$. Now, $R^*_{P_\infty} \not\models_{HCL} D$ implies that $R^*_{P_\infty} \not\models_{HCL} D_i$ for at least one D_i and thus C_σ cannot be minimal w.r.t. $\succ_C$.

2. $C\sigma$ is of the form $L' \Leftarrow \Gamma, t\sigma \simeq t'\sigma$, s.t. $t\sigma \simeq t'\sigma$ is maximal in $C\sigma$:
   (a) If $t\sigma$ is $t'\sigma$, then $C\sigma$ is redundant, since we may apply equality resolution to $C\,[\![T]\!]$, which is a $\mathcal{HCL}$-redundant inference leading as before to a contradiction.
   (b) If $t\sigma \succ t'\sigma$, then $R^*_{P_\infty} \models_{HCL} t\sigma \simeq t'\sigma$, since $R^*_{P_\infty} \not\models_{HCL} C\sigma$. $R^*_{P_\infty} \models_{HCL} t\sigma \simeq t'\sigma$ implies $t\sigma \rightarrow^*_{R_{P_\infty}} \circ \leftarrow^*_{R_{P_\infty}} t'\sigma$ by the confluence of $R^*_{P_\infty}$ implying reducibility of $t\sigma$, since $t\sigma \succ t'\sigma$ and thus $t\sigma \neq t'\sigma$. Assume the applicable rule is $s\theta \rightarrow s'\theta$ in R_{P_∞} generated by a clause in P_∞ of the form $s \simeq s' \Leftarrow \Gamma'$, where $t\sigma|_u = s\theta$. The following inference π can be performed using *strict basic superposition into premise*:

$$\frac{s \simeq s' \Leftarrow \Gamma' [\![T']\!] \; L \Leftarrow \Gamma, t \simeq t'}{L \Leftarrow \Gamma, \Gamma', t[s']_u \simeq t' [\![T, T', t|_u = s]\!]}$$

   For the corresponding ground instance, $C\sigma$ is the strictly maximal premise, since $t\sigma \succeq s\theta \succ s'\theta$, $s\theta \simeq s'\theta$ is maximal in the associated clause and $t\sigma \simeq t'\sigma$ appears in the premise, i.e. the multiset expression for $t\sigma \simeq t'\sigma$, which is $\{\{t\sigma, t\sigma\}, \{t'\sigma, t'\sigma\}\}$, is strictly greater than the one of $s\theta \simeq s'\theta$, which is $\{\{s\theta\}, \{s'\theta\}\}$, and the ones of the other atoms in the premise of that clause. If D stands for its conclusion then $R^*_{P_\infty} \not\models_{HCL} D$. As before, π is HCL-redundant which leads to a contradiction.

□

Theorem 4.1.15 *Let $P_0, P_1, \ldots$ be a fair HCL-theorem proving derivation, where P_0 is a set of clauses with empty constraints. Then P_0 is consistent if and only if the empty clause belongs to some P_j. Furthermore, if the empty clause does not belong to P_∞, then $R^*_{P_\infty} \models_{HCL} P_0$.*

Proof: By the soundness of $\mathcal{NR}$ (cf. Theorem 4.1.13), we know that P_0 is inconsistent if P_j contains the empty clause. In the other direction, we prove that P_0 is consistent if the empty clause does not belong to any P_j. If this is the case, then it is not in P_∞ and therefore $R^*_{P_\infty} \models_{HCL} irred_{R_{P_\infty}}(P_\infty)$ by Lemma 4.1.14 and 4.1.12.

Next, suppose $C\,[\![true]\!]$ is in P_0 and $C\sigma$ is a ground instance of $C\,[\![true]\!]$ in $irred_{R_{P_\infty}}(P_0)$. By Lemma 4.1.10, we get the existence of instances $D_1, \ldots, D_m$ in $irred_{R_{P_\infty}}(P_\infty)$, s.t.:

$$R_{P_\infty} \cup \{D_1, \ldots, D_m\} \models_{HCL} C\sigma$$

Hence, $R^*_{P_\infty} \cup irred_{R_{P_\infty}}(P_\infty) \models_{HCL} irred_{R_{P_\infty}}(P_0)$.

Therefore, $R^*_{P_\infty} \models_{HCL} P_0$, since any ground instance $D\theta$ of a clause D $[\![true]\!]$ in P_0 satisfies $R^*_{P_\infty} \models_{HCL} D\theta$: Consider the normal form θ' of θ in R_{P_∞}, i.e. $R_{P_\infty} \cup \{D\theta'\} \models_{HCL} D\theta$ and $D\theta'$ is in $irred_{R_{P_\infty}}(P_0)$. Therefore, as we have shown above, $R^*_{P_\infty} \models_{HCL} D\theta'$. Altogether, this gives $R^*_{P_\infty} \models_{HCL} D\theta$.

Now, since $R^*_{P_\infty} \models_{HCL} P_0$ we know that P_0 is consistent. □

Clearly, R_{P_∞} is a confluent ground rewrite system. However, due to extra-variables in clauses, i.e. variables occuring in the body but not in the head of clauses, we cannot use P_∞ as a (quasi-)reductive CTRS [Ganzinger, 1991b; Jouannaud et Waldmann, 1986] in general, since these variables are in general not predetermined and therefore existentially quantified. Hence, we restrict to SDTRSs which allow us to find an evaluation order that avoids narrowing steps.

Lemma 4.1.16 *Let $P_0, P_1, \ldots$ be a fair theorem proving derivation, where P_0 is a set of clauses with empty constraints, s.t. the empty clause does not belong to P_∞.*

If $P_R \subseteq P_\infty$ contains all generating instances of clauses and P_R is an SDTRS, then no narrowing steps are needed in order to solve the goal $\Leftarrow t = x$ for any ground term t in NR.

Proof: The irreducibility of the right hand sides of conditions in SDTRSs guarantees that we only need to reduce ground terms, because of the restrictions of variable occurrences in conditions of SDTRS rules (cf. Definition 3.2.4). □

Theorem 4.1.17 *Let $P_0, P_1, \ldots$ be a fair theorem proving derivation, where P_0 is a set of clauses with empty constraints, s.t. the empty clause does not belong to P_∞.*

If $P_R \subseteq P_\infty$ contains all generating instances of clauses and P_R is an SDTRS, then P_R is ground confluent.

Proof: First of all remark that any generating instance of a quasi-reductive conditional equation must be such that extra-variables are instantiated with minimal terms, s.t. the condition holds. Hence, evaluating equations in conditions can be done by reducing only ground members whose normal forms are matched against the non-ground member and the resulting substitution can be applied to the remaining equations in the condition and the right hand side of the conclusion before solving the next condition or, if all are solved, returning the right hand side.

We know by Theorem 4.1.15 that $R^*_{P_\infty}$ is a model of P_0 and that $R^*_{P_\infty}$ is complete w.r.t. P_0. Hence, all equations valid in P_0 can be proved in $R^*_{P_\infty}$. Now, since $R^*_{P_\infty}$ is ground confluent, we can resolve any peak on a ground term. Furthermore, by definition, every rule $l \to r$ in R_{P_∞} is a generating instance of a clause in P_R, i.e. it is subsumed by the operational semantics of SDTRSs, as mentioned above. □

Remark that a normal form of a ground term t can be calculated by solving the goal C of the form $\Leftarrow t \simeq x$ for a new variable x. Saturating $P_\infty \cup \{C\ [\![true]\!]\}$ then gives a new limit presentation P'_∞ in which we will find a descendant $\square\ [\![T]\!]$ of $C\ [\![true]\!]$ containing the normal form in form of $\sigma(x)$, where σ is the principal solution of the equational constraint T.

2 Reduction of R^n-/G^n-to $\mathcal{OSE}$-Logics

In this section, we give proof transformations allowing us to exchange **RNL** and **GNL**, the deduction systems from Sections 2.3 and 2.4, respectively, against alternative calculi being closer to **OSL**, the order-sorted equational Horn clause calculus from Section 2.2, which is itself closer to calculi underlying $\mathcal{HCL}$-logics, which was used in the last section as semantics for the $\mathcal{NR}$-calculus. The reason for these transformations is thus to simplify the construction of a saturation calculus for R^n-and G^n-logic.

2.1 Eliminating the Set Theoretic Inference Rules of R^n-Logics

First, we prove that under initial semantics, it is harmless to replace the 'extra'-deduction rules **Choice**, **Ref**, **Deref** and **Ext** by corresponding Horn clauses, since the deducible ground atoms remain the same.

Let $\mathbf{P}$ be a R^n-Σ-presentation in the following and $\mathbf{P}^+$ be $\mathbf{P}$ extended by the following set of clauses:

$$\mathbf{P}_R = \left\{ \begin{array}{l} choose(x^m) \in x^m, \\ ref(deref(x^{kr})) = x^{kr}, \\ deref(ref(x^k)) = x^k, \\ u^m = t^m \Leftarrow choose(u^m) \in t^m, choose(t^m) \in u^m \end{array} \right\}_{(m \in [1..n], k \in [0..n])}$$

Let $\vdash_{RNL}$ stand for deducibility in R^n-logics using the set of deduction rules **RNL** from Section 2.3 and $\vdash_{OSL}$ for deducibility in **RNL** without the rules **Choice**, **Ref**, **Deref** and **Ext**, i.e. with the set of $\mathcal{OSE}$-deduction rules **OSL** from Section 2.2.

Theorem 4.2.1 *Let L be a ground Σ-atom.*
Then $\mathbf{P} \vdash_{RNL} L$ iff $\mathbf{P}^+ \vdash_{OSL} L$.

Proof: From left to right this is obvious, since any application of **Ext** can be replaced by two times **Cut** with the Horn clause replacement and any application of the other three rules may be replaced by **Axioms**.

From right to left, we proceed by induction on the size of the deduction tree. Let τ be a ground substitution with domain $\mathbf{V}$ and $L' \Leftarrow \Gamma$ a Σ-clause. We prove that $\mathbf{P}^+ \vdash_{OSL} L' \Leftarrow \Gamma$ in m_1 steps and $\mathbf{P}^+ \vdash_{OSL} \tau(\Gamma)$ in m_2 steps, both using only variables in $\mathbf{V}$, implies that $\mathbf{P} \vdash_{RNL} \tau(L')$ by induction on $m = m_1 + m_2$. We distinguish the different cases for the last applied deduction rule[8] in $\mathbf{P}^+ \vdash_{OSL} L' \Leftarrow \Gamma$:

- **Reflexivity**: Apply **Substitutivity** with τ after **Reflexivity**.
- **Axioms**: If $L \Leftarrow \Gamma$ is in $\mathbf{P}$, then this is similar to the last case. Otherwise, it must be in $\mathbf{P}_R$, i.e. either L' is one of **Ref**, **Deref**, **Choice** or $L' \Leftarrow \Gamma$ is **Ext**. In the first case we may apply again **Axioms** followed by **Substitutivity** in order to get $\tau(L')$. In the second case, we know by the induction hypothesis that $\mathbf{P} \vdash_{RNL} \tau(\Gamma)$, where $\Gamma = (choose(u^m) \in t^m, choose(t^m) \in u^m)$, and therefore we may use **Cut** twice (after **Axioms** for the clause replacement of **Ext** and **Substitutivity** with τ), yielding $\tau(t^m) = \tau(u^m)$.
- **Substitutivity**: Let $\sigma(L) \Leftarrow \sigma(\Gamma)$ be the conclusion. We know that $\mathbf{P} \vdash_{RNL} \tau(\sigma(\Gamma))$ and therefore $\mathbf{P} \vdash_{RNL} \tau(\sigma(L))$ by the induction hypothesis.

[8] Remark that **Axioms** and **Reflexivity** give the base case $m = 1$.

- **Cut**: Let $L \Leftarrow \Gamma, L'$ and $L' \Leftarrow \Gamma'$ be the premises and $L \Leftarrow \Gamma, \Gamma'$ be the conclusion. We know that $\mathbf{P} \vdash_{RNL} \tau(\Gamma, \Gamma')$ and hence $\mathbf{P} \vdash_{RNL} \tau(L')$ by the induction hypothesis and **Cut**. Consequently, $\mathbf{P} \vdash_{RNL} \tau(\Gamma, L')$ and once again by induction hypothesis and **Cut**, we get $\mathbf{P} \vdash_{RNL} \tau(L)$.
- **Paramodulation**: Let $L[s] \Leftarrow \Gamma$ and $(s = t) \Leftarrow \Gamma'$ be the premises and $L[t] \Leftarrow \Gamma, \Gamma'$ be the conclusion. We know that $\mathbf{P} \vdash_{RNL} \tau(\Gamma, \Gamma')$ and hence $\mathbf{P} \vdash_{RNL} \tau(L[s], s = t)$ by the induction hypothesis and **Cut**. Consequently, by **Paramodulation**, we get $\mathbf{P} \vdash_{RNL} \tau(L[t])$.

□

2.2 Eliminating the Set Theoretic Inference Rules of G^n-Logics

Analogously, if $\mathcal{P}$ is a G^n-presentation, let $\mathcal{P}^+$ be defined as for R^n-presentations and **GNL**$^-$ be **GNL** from Section 2.4 without **Choice**, **Ref**, **Deref** and **Ext**.

Theorem 4.2.2 *Let L be a ground Σ-atom.*
Then $\mathbf{P} \vdash_{GNL} L$ iff $\mathbf{P}^+ \vdash_{GNL^-} L$.

Proof: Analogously to Theorem 4.2.1, the right to left direction is trivial. Let τ be a ground substitution in $\mathbf{P}$-$SUBST_\Sigma(\mathbf{V})$ and $L' \Leftarrow \Gamma$ a Σ-clause. From left to right, we prove as before that $\mathbf{P}^+ \vdash_{GNL^-} L' \Leftarrow \Gamma$ in m_1 steps and $\mathbf{P}^+ \vdash_{GNL^-} \tau(\Gamma)$ in m_2 steps, both using only variables in $\mathbf{V}$, implies that $\mathbf{P} \vdash_{GNL} \tau(L')$ by induction on $m = m_1 + m_2$. We distinguish the different cases for the last applied deduction rule[9] in $\mathbf{P}^+ \vdash_{GNL^-} L' \Leftarrow \Gamma$:

- **WellDef**: We know that $\mathbf{P} \vdash_{GNL} \tau(\Phi[t])$ by the induction hypothesis and hence $\mathbf{P} \vdash_{GNL} EX\ \tau(t)$ by **WellDef**.
- **PartialReflex**: We know that $\mathbf{P} \vdash_{GNL} EX\ \tau(t)$ by the induction hypothesis and hence $\mathbf{P} \vdash_{GNL} \tau(t = t)$ by **PartialReflex**.
- **Axioms**: Analogous to Theorem 4.2.1 except that **Substitutivity** has to be replaced by **SubstConform**.
- **SubstConform**: Let $\sigma(L) \Leftarrow \sigma(\Gamma)$ be the conclusion. We know that $\mathbf{P}^+ \vdash_{GNL^-} \tau(\sigma(\Gamma))$ and hence by induction hypothesis that $\mathbf{P} \vdash_{GNL} \tau(\sigma(L'))$. Remark that $\mathbf{P}$-conform substitutions are closed under composition.
- **Cut**, **Paramodulation**: Analogous to Theorem 4.2.1.

□

2.3 An EX - Elimination Theorem

In the following, unless specified explicitely, let Σ be a G^n-signature and $\mathbf{P}$ be a G^n-presentation. We argue that the existential formulas $EX\ t$ are in principal not necessary in the calculus since they can be replaced by membership formulas of the form $t \in \Omega$ for an additional constant Ω representing the universe of individuals. Furthermore, we can transform the clauses s.t. the deduction system **OSL** with a changed reflexivity rule is sufficient for deduction in G^n-logic.

[9] Remark that **Axioms** gives the base case $m = 1$.

The reason for this rather complex transformation is the bad behaviour of the **WellDef** inference rule for existential formulas in the **GNL**-calculus. A naive approach for this rule is an almost literal translation of it into a clause of the form:

$$EX\ x \Leftarrow EX\ f(\ldots x \ldots).$$

However, this clause is clearly more adapted to forward chaining than to backward chaining. The result is that a saturation calculus tries to instantiate and reduce the bigger term $f(\ldots x \ldots)$ for all possible cases. Clearly, this makes the search space explode. Therefore, we prefer to add new clauses that get hopefully subsumed by simple membership rules.

Actually, we have to add $n+1$ constants: one for individuals (Ω_0) and one for sets of each order i up to n (Ω_i). For variables x that appear in the head of a clause but not in the condition we have then to add a condition $x \in \Omega_j$ for the constant Ω_j corresponding with the sort of the variable. To summarise, we want to apply the following transformation Ξ to $\mathbf{P}$ and Σ, assuming $ls(t)$ is the least sort of t:

1. Let $\Xi(\Sigma)$ be a G^{n+1}-signature consisting of Σ extended by new constants $\Omega_0, \ldots, \Omega_n$, s.t. $\Omega_i : \rightarrow s_{i+1}$ for $i \in [0..n]$.
2. $\Xi(EX\ t) = (t \in \Omega_i)$, s.t. $ls(t) \leq s'_i$ where $i \in [0..n]$ and $\Xi(L) = L$ for all other atoms L.
3. For all clauses $L \Leftarrow \Gamma$ in $\mathbf{P}$, $\Xi(L \Leftarrow \Gamma) = \Xi(L) \Leftarrow \Gamma'$, where Γ' is $(\Xi(L') \mid L' \in \Gamma)$ plus $(x \in \Omega_i \mid x \in \mathcal{V}ar(L)$ and x does not occur in Γ and $sort(x) \leq s'_i)$.
4. $\Xi(\mathbf{P}) = \{\Xi(C) \mid C \in \mathbf{P}\} \cup \{u \in \Omega_i \Leftarrow \Gamma \mid (L[u] \Leftarrow \Gamma) \in \{\Xi(C) \mid C \in \mathbf{P}\}, u \notin \mathbf{X}$ and $ls(u) \leq s'_i\}$.[10]

Furthermore, let Ξ(**PartialReflex**) be the following rule:

$$\frac{t \in \Omega_i}{t = t} \quad \text{if } i \in \{0, 1\}$$

Let **OSGL** stand for **OSL** with Ξ(**PartialReflex**) instead of **Reflexivity**. Remark that Ξ is not bijective, since the clauses added in point 3 may be confused with already existing ones. However, we may define a function Θ being almost an inverse function for Ξ:

1. $\Theta((\mathbf{S}, \mathbf{F}))$ is the G^n-signature Σ' defined as to Σ, but without $\Omega_0, \ldots, \Omega_n$.
2. $\Theta(t \in \Omega_i) = (EX\ t)$ for $i \in [0..n]$, $\Theta(L) = L$ for all other atoms L and $\Theta(L \Leftarrow L_1, \ldots, L_m) = (\Theta(L) \Leftarrow \Theta(L_1), \ldots, \Theta(L_m))$.
3. $\Theta(\mathbf{P}) = \{\Theta(C) \mid C \in \mathbf{P}\}$.

Clearly, Θ is the inverse of Ξ on ground atoms. Note also that we also have to be careful with variables that disappear or are replaced in a clause (by simplification or constraint resolution), since this is only allowed if there was a typing condition on the variable and the replacement can be proven to be of this type. But this happens to be an invariant if all variables in the initial $\mathbf{P}$ have typing conditions as this is the case after the syntactic extension of clauses as described above. Recall that $\mathbf{P}$-$SUBST_\Sigma(\mathbf{V})$ stands for the set of $\mathbf{P}$-conform Σ-substitutions over $\mathbf{V}$, i.e. those with provably defined images, before we continue.

Lemma 4.2.3 *For all G^n-presentations* $\mathbf{P}$, *all G^n-interpretations* $\mathbf{A}$, $\mathbf{A} \models_{G^n} \mathbf{P}$ *iff* $\mathbf{A} \models_{G^n} \Theta(\Xi(\mathbf{P}))$.

[10] Most of these clauses should be immediately subsumed by similar ones derived from simple membership rules.

Proof: Remark that the only difference between a G^n-logic clause $L \Leftarrow \Gamma$ and $\Theta(\Xi(L \Leftarrow \Gamma))$ is that the latter clause may have additional conditions of the shape $EX\ x$ for variables x that occur in L but not in Γ. These conditions are always fulfilled, since $(\mathbf{A}, \mathbf{X})$-variable assignments δ for G^n-logics are functions $\delta : \mathbf{X} \to C^{\mathbf{A}}$, where $C^{\mathbf{A}}$ is the carrier of $\mathbf{A}$ and $EX\ x$ has to be interpreted as $\delta(x) \in C^{\mathbf{A}}$. The lemma now follows from the fact that for all for G^n-clauses C, $\mathbf{A} \models_{G^n} C$ is just a shorthand for $\mathbf{A}, \delta \models_{G^n} C$ for all variable assignments δ. □

We come to the main theorem of this section.

Theorem 4.2.4 *Let Σ be a G^n-signature, L be a ground Σ-atom and $\mathbf{P}$ be a G^n-presentation. Then $\mathbf{P} \vdash_{GNL} L$ iff $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(L)$.*

Proof: In order to reduce the number of different cases to be considered, we prove $\mathbf{P}^+ \vdash_{GNL^-} L$ iff $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(L)$, which implies this theorem by Theorem 4.2.2. Let τ be a ground substitution in $\mathbf{P}\text{-}SUBST_\Sigma(\mathbf{V})$ and $L' \Leftarrow \Gamma$ a Σ-clause.

From left to right, we prove that $\mathbf{P}^+ \vdash_{GNL^-} L' \Leftarrow \Gamma$ in m_1 steps and $\mathbf{P}^+ \vdash_{GNL^-} \tau(\Gamma)$ in m_2 steps, both using only variables in $\mathbf{V}$, implies that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(L'))$ and $\Xi(\mathbf{P}^+) \vdash_{OSGL} \tau(t) \in \Omega$ for any subterm[11] t of L' by induction on $m = m_1 + m_2 + m_3$, where m_3 is the number of steps needed for the conformity proof of τ. We distinguish the different cases for the last applied deduction rule[12] in $\mathbf{P}^+ \vdash_{GNL^-} L' \Leftarrow \Gamma$:

- **WellDef**: We know that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \tau(t) \in \Omega$ by the induction hypothesis. Hence $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(EX\ \tau(t))$ since $\Xi(EX\ \tau(t))$ is $\tau(t) \in \Omega$.
- **PartialReflex**: We know that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(EX\ \tau(t))$, where $\Xi(EX\ \tau(t))$ is $\tau(t) \in \Omega$, by the induction hypothesis and hence we get $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(t = t))$ by Ξ(**PartialReflex**). The result for subterms of t is subsumed by the induction hypothesis.
- **Axioms**: We know that $L' \Leftarrow \Gamma$ is in $\mathbf{P}^+$. Furthermore, $\mathbf{P}^+ \vdash_{GNL^-} \tau(\Gamma)$ holds and also $\mathbf{P}^+ \vdash_{GNL^-} (EX\ \tau(x))_{x \in \mathcal{D}om(\tau)}$. Hence, by induction hypothesis we get $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(\Gamma), (\tau(x) \in \Omega)_{x \in \mathcal{D}om(\tau)})$. Now, we can apply **Cut** followed by **Substitutivity** in order to get $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(L'))$ and $\tau(t) \in \Omega$ by definition of Ξ. Remark that the additional premises in $\Xi(L' \Leftarrow \Gamma)$ w.r.t. $L' \Leftarrow \Gamma$ are of the form $x \in \Omega$ and are covered by the proofs of $\tau(x) \in \Omega$ for all $x \in \mathcal{D}om(\tau)$, since τ is a ground substitution for all $x \in \mathbf{V}$. Remark furthermore that these proofs justify $u \notin \mathbf{X}$ in the definition of $\Xi(\mathbf{P})$.
- **SubstConform**: Let $\sigma(L) \Leftarrow \sigma(\Gamma)$ be the conclusion. We know that $\mathbf{P}^+ \vdash_{GNL^-} \tau(\sigma(\Gamma))$ and hence by induction hypothesis that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(\sigma(L'))), \tau(\sigma(t)) \in \Omega$. Remark that $\mathbf{P}$-conform substitutions are closed under composition and that the conformity proof of σ is assumed to be part of the $\mathbf{GNL}^-$ proof, i.e. the necessary steps are included in m_1.
- **Cut**: Let $L \Leftarrow \Gamma, L'$ and $L' \Leftarrow \Gamma'$ be the premises and $L \Leftarrow \Gamma, \Gamma'$ be the conclusion. We know that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(\Gamma, \Gamma')), (\tau(x) \in \Omega)_{x \in \mathcal{D}om(\tau)}$ and hence $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(L'))$ by the induction hypothesis and **Cut**. Consequently, we also know that

[11] Let Ω stand for the right Ω_i in the following, depending on the least sort of t. This negligence is harmless in the following and simplifies the proof.

[12] Remark that **Axioms** gives the base case $m = 1$.

$\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(\Gamma, L'))$ and once again by induction hypothesis and **Cut**, we get $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(L))$. The subterm case is analogous.

- **Paramodulation**: Let $L[s] \Leftarrow \Gamma$ and $(s = t) \Leftarrow \Gamma'$ be the premises and $L[t] \Leftarrow \Gamma, \Gamma'$ be the conclusion. We know that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(\Gamma, \Gamma')), (\tau(x) \in \Omega)_{x \in \mathcal{D}om(\tau)}$. Therefore, we get $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(L[s], s = t))$ by the induction hypothesis and **Cut**. Consequently, by **Paramodulation**, we get $\Xi(\mathbf{P}^+) \vdash_{OSGL} \Xi(\tau(L[t]))$. The subterm case is analogous.

Let ρ be an arbitrary ground Σ'-substitution. From right to left, we prove that whenever $\Xi(\mathbf{P}^+) \vdash_{OSGL} L' \Leftarrow \Gamma$ in m_1 steps and $\Xi(\mathbf{P}^+) \vdash_{OSGL} \rho(\Gamma)$ in m_2 steps, both using only variables in $\mathbf{V}$, then also $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(L'))$ by induction on $m = m_1 + m_2$. First of all remark that if for all clauses in $\Xi(\mathbf{P}^+)$ all variables of the conclusion occur in the premise, then this property is preserved under deduction. We distinguish the different cases for the last applied deduction rule[13] in $\Xi(\mathbf{P}^+) \vdash_{OSGL} L' \Leftarrow \Gamma$:

- Ξ(**PartialReflex**): Let $t \in \Omega$ be the premise and $t = t$ be the conclusion. We know by the induction hypothesis that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(t \in \Omega))$, where $\Theta(\rho(t \in \Omega))$ is $EX\ \rho(t)$. So we can apply **PartialReflex** to obtain $\mathbf{P}^+ \vdash_{GNL-} \rho(t = t)$, i.e. $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(t = t))$, since t must be ground due to the fact that $t \in \Omega$ has no premise and the above observation that no variable occurs in the conclusion of a clause without being also in the premise.
- **Axioms**: Let $L' \Leftarrow \Gamma$ be the conclusion, i.e. $(L' \Leftarrow \Gamma) \in \Xi(\mathbf{P}^+)$. Remark that by construction of Ξ, there must be a clause $L'' \Leftarrow \Gamma''$ in $\mathbf{P}^+$, s.t. $\Gamma'' \subseteq \Theta(\Gamma)$ and either $\Theta(L') = L''$ or L' is of the form $t \in \Omega$ for some subterm t of L''. We know that $\Xi(\mathbf{P}^+) \vdash_{OSGL} \rho(\Gamma)$. Hence, $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(\Gamma))$ by the induction hypothesis. Consequently, we can apply **Cut** after **Axioms** and **SubstConform** with $\rho_{|_{\mathcal{V}ar(L'' \Leftarrow \Gamma'')}}$, in order to get $\mathbf{P}^+ \vdash_{GNL-} \rho(L'')$. Remark that the conformity proof of $\rho_{|_{\mathcal{V}ar(L'' \Leftarrow \Gamma'')}}$ is covered by the proofs of $\Theta(\rho(\Gamma))$ and **WellDef**, since all variables in L'' also occur in Γ by definition of Ξ. Now, either $\Theta(L') = L''$ and therefore $\Theta(\rho(L')) = \rho(L'')$ or L' is of the form $t \in \Omega$ for some subterm t of L'', i.e. we can get $\Theta(\rho(L'))$ (which is $(EX\ t)$) from $\mathbf{P}^+ \vdash_{GNL-} \rho(L'')$ by **WellDef**.
- **Substitutivity**: Let $\sigma(L') \Leftarrow \sigma(\Gamma)$ be the conclusion and $L' \Leftarrow \Gamma$ the premise. We know that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(\sigma(\Gamma)))$ and therefore $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(\sigma(L')))$ by the induction hypothesis on $L' \Leftarrow \Gamma$.
- **Cut**: Let $L \Leftarrow \Gamma, L'$ and $L' \Leftarrow \Gamma'$ be the premises and $L \Leftarrow \Gamma, \Gamma'$ be the conclusion. We know that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(\Gamma, \Gamma'))$ and hence we get by the induction hypothesis that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(L'))$, i.e. once again by induction hypothesis that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(L))$.
- **Paramodulation**: Let $L[s] \Leftarrow \Gamma$ and $(s = t) \Leftarrow \Gamma'$ be the premises and $L[t] \Leftarrow \Gamma, \Gamma'$ be the conclusion. We know that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(\Gamma, \Gamma'))$ and hence we get by the induction hypothesis for both premises that $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(L[s], s = t))$. Consequently, we get $\mathbf{P}^+ \vdash_{GNL-} \Theta(\rho(L[t]))$ by **Paramodulation**.

The Theorem now follows from the fact that Θ is the inverse function of Ξ on ground atoms. □

[13] Remark again that **Axioms** gives the base case $m = 1$.

2.4 From Predicates to Propositional Functions

A last commonly used step for superposition calculi is the replacement of atoms $p(t_1, \ldots, t_n)$ by equations of the form $f_p(t_1, \ldots, t_n) = tt$. We prove that this is proof theoretically equivalent for all equations in s'_i, $i \in [0..n]$, if we put the propositional functions f_p in s'_{n+1}.

Theorem 4.2.5 *Let $\Sigma = (\mathbf{S}, \leq_S, \mathbf{F}, \mathbf{R})$ be a R^n-or G^n-signature and $\mathbf{P}$ be a Σ-R^n-or Σ-G^n-presentation. If $\Sigma' = (\mathbf{S}', \leq_{S'}, \mathbf{F}', \mathbf{R}')$ with $\mathbf{F}' = \mathbf{F} \cup \{f_p \mid p \in \mathbf{R}\}$, where the codomain of all f_p is $s_{n'}$ and the domains are those of p, $\mathbf{R}' = \{=\}$ is a $R^{n'}$- or $G^{n'}$-signature with $n' = n + 1$. Let $\mathbf{P}^=$ be $\mathbf{P}$ where every non-equational atoms $p(t_1, \ldots, t_n)$ is replaced by an equation of the shape $f_p(t_1, \ldots, t_n) = tt$.*

Then, for all ground Σ-atoms L, $\mathbf{P} \vdash_{RNL} L$ iff $\mathbf{P}^= \vdash_{RNL} L^=$ and $\mathbf{P} \vdash_{GNL} L$ iff $\mathbf{P}^= \cup \{tt = tt\} \vdash_{GNL} L^=$, respectively, where $L^=$ is obtained from L by replacing non-equational atoms $p(t_1, \ldots, t_n)$ by equations of the shape $f_p(t_1, \ldots, t_n) = tt$.

Proof: As for *EX*-elimination, we reuse the previous transformations for the replacement of set theoretic deduction rules. In the case of **RNL**, we know by Theorem 4.2.1 that $\mathbf{P} \vdash_{RNL} L$ iff $\mathbf{P}^+ \vdash_{OSL} L$. Hence, it is sufficient to show that $\mathbf{P}^+ \vdash_{OSL} L$ iff $(\mathbf{P}^+)^= \vdash_{OSL} L^=$.

We proceed by induction on the size of the deduction tree. From left to right, we prove that $\mathbf{P}^+ \vdash_{OSL} L \Leftarrow \Gamma$ in m steps implies that $(\mathbf{P}^+)^= \vdash_{OSL} L^= \Leftarrow \Gamma^=$. The proof is by induction on m. We distinguish the different cases for the last applied deduction rule[14] in $\mathbf{P}^+ \vdash_{OSL} L \Leftarrow \Gamma$:

- **Reflexivity**: The result is equational and hence there is nothing to change.
- **Axioms**: By construction of $(\mathbf{P}^+)^=$.
- **Substitutivity,Cut,Paramodulation**: Apply the same rule to the induction hypothesis.

From right to left, we prove that $(\mathbf{P}^+)^= \vdash_{OSL} L^= \Leftarrow \Gamma^=$ in m steps implies that $\mathbf{P}^+ \vdash_{OSL} L \Leftarrow \Gamma$, where $L \Leftarrow \Gamma$ is obtained from $L^= \Leftarrow \Gamma^=$ by forgetting all literals of the form $tt = tt$ in $\Gamma^=$ and replacing any equation of the form $f_p(t_1, \ldots, t_n) = tt$ by $p(t_1, \ldots, t_n)$, except when $L^=$ is of the form $tt = tt$, in which case we get $\mathbf{P}^+ \vdash_{OSL} tt = tt$ by reflexivity. The proof is again by induction on m. We distinguish the different cases for the last applied deduction rule[15] in $(\mathbf{P}^+)^= \vdash_{OSL} L^= \Leftarrow \Gamma^=$:

- **Reflexivity,Substitutivity,Cut**: Apply the same rule, except for tt.
- **Axioms**: By construction of $\mathbf{P}^=$.
- **Paramodulation**: We have to distinguish three cases: if $s = t$ has the shape $tt = tt$, $f_p(t_1, \ldots, t_n) = tt$ or anything else. In the first two cases, the replacement has to be on top because of the fact that tt is of sort $s_{n'}$ and there is no function taking arguments of that sort in Σ'. The case $tt = tt$ is trivial, i.e. the claim follows from the induction hypothesis, since the step is redundant. In the second case, the result is a clause with the head $tt = tt$, which we excluded in our assumptions for this induction. In the third case, we may apply **Paramodulation** to the induction hypothesis in order to get the claim.

[14] Remark that **Axioms** and **Reflexivity** give the base case $m = 1$.

[15] Remark again that **Axioms** and **Reflexivity** give the base case $m = 1$.

The case of **GNL** is completely analogous, except that at the places where reflexivity is used for tt, we have to use the equation $tt = tt$, which is added to $\Xi(\mathbf{P}^+)$ in order to give $\Xi(\mathbf{P}^+)^=$, and the claim to be proved is $\Xi(\mathbf{P}^+) \vdash_{OSGL} L$ iff $\Xi(\mathbf{P}^+)^= \vdash_{OSGL^=} L^=$, where $\mathbf{OSGL}^=$ is obtained from **OSGL** through replacement of $\Xi(\mathbf{PartialReflex})$ by a similar rule $\Xi(\mathbf{PartialReflex})^=$ where $t \in \Omega_i$ in the premise is replaced by $(t \in \Omega_i)^=$, which is written $t : \Omega_i = tt$ in the following. □

Let the inverse transformation of $.^=$, which was defined and used in the right-to-left direction of the proof, be written $.^{rel}$.

3 Clausal Theorem Proving for R^n-and G^n-Logic

In this section, we use the transformations from Section 4.2 in order to define saturation calculi for R^n-and G^n-logics. Hence, we work now implicitely with $\mathbf{P}^+$ in case of R^n-logic and $\Xi(\mathbf{P}^+)$ in the case of G^n-logic. The first step towards a saturation calculus in both the R^n- and G^n-case, is then to replace membership formulas $s \in t$ by equations of the form $s : t \simeq tt$, where the colon and the tt operator are new and of sort $s_{n'}$, where $n' = n + 1$ for R^n-logics and $n' = n + 2$ for G^n-logics, for all arguments (i.e. we implicitly work with an extended type system), in order to meet the formalism of Nieuwenhuis and Rubio. More formally, $arity(tt) = \{tt : \rightarrow s_{n'}\}$ and $arity(\cdot : \cdot) = \{(\cdot : \cdot : s''s''' \rightarrow s_{n'}) \mid (\in : s''s''') \in arity(\in)\}$. Remark that tt is a constant that is by definition always defined in the case of G^n-logics. This can more formally be expressed by adding $tt = tt$ as fact to the current set of clauses or as initial sequent. Theorem 4.2.5 guarantees us the soundness of this step.

Furthermore, we adopt the term generated semantics for R^n-and G^n-logics. Hence, in R^n-logics, a constrained clause $C \; [\![T]\!]$ of the form $L \Leftarrow \Gamma \; [\![T]\!]$ is satisfied by an interpretation I, written $I \models_{R^n} C \; [\![T]\!]$, if all ground Σ-instances $C\sigma$ of $C \; [\![T]\!]$ are satisfied by I. A ground instance $L\sigma \Leftarrow \Gamma\sigma$ is satisfied by I, if it is true in the initial model, i.e. if $L\sigma$ can be deduced whenever $\Gamma\sigma$ can be deduced from $\mathbf{P}$ using the deduction system **RNL** in Section 2.3, assuming all sorts to be non-empty.

In the case of G^n-logics, we have a premise for the reflexivity rule. An interpretation I is a G^n-congruence on ground terms, i.e. if $(t[u], t[u]), (u, v) \in I$, then $(t[v], t[v])$ is in I. Remark that this is a weaker requirement than before, since not all ground terms may occur in I. Hence, in G^n-logics, a constrained clause $C \; [\![T]\!]$ of the form $L \Leftarrow \Gamma \; [\![T]\!]$ is satisfied by I, written $I \models_{G^n} C \; [\![T]\!]$, if all ground Σ-instances $C\sigma$ of $C \; [\![T]\!]$, s.t. σ is a $\mathcal{P}$-conform substitution, are satisfied by I.

This restriction of the ground instances is in fact the only difference to R^n-logics, but it leads to changes whenever we want to resolve premises in clauses. We call the corresponding satisfaction relation in the following G^n-satisfaction. As before, a ground instance $L\sigma \Leftarrow \Gamma\sigma$ is satisfied, if it is true in the initial model, i.e. if $L\sigma$ can be deduced whenever $\Gamma\sigma$ can be deduced from $\mathbf{P}$ using the deduction system **GNL** in Section 2.4, assuming all sorts to be non-empty.

3.1 A Saturation Calculus for R^n-Logics

Let $\mathbf{P}$ be an R^n-presentation and $\mathbf{P}^+$ be $\mathbf{P}$ extended by the axioms of choice, extensionality and bijectivity for *ref*/*deref* in form of Horn clauses (as defined in Section 4.2.1). From Theorem 4.2.1 we know that **RNL** deduction (cf. Section 2.3 from $\mathbf{P}$ is equivalent to **OSL** deduction 2.2 from $\mathbf{P}^+$. Let $\mathcal{R}$ be $\mathcal{NR}$ from Section 4.1.3 with a constraint solver for order-sorted unification. Since

$\mathbf{P}^+$ only contains sort-preserving Σ-equations as literals, we come immediately to the following conclusion for $\mathcal{R}$:

Theorem 4.3.1 *Let $P_0, P_1, \ldots$ be a fair R^n-theorem proving derivation, where $P_0 = \mathbf{P}^+$ is a set of clauses with empty constraints. Then P_0 is consistent if and only if the empty clause belongs to some P_j. Furthermore, if the empty clause does not belong to P_∞, then $R^*_{P_\infty} \not\models_{R^n} P_0$.*

Proof: Literally the same as for $\mathcal{HCL}$-logics (cf. Lemma 4.1.14 and Theorem 4.1.15). □

Remark that we do not need a weakening rule as in [Walther, 1983], since all equations are sort-preserving. Note furthermore that multiset expressions for equations are defined literally in the same way as in Section 4.1.2. The associated constraint solver is shown in Figure 4.2, which is the algorithm for order-sorted unification in regular signatures from [Jouannaud et Kirchner, 1991], whose soundness and completeness follows from the work of M. Schmidt-Schauss [Schmidt-Schauß, 1989] (cf. also [Waldmann, 1989b] for a similar system for order-sorted unification). Remark that *index* is a function assigning a unique index for each variable[16] s.t. $index(x) < index(y)$ if $sort(x) < sort(y)$. Note furthermore that the **Abstract**-rule can be deterministic if the signature is also coregular [Goguen et Meseguer, 1988; Goguen et Meseguer, 1992], but in general it is not. Hence, there may be a disjunction of constraints, corresponding with different solutions of the equational constraints. In order to keep the framework simple we do not extend our syntax for this purpose but simply assume that such a constraint is normalised into its disjunctive normal form and then several copies of the same clause, each one with a different conjunct from the normal form, are created.

In order to illustrate that the restriction to sort-preserving is necessary, consider the following example:

Example 4.3.2 *Remember the example from the introduction illustrating the problems of order-sorted completion in the case of non-sort-decreasing equations:*

$\Sigma = (\{A, B\}, \{f, a, b\})$, *s.t.* $A \leq B$, $arity(a) = \{a \ : \rightarrow A\}$, $arity(b) = \{b \ : \rightarrow B\}$, $arity(f) = \{(f : A \rightarrow A), (f : B \rightarrow B)\}$ *and* $\mathbf{P}$ *is defined as:*

$$\{(\Leftarrow f(b) = b), (f(x :: A) = x), (a = b)\}$$

Remark that $\mathbf{P}$ *is inconsistent, since* $\mathbf{P} \models_{HCL} f(b) = f(a) = a = b$, *which is in contradiction to the first clause. However, if* $f \succ a \succ b$, *then there is no superposition possible and hence,* $\mathbf{P}$ *should be saturated and thus consistent, according to Theorem 4.3.1 if we forget about the sort preservation condition for R^n-logic. Remark that* $R_{\mathbf{P}}$ *is* $\{(f(a) \rightarrow a), (a \rightarrow b)\}$ *and therefore* $R^*_{\mathbf{P}} \models_{HCL} \{(f(x :: A) = x), (a = b)\}$, *but not* $R^*_{\mathbf{P}} \models_{HCL} irred_{R_{\mathbf{P}}}(\mathbf{P})$, *since* $R^*_{\mathbf{P}} \not\models_{HCL} (\Leftarrow f(b) = b)$, *i.e.* $R^*_{\mathbf{P}} \models_{HCL} f(b) = b$ *(by the unjoinable peak* $b \leftarrow_{R_P} a \leftarrow_{R_P} f(a) \rightarrow_{R_P} f(b)$*) and the empty clause is not in* $\mathbf{P}$. *This is due to the lack of confluence for* $R_{\mathbf{P}}$ *(cf. the corresponding case in Lemma 4.1.14).*

Note also that due to non-sort-decreasingness of $a \rightarrow b$, *we would need superposition into variable occurrences of* $f(x :: A) = x$, *in order to derive the empty clause. The principal problem that makes the proof of Lemma 4.1.14 and therefore Theorem 4.3.1 fail for this case is that* $f(b)$ *is not an instance of* $f(x :: A)$, *although* $f(a)$ *is and* $f(a) = f(b)$ *holds. This is analogous to the incompleteness of the usual definition of critical pairs for order-sorted algebras, if non-sort-decreasing rules are present. Remark last but not least that sort preservation implies sort decreasingness for both directions of an equation, i.e. the above problems are completely avoided in this case.*

[16] Remember that $\mathbf{X}$ is denumerable.

Delete $\dfrac{[\![t =^? t, T]\!]}{[\![T]\!]}$

Decompose $\dfrac{[\![f(t_1,\ldots,t_m) =^? f(t'_1,\ldots,t'_m), T]\!]}{[\![t_1 =^? t'_1,\ldots,t_m =^? t'_m, T]\!]}$

Conflict $\dfrac{[\![f(t_1,\ldots,t_m) =^? g(t'_1,\ldots,t'_{m'}), T]\!]}{\bot}$ if $f \neq g$ or $m \neq m'$.

Coalesce $\dfrac{[\![x =^? y, T]\!]}{[\![x =^? y, T\{x \mapsto y\}]\!]}$ if $index(y) < index(x)$ and $x \in \mathcal{V}ar(T)$.

Check* $\dfrac{[\![x_1 =^? s_1[x_2],\ldots,x_n =^? s_n[x_1], T]\!]}{\bot}$

Merge $\dfrac{[\![x =^? t', x =^? t'', T]\!]}{[\![x =^? t', t' =^? t'', T]\!]}$ if $0 < |t'| \leq |t''|$.

Abstract $\dfrac{[\![x =^? f(t_1,\ldots,t_n), T]\!]}{[\![x =^? f(x_1,\ldots,x_n), x_1 =^? t_1,\ldots,x_n =^? t_n T]\!]}$
if $x :: s, \exists s' \leq s, (f : s_1,\ldots,s_n \to s') \in arity(f)$
and $\forall i \in [1..n], x_i :: s_i$.

Remove $\dfrac{[\![x =^? y, T]\!]}{\bot}$ if $A \not\leq B, B \not\leq A,$
$\nexists C : C \leq A$ and $C \leq B$.

Intersect $\dfrac{[\![x =^? y, T]\!]}{[\![x =^? z, y =^? z, T]\!]}$ if z is a new variable of sort C, $A \not\leq B$, $B \not\leq A$,
$C \leq A$ and $C \leq B$.

Figure 4.2: A constraint solver for order-sorted unification

The problem here is that $\succ$ may not be compatible with the subsort relation and therefore the order-sorted unification algorithm in Figure 4.2 does not return all instances necessary for equational deduction. However, when we have sort-preserving equations as in R^n-logic, then Lemma 2.2.17 guarantees us that all terms in an equivalence class generated by $\mathbf{P}$ have the same minimal sort and hence the set of all syntactic instantiations of a clause coincides with the set of those instantiations needed for equational deduction.

3.2 A Saturation Calculus for G^n-Logics

Let $\mathbf{P}$ be a G^n-presentation. Before we start, recall the remarks of the beginning of this section, i.e. that we work with $\Xi(\mathbf{P}^+) \cup \{tt \simeq tt\}$ and the extension of the signature by tt and : in the following. Additionally, we have to change the multiset expressions for equations, due to the elimination of existential formulas. The multiset expression of an equation $t \simeq t'$ in a G^n-logic clause $L \Leftarrow \Gamma$ depends on the sort of t (which is equal to t' since all clauses in G^n-logical programs

are sort preserving).

Before we give the calculus, we refine the definition of multiset expressions in order to cope more easily with interdependant set definitions. The problem is illustrated by the following example:

Example 4.3.3 *Let the R^n-or G^n-presentation* **P** *be defined as follows:*

$$\mathbf{P} = \left\{ \begin{array}{l} (pop(push(x,y)) : St \simeq tt \Leftarrow x : E \simeq tt, y : St \simeq tt), \\ (top(push(x,y)) : E \simeq tt \Leftarrow x : E \simeq tt, y : St \simeq tt), \end{array} \right\}$$

If we want to prove with an LPO that $pop(push(x,y)) : St \simeq tt$ is greater than $x : E \simeq tt$ and $y : St \simeq tt$ and at the same time that $top(push(x,y)) : E \simeq tt$ is greater than $x : E \simeq tt$ and $y : St \simeq tt$, then, in the first case, we needed $St \succ E$ and in the second $E \succ St$.

However, this may be solved using multisets: $\{pop(push(x,y)), St\}$ and $\{top(push(x,y)), E\}$ can be proven to be greater than both $\{x, E\}$ and $\{y, St\}$, using the same, total precedence $top \succ pop \succ push \succ St \succ E$, since the set terms St and E are then subsumed by $pop(push(x,y))$ and $top(push(x,y))$, respectively. Hence, the solution consists in considering equations of the shape $f_p(t_1, \ldots, t_n) \simeq t'$, s.t. f_p is a propositional function introduced by $.^=$, as the multiset $\{f_p, t_1, \ldots, t_n\}$, where we implicitely think of $A(t)$ instead of $t : A$. Remark that this allows us furthermore to use set variables in membership conditions, as this is sometimes necessary for defining parametric sets – think of E as a variable and replace St in the above example by $St(E)$. The same proof is still possible.

This motivates the following definition:

Definition 4.3.4 *Assume $n' = n + 2$ is the maximal nesting depth of a set after eliminating existential formulas using Ξ and replacing relations p by propositional functions f_p using $.^=$.*

Let $e(t)$ be $\{\Omega_i, t\}$ and $e_2(t)$ be $\{\Omega_i, t, \Omega_i, t\}$, if t is a term of sort lower than s'_i with $i < n'$. Otherwise, i.e. the top symbol is a propositional function f_p, let $e(f_p(t_1, \ldots, t_n))$ be $\{f_p, t_1, \ldots, t_n\}$ and $e_2(f_p(t_1, \ldots, t_n))$ be $\{f_p, t_1, \ldots, t_n, f_p, t_1, \ldots, t_n\}$, except for terms of the shape $t : A$, where $e(t : A)$ is $\{A, t\}$ and $e_2(t : A)$ is $\{A, t, A, t\}$.

*If $t \simeq t'$ is an equation in the conclusion of a clause, then the (G^n-)*multiset expression of an equation *is defined as $\{e(t), e(t')\}$, otherwise, in the premise, it is $\{e_2(t), e_2(t')\}$.*

Remark that this definition can also be used with the saturation calculus for R^n-logic, once we have added the constants Ω_i with clauses $(x :: s'_i) \in \Omega_i$ for $i \in [1..n]$. Alternatively, one might replace (Ω_i, t) by t in the above definition of e and e_2, which makes the signature extension by the Ω_i superfluous. It has the advantage that when one decides to use LPO, it is easier to find a precedence adapted to conditional membership rules, since instead of the colon operator one can use set term constructors in the precedence. Furthermore, only terms appearing in the multiset expressions constructed above have to be comparable, i.e. there is no need to respect the colon operator, since it is completely eliminated. The following example illustrates this in a simple way:

Example 4.3.5 *Assume we want to axiomatise lists, ordered sets and multisets at once. Let $\mathbf{F} = \{0, s, <, \leq, \epsilon, ., Nat, List, Omset, Oset\}$, where $Oset$ intuitively stands for the set of ordered sets, $Omset$ for ordered multisets, $List$ for lists of natural numbers (Nat). Analogously, 0 stands for zero, s for successor, $<, \leq$ are the standard orderings on natural numbers, ϵ is the empty*

list and "." is the concatenation constructor. Let furthermore **P** *be the following R^n-(or G^n-) presentation:*

$$\left\{\begin{array}{l}(0 \in Nat),\\ (s(x) : Nat \simeq tt \Leftarrow x : Nat \simeq tt),\\ \hline (0 \leq x \simeq tt \Leftarrow x : Nat \simeq tt),\\ (s(x) \leq s(y) \simeq tt \Leftarrow x \leq y \simeq tt),\\ (0 < s(x) \simeq tt \Leftarrow x : Nat \simeq tt),\\ (s(x) < s(y) \simeq tt \Leftarrow x < y \simeq tt, x : Nat \simeq tt, y : Nat \simeq tt),\\ \hline (x : List \simeq tt \Leftarrow x : Omset \simeq tt),\\ (x : Omset \simeq tt \Leftarrow x : Oset \simeq tt),\\ \hline (\epsilon : Oset \simeq tt),\\ (x.\epsilon : Oset \simeq tt \Leftarrow x : Nat \simeq tt),\\ (x.x'.y : Oset \simeq tt \Leftarrow x < x' \simeq tt, x'.y : Oset \simeq tt,\\ \qquad x : Nat \simeq tt, x' : Nat \simeq tt, y : Oset \simeq tt),\\ \hline (x.x'.y : Omset \simeq tt \Leftarrow x \leq x' \simeq tt, x'.y : Omset \simeq tt,\\ \qquad x : Nat \simeq tt, x' : Nat \simeq tt, y : Omset \simeq tt),\\ \hline (x.y : List \simeq tt \Leftarrow x : Nat \simeq tt, y : List \simeq tt)\end{array}\right\}$$

P *is R^n-saturated for the LPO with the following precedence:*

$$List \succ Omset \succ Oset \succ . \succ \epsilon \succ \leq \succ < \succ Nat \succ s \succ 0 \succ tt$$

This can easily be verified since there are no superpositions into the conclusion and all clauses are decreasing, i.e. the head is always greater then any literal in the body w.r.t. the LPO defined above. Remark that if membership equations had to be compared syntactically, then there were no possibility to give an LPO, since $:\succ\leq$ and $\leq\succ:$ would have to be satisfied.

We may extract from the above example the following heuristic: If membership of a term t to a set s depends on a function f, then the top operator of s should be greater in the precedence then f, provided one uses LPO. This shows us in particular that membership in sets is seen here in a functional way in that $t \in f(s_1, \ldots, s_n)$ is treated as $(\chi_f(s_1, \ldots, s_n), t) \simeq tt$, where $\chi_f(s_1, \ldots, s_n)$, which can be seen as unary function $\chi'_{f(s_1,\ldots,s_n)}(x)$, is the characteristic function[17] of the set $f(s_1, \ldots, s_n)$. Following this observation, one could as well have chosen $f(s_1, \ldots, s_n, t) \simeq tt$ as translation of $t \in f(s_1, \ldots, s_n)$ instead of $t : f(s_1, \ldots, s_n) \simeq tt$, right from the beginning. We prefered to use the latter in order to have more homogeneity in our formalism and to allow for set variables in membership conditions, as, e.g., for $cons(x, y) \in list(z) \Leftarrow x \in z, y \in list(z)$.

The inference system

We consider the following set $\mathcal{G}$ of inference rules, which is an adaption of $\mathcal{NR}$ from Section 4.1.3 for partial functions:

1. *strict basic superposition into conclusion* :

$$\frac{s \simeq s' \Leftarrow \Gamma' \; [\![T']\!] \qquad t \simeq t' \Leftarrow \Gamma \; [\![T]\!]}{t[s']_u \simeq t' \Leftarrow \Gamma, \Gamma' \; [\![T, T', t|_u =^? s]\!]}$$

[17] Remember the remarks on intuitionistic set theory in the introduction.

where $t|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t|_u =^? s]\!]$ has a ground solution σ, s.t.:

(a) $t\sigma \succ t'\sigma, s\sigma \succ s'\sigma$ and $t\sigma \simeq t'\sigma \succ_E s\sigma \simeq s'\sigma$,

(b) $s\sigma \simeq s'\sigma$ is strictly maximal in $s\sigma \simeq s'\sigma \Leftarrow \Gamma'\sigma$ and

(c) $t\sigma \simeq t'\sigma$ is strictly maximal in $t\sigma \simeq t'\sigma \Leftarrow \Gamma$.

2. *strict basic superposition into premise* :

$$\frac{s \simeq s' \Leftarrow \Gamma' \; [\![T']\!] \qquad L \Leftarrow \Gamma, t \simeq t' \; [\![T]\!]}{L \Leftarrow t[s']_u \simeq t', \Gamma, \Gamma' \; [\![T, T', t|_u =^? s]\!]}$$

where $t|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t|_u =^? s]\!]$ has a ground solution σ, s.t.:

(a) $t\sigma \succ t'\sigma$ and $s\sigma \succ s'\sigma$,

(b) $s\sigma \simeq s'\sigma$ is strictly maximal in $s\sigma \simeq s'\sigma \Leftarrow \Gamma'\sigma$ and

(c) $t\sigma \simeq t'\sigma$ is maximal in $L\sigma \Leftarrow \Gamma\sigma, t\sigma \simeq t'\sigma$.

3. *G^n-equality resolution in conclusion* :

$$\frac{t \simeq t' \Leftarrow \Gamma' \; [\![T']\!]}{t : \Omega_i = tt \Leftarrow \Gamma' \; [\![T', t =^? t']\!]}$$

where the least sort of t is less or equal to s'_i, $i \in [0..n+1]$, and $[\![T', t =^? t']\!]$ has a ground solution σ, s.t. $t\sigma \simeq t'\sigma$ is maximal in $t\sigma \simeq t'\sigma \Leftarrow \Gamma'\sigma$.

4. *G^n-equality resolution in premise* :

$$\frac{L \Leftarrow t \simeq t', \Gamma' \; [\![T']\!]}{L \Leftarrow t : \Omega_i = tt, \Gamma' \; [\![T', T'']\!]}$$

where the least sort of t is less or equal to s'_i, $i \in [0..n+1]$, and $[\![T', t =^? t']\!]$ has a ground solution σ, s.t. $t\sigma \simeq t'\sigma$ is maximal in $L \Leftarrow t\sigma \simeq t'\sigma, \Gamma'\sigma$.

5. *G^n-tautology resolution in conclusion (subsumed by redundancy)* :

$$\frac{}{tt \simeq tt \Leftarrow \Gamma' \; [\![T']\!]}$$

6. *G^n-tautology resolution in premise* :

$$\frac{L \Leftarrow \Gamma', tt \simeq tt \; [\![T']\!]}{L \Leftarrow \Gamma' \; [\![T']\!]}$$

The constraint solver is the same as in Section 4.3.1 for R^n-logic. We start with the proof of soundness for $\mathcal{G}$ w.r.t. G^n-deduction of ground atoms, assuming that $\simeq$ is identified with $=$.

Theorem 4.3.6 *The inference system $\mathcal{G}$ is sound for G^n-deduction from any $P_0 = \Xi(\mathbf{P}^+) \cup \{tt \simeq tt\}$, where $\mathbf{P}$ is a G^n-presentation and $\Xi(\mathbf{P}^+)^=$ is defined as in Section 4.2.3 and 4.2.4, i.e.:*

$$\text{for all ground atoms } L, \mathbf{P} \models_{G^n} L \text{ whenever } \Xi(\mathbf{P}^+)^= \cup \{tt \simeq tt\} \vdash_G \Xi(L)^=$$

Proof: Instead of the above expression, we may prove the equivalent one below, because of Theorems 4.2.4 and 4.2.5 together with soundness and completeness of **GNL** (cf. Theorems 2.4.5 and 2.4.14):

for all ground atoms $L, \Xi(\mathbf{P}^+)^= \vdash_{OSGL^=} L^=$ whenever $\Xi(\mathbf{P}^+)^= \cup \{tt \simeq tt\} \vdash_G \Xi(L)^=$

We will reduce any proof of a ground atom with $\mathcal{G}$ into one using $\mathbf{OSGL}^=$. Clearly, **Paramodulation** subsumes *strict basic superposition into conclusion. G^n-equality resolution in premise* and *G^n-equality resolution in conclusion* are sound, since we can deduce $t = t$ if and only if we can deduce $EX\ t$ from a G^n-presentation, both using **GNL**. Now, since $EX\ t$, translates into $t \in \Omega_i$ by Ξ, we get the proof by Theorem 4.2.4. *G^n-tautology resolution in conclusion* is subsumed by $tt \simeq tt$ and *G^n-tautology resolution in premise* is a consequence of **Cut** and this equation.

There remains *strict basic superposition into premise.* We prove that $\mathbf{OSGL}^=$ together with this rule, forgetting about ordering constraints, proves the same ground atoms as $\mathbf{OSGL}^=$. This simplified (and therefore more general rule) has the following shape:

$$\frac{(s = s') \Leftarrow \Gamma',\ L' \Leftarrow t = t', \Gamma}{\sigma(L' \Leftarrow t[s']_u = t', \Gamma, \Gamma')} \quad \text{if } \sigma \text{ is a unifier of } s \text{ and } t|_u$$

Assume $\mathbf{P}'$ is $\Xi(\mathbf{P}^+)^=$, $\mathbf{OSGL}^+$ is the extended $\mathbf{OSGL}^=$ and L is any ground Σ-atom. This can then be formulated as:

$$\mathbf{P}' \vdash_{OSGL^=} L^= \text{ iff } \mathbf{P}' \vdash_{OSGL^+} L^=$$

Clearly, from left to right this is trivial. From right to left, let $\mathbf{V}$ be a set of variables, $L \Leftarrow \Gamma$ a Σ-clause and τ a ground Σ-substitution with $Dom(\tau) \subseteq \mathbf{V}$. We prove that whenever $\mathbf{P}' \vdash_{OSGL^+} L^= \Leftarrow \Gamma^=$ in m_1 steps and $\mathbf{P}' \vdash_{OSGL^+} \tau(\Gamma^=)$ in m_2 steps, both proofs only using variables in $\mathbf{V}$, then $\mathbf{P}' \vdash_{OSGL^=} \tau(L)^=$ by induction on $m = m_1 + m_2$. We distinguish the last applied rule, **Axioms** being the base case.

- $\Xi(\mathbf{PartialReflex})^=$: Let $(t \in \Omega_i)^=$, i.e. $t : \Omega_i = tt$, be the premise and $(t = t)^=$ be the conclusion. Then, we know that $\mathbf{P}' \vdash_{OSGL^=} \tau(t \in \Omega_i)^=$ by induction hypothesis, i.e. $\mathbf{P}' \vdash_{OSGL^=} \tau(t = t)^=$ also by $\Xi(\mathbf{PartialReflex})^=$.
- **Axioms**: Let $L^= \Leftarrow \Gamma^=$ be the conclusion. We know by the induction hypothesis that $\mathbf{P}' \vdash_{OSGL^=} \tau(\Gamma)^=$ and therefore we can apply **Cut** after **Substitutivity** and **Axioms** to get $\mathbf{P}' \vdash_{OSGL^=} \tau(L)^=$.
- **Substitutivity**: Let $\sigma(L)^= \Leftarrow \sigma(\Gamma)^=$ be the conclusion and $L^= \Leftarrow \Gamma^=$ the premise. Then we get $\mathbf{P}' \vdash_{OSGL^=} \tau(\sigma(L'))^=$ by the induction hypothesis.
- **Cut**: Let $L^= \Leftarrow \Gamma^=, (L')^=$ and $(L')^= \Leftarrow (\Gamma')^=$ be the premises and $L^= \Leftarrow \Gamma^=, (\Gamma')^=$ be the conclusion. We know by the induction hypothesis that $\mathbf{P}^= \vdash_{OSGL^=} \tau(\Gamma, \Gamma')^=$ and hence by the induction hypothesis over $(L')^= \Leftarrow (\Gamma')^=$ that $\mathbf{P}^= \vdash_{OSGL^=} \tau(L)^=$ and therefore by the induction hypothesis over $L^= \Leftarrow \Gamma^=, (L')^=$ that $\mathbf{P}^= \vdash_{OSGL^=} \tau(L)^=$.
- **Paramodulation**: Let $L[s]^= \Leftarrow \Gamma^=$ and $(s = t)^= \Leftarrow (\Gamma')^=$ be the premises and $L[t]^= \Leftarrow \Gamma^=, (\Gamma')^=$ be the conclusion. We know by the induction hypothesis that $\mathbf{P}^= \vdash_{OSGL^=} \tau(\Gamma, \Gamma')^=$ and hence by the induction hypothesis over $(s = t)^= \Leftarrow (\Gamma')^=$ and $L[s]^= \Leftarrow \Gamma^=$ that $\mathbf{P}^= \vdash_{OSGL^=} \tau(s = t, L[s])^=$ and therefore by **Paramodulation** that $\mathbf{P}^= \vdash_{OSGL^=} \tau(L[t])^=$.

- *strict basic superposition into premise*: Let $(s = s')^= \Leftarrow (\Gamma')^=$ and $(L')^= \Leftarrow (t = t', \Gamma)^=$ be the premises and $\sigma(L' \Leftarrow t[s']_u = t', \Gamma.\Gamma')^=$ be the conclusion, where σ is a unifier of $t|_u$ and s. Hence, we know by induction hypothesis that $\mathbf{P}' \vdash_{OSGL^=} \tau(\sigma(t[s']_u = t', \Gamma, \Gamma'))^=$, which gives us by the induction hypothesis over $(s = s')^= \Leftarrow (\Gamma')^=$ that $\mathbf{P}^= \vdash_{OSGL^=} \tau(\sigma(s = s'))^=$ and therefore by **Paramodulation** of $\tau(\sigma(s = s'))^=$ into $\tau(\sigma(t[s']_u = t'))^=$ also $\mathbf{P}^= \vdash_{OSGL} \tau(\sigma(t = t'))^=$. Now we can apply **Cut** in order to get $\mathbf{P}^= \vdash_{OSGL^=} \tau(\sigma(L'))^=$.

□

The following lemma is the analogue of Lemma 4.1.14 for $\mathcal{HCL}$-logics and the $\mathcal{NR}$-calculus. The difference lies in the subcases where the two members of the investigated equation are equal.

Lemma 4.3.7 *Let $P_0, P_1, \ldots$ be a fair G^n-theorem proving derivation, where $P_0 = \Xi(\mathbf{P}^+)^= \cup \{tt \simeq tt\}$, s.t. P_∞ does not contain the empty clause. Then $R^*_{P_\infty} \models_{G^n} irred_{R_{P_\infty}}(P_\infty)$.*

Proof: In order to derive a contradiction, assume that $C\sigma \in irred_{R_{P_\infty}}(P_\infty)$ is a minimal (w.r.t. $\succ_C$) instance of a clause $C[\![T]\!]$ in P_∞, s.t. $R^*_{P_\infty} \not\models_{G^n} C\sigma$. We distinguish the different cases for $C\sigma$:

1. $C\sigma$ is of the form $(t\sigma \simeq t'\sigma) \Leftarrow \Gamma$, s.t. $t\sigma \simeq t'\sigma$ is maximal in $C\sigma$:
   (a) If $t\sigma$ is $t'\sigma$, then $C\sigma$ is redundant, since either $t\sigma$ is tt and *G^n-tautology resolution in conclusion* can be applied, which generates a strictly smaller clause w.r.t. $\succ_C$ entailing $C\sigma$. Or otherwise $\{t\sigma : \Omega_i = tt \Leftarrow \Gamma\} \models_{G^n} (t\sigma \simeq t'\sigma) \Leftarrow \Gamma$ and the former can be derived from the latter using *G^n-equality resolution in conclusion.* Remark that it is strictly smaller w.r.t. $\succ_C$ since $t\sigma : \Omega_i \succ tt$ and therefore $(t\sigma = t'\sigma) \succ_E (t\sigma : \Omega_i = tt)$.
   (b) If $t\sigma \succ t'\sigma$, then the argumentation is literally identical to the one in the proof of Lemma 4.1.14, except that $\mathcal{HCL}$ and $\mathcal{NR}$ have to changed into G^n and $\mathcal{G}$.
2. $C\sigma$ is of the form $L' \Leftarrow \Gamma, t\sigma \simeq t'\sigma$, s.t. $t\sigma \simeq t'\sigma$ is maximal in $C\sigma$:
   (a) If $t\sigma$ is $t'\sigma$, then $C\sigma$ is redundant, since either $t\sigma$ is tt and therefore *G^n-tautology resolution in premise* can be applied, which generates a strictly smaller clause w.r.t. $\succ_C$, entailing $C\sigma$. Or otherwise $\{L \Leftarrow \Gamma, t\sigma : \Omega_i = tt\} \models_{G^n} (L \Leftarrow \Gamma, t\sigma \simeq t'\sigma)$ and the former, which is as before strictly smaller w.r.t. $\succ_C$, can be derived from the latter using *G^n-equality resolution in premise.*
   (b) If $t\sigma \succ t'\sigma$, then once again the argumentation is literally identical to the one for Lemma 4.1.14, except that $\mathcal{HCL}$ and $\mathcal{NR}$ have to changed into G^n and $\mathcal{G}$.

□

Theorem 4.3.8 *Let $P_0, P_1, \ldots$ be a fair G^n-theorem proving derivation, where P_0 is a set of clauses with empty constraints. Then P_0 is consistent if and only if the empty clause belongs to some P_j. Furthermore, if the empty clause does not belong to P_∞, then $R^*_{P_\infty} \models_{G^n} P_0$.*

Proof: Literally the same as for Theorem 4.1.15, except that $\mathcal{HCL}$ and $\mathcal{NR}$ have to changed into G^n and $\mathcal{G}$, as well as the references to Lemma 4.1.14 and Theorem 4.1.13 have to be replaced by ones to Lemma 4.3.7 and Theorem 4.3.6, □

Let us finish this section with an example for the use of $\mathcal{G}$:

Example 4.3.9 *Assume we want to define stacks as set St over elements of sort E. Let the G^n-presentation* $\mathbf{P}$ *be defined as follows:*

$$\mathbf{P} = \left\{ \begin{array}{l} (\epsilon : E = tt), \\ (empty : St = tt), \\ (pop(push(x,y)) : St = tt \Leftarrow x : E = tt, y : St = tt), \\ (top(push(x,y)) : E = tt \Leftarrow x : E = tt, y : St = tt), \\ (push(x,y) : St = tt \Leftarrow x : E = tt, y : St = tt), \\ (pop(push(x,y)) = y \Leftarrow x : E = tt, y : St = tt), \\ (top(push(x,y)) = x \Leftarrow x : E = tt, y : St = tt) \end{array} \right\}$$

For simplicity we ignore the transformation into $\Xi(\mathbf{P}^+)$ for the moment. Remark that top, pop are undefined on empty.
Let $pop \succ top \succ push \succ empty \succ \epsilon \succ E \succ St$ be the precedence for an LPO, using multiset expressions for equations as given in Definition 4.3.4.
We can eliminate the membership formulas for $pop(push(x,y))$ and $top(push(x,y))$ by simplification with the two equations.
Refuting $(\Leftarrow top(pop(push(x,y))) = top(y))$ then gives $(\Leftarrow top(y) = top(y))$ which has the solutions $\{y \mapsto push(\epsilon, \ldots push(\epsilon, empty)\ldots)\}$.
(And no more the empty substitution, as in the case of R^n-logic.)

Remark that for doing term rewriting with the saturated presentation, we have to respect partial reflexivity, i.e. for proving a condition of the shape $t \simeq t'$, we first have to normalise the two terms until they get equal, say $t \downarrow_{P_\infty} = t'' = t' \downarrow_{P_\infty}$, then we have to prove $t'' : \Omega_0 \simeq tt$. Only if both proofs can be done, equality and existence, then the condition is satisfied. In the case of rewriting with SDTRSs, this is of course also necessary for oriented conditions.

3.3 Adding Assertions

The idea in this section is to add a cache memory mechanism to the saturation calculus of the last section. We therefore add assertions. Constraints are widely known as restriction of the clause (see e.g. [Kirchner *et al.*, 1990; Nieuwenhuis et Rubio, 1992a]) and are written in double square brackets $[\![.]\!]$ after the clause. Assertions play the opposite role in that they have an affirmative character. One may interprete them as some kind of hidden (or cache) memory used to store already derived formulas on subterms of the clause, which are implied by its premise. Therefore we write assertions on the left side of the clause. Altogether, a *CAsed* (C for constraints and As for assertions, in the following just written cased) clause is a triple $\langle S, C, T\rangle$ written in the following shape:

$$[\![S]\!]\ L \Leftarrow \Gamma\ [\![T]\!]$$

Here, S is the assertion and T is the constraint. Cased clauses with empty constraints and assertions are identified with the uncased ones.

A *ground instance* of a cased clause $[\![S]\!]\ L \Leftarrow \Gamma\ [\![T]\!]$ is one of the constrained clause $L \Leftarrow \Gamma\ [\![T]\!]$. Redundancy is also defined by simply neglecting assertions and reusing the definition of constrained clauses. On the semantic level, if $[\![S]\!]\ L \Leftarrow \Gamma\ [\![T]\!]$ is a cased clause, then an interpretation I is a model of it whenever I is a model of $L \Leftarrow \Gamma\ [\![T]\!]$ and for all ground substitutions σ satisfying T and all $L'' \in S$, there is some $\Gamma' \subseteq \Gamma$ with $\{L''\sigma\} \succ_C \{\Gamma'\sigma\}$ and I is a model of $L'\sigma \Leftarrow \Gamma'\sigma$. This conjunctive semantics of assertions[18] underlines the conjunction-like

[18] Remark the 'for all $L'' \in S$' in the last sentence.

character of the comma used as concatenation operator in our clauses and the implication-like character of our left arrow $\Leftarrow$, in contrast to the work of Nieuwenhuis and Rubio, where a sequent arrow is used with the usual disjunctive interpretation of the comma in for succedents, whereas the comma is conjunctive in antecedents.

We will start with the rules of $\mathcal{G}$ adapted for cased clauses. This cased clause calculus is called $\mathcal{GC}3$. These rules together with an order-sorted equational constraint solver as in Figure 4.2 give the calculus $\mathcal{G}$ adapted for deduction in G^n-logic using cased clauses.

1. *strict basic cased superposition into conclusion* :

$$\frac{[\![S']\!]\ s \simeq s' \Leftarrow \Gamma'\ [\![T']\!] \qquad [\![S]\!]\ t \simeq t' \Leftarrow \Gamma\ [\![T]\!]}{[\![S, S']\!]\ t[s']_u \simeq t' \Leftarrow \Gamma, \Gamma'\ [\![T, T', t|_u =^? s]\!]}$$

where $t|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t|_u =^? s]\!]$ has a ground solution σ, s.t.:

(a) $t\sigma \succ t'\sigma, s\sigma \succ s'\sigma$ and $t\sigma \simeq t'\sigma \succ_E s\sigma \simeq s'\sigma$,

(b) $s\sigma \simeq s'\sigma$ is strictly maximal in $s\sigma \simeq s'\sigma \Leftarrow \Gamma'\sigma$ and

(c) $t\sigma \simeq t'\sigma$ is strictly maximal in $t\sigma \simeq t'\sigma \Leftarrow \Gamma$.

2. *strict basic cased superposition into premise* :

$$\frac{[\![S']\!]\ s \simeq s' \Leftarrow \Gamma'\ [\![T']\!] \qquad [\![S]\!]\ L \Leftarrow \Gamma, t \simeq t'\ [\![T]\!]}{[\![S, S']\!]\ L \Leftarrow t[s']_u \simeq t', \Gamma, \Gamma'\ [\![T, T', t|_u =^? s]\!]}$$

where $t|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t|_u =^? s]\!]$ has a ground solution σ, s.t.:

(a) $t\sigma \succ t'\sigma$ and $s\sigma \succ s'\sigma$,

(b) $s\sigma \simeq s'\sigma$ is strictly maximal in $s\sigma \simeq s'\sigma \Leftarrow \Gamma'\sigma$ and

(c) $t\sigma \simeq t'\sigma$ is maximal in $L\sigma \Leftarrow \Gamma\sigma, t\sigma \simeq t'\sigma$.

3. *cased G^n-equality resolution in conclusion* :

$$\frac{[\![S']\!]\ t \simeq t' \Leftarrow \Gamma'\ [\![T']\!]}{[\![S']\!]\ t : \Omega_i = tt \Leftarrow \Gamma'\ [\![T', t =^? t']\!]}$$

where the least sort of t is less or equal to s'_i, $i \in [0..n+1]$, and $[\![T', t =^? t']\!]$ has a ground solution σ, s.t. $t\sigma \simeq t'\sigma$ is maximal in $t\sigma \simeq t'\sigma \Leftarrow \Gamma'\sigma$.

4. *cased G^n-equality resolution in premise* :

$$\frac{[\![S']\!]\ L \Leftarrow t \simeq t', \Gamma'\ [\![T']\!]}{[\![S']\!]\ L \Leftarrow t : \Omega_i = tt, \Gamma'\ [\![T', t =^? t']\!]}$$

where the least sort of t is less or equal to s'_i, $i \in [0..n+1]$, and $[\![T', t =^? t']\!]$ has a ground solution σ, s.t. $t\sigma \simeq t'\sigma$ is maximal in $L \Leftarrow t\sigma \simeq t'\sigma, \Gamma'\sigma$.

5. *cased G^n-tautology resolution in conclusion (subsumed by redundancy)* :

$$\frac{}{[\![S']\!]\ tt \simeq tt \Leftarrow \Gamma'\ [\![T']\!]}$$

6. *G^n-tautology resolution in premise* :

$$\frac{[\![S']\!]\ L \Leftarrow \Gamma', tt \simeq tt\ [\![T']\!]}{[\![S']\!]\ L \Leftarrow \Gamma'\ [\![T']\!]}$$

The next step is to define the inferences concerning the cache memory. *Write cache* basically adds membership consequences of the premise of a clause concerning subterms of a clause or its constraint. *Read cache* eliminates premises of a clause that can be found in the cache. Remark that write and read cache do not build a cycle because of condition *(a)* in write cache, i.e. *read cache* can only eliminate membership formulas that are consequences of strictly smaller premises. *Discard cache* forgets membership formulas on subterms of a clause that have disappeared due to simplifications.

Remark that all clauses $L' \Leftarrow \Gamma'$ associated with $L' \in S$ in $[\![S]\!]\ L \Leftarrow \Gamma$ are redundant since they follow from strictly smaller clauses (cf. conditions of *write cache*). Hence, ignoring them in the definition of clause instances is harmless. This implies also that the application of the cache rules is optional, because the set of instances is unchanged. Furthermore, since we chose in our semantics to unfold all cased clauses, the premise is clearly subsumed by the conclusion, except for *discard cache*, where we can use redundancy w.r.t. the whole presentation. Consequently, the cache rules can be applied deterministically, i.e. the premise can be replaced by the conclusion.

1. write cache :

$$\frac{[\![S']\!]\ L \Leftarrow \Gamma'\ [\![T']\!]}{[\![L', S']\!]\ L \Leftarrow \Gamma'\ [\![T']\!]}$$

if for all ground substitutions σ satisfying T',
there are $(L_1, \ldots, L_k) \subseteq \Gamma'\sigma$, s.t.:

(a) $P_0 \models_{G^n} \Theta(L'\sigma)$ whenever $P_0 \models_{G^n} \Theta(L_1, \ldots, L_k)$ and
(b) $L'\sigma \succ_C (L_1, \ldots, L_k)$,
(c) $L' \notin S'$,

2. read cache :

$$\frac{[\![S']\!]\ L \Leftarrow \Gamma', L'\ [\![T']\!]}{[\![S']\!]\ L \Leftarrow \Gamma'\ [\![T']\!]}$$

if $L'\sigma \in S'\sigma$ for all ground substitutions σ satisfying T'.

3. discard cache :

$$\frac{[\![S', L']\!]\ L \Leftarrow \Gamma'\ [\![T']\!]}{[\![S']\!]\ L \Leftarrow \Gamma'\ [\![T']\!]}$$

Remark that $P_0 \cup \{L'' \mid L'' \in \Gamma\sigma\} \models_{G^n} \Theta(L'\sigma)$ subsumes simpler conditions like $(L' \Leftarrow \Gamma)\sigma \in P$, where P is the current set of cased clauses. This corresponds with context rewriting. In orderto guarantee that assertions are valid deductions from the context given by the conditions of a clause for all clauses occurring in a derivation, we extend the definition of G^n-theorem provingderivations to cased G^n-theorem proving derivations. This corresponds with the changed

model notion (cf. beginning of this section). Remember that instances of cased clauses ignore assertions, but models have to satisfy them.

Definition 4.3.10 *Let $P_0 = \Xi(\mathbf{P}^+)^= \cup \{tt \simeq tt\}$, where $\mathbf{P}$ is a G^n-presentation. A G^n-theorem proving derivation $P_0, P_1, \ldots$ is a* cased G^n-theorem proving derivation *if for all cased clause of the form $[\![S]\!]L \Leftarrow \Gamma'[\![T]\!]$ in the derivation, the asertions S are* valid!cased clauses.

I.e. formally, for all $L' \in S$ and all ground substitutions σ satisfying T, there are instances $L_1, \ldots, L_k \in \Gamma'\sigma$, s.t.:

- $\mathbf{P} \models_{G^n} \Theta((L'\sigma)^{rel})$ *whenever* $\mathbf{P} \models_{G^n} \Theta((L_1, \ldots, L_k)^{rel})$ *and*
- $L'\sigma \succ_C (L_1, \ldots, L_k)$.

Before we start with the soundness proof for $\mathcal{GC}3$, recall also the definition of $.^=$ and its inverse transformation of presentations $.^{rel}$ from Section 4.2.4.

Theorem 4.3.11 *Let $\mathbf{P}$ be a G^n-presentation, Any sequence of cased clauses $P_0, P_1, \ldots$ obtained using $\mathcal{GC}3$ from $P_0 = \Xi(\mathbf{P}^+)^= \cup \{tt \simeq tt\}$ is a cased G^n-theorem proving derivation.*

Proof: Instead of the expressions given in Definition 4.3.10, we may prove the ones below, because of Theorems 4.2.4 and 4.2.5, together with soundness and completeness of **GNL** (cf. Theorems 2.4.5 and 2.4.14): For all $L' \in S$ and all ground substitutions σ satisfying T, there are instances $L_1, \ldots, L_k \in \Gamma'\sigma$, s.t.:

- $P_0 \vdash_{OSGL} L\sigma$ whenever $P_0 \vdash_{OSGL} \Gamma'\sigma$,
- $P_0 \vdash_{OSGL} L'\sigma$ whenever $P_0 \vdash_{OSGL} (L_1, \ldots, L_k)$ and
- $L'\sigma \succ_C (L_1, \ldots, L_k)$.

The prove is by induction on the number of inference steps from P_0 using $\mathcal{GC}3$. The base case is trivial, since all assertions in P_0 are empty. For the induction step, we distinguish the last applied inference rule:

- *strict basic cased superposition into conclusion:*
  Let σ be a ground solution of $T, T', t|_u =^? s$. The G^n-deducibility of $L\sigma$ is a direct consequence of **Paramodulation**, **Cut** and the induction hypothesis. For $L' \in S \cup S'$, the claim is covered by the induction hypothesis.
- *strict basic cased superposition into premise:*
  For the G^n-deducibility of $L\sigma$ for all ground σ satisfying $T, T', t|_u =^? s$, assume that $P_0 \vdash_{OSGL} (t[s']_u \simeq t', \Gamma, \Gamma')\sigma$. Hence, by induction hypothesis on the first premise and **Cut**, we get $P_0 \vdash_{OSGL} s\sigma \simeq s'\sigma$. Consequently, by **Paramodulation**, we also get $P_0 \vdash_{OSGL} t[s]_u\sigma \simeq t'\sigma$ and therefore $P_0 \vdash_{OSGL} L\sigma$ by the induction hypothesis on the second premise and **Cut**. For $L' \in S \cup S'$, the claim is covered again by the induction hypothesis.
- *cased G^n-equality resolution in conclusion:*
  Assume that σ is a ground solution of $T', t =^? t'$ and $P_0 \vdash_{OSGL} \Gamma\sigma$. Hence, $P_0 \vdash_{OSGL} t\sigma \simeq t'\sigma$ by the induction hypothesis. Now, recall that for all ground terms t we can deduce $t = t$ if and only if we can deduce $EX\ t$ from a G^n-presentation, both times using **GNL**. Since $EX\ t$, translates into $t \in \Omega_i$ by Ξ, we get that $P_0 \vdash_{OSGL} t\sigma \simeq t'\sigma$ iff $P_0 \vdash_{OSGL} t\sigma \in \Omega_i$ by Theorem 4.2.4. Once again, for $L' \in S$, the claim is covered by the induction hypothesis.

- *cased G^n-equality resolution in premise*:
  Assume that σ is a ground solution of $T', t =^? t'$ and $P_0 \vdash_{OSGL} t\sigma \in \Omega_i, \Gamma\sigma$. Hence, $P_0 \vdash_{OSGL} t\sigma \simeq t'\sigma, \Gamma\sigma$ by $\Xi(PartialReflex)$. Now, by the induction hypothesis, we get $P_0 \vdash_{OSGL} L$. Again, for $L' \in S$, the claim is covered by the induction hypothesis.
- *cased G^n-tautology resolution in conclusion* :
  Trivial.
- *cased G^n-tautology resolution in premise* :
  $P_0 \vdash_{OSGL} tt = tt$, since $(tt \simeq tt) \in P_0$ and again, for $L' \in S$, the claim is covered by the induction hypothesis.
- *write cache*:
  $P_0 \vdash_{OSGL} L\sigma$ whenever $P_0 \vdash_{OSGL} \Gamma'\sigma$ follows from the induction hypothesis. Concerning $L' \in S$, the claim is satisfied through the conditions of the inference rule.
- *read cache*:
  By induction hypothesis, we know that whenever $P_0 \vdash_{OSGL} (L', \Gamma')\sigma$ for some ground substitution σ satisfying T', then $P_0 \vdash_{OSGL} L\sigma$. Furthermore, we know by the induction hypothesis on the premise of *read cache* that there are instances $L_1, \ldots, L_k \in \Gamma'\sigma$, s.t. $P_0 \vdash_{OSGL} L'\sigma$ whenever $P_0 \vdash_{OSGL} (L_1, \ldots, L_k)$ and $L'\sigma \succ_C (L_1, \ldots, L_k)$. From the latter we get $L_1, \ldots, L_k \in \Gamma'\sigma \setminus \{L'\sigma\}$, since $\succ_C$ is well-founded and total on ground clauses. Consequently, assuming $P_0 \vdash_{OSGL} \Gamma'\sigma$, we get by induction hypothesis on $L' \in S$ that $P_0 \vdash_{OSGL} L'\sigma$ holds and therefore once again by induction hypothesis, but on $L \Leftarrow \Gamma', L'$, that $P_0 \vdash_{OSGL} L\sigma$. Since S does not change, the claim on S is subsumed by the induction hypothesis on the premise.
- *discard cache*:
  The claim on the conclusion is subsumed by the one on the premise.

□

Note that the last theorem also holds for a similar extension of $\mathcal{R}$ by assertions, say $\mathcal{RC}3$ – the proof is completely analogous, since the extension is orthogonal to the differences between R^n-and G^n-logics. However, we do not make use of this calculus and therefore leave out the corresponding theorems. If $P_0, P_1, \ldots$ is a cased G^n-theorem proving derivation, then let $R^*_{P_\infty}$ and $irred_{R_{P_\infty}}(P_\infty)$ be defined as for G^n-theorem proving derivation, i.e. ignoring assertions.

Lemma 4.3.12 *Let $P_0 = \Xi(\mathbf{P}^+) \cup \{tt \simeq tt\}$ and $P_0, P_1, \ldots$ be a fair cased G^n-theorem proving derivation using $\mathcal{GC}3$, s.t. P_∞ does not contain the empty clause. Then $R^*_{P_\infty} \models_{G^n} irred_{R_{P_\infty}}(P_\infty)$.*

Proof: Unchanged w.r.t. Lemma 4.3.7, since it only depends on the presence of the superposition and equality resolution rules. □

Theorem 4.3.13 *Let $P_0, P_1, \ldots$ be a fair G^n-theorem proving derivation, where P_0 is a set of clauses with empty constraints. Then P_0 is G^n-consistent if and only if the empty clause belongs to some P_j.*

Proof: Also unchanged w.r.t. Theorem 4.3.8, using Lemma 4.3.12 in the right places. □

Since $\mathcal{GC}3$ is just an intermediate calculus, we close this section without example, but with the proof that propagating variables from constraints to clauses is sound. Let the corresponding

rule be defined as follows:

variable elimination:

$$\frac{[\![S]\!]\ C\ [\![T, x =^? t]\!]}{[\![S\{x \mapsto t\}]\!]\ C\{x \mapsto t\}\ [\![T\{x \mapsto t\}]\!]} \text{ if } [\![T, x =^? t]\!] \text{ is satisfiable}$$

Lemma 4.3.14 variable elimination *is a sound simplification rule, i.e. if* $P_0, P_1, \ldots, P_k$ *is a cased* G^n*-theorem proving derivation and* P_{k+1} *is obtained from* P_k *by replacing a cased clause* $[\![S]\!]\ C\ [\![T, x =^? t]\!]$ *by* $[\![S]\!]\ C\{x \mapsto t\}\ [\![T\{x \mapsto t\}]\!]$*, then the second is a* G^n*-consequence of* P_k *and the first is* G^n*-redundant in* P_{k+1}*. Furthermore,* $P_0, P_1, \ldots, P_k, P_{k+1}$ *is a cased* G^n*-theorem proving derivation.*

Proof: Obviously, the instances of the two clauses are the same and hence the derived clause is a consequence of P_k subsuming the initial clause. The new derivation is a cased G^1-theorem proving derivation, since we only specialised the assertions and hence the set of ground instances for which we have the prove the implication from strictly smaller instances of the condition has decreased. Consequently, the requirements for $P_0, P_1, \ldots, P_k, P_{k+1}$ to be such a derivation are the same as for $P_0, P_1, \ldots, P_k$. □

As a consequence of this, we may use *variable elimination* for saturation without influencing the validity of Theorem 4.3.13.

Part II

Operationalisation

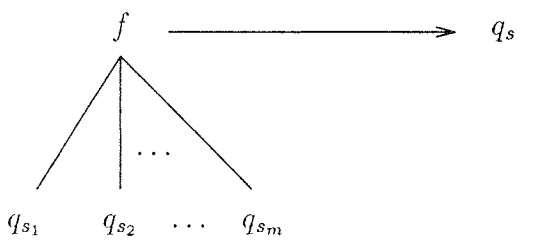
Introduction to Decorated Term Calculi

The second part of this thesis is devoted to operationalisation of G-algebra and G^n-logics based on decorated terms, an extension of the usual term structure by membership information in the term nodes, e.g. $plus(p(0),0)$ may become $plus(p(0^{:Nat})^{:Int}, 0^{:Nat})^{:Int}$ after the application of *decoration rules*, where *plus* is addition, p is the predecessor function, 0 is zero and Nat, Int are the sets of natural numbers and integers, respectively.

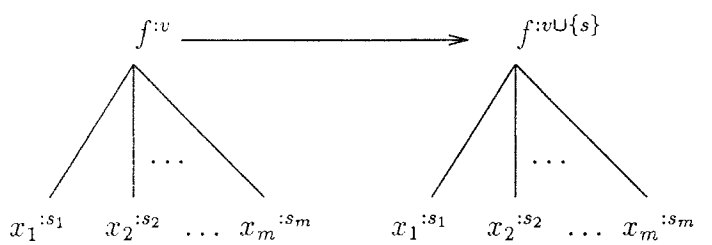
The basic idea for using decorations is to reuse typing proofs during a computation. When such a proof has been accomplished, the result is stored in the decoration and whenever this information is needed again, there is no need to recompute the same typing proof once more. This is in the sense of static typing techniques, where such proofs are done once when the term to be evaluated is parsed. But, such parser-based typing is not possible in this context, since the type of terms may change in order-sorted equational deduction (see Example 1.1.4).

Rule-based Typing

Usual typing algorithms are based on acceptors in form of automata. This idea is underlying the approach of H. Comon and C. Delors [Comon, 1991], who suggested to use tree automata corresponding with second order monadic logic in order to formalize typing issues. Tree automata are similar to usual automata in that they are state based. A bottom-up tree automata rule has the following shape:



Intuitively, sorts s correspond with states q_s and a term is accepted to be of sort s by such an automata, if replacing subterms by states with a bottom-up strategy results in q_s. The rule thus corresponds with the function declaration $f : s_1, \ldots, s_m \to s$. The idea is now to combine this with classical rewriting, i.e. not to consider typing proofs as an independent task but as a part of equational proofs through rewriting, similar to attribute grammars. Indeed, the above rule is a rewrite rule. However, it forgets[19] the structure of the term by reducing to the state corresponding with a sort to which the term belongs. Instead, we suggest now to keep the term structure and additionally to attach sorts to the nodes of a term. The above rule gets then the following shape, where exponents contain the sort information and $x_1, \ldots, x_m$ are variables for decorated terms and v is a variable for decorations:



[19] In [Bogaert et Tison, 1992], the structure is not completely forgotten, but state symbols are mixed into the term structure, which results in the same problems for term rewriting.

Such a rule is called *decoration (rewrite) rule*, written like this in the following:

$$f(x_1{}^{:s_1}, \ldots, x_m{}^{:s_m})^{:v} \rightarrow f(x_1{}^{:s_1}, \ldots, x_m{}^{:s_m})^{:v \cup \{s\}}$$

Using this formalisation, decorations may play a restrictive and an assuring role: The decoration of variables in the left hand side of the rule represent a requirement, i.e. a corresponding proof must already be done for the variable image, whereas the decoration $\{s\}$ added on top on the right hand side represents an assertion, i.e. the proof that the instantiated term belongs to s is done by application of the rule. The careful reader may recognise constraints and assertions from Section 4.3.3 in restrictive and assuring decorations, respectively.

As suggested in [Comon, 1992], this maybe extended by equational side-conditions for the arguments of f. The usefulness of such extensions is illustrated by the following example:

Example 4.3.15 *Recall the* `square`*-function of Example 1.1.4. It is defined by* `square(x) = times(x,x)`*. The problem arising with typing by tree automata without side-conditions in this case is that* `times(x,x)` *cannot be declared as a term of sort* `Nat` *(for natural numbers), even if* `x` *is of sort* `Int` *(for integers), although semantically, i.e. by interpretation of* `times` *as multiplication, such a declaration would hold. Hence, we loose sort information by evaluation, assuming that* `square` *is evaluated through* `times`*, as usual.*

The only kind of terms for which we may give declarations are linear ones, i.e. without repeated variables, such as `times(x,y)`*. Clearly,* `times(x,y)` *is not of sort* `Nat` *in general. Hence, we need to add an equality condition on* `x` *and* `y`*, in order to express that this term is of sort* `Nat`*. This can be performed by tree automata with equality tests [Bogaert et Tison, 1992].*

In the same vein, we may use non-linear decoration rules in order to achieve the same effect as equality tests:

$$times(x^{:\{Int\}}, x^{:\{Int\}})^{:s} \rightarrow times(x^{:\{Int\}}, x^{:\{Int\}})^{:s \cup \{Nat\}}$$

The crucial point for the use of this kind of declarations, simply called term declarations in order-sorted frameworks [Schmidt-Schauß, 1989], is the unification of variables, which requires decidability of the emptiness of sort intersections. This is known to be undecidable in the general case of term declarations [Schmidt-Schauß, 1989]. However, in certain cases, e.g. equality tests for immediate subterms of a function, decision procedures are known to exist [Uribe, 1992; Bogaert et Tison, 1992].

This is also the point where the analogy between the two approaches, tree automata and decoration rules, reaches its end. Automata-based approaches are bound to the kind of automata chosen for the current framework, e.g. *regular* tree automata in [Comon, 1992], which correspond with linear term declarations. This offers advantages if one wants to use not only membership in a sort but also its negation, i.e. membership in its complement, or simple operations on sorts like intersection or union, provided the class of automata is *closed* under these operations. On the other hand, this closure condition may be restrictive. Clearly, for any given class of signatures K which allows us to construct an automata computing the intersection of sorts, such that it is also decidable if a signature belongs to K or not, there is always a signature $\Sigma \notin K$, where this intersection is also computable. Otherwise, membership in K would give us an algorithm computing the intersection in the general case, in contradiction to the result of M. Schmidt-Schauss (cf. [Schmidt-Schauß, 1989]).

Hence, we take the opposite standpoint: we allow for all possible signatures, also those with possibly undecidable intersection, and let then a Knuth-Bendix completion procedure try to

construct an algorithm for the decision of membership. This is in analogy to the spirit of such a procedure in the case of unsorted equational theories and the corresponding word problem: equality of two terms in such theories is in general undecidable, but after successful completion, we have a decision procedure for equational problems. If the completion is not successful, of course, then we do not know if the equational theory is decidable or not.

Completion and Saturation with Sort Inheritance

The main problem in this approach is obviously unification, which is needed for the computation of superpositions in Knuth-Bendix completion algorithms. It still depends on the decidability of intersection, which is unknown until the end of completion. This interdependence, which seems contradictory at first glance, can be broken using a restriction of signatures called *sort-inheritance*, which is a slight generalisation of regularity (cf. Section 2.2, p. 48). Sort inheritance means that whenever a term t belongs to all sorts A in a set S of sorts, then there is a common subsort B of all sorts $A \in S$. Intuitively, it may be rephrased into '(non-)emptiness of sort intersections is implicitly given by the sort structure'.

Under the assumption of sort-inheritance, it is possible to compute unifiers. Fortunately, the structure of typing rules and the requirement of a total ordering on ground formulas allows us to post-pone a test of sort-inheritance to the end of completion. Hence, we may do completion using the assumption and test it only after the completion has eventually terminated.

Sort-inheritance is therefore a restriction w.r.t. a tree automata approach, similar to regularity. But we argue that it is not an essential restriction, since our sort-inheritance test is constructive in that we may use it to suggest signature extensions when a counter-example for sort-inheritance is found. Furthermore, we think that usual specifications are sort-inheriting or may easily be extended to sort-inheriting ones.

Similar results can be achieved for a saturation calculus using decorated terms, for which the refutational completeness can be proven with the help of the cased clause calculus of Section 4.3.3. The difference between this calculus and the Knuth-Bendix completion described above is the possibility to use conditional rules, especially conditional decoration rules, which are written $(t^{:v} \to t^{:v \cup \{s\}}) \Leftarrow \Gamma$. These rules clearly cover all theories where the membership in sorts is decidable. Furthermore, sorts may be parametric.

The Impact of Decorated Terms

For both calculi, completion and saturation, we emphasize that they support dynamic typing and partial functions through decorations. The former is handled by superpositions into decoration rules, yielding new decoration rules. The latter through translations of well-definedness theorems $EX\ t$ into membership theorems of the shape $t \in \Omega$, in the spirit of Section 4.2.3. Hence, the support of dynamic typing and strict partial functions was made possible through decorations.

Furthermore, decorated terms allow us to give a Birkhoff Lemma for order-sorted theories with the help of decoration rules and equations over decorated terms. This kind of *decorated rewriting* allows us also to define complete sets of critical pairs (assuming sort-inheritance) and a corresponding critical pair lemma. This results in a completion procedure coping with rewrite rules, that are not sort-decreasing in the case of static typing, but get sort-decreasing by construction using dynamic typing through decoration rules. The problem treated in the completion procedure can be interpreted as an instance of the interaction problem between constraints and equalities in constrained deduction [Kirchner *et al.*, 1990]. In our case, decorated terms and sort-inheritance allow us to avoid any dependency of the constraint solver on equalities.

On the level of saturation calculi, decorated terms and sort-inheritance allow reusing the techniques from our completion procedure in order to establish refutational completeness for equational Horn clause sets with set terms ordered by inclusion. Recall that this is not trivial, as Example 4.3.2 illustrates. Furthermore, a finite set of decoration rules are the basis of the sort-inheritance test, which corresponds with an infinite set of clauses in the cased clause calculus of Section 4.3.3. This finiteness makes it possible to test this property through the resolution of a finite set of goals.

Last but not least, we emphasise that variables in decorated terms compare with the variables in the $\mathcal{G}$-calculus like order-sorted with unsorted variables. This may result in considerable reductions of the search space, as illustrated by the example of Schubert's Steamroller, which made common automated theorem provers explode in search space, but behaved reasonably with order-sorted resolution [Walther, 1985], because of the order-sorted terms schematising large sets of unsorted terms.

Outline of Part II

We start with a definition of decorated term algebras (Chapter 5), which are based on G-algebra semantics, but whose definition of decorated terms is reused in our work on saturation for G^n-logic towards the end of Part II. Until then, we work on G-algebra only. Hence, this chapter is also foundational for both, the completion and the saturation procedure, which give us the main results in this document (cf. the structure of this thesis, Figure 1.1). After the introduction of a partial ordering on sorts and corresponding transformations, decorated terms are defined and a matching and a unification algorithm are given together with a proof of soundness, completeness and termination w.r.t. the given notion of solution sets. The latter sets differ from the usual definitions for unifiers, where completeness of a set is relative to a given set of terms.

The next step (Chapter 6) in the development of a completion procedure for G-algebra presentations $\mathcal{P}$ is to define formally decoration rules, equalities and rewrite rules for decorated terms and a transformation of $\mathcal{P}$ into a set $\mathcal{P}_0$ of decoration rules and equalities over decorated terms. The soundness and completeness of deduction using $\mathcal{P}_0$ with respect to $\mathcal{P}$ is given by a Birkhoff theorem.

Chapter 7 is devoted to the definition of the completion algorithm for G-algebra presentations based on decorated terms. An overview over the general purpose of a completion procedure leads us to the definition of critical pairs, which allows giving transformation rules for the completion algorithm. Assuming sort-inheritance, a proof of soundness and rewrite proof reduction is given, leading to a theorem stating soundness and completion of deduction using confluent rewrite proofs over decorated terms, whenever the completion was successful. There are two additional failure cases for this completion procedure, apart from unorientable persisting equations: extensions of the subsort relation and counter-examples for sort-inheritance. A transformation rule detecting the first kind of failure is suggested at the end of the chapter together with a conservative signature extension technique that can be used in case the rule is applicable. We argue that this extension technique makes this kind of failure inessential.

Checking sort-inheritance is the remaining problem to be solved in order to achieve complete rewrite proof reduction and therefore confluence of the final presentation in case of success. We consider this issue in Chapter 8. First, a test for confluent sets of decoration rules is developed. Then, we try to use this test for different cases of decoration rules: flat and linear, flat (not necessarily linear) and general term declarations. In the first case, the procedure can be applied with little restrictions on the simplification of decoration rules and the test can be post-poned to the end, in case of success and termination. The second case is similar, although a precompletion

with a different kind of rewrite relation has to be performed first, which also needs a different definition of critical pairs and the complete interdiction of simplification of decoration rules by rewrite rules over decorated terms. In the third case, the precompletion gets more complex, as well as the the critical pairs are more general and the set of applicable deduction rules excludes any simplification or deletion of decoration rules. The test detecting a counter-example for sort-inheritance has to be applied during precompletion. Furthermore, a particular strategy has to be used to get a complete proof reduction relation. At the end of the chapter, the three cases are compared and a conservative signature extension technique that can be used in case the rule detecting a counter-example for sort-inheritance is applicable. We argue again that this extension technique makes this kind of failure inessential.

Chapter 9 is devoted entirely to the related work on completion of order-sorted presentations, their discussion on examples and our conclusion from it. A comparison of sort-inheritance with regularity opens the chapter, followed by an illustration how retracts can be completely eliminated using decorated terms. We show also that our procedure is a conservative extension of the former approach suggested in [Gnaedig *et al.*, 1990]. We focus then on three other approaches, namely the tree automata approach of [Comon, 1992], the signature extension approach [Chen et Hsiang, 1991] and the T-contact method of [Werner, 1993], underlining the differences with the help of examples. The chapter closes with a brief discussion of other approaches and a conclusion from the work on decorated completion.

Based on the cased clause calculus of Section 4.3.3, we develop in Chapter 10 a superposition calculus using decorated terms. Similarly to the development of the decorated completion procedure, we first define a subset relation and then show how to relate the decorated calculus to the undecorated one. We adapt once more the sort inheritance restriction, which is no more necessary for completeness of unification, but allows us decrease the number of clauses during saturation by filtering out those clauses for which the condition infeasible under the assumption of sort inheritance. Sort inheritance can be expressed at the undecorated level as a possibly infinite number of clauses to be added to the presentation. After the definition of the initial decorated presentation and the corresponding unsorted one, we define the saturation calculus together with its equational constraint solver. Refutational completeness is then proved by reduction to the cased clause calculus of Section 4.3.3 and it is shown how decorated rewriting emerges from saturated clause sets. In analogy to the results of Chapter 7, a completeness theorem is given for a test detecting counter-examples for sort-inheritance, which can be post-poned until the end of saturation, but which involves further saturation in the general case. Before we conclude from the results on the saturation calculus, we discuss some examples.

Part II finishes with a conclusion discussing the obtained results, encountered problems and further extensions.

Chapitre 5

Unconditional Decorated Term Algebra

In this chapter we discuss syntactical issues concerning sorts and terms. We start in Section 5.1 with an extraction of a sort structure from a G-algebra presentation, list the assumptions on sort membership for the completion procedure and discuss how to achieve them next. In Section 5.2 we define decorated terms, the property of sort-inheritance for sets of such terms and a subsumption relation on them. The same is done subsequently for decorated substitutions.

Algorithms for matching and unification on decorated terms are designed Sections 5.3 and 5.4. In the following, we assume all terms in the input problems of the algorithms to be valid. Matching and unification are called strict, because they require to take into account at each node both the identity of function symbols and the equality of decorations modulo the sort inheritance closure. Soundness, completeness and termination are shown for both algorithms, assuming that the presentation is sort-inheriting for unification. Section 5.5 finally introduces the notion of subterm conservative solution and includes proofs showing that the above mentioned matching and unification algorithms satisfy this requirement, which will be usefull for the completeness of unification for the superpositions during completion.

1 Sort Membership

1.1 Associating a Partial Ordering over Sorts

In the following sections, we use a quasi-ordering $\leq_{\mathcal{S}}^{syn}$ over the sorts, which is extracted from the variable definitions and declarations in the current presentation $\mathcal{P}$. This simplifies notations and mainly allows for a modularization of deduction. In particular the unification process will heavily rely on this quasi-ordering.

Definition 5.1.1 *Let $\mathcal{P}$ be a presentation. The* syntactic sort ordering in $\mathcal{P}$, *written $\leq_{\mathcal{S}}^{syn}$, is the transitive and reflexive closure of the relation:*

$$A \leq_{\mathcal{S}}^{syn} B \text{ if } \exists x. \{x :: A,\ x : B\} \subseteq \mathcal{P}, \text{ or } B = \Omega$$

A is called subsort *of B, if $A \leq_{\mathcal{S}}^{syn} B$. If $A \leq_{\mathcal{S}}^{syn} B$ and $B \leq_{\mathcal{S}}^{syn} A$, we write $A \sim_{\mathcal{S}}^{syn} B$. The negation is written $A \not\sim_{\mathcal{S}}^{syn} B$. When $A \leq_{\mathcal{S}}^{syn} B$ but $A \not\sim_{\mathcal{S}}^{syn} B$, A is a* strict subsort *of B, written $A <_{\mathcal{S}}^{syn} B$. If neither $A \leq_{\mathcal{S}}^{syn} B$ nor $B \leq_{\mathcal{S}}^{syn} A$, A and B are said* incomparable, *written $A \bowtie_{\mathcal{S}}^{syn} B$.*

The upward closure *of a sort A, written $A\uparrow$, is the set $\{B \mid A \leq_{\mathcal{S}}^{syn} B\}$.*

The (semantic) sort ordering in $\mathcal{P}$, *written $\leq_{\mathcal{S}}^{sem}$, is the transitive and reflexive closure of the following relation:*

$$A \leq_{\mathcal{S}}^{sem} B \text{ if } \exists (x :: A) \in \mathcal{P} \text{ such that } \mathcal{P} \models_G x : B,$$

Clearly, $\leq_{\mathcal{S}}^{syn} \subseteq \leq_{\mathcal{S}}^{sem}$, but not always $\leq_{\mathcal{S}}^{syn} = \leq_{\mathcal{S}}^{sem}$, as the following example illustrates:

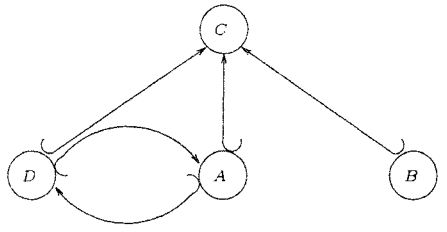
Example 5.1.2 *Let $\mathcal{P} = \{x :: A, y :: B, f(x,y) : B, f(x,y) = x\}$. Clearly,*

$$\mathcal{P} \models_G x : B \text{ and } x :: A \in \mathcal{P}.$$

Therefore $A \leq_{\mathcal{S}}^{sem} B$, but $A \leq_{\mathcal{S}}^{syn} B$ is not true.

The following example illustrates the notions that we just introduced:

Example 5.1.3 *Let $\mathcal{P} = \{x :: A, x : C, x : D, y :: B, y : C, z :: C, u :: D, u : A, x = y\}$. Then $A <_{\mathcal{S}}^{syn} C$, $B <_{\mathcal{S}}^{syn} C$, $D \sim_{\mathcal{S}}^{syn} A$, $D <_{\mathcal{S}}^{syn} C$, $A \bowtie_{\mathcal{S}}^{syn} B$ and $D \bowtie_{\mathcal{S}}^{syn} B$. This is represented by the following graph:*



Therefore, we also have $A <_{\mathcal{S}}^{sem} C$, $B <_{\mathcal{S}}^{sem} C$, $D \sim_{\mathcal{S}}^{sem} A$, $D <_{\mathcal{S}}^{sem} C$, $A \sim_{\mathcal{S}}^{sem} B$ and $D \bowtie_{\mathcal{S}}^{sem} B$. Note that $\mathcal{P} \models_G x : B, y : A$, but neither $A \leq_{\mathcal{S}}^{syn} B$ nor $B \leq_{\mathcal{S}}^{syn} A$.

Unfortunately, as one can expect, we get the following negative result for the semantic relation $\leq_{\mathcal{S}}^{sem}$:

Proposition 5.1.4 *It is undecidable whether for an arbitrary given specification $((\mathcal{S}, \mathcal{F}), \mathcal{P})$, and two sorts $A, B \in \mathcal{S}$, $A \leq_{\mathcal{S}}^{sem} B$ holds.*

Proof: We actually show that $A \sim_{\mathcal{S}}^{sem} B$ is undecidable. This is sufficient since assuming $\leq_{\mathcal{S}}^{sem}$ decidable implies immediately that $\sim_{\mathcal{S}}^{sem}$ is decidable, too. Let Θ be a specification transformation adding two new sorts $\Delta\mathcal{S} = \{A, B\}$ and the set of formulas $\Delta\mathcal{P} = \{t = x, x :: A, t' = x', x' :: B\}$ for some arbitrary terms t, t', i.e. Θ is a non-deterministic function returning $((\mathcal{S} \cup \Delta\mathcal{S}, \mathcal{F}), \mathcal{P} \cup \Delta\mathcal{P})$ for $((\mathcal{S}, \mathcal{F}), \mathcal{P})$.

Suppose now $A \sim_{\mathcal{S}}^{sem} B$ was decidable in $\{\Theta(Spec) \mid Spec \text{ is a specification}\}$. Then $t = t'$ were decidable in arbitrary specifications, which clearly does not hold. □

1.2 Assumptions Concerning the Sort Membership

In order to keep the unification of two variables decidable, we restrict the used signatures to sort inheriting ones. This is a more semantical notion than classical regularity, i.e. the existence of a unique least sort for each term. Sort inheritance just means that if a term can be proved (semantically) to be of multiple sorts, then all these sorts have a common subsort:

Definition 5.1.5 *Let $\Sigma = (\mathcal{S}, \mathcal{F})$ be a signature. A specification $(\Sigma, \mathcal{P})$ is* sort inheriting *if $\forall t \in \mathcal{T}(\Sigma, \mathcal{X}), \forall T \subseteq \mathcal{S}$:*

$$(\forall A \in T, \mathcal{P} \vdash t : A) \Rightarrow (\exists C \in \mathcal{S}, \forall A \in T, C \leq_{\mathcal{S}}^{syn} A)$$

Another way this definition can be understood is that a specification $(\Sigma, \mathcal{P})$ is sort inheriting if the sort information present in $\mathcal{P}$ is complete w.r.t. the semantic sort membership.

Altogether, we make the following general assumptions in this thesis.

General Assumption 5.1.6

5.1.6.1 *The sort relation does not contain cycles.*

5.1.6.2 *The set $mlb(S)$ of maximal elements of the set of lower bounds of any subset of sorts $S \subseteq \mathcal{S}$ is computable.*

5.1.6.3 *All sorts are non-empty.*

5.1.6.4 *The presentation is sort inheriting.*

5.1.6.5 *The specification has bounded membership, i.e. $\#\{A \mid \mathcal{P} \vdash t : A\}$ is finite for all $t \in \mathcal{T}(\Sigma, \mathcal{X})$.*

In signatures with finite sort sets, the points 5.1.6.1, 5.1.6.2 and 5.1.6.5 are obviously decidable (cf. [Schmidt-Schauß, 1987]). However, if parametric sorts are used, more sophisticated properties and sort concepts have to be introduced (see [Smolka, 1989]). Point 5.1.6.3 is undecidable in general but can be enforced by the stricter but decidable requirement that all sorts are syntactically non-empty. Finally, point 5.1.6.4 is undecidable in general but a constructive test exhibiting terms that destroy sort inheritance is developed in Chapter 8. Of course all specifications whose set of sorts is finite have the bounded membership property. For parametric sort theories, Chapter 10 may be seen as an approach. A weakening could be reached by demanding a finite representation of all sorts of a term instead of the set of all these sorts.

We give two non-regular examples to explain the difference between regularity and sort inheritance in parametric signatures.

Example 5.1.7 *Let $list(\Omega)$ denote the sort of lists over arbitrary objects of the universe Ω and nil the empty list, occurring in each of the $list(\ldots(\Omega)\ldots)$-sorts. So the first structure illustrated in figure 5.1 is sort inheriting but not regular.*

Example 5.1.8 *The second structure in figure 5.1 extends the first example by adding a sort constructor for Ω-term tuples called $tup(\Omega)$. For this example let us assume that $tup(\Omega)$ contains also all lists of lists of elements of Ω and that $list(\Omega)$ contains also all tuples of tuples of elements of Ω. Furthermore, we would like to represent the empty tuple with the empty list and thus we know that nil is contained in each of the two sorts $list^i(\Omega)$ and $tup^i(\Omega)$, giving a more complex structure.*

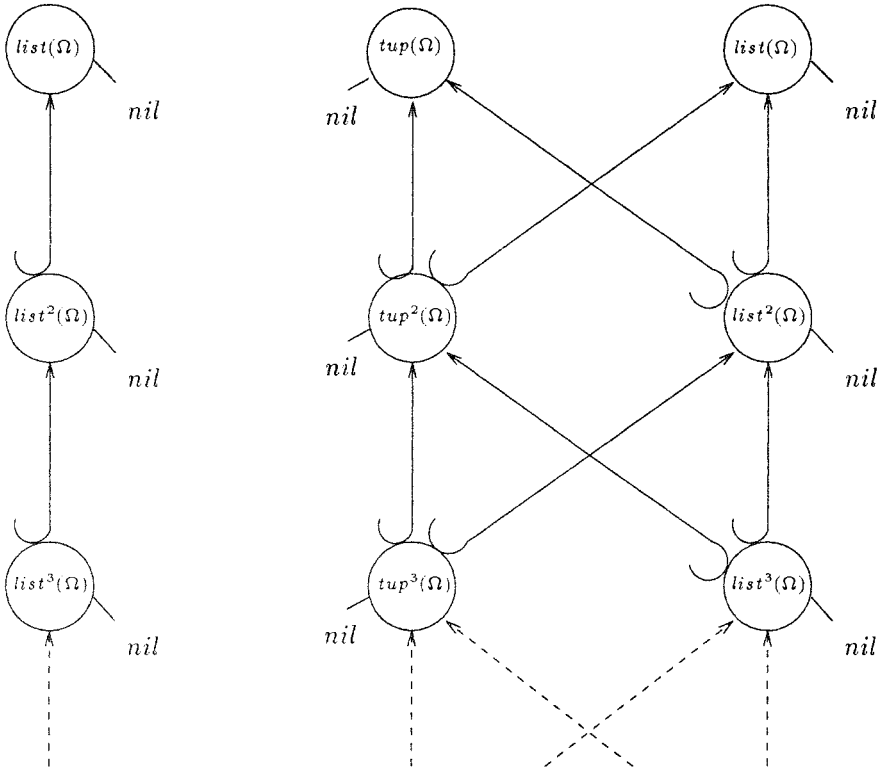


Figure 5.1: Two non-regular but sort inheriting sort structures

In each of the two cases the precondition of classical unification algorithms, namely regularity, is not fulfilled, but unification is clearly decidable. Another solution to the problem of non-regularity in these cases would be the introduction of a new sort *empty_lt* being subsort of all others and containing only *nil*, i.e. the signature would become regular. This technique could even be extended to all elements of a polymorphic sort being independent of the sort's parameters, that are consequently elements of all instances of the polymorphic sort. However, our approach seems to be closer to the intentions of the specifier. Furthermore, the sort inheritance condition is intuitively easy to understand, even though the formal definition seems to be complicated. In general these infinite sort structures[20] also look like the structures used in unified algebras (see [Mosses, 1989] for further details), although there is no difference between the notions of sort and term. Comparing unified and G-algebras is indirectly done in Section 2.3.3, where we compare unified algebras with R^n-logics.

1.3 Eliminating cycles

For the resolution of peaks by critical pairs, presentations without cycles in the sort structure will be required. We investigate cycle elimination for finite $\mathcal{S}$ in G-algebra. We suggest a transformation of presentations replacing all sort symbols occurring in such a cycle by a unique representative of the sort symbols in the cycle. The equational theory described by the presentation remains the same, because G-algebras force the models of a cyclic presentation $\mathcal{P}$ by the rule **MeSubstitutivity** to include the domain of a sort S_i in the domain of a sort S_{i+1} whenever

[20] They are not necessarily lattices, as the second example shows.

$\{x_i :: S_i, x_i : S_{i+1}\} \subseteq \mathcal{P}$. Hence, all the inclusions of sort domains in a cycle cause the equality of the sort interpretations for all sorts occurring in the cycle, i.e. variables of different sorts in the same cycle range over the same subdomains of the model. We start with the formalization of cycles.

Definition 5.1.9 *A G-algebra presentation $\mathcal{P}$ is called* cyclic, *if there is a finite chain of declarations $\{x_i : S_i, x_i :: S_{i+1}\}_{i\in[1..n]} \subseteq \mathcal{P}$ with $S_1 = S_n$. $C = \{S_i \mid i \in [1..n]\}$ is then a* cycle.

The cycles of a finite presentation can easily be found with one of the various algorithms existing in graph theory. The easiest one is surely the bounded depth first search with $|\mathcal{S}|$ as limit. For the real transformation we wish to find a minimal set containing all cycles included in the variable declaration part of the current presentation.

Definition 5.1.10 *Let $\mathcal{C}$ be the set of all cycles contained in a G-algebra presentation $\mathcal{P}$. Then the set $\mathcal{C}_c$ is called a* minimal, complete set of cycles in $\mathcal{P}$, *if:*

- $\forall c_1 \in \mathcal{C}\ \exists c_2 \in \mathcal{C}_c\ :\ c_1 \subseteq c_2$ *(completeness)*
- $\mathcal{C}_c \subseteq \mathcal{C}$ *(soundness)*
- $\mathcal{C}_c$ *has minimal cardinality* *(minimality)*

Note that in infinite sort structures such a set of cycles does not necessarily exist. Just think of an infinite set of sorts and presentations with an infinite number of disjoint cycles. But if such a set exists, it is trivially unique:

Proposition 5.1.11 *Let $\mathcal{P}$ be a presentation and $\mathcal{C}_c$ a minimal, complete set of cycles in $\mathcal{P}$. Then $\mathcal{C}_c$ is unique.*

Proof: Let $\mathcal{C}_c^1$ and $\mathcal{C}_c^2$ be two different minimal, complete sets of cycles in $\mathcal{P}$ and let c be in $\mathcal{C}_c^1$ but not in $\mathcal{C}_c^2$. $\mathcal{C}_c^2$ is complete and thus there is a $c_2 \in \mathcal{C}_c^2$, such that $c \subseteq c_2$. But $\mathcal{C}_c^1$ is also complete and therefore we have a $c_1 \in \mathcal{C}_c^1$ with $c_2 \subseteq c_1$. Furthermore, $\mathcal{C}_c^1$ is minimal, i.e. $c = c_2 = c_1$ and ergo $\mathcal{C}_c^1 = \mathcal{C}_c^2$. □

The uniqueness of the minimal complete set of cycles in $\mathcal{P}$ allows us to replace all sorts in a cycle by a unique representative.

Proposition 5.1.12 *Let $\mathcal{C}_c = \{C_1, \ldots, C_n\}$ be the minimal, complete set of cycles in $\mathcal{P}$ and $\mathcal{S} = \{S_1, \ldots, S_m\}$. Then the set $\mathcal{S}_c(\mathcal{C}_c)$ with:*

$$\mathcal{S}_c(\mathcal{C}_c) = \{S_k \mid i \in [1..m] \text{ and } k = max(\{j \mid S_j \in C_i\})\}$$

is identical with $\mathcal{C}_c$ when all sorts are interpreted by a model of $\mathcal{P}$.

Proof: Trivial by the inclusions in the cycle. □

Corollary 5.1.13 *Let $\mathcal{P}$ be a presentation, $\mathcal{C}_c$ the minimal complete set of cycles in $\mathcal{P}$, Θ the transformation defined in replacing each sort in a cycle by its representative in $\mathcal{S}_c(\mathcal{C}_c)$ Then:*

$$\mathcal{P} \models_G \Psi \text{ iff } \Theta(\mathcal{P}) \models_G \Theta(\Psi).$$

MarkByDeclaration	$(M \cup \{A\}, N, \mathcal{P} \cup \{t : A\})$ $\Longrightarrow (M, N \cup \{A\}, \mathcal{P})$ if $\forall (x :: B) \in \mathcal{V}ar(t), B \in N$

Figure 5.2: Rule to search syntactically empty sorts

This was already mentioned without complete proof in [Schmidt-Schauß, 1987]. Finally, we give a simple example for the defined transformation Θ:

Example 5.1.14 *Let* $\mathcal{S} = \{S_1, S_2, S_3\}$ *and* $\mathcal{P} = \{x :: S_1, y :: S_2, z :: S_3, x : S_2, y : S_3, z : S_1, t : S_3\}$.
Then $\Theta(\mathcal{P}) = \{x :: S_3, y :: S_3, z :: S_3, t : S_3\}$ *and:*

$$\mathcal{P} \models_G t : S_1 \text{ iff } \Theta(\mathcal{P}) \models_G t : S_3,$$

since $t : S_1 \in \mathcal{P}$ *and* $t : S_3 \in \Theta(\mathcal{P})$.

1.4 Non-empty sorts

The definition of models for a specification in G-Algebras does not allow for non-empty sort interpretations. But from a practical point of view, it is interesting to give a syntactical test of non-emptiness similar to the ones of Sections 2.2.2 and 2.4.2, in order to make this condition on models of a specification superfluous.

Definition 5.1.15 *Let* $(\Sigma, \mathcal{P})$ *with* $\Sigma = (\mathcal{S}, \mathcal{F})$ *be a specification. A sort* $A \in \mathcal{S}$ *is called* syntactically empty, *if there is no ground term* $t \in \mathcal{T}(\mathcal{F})$, *s.t.* $\mathcal{P} \vdash^{*}_{DedMem} t : A$, *where* **DedMem** *is the set of rules* **Globality**, **MeSubstitutivity**, *else* A *is called* syntactically non-empty. $(\Sigma, \mathcal{P})$ *has* syntactically non-empty sorts, *if all* $A \in \mathcal{S}$ *are syntactically non-empty.*

Figure 5.2 gives a decision procedure for the syntactical non-emptiness of sorts, assuming the set of sorts to be finite. Starting with the triple $(\mathcal{S}, \emptyset, \mathcal{P})$ the algorithm obviously terminates with $(\emptyset, \mathcal{S}, \mathcal{P}')$ if all sorts contain at least one ground term. Else, the sorts in the first set of the tuple are those without ground terms.

Proposition 5.1.16 *Let* $(\Sigma, \mathcal{P})$ *be a specification with* $\Sigma = (\mathcal{S}, \mathcal{F})$. *The application of the rule* **MarkByDeclaration** *is sound, complete and terminates when applied in a saturating way starting with* $(\mathcal{S}, \emptyset, \mathcal{P})$, *i.e. if* $(M, N, \mathcal{P}')$ *is the triple obtained after termination, then the sorts in* M *are syntactically empty and those in* N *are syntactically non-empty.*

Proof: Termination is obvious, since $\mathcal{S}$ is finite and the first member of the triple is strictly reduced at each step of the application. In the rest of the proof we refer to the first triple member with M, the second with N and the third with $\mathcal{P}$.

The soundness is shown by induction over the number of rule applications. If a sort A is in N after the first step, then there must be a constant declaration $a : A$ in $\mathcal{P}$, because N was empty in the beginning and so no variable can occur in the used membership declaration. Assume now that there is a membership proof $\mathcal{P} \vdash t_i : A_i$ for any $A_i \in N$.

When **MarkByDeclaration** is applied to $t : A$ with $Var(t) = \{x_j \mid j \in J\}$, we take $\bigcup_{j\in J}\{x_j :: A_j \mapsto t_j\}$ as substitution σ for the application of **MeSubstitutivity** to $t : A$ as a ground term membership proof implying the non-emptiness of A. Hence, any sequence of **MarkByDeclaration**-rule applications corresponds with a ground term membership proof in **DedMem.**

The rule's completeness is obvious, since any membership proof for a ground term t based on the rules **Globality** and **MeSubstitutivity** can be transformed to a sequence of **MarkByDeclaration**-rule applications, where **Globality** applied to $t' : A$ is replaced by **MarkByDeclaration** applied to $x : \Omega$ for some x with $x :: A$ in $\mathcal{P}$ – such a variable x must exist in any presentation taken under consideration here, since any sort is a subsort of Ω – and **MeSubstitutivity** is simply replaced by the equivalent **MarkByDeclaration** application. Starting from the leaves of the proof with **DedMem** upwards, it can happen that **MarkByDeclaration** isn't applicable anymore, because the sort A used in a membership declaration is already in N or the membership declaration is not in $\mathcal{P}$. But in this case there was already a ground term membership proof for A, because of the soundness, and therefore A is in N if and only if there is a ground term membership proof using **DedMember.** □

1.5 Sort completion

To compute the unifier of two variables in a sort inheriting presentation, we need an extended sort structure $(\mathcal{S}_\Diamond, \leq^{syn}_{\mathcal{S}_\Diamond})$ containing new sorts representing sort intersections. The transformation is very close to the one in [Schmidt-Schauß, 1987], which performs the conversion of arbitrary signatures with flat, linear term declarations into regular ones. However, we do not have a restriction to such term declarations, neither do we have syntactical sorts. Since typing in G-algebras does not only depend on term declarations, but also on equalities, we cannot precompute the exact contents of sorts representing the intersection.

In order to avoid problems related to empty sorts, as stated in [Goguen et Meseguer, 1992] and [Smolka *et al.*, 1989], we only add new sorts, which surely are non-empty. This non-emptiness can be guaranteed by the condition that there is at least one sort in $\mathcal{S}$ below the new sort, since all sorts in $\mathcal{S}$ are non-empty by general assumption 5.1.6. Finally, we want the number of new sorts to be minimal. Unfortunately, the number may still be exponential, since the construction ranges over the power set $2^{\mathcal{S}}$.

Definition 5.1.17 *Under the General Assumption 5.1.6 we can complete the poset* $(\mathcal{S}, \leq^{syn}_{\mathcal{S}})$ *into* $(\mathcal{S}_\Diamond, \leq^{syn}_{\mathcal{S}_\Diamond})$ *in the following way:*

1. $\forall S \in 2^{\mathcal{S}}$, $\langle S\rangle \in \mathcal{S}_\Diamond$ *provided:*
   (a) $mlb(S) \neq \emptyset$,
   (b) *all elements in* S *are incomparable w.r.t.* $\leq^{syn}_{\mathcal{S}}$ *in* $(\mathcal{S}, \leq^{syn}_{\mathcal{S}})$,
2. $\langle S\rangle \leq^{syn}_{\mathcal{S}_\Diamond} \langle S'\rangle$ *if* $\forall B \in S', \exists A \in S$ *s.t.* $A \leq^{syn}_{\mathcal{S}} B$.

Since $(\mathcal{S}_\Diamond, \leq^{syn}_{\mathcal{S}_\Diamond})$ *is a natural extension of* $(\mathcal{S}, \leq^{syn}_{\mathcal{S}})$, *we identify* A *and* $\langle A\rangle$ *for any* $A \in \mathcal{S}$ *and call* $\mathcal{S}_\Diamond$ *and* $(\mathcal{S}_\Diamond, \leq^{syn}_{\mathcal{S}_\Diamond})$ *the* completing sorts *and* completed sort structure, *respectively, of* $(\mathcal{S}, \leq^{syn}_{\mathcal{S}})$. *In order to simplify notations when* S *is given in extension i.e.* $S = \{A, B, \ldots\}$, *we denote* $\langle S\rangle$ *by* $\langle A, B, \ldots\rangle$.

Remark that we do not want to call sorts in $\mathcal{S}_\Diamond \setminus \mathcal{S}$ *intersection sorts*, since $\mathcal{S}_\Diamond$ does not contain all $\langle S \rangle$ with $S \in \mathcal{S}$, due to the possible emptiness of such sorts.

Example 5.1.18 *Let* $\mathcal{S} = \{A, B, C, D, E\}$ *with* $A \leq_{\mathcal{S}}^{syn} B \leq_{\mathcal{S}}^{syn} C$ *and* $A \leq_{\mathcal{S}}^{syn} D$ *be the initial sort structure. Then, the associated completed sort structure is:*

$$\mathcal{S}_\Diamond = \{\langle A\rangle, \langle B\rangle, \langle C\rangle, \langle D\rangle, \langle E\rangle, \langle C, D\rangle, \langle B, D\rangle\}$$

together with:

$$\begin{array}{l}\langle A\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle B, D\rangle,\\ \langle B, D\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle C, D\rangle,\\ \langle B, D\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle B\rangle,\\ \langle B\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle C\rangle,\\ \langle C, D\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle C\rangle,\\ \langle C, D\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle D\rangle.\end{array}$$

Clearly, the construction also yields the uniqueness of maximal common subsorts for a set of sorts $S \subseteq \mathcal{S}_\Diamond$. Let $S \in \mathcal{S}_\Diamond$. Then $min(S)$ represents the set of minimal sorts in S w.r.t. $\leq_{\mathcal{S}_\Diamond}^{syn}$, which is unique since $\mathcal{S}$ does not contain cycles.

Definition 5.1.19 *Let* $(\Sigma, \mathcal{P})$ *be a specification with* $\Sigma = (\mathcal{S}, \mathcal{F})$, *s.t.* $S \subseteq \mathcal{S}_\Diamond$ *and let* $t^{:S}$ *be in* $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. *The* sort inheritance closure *of* S, *written* $\widehat{S}$ *is the set:*

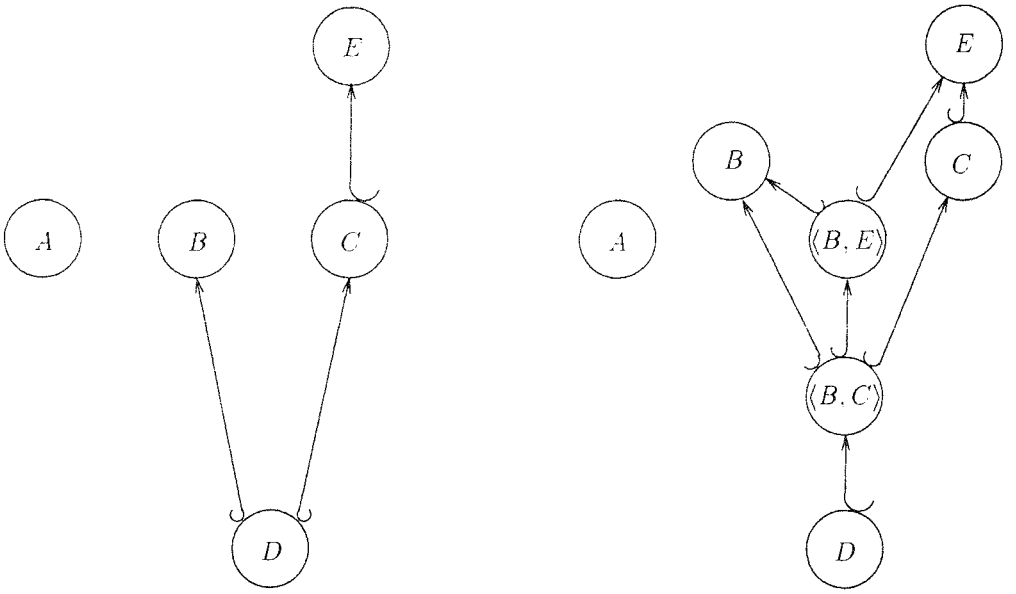
$$\widehat{S} = \{D \in \mathcal{S}_\Diamond \mid \exists \langle T_1\rangle, \ldots, \langle T_n\rangle \in S \ : \ \langle min(\bigcup_{i\in[1..n]} T_i)\rangle \leq_{\mathcal{S}_\Diamond}^{syn} D\}.$$

We illustrate this with an example:

Example 5.1.20 *Let* $\mathcal{S} = \{A, B, C, D, E\}$ *with* $C \leq_{\mathcal{S}}^{syn} E$, $D \leq_{\mathcal{S}}^{syn} B, C$ *and therefore:*

$$\mathcal{S}_\Diamond = \{A, B, C, D, E, \langle B, C\rangle, \langle B, E\rangle\}$$

with $C \leq_{\mathcal{S}_\Diamond}^{syn} E$, $D \leq_{\mathcal{S}_\Diamond}^{syn} \langle B, C\rangle \leq_{\mathcal{S}_\Diamond}^{syn} \langle B, E\rangle \leq_{\mathcal{S}_\Diamond}^{syn} E$, $\langle B, C\rangle \leq_{\mathcal{S}_\Diamond}^{syn} B, C$ *and* $\langle B, E\rangle \leq_{\mathcal{S}_\Diamond}^{syn} B, E$:



Let furthermore $S_1 = \{A\}$, $S_2 = \{B, C\}$ *and* $S_3 = \{A, B, C\}$ *be subsets of* $\mathcal{S}_\Diamond$.

Then $\widehat{S_1} = \{A\}$, $\widehat{S_2} = \{B, C, \langle B, C\rangle, \langle B, E\rangle, E\}$ *and* $\widehat{S_3} = \{A, B, C, \langle B, C\rangle, \langle B, E\rangle, E\}$.

If $S = \{A\}$, we write $\widehat{A}$ instead of $\widehat{S}$. For finite sort sets S, S', the notation $S \lesssim S'$ stands for $\widehat{S} \subseteq \widehat{S'}$ and $\not\lesssim$ for its negation. If $S \lesssim S'$ and $S' \lesssim S$, we write $S \approx S'$.

2 Decorated Terms

The undecidability of typing in G-algebra and its dynamic behavior lead us to adopt a specific term structure with a kind of memorizing technique for intermediate results of membership proofs. Furthermore the equality of two terms and the membership of a term to a sort are two sources of information that should be treated and maintained in a parallel way, because the corresponding formulas have a proper separated status in G-algebras, but with similar deduction rules for substitutivity and replacement. This is taken into account by the particular data structure, called decorated terms. As a matter of fact, the reader familiar with deduction with constraints will notice that the decorations are actually membership constraints spread in the term and defining a kind of local constraints.

2.1 The Term Structure

Given a G-algebra signature $\Sigma = (\mathcal{S}, \mathcal{F})$ and $(\mathcal{S}_\Diamond, \leq^{syn}_{\mathcal{S}_\Diamond})$ its completed sort structure, a $\mathcal{S}_\Diamond$-sorted variable set $\mathcal{X}_\Diamond$ and a set of set-variables $\mathbf{V}$ disjoint from $\mathcal{X}_\Diamond$, we define as follows the decorated $(\Sigma, \mathcal{X}_\Diamond)$-terms.

Definition 5.2.1 *A* decoration *is either a subset of* $\mathcal{S}_\Diamond$, *called ground decoration, or the union of a ground decoration and a variable* s *in* $\mathbf{V}$, *representing a ground decoration. Then, a decorated* $(\Sigma, \mathcal{X}_\Diamond)$*-term, or* decorated term *for short if* $(\Sigma, \mathcal{X}_\Diamond)$ *is clear from the context, is:*

1. *either a pair* (x, S) *for a variable* $x \in \mathcal{X}_\Diamond$ *and a decoration* $S = \{sort(x)\}$ *(also called decorated variable), written* $x^{:S}$,

2. *or of the form* $(f, S)((t_1, S_1), \ldots, (t_n, S_n))$, *if* $(t_1, S_1), \ldots, (t_n, S_n)$ *are decorated* $(\Sigma, \mathcal{X}_\Diamond)$*-terms,* $f \in \mathcal{F}$ *with* $ar(f) = n$ *and* S *is a decoration. Such a term is written* $f({t_1}^{:S_1}, \ldots, {t_n}^{:S_n})^{:S}$.

$Var(t^{:S})$ *stands for the set of variables without their decorations contained in* $t^{:S}$.

$\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ is the set of decorated $(\Sigma, \mathcal{X}_\Diamond)$-terms, $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F})$ the set of ground decorated $(\Sigma, \emptyset)$-terms, s.t. all decorations are ground decorations (in both cases). Remark that we do not consider decorated variables with the shape $x^{:\langle A\rangle, \langle B\rangle}$, where $x :: \langle A\rangle$ and $A \leq^{syn}_{\mathcal{S}} B$. Throughout this work, this situation is avoided during the construction of decorated terms. However, variables x may have any subset of $\mathcal{S}_\Diamond$ as decoration that is equivalent modulo sort inheritance to $\{sort(x)\}$.

Now, we can extend the classical notions of undecorated terms to decorated terms.

Definition 5.2.2 *Let* $t^{:S}$ *be a decorated term in* $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$.

The theory of occurrences, written $\mathcal{O}cc$, is the monoid $(\mathbf{N} \cup \{\Lambda\}, \cdot)$, where Λ denotes the neutral element and $\mathbf{N}$ is the set of natural numbers. The occurrences of $t^{:S}$, written $\mathcal{O}cc(t^{:S})$, with $\mathcal{O}cc(t^{:S}) \subseteq \mathcal{O}cc$, are defined as follows:

$$\begin{array}{lll} \mathcal{O}cc(x^{:S}) & = \{\Lambda\} & \text{if } x \in \mathcal{X}_\Diamond \\ \mathcal{O}cc(f(t_1, \ldots, t_n)^{:S}) & = \{\Lambda\} \bigcup_{i \in [1..n]} \{i.\omega \mid \omega \in \mathcal{O}cc(t_i)\} & \text{otherwise.} \end{array}$$

Let $\omega, \upsilon \in \mathcal{O}cc$. The occurrence prefix ordering $\leq$ *is the smallest binary relation over occurrences, s.t. $\omega \leq \upsilon$ if there exists some $\nu \in \mathcal{O}cc$ with $\upsilon = \omega.\nu$.*
The lexicographic occurrence ordering $\leq_{lex}$ *is the smallest binary relation over occurrences, s.t. $\omega \leq_{lex} \upsilon$ if there exist $\nu, \nu_1, \nu_2 \in \mathcal{O}cc$ and $m, n \in \mathbf{N}$, s.t. $\omega = \nu.m.\nu_1$, $\upsilon = \nu.n.\nu_2$ and $m < n$.*
The term projection *function $._{|} : \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond) \times \mathcal{O}cc \mapsto \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, is defined as follows:*

$$\begin{array}{lll} t^{:S}{}_{|\Lambda} & = & t^{:S} \\ f(t_1, \ldots, t_n)^{:S}{}_{|i.\omega} & = & t_{i|\omega} \quad \text{if } i \in [1..n] \end{array}$$

The variable occurrences *of $t^{:S}$, written $\mathcal{VO}cc(t^{:S})$, are defined as $\{\omega \in \mathcal{O}cc(t^{:S}) \mid t^{:S}{}_{|\omega} \in \mathcal{X}_\Diamond\}$.*
The non-variable occurrences *of $t^{:S}$, written $\mathcal{NVO}cc(t^{:S})$, are those in $\mathcal{O}cc(t^{:S}) \setminus \mathcal{VO}cc(t^{:S})$.*
The decoration S of a term $t^{:S}$ is denoted by $Deco(t^{:S})$.

The term $(t^{:S})_{nd}$ is $t^{:S}$ without its decoration, i.e. a term in $\mathcal{T}((\mathcal{S}_\Diamond, \mathcal{F}), \mathcal{X}_\Diamond)$. This also extends to sets of decorated terms.

Instead of $t^{:\{A,B\}}$, we may also write $t^{:A,B}$ and, when the top decoration does not matter, we also omit the exponent, in order to simplify the syntax.

Let $x^{:S} \in \mathcal{X}_\Diamond$, $t \in \mathcal{T}(\Sigma, \mathcal{X})$ and $(x :: \langle A_1, \ldots, A_n \rangle)$, i.e. $S \approx \{\langle A_1, \ldots, A_n \rangle\}$. Then t is a $\mathcal{T}(\Sigma, \mathcal{X})$-*instance* of $x^{:S}$ if $\forall i = 1, \ldots, n, \mathcal{P} \vdash_{GA} (t : A_i)$. A map $\alpha : \mathcal{X}_\Diamond \to \mathcal{T}(\Sigma, \mathcal{X})$, s.t. $\alpha(x^{:S})$ is a $\mathcal{T}(\Sigma, \mathcal{X})$-instance of the decorated variable $x^{:S}$ for all $x \in \mathcal{X}_\Diamond$, is generating a unique homomorphism $\alpha^* : \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond) \to \mathcal{T}(\Sigma, \mathcal{X})$, called $\mathcal{T}(\Sigma, \mathcal{X})$-*assignment*. Remark that the decorations are dropped with the application of α^*. For simplicity of notation and because of the unicity of α^*, we omit the star exponent. This extends canonically to sets of decorated variables and decorated terms. The set of $\mathcal{T}(\Sigma, \mathcal{X})$-assignments for a variable set $\mathcal{X}_\Diamond$ is written $\mathcal{VASS}^{\mathcal{X}_\Diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$. They serve as a tool for replacing variables of sorts in $\mathcal{S}_\Diamond \setminus \mathcal{S}$, in order to get sets of terms $\mathcal{T} \subseteq \mathcal{T}(\Sigma, \mathcal{X})$. This is used in the following definition of extraction of the membership formula part present in a decorated term:

Definition 5.2.3 *The* decoration formulas *associated with a decorated term t, noted $\mathcal{P}_t$, are the set*

$$\{\alpha(u^{:U}) : B \mid \exists u^{:U} \in subterm\_set(t), \exists \alpha \in \mathcal{VASS}^{\mathcal{X}_\Diamond}_{\mathcal{T}(\Sigma,\mathcal{X})} \text{ and } \exists \langle B, \ldots \rangle \in U\}$$

A decorated $(\Sigma, \mathcal{X}_\Diamond)$-term $t^{:S}$ is valid *in a presentation $\mathcal{P}$, if $\mathcal{P}_t \subseteq Th(\mathcal{P})$. $ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and $ValidT_d(\mathcal{S}_\Diamond, \mathcal{F})$ are the set of all valid, decorated terms, and valid, decorated ground terms, respectively.*

Hence, valid decorated terms are a combined representation of terms and true membership formulas. Obviously, all subterms of a valid term are also valid, since $\mathcal{P}_u \subseteq \mathcal{P}_{t[u]_\omega}$ for all $\omega \in \mathcal{O}cc(t)$.

Intuitively, validity of a ground decorated term $t^{:S}$ means that $(t^{:S})_{nd}$ belongs to all sorts $A \in \mathcal{S}$ occurring in S, in every model of $\mathcal{P}$, and the same holds analogously for all subterms. For non-ground decorated terms $t^{:S}$, it just means that all $\mathcal{T}(\Sigma, \mathcal{X})$-ground instances t' of $(t^{:S})_{nd}$

belong to all sorts $A \in \mathcal{S}$ occurring in S. The same has to be true for all subterms, too. Hence, e.g. $t^{:\langle A\rangle,\langle B\rangle}$, which may also be written $t^{:A,B}$ or $t^{:\{\langle A\rangle,\langle B\rangle\}}$ using our abbreviations from above, is valid iff $t^{:\langle A,B\rangle}$ is.

Each term t in $\mathcal{T}(\Sigma,\mathcal{X})$ is identified with the decorated term $t^{:\downarrow\emptyset}$ with an empty set of sorts at each node except the variable positions, where the decoration is equal to the singleton of the sort of the variable. Notice that the definition of valid terms allows for empty decorations in non-variable positions, i.e. every $t^{:\downarrow\emptyset} \in \mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$ is valid. Furthermore, if $t^{:S} \in \mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$, then $(t^{:S})^{:\downarrow\emptyset}$ stands for $((t^{:S})_{nd})^{:\downarrow\emptyset}$ and $\mathcal{T}^{:\downarrow\emptyset}$ for $\{t^{:\downarrow\emptyset} \mid t \in \mathcal{T}\}$ for any $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$.

All other notions concerning decorated terms are defined in the same way as for classical terms. The top occurrence in decorated terms is denoted by Λ. The equality over decorated terms, noted $=_d$, is the conjunction of classical equality over variables and function symbols with the set identity modulo sort inheritance over the corresponding decorations. The negation of $=_d$ is denoted $\neq_d$. $t^{:S} =_{nd} t'^{:S'}$ stands for $(t^{:S})_{nd} = (t'^{:S'})_{nd}$.

Several decorations may seem to be in conflict when we write for instance: $\forall\omega \in \mathcal{O}cc(t^{:S}) : (t^{:S}{}_{|_\omega})^{:S'} =_d t'$ in the case where $\omega = \Lambda$. Then the actual decoration of t' is always the outermost mentioned, i.e. in this case S' and not S. The only exception is of course the replacement of subterms, i.e. the decoration of $t^{:S}[u^{:S'}]_\omega$ at occurrence ω is clearly S' and not the one of $t^{:S}{}_{|_\omega}$.

Furthermore, we need to adapt the notion of sort inheritance on decorated terms. The definition given now is relative to a subset of valid decorated terms.

Definition 5.2.4 *Let $\mathcal{T} \subseteq Valid\mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$, s.t. no variable of* $\mathbf{V}$ *occurs in a decoration of some term in $\mathcal{T}$ and $\leq$ be a subsort relation. A specification $((\mathcal{S},\mathcal{F}),\mathcal{P})$ is* $\mathcal{T}$-sort inheriting *with respect to $\leq$, if:*

$$\forall t^{:T} \in \mathcal{T}, \exists C \in \mathcal{S}, \forall\langle A\ldots\rangle \in T \ : \ C \leq A.$$

Remark that C might be in $\mathcal{S}_\Diamond$ as well, since this is equivalent by definition of $\mathcal{S}_\Diamond$. The existence of C guarantees that there is a variable $x \in \mathcal{X}_\Diamond$ for each valid decorated term t, s.t. x belongs exactly to the same sorts in $\mathcal{S}_\Diamond$ as t. Clearly, $Valid\mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$-sort inheritance with respect to $\leq_{\mathcal{S}}^{syn}$ is equivalent to sort inheritance of the specification. Consequently, we can write "sort inheritance" instead of "$Valid\mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$-sort inheritance w.r.t. $\leq_{\mathcal{S}}^{syn}$". As an immediate consequence of this definition and since $\leq_{\mathcal{S}}^{syn} \subseteq \leq_{\mathcal{S}}^{sem}$, the following implication holds:

Lemma 5.2.5 *If $((\mathcal{S},\mathcal{F}),\mathcal{P})$ is $\mathcal{T}$-sort inheriting w.r.t. $\leq_{\mathcal{S}}^{syn}$, then it is also $\mathcal{T}$-sort inheriting w.r.t. $\leq_{\mathcal{S}}^{sem}$.*

In the rest of the paper, we mean implicitely $\mathcal{T}$-sort inheritance w.r.t. $\leq_{\mathcal{S}}^{syn}$, when we use $\mathcal{T}$-sort inheritance. The motivation for the following definition is to generalize the fact that when a term t belongs to a sort A, it also belongs to all sorts greater than A.

Definition 5.2.6 *Let $(\Sigma,\mathcal{P})$ be a specification with $\Sigma = (\mathcal{S},\mathcal{F})$, s.t. $S \subseteq \mathcal{S}_\Diamond$ and let $t^{:S}$ be in $\mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)$. The* sort inheritance closure *of $t^{:S}$ is the term $t'^{:S'}$ satisfying:*

$$t^{:S} =_{nd} t'^{:S'} \text{ and } \forall\omega \in \mathcal{O}cc(t^{:S}) \ : \ Deco(t'^{:S'}{}_{|_\omega}) = \widehat{Deco(t^{:S}{}_{|_\omega})}.$$

Furthermore, $t^{:S} \cong_d t'^{:S'}$ stands for $\widehat{t^{:S}} =_d \widehat{t'^{:S'}}$. When $\mathcal{T}$ and $\mathcal{T}'$ are sets of decorated terms, then $\mathcal{T} \cong_d \mathcal{T}'$ if $\{\widehat{t} \mid t \in \mathcal{T}\} = \{\widehat{t'} \mid t' \in \mathcal{T}'\}$. The finiteness of $\mathcal{S}$ in G-algebras gives us the decidability of these relations.

2.2 Subsumption for Decorated Terms

Let us extend the classical term subsumption on decorated terms. We give the definition without defining substitutions in order to separate notions on terms clearly from those on substitutions.

Definition 5.2.7 *Let $(\Sigma, \mathcal{P})$ be a specification with $\Sigma = (\mathcal{S}, \mathcal{F})$ and $t^{:S}, t'^{:S'}$ be valid decorated terms. Then $t'^{:S'}$ is* subsumed modulo sort inheritance *by $t^{:S}$, written $t^{:S} \lesssim_d t'^{:S'}$, if:*

1. $\mathcal{O}cc(t^{:S}) \subseteq \mathcal{O}cc(t'^{:S'})$,
2. $\forall \omega \in \mathcal{NVO}cc(t^{:S}) \ : \ (t_{nd})(\omega) = (t'_{nd})(\omega)$,
3. $\forall \omega \in \mathcal{NVO}cc(t^{:S}) \ : \ Deco(t^{:S}{}_{|\omega}) \approx Deco(t'^{:S'}{}_{|\omega})$ *and*
4. $\forall x \in \mathcal{V}ar(t^{:S}) \ : \ \exists t_x{}^{:T_x} \in \mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond) \ : \ \forall \omega \in \mathcal{O}cc(t^{:S}) \ : \ (t^{:S}{}_{|\omega} \cong_d x^{:S_x} \Rightarrow (t'^{:S'}{}_{|\omega} \cong_d t_x{}^{:T_x}$ *and* $S_x \lesssim T_x))$.

The last condition guarantees that each variable is bound to a unique decorated term modulo sort inheritance. Clearly, if $t^{:S} \lesssim_d t'^{:S'}$ and $t'^{:S'} \lesssim_d t^{:S}$, then $t^{:S} \cong_d t'^{:S'}$ modulo variable renaming. Note that due to the possible variable renaming $\lesssim_d \cap \gtrsim_d$ is not always equivalent to $\cong_d$. Let in the following $\lesssim_d$ stand for $\lesssim_d$ without $\lesssim_d \cap \gtrsim_d$.

Example 5.2.8 *Let $\mathcal{S}_\Diamond = \{A, B, C\}$ and $\leq_{\mathcal{S}}^{syn} = \emptyset$.*

Assuming all terms in the following to be valid, we get

$f(x^{:\{A\}}, x^{:\{A\}}, y^{:\{B\}})^{:\{C\}} \lesssim_d f(a^{:\{A\}}, a^{:\{A\}}, b^{:\{B\}})^{:\{C\}}$,

but neither $f(x^{:\{A\}}, x^{:\{A\}}, y^{:\{B\}})^{:\{C\}} \lesssim_d f(a^{:\{B\}}, a^{:\{A\}}, b^{:\{B\}})^{:\{C\}}$

nor $f(x^{:\{A\}}, x^{:\{A\}}, y^{:\{B\}})^{:\{C\}} \lesssim_d f(a^{:\{A,B\}}, a^{:\{A\}}, b^{:\{B\}})^{:\{C\}}$

or $f(x^{:\{A\}}, x^{:\{A\}}, y^{:\{B\}})^{:\{C\}} \lesssim_d f(b^{:\{A\}}, a^{:\{A\}}, b^{:\{B\}})^{:\{C\}}$ holds.

2.3 Substitutions

Decorated substitutions are a subset of the classical well-defined order-sorted substitutions. As already mentioned, we restrict the used membership theory to the information already existing in the term nodes. Recall that $(:{}^{\downarrow\emptyset}x) =_d x^{:\{sort(x)\}}$.

Definition 5.2.9 *A* decorated substitution *σ is a function replacing decorated variables from $\mathcal{X}_\Diamond$ by decorated terms, such that if $\sigma(x^{:S}) =_d t^{:T}$ with $t^{:T} \not\cong_d x^{:S}$, then $S \subsetneq T$ and $t^{:T}$ is valid. As usual, the set $\mathcal{D}om(\sigma) = \{x \mid \sigma(x^{:\downarrow\emptyset}) \not\cong_d x^{:\downarrow\emptyset}\}$ must be finite. A decorated substitution σ with $\mathcal{D}om(\sigma) = \{x_i \mid i \in [1..n]\}$ is represented by its graph $\bigcup_{i\in[1..n]}\{x_i{}^{:\downarrow\emptyset} \mapsto t_i{}^{:T_i}\}$. Let $\mathcal{I}m(\sigma) = \{t_i{}^{:T_i} \mid i \in [1..n]\}$, $\mathcal{R}an(\sigma) = \mathcal{V}ar(\mathcal{I}m(\sigma))$. Decorated substitutions can be extended to homomorphisms over valid decorated terms as follows:*

$$\begin{array}{rcll} \sigma(x^{:S}) & =_d & t^{:T} & \text{if } (x^{:\downarrow\emptyset} \mapsto t^{:T}) \in \sigma \\ \sigma(y^{:S}) & =_d & y^{:S} & \text{if } y \in \mathcal{X}_\Diamond \text{ and } y \notin \mathcal{D}om(\sigma) \\ \sigma(f(t_1, \ldots, t_n)) & =_d & f(\sigma(t_1), \ldots, \sigma(t_n)) & \text{otherwise.} \end{array}$$

$SUBST$ denotes the set of all decorated substitutions. Let $\mathcal{T}, \mathcal{T}_X \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. The set of all decorated substitutions of $\mathcal{T}_X$ into $\mathcal{T}$, *written $SUBST_{|\mathcal{T}_X \to \mathcal{T}}$, is the set of all decorated substitutions σ, s.t. $\forall t^{:T} \in \mathcal{T}_X \ : \ \sigma(t^{:T}) \in \mathcal{T}$.*

The concatenation of two decorated substitutions σ, τ, written $\sigma \circ \tau$, is the composition of the corresponding functions ($\sigma \circ \tau(t) = \sigma(\tau(t))$ for any term t).

It is important to notice that we do not try to compute the lowest sort of the term in the image of σ, in order to test if this term belongs to a subsort of the variable. Instead, we simply decide this using the sorts in the top decoration of the term modulo sort inheritance.

Lemma 5.2.10 *If t is a valid decorated term and σ a decorated substitution, then $\sigma(t)$ is a valid decorated term.*

Proof: $\mathcal{P}_{\sigma(t)}$ can be obtained from $\mathcal{P}_t$ via **MeSubstitutivity**, where the proof of $\mathcal{P}$-conformity for σ_{nf} is a consequence of the conditions for variable images in decorated substitutions. □

Corollary 5.2.11 *The composition $\sigma \circ \tau$ of two decorated substitutions σ, τ is a decorated substitution.*

Equality and orderings over decorated substitutions are essentially the same as in the classical order-sorted case.

Definition 5.2.12 *Let σ and τ be two decorated substitutions, $\mathcal{V} \subseteq \mathcal{X}_\Diamond$ a finite variable set. σ is* more general than τ over $\mathcal{V}$, *written $\sigma \lesssim_d^{\mathcal{V}} \tau$, if $\forall t \in ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ with $Var(t) \subseteq \mathcal{V}$: $\sigma(t) \lesssim_d \tau(t)$. Then we say that σ* is equal to τ over $\mathcal{V}$, *written $\sigma \cong_d^{\mathcal{V}} \tau$, if $\sigma(x^{:\downarrow\emptyset}) \cong_d \tau(x^{:\downarrow\emptyset})$ for all $x \in \mathcal{V}$.*

Of course, we get the classical equivalence of matching and term subsumption:

Proposition 5.2.13 *Let $t^{:S}, t'^{:S'} \in ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. Then $\exists \sigma \in SUBST$: $\sigma(t) \cong_d t'$ is equivalent to $t \lesssim_d t'$.*

Proof: $\Rightarrow$: Take $t_x{}^{:T_x} = \sigma(x^{:\downarrow\emptyset})$ for all $x \in Var(t^{:S})$.
$\Leftarrow$: Define σ as $\{x^{:\downarrow\emptyset} \mapsto t_x{}^{:T_x}\}_{x \in Var(t^{:S})}$. □

As expected, it is sufficient to find a complementary substitution in order to prove subsumption. However, as already in the unsorted universal algebra, this is not a necessary condition and adding decorations does not change anything in this respect.

Proposition 5.2.14 *Let $(\Sigma, \mathcal{P})$ be a specification, σ, τ be two decorated substitutions and $\mathcal{V} \subseteq \mathcal{X}_\Diamond$ a finite set of variables. Then:*

$$\exists \rho \ : \ \rho \circ \sigma \cong_d^{\mathcal{V}} \tau \Rightarrow \sigma \lesssim_d^{\mathcal{V}} \tau.$$

Proof: Assume $t \in ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ with $Var(t) \subseteq \mathcal{V}$. We have $\rho \circ \sigma(t) \cong_d \tau(t)$. Take $t_x{}^{:T_x} =_d \rho(x^{:\downarrow\emptyset})$ for any $x \in Var(\sigma(t))$. Then $\sigma \lesssim_d^{\mathcal{V}} \tau$ is an immediate consequence of Definition 5.2.7. □

The other direction is not always true, as the following example shows:

Example 5.2.15 *Let $\Sigma = (\{\Omega\}, \{f, a, b\})$, s.t. $ar(f) = 1$, $ar(a) = ar(b) = 0$ and $\mathcal{V} = \{x :: \Omega, y :: \Omega\}$. Then $\sigma = \{x \mapsto f(z^{:\{\Omega\}})^{:\Omega}, y \mapsto f(z^{:\{\Omega\}})^{:\Omega}\}$ subsumes $\tau = \{x \mapsto f(a^{:\{\Omega\}})^{:\Omega}, y \mapsto f(b^{:\{\Omega\}})^{:\Omega}\}$ over $\mathcal{V}$, since there is no term containing x and y at the same time, but there is no ρ, s.t. $\rho \circ \sigma \cong_d^{\mathcal{V}} \tau$.*

However, we get the following result adapted from [Huet, 1976]:

Proposition 5.2.16 *Let $(\Sigma, \mathcal{P})$ be a specification containing some non-monadic function symbol defined over the universe for each argument, σ, τ be two decorated substitutions and $\mathcal{V}$ a finite set of decorated variables. Then:*

$$\sigma \lesssim_d^{\mathcal{V}} \tau \Leftrightarrow \exists \rho \ : \ \rho \circ \sigma \cong_d^{\mathcal{V}} \tau.$$

Proof: We only have to prove the direction from left to right. The fact that Σ contains some non-monadic function symbol defined over the universe for each argument guarantees the existence of a n-ary symbol f, $n > 1$, that allows us to construct some term t containing all variables in $\mathcal{V}$. By the assumption $\sigma \lesssim_d^{\mathcal{V}} \tau$, we get $\sigma(t) \lesssim_d \tau(t)$, i.e. for all $x \in \mathcal{V}$, there is a ${t_x}^{:T_x}$ with $\tau(t)_{|\omega} \cong_d {t_x}^{:T_x}$ whenever $\sigma(t)_{|\omega} = x^{:\downarrow\emptyset}$. Therefore we can take:

$$\rho = \{x^{:\downarrow\emptyset} \mapsto {t_x}^{:T_x} \mid x^{:\downarrow\emptyset} \ncong_d {t_x}^{:T_x}\}.$$

Clearly, ρ and $\rho \circ \sigma$ are decorated substitutions and ${t_x}^{:T_x}$ is a valid term, since all terms in $\mathcal{I}m(\tau)$ are and t is. Therefore, by the construction of t and ${t_x}^{:T_x}$, we get for all $x \in \mathcal{V}ar(\sigma(t)) : \ \rho(x^{:\downarrow\emptyset}) \cong_d \tau(x^{:\downarrow\emptyset})$. Consequently, there is a ρ with $\rho \circ \sigma(t) \cong_d \tau(t)$ for all $t \in \mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ with $\mathcal{V}ar(t) \subseteq \mathcal{V}$. □

3 A Restricted Version of Semantical Order-Sorted Matching

The principal difference between the matching algorithm over decorated terms and the classical order-sorted ones is the use of local membership information. Our algorithm uses only the sorts of a term that are already present in the decoration, in order to avoid the undecidability problems involved by a real membership proof in G-algebras. These proofs are done outside of the matching algorithm in the procedure using the matching algorithm.

We start with some general definitions which help us to specify the matching problem and the completeness and soundness proofs.

3.1 Generalities

Let us start with the definition of matching problems and their solved forms.

Definition 5.3.1 *Let t and t' be decorated terms. A* (decorated) matching equation *has the form $t \lesssim_d^? t'$. A* (decorated) matching problem *$\mathcal{M}$ is a finite conjunction of matching equations of $\mathbb{F}$. If $\mathcal{M} = \bigwedge_{i \in I}(t_i \lesssim_d^? s_i)$ is a matching problem, then $Conj(\mathcal{M})$ denotes the set $\bigcup_{i \in I}\{t_i \lesssim_d^? s_i\}$.*

The trivial matching problem $\mathbb{T}$ is equivalent to the empty conjunction $\bigwedge_{i \in \emptyset}$ and the unsolvable matching problem $\mathbb{F}$ has simply no solution. Conjunctions are associative and commutative.

Definition 5.3.2 *A matching problem is in* solved form, *if it has one of the shapes $\mathbb{T}$, $\mathbb{F}$, or $\bigwedge_{i \in I}({x_i}^{:\downarrow\emptyset} \lesssim_d^? {t_i}^{:S_i})$, such that $x_m \neq x_n \in \mathcal{X}_\Diamond$ for $m, n \in I$ with $m \neq n$.*

The correction of the matching algorithm, in particular the delete rule as we will see later on, can only be proved for variable disjoint matching problems, which are defined as follows :

Definition 5.3.3 *A matching problem $\mathcal{M} = \bigwedge_{i \in I}(s_i \lesssim_d^? t_i)$ is called* variable disjoint *if it satisfies the following restriction :*

$$\bigcup_{i \in I}(\mathcal{V}ar(s_i)) \cap \bigcup_{i \in I}(\mathcal{V}ar(t_i)) = \emptyset.$$

MDelete	$\mathcal{M} \wedge t \lesssim_d^? t'$	$\Longrightarrow \mathcal{M}$ if $t \cong_d t'$
MDecompose	$\mathcal{M} \wedge f(t_1, \ldots, t_n)^{:S} \lesssim_d^? f(t'_1, \ldots, t'_n)^{:S'}$	$\Longrightarrow \mathcal{M} \wedge \bigwedge_{i=1,\ldots,n}(t_i \lesssim_d^? t'_i)$ if $S \approx S'$
MConflict	$\mathcal{M} \wedge f(t_1, \ldots, t_n)^{:S} \lesssim_d^? g(t'_1, \ldots, t'_m)^{:S'}$	$\Longrightarrow \mathbb{F}$ if $f \neq g$ or $m \neq n$ or $S \not\approx S'$
MMerge	$\mathcal{M} \wedge x^{:\downarrow\emptyset} \lesssim_d^? t^{:S} \wedge x^{:\downarrow\emptyset} \lesssim_d^? t'^{:S'}$	$\Longrightarrow \mathcal{M} \wedge x^{:\downarrow\emptyset} \lesssim_d^? t^{:S}$ if $x \in \mathcal{X}_\Diamond$ and $t^{:S} \cong_d t'^{:S'}$
MMergeClash	$\mathcal{M} \wedge x^{:\downarrow\emptyset} \lesssim_d^? t \wedge x^{:\downarrow\emptyset} \lesssim_d^? t'$	$\Longrightarrow \mathbb{F}$ if $x \in \mathcal{X}_\Diamond$ and $t \not\cong_d t'$
MVariableClash	$\mathcal{M} \wedge f(t_1, \ldots, t_n)^{:S} \lesssim_d^? x^{:\downarrow\emptyset}$	$\Longrightarrow \mathbb{F}$ if $x \in \mathcal{X}_\Diamond$

MATCH$_d$

Figure 5.3: Rules for Strict, Syntactic, Decorated Matching

i.e. the variables of the equation's left hand sides are different from those on the right side.

This is a very natural restriction, if we use the matching algorithm for rule choice applications in functional programming languages based on term rewriting, where the rules are universally quantified. Thus, we can rename arbitrarily the variables on the left hand side of the rule.

3.2 The Algorithm

First, we define strict solutions of a matching algorithm.

Definition 5.3.4 *A substitution σ is a* strict solution *(or just $\mathcal{D}$ - solution or $\mathcal{D}$-match) of the matching problem $\mathcal{M}$, if for each $(t \lesssim_d^? t') \in Conj(\mathcal{M})$:*

$$\sigma(t) \cong_d t'$$

A decorated substitution $\sigma \in S(\mathcal{M})$ is called principal solution *of $\mathcal{M}$, if for all decorated substitutions $\tau \in S(\mathcal{M})$: $\sigma \lesssim_d^{Var(\mathcal{M})} \tau$.*

The algorithm in figure 5.3 shows how to calculate a unique solved form for decorated matching problems. Note that the matching procedure is not the classical matching, because we transferred the typing part outside of the matching into the decoration rewrite rules (see Section 6.1.2). So the membership theory actually used by the algorithm presented in this section is only a part of the G-algebra membership theory. This restriction is due to the fact that the membership theories in G-algebras can be undecidable, if the equational part is undecidable. So we have to restrict ourselves to some part of the theory.

Using decorated terms this can be done easily: We only use the results *stored* in the nodes of the terms from previous applications of typing rules. Therefore we have a trivially decidable part of the membership theory, that can be used as criteria for the validity of a variable substitution – the problem we have to decide actually during matching. Given a matching problem $\mathcal{M}$, the

algorithm shown in figure 5.3 is complete, sound and calculates a solved form enabling us to extract a principal solution of $\mathcal{M}$.

If the rule **MDelete** is deleting the last equation of the matching problem, then we obtain the empty problem, denoted by $\mathbb{T}$. The matching problem $\mathbb{F}$ is interpreted as unsolvable matching problem. Obviously, a variable disjoint $\mathcal{M}$ remains variable disjoint when we apply one of these rules, because none of them changes the side of a variable.

We will now prove that the algorithm reduces any matching problem into a unique solved form if the input problem is variable disjoint.

Proposition 5.3.5 *The rules of the algorithm* $\mathbf{MATCH_d}$ *are sound and complete with respect to the definition of a* $\mathcal{D}$ *- solution for an arbitrary, variable disjoint matching problem.*

Proof: We show that each rule of $\mathbf{MATCH_d}$ conserves every and does not add any $\mathcal{D}$ - solution of the matching problem at which it is applied. Let $\mathcal{M}'$ be the initial problem and $\mathcal{M}''$ the obtained problem. Thus, we have to show $S(\mathcal{M}') = S(\mathcal{M}'')$. The problem $\mathcal{M}$ in the proof represents the instantiation of the problem variable $\mathcal{M}$ in the rule.

- **MDelete** : The term t must be a ground term, because $\mathcal{M}'$ is variable disjoint, i.e. $\sigma(t) \cong_d t$ for any decorated substitution σ. Therefore we have $S(\mathcal{M}) = S(\mathcal{M} \wedge t \lesssim_d^? t)$ and **MDelete** is complete and sound.
- **MDecompose** : Clearly, every $\mathcal{D}$ - solution σ of the initial problem is also a $\mathcal{D}$ - solution of the obtained problem. Thus, **MDecompose** is complete. Now, let σ be in $S(\mathcal{M} \wedge \{\bigwedge_{i=1,\ldots,n} t_i \lesssim_d^? t_i'\})$, i.e. $\forall i = 1, \ldots, n : \sigma(t_i) \cong_d t_i'$ and σ is a $\mathcal{D}$ - solution of $\mathcal{M}$. Since decorated substitutions are morphisms and $S \approx S'$, $\sigma(f(t_1, \ldots, t_n)^{:S}) \cong_d f(t_1', \ldots, t_n')^{:S'}$ and consequently $\sigma \in S(\mathcal{M}')$, i.e. **MDecompose** is also sound.
- **MConflict** : Obviously, there is no substitution that is able to make $f(t_1, \ldots, t_n)^{:S}$ and $g(t_1', \ldots, t_m')^{:S'}$ equal, because $f \neq g$ or $m \neq n$ or $S \not\approx S'$, i.e. the initial matching problem has no $\mathcal{D}$ - solution, like the unsolvable problem $\mathbb{F}$. Hence, **MConflict** is sound and complete.
- **MMerge** : Suppose σ is a solution of $x^{:\downarrow\emptyset} \lesssim_d^? t^{:S} \wedge x^{:\downarrow\emptyset} \lesssim_d^? t'^{:S'}$, i.e. $\sigma(x^{:\downarrow\emptyset}) \cong_d t^{:S}$ and $\sigma(x^{:\downarrow\emptyset}) \cong_d t'^{:S'}$ are syntactically equivalent equations modulo sort inheritance. Therefore, they are equivalent to $\sigma(x^{:\downarrow\emptyset}) \cong_d t^{:S}$.
- **MMergeClash** : Suppose there is a $\sigma \in S(\mathcal{M}')$. Then $t \cong_d \sigma(x^{:\downarrow\emptyset}) \cong_d t'$ must hold, but this contradicts the condition of the rule. Therefore we have $S(\mathcal{M}') = S(\mathbb{F}) = \emptyset$.
- **MVariableClash** : There is no substitution σ with $\sigma(f(t_1, \ldots, t_n)^{:S}) \cong_d x^{:\downarrow\emptyset}$, because substitutions cannot replace function symbols. Therefore, $S(\mathcal{M}') = S(\mathbb{F}) = \emptyset$ and thus this rule is sound and complete.

Therefore, the algorithm is sound and complete, since all the rules of $\mathbf{MATCH_d}$ are sound and complete. □

Proposition 5.3.6 *The algorithm* $\mathbf{MATCH_d}$ *terminates for any matching problem.*

Proof: Let $terms(\mathcal{M})$ be the multiset of all the terms appearing as member in some equation in $\mathcal{M}$, $|t|$ the size of a decorated term, i.e. the number of function symbols occurring in t, $|\mathcal{M}| = \Sigma_{t \in terms(\mathcal{M})} |t|$ and $\#\mathcal{M}$ the number of equations in $\mathcal{M}$. Therefore, the complexity measure $(|\mathcal{M}|, \#\mathcal{M})$ is well-founded.

Clearly, **MDecompose** decreases $|\mathcal{M}|$ and all other rules decrease or keep $|\mathcal{M}|$, but they decrease the number of matching equations in $\mathcal{M}$, i.e. $\#\mathcal{M}$. Consequently, the algorithm terminates for any input matching problem. □

Proposition 5.3.7 *The normal form $\mathcal{M}'$ constructed by the algorithm* $\mathbf{MATCH_d}$ *starting with the variable disjoint matching problem $\mathcal{M}$ is unique and in solved form.*

Proof:

- The normal form is a solved form :

  We will prove that for each case of $\mathcal{M}'$ to be not in solved form we have an applicable rule in $\mathbf{MATCH_d}$ and consequently this case is impossible, because $\mathcal{M}'$ is supposed to be a normal form.

  There are two principal possibilities for $\mathcal{M}'$ not to be in solved form. The first one consists in the case when there exists an equation $(t \lesssim^?_d t') \in Conj(\mathcal{M}')$, such that $t \in \mathcal{X}_\Diamond$ and there is another equation $(t \lesssim^?_d s') \in Conj(\mathcal{M}')$. The second possibility is that t is a non-variable term $f(t_1, \ldots, t_n)^{:S}$. Therefore, the following four cases cover all the possibilities :

  1. Let $t \in \mathcal{X}_\Diamond$ and $(t \lesssim^?_d s') \in Conj(\mathcal{M}')$ exists : Hence, one of the rules **MMerge** or **MMergeClash** would be applicable.
  2. Let $t =_d f(t_1, \ldots, t_n)^{:S}$ and $t' \in \mathcal{X}_\Diamond$: Thus, the rule **MVariableClash** is applicable.
  3. Let $t =_d f(t_1, \ldots, t_n)^{:S}$ and $t' =_d f(s_1, \ldots, s_n)^{:S'}$ and $S \approx S'$: Then the rule **MDecompose** is applicable.
  4. Let $t =_d f(t_1, \ldots, t_n)^{:S}$ and $t' =_d g(s_1, \ldots, s_m)^{:S'}$ and $(f \neq g$ or $S \not\approx S')$: Then, **MMergeClash** is applicable.

  Hence, $\mathcal{M}'$ must be in normal form.

- Uniqueness of the obtained normal form :

  We know from proposition 5.3.6, that the rules in $\mathbf{MATCH_d}$ terminate for each matching problem. Consequently, it is sufficient to prove the local confluence. If we regard the left hand sides of the rules in $\mathbf{MATCH_d}$ and take into consideration, that **MDelete** is only applied to ground terms, we get the following possible superpositions :

  - **MDelete** and **MDecompose** : Let $t =_d f(t_1, \ldots t_n)^{:S}$. Therefore the matching problem $\bigwedge_{i\in[1,\ldots,n]}(t_i \lesssim^?_d t_i)$ can also be reduced to $\mathbb{T}$ by an n-times application of **MDelete**, i.e. the local confluence is guaranteed.
  - **MMerge** and **MMergeClash** : We have :

    $$x^{:\downarrow\emptyset} \lesssim^?_d t^{:S} \wedge x^{:\downarrow\emptyset} \lesssim^?_d t'^{:S'} \wedge x^{:\downarrow\emptyset} \lesssim^?_d t''^{:S''} \Rightarrow_{MMergeClash} F$$

    By the condition of the rules we have $t'^{:S'} \not\approx_d t''^{:S''}$, i.e. we can deduce the following :

    $$\begin{array}{ll} & x^{:\downarrow\emptyset} \lesssim^?_d t^{:S} \wedge x^{:\downarrow\emptyset} \lesssim^?_d t'^{:S'} \wedge x^{:\downarrow\emptyset} \lesssim^?_d t''^{:S''} \\ \Rightarrow_{MMerge} & x^{:\downarrow\emptyset} \lesssim^?_d t^{:S} \wedge x^{:\downarrow\emptyset} \lesssim^?_d t''^{:S''} \\ \Rightarrow_{MMergeClash} & F \end{array}$$

    Hence, this pair is convergent.

– **MMerge** and **MMerge** : obvious.

All other superpositions converge trivially. Hence, the rule system is confluent and we have the uniqueness of the normal form, since the derivations are all finite (see proposition 5.3.6).

□

Proposition 5.3.8 *Let $\mathcal{M}_{sf}$ be the solved form of a matching problem $\mathcal{M}$ found by the algorithm* $\mathbf{MATCH_d}$. *If $\mathcal{M}_{sf}$ has the form $\bigwedge_{i\in I}(x_i{}^{:\downarrow\emptyset} \lesssim_d^? t_i{}^{:T_i})$ and $\forall i \in I, \{sort(x_i)\} \subsetneq T_i$ is satisfied, then the decorated substitution $\sigma = \{x_i{}^{:\downarrow\emptyset} \mapsto t_i{}^{:T_i}\}_{i\in I}$ is a principal $\mathcal{D}$ - solution of $\mathcal{M}$, else there is no $\mathcal{D}$ - solution for $\mathcal{M}$.*

Proof: We have already proven soundness and completeness of the algorithm, i.e. if $\mathcal{M}_{sf} = \mathbb{F}$, then $S(\mathcal{M}) = \emptyset$. Remains the proof that σ is a principal solution in the other case, i.e. if $\mathcal{M}_{sf} = \bigwedge_{i\in I}(x_i{}^{:\downarrow\emptyset} \lesssim_d^? t_i{}^{:T_i})$.

- σ is a $\mathcal{D}$ - solution of $\mathcal{M}$:
  Thanks to the condition for all $i \in I$, we are sure that σ is a decorated substitution. By the construction of σ we have $\sigma(x_i{}^{:\downarrow\emptyset}) \cong_d t_i{}^{:T_i}$ for all $i \in I$. Thus, σ is a $\mathcal{D}$ - solution of $\mathcal{M}_{sf}$. With soundness and completeness of the algorithm $\mathbf{MATCH_d}$ we know that σ is also a $\mathcal{D}$ - solution of $\mathcal{M}$.
- σ is a principal $\mathcal{D}$ - solution of $\mathcal{M}_{sf}$:
  We know soundness and completeness of the algorithm and thus it is sufficient to prove that σ is complete w.r.t. the $\mathcal{D}$ - solutions of $\mathcal{M}_{sf}$.
  Let τ be any other solution of $\mathcal{M}_{sf} = \bigwedge_{i\in I}(x_i{}^{:\downarrow\emptyset} \lesssim_d^? t_i{}^{:T_i})$. Hence, we know that $\tau(x_i{}^{:\downarrow\emptyset}) \cong_d t_i{}^{:T_i} \cong_d \sigma(x_i{}^{:\downarrow\emptyset})$, i.e. there is a decorated substitution $\rho = id_d$, such that $\rho \circ \sigma \cong_d^{Var(\mathcal{M})} \tau$, and thus σ is a principal $\mathcal{D}$ - solution.

Nota bene: the extraction of the principal solution is also correct for the empty matching problem $\mathbb{T}$, because we obtain the identity substitution, which subsumes any other substitution. □

Corollary 5.3.9 *Strict decorated matching is decidable.*

4 Strict Decorated Unification in Sort Inheriting Presentations

In this section we prove soundness and completeness of a strict unification algorithm over decorated terms. This algorithm does not use all term declarations contained in the current presentation. In fact, we use only variable declarations and we do not try to prove that a non-variable term t belongs to the sort of a variable with which it should be unified. The undecidability of a more general unification using the whole declaration part of the current presentation is proven in [Hintermeier, 1992] via a variable-variable unification problem. This shows the necessity of a signature restriction preventing the undecidability of variable unification. Since term declarations can be introduced during the completion process, where we want to use the unification algorithm for the computation of critical pairs, we cannot use semi-linearity or some other structural restriction for these term declarations.

However, regularity is one possibility to achieve decidability. In the following, we show that sort inheritance is another way to do it. This property is in fact equivalent to regularity when

we restrict ourselves to finite, cycle-free sort structures, as this is the case in G-algebras. But it is strictly weaker than classical regularity, because it also allows for parametric sort theories, where terms can belong to an infinite number of sorts, i.e. they need not have a minimal sort (cf. Section 5.1.2).

4.1 Generalities

We will start with similar definitions as for the matching algorithms.

Definition 5.4.1 *Let $t^{:S}$ and $t'^{:S'}$ be two decorated terms. A* (decorated) unification equation *has the form $t^{:S} \cong_d^? t'^{:S'}$. A* (decorated) unification problem *$\mathcal{U}$ is a finite conjunction of unification equations or $\mathbb{F}$. If $\mathcal{U} = \bigwedge_{i\in I}(t_i \cong_d^? s_i)$ is a matching problem, then $Conj(\mathcal{U})$ denotes the set $\bigcup_{i\in I}\{t_i \cong_d^? s_i\}$.*

The trivial unification problem $\mathbb{T}$ is equivalent to the empty conjunction $\bigwedge_{i\in\emptyset}$ and the unsolvable unification problem $\mathbb{F}$ has simply no solution. Conjunctions are associative and commutative. For one rule in our unification algorithm (**Coalesce**) and the termination proof, we need to define a well-founded ordering on decorated variables. We show in the following how this may be attained.

Let $\mathcal{U}$ be the initial unification problem for the algorithm in Figure 5.4 and $\mathcal{S}^{\mathcal{U}} = \{A \in \mathcal{S} \mid \exists x \in \mathcal{V}ar(\mathcal{U}) \text{ with } sort(x) = \langle A, \ldots\rangle\}$. Let furthermore $\mathcal{S}_\Diamond^{\mathcal{U}}$ be the set of all $\{\langle T\rangle \in \mathcal{S}_\Diamond \mid T \subseteq \mathcal{S}^{\mathcal{U}}\}$. Consequently, all variables introduced by **UIntersect** have some sort in $\mathcal{S}_\Diamond^{\mathcal{U}}$. Remark furthermore, that $\mathcal{S}_\Diamond^{\mathcal{U}}$ is finite since $\mathcal{S}$ is. Finally, let $\leq_{\mathcal{U}}$ be a conservative, linear extension of $\leq_{\mathcal{S}_\Diamond}^{syn}$ restricted to $\mathcal{S}_\Diamond^{\mathcal{U}}$. Consequently, $(T, \leq_{\mathcal{U}})$ is well-ordered for every initial segment T of $\mathcal{S}_\Diamond^{\mathcal{U}}$.

Now we can define $v(x)$, where $x \in \mathcal{X}_\Diamond$ and $sort(x) = A$, to be $\#\{B \mid B \leq_{\mathcal{U}} A\} + 1$ and $MaxSort$ be $\#\mathcal{S}_\Diamond^{\mathcal{U}}$. In order to extend $v(x)$ also to non-variable terms t, we define $v(t) = 0$ if $t \notin \mathcal{X}_\Diamond$. This is also the reason why we took the successor of the cardinality in the definition of $v(x)$ for $x \in \mathcal{X}_\Diamond$.

Since we can enumerate all variables of a sort, we can define $ind(x)$ to be the unique index of a variable x of sort A w.r.t. the enumeration of all variables in A. Moreover, we can define $sorted\_ind(x)$ to be the ordered pair $(v(x), ind(x))$. In the following, we will write $x \leq_{\mathcal{V}} y$, if $sorted\_ind(x)$ $(\leq, \leq)$ $sorted\_ind(y)$, and $<_{\mathcal{V}}$ resp. $=_{\mathcal{V}}$ for the associated strict ordering resp. equality. Finally, $w(t)$ stands for the multiset of all $sorted\_ind(x)$, s.t. $x \in \mathcal{V}ar(t)$.

The goal of the algorithms is the transformation of any unification problem into a solved form, simplifying the extraction of a solution :

Definition 5.4.2 *A unification problem $\mathcal{U}$ is in* solved form, *if it is of the form $\mathbb{T}$, $\mathbb{F}$, or $\exists z_1, \ldots z_m \bigwedge_{i\in[1..n]} x_i^{:\downarrow\emptyset} \cong_d^? t_i^{:T_i}$ with $x_i \in \mathcal{X}_\Diamond$, such that :*

1. $\forall 1 \leq i < j \leq n \; : \; x_i \neq x_j$
2. $\forall 1 \leq i \leq j \leq n \; : \; x_i \notin \mathcal{V}ar(t_j)$
3. $\forall 1 \leq i \leq n \; : \;$ if $t_i \in \mathcal{X}_\Diamond$, then $x_i \notin \{z_1, \ldots z_m\}$ and $\forall 1 \leq j \leq n \; : \; x_i \notin \mathcal{V}ar(t_j)$
4. $\forall 1 \leq k \leq m \; : \; \exists 1 \leq j \leq n \; : \; z_k \in \mathcal{V}ar(t_j)$
5. $\forall 1 \leq i \leq n \; : \; \{sort(x_i)\} \subsetneq T_i$

This definition is a decorated version of a *dag-solved form*, defined in [Jouannaud et Kirchner, 1991]. It differs in point 5, where we replaced least sorts by decoration inclusion modulo sort inheritance. We will drop the variable quantifiers in the following, whenever they are evident. Before we start with the definition of the unification algorithms, we will extend the formalism by some useful notions, corresponding with those in the section about matching problems.

A solution of a strict unification problem with respect to a term set is defined as follows:

Definition 5.4.3 *Let $\mathcal{P}$ be a presentation, $\mathcal{U}$ be a conjunction of unification equations, $\mathcal{T}_{\mathcal{U}} = terms(\mathcal{U})$ and $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$.*

A decorated substitution $\sigma \in SUBST_{|_{\mathcal{T}_{\mathcal{U}} \to \mathcal{T}}}$ is a strict decorated $\mathcal{T}$-unifier *(w.r.t. $\mathcal{P}$), if :*

$$\forall (t \cong_d^? t') \in Conj(\mathcal{U}), \sigma(t) \cong_d \sigma(t')$$

$SU_{\mathcal{P}}(\mathcal{U})_{|_{\mathcal{T}}}$, called strict $\mathcal{T}$-solution set *of $\mathcal{U}$, is the set of all strict decorated $\mathcal{T}$-solutions of $\mathcal{U}$.*

In the case of $\mathcal{T} = ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, σ may simply be called strict decorated unifier, $\mathcal{D}$-unifier, strict solution or $\mathcal{D}$ - solution of $\mathcal{U}$ and the suffix $|_{\mathcal{T}}$ may be omitted. If $\mathcal{P}$ is clear from the context, we will write $SU(\mathcal{U})$ instead of $SU_{\mathcal{P}}(\mathcal{U})$. As usual, we will give a positive and a negative example for the notion of a $\mathcal{D}$ - solution :

Example 5.4.4 *Let $\mathcal{P} = \{x :: A, y :: B, x : B, a : A\}$ and*
$\mathcal{U} = (x^{:\downarrow\emptyset} \cong_d^? a^{:\{A\}} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset} \wedge y^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}})$.
$\mathcal{U}$ has no $\mathcal{D}$ - solution, because $\sigma(x)$ should have the decorations $\{A\}$ and $\{A, B\}$ at the same time.

Example 5.4.5 *Let $\mathcal{P} = \{x :: A, y :: B, x : B, a : A\}$ and*
$\mathcal{U} = (x^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset} \wedge y^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}})$.
$\mathcal{U}$ has the $\mathcal{D}$ - solution $\sigma = \{x^{:\downarrow\emptyset} \mapsto a^{:\{A,B\}}, y^{:\downarrow\emptyset} \mapsto a^{:\{A,B\}}\}$.

For complete sets of unifiers, we give a definition that is also parameterised with a set of decorated terms. This is useful for proof theoretic purposes. The idempotency restriction remains unchanged. However, the notions of soundness and completeness of solution sets must be refined for the proofs in the rest of this paper. The following redefinition associates a term set as reference.

Definition 5.4.6 *Let the specification $((\mathcal{S}, \mathcal{F}), \mathcal{P})$, the decorated unification problem $\mathcal{U}$ of the shape $\bigwedge_{i \in I}(s_i \cong_d^? t_i)$ and $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ be given. $CSU_{\mathcal{P}}(\mathcal{U})_{|_{\mathcal{T}}}$ is a $\mathcal{T}$-complete set of unifiers of the unification problem $\mathcal{U}$, if :*

1. $CSU_{\mathcal{P}}(\mathcal{U})_{|_{\mathcal{T}}} \subseteq SU_{\mathcal{P}}(\mathcal{U})_{|_{ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)}}$. *(soundness)*
2. $\forall \phi \in SU_{\mathcal{P}}(\mathcal{U})_{|_{\mathcal{T}}} : (\exists \sigma \in CSU_{\mathcal{P}}(\mathcal{U})_{|_{\mathcal{T}}} : \sigma \lesssim_d^{Var(\mathcal{U})} \phi)$. *($\mathcal{T}$-completeness)*
3. $\forall \sigma \in CSU_{\mathcal{P}}(\mathcal{U})_{|_{\mathcal{T}}}, \sigma \circ \sigma = \sigma$. *(idempotency)*

We may omit $\mathcal{T}$ in the case $\mathcal{T} = ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ or $\mathcal{P}$ if the current presentation is non-ambiguous. When $SU_{\mathcal{P}}(\mathcal{U})$ is non-empty, we say that $\mathcal{U}$ is *solvable*, otherwise it is called *unsolvable*.

UDelete
$$\exists \vec{z} : (\mathcal{U} \wedge t \cong_d^? t') \Longrightarrow \exists \vec{z} : (\mathcal{U}) \quad \text{if } t \cong_d t'$$

UDecompose
$$\exists \vec{z} : (\mathcal{U} \wedge f(t_1,\ldots,t_n)^{:S} \cong_d^? f(t'_1,\ldots,t'_n)^{:S'}) \Longrightarrow \exists \vec{z} : (\mathcal{U} \wedge \bigwedge_{i\in[1..n]}(t_i \cong_d^? t'_i)) \quad \text{if } S \approx S'$$

UCoalesce
$$\exists \vec{z} : (\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset}) \Longrightarrow \exists \vec{z} : (\mathcal{U}\{x^{:\downarrow\emptyset} \mapsto y^{:\downarrow\emptyset}\} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset}) \quad \text{if } x \in \mathcal{V}ar(\mathcal{U}) \text{ and } y <_V x$$

UMerge
$$\exists \vec{z} : (\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? t^{:S} \wedge x^{:\downarrow\emptyset} \cong_d^? t'^{:S'}) \Longrightarrow \exists \vec{z} : (\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? t^{:S} \wedge t^{:S} \cong_d^? t'^{:S'}) \quad \text{if } x \in \mathcal{X}_\Diamond \text{ and } t^{:S} \notin \mathcal{X}_\Diamond \text{ and } |t^{:S}| \leq |t'^{:S'}|$$

UErase
$$\exists \vec{z}, z' : (\mathcal{U} \wedge z'^{:\downarrow\emptyset} \cong_d^? t^{:T}) \Longrightarrow \exists \vec{z} : (\mathcal{U}) \quad \text{if } z' \notin \mathcal{V}ar(\mathcal{U}) \cup \mathcal{V}ar(t^{:T}) \text{ and } t^{:T} \in \mathcal{X}_\Diamond \text{ implies } sort(t) \leq_{\mathcal{S}}^{syn} sort(z')$$

UIntersect
$$\exists \vec{z} : (\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset}) \Longrightarrow \exists \vec{z}, z' : \mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? z'^{:\downarrow\emptyset} \wedge y^{:\downarrow\emptyset} \cong_d^? z'^{:\downarrow\emptyset}$$
$$\text{if } sort(x) = \langle T \rangle \bowtie_{\mathcal{S}_\Diamond}^{syn} \langle T' \rangle = sort(y) \text{ and } sort(z') = \langle min(T \cup T') \rangle, z' \notin \mathcal{V}ar(\mathcal{U}) \cup \{x, y\}$$

Figure 5.4: Transformation Rules of $\mathbf{UNIF_d}$

4.2 The Algorithm

During a completion procedure, we may probably not have the problem to solve all kinds of unification equations of the form $x \cong_d^? y$ with $x, y \in \mathcal{X}_\Diamond$, i.e. we don't need the satisfaction of the sort inheritance proprty, which allows us to decide emptiness of sort intersection, for all possible unification problems. If V_i for $i \in I$ are the variable sets actually to be unified and that do not have a non-variable potential instantiation, we can in fact restrict this condition to something that might be called simultaneous V_i-sort inheritance for $i \in I$. But the V_i depend on the used strategy and the unification algorithm. Therefore we preferred the more restricting general sort inheritance for a clear definition. But the propositions that will follow in this section also hold for simultaneous V_i-sort inheritance.

The algorithm shown in figures 5.4 and 5.5 calculates a solved form of any unification problem. It is sound, complete and terminating, as we prove later in this section. The unification equation symbol $\cong_d^?$ is supposed to be associative and commutative.

The essential difference between this algorithm and the classical ones for regular unification with flat term declaration as in [Schmidt-Schauß, 1987], [Jouannaud et Kirchner, 1991] or [Waldmann, 1989b], is the treatment of unification of a variable with a non-variable term. In the classical case, such an equation $x \cong_d^? t$ is solved searching a substitution σ for the variables in t, such that the least sort parse of $\sigma(t)$ is lower than the sort of x, but still as general as possible. This calculation may be quite expensive. When the flat term declarations are replaced by general ones and one does not worry about regularity, it is even possible to prove the undecidability of the $x \cong_d^? t$ - case (cf. [Schmidt-Schauß, 1987]).

UConflict	$\exists \vec{z}$: $(\mathcal{U} \wedge f(t_1,\ldots,t_n)^{:S} \cong_d^? g(t'_1,\ldots,t'_p)^{:T})$	$\Longrightarrow \mathbb{F}$
	if $f \neq g$ or $m \neq n$ or $S \not\approx T$	
UCheck*	$\exists \vec{z}$: $(\mathcal{U} \wedge x_1^{:\downarrow\emptyset} \cong_d^? t_1[x_2^{:\downarrow\emptyset}]_{p_1} \wedge \ldots \wedge x_n^{:\downarrow\emptyset} \cong_d^? t_n[x_1^{:\downarrow\emptyset}]_{p_n})$	$\Longrightarrow \mathbb{F}$
	if $p_i \neq \wedge$ for some $i \in [1..n]$	
UDecoClash	$\exists \vec{z}$: $(\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? f(t_1,\ldots,t_n)^{:T})$	$\Longrightarrow \mathbb{F}$
	if $sort(x) \not\subseteq T$	
URemove	$\exists \vec{z}$: $(\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset})$	$\Longrightarrow \mathbb{F}$
	if $sort(x) = \langle T\rangle \bowtie^{syn}_{\mathcal{S}_\Diamond} \langle T'\rangle = sort(y)$ and $\nexists \langle min(T \cup T')\rangle \in \mathcal{S}_\Diamond$	

Figure 5.5: Failure Rules of $\mathbf{UNIF_d}$

Our algorithm does not search this most general least sort parse. On the contrary, it only examines the decoration of t and x. If the decoration of x is not in the one of t, unification simply fails. The sort parses taken into consideration are therefore only those already present in the decoration of t. This became possible by the integration of the typing of non-variable terms into decoration rules that are independant from the unification algorithm. So we can decide whether a variable substitution is valid or not, as this was already the case for the corresponding matching algorithm.

But this is not the only problem concerning the variables. Another one is the unification of two variables that is proven undecidable in [Schmidt-Schauß, 1987] for the general case and in [Hintermeier, 1992] almost in the same way for general, decorated unification. One way to avoid this undecidability is the restriction of the term declarations, for example to shallow theories, what is not convenient for this application, because the completion algorithm may generate arbitrarily formed terms as left hand sides for typing rules which will be used as term declarations.

So we preferred excluding the term declarations completely from the unification algorithm, as we did for matching. The resulting problem from this extraction is the simplification of the $x \cong_d^? y$ - cases. Therefore we have to be able to compute a finite representation of the intersection of two sorts and to decide whether this intersection is empty or not. These two problems are solved by the restriction to sort inheritance , that guarantees that the intersection of an arbitrary number of sorts must always be representable by a union of sorts.

We continue now with the fundamental proofs for the decidability theorem. As we have already mentioned, the rules of the algorithm can be seen as conditional term rewriting rules. Hence, we still have to prove termination.

Proposition 5.4.7 *The application of the rules in* $\mathbf{UNIF_d}$ *terminates for any input unification problem.*

Proof: We do not have to regard the rules returning $\mathbb{F}$, because $\mathbb{F}$ is irreducible and thus the algorithm terminates when we apply one of those rules. But we still have to prove the termination of the normalization with the six rules **UDecompose**, **UCoalesce**, **UMerge**, **UDelete**, **UErase** and **UIntersection**. The figure 5.6 illustrates this proof.

The principle of the termination proof in [Waldmann, 1989b] can be extended for our algorithm. We define for any unification equation $t \cong_d^? t'$ the following complexity measures $M = max(|t|, |t'|)$, $V = v(t) + v(t')$ and for any decorated term t, $S = w(t)$. Let MM

rule	MM	MV	MS
UDecompose	$<$	$-$	$-$
UMerge	$\leq$	$<$	$-$
UCoalesce	$\leq$	$\leq$	$<$
UIntersection	$=$	$<$	$-$
UDelete, UErase	$<$	$-$	$-$
or	$=$	$<$	$-$

Figure 5.6: sketch of the termination proof of $\mathbf{UNIF_d}$

be the multiset of all M of equations with t or $t' \notin \mathcal{X}_\Diamond$, MV the multiset of Vs of all equations, MS the multiset of all S of terms in the current unification problem and LE the triple (MM, MV, MS) (compared with the usual lexicographic ordering).

UDecompose replaces the equation $f(t_1, \ldots, t_n)^{:S} = f(s_1, \ldots, s_n)^{:S'}$ by equations, that are smaller with respect to M. The new equations will therefore be subsumed by the decomposed equation w.r.t. the multiset ordering MM and thus LE will be strictly decreased.

UMerge yields, by the condition of the rule, a smaller problem, because $x^{:\downarrow\emptyset} \cong_d^? t^{:S}$ is already in the initial problem and for $t^{:S} \cong_d^? t'^{:S'}$, M does not change, because of $|t| \leq |t'|$, but V decreases strictly with respect to $x^{:\downarrow\emptyset} \cong_d^? t'^{:S'}$, because of $v(t) = v(t') = 0$ and $v(()x) > 0$. Remark that $t^{:S} \notin \mathcal{X}_\Diamond$ and $|t^{:S}| \leq |t'^{:S'}|$ implies $t'^{:S'} \notin \mathcal{X}_\Diamond$.

UCoalesce does not change or decreases M and V for all equations in $\mathcal{U}$, but strictly decreases S for at least one equation in $\mathcal{U}$, since $y \in \mathcal{V}ar(\mathcal{U})$ and $y <_V x$.

UIntersection does not change MM, since $x, y, z \in \mathcal{X}_\Diamond$, but strictly decreases MV, because of $sort(z) <_{\mathcal{U}} sort(x), sort(y)$.

Finally, **UDelete** and **UErase** either strictly decrease MM or do not change it and strictly decrease MV.

Thus LE is decreased by every rule and therefore the application of $\mathbf{UNIF_d}$ terminates for arbitrary unification problems. □

The normal form of a unification problem is already well-formed for the extraction of a $\mathcal{D}$ - solution.

Proposition 5.4.8 *The normal form of a unification problem $\mathcal{U}$ is always in solved form.*

Proof: Let $\mathcal{U}_{nf} = \exists z_1, \ldots z_m \bigwedge_{i\in[1..n]} s_i^{:S_i} \cong_d^? t_i^{:T_i}$. We can suppose the finiteness of $\mathcal{U}_{nf}$ since the initial problem is finite and the algorithm terminates due to proposition 5.4.7. Assume that $\mathcal{U}_{nf}$ is not in solved form. If there was an i, s.t. $s_i^{:S_i}, t_i^{:T_i} \notin \mathcal{X}_\Diamond$, then either **UDecompose** or **UConflict** would be applicable.

Furthermore, if there is an $i \in [1..n]$, s.t. $s_i^{:S_i}, t_i^{:T_i} \in \mathcal{X}_\Diamond$, then either $sort(s_i) \leq_S^{syn} sort(t_i)$ or vice versa, since otherwise the rule **UIntersect** or **URemove** would be applicable. Let us assume w.l.o.g. that $sort(s_i) \geq_S^{syn} sort(t_i)$. Then either $s_i = t_i$, making **UDelete** applicable, or s_i does not occur anywhere else in $\mathcal{U}_{nf}$, since **UCoalesce** would be applicable, i.e. if $t_i \in \mathcal{X}_\Diamond$, then point 1 of Definition 5.4.2 is satisfied. Furthermore, if $s_i \in \{z_1, \ldots z_m\}$, then **UErase** is applicable, implying point 3.

From now on, we read all such variable-variable unification equations in $>_V$-decreasing order, i.e. $s_i, t_i \in \mathcal{X}_\Diamond$ implies $s_i >_V t_i$. Assume $t_i \notin \mathcal{X}_\Diamond$ and there is some $j \in [1..n]$ with $i \neq j$, s.t. $s_j = s_i$. If $t_j \in \mathcal{X}_\Diamond$, then $s_j >_V t_j$ and **UCoalesce** would be applicable. If $t_j \notin \mathcal{X}_\Diamond$, then **UMerge** would be applicable. Consequently, point 1 is also satisfied if $t_i \notin \mathcal{X}_\Diamond$.

UCheck* guarantees us that there are no cycles in the occur-check ordering, i.e. we can read the equations in a way such that $\forall 1 \leq i \leq j \leq n \ : \ s_i \notin \mathcal{V}ar(t_j)$, implying point 2. Remark that this is compatible with the preceeding reading convention.

If there was an $k \in [1..m]$, s.t. $z_k \notin \mathcal{V}ar(\bigwedge_{i\in[1..n]} s_i{}^{:S_i} \cong_d^? t_i{}^{:T_i})$, then z_k was introduced by **UIntersection**. One can observe that any rule preserves the property of z_k being a left or right hand side of an equation with a different variable at the other hand side. Therefore, only **UErase** can make z_k disappear. But this is in contradiction with the fact that z_k is still in the quantifier part of $\mathcal{U}_{nf}$.

Finally, point 5 is a direct consequence of rule **UDecoClash**, **URemove** and the fact that we can read variable-variable equations $s_i{}^{:S_i} \cong_d^? t_i{}^{:T_i}$ such that $s_i >_V t_i$. □

Proposition 5.4.9 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and $\mathcal{P}$ be $\mathcal{T}$-sort inheriting.*
All transformation rules in $\mathbf{UNIF_d}$ *are sound and $\mathcal{T}$-complete with respect to the definition of a strict decorated unifier.*

Proof: Let $\mathcal{U}'$ be the initial problem at which we apply the rule, $\mathcal{U}''$ the one obtained by the application and $\mathcal{U}$ the instantiation of the problem variable in the rule. Let furthermore $\mathcal{V}$ stand for the set of variables in $\mathcal{U}'$ that are not existentially quantified. In most of the cases we will show that $SU(\mathcal{U}')_{|_\mathcal{T}} = SU(\mathcal{U}'')_{|_\mathcal{T}}$, implying trivially $SU(\mathcal{U}')_{|_\mathcal{T}} \subseteq SU(\mathcal{U}'')_{|_\mathcal{T}}$ and $\forall \sigma \in SU(\mathcal{U}'')_{|_\mathcal{T}} \ : \ \exists \rho \in SU(\mathcal{U}')_{|_\mathcal{T}} \ : \ \rho \leq^\mathcal{V} \sigma$, which corresponds with definition 5.4.6. Since $SU(\mathcal{U}_1 \wedge \mathcal{U}_2) = SU(\mathcal{U}_1) \cap SU(\mathcal{U}_2)$ for any unification problems $\mathcal{U}_1$ and $\mathcal{U}_2$, it is sufficient to proof the soundness and completeness of the pattern part w.r.t. its replacement in each rule.

- **UDelete** : Obviously $SU(\mathcal{U})_{|_\mathcal{T}} = SU(\mathcal{U} \wedge t \cong_d^? t')_{|_\mathcal{T}}$, because $\sigma(t) \cong_d \sigma(t')$ holds for any valid decorated substitution for t and t'.
- **UDecompose** : Since decorated substitutions are morphisms, $\bigwedge_{i\in[1..n]} \sigma(t_i) \cong_d \sigma(t'_i)$ and $S \approx S'$ is equivalent to $\sigma(f(t_1, \ldots, t_n)^{:S}) \cong_d \sigma(f(t'_1, \ldots, t'_n)^{:S})$. Hence, we know that $SU(\mathcal{U}')_{|_\mathcal{T}} = SU(\mathcal{U}'')_{|_\mathcal{T}}$.
- **UCoalesce** : Obviously, $SU(\mathcal{U}')_{|_\mathcal{T}} = SU(\mathcal{U}'')_{|_\mathcal{T}}$, since this rule replaces equal by equal.
- **UMerge** : $SU(\mathcal{U}')_{|_\mathcal{T}} = SU(\mathcal{U}'')_{|_\mathcal{T}}$, because $\sigma(x^{:\downarrow\emptyset}) \cong_d \sigma(t^{:S})$ and $\sigma(x^{:\downarrow\emptyset}) \cong_d \sigma(t'^{:S'})$ iff $\sigma(x^{:\downarrow\emptyset}) \cong_d \sigma(t^{:S})$ and $\sigma(t^{:S}) \cong_d \sigma(t'^{:S'})$ by the obvious transitivity of $\cong_d$.
- **UErase** : Obviously **UErase** is complete, because $Conj(\mathcal{U}'') \subseteq Conj(\mathcal{U}')$. Suppose σ is a solution for $\mathcal{U}''$. If $t^{:T} \in \mathcal{X}_\Diamond$, then $sort(t) \leq_\mathcal{S}^{syn} sort(z')$ by the condition of the rule. Let $\sigma' = \sigma \cup \{z'^{:\downarrow\emptyset} \mapsto t^{:T}\}$. Clearly, σ' is a decorated substitution, which is a solution for $\mathcal{U}'$ with $\sigma' \cong_d^\mathcal{V} \sigma$, since $z' \notin \mathcal{V}$.
- **UIntersect** : Clearly, $Conj(\mathcal{U}') \subseteq Conj(\mathcal{U}'')$ and therefore **UIntersect** is sound. Suppose $\sigma \in SU(\mathcal{U}')_{|_\mathcal{T}}$ with $\langle U \rangle = Deco(\sigma(x^{:\downarrow\emptyset}))$. Then $\sigma' = \sigma \cup \{z^{:\downarrow\emptyset} \mapsto \sigma(x^{:\downarrow\emptyset})\}$ is a solution for $\mathcal{U}''$ with $\sigma \cong_d^\mathcal{V} \sigma'$, since $z \notin \mathcal{V}$. Remark that σ' is a decorated substitution, since $\{\langle T \rangle\}, \{\langle T' \rangle\} \lesssim U$ and therefore $\{\langle min(T \cup T') \rangle\} \lesssim U$.

- **UConflict** : Obviously, if $f \neq g$ or $m \neq n$ or $S \not\approx T$, then there cannot be a decorated substitution that yields $\sigma(f(t_1,\ldots,t_n)^{:S}) \cong_d \sigma(g(t'_1,\ldots,t'_p)^{:T})$, because substitutions do not change non-variable occurrences of a term. Consequently, there is also no $\mathcal{D}$ - solution for $\mathcal{U}'$ and thus $SU(\mathcal{U}')_{|_\mathcal{T}} = SU(\mathbb{F})_{|_\mathcal{T}} = SU(\mathcal{U}'')_{|_\mathcal{T}} = \emptyset$.
- **UCheck*** : Suppose $\sigma \in SU(\mathcal{U}')_{|_\mathcal{T}}$, i.e. $\sigma(x_1{}^{:\downarrow\emptyset}) \cong_d \sigma(t_1[x_2{}^{:\downarrow\emptyset}]_{p_1})$, ..., $\sigma(x_n{}^{:\downarrow\emptyset}) \cong_d \sigma(t_n[x_1{}^{:\downarrow\emptyset}]_{p_n})$ with some $p_i \neq \epsilon$. Therefore, $\sigma(x_1)$ must have itself as subterm modulo sort inheritance, i.e. the height of $\sigma(x_1{}^{:\downarrow\emptyset})$ is infinite. This is in contradiction with the definition of a term. Hence, there is no $\mathcal{D}$ - solution of $\mathcal{U}'$, like for $\mathcal{U}'' = \mathbb{F}$.
- **UDecoClash**: The rule is obviously sound, because $SU(\mathcal{U}'')_{|_\mathcal{T}}$ is empty. Suppose that there is a $\sigma \in SU(\mathcal{U}')_{|_\mathcal{T}}$. The sort of x has to be in $Deco(\sigma(x^{:\downarrow\emptyset}))$ modulo sort inheritance, because σ is a decorated substitution, but $\{sort(x)\} \not\subseteq T$. Since $\sigma(f(t_1,\ldots,t_n)^{:T}) \cong_d f(\sigma(t_1),\ldots,\sigma(t_n))^{:T}$, $S \not\subseteq Deco(\sigma(x^{:\downarrow\emptyset}))$ and furthermore we know that $Deco(\sigma(x^{:\downarrow\emptyset})) \approx Deco(\sigma(f(t_1,\ldots,t_n)^{:T})) \approx T$ holds and therefore σ cannot exist, i.e. $SU(\mathcal{U}')_{|_\mathcal{T}} = SU(\mathcal{U}'')_{|_\mathcal{T}} = \emptyset$.
- **URemove** : Trivially, this rule is sound. Suppose there is a solution $\sigma \in SU(\mathcal{U}')_{|_\mathcal{T}}$. Then $\{\langle T\rangle\}, \{\langle T'\rangle\} \subsetneq Deco(\sigma(x^{:\downarrow\emptyset}))$ and $\sigma(x^{:\downarrow\emptyset})$ is a valid decorated term in $\mathcal{T}$. But this is in contradiction with the $\mathcal{T}$-sort inheritance of $\mathcal{P}$.

□

Proposition 5.4.10 *Let $\mathcal{T} \subseteq ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $(\Sigma, \mathcal{P})$ be a $\mathcal{T}$-sort inheriting specification w.r.t. $\leq_\mathcal{S}^{syn}$. Then the rules in* $\mathbf{UNIF_d}$ *are sound, $\mathcal{T}$-complete and terminate for every input unification problem $\mathcal{U}$. Let $\mathcal{U}_{sf} = \bigwedge_{i\in I} x_i{}^{:\downarrow\emptyset} \cong_d^? t_i{}^{:T_i}$ be the solved form obtained by the application of* $\mathbf{UNIF_d}$ *on a unification problem $\mathcal{U}$ and let $\sigma = \bigcup_{i\in[1..n]}\{x_i{}^{:\downarrow\emptyset} \mapsto t_i{}^{:T_i}\}$, s.t. $x_i >_V t_i$, if $t_i \in \mathcal{X}_\Diamond$.*

Then $\sigma_{nf} = \sigma^n$ is a $\mathcal{T}$-complete most general unifier for $\mathcal{U}$, i.e. $\{\sigma_{nf}\}$ is a $CSU_\mathcal{P}(\mathcal{U})_{|_\mathcal{T}}$. If $\mathcal{U}_{nf} = \mathbb{F}$, then $\mathcal{U}$ has no strict decorated $\mathcal{T}$-solution and if $\mathcal{U}_{nf} = \mathbb{T}$, then any decorated substitution of terms($\mathcal{U}$) into $\mathcal{T}$ is a strict decorated $\mathcal{T}$-solution of $\mathcal{U}$.

Proof: By the proof of completeness and soundness of the algorithm $\mathbf{UNIF_d}$ we know, that in the cases of $\mathbb{T}$ and $\mathbb{F}$ for $\mathcal{U}_{nf}$ are trivial. In order to prove that σ is a sound and complete $\mathcal{D}$ - solution for the corresponding unification problem $\mathcal{U}_{sf}$, we have to show soundness and completeness of the solution's extraction.

Clearly, σ_{nf} is a decorated substitution due to point 5 of definition 5.4.2, the given uniqueness of variable decorations modulo sort inheritance and proposition 5.2.11. Furthermore it is idempotent by point 2 of definition 5.4.2.

In the following, we will read the equations in $\mathcal{U}_{sf}$ like in proposition 5.4.8, i.e. sort decreasing in the case of variable equations and descending w.r.t. the occurrence ordering.

Let $\sigma_i = \{x_i{}^{:\downarrow\emptyset} \mapsto t_i{}^{:T_i}\}$ and $\sigma'_k = \sigma_k \circ \ldots \circ \sigma_n$. Clearly, $\sigma_{nf} = \sigma'_1$. For all $x_i{}^{:\downarrow\emptyset} \cong_d^? t_i{}^{:T_i}$ in $\mathcal{U}_{sf}$, we have $\sigma_i(x_i{}^{:\downarrow\emptyset}) \cong_d \sigma_i(t_i{}^{:T_i}) =_d t_i{}^{:T_i}$ by construction of σ_i and the fact that $x_i \notin Var(t_i{}^{:T_i})$. Since none of the x_j with $1 \leq j \leq i$ occurs in $t_i{}^{:T_i}$ and the fact that $\cong_d$ is obviously stable under decorated substitution, we can also be sure that $\sigma'_i(x_i{}^{:S_i}) \cong_d \sigma'_i(t_i{}^{:T_i})$. Consequently, by induction over i, we get $\sigma_{nf}(x_i{}^{:S_i}) \cong_d \sigma_{nf}(t_i{}^{:T_i})$ for all $i \in [1..n]$, i.e. σ_{nf} is a solution and the extraction is sound.

Let now the final problem be furthermore transformed by an exhaustive application of the variable elimination rule **UEliminate** illustrated in figure 5.7. Obviously, this rule is

UEliminate $\mathcal{U} \wedge x^{:\downarrow\emptyset} \cong_d^? t^{:T}$
$\Longrightarrow \mathcal{U}\{x^{:\downarrow\emptyset} \mapsto t^{:T}\} \wedge x^{:\downarrow\emptyset} \cong_d^? t^{:T}$
if $x \in \mathcal{X}_\Diamond$ and $sort(x) \subsetneq T$ and $t \notin \mathcal{X}_\Diamond$

Figure 5.7: Variable elimination rule for completeness proof

sound and complete and its application terminates. Furthermore, there exists a problem $\mathcal{U}'_{sf} = \bigwedge_{x \in \mathcal{D}om(\sigma_{nf})} x^{:\downarrow\emptyset} \cong_d^? \sigma_{nf}(x^{:\downarrow\emptyset})$, that can be obtained with **UEliminate** from $\mathcal{U}_{sf}$ – just apply this rule for all equations (if possible) starting with index n until 1 and you get the construction of $\sigma_{nf} = \sigma'_1$ as above.

Now, suppose τ is a solution of $\mathcal{U}$. Hence, there is also a solution $\overline{\tau}$ of $\mathcal{U}_{sf}$ and therefore a solution τ' of $\mathcal{U}'_{sf}$, s.t. $\tau' \lesssim_d^{\mathcal{V}ar(\mathcal{U})} \overline{\tau} \lesssim_d^{\mathcal{V}ar(\mathcal{U})} \tau$ by completeness, i.e. $\tau' \lesssim_d^{\mathcal{V}ar(\mathcal{U})} \tau$ by transitivity of $\lesssim_d^{\mathcal{V}ar(\mathcal{U})}$.

Therefore, $\tau'(x^{:\downarrow\emptyset}) \cong_d \tau'(\sigma_{nf}(x^{:\downarrow\emptyset}))$ for all $x \in \mathcal{V}ar(\mathcal{U})$ and consequently $\tau' \circ \sigma_{nf} \cong_d^{\mathcal{V}ar(\mathcal{U})} \tau'$ implying $\sigma_{nf} \lesssim_d^{\mathcal{V}ar(\mathcal{U})} \tau'$ by proposition 5.2.14. By transitivity we get $\sigma_{nf} \lesssim_d^{\mathcal{V}ar(\mathcal{U})} \tau$.

Hence, the solution extraction is also complete. □

Corollary 5.4.11 *Let $\mathcal{T} \subseteq \mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and $(\Sigma, \mathcal{P})$ be a $\mathcal{T}$-sort inheriting specification w.r.t. $\leq_{\mathcal{S}}^{syn}$.*

Then sound, $\mathcal{T}$-complete, strict decorated $\mathcal{D}$-unification is decidable.

Finally, let us illustrate how the algorithm $\mathbf{UNIF_d}$ works pn the two examples that we gave above:

Example 5.4.12 *Let $\mathcal{P} = \{x :: A, y :: B, x : B, a : A\}$ and:*

$$\mathcal{U} = (x^{:\downarrow\emptyset} \cong_d^? a^{:\{B\}} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset} \wedge y^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}}).$$

This is transformed by **UCoalesce** *and* **UMerge** *into:*

$$\mathcal{U}' = (x^{:\downarrow\emptyset} \cong_d^? a^{:\{B\}} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset} \wedge a^{:\{B\}} \cong_d^? a^{:\{A,B\}}),$$

which clashes because of **UConflict**. *Hence, $\mathcal{U}$ has no $\mathcal{D}$-unifier, because $\sigma(x^{:\downarrow\emptyset})$ should have the decorations $\{B\}$ and $\{A, B\}$ at the same time, but they are not equal modulo sort inheritance. Clearly, the required equality of the decorations of the potential images for x and y prohibited the solvability of $\mathcal{U}$, although $\mathcal{P} \models_G a : A$ is trivial. Only if the sort A is already in the decoration of a, then there is a $\mathcal{D}$ - solution of $\mathcal{U}$.*

Example 5.4.13 *Let $\mathcal{P} = \{x :: A, y :: B, x : B, a : A\}$ and:*

$$\mathcal{U} = (x^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset} \wedge y^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}}).$$

This is transformed by **UCoalesce**, **UMerge** *and* **Delete** *yielding:*

$$\mathcal{U}' = (x^{:\downarrow\emptyset} \cong_d^? a^{:\{A,B\}} \wedge x^{:\downarrow\emptyset} \cong_d^? y^{:\downarrow\emptyset}),$$

which is in solved form. Hence, $\mathcal{U}$ has the $\mathcal{D}$-unifier:

$$\sigma = \{x^{:\downarrow\emptyset} \mapsto a^{:\{A,B\}}, y^{:\downarrow\emptyset} \mapsto a^{:\{A,B\}}\}.$$

5 Subterm Conservative Solutions

In order to solve typing problems, we need to ensure that in solutions consisting of substitution sets, the images of the substitutions are subterms of the original problem. This gives us the bottom-up typability of the terms in the image, if the terms in the initial problem were bottom-up typable – a very useful property for doing some of the induction proofs needed in the sequel. The following definition intuitively says that any term in $\Im\sigma$ is equal to a subterm in Pb modulo a specialisation.

Definition 5.5.1 *Let $\mathcal{C}$ be a class of problems Pb, whose minimal, complete set of solutions can be described as a set of decorated substitutions $MCSol(Pb)$ in dag-solved form (see [Jouannaud et Kirchner, 1991]).*

$\mathcal{C}$ is subterm conservative *if for all problems $Pb \in \mathcal{C}$ there is a $MCSol(Pb)$ that is subterm conservative, i.e. for all $\sigma \in MCSol(Pb)$ and for all terms $t \in \mathcal{I}m(\sigma)$, there exist a decorated substitution τ, which is a specialisation [Waldmann, 1992], i.e. τ maps variables to variables, but maybe not a renaming, since the sorts may decrease, and there is a $t' \in subterm\_set(Pb)$* s.t. *$t \cong_d \tau(t')$.*

$\mathcal{C}$ is strictly subterm conservative *if $\mathcal{C}$ is subterm conservative and $\mathcal{D}om(\tau) = \emptyset$, i.e. τ is the identity substitution.*

Therefore a problem class is subterm conservative, if there is a way to express a minimal complete set of solutions by a set of dag-solved form substitutions constructed from subterms in the original problem modulo variable renaming/specialization, and strictly subterm conservative if there is no variable renaming.

Proposition 5.5.2 *Let $(\Sigma, \mathcal{P})$ be a sort inheriting specification. Then strict decorated unification is subterm conservative and strict decorated matching is strictly subterm conservative.*

Proof: The following proof is only a sketch.

For strict decorated matching, the claim is trivial since we can only change the variables of the left-hand side of a matching equation. Therefore the introduction of a subterm that does not belong to the original problem leads immediately to a contradiction with the definition of a solution.

In the case of strict decorated unification, we can use the fact that $\mathbf{UNIF_d}$ yields a complete solution set as singleton $\{\sigma_{nf}\}$. Therefore the minimal complete set of solutions is unique up to variable renaming.

No rule in the unification algorithm decomposes a term that is bound to a variable, neither constructs a new term. The only new term parts introduced are variables. But this is done by decreasing the sort w.r.t. the 'old' variable which is bound to the new one.

Together with the fact that variable decorations of new variables do never change, since there are by definition no old occurrences of the same variable, this guarantees that for any substitution σ in the solution set, there must also be a decorated substitution τ, that is a specialisation, and a $t' \in subterm\_set(Pb)$ s.t. $t \cong_d \tau(t')$ for all terms in $\mathcal{I}m(\sigma)$. □

This proposition would not necessarily hold if we were using term declarations during the unification, like for unification in non-regular theories, or other mechanisms introducing new term parts, like equational theories, etc. However, semantical arguments could help us finding the typability of these newly introduced term parts.

Chapitre 6

Rewriting Decorated Terms

In this chapter, we define equational replacement for decorated terms. As already outlined in the introduction to this part of the thesis, the main goal is to integrate typing proofs into equational reasoning. This leads us to an alternative way of deduction w.r.t. **GA** and the basis for a completion algorithm.

Following the classical approach to term rewriting, we define equalities and rewrite rules on decorated terms, used for replacement of equal by equal, the former unoriented, the latter oriented. Furthermore, we add decoration rewrite rules, which do not change the structure of a term, but add decorations. Elementary properties of decorated term rewriting are investigated and allow us to give a Birkhoff Theorem stating completeness of deduction using decorated equalities and decoration rules extracted from a G-algebra presentation.

1 Decorated Term Rewriting Systems

1.1 Decorated Equalities

Definition 6.1.1 *A* decorated equality *is a pair of two decorated terms, denoted by* $(p^{:S} = q^{:S'})$ *where* $p^{:S}, q^{:S'} \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$.

The next definition of replacement of equal by equal, as well as the definition of a rewriting step, are given for a set of positions in order to later define a notion of parallel reduction.

Definition 6.1.2 *Let* $t^{:T}, t'^{:T'}, p^{:S}, q^{:S'} \in Valid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. *We say that* $t^{:T}$ *is* one-step-equal *to* $t'^{:T'}$, *using the set of decorated equalities* E, *if there exist a set of incomparable (w.r.t.* $\leq$*) occurrences* $O \neq \emptyset$ *with* $t^{:T} \cong_d t^{:T}[u^{:U}]_O$, $t'^{:T'} \cong_d t'^{:T'}[u'^{:U'}]_O$, *a decorated substitution* σ *and a decorated equality* $\phi = (p^{:S} = q^{:S'})$ *in* E, *such that* σ *is a solution of* $(p^{:S} \lesssim_d^? u^{:U} \wedge q^{:S'} \lesssim_d^? u'^{:U'})$. *This is denoted by* $t^{:T} \longleftrightarrow_E^{O,\sigma,\phi} t'^{:T'}$. *If* O *is a singleton* $\{\omega\}$, *we write* $t^{:T} \longleftrightarrow_E^{\omega,\sigma,\phi} t'^{:T'}$

Remark that this relation is in general undecidable, since validity of terms is undecidable. For deciding the preservation of validity by decorated equational replacement we also need the decidability of variable image validity.

1.2 Decorated Rewriting

A decorated rewrite rule is an oriented pair of decorated terms. It is applied on a decorated term by $\mathcal{D}$-matching the left-hand side to some subterm.

Definition 6.1.3 *A* decorated rewrite rule *is a pair of decorated terms denoted by* $(l^{:S_l} \to r^{:S_r})$, *where* $l^{:S_l}, r^{:S_r} \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, *such that* $S_l \lesssim S_r$ *and* $\mathcal{V}ar(l^{:S_l}) \supseteq \mathcal{V}ar(r^{:S_r})$. *A* decorated rewrite system *is a set of decorated rewrite rules.*

Definition 6.1.4 *A decorated term* $t^{:S}$ rewrites *to* $t'^{:S'}$ *using the decorated rewrite system* R *if there exist an occurrence set* $O \neq \emptyset$ *with* $t^{:S} \cong_d t^{:S}[u^{:U}]_O$, *a decorated substitution* σ *and a decorated rewrite rule* $\phi = (l^{:S_l} \to r^{:S_r})$ *in* R *such that:*

1. σ *is a* $\mathcal{D}$*-match from* $l^{:S_l}$ *to* $u^{:U}$,
2. $t'^{:S'} =_d t^{:S}[\sigma(r^{:S_r})]_O$.

This is denoted by $t^{:S} \rightarrow_R^{O,\sigma,\phi} t'^{:S'}$. *If* O *is a singleton* $\{\omega\}$, *we also write* $t^{:S} \rightarrow_R^{\omega,\sigma,\phi} t'^{:S'}$. *The symmetric closure of* $\rightarrow_R$ *is written* $\leftarrow\!\!\rightarrow_R$.

Notice the accumulation of decorations at the redex occurrence, due to the condition $S_l \lesssim S_r$ for decorated rewrite rules.

Example 6.1.5 *Let* $a^{:\{A\}} \to b^{:\{A,B\}}$ *be a decorated rewrite rule. Then* $f^{:\emptyset}(a^{:\emptyset})$ *cannot be rewritten, but* $f^{:\emptyset}(a^{:\{A\}}) \rightarrow f^{:\emptyset}(b^{:\{A,B\}})$.

The symmetric closure of the rewrite relation is written $t^{:S} \leftarrow\!\!\rightarrow_R^{\omega,\sigma,\phi} t'^{:S'}$. Remark that this relation is also in general undecidable, since validity of terms is undecidable. However, the inclusion of the set of variables in the right-hand side in the one of the right-hand sides allows us to propagate validity. We can conclude that $\rightarrow_E$ is a conservative extension of $\rightarrow_R$, i.e. any decorated rewriting step can be seen as application of an equality.

1.3 Decoration Rewriting

We need to introduce decoration rewrite rules, whose purpose is to locally increment the set of sorts associated to some node in a decorated term. A decoration rewrite rule is given by a decorated term, say l, a set-variable $s \in \mathbf{V}$ that decorates the left-hand side, a sort expression $s \cup S_l$, where S_l is a set of sorts, to decorate the right-hand side, and a condition $c(s)$ depending on the variable s. Applying to $t^{:U}$ a decoration rewrite rule $(l^{:s} \to l^{:s \cup S_l}$ if $c(s))$ amounts to $\mathcal{D}$-match the decorated term $l^{:U}$ to the decorated term $t^{:U}$, to check the condition $c(s)$ when s takes the value U and to enrich the decoration U with S_l if $c(U)$ is true.

Definition 6.1.6 *A* decoration rewrite rule *is a conditional rewrite rule denoted by* $(l^{:s} \to l^{:s \cup S_l}$ if $S_l \lesssim s)$ *where* $l^{:S_l} \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, *such that* $s \in \mathbf{V}$, $l \notin \mathcal{X}_\Diamond$ *and* $S_l \subseteq \mathcal{S}_\Diamond$ *is a set of sorts. A* decoration rewrite system *is a set of decoration rewrite rules.*

Definition 6.1.7 *A decorated term* $t^{:S}$ rewrites *to* $t'^{:S'}$ *using the decoration rewrite system* D *if there exist a set of occurrences* $O \neq \emptyset$ *with* $t^{:S} \cong_d t^{:S}[u^{:U}]_O$, *a decorated substitution* σ *and a decoration rewrite rule* $\phi = (l^{:s} \to l^{:s \cup S_l}$ if $S_l \lesssim s)$ *such that:*

1. σ *is a* $\mathcal{D}$*-match from* $l^{:U}$ *to* $u^{:U}$,

2. $S_l \not\subseteq U$,

3. $t'^{:S'} =_d t^{:S}[\sigma(l^{:U})^{:U \cup S_l}]_O$.

This is denoted by $t^{:S} \rightarrowtail_D^{O,\sigma,\phi} t'^{:S'}$. *If* O *is a singleton* $\{\omega\}$, *this is also written* $t^{:S} \rightarrowtail_D^{\omega,\sigma,\phi} t'^{:S'}$. *The symmetric closure of* $\rightarrowtail_D$ *is written* $\leftrightarrowtail_D$.

Example 6.1.8 *Let* $(a^{:s} \rightarrow a^{:s \cup \{A\}}$ if $\{A\} \not\subseteq s)$ *be a decoration rewrite rule. Then we have* $f(a^{:\{B\}})^{:\emptyset} \rightarrowtail_D f(a^{:\{A,B\}})^{:\emptyset}$.
Let $(f(x^{:\{A\}})^{:s} \rightarrow f(x^{:\{A\}})^{:s \cup \{A\}})$ if $\{A\} \not\subseteq s$ *be another decoration rewrite rule. Therefore* $f(a^{:\{A,B\}})^{:\emptyset} \rightarrowtail_D f(a^{:\{A,B\}})^{:\{A\}}$.

Given a pair (D, R) of decorated rewrite rules and decoration rules, we define for two decorated terms $t^{:S}$ and $t'^{:S'}$, $t^{:S} \rightarrowtail_{D \cup R} t'^{:S'}$ if $t^{:S} \rightarrowtail_R t'^{:S'}$ or $t^{:S} \rightarrowtail_D t'^{:S'}$.

From now on, $rhs(\phi)$ and $lhs(\phi)$ denote as usual the right-hand side and left-hand side, respectively, of a decorated/decoration rewrite rule ϕ.

2 Elementary Properties of Rewriting

In this section, we prove the soundness of deduction with the relations introduced in the last section and the decidability of the decorated/decoration rewrite relation in any sequence starting with a term with empty decorations. We prove furthermore stability of decorated/decoration rewriting under context and substitution.

Let us first introduce the notions of canonical validity and theorems associated to sets of decorated and decoration rewrite rules and equalities. Here we use extensively the $\mathcal{T}(\Sigma, \mathcal{X})$-assignments in order to link decorated terms in $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and G-algebra terms in $\mathcal{T}(\Sigma, \mathcal{X})$.

Definition 6.2.1 *For any sets of decoration rewrite rules* D, *decorated rewrite rules* R *and decorated equalities* E, *we say that:*

1. $(l^{:s} \rightarrow l^{:s \cup S_l}$ if $S_l \not\subseteq s) \in D$ *is* valid *in* $\mathcal{P}$, *if* $l^{:S_l}$ *is valid in* $\mathcal{P}$.

2. $(l^{:S_l} \rightarrow r^{:S_r}) \in R$ *is* valid *in* $\mathcal{P}$, *if* $l^{:S_l}, r^{:S_r}$ *are valid in* $\mathcal{P}$ *and for all* $\alpha \in \mathcal{VASS}_{\mathcal{T}(\Sigma,\mathcal{X})}^{\mathcal{X}_\Diamond}$, $\mathcal{P} \vdash_G \alpha(l^{:S_l}) = \alpha(r^{:S_r})$.

3. $(p^{:S_p} = q^{:S_q}) \in E$ *is* valid *in* $\mathcal{P}$ *if* $p^{:S_p}, q^{:S_q}$ *are valid in* $\mathcal{P}$ *and for all* $\alpha \in \mathcal{VASS}_{\mathcal{T}(\Sigma,\mathcal{X})}^{\mathcal{X}_\Diamond}$, $\mathcal{P} \vdash_G \alpha(p^{:S_p}) = \alpha(q^{:S_q})$.

Definition 6.2.2 *To any sets of valid decoration rewrite rules* D, *of valid decorated equalities* E *and valid decorated rewrite rules* R, *we associate canonically a set of formulas* $Form(D, E, R)$, *that is the union of:*

1. $\{t = t' \mid t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$ and $\exists t_0 \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond), A \in \mathcal{S}_\Diamond$, s.t. $t^{:\downarrow\emptyset} \leftrightarrowtail^*_{D \cup E \cup R} t_0^{:\{A\} \cup U} \leftrightarrowtail^*_{D \cup E \cup R} t'^{:\downarrow\emptyset}\}$,

2. $\{t : A \mid t \in \mathcal{T}(\Sigma, \mathcal{X})$ and $\exists t' \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond), A' \in \mathcal{S}_\Diamond$ with $A' \leq^{syn}_{\mathcal{S}_\Diamond} \langle A \rangle$, s.t. $t^{:\downarrow\emptyset} \leftrightarrowtail^*_{D \cup E \cup R} t'^{:\{A'\} \cup U}\}$ *and*

3. $\{EX\ t \mid t \in \mathcal{T}(\Sigma, \mathcal{X})$ and $\exists t' \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond), A \in \mathcal{S}_\Diamond$, s.t. $t^{:\downarrow\emptyset} \leftrightarrowtail^*_{D \cup E \cup R} t'^{:\{A\} \cup U}\}$.

Remark that $Form(D, E, R)$ for arbitrary, not valid D, E, R is not a theory, i.e. it is not necessarily closed under G-algebra inferences, but for those that we consider from Section 6.3 on, this will be provably (Lemma 6.2.8, Theorem 6.5.1 and Proposition 7.2.3) the case.

The following properties of rewriting are easy to check but quite useful in what follows.

Lemma 6.2.3 *Let $t^{:S}$ be a valid term and ϕ a valid element of $D \cup E \cup R$ in $\mathcal{P}$. If $t^{:S} \longleftrightarrow^{\omega,\sigma,\phi}_{D\cup E\cup R} t'^{:S'}$ then:*

1. $\forall A \in \mathcal{S}, \forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G \alpha(t^{:S}) : A)$ *iff* $(\mathcal{P} \vdash_G \alpha(t'^{:S'}) : A)$,
2. $\forall \langle A, \ldots\rangle \in S' \setminus S, \forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G \alpha(t^{:S}) : A)$,
3. $\forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G EX\ \alpha(t^{:S})) \Rightarrow (\mathcal{P} \vdash_G \alpha(t^{:S}) = \alpha(t'^{:S'}))$ *and*
4. $t'^{:S'}$ *is also valid in* $\mathcal{P}$.

Proof: (1) If $\phi \in D$, then $t^{:S} =_{nd} t'^{:S'}$ and therefore the property is trivial. Otherwise $\phi \in E \cup R$ and we can easily construct a proof using the axioms **Symmetry**, **EqSubstitutivity** and **MeReplacement**, where the validity of the terms in $\mathcal{I}m(\sigma)$ is needed. The validity of these terms is automatically provided if $\phi \in R$ and the rule is applied left-to-right, since $t^{:S}$ is valid and strict decorated matching is subterm conservative.

(2) This follows immediately from (4) and (1).

(3) With the help of **Reflexivity**, **Symmetry**, **EqSubstitutivity** and **EqReplacement**, we can easily construct the needed proof. Remark that the precondition $\mathcal{P} \vdash_G EX\ \alpha(t^{:S})$ is needed for **Reflexivity** whenever $S = \emptyset$.

(4) If $\phi \in D$, we can use the formula corresponding to ϕ and apply **MeSubstitutivity** to it, when ϕ is applied left-to-right, else the validity of $t'^{:S'}$ is subsumed by the one of $t^{:S}$.

Otherwise $\phi \in E \cup R$ and we have to take the different cases for the occurrences ν into consideration, where we want to prove the validity of the decoration. If $\nu \asymp \omega$, then validity of $(t'^{:S'})_{|_\nu}$ is equivalent to the one of $(t^{:S})_{|_\nu}$. If $\nu \leq \omega$, then we can construct a proof as in (1). If $\nu > \omega$, then the validity of $(t'^{:S'})_{|_\nu}$ follows from the validity of ϕ and those in $\mathcal{I}m(\sigma)$, using axiom **MeSubstitutivity**. Remark that the validity of the terms in $\mathcal{I}m(\sigma)$ is subsumed by the one of $t^{:S}$ if $\phi \in R$ is applied left-to-right. □

Analysing the proof of Lemma 6.2.3 for the left-to-right application of ϕ leads to:

Corollary 6.2.4 *Let $t^{:S}$ and ϕ be valid in $\mathcal{P}$. If $t^{:S} \rightarrow^{\omega,\sigma,\phi}_{D\cup R} t'^{:S'}$ then all terms in $\mathcal{I}m(\sigma)$ are valid. Furthermore:*

1. $(\exists \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})} : (\mathcal{P} \vdash_G \alpha(t^{:S}) : A))$ *iff* $(\exists \alpha' \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})} : (\mathcal{P} \vdash_G \alpha'(t'^{:S'}) : A))$,
2. $\forall \langle A, \ldots\rangle \in S' \setminus S,\ \forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G \alpha(t^{:S}) : A)$,
3. $\forall \alpha \in \mathcal{VASS}^{t^{:S}}_{\mathcal{T}(\Sigma,\mathcal{X})} \cap \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G EX\ \alpha(t'^{:S'})) \Rightarrow (\mathcal{P} \vdash_G \alpha(t^{:S}) = \alpha(t'^{:S'}))$ *and*
4. $t'^{:S'}$ *is also valid in* $\mathcal{P}$.

Therefore we don't need to check the $\mathcal{P}$-validity of terms neither the $\mathcal{P}$-conformity of substitutions in rewriting proofs since clearly, this result extends to multiple rewrite rule applications. The following statement tells that a decoration can be inherited along replacement of equal by equal using valid D, E and R.

Lemma 6.2.5 *For all decorated terms $t^{:S}, t'^{:S'}$, where $t^{:S}$ and $D \cup E \cup R$ are valid in $\mathcal{P}$ and $t^{:S} \longleftrightarrow^*_{D \cup E \cup R} t'^{:S'}$,*

$$\forall \langle A, \ldots \rangle \in S \cup S', \forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})},\ (\mathcal{P} \vdash_G \alpha(t^{:S}) : A).$$

Thus if $t^{:S} \longleftrightarrow^*_{D \cup E \cup R} t'^{:S'}$, then $t^{:S \cup S'}$ is a valid decorated term.

Proof: If A belongs to S, it is obvious by the validity of $t^{:S}$. Otherwise if A belongs to $S' \setminus S$, let us prove it by induction on the length of $\longleftrightarrow^*_{D \cup E \cup R}$. If this length is zero the result is clear. Otherwise

$$t^{:S} \longleftrightarrow_{D \cup E \cup R} t_1{}^{:S_1} \longleftrightarrow^*_{D \cup E \cup R} t'^{:S'}$$

and by the induction hypothesis, $\langle A, \ldots \rangle \in S'$ implies that $\mathcal{P} \vdash_G \alpha(t_1{}^{:S_1}) : A$. Now $(\alpha(t^{:S}) : A)$ is a consequence of Lemma 6.2.3. □

This usefully extends to rewriting terms with empty decorations:

Lemma 6.2.6 *If $D \cup R$ is valid in $\mathcal{P}$ and $t^{:\downarrow\emptyset} \rightarrow^*_{D \cup R} t'^{:S'}$, then:*

1. $\forall A \in \mathcal{S}, \forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G \alpha(t^{:\downarrow\emptyset}) : A)$ *iff* $(\mathcal{P} \vdash_G \alpha(t'^{:S'}) : A)$,
2. $S' \neq \emptyset \Rightarrow \forall \alpha \in \mathcal{VASS}^{\mathcal{X}_\diamond}_{\mathcal{T}(\Sigma,\mathcal{X})}$: $(\mathcal{P} \vdash_G \alpha(t^{:\downarrow\emptyset}) = \alpha(t'^{:S'}))$ *and*
3. $t'^{:S'}$ *is valid in* $\mathcal{P}$.

Proof: This can be shown by induction on the length of the rewriting proof. If the length is zero, $t^{:\downarrow\emptyset} \cong_d t'^{:S'}$ and therefore the lemma is trivial. Otherwise we cut off the last step:

$$t^{:\downarrow\emptyset} \rightarrow^*_{D \cup R} t_0{}^{:S_0} \rightarrow_{D \cup R} t'^{:S'}.$$

The induction hypothesis gives us the validity of $t_0{}^{:S_0}$ and the equivalence of membership for $t^{:\downarrow\emptyset}{}_{nd}$ and $(t_0{}^{:S_0})_{nd}$. For the last step we now can apply corollary 6.2.4. □

Remark that rewrite sequences starting with terms with empty decorations avoid the undecidability of the rewrite relation on decorated terms through undecidable $\mathcal{P}$-conformity checks for the used matcher, since the validity of the decorated variable instantiations follows from the validity of the rewritten term. In order to show that the validity-condition is needed for deduction with decorated rewrite rules in the reverse direction, consider the following example:

Example 6.2.7 *Let $\mathcal{P} = \{a : A, b : B, x :: A, y :: B, f(x,y) = x\}$ and $D \cup R$ be:*

$$\begin{array}{rcll} f(x^{:\{A\}}, y^{:\{B\}})^{:\emptyset} & \rightarrow & x^{:\{A\}} & \\ a^{:s} & \rightarrow & a^{:s \cup \{A\}} & \text{if } \{A\} \not\subseteq s \\ b^{:s} & \rightarrow & b^{:s \cup \{B\}} & \text{if } \{B\} \not\subseteq s \end{array}$$

Then without checking the validity of the terms in the image of the used substitution we would have $a^{:\{A\}} \longleftrightarrow_R f(a^{:\{A\}}, a^{:\{B\}})^{:\emptyset}$, but $\mathbf{P} \not\models_G a = f(a,a)$. The reason for this is that $f(a^{:\{A\}}, a^{:\{B\}})^{:\emptyset}$ is not valid, because of $a^{:\{B\}}$, since a does not belong to B in $\mathbf{P}$.

Clearly, if we allowed non-valid terms in the range of substitutions, then we could prove arbitrary membership formulas, provided we interpret all terms generated by $\leftarrow\!\!\!\!\!\!\rightarrow$ as valid. Another consequence of Lemma 6.2.3 is the soundness of proofs in $\leftarrow\!\!\!\!\!\!\rightarrow_{D\cup E\cup R}$.

Lemma 6.2.8 *Let D be a set of decoration rewrite rules, R a set of decorated rewrite rules and E a set of decorated equations, s.t. all $\phi' \in D \cup E \cup R$ are valid.*
Then $Form(D, E, R) \subseteq Th(\mathcal{P})$

Proof: Let ϕ be a formula in $Form(D, E, R)$. Then Definition 6.2.2 says, that there must be a proof

$t^{:\downarrow\emptyset} \leftarrow\!\!\!\!\!\!\rightarrow_{D\cup E\cup R} t_0^{:\{A'\}\cup U} \leftarrow\!\!\!\!\!\!\rightarrow_{D\cup E\cup R} t'^{:\downarrow\emptyset}$ if $\phi = (t = t')$,
$t^{:\downarrow\emptyset} \leftarrow\!\!\!\!\!\!\rightarrow_{D\cup E\cup R} t_0^{:\{A'\}\cup U}$ if $\phi = (t : A)$ and
$t^{:\downarrow\emptyset} \leftarrow\!\!\!\!\!\!\rightarrow_{D\cup E\cup R} t_0^{:\{A'\}\cup U}$ if $\phi = (EX\ t)$,

s.t. $A' \leq^{syn}_{\mathcal{S}_\emptyset} \langle A \rangle$, if $\phi = (t : A)$. By definition of $\leq^{syn}_{\mathcal{S}_\emptyset}$ (see page 145), there exists a $B \in \mathcal{S}$, s.t. $A' = \langle B, \ldots \rangle$ with $B \leq^{syn}_{\mathcal{S}} A$ in this case.

Let us start with $\phi = (t : A)$. Since $\leftarrow\!\!\!\!\!\!\rightarrow_{D\cup E\cup R}$ preserves membership and validity of terms (Lemma 6.2.5) and $t^{:\downarrow\emptyset}$ is valid, we are sure that for all $\alpha \in \mathcal{VASS}^{\mathcal{X}_\emptyset}_{\mathcal{T}(\Sigma,\mathcal{X})}$, that $\mathcal{P} \vdash_G (\alpha(t_0^{:\{A'\}\cup U}) : B)$ holds, as well as $\mathcal{P} \vdash_G (\alpha(t^{:\downarrow\emptyset}) : B)$. From $B \leq^{syn}_{\mathcal{S}} A$ follows, that there is some variable x of sort B, s.t. $\mathcal{P} \vdash_G (x : A)$. Hence, **MeSubstitutivity** applied to $(x : A)$ with $\sigma = \{x \mapsto \alpha(t^{:\downarrow\emptyset})\}$ gives a proof of $\mathcal{P} \vdash_G \alpha(t^{:\downarrow\emptyset}) : A$.

In the case of $\phi = (t = t')$, we get by Lemma 6.2.6 that for all $\alpha \in \mathcal{VASS}^{\mathcal{X}_\emptyset}_{\mathcal{T}(\Sigma,\mathcal{X})}$, $\mathcal{P} \vdash_G \alpha(t^{:\downarrow\emptyset}) = \alpha(t_0^{:\{A'\}\cup U})$ and $\mathcal{P} \vdash_G \alpha(t'^{:\downarrow\emptyset}) = \alpha(t_0^{:\{A'\}\cup U})$. Hence we get $\mathcal{P} \vdash_G \alpha(t^{:\downarrow\emptyset}) = \alpha(t'^{:\downarrow\emptyset})$ by **Transitivity** and **Symmetry**.

If $\phi = (EX\ t)$, then let w.l.o.g. $A' = \langle C, \ldots \rangle$ with $C \in \mathcal{S}$. Consequently, Lemma 6.2.6 implies that for all $\alpha \in \mathcal{VASS}^{\mathcal{X}_\emptyset}_{\mathcal{T}(\Sigma,\mathcal{X})}$, $\mathcal{P} \vdash_G \alpha(t^{:\downarrow\emptyset}) : C$. Now, we get $\mathcal{P} \vdash_G EX\ \alpha(t^{:\downarrow\emptyset})$ by **ExMembership**.

Since $t, t' \in \mathcal{T}(\Sigma, \mathcal{X})$, we can choose in any of the three cases above an α, s.t. $\alpha_{|_{Var(t)}}$ is the identity, i.e. $Form(D, E, R) \subseteq Th(\mathcal{P})$. Note that we are sure that all sorts are non-empty, thanks to General Assumption 5.1.6. □

Decorated and decoration rewriting are stable by context and substitution:

Proposition 6.2.9 *If $t^{:S} \rightarrow^{\omega,\sigma,\phi}_{D\cup R} t'^{:S'}$:*

1. *for every decorated substitution σ for $t^{:S}$, $\sigma(t^{:S}) \rightarrow^{\omega,\sigma,\phi}_{D\cup R} \sigma(t'^{:S'})$,*
2. *for every decorated term u and any position v in u, $u[t^{:S}]_v \rightarrow^{v.\omega,\sigma,\phi}_{D\cup R} u[t'^{:S'}]_v$.*

Proof: This comes from the fact that substitution and context act on both sides of the rewrite step in an identical way, thus leaving the sort information in the same respective situation. Let us detail this. Assume that $t^{:S} \rightarrow^{\omega,\sigma,\phi}_{D\cup R} t'^{:S'}$ and $\phi : l^{:S_l} \rightarrow r^{:S_r}$ respectively $\phi : l^{:s} \rightarrow l^{:s\cup T}$ if $T \subsetneq s$. Let furthermore $t^{:S} =_d t^{:S}[u^{:U}]_\omega$.
In the case of decorated rewriting, $\sigma(l^{:S_l}) \cong_d t^{:S}{}_{|_\omega}$ and thus for the context $C[]$ whose hole is at occurrence v we have $\sigma(l^{:S_l}) \cong_d C[t^{:S}]_{|_{v.\omega}}$ which proves that $C[t]$ rewrites to $C[t']$.
In the case of decoration rewriting, $\sigma(l^{:U}) =_d (t_{|_\omega})^{:U}$ and thus for the context $C[]$ whose hole is at occurrence v we have $\sigma(l^{:U}) \cong_d C[t^{:S}]_{|_{v.\omega}}$ which proves that $C[t]$ rewrites to $C[t']$.
The stability by substitution is a consequence of the definitions. □

1. $(u^{:\downarrow\emptyset})^{:s} \rightarrow (u^{:\downarrow\emptyset})^{:s\cup\{\Omega\}}$ if $\{\Omega\} \not\subseteq s$
   For each non-variable subterm u of a term v appearing in a formula in $\mathcal{P}$,
   For such a rule, $u^{:\downarrow\emptyset} \in \mathcal{T}_d(\mathcal{S}_\diamond, \mathcal{F}, \mathcal{X}_\diamond)$, $s \in \mathbf{V}$.

2. $(u^{:\downarrow\emptyset})^{:s} \rightarrow (u^{:\downarrow\emptyset})^{:s\cup\{A\}}$ if $\{A\} \not\subseteq s$
   For all $(u : A)$ in $\mathcal{P}$, s.t. $u \notin \mathcal{X}$.
   For such a rule, $u^{:\downarrow\emptyset} \in \mathcal{T}_d(\mathcal{S}_\diamond, \mathcal{F}, \mathcal{X}_\diamond)$, $s \in \mathbf{V}$, $A \in \mathcal{S}$.

$D_\mathcal{P}$: decoration rules associated to $\mathcal{P}$

1. $p^{:\downarrow\emptyset} = q^{:\downarrow\emptyset}$
   For each equality $(p = q)$ in $\mathcal{P}$

$E_\mathcal{P}$: decorated equalities associated to $\mathcal{P}$

Figure 6.1: Extraction of decorated/decoration rewrite rules from the Presentation

As a consequence, the symmetric relation $\leftrightarrow$ is also stable by context and substitution. A last property allows us to work with representatives of $\cong_d$-equivalence classes when proving rewriting proof properties in the sequel.

Proposition 6.2.10 *Let $t, t', \bar{t}, \bar{t'} \in \mathcal{T}_d(\mathcal{S}_\diamond, \mathcal{F}, \mathcal{X}_\diamond)$ s.t. $t \cong_d t'$. Then $t \rightarrowtail^{\omega,\sigma,\phi}_{D\cup R} \bar{t}$ and $t' \rightarrowtail^{\omega,\sigma',\phi}_{D\cup R} \bar{t'}$ implies $\bar{t} \cong_d \bar{t'}$.*

Proof: Let $t_{|_\omega} =_d u^{:U}$ and $\bar{t}_{|_\omega} =_d u'^{:U'}$ and $\phi \in R$ be $l^{:S_l} \rightarrow r^{:S_r}$ (resp. $\phi \in D$ and $l^{:s} \rightarrow l^{:s\cup S_l}$ if $S_l \not\subseteq s$). Clearly, $t[\sigma(l^{:S_l})] \cong_d t \cong_d t' \cong_d t'[\sigma'(l^{:S_l})]$ (resp. $t[\sigma(l^{:U})] \cong_d t \cong_d t' \cong_d t'[\sigma'(l^{:U})]$). Therefore $\sigma \cong_d^{Var(l^{:S_l})} \sigma'$ and consequently $t[\sigma(r^{:S_r})] \cong_d t'[\sigma'(r^{:S_r})]$ (resp. $t[\sigma(l^{:U})^{:U\cup S_l}] \cong_d t'[\sigma'(l^{:U})^{:U\cup S_l}]$). □

3 Converting Presentations to Decorated TRSs

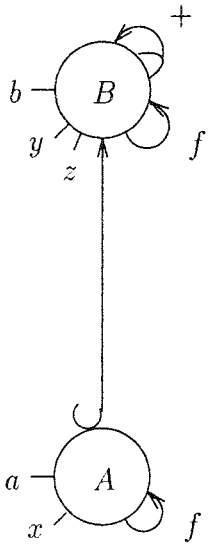
Given a presentation $\mathcal{P}$, we describe in Figure 6.1 how to extract a set of decorated equalities $E_\mathcal{P}$ and a set of decoration rules $D_\mathcal{P}$. Remark that all variables in the constructed rules are supposed to have their sort in their decoration – otherwise any rule ϕ containing a variable in $t = lhs(\phi)$ would not even be applicable on t itself, since the needed substitution would not fulfill the conditions for decorated substitutions.

The first kind of rules allows using existential declarations, the second term declarations. It is superfluous to introduce decoration rules corresponding to variable definitions $(x :: A)$ or declarations $(x : B) \in \mathcal{P}$, thanks to the matching modulo sort inheritance. The sort Ω and the first kind of rules are useful at a theoretical level to deduce existence formulas. In the sequel, we most often omit Ω on the decorations in order to ease notations.

The application of the decoration rules corresponds with a partial typing process, adding all syntactic sorts of a term to its decorations.

Example 6.3.1 *Let us consider the following specification $\mathcal{S}_1$ which set of sorts is $\{A, B\}$ the set of operators is $\{f, +, a, b\}$ and satisfying the following formulas:*

$$
\begin{array}{rcll}
x & :: & A & (6.1)\\
y & :: & B & (6.2)\\
z & :: & B & (6.3)\\
x & : & B & (6.4)\\
x+b & : & A & (6.5)\\
 & & & \\
f(x) & : & A & (6.6)\\
f(y) & : & B & (6.7)\\
y+z & : & B & (6.8)\\
a & : & A & (6.9)\\
b & : & B & (6.10)\\
 & & & \\
f(y) & = & a & (6.11)\\
a+y & = & y. & (6.12)
\end{array}
$$



Declarations 6.1 and 6.4 correspond to specifying the subsort declaration $A \leq_{\mathcal{S}}^{syn} B$. *From this specification, the following set of decorated rewrite rules* D_1 *is generated:*

$$
\begin{array}{lcll}
a^{:s} & \rightarrowtail & a^{:s\cup\{A\}} & \text{if } A \notin s\\
b^{:s} & \rightarrowtail & b^{:s\cup\{B\}} & \text{if } B \notin s\\
f(x^{:\{A\}})^{:s} & \rightarrowtail & f(x^{:\{A\}})^{:s\cup\{A\}} & \text{if } \{A\} \not\subseteq_{\mathcal{S}} s\\
f(y^{:\{B\}})^{:s} & \rightarrowtail & f(y^{:\{B\}})^{:s\cup\{B\}} & \text{if } \{B\} \not\subseteq_{\mathcal{S}} s\\
(x^{:\{A\}}+b^{:\emptyset})^{:s} & \rightarrowtail & (x^{:\{A\}}+b^{:\emptyset})^{:s\cup\{A\}} & \text{if } \{A\} \not\subseteq_{\mathcal{S}} s\\
(y^{:\{B\}}+z^{:\{B\}})^{:s} & \rightarrowtail & (y^{:\{B\}}+z^{:\{B\}})^{:s\cup\{B\}} & \text{if } B \notin s.
\end{array}
$$

We have omitted the decoration rules involving the sort Ω *that might irritate by their number and are not really useful in order to see the principles of decoration rule application, since they are treated in the same way. The term* $a^{:\emptyset}+b^{:\emptyset}$ *can be decorated, using* D_1 *in the following way:*

$$(a^{:\emptyset}+b^{:\emptyset})^{:\emptyset} \rightarrowtail_{D_1} (a^{:\{A\}}+b^{:\emptyset})^{:\emptyset} \rightarrowtail_{D_1} (a^{:\{A\}}+b^{:\emptyset})^{:\{A\}} \rightarrowtail_{D_1} (a^{:\{A\}}+b^{:\{B\}})^{:\{A\}}$$

Notice that 6.11 and 6.12 are not yet used nor yet transformed at this point. In $E_{\mathcal{P}}$, *they are represented by:*

$$
\begin{array}{rcl}
f(y^{:\{B\}})^{:\emptyset} & = & a^{:\{A\}}\\
(a^{:\{A\}}+y^{:\{B\}})^{:\emptyset} & = & y^{:\{B\}}
\end{array}
$$

This rewrite based decoration process is a restricted application of the deduction rules of G-algebra concerned only by membership formula deduction. The decoration process does not use the equational part of the specification for deriving new sorts from equalities. It is thus in general incomplete in the sense that it does not compute all the possible sorts of a term, even when this is computable.

4 Decorated Orderings

Finally, in order to find an extension of the classical reduction ordering over undecorated terms, we compare the decorations of $=_{nd}$-equal terms:

Definition 6.4.1 *Let $t^{:S}$ and $t'^{:S'}$ be decorated terms, such that $t^{:S} =_{nd} t'^{:S'}$. $t^{:S}$ is* less decorated *than $t'^{:S'}$, written $t^{:S} \Subset t'^{:S'}$ if $\forall \omega \in Occ(t), Deco(t_{|\omega}) \subsetneq Deco(t'_{|\omega})$ and $\exists \omega \in Occ(t), Deco(t_{|\omega}) \not\approx Deco(t'_{|\omega})$. The relation $\Subset$ is called decoration subsumption ordering.*

The classical notion of reduction ordering can now be extended on decorated terms in a straightforward way. Such an ordering is useful for termination proofs.

Definition 6.4.2 *Let $((\mathcal{S}, \mathcal{F}), \mathcal{P})$ be a presentation with bounded membership. A quasi-ordering $>_d$ over $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ is a* decorated reduction ordering *w.r.t. to a rule set R, if there is an (undecorated) reduction ordering $>$ on $\mathcal{T}(\Sigma, \mathcal{X})$, s.t.:*

1. *$t >_d t'$ iff $(t_{nd}, t)(>, \Subset)(t'_{nd}, t')$,*
   i.e. $>_d$ is the lexicographic ordering based on the undecorated ordering and the decoration subsumption over pairs (t_{nd}, t),
2. *$>_d$ is compatible with R, i.e. $\forall (l \to r) \in R$, $l >_d r$.*

A last notion, necessary for later induction proofs, is the downward closure of a set of decorated terms w.r.t. a given ordering.

Definition 6.4.3 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F})$ and $>$ be a quasi-ordering over $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F})$. $\mathcal{T}$ is* downward closed *w.r.t. $>$ iff for all $t^{:S}, t'^{:S'}$ the following holds:*

$$t^{:S} \in \mathcal{T} \text{ and } t^{:S} > t'^{:S'} \Rightarrow t'^{:S'} \in \mathcal{T}.$$

5 A Birkhoff Theorem for G-Algebra

In order to get an executable version of deduction in G-algebra, we need to prove a Birkhoff-like theorem stating that two terms can be proved to be equivalent using the deduction rules of G-algebra iff they can be proved equivalent by replacement of equal by equal on decorated terms.

Given a set E of decorated equalities and a set D of decoration rewrite rules, extracted for instance from a presentation $\mathcal{P}$ as in the previous section, $\longleftrightarrow\!\!\!\!\!\!\rightarrow_{D \cup E}$ is $\longleftrightarrow\!\!\!\!\!\!\rightarrow_{E} \cup \longleftrightarrow\!\!\!\!\!\!\rightarrow_{D}$ and $\overset{*}{\longleftrightarrow\!\!\!\!\!\!\rightarrow}_{D \cup E}$ its reflexive, symmetric and transitive closure.

Theorem 6.5.1 *Let $\mathcal{P}$ be a Σ-presentation. Let furthermore $D_\mathcal{P}$ and $E_\mathcal{P}$ be sets of decoration rewrite rules and equational axioms, respectively, associated to $\mathcal{P}$ and $R = \emptyset$. Then $Th(\mathcal{P}) = Form(D_\mathcal{P}, E_\mathcal{P}, R)$.*

The proof is rather technical and long. It relies on the step by step description of the correspondence between deductions and equational proofs on decorated terms. The first step consists in proving that the use of $\longleftrightarrow\!\!\!\!\!\!\rightarrow_{D \cup E}$ is sound with respect to G-deduction. This is an obvious consequence of Lemma 6.2.8, since D and E are valid by construction.

Corollary 6.5.2 *Let $\mathcal{P}$ be a Σ-presentation. Then:*

If $\exists t', A'$ with $A' \leq^{syn}_{\mathcal{S}_\Diamond} \langle A \rangle$, s.t. $t^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow\!\!\!\!\!\!\rightarrow}_{D_\mathcal{P} \cup E_\mathcal{P}} t'^{:\{A'\} \cup V}$	*then $\mathcal{P} \vdash_G t : A$.*
If $\exists t', A$ s.t. $t^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow\!\!\!\!\!\!\rightarrow}_{D_\mathcal{P} \cup E_\mathcal{P}} t'^{:\{A\} \cup V}$	*then $\mathcal{P} \vdash_G EX\ t$.*
If $\exists t_0, A$ s.t. $t^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow\!\!\!\!\!\!\rightarrow}_{D_\mathcal{P} \cup E_\mathcal{P}} t_0^{:\{A\} \cup U} \overset{*}{\longleftrightarrow\!\!\!\!\!\!\rightarrow}_{D \cup E} t'^{:\downarrow\emptyset}$	*then $\mathcal{P} \vdash_G t = t'$.*

Note that imposing that the set of decorations becomes non-empty at the top is essential, as shown by the following example. For the presentation $\mathcal{P}$:

$$a : A,\ b : B,\ f : C \to C,\ a = b$$

using $\longleftrightarrow_{D_\mathcal{P} \cup E_\mathcal{P}}$, we get

$$f(a^{:\emptyset})^{:\emptyset} \rightarrowtail_{D_\mathcal{P}} f(a^{:\{A\}})^{:\emptyset} \longleftrightarrow_{E_\mathcal{P}} f(b^{:\{B\}})^{:\emptyset}$$

but $\mathcal{P} \vdash_G f(a) = f(b)$ is not a valid deduction, since there is no way to deduce that $f(a)$ exists in this G-algebra.

The second step of the proof consists of showing that every G-deduction can be mapped to some replacement of equal by equal on decorated terms, using decorated substitutions. We use an induction on G-deduction trees of the considered formulas (i.e. either $t = t'$, $t : A$ or $EX\ t$). Two deduction trees are compared using the lexicographic combination of the number of branching as first component, and the length of the derivation as second component. (In other words, the induction works also on the number of nodes in the deduction tree). Note that all terms in the constructed proofs are valid and the used substitutions are the same as in the proofs given by the induction hypothesis, i.e. they fulfill the conditions of decorated substitutions. In the following, we will simply write E instead of $E_\mathcal{P}$ and D instead of $D_\mathcal{P}$, in order to be more concise.

Since we have three types of formulas, we distinguish:

CASE A: $(\mathcal{P} \vdash_G t = t') \Rightarrow \exists t_0, A\ s.t.\ t^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t_0^{:\{A\}\cup U} \longleftrightarrow^*_{D\cup E} t'^{:\downarrow\emptyset}$.
CASE B: $(\mathcal{P} \vdash_G t : A) \Rightarrow \exists t', A'\ with\ A' \leq^{syn}_{\mathcal{S}_\Diamond} \langle A\rangle\ s.t.\ t^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t'^{:\{A'\}\cup V}$.
CASE C: $(\mathcal{P} \vdash_G EX\ t) \Rightarrow \exists t'\ s.t.\ t^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t'^{:\{A\}\cup V}$.

Remark that all terms $t^{:S}$ in the constructed proof satisfy $((t^{:S})^{:\downarrow\emptyset})_{nd} \in \mathcal{T}(\Sigma, \mathcal{X})$ (written (H_1)). $(t^{:S})^{:\downarrow\emptyset} \twoheadrightarrow_D t^{:S}$ (written (H_2)) and for all subterms $u^{:U}$ of $t^{:S}$, there is a term $u'^{:U'}$, s.t. $(u^{:U})^{:\downarrow\emptyset} \twoheadrightarrow_D u'^{:U'}$ and $U' \neq \emptyset$ (written (H_3)). All three are very useful properties.

CASE A: We reason by case on the possible production of $t = t'$.

1. **Base case**
   If $t = t'$ is an equality in $\mathcal{P}$, then $t^{:\downarrow\emptyset} \rightarrow_D t^{:\{\Omega\}}$ by the rule $(t^{:s} \to t^{:s\cup\{\Omega\}}$ if $\{\Omega\} \not\subseteq s)$. Therefore, $t^{:\downarrow\emptyset} \rightarrow_D t^{:\downarrow\emptyset:\{\Omega\}} \leftarrow_D t^{:\downarrow\emptyset} \longleftrightarrow_E t'^{:\downarrow\emptyset}$. Note that all substitutions needed for the application of the rules always satisfy the conditions of decorated substitutions since variables are supposed to have their sort automatically in their decoration. Properties (H_1) and (H_2) are obviously satisfied and property (H_3) follows by the decoration rule $(u^{:s} \to u^{:s\cup\{\Omega\}}$ if $\{\Omega\} \not\subseteq s)$, which is by definition in $D_\mathcal{P}$.

   Otherwise $t = t'$ may be obtained from the application of one of the rules **Reflexivity**, **Symmetry**, **Transitivity**, **EqSubstitutivity** or **EqReplacement**. Let us consider each case.

2. **Reflexivity** $EX\ t \vdash_G t = t$.
   Since the deduction tree for $EX\ t$ is smaller, we can apply the induction hypothesis and thus:
   $$\mathcal{P} \vdash_G EX\ t \Rightarrow \exists t'\ s.t.\ t^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t'^{:\{A\}\cup V}$$

exists and satisfies $(H_i)_{i=1,2}$, which proves that:

$$t^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D\cup E} t'^{:\{A\}\cup V} \overset{*}{\longleftrightarrow}_{D\cup E} t^{:\downarrow\emptyset}.$$

The properties $(H_i)_{i\in[1..3]}$ also hold for the constructed proof by the induction hypothesis.

3. **Symmetry** $u = v \quad \vdash_G \quad v = u$.
Since the deduction tree for $u = v$ is smaller, we can apply the induction hypothesis and thus get the result which is symmetric. The properties $(H_i)_{i\in[1..3]}$ hold again for the constructed proof, since they are subsumed by the induction hypothesis.

4. **Transitivity** $u = v, v = w \quad \vdash_G \quad u = w$.
Since the deduction trees for $u = v$ and $v = w$ are smaller, we can apply twice the induction hypothesis and concatenate the equational proofs. The properties $(H_i)_{i\in[1..3]}$ hold once more for the constructed proof, since they are subsumed by the induction hypothesis.

5. **EqSubstitutivity** $r = r' \quad \vdash_G \quad \sigma(r) = \sigma(r')$,
with $t \equiv \sigma(r)$ and $t' \equiv \sigma(r')$, where $\equiv$ denotes syntactic identity on terms in $\mathcal{T}(\Sigma, \mathcal{X})$.
In this case, by induction hypothesis:

$$\mathcal{P} \vdash_G r = r' \Rightarrow \exists r_0, A \ s.t. \ r^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D\cup E} r_0{}^{:\{A\}\cup U} \overset{*}{\longleftrightarrow}_{D\cup E} r'^{:\downarrow\emptyset}.$$

Note that to any $\mathcal{P}$-conform substitution in G-algebra that associates to $(x_i :: A_i)$ a term t_i for which there exists a proof $(t_i : A_i)$, we have $(t_i : A_i)$ as an implicit subproof for the preconditions of **EqSubstitutivity**. We apply the induction hypothesis also on these membership proofs and have therefore:

$$\mathcal{P} \vdash_G t_i : A_i \Rightarrow \exists t'_i, A'_i \leq^{syn}_{\mathcal{S}_\diamond} \langle A_i \rangle \ s.t. \ t_i{}^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D\cup E} t'_i{}^{:\{A'_i\}\cup U_i}$$

Remark that all substitutions in these proofs must be decorated substitutions, i.e. all terms in their ranges are valid. Consequently,

$$\exists A_i, i \in [1..n], s.t. \ \sigma(r)^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D\cup E} r^{:\downarrow\emptyset}[t'_i{}^{:\{A'_i\}\cup U_i}] \overset{*}{\longleftrightarrow}_{D\cup E} r'^{:\downarrow\emptyset}[t'_i{}^{:\{A'_i\}\cup U_i}] \overset{*}{\longleftrightarrow}_{D\cup E} \sigma(r')^{:\downarrow\emptyset}.$$

The properties $(H_i)_{i\in[1..3]}$ follow from the induction hypothesis for $(r = r')$ and all $(t_i : A_i)$ together with stability of decoration rewriting by substitutivity or context, respectively.

6. **EqReplacement** $t[u] = w, u = v \quad \vdash_G \quad t[v] = w$.
Both formulas $t[u] = w$ and $u = v$ are smaller than $t[v] = w$ and by applying the induction hypothesis we get:

$$\exists t_0, A \ s.t. \ t[u]^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D\cup E} t_0{}^{:\{A\}\cup U} \overset{*}{\longleftrightarrow}_{D\cup E} w^{:\downarrow\emptyset}$$
$$\text{and} \quad \exists t_1, B \ s.t. \ u^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D\cup E} t_1{}^{:\{B\}\cup U'} \overset{*}{\longleftrightarrow}_{D\cup E} v^{:\downarrow\emptyset}.$$

But since $\overset{*}{\longleftrightarrow}_{D\cup E}$ is stable under context, we can write:

$$\begin{aligned} t^{:\downarrow\emptyset}[v^{:\downarrow\emptyset}] &\overset{*}{\longleftrightarrow}_{D\cup E} t^{:\downarrow\emptyset}[t_1{}^{:\{B\}\cup U'}] \\ &\overset{*}{\longleftrightarrow}_{D\cup E} t^{:\downarrow\emptyset}[u^{:\downarrow\emptyset}] =_d t[u]^{:\downarrow\emptyset} \\ &\overset{*}{\longleftrightarrow}_{D\cup E} t_0{}^{:\{A\}\cup U} \\ &\overset{*}{\longleftrightarrow}_{D\cup E} w^{:\downarrow\emptyset}. \end{aligned}$$

The properties $(H_i)_{i\in[1..3]}$ are once more guaranteed by the induction hypothesis on $(t[u] = w)$ and $(u = v)$ together with stability of decoration rewriting by context.

CASE B: The proof goes on the same principle as for CASE A.

1. **Base case**
   If $t : A$ is a formula in $\mathcal{P}$, then $t^{:\downarrow\emptyset} \rightarrowtail_D t^{:\{A\}}$ and $u^{:\downarrow\emptyset} \rightarrowtail_D u^{:\{\Omega\}}$ for all subterms $u^{:\downarrow\emptyset}$ of $t^{:\downarrow\emptyset}$. Clearly, this guarantees properties $(H_i)_{i=1,2}$. Otherwise $t : A$ may be obtained from the application of one of the rules **Globality**, **VariableMembership**, **MeReplacement** or **MeSubstitutivity**.

2. **Globality** $EX\, t \quad \vdash_G \quad t : \Omega$.
   By induction we get $t^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t'^{:\{A\}\cup V}$. Then $A \leq^{syn}_{\mathcal{S}_\Diamond} \langle\Omega\rangle$ and we are done. This trivially provides for properties $(H_i)_{i\in[1..3]}$.

3. **VariableMembership** $x :: A \quad \vdash_G \quad x : A$.
   True because each variable is decorated by the sort given by its declaration. $(H_i)_{i\in[1..3]}$ are trivial.

4. **MeReplacement** $t[u] : A, u = v \quad \vdash_G \quad t[v] : A$.
   In this case, we apply the induction hypothesis on the first premise and the result of CASE A on the second one. Thus we have:

$$\begin{aligned} &\exists t', A' \leq^{syn}_{\mathcal{S}_\Diamond} A \text{ s.t. } t[u]^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t'^{:\{A'\}\cup U} \\ \textit{and}\quad &\exists t_1, B \text{ s.t. } u^{:\downarrow\emptyset} \longleftrightarrow^*_{D\cup E} t_1{}^{:\{B\}\cup U} \longleftrightarrow^*_{D\cup E} v^{:\downarrow\emptyset}. \end{aligned}$$

   But since $\longleftrightarrow^*_{D\cup E}$ is stable under context, we can write:

$$\begin{aligned} t^{:\downarrow\emptyset}[v^{:\downarrow\emptyset}] &\longleftrightarrow^*_{D\cup E} t^{:\downarrow\emptyset}[t_1{}^{:\{B\}\cup U}] \\ &\longleftrightarrow^*_{D\cup E} t^{:\downarrow\emptyset}[u^{:\downarrow\emptyset}] =_d t[u]^{:\downarrow\emptyset} \\ &\longleftrightarrow^*_{D\cup E} t'^{:\{A'\}\cup U}, \end{aligned}$$

   which concludes this case. For $(H_i)_{i\in[1..3]}$ see CASE A, subcase **EqReplacement**.

5. **MeSubstitutivity** $t : A \quad \vdash_G \quad \sigma(t) : A$.
   Applying the induction hypothesis to $t : A$ and all the $t_i : A_i$ necessary for the proof of $\mathcal{P}$ - *conformity* of σ, we get the conclusion, using the same construction as in the **EqSubstitutivity** - subcase of CASE A.

CASE C: This is the most complex case where we need to reason on more specific sequences in the deduction tree. Let us begin with the easy cases:

1. **Base case**
   If $EX\ t$ is a formula in $\mathcal{P}$, then $t^{:\downarrow\emptyset} \rightarrow_D t^{:\{\Omega\}}$. Clearly, these proofs have the properties $(H_i)_{i\in[1..3]}$. Otherwise $EX\ t$ may be obtained from the application of one **ExMembership**, **ExEquality**, **ExReplacement**, **ExSubstitutivity** or **ExSubterm**.

2. **ExMembership** $t : A \quad \vdash_G \quad EX\ t$.
   Applying the result of CASE B on $t : A$, we get the result.

3. **ExEquality** $t = t' \quad \vdash_G \quad EX\ t$.
   Applying the result of CASE A on $t = t'$, we extract the result.

4. **ExReplacement** $EX\ t[u], u = v \quad \vdash_G \quad EX\ t[v]$.
Both formulas $EX\ t[u]$ and $u = v$ have a complexity smaller than $EX\ t[v]$ and by applying the induction hypothesis we get:

$$\begin{array}{rl} & \exists t', A \text{ s.t. } t[u]^{:\downarrow\emptyset} \longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E} t'^{:\{A\}\cup U} \\ \textit{and} & \exists t_1, B \text{ s.t. } u^{:\downarrow\emptyset} \longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E} t_1{}^{:\{B\}\cup U} \longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E} v^{:\downarrow\emptyset}. \end{array}$$

But since $\longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E}$ is stable under context, we can write:

$$\begin{array}{ll} t^{:\downarrow\emptyset}[v^{:\downarrow\emptyset}] & \longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E} t^{:\downarrow\emptyset}[t_1{}^{:\{B\}\cup U}] \\ & \longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E} t^{:\downarrow\emptyset}[u^{:\downarrow\emptyset}] =_d t[u]^{:\downarrow\emptyset} \\ & \longleftrightarrow\!\!\!\!\!\rightarrow^{*}_{D\cup E} t'^{:\{A\}\cup U}, \end{array}$$

which concludes this case. For $(H_i)_{i\in[1..3]}$, see again CASE A, subcase **EqReplacement.**

5. **ExSubstitutivity** $EX\ t \quad \vdash_G \quad EX\ \sigma(t)$.
Applying the induction hypothesis to $EX\ t$ and all the $t_i : A_i$ necessary for the proof of $\mathcal{P}$ - *conformity* of σ, we get the conclusion, using the same construction as in the **EqSubstitutivity** - subcase of CASE A.

6. **ExSubterm** $EX\ t[u] \quad \vdash_G \quad EX\ u$.
This case follows from the induction hypothesis, property (H_3).

This concludes the proof of the theorem. We illustrate this result with an example:

Example 6.5.3 *Consider the specification in Example 6.3.1. We can prove there* $f(y) : A, a + f(y) = a$ *and* $EX\ a + a$. *The corresponding proves in* $D_{\mathcal{P}} \cup E_{\mathcal{P}}$ *are:*

$$\begin{array}{lll} f(y) : A \qquad f(y^{:\{B\}})^{:\emptyset} & \longleftrightarrow\!\!\!\!\!\rightarrow_{E_{\mathcal{P}}} & a^{:\emptyset} \\ & \rightarrowtail_{D_{\mathcal{P}}} & a^{:\{A\}} \\ \\ a + f(y) = a \ (a^{:\emptyset} + f(y^{:\{B\}})^{:\emptyset})^{:\emptyset} & \longleftrightarrow\!\!\!\!\!\rightarrow_{E_{\mathcal{P}}} & (a^{:\emptyset} + a^{:\emptyset})^{:\emptyset} \\ & \rightarrowtail_{D_{\mathcal{P}}} & (a^{:\emptyset} + a^{:\{A\}})^{:\emptyset} \\ & \longleftrightarrow\!\!\!\!\!\rightarrow_{E_{\mathcal{P}}} & a^{:\emptyset} \\ & \rightarrowtail_{D_{\mathcal{P}}} & a^{:\{A\}} \\ \\ EX\ a + a \qquad (a^{:\emptyset} + a^{:\emptyset})^{:\emptyset} & \rightarrowtail_{D_{\mathcal{P}}} & (a^{:\emptyset} + a^{:\{A\}})^{:\emptyset} \\ & \rightarrowtail_{D_{\mathcal{P}}} & (a^{:\{A\}} + a^{:\{A\}})^{:\emptyset} \\ & \rightarrowtail_{D_{\mathcal{P}}} & (a^{:\{A\}} + a^{:\{A\}})^{:\{A\}} \end{array}$$

Chapitre 7

Decorated Completion in Sort Inheriting Presentations

Our purpose is now to design a completion process that provides, whenever it does not fail, from an initial sort inheriting presentation $\mathcal{P}_0$, a saturated presentation $\mathcal{P}_\infty$ such that $Th(\mathcal{P}) = Form(\mathcal{P}_\infty)$ and any formula $\phi \in Form(\mathcal{P}_\infty)$ has a rewrite proof. Our notations are consistent with [Bachmair, 1991].

To $\mathcal{P}_0$ is associated the initial triple (D_0, E_0, R_0) where E_0 is usually the set of decorated equalities $E_{\mathcal{P}_0}$, D the set of decoration rules $D_{\mathcal{P}_0}$ associated to $\mathcal{P}_0$ as defined in Section 6.3 and R_0 is empty. The completion process is defined as a transformation rule system $\mathcal{OSC}$ that transforms decorated presentations: $\mathcal{P}_0 \vdash_{\mathcal{OSC}} \ldots \vdash_{\mathcal{OSC}} \mathcal{P}_k \ldots$. The resulting decorated presentation is given by the limit $\mathcal{P}_\infty = (D_\infty, E_\infty, R_\infty)$, where $E_\infty = \emptyset$.

Notation: We write $t^{:S} \rightarrowtail_{\mathcal{P}_\infty} t'^{:S'}$ instead of $t^{:S} \ (\rightarrowtail_{R_\infty \cup D_\infty}) \ t'^{:S'}$.

$\mathcal{P}_\infty$ satisfies the following properties, if the completion does not fail:

1. The Church-Rosser property of $\mathcal{P}_\infty$: any equational theorem $(t = t')$ has a rewrite proof using $\mathcal{P}_\infty$, i.e. there exists a term t'' decorated by at least one sort A, such that:

$$t^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_{\mathcal{P}_\infty} t''^{:\{A\}\cup S''} \overset{*}{\leftarrowtail}_{\mathcal{P}_\infty} t'^{:\downarrow\emptyset},$$

   where S'' is a decoration.

2. The type completeness property of $\mathcal{P}_\infty$: any membership theorem $(t : A)$, where $A \in \mathcal{S}$, has a rewrite proof using $\mathcal{P}_\infty$, i.e. there exists t' and $A' \in \mathcal{S}_\Diamond$ with $A' \leq^{syn}_{\mathcal{S}_\Diamond} \langle A \rangle$, such that:

$$t^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_{\mathcal{P}_\infty} t'^{:\{A'\}\cup S'},$$

   where S' is a decoration.

3. The existential completeness property of $\mathcal{P}_\infty$: any existential theorem $(EX\ t)$ has a rewrite proof using $\mathcal{P}_\infty$, i.e. there exist t' and $A \in \mathcal{S}_\Diamond$ such that:

$$t^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_{\mathcal{P}_\infty} t'^{:\{A\}\cup S'},$$

   where S' is an arbitrary decoration.

Since $E_\infty = \emptyset$, $\overset{*}{\rightarrowtail}_{\mathcal{P}_\infty}$ is actually rewriting with decoration rules in D_∞ and decorated rules in R_∞.

The dual aspect of completion is the proof reduction process. Let us make this notion more precise. Using the completeness Theorem 6.5.1, proofs of interest in a decorated presentation $\mathcal{P}$ given by (D, E, R) are of the form $t^{:\downarrow\emptyset} \overset{*}{\leftarrowtail\!\!\rightarrowtail}_{D\cup E} t_0^{:\{A\}\cup U} \overset{*}{\leftarrowtail\!\!\rightarrowtail}_{D\cup E} t'^{:\downarrow\emptyset}$. The set of completion rules is mirrored by a set of proof reduction rules which normalises such a proof into a rewrite proof $t^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_{\mathcal{P}_\infty} t''^{:\{A\}\cup S''} \overset{*}{\leftarrowtail}_{\mathcal{P}_\infty} t'^{:\downarrow\emptyset}$.

1 Confluence and Critical Pairs

We now consider the case where equalities can be ordered into rewrite rules. Our concern is to check under which conditions on the presentation, we can obtain a set of decoration and decorated rewrite rules that operationally are powerful enough to prove equational, existential and membership theorems. This is the case if any equational proof using $D \cup R$ has a rewrite proof, i.e. a proof without peaks.

The first goal of this section is to characterise confluence on a set $\mathcal{T}$ of valid terms. Defining notions on non-valid terms is obviously non-sense, since this would allow for incorrect interpretations using Theorem 6.5.1.

This can be exemplified by the following example:

Example 7.1.1 *Let $\mathcal{P}_{DER}$ be the (sort inheriting) presentation associated with $(D, \emptyset, R)$, where D and R are defined as follows:*

$$\begin{aligned} D &= \{a^{:s} \to a^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s, \\ &\quad\ b^{:s} \to b^{:s\cup\{B\}} \text{ if } \{B\} \not\subseteq s, \\ &\quad\ f((x :: A)^{:\{A\}})^{:s} \to f(x^{:\{A\}})^{:s\cup\{C\}} \text{ if } \{C\} \not\subseteq s, \\ &\quad\ f(y^{:\{B\}})^{:s} \to f(y^{:\{B\}})^{:s\cup\{C\}} \text{ if } \{C\} \not\subseteq s\} \\ R &= \{f(x^{:\{A\}})^{:\{A\}} \to x^{:\{A\}}, \\ &\quad\ f(y^{:\{B\}})^{:\{B\}} \to b^{:\{B\}}\} \end{aligned}$$

Clearly, $D \cup R$ is confluent on all valid terms, but the non-valid term $f(a^{:\{A,B\}})^{:\{C\}}$ rewrites to $a^{:\{A,B\}}$ respectively $b^{:\{B\}}$, which are both irreducible.

1.1 $\mathcal{T}$-Confluence

Well-known notions are redefined in a partial manner, i.e. with respect to a term set $\mathcal{T}$, as this was already done for unification.

Definition 7.1.2 *Let R be a set of decorated rewrite rules, D a set of decoration rules and $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ a set of decorated terms.*

$D \cup R$ is called locally confluent on $\mathcal{T}$ *iff for all $t^{:T}, t_1^{:T_1}, t_2^{:T_2} \in \mathcal{T}$:*

$$t_1^{:T_1} \leftarrowtail_{D\cup R} t^{:T} \rightarrowtail_{D\cup R} t_2^{:T_2} \text{ implies } \exists t_0^{:T_0} \in \mathcal{T} \ :\ t_1^{:T_1} \overset{*}{\rightarrowtail}_{D\cup R} t_0^{:T_0} \overset{*}{\leftarrowtail}_{D\cup R} t_2^{:T_2}.$$

$D \cup R$ is called confluent on $\mathcal{T}$ *iff for all $t^{:T}, t_1^{:T_1}, t_2^{:T_2} \in \mathcal{T}$:*

$$t_1^{:T_1} \overset{*}{\leftarrowtail}_{D\cup R} t^{:T} \overset{*}{\rightarrowtail}_{D\cup R} t_2^{:T_2} \text{ implies } \exists t_0^{:T_0} \in \mathcal{T} \ :\ t_1^{:T_1} \overset{*}{\rightarrowtail}_{D\cup R} t_0^{:T_0} \overset{*}{\leftarrowtail}_{D\cup R} t_2^{:T_2}.$$

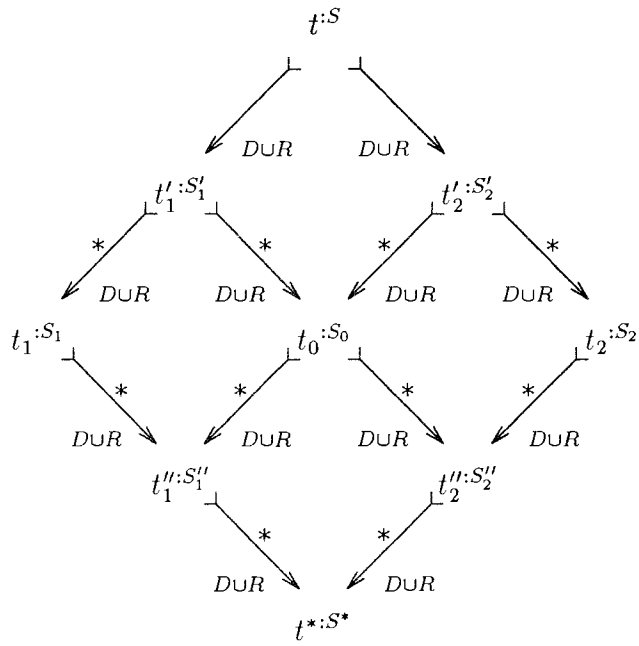


Figure 7.1: The Newman Lemma for Decorated Rewriting

Newman's Lemma still has its equivalent in the decorated case, when $\mathcal{T}$ is downward closed w.r.t. a reduction ordering, namely $\rightarrowtail_{D \cup R}$.

Lemma 7.1.3 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ be a downward closed set of decorated terms w.r.t. $\rightarrowtail_{D \cup R}$. If $\rightarrowtail_{D \cup R}$ is terminating on $\mathcal{T}$ then the local confluence of $\rightarrowtail_{D \cup R}$ on $\mathcal{T}$ implies that $D \cup R$ is confluent on $\mathcal{T}$.*

Proof: The proof follows the standard one (e.g. [Huet, 1980]). Assume $t^{:S} \in \mathcal{T}$ and

$$t_1^{:S_1} \stackrel{*}{\leftarrowtail}_{D \cup R} t^{:S} \stackrel{*}{\rightarrowtail}_{D \cup R} t_2^{:S_2}.$$

Let us show confluence by noetherian induction on $\leftarrowtail_{D \cup R}$. If $t_1^{:S_1} =_d t^{:S}$ or $t_2^{:S_2} =_d t^{:S}$, then confluence is trivial. Else we can isolate the first rewriting steps in each direction:

$$t_1^{:S_1} \stackrel{*}{\leftarrowtail}_{D \cup R} t_1'^{:S_1'} \leftarrowtail_{D \cup R} t^{:S} \rightarrowtail_{D \cup R} t_2'^{:S_2'} \stackrel{*}{\rightarrowtail}_{D \cup R} t_2^{:S_2}.$$

The following argumentation is illustrated in Figure 7.1. Since we have the local confluence on $\mathcal{T}$ and $t^{:S} \in \mathcal{T}$, we know that $t_1'^{:S_1'} \stackrel{*}{\rightarrowtail}_{D \cup R} t_0^{:S_0} \stackrel{*}{\leftarrowtail}_{D \cup R} t_2'^{:S_2'}$ for some $t_0^{:S_0}$ since $\mathcal{T}$ is downward closed. Together with the confluence on smaller terms $(t_1'^{:S_1'}, t_2'^{:S_2'})$ by induction hypothesis we get $t_1^{:S_1} \stackrel{*}{\rightarrowtail}_{D \cup R} t_1''^{:S_1''} \stackrel{*}{\leftarrowtail}_{D \cup R} t_0^{:S_0}$ and $t_0^{:S_0} \stackrel{*}{\rightarrowtail}_{D \cup R} t_2''^{:S_2''} \stackrel{*}{\leftarrowtail}_{D \cup R} t_2^{:S_2}$. Using once more the confluence on $t_0^{:S_0}$ gives us $t_1''^{:S_1''} \stackrel{*}{\rightarrowtail}_{D \cup R} t^{*:S^*} \stackrel{*}{\leftarrowtail}_{D \cup R} t_2''^{:S_2''}$. Consequently,

$$t_1^{:S_1} \stackrel{*}{\rightarrowtail}_{D \cup R} t^{*:S^*} \stackrel{*}{\leftarrowtail}_{D \cup R} t_2^{:S_2},$$

i.e. $D \cup R$ is also confluent on $\mathcal{T}$. □

A simple example for a $\rightarrowtail_{D \cup R}$-downward closed set is $\mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, as we proved in Lemma 6.2.6.

1.2 Definition and Properties of Critical Pairs

The rewrite rule version of the typing process has been designed for expressing the critical interactions between membership declarations and equalities in a convenient way. Since the first are now encoded into the decoration rewrite system D, this amounts to define adequate notions of critical pairs. Although the superposition mechanism used to compute all critical pairs is uniform, we distinguish between decorated critical pairs, that are equalities obtained by superposition inside a decorated rewrite rule of R, and decoration critical pairs, that are conditional equalities obtained by superposition inside a decoration rule of D. This last kind of conditional critical pair will give rise to a new decoration rule.

The main difference with the classical definitions is once more partiality.

Definition 7.1.4 Decorated critical pairs (from superposition *into* rules of R)
Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $(\phi_R : g^{:S_g} \to d^{:S_d}) \in R$,
$(\phi'_R : l^{:S_l} \to r^{:S_r}) \in R$, and
$(\phi_D : l^{:s} \to l^{:s \cup S_l} \text{ if } S_l \not\subseteq s) \in D$
have disjoint sets of variables.

The two rules ϕ_R *and* ϕ'_R *overlap if there exists a position* $\omega \in \mathcal{NVO}cc(g^{:S_g})$, *such that the decorated terms* $g^{:S_g}{}_{|\omega}$ *and* $l^{:S_l}$ *have the* $\mathcal{T}$*-complete, sound, non-empty set* Ψ *of strict decorated unifiers. Then for any* $\psi \in \Psi$, *the overlapped decorated term* $\psi(g^{:S_g})$ *produces the* $\mathcal{T}$*-decorated critical pair* $(p^{:S_1} = q^{:S_2})$ *where* $p^{:S_1} =_d \psi(g^{:S_g}[r^{:S_r}]_\omega)$ *and* $q^{:S_2} =_d \psi(d^{:S_d})$.

Analogously, ϕ_D *overlaps into* ϕ_R *if there exists a position* $\omega \in \mathcal{NVO}cc(g^{:S_g})$, *such that the decorated terms* $g^{:S_g}{}_{|\omega}$, *with* $Deco(g^{:S_g}{}_{|\omega}) = U$ *and* $S_l \not\subseteq U$, *and* $l^{:U}$ *have the* $\mathcal{T}$*-complete, sound, non-empty set* Ψ *of strict decorated unifiers. Then for any* $\psi \in \Psi$, *the overlapped decorated term* $\psi(g^{:S_g})$ *produces the* $\mathcal{T}$*-decorated critical pair* $(p^{:S_1} = q^{:S_2})$ *where* $p^{:S_1} =_d \psi(g^{:S_g})[\psi(l^{:U})^{:U \cup S_l}]_\omega$ *and* $q^{:S_2} =_d \psi(d^{:S_d})$.

In the particular case where $\omega = \wedge$, we have $U = S_g$ and so the critical pairs are:

$p^{:S_1} =_d \psi(r^{:S_r})$ and $q^{:S_2} =_d \psi(d^{:S_d})$ for the overlap of ϕ_R into ϕ'_R and
$p^{:S_1} =_d \psi(l)^{:S_g \cup S_l}$ and $q^{:S_2} =_d \psi(d^{:S_d})$ for the overlap of ϕ_R into ϕ_D.

The set of such $\mathcal{T}$-decorated critical pairs is denoted by $CP(R, R)_{|\mathcal{T}}$ for those amongst rules of R and $CP(D, R)_{|\mathcal{T}}$ for those between rules of D and R.

Definition 7.1.5 Decoration critical pairs (from superposition *into* rules of D)
Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $(\phi_D : g^{:s} \to g^{:s \cup S_g} \text{ if } S_g \not\subseteq s) \in D$,
$(\phi'_D : l^{:s} \to l^{:s \cup S_l} \text{ if } S_l \not\subseteq s) \in D$ and
$(l^{:S_l} \to r^{:S_r}) \in R$
have disjoint sets of variables.

The two rules ϕ_R *and* ϕ_D *overlap if there exists a position* ω *in the set of non-variable positions of* $g^{:s}$, *such that the decorated terms* $g^{:\emptyset}{}_{|\omega}$ *and* $l^{:S_l}$ *have the* $\mathcal{T}$*-complete, sound, non-empty set* Ψ *of strict decorated unifiers. Then for any* $\psi \in \Psi$, *the overlapped decorated term* $\psi(g)^{:s}$ *produces the* $\mathcal{T}$*-decoration critical pair* $(p^{:s} = q^{:s \cup S} \text{ if } S_g \not\subseteq s)$, *where* $p^{:s} =_d \psi(g[r^{:S_r}]_\omega)^{:s}$ *and* $q^{:s \cup S} =_d \psi(g)^{:s \cup S_g}$.

The two rules ϕ_D *and* ϕ'_D *overlap if there exists a position* ω *in the set of non-variable positions of* $g^{:s}$, *such that the decorated terms* $g^{:\emptyset}{}_{|\omega}$, *with* $Deco(g^{:\emptyset}{}_{|\omega}) = U$ *and* $S_l \not\subseteq U$, *and* $l^{:U}$ *have the* $\mathcal{T}$*-complete, sound, non-empty set* Ψ *of strict decorated unifiers. Then for any* $\psi \in \Psi$, *the overlapped decorated term* $\psi(g)^{:s}$ *produces the* $\mathcal{T}$*-decoration critical pair* $(p^{:s} = q^{:s \cup S} \text{ if } S_g \not\subseteq s)$, *where* $p^{:s} =_d \psi(g[l^{:U \cup S_l}]_\omega)^{:s}$ *and* $q^{:s \cup S} =_d \psi(g)^{:s \cup S_g}$.

The sets of such $\mathcal{T}$-decoration critical pairs are denoted by $CP(R,D)_{|_\mathcal{T}}$ for overlaps between rules of R and D and $CP(D,D)_{|_\mathcal{T}}$ for overlaps amongst rules in D. With $\mathcal{T}$-critical pairs we mean $\mathcal{T}$-decorated critical pairs or $\mathcal{T}$-decoration critical pairs. In the previous definition, we can restrict our attention to superposition at occurrences $\omega \neq \Lambda$ in the set of non-variable positions of g, for the case of superposition of a rule of R into a rule of D, since superpositions at the top occurrence are already handled in the definition of a decorated critical pair. Let us consider an example of critical pairs between rules of D:

Example 7.1.6 *Assuming $x :: A$, $y :: B$, the rules:*

$$\begin{aligned} f(b^{:\{B\}})^{:s} &\to f(b^{:\{B\}})^{:s\cup\{C\}} \text{ if } \{C\} \not\subseteq s \\ (x^{:\{A\}} + f(y^{:\{B\}})^{:\emptyset})^{:s} &\to (x^{:\{A\}} + f(y^{:\{B\}})^{:\emptyset})^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s \end{aligned}$$

overlap at position 2 of the second rule. This overlapping produces the critical pair:

$$(x^{:\{A\}} + f(b^{:\{B\}})^{:\{C\}})^{:s} = (x^{:\{A\}} + f(b^{:\{B\}})^{:\emptyset})^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s$$

Let us now consider an example of superposition of a rule of R into a rule of D:

Example 7.1.7 *Assuming $x :: A$, $y :: B$, the rule:*

$$f(y^{:\{B\}})^{:\{B\}} \to a^{:\{A,B\}}$$

in R overlaps the rule:

$$(x^{:\{A\}} + f(b^{:\{B\}})^{:\{B\}})^{:s} \to (x^{:\{A\}} + f(b^{:\{B\}})^{:\{B\}})^{:s\cup\{A\}} \text{ if } A \not\subseteq s$$

in D at position 2. This overlapping produces the critical pair:

$$(x^{:\{A\}} + a^{:\{A,B\}})^{:s} = (x^{:\{A\}} + f(b^{:\{B\}})^{:\{B\}})^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s.$$

Note that the right-hand side of this conditional critical pair is immediately reducible by the rule in R. The simplified pair can be oriented into a new rule of D:

$$(x^{:\{A\}} + a^{:\{A,B\}})^{:s} \to (x^{:\{A\}} + a^{:\{A,B\}})^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s.$$

To conclude this section, we prove the completeness of $\mathbf{UNIF_d}$ for the unification problems in the $CP_\mathcal{T}$'s, assuming the completeness over the set $strict\_subterm\_set(\mathcal{T})$ of all strict subterms of elements in $\mathcal{T}$. This is a consequence of the fact that we do not superpose into variables.

Lemma 7.1.8 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and let us assume furthermore that $terms(D \cup R) \subseteq Valid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. If $\mathbf{UNIF_d}$ (see Figures 5.4 and 5.5) is $strict\_subterm\_set(\mathcal{T})$-complete, then it is also $\mathcal{T}$-complete for all unification problems in $CP(R,R)_{|_\mathcal{T}}$, $CP(R,D)_{|_\mathcal{T}}$, $CP(D,D)_{|_\mathcal{T}}$ and $CP(D,R)_{|_\mathcal{T}}$.*

Proof: Suppose that the unification problem has the form $x^{:S} \cong_d^? f(t_1,\ldots,t_n)^{:T}$. If $S \subsetneq T$, this is already a problem in solved form. Otherwise the two terms are not unifiable. $\mathbf{UNIF_d}$ decides accordingly and returns $\{\sigma\}$ with $\sigma = \{x^{:S} \mapsto f(t_1,\ldots,t_n)^{:T}\}$ in the first case, which is trivially a complete solution, and $\emptyset$ in the second.

Since the Definitions 7.1.5 and 7.1.4 do not allow variable overlaps, we must have in all other cases an initial unification problem $\mathcal{U}$ of the form $f(t_1,\ldots,t_n)^{:T} \cong_d^? g(t'_1,\ldots,t'_m)^{:T'}$. If $f \neq g$, $m \neq n$ or $T \not\approx T'$, then there is no solution, else the solutions of $\mathcal{U}$ in $\mathcal{T}$ are the same as those of $\bigwedge_{i\in[1..n]} t_i \cong_d^? t'_i$ in $strict\_subterm\_set(\mathcal{T})$, where $\mathbf{UNIF_d}$ is complete by assumption. □

1.3 Overlapping Computations

Let us now classify peaks of $D \cup R$. Let $t^{:V}, t'^{:S'}, t''^{:S''}$ be decorated terms such that:

$$t'^{:S'} \leftarrow^{\omega,\alpha,\phi'}_{D\cup R} t^{:V} \rightarrowtail^{\upsilon,\beta,\phi''}_{D\cup R} t''^{:S''},$$

with l and g being respectively the left-hand sides of ϕ' and ϕ''.

In the following it is useful to keep in mind that for any rule $l \to r$ with $l \notin \mathcal{X}_\Diamond$, $Deco(l) \subsetneq Deco(r)$ and therefore $Deco(\sigma(l)) \subsetneq Deco(\sigma(r))$ (since substitutivity is fulfilled by reduction orderings and $Var(r) \subseteq Var(l)$) for any decorated substitution σ, as this is required by the definition of decorated rewrite rules and by construction for the decoration rules. Remark that the restriction to $l \notin \mathcal{X}_\Diamond$ is necessary, as the following example illustrates:

Example 7.1.9 *Let $B \leq^{syn}_{S_\Diamond} A$, $\mathcal{P} = \{b : B, a : A, x :: A, x = a\}$ and $\phi : x^{:\{A\}} \to a^{:\{A\}}$. Then, $b^{:\{A,B\}} \not\rightarrowtail_\phi a^{:\{A\}}$ and therefore the decoration decreases. However, we also have $b^{:\{A,B\}} \not\rightarrowtail_\phi b^{:\{A,B\}}$ and hence the rewrite relation is not well-founded in such a case.*

This example leads us to the following Corollary:

Corollary 7.1.10 *If $D \cup R$ is a set of valid decorated rewrite rules, s.t. there exists a rule $\phi \in R$ with $lhs(\phi) \in \mathcal{X}_\Diamond$, then $\rightarrowtail_{D\cup R}$ is not well-founded.*

Proof: If $\phi \in R$, take $\sigma = \{lhs(\phi) \mapsto rhs(\phi)\}$ in order to prove $\sigma(lhs(\phi)) \rightarrowtail_R t[\sigma(lhs(\phi))]$, for some term t. □

We therefore restrict to well-founded rewrite relations in the following. As usual for classifying peaks, let us consider the relative positions of the redexes $t_{|\omega}$ and $t_{|\upsilon}$.

Disjoint case: υ and ω are incomparable positions.

Variable overlap case: we can assume without loss of generality that υ is the top position Λ in t. Thus $t \cong_d \beta(g)$ and there exists a variable x in g whose position in t is prefix of ω.

Critical overlap case: again we can assume without loss of generality that $\upsilon = \Lambda$. Then $t \cong_d \beta(g)$ and ω is a non-variable position in g.

Lemma 7.1.11 Variable Overlap Lemma
Let $\rightarrowtail_{D\cup R}$ be well-founded and $t^{:V}, t'^{:S'}, t''^{:S''}$ be decorated terms such that $t^{:V}$ is valid and:

$$t'^{:S'} \leftarrow^{\omega,\alpha,\phi'}_{D\cup R} t^{:V} \rightarrowtail^{\Lambda,\beta,\phi''}_{D\cup R} t''^{:S''},$$

with l and g being the left-hand sides of ϕ' and ϕ'', respectively, r and d being the right-hand side of ϕ' and ϕ'', respectively, and ω being below a variable position in g. Then, the peak is convergent: $t'^{:S'} \downarrow_{D\cup R} t''^{:S''}$.

Proof: Let $x :: A$ be the variable of g, at position υ' under which the rewrite step occurs. Then $\beta(x)$ contains the redex $\alpha(l)$ at some position υ''. So $\omega = \upsilon'\upsilon''$.

Let σ be the decorated substitution defined by $\sigma(y) = \beta(y)$ if $y \neq x$ and $\sigma(x) = \beta(x)[\alpha(r)]_{\upsilon''}$. The last term is valid by Corollary 6.2.4. Furthermore, since $\rightarrowtail_{D\cup R}$ is well-founded, we know that $Deco(\beta(l)) \subsetneq Deco(\beta(r))$, and hence σ is also a decorated substitution. Since $\alpha(l) \cong_d t_{|\omega}$, the peak is convergent:

$$t'^{:S'} \overset{*}{\leftarrow}^{\alpha,\phi'}_{D\cup R} \sigma(g) \rightarrowtail^{\Lambda,\sigma,\phi''}_{D\cup R} \sigma(d) \overset{*}{\leftarrow}^{\alpha,\phi'}_{D\cup R} t''^{:S''}.$$

□

Lemma 7.1.12 $\mathcal{T}$**-Critical Pair Lemma**
Let $t^{:V}, t'^{:S'}, t''^{:S''} \in \mathcal{T}$ *be decorated terms such that:*

$$t'^{:S'} \leftarrow^{\omega,\alpha,\phi'}_{D \cup R} t^{:V} \rightarrowtail^{\Lambda,\beta,\phi''}_{D \cup R} t''^{:S''}$$

with ω *being a non-variable position in the left-hand side of* ϕ''*. Then, there exists a* $\mathcal{T}$*-critical pair:*

$$(p^{:S_1} = q^{:S_2}) \text{ if } \phi'' \in R, \text{ or } (p^{:s} = q^{:s \cup S} \text{ if } S \not\subseteq s) \text{ if } \phi'' \in D,$$

of the rule ϕ' *on the rule* ϕ'' *at position* ω*.*

Moreover, there is a decorated substitution ψ *in a* $\mathcal{T}$*-complete set of decorated unifiers according to Definitions 7.1.4, 7.1.5, such that* $\beta\alpha \gtrsim^W_d \psi$ *with* $W = Var(g) \cup Var(l)$ *and there exists a decorated substitution* τ*, such that* $t'^{:S'} \cong_d \tau(p^{:S_1})$ *and* $t''^{:S''} \cong_d \tau(q^{:S_2})$ *if* $\phi'' \in R$*, and* $t'^{:S'} \cong_d \tau(p^{:U})$*,* $t''^{:S''} \cong_d \tau(p^{:S \cup U})$ *for some* $U \not\subseteq S$ *if* $\phi'' \in D$*.*

Proof: The proof is almost similar to the usual one (cf. [Huet, 1980; Jouannaud et Kirchner, 1986]).
We prove the existence of ψ for all four cases. Let therefore ϕ' be of the form $l_1{}^{:S_{l_1}} \to r_1{}^{:S_{r_1}}$ if it is in R (resp. $(l_1{}^{:s} \to l_1{}^{:s \cup S_1}$ if $S_1 \not\subseteq s)$ if it is in D), and ϕ'' of the form $l_2{}^{:S_{l_2}} \to r_2{}^{:S_{r_2}}$ if it is in R (resp. $(l_2{}^{:s} \to l_2{}^{:s \cup S_2}$ if $S_2 \not\subseteq s)$ if it is in D).

- $\phi' \in R$, $\phi'' \in R$:
  Since $t^{:V} \cong_d \beta(l_2{}^{:S_{l_2}})$ and $t^{:V}{}_{|\omega} \cong_d \beta(l_2{}^{:S_{l_2}}{}_{|\omega}) \cong_d \alpha(l_1{}^{:S_{l_1}})$, by definition of decorated matching, and since the substitutions α and β can be supposed to have disjoint domains, $\alpha\beta$ is a decorated unifier of $l_2{}^{:S_{l_2}}{}_{|\omega} \cong^?_d l_1{}^{:S_{l_1}}$, i.e. in any $\mathcal{T}$-complete set of $\mathcal{D}$-unifiers we have a ψ with $\beta\alpha \gtrsim^W_d \psi$.
- $\phi' \in D$, $\phi'' \in R$:
  Let $Deco(t^{:V}{}_{|\omega}) = U$. Therefore, $t^{:V} \cong_d \beta(l_2{}^{:S_{l_2}})$ and $t^{:V}{}_{|\omega} \cong_d \beta(l_2{}^{:S_{l_2}}{}_{|\omega}) \cong_d \alpha(l_1{}^{:U})$. Consequently, $\alpha\beta$ is a decorated unifier of $l_2{}^{:S_{l_2}}{}_{|\omega} \cong^?_d l_1{}^{:U}$ and any $\mathcal{T}$-complete $\mathcal{D}$-unifier set must contain a ψ with $\beta\alpha \gtrsim^W_d \psi$. Furthermore U satisfies the conditions of ϕ' for s.
- $\phi' \in R$, $\phi'' \in D$:
  If $\omega = \Lambda$, then this case is already covered by the one with $\phi' \in D$ and $\phi'' \in R$. Else we have $t^{:V} \cong_d \beta(l_2{}^{:V})$ and $t^{:V}{}_{|\omega} \cong_d \beta(l_2{}^{:V}{}_{|\omega}) \cong_d \alpha(l_1{}^{:S_{l_1}})$. The existence of ψ can be obtained as before. Remark also, that V satisfies the conditions of ϕ'' for s.
- $\phi' \in D$, $\phi'' \in D$:
  Let $Deco(t^{:V}{}_{|\omega}) = U$. We have $t^{:V} \cong_d \beta(l_2{}^{:V})$ and $t^{:V}{}_{|\omega} \cong_d \beta(l_2{}^{:V}{}_{|\omega}) \cong_d \alpha(l_1{}^{:U})$. Now, $\alpha\beta$ is a $\mathcal{T}$-decorated unifier of $l_1{}^{:U} \cong^?_d l_2{}^{:V}{}_{|\omega}$. Since $l_1, l_2 \notin \mathcal{X}_\emptyset$ by the definition of a decoration rewrite rule, $\alpha\beta$ must also be a decorated $\mathcal{T}$-unifier of $l_1{}^{:\emptyset} \cong^?_d l_2{}^{:\emptyset}{}_{|\omega}$. The existence of ψ follows now as before. Remark once more, that the conditions of ϕ' and ϕ'' are fulfilled for U and V, respectively.

The critical pairs can be constructed applying the Definitions 7.1.4 and 7.1.5. Since $\psi \lesssim^W_d \alpha\beta$, Definitions 5.2.12, 5.4.6 and Proposition 5.2.14 give us the existence of a substitution τ with $\tau(p^{:S_1}) \cong_d t'^{:S'}$ and $\tau(q^{:S_2}) \cong_d t''^{:S''}$ if $\phi'' \in R$, $\tau(p^{:U}) \cong_d t'^{:S'}$ and $\tau(q^{:U \cup S}) \cong_d t''^{:S''}$ if $\phi'' \in D$. □

Definition 7.1.13 *A decorated critical pair* $(p^{:S_1} = q^{:S_2})$ *is* solved *if both decorated terms can be rewritten using* $\rightarrowtail_{D \cup R}$ *into the same decorated term, written* $p^{:S_1} \downarrow_{D \cup R} q^{:S_2}$*.*

Definition 7.1.14 *A decoration critical pair* $(p^{:s} = q^{:s \cup S}$ if $S \subsetneq s)$ *is* solved *if for any set of sorts* U *such that* $S \subsetneq U$, *we have* $p^{:U} \downarrow_{D \cup R} q^{:U \cup S}$.

Theorem 7.1.15 *Assume that* $\mathcal{P}$ *is a presentation given by a decorated rewrite system* R *and a set of decoration rewrite rules* D, *such that* $\rightarrowtail_{D \cup R}$ *is well-founded and* $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F})$ *is downward closed w.r.t.* $\rightarrowtail_{D \cup R} \cup \rhd$, *where* $\rhd$ *is the subterm ordering.*

If all $\mathcal{T}$*-critical pairs of* $D \cup R$ *are solved, then for any proof on decorated terms* $t'^{:S'} \overset{*}{\leftrightarrow}_{D \cup R} t''^{:S''}$ *containing only terms in* $\mathcal{T}$, *there exists a rewrite proof:*

$$t'^{:S'} \overset{*}{\rightarrow}_{D \cup R} u^{:U} \overset{*}{\leftarrow}_{D \cup R} t''^{:S''}.$$

Proof: Consider the proof P:

$$t' = t_0 \leftarrow\!\rightarrow_{D \cup R} t_1 \ldots t_{k-1} \leftarrow\!\rightarrow_{D \cup R} t_k \leftarrow\!\rightarrow_{D \cup R} t_{k+1} \ldots \leftarrow\!\rightarrow_{D \cup R} t_n = t''.$$

We prove by induction on the multiset $\{t_1, \ldots, t_n\}$ using the multiset-extension of $\rightarrowtail_{D \cup R} \cup \rhd$ as induction ordering, that there is a rewrite proof without peak. Remark that the combined ordering $\rightarrowtail_{D \cup R} \cup \rhd$ is well-founded if $\rightarrowtail_{D \cup R}$ is.

This can be shown by the following transformation Θ of descending sequences in the combined ordering $\blacktriangleright$ into descending sequences w.r.t. $\rightarrowtail_{D \cup R}$:

$$\Theta(t_1, t_2, \ldots) = \begin{cases} t_1, \Theta(t_2 \ldots) & \text{if } t_1 \rightarrowtail_{D \cup R} t_2 \\ t_1, t_1[\Theta(t_2 \ldots)]_\omega & \text{if } t_{1|_\omega} =_d t_2 \end{cases}$$

For all finite sequences $t_1, \ldots, t_n$ w.r.t. $\blacktriangleright$ it can be shown by induction on the length of the sequence, that $\Theta(t_1, t_2, \ldots)$ is indeed a descending sequence w.r.t. $\rightarrowtail_{D \cup R}$, since $\rightarrowtail_{D \cup R}$ is stable by context. Now, there can be no bound on the length of descending $\rightarrowtail_{D \cup R}$ sequences, if there is an infinite descending sequence w.r.t. $\blacktriangleright$, since Θ preserves the length of sequences. Consequently, $\blacktriangleright$ must be well-founded if $\rightarrowtail_{D \cup R}$ is.

Assume P contains a peak:

$$t_{k-1} \overset{\omega, \alpha, \phi'}{\underset{D \cup R}{\leftarrow}} t_k \overset{\upsilon, \beta, \phi''}{\underset{D \cup R}{\rightarrow}} t_{k+1}.$$

We are in one of the following cases, according to the relative positions of the redexes $t_{k|_\omega}$ and $t_{k|_\upsilon}$.

Disjoint case: Then the two reductions commute:

$$t_{k-1} \overset{\upsilon, \beta, \phi''}{\underset{D \cup R}{\rightarrow}} u \overset{\omega, \alpha, \phi'}{\underset{D \cup R}{\leftarrow}} t_{k+1}.$$

since the sort information that is used is independent. The new proof is smaller w.r.t. the induction ordering, since u is smaller than t_k. Therefore, there must be a rewrite proof for P':

$$t' = t_0 \leftarrow\!\rightarrow_{D \cup R} t_1 \ldots t_{k-1} \overset{\upsilon, \beta, \phi''}{\underset{D \cup R}{\rightarrow}} u \overset{\omega, \alpha, \phi'}{\underset{D \cup R}{\leftarrow}} t_{k+1} \ldots \leftarrow\!\rightarrow_{D \cup R} t_n = t''.$$

This must clearly also be a rewrite proof for P. Remark that $u \in \mathcal{T}$, since $\mathcal{T}$ is downward closed w.r.t. $\rightarrowtail_{D \cup R} \cup \rhd$.

Variable overlap case: Using Lemma 7.1.11, the peak is convergent: $t_{k-1} \downarrow_{D \cup R} t_{k+1}$. We can now use once more the induction hypothesis, as in the first case. Remark that, as before, t_{k-1}, t_{k+1} are in $\mathcal{T}$.

Critical overlap case: If the upper rule application does not happen on Λ, then we can find a peak, where all terms are strict subterms of the current peak and therefore we get the confluence by the downward closure of $\mathcal{T}$ w.r.t. $\rhd$ and stability by context of the rewrite relation.

Suppose now, the upper rule application happens on Λ. If it is a *decorated* critical pair, since all decorated $\mathcal{T}$-critical pairs are solved and by stability of the rewrite relation by substitution and context proved in Proposition 6.2.9, we have $t_{k-1} \stackrel{*}{\rightarrow}_{D \cup R} u \stackrel{*}{\leftarrow}_{D \cup R} t_{k+1}$. Since t_{k-1}, t_{k+1} are all smaller than t_k, we have by induction hypothesis a rewrite proof, thanks to downward closure of $\mathcal{T}$.

Assume now that this peak corresponds to a *decoration* critical pair of the shape ($p^{:s} = q^{:s \cup S_0}$ if $S_0 \not\subseteq s$). Since all decoration $\mathcal{T}$-critical pairs are solved, this implies again that $t_{k-1} \stackrel{*}{\rightarrow}_{D \cup R} u \stackrel{*}{\leftarrow}_{D \cup R} t_{k+1}$, giving us as before the existence of a rewrite proof.

□

Let us now consider how to solve critical pairs in order to eliminate some easy cases.

Proposition 7.1.16 *Any decoration critical pair between decoration rules in D at a position $\omega \neq \Lambda$ is solved by adding an enriched version of the upper decoration rule. Decoration critical pairs at Λ are always solved.*

Proof: Consider the decoration critical pair

$$(\psi(g[l^{:U \cup S_l}]_\omega)^{:s} = \psi(g)^{:s \cup S_g} \text{ if } S_g \not\subseteq s),$$

produced by overlapping the rule

$$\phi_1 \ : \ (l^{:s'} \rightarrow l^{:s' \cup S_l} \text{ if } S_l \not\subseteq s')$$

of D into the rule

$$\phi_2 \ : \ (g^{:s} \rightarrow g^{:s \cup S_g} \text{ if } S_g \not\subseteq s)$$

in D. Then, if $\omega \neq \Lambda$, then $q^{:s \cup S}$ contains at position ω the subterm $\psi(g^{:s \cup S_g}{}_{|\omega})$ with $Deco(\psi(g^{:\emptyset}{}_{|\omega})) = U$ and we know that $\psi(g^{:s \cup S_g}{}_{|\omega}) \cong_d \psi(l^{:U})$, since ψ is a $\mathcal{D}$-unifier of these two decorated terms. Thus the rule ϕ_2 of D applies on $q^{:s \cup S}$ and yields $\psi(g[l^{:U \cup S_l}]_\omega)^{:s \cup S_g}$. Assume now that the decoration rule

$$\phi \ : \ \psi(g[l^{:U \cup S_l}]_\omega)^{:s} \rightarrow \psi(g[l^{:U \cup S_l}]_\omega)^{:s \cup S_g} \text{ if } S_g \not\subseteq s$$

is added in D. Then for any set of sorts U' such that $S_g \not\subseteq U'$,

$$\psi(g[l^{:U \cup S_l}]_\omega)^{:U'} \rightarrow^{\phi}_{D \cup R} \psi(g[l^{:U \cup S_l}]_\omega)^{:U' \cup S_g}.$$

The critical pair is thus solved.

Superposition at Λ of two rules of D gives $p^{:s} = \psi(g)^{:s}$ and $q^{:s \cup S} = \psi(g)^{:s \cup S_g}$, which is solved in an obvious way using once more ϕ_2. □

To illustrate why the $\omega = \Lambda$ case is trivial consider the following small example:

Example 7.1.17 *Let* $D = \left\{ \begin{array}{l} (\phi_1 \ : \ a^{:s} \to a^{:s \cup \{A\}} \text{ if } A \not\subsetneq s), \\ (\phi_2 \ : \ a^{:s} \to a^{:s \cup \{B\}} \text{ if } B \not\subsetneq s) \end{array} \right\}$.
Then, $a^{:\{A\}} \leftarrow_D^{\phi_1} a^{:\emptyset} \rightarrow_D^{\phi_2} a^{:\{B\}}$ *and clearly* $a^{:\{A\}} \rightarrow_D^{\phi_2} a^{:\{A,B\}} \leftarrow_D^{\phi_1} a^{:\{B\}}$.

Proposition 7.1.18 *Any decorated critical pair in* $CP(D,R)$ *obtained by overlapping a decoration rule into a decorated rewrite rule is solved by adding an enriched version of the decorated rewrite rule and if* $\omega = \Lambda$, *then also a new decoration rule.*

Proof: The critical pair obtained by overlapping $(l^{:s} \to l^{:s \cup S_l} \text{ if } S_l \not\subsetneq s)$ into $(g^{:S_g} \to d^{:S_d})$ using a strict decorated unifier ψ is solved in the case of $\omega \neq \Lambda$ by adding

$$\psi(g^{:S_g})[\psi(l^{:U})^{:U \cup S_l}]_\omega) \to \psi(d^{:S_d})$$

where $U = Deco(g^{:S_g}{}_{|_\omega})$ and in the case of $\omega = \Lambda$ by adding

$$\psi(g^{:S_g})[\psi(l^{:S_g})^{:S_g \cup S_l}]_\Lambda \to \psi(d^{:S_d})^{:S_d \cup S_g \cup S_l}$$
and
$$\psi(d)^{:s} \to \psi(d)^{:s \cup S_g \cup S_l \setminus S_d} \text{ if } S_g \cup S_l \setminus S_d \not\subsetneq s.$$

Remark that $g \notin \mathcal{X}_\Diamond$ if $\omega = \Lambda$, since we do not calculate critical pairs on variable positions. □

Note that $S_g \subsetneq S_d$ by the definition of decorated rewrite rules. Therefore, the new rule also represents a well-formed decorated rewrite rule in the case of $\omega \neq \Lambda$. In practice we take the critical pair as equality and normalise it first, of course, before we orient it into a rule. This may let disappear the decoration subsumption property ($S_g \subsetneq S_d$) and so it becomes necessary to add a new decoration rule when the subsumption property isn't fulfilled (see Section 7.2 for details), using the same construction as above. The problem of solving decoration critical pairs may seem complex at a first glance, but becomes quite obvious by looking more closely at their form (see Examples 7.1.6 and 7.1.7). Before we come to the next proposition, recall that top-overlaps of R into D are equivalent to top-overlaps of D into R – a case that we discussed already in Proposition 7.1.18.

Proposition 7.1.19 *Any decoration critical pair from* R *into* D *at a non-top position can be solved by adding a decoration rule.*

Proof: Consider the decoration critical pair

$$(\psi(g[r^{:S_r}]_\omega)^{:s} = \psi(g)^{:s \cup S_g} \text{ if } S_g \not\subsetneq s),$$

produced by overlapping the rule

$$\phi_1 \ : \ l^{:S_l} \to r^{:S_r}$$

of R in the rule

$$\phi_2 \ : \ (g^{:s} \to g^{:s \cup S_g} \text{ if } S_g \not\subsetneq s)$$

in D. Then $\psi(g)^{:s\cup S_g}$ contains at position $\omega \neq \Lambda$ the subterm $\psi(g^{:\emptyset}|_\omega) =_d \psi(g^{:\emptyset}|_\omega)^{:U}$ and $\psi(g^{:\emptyset}|_\omega)^{:U} \cong_d \psi(l^{:S_l})$, since ψ is a $\mathcal{D}$-unifier of these two decorated terms. Thus the rule ϕ_1 of R applies on $\psi(g)^{:s\cup S_g}$ and yields $\psi(g[r^{:S_r}]_\omega)^{:s\cup S_g}$. Now for any set of sorts U' such that $S_g \not\subseteq U'$,

$$\psi(g[r^{:S_r}]_\omega)^{:U'} \rightarrowtail^{\phi}_{D\cup R} \psi(g[r^{:S_r}]_\omega)^{:U'\cup S_g}$$

using the rule:

$$\phi : (\psi(g[r^{:S_r}]_\omega)^{:s} \rightarrow \psi(g[r^{:S_r}]_\omega)^{:s\cup S_g} \text{ if } S_g \not\subseteq s).$$

The critical pair is thus solved by adding to D this rule obtained by a straightforward orientation of the reduced critical pairs. □

Notation: $\overline{CP(R,D)}$ and $\overline{CP(D,D)}$ stand for the sets of decoration rules used to solve these critical pairs. $\overline{CP(\phi,D)}$ stands for the set of decoration rules generated by some rule $\phi \in D\cup R$.

2 Transformation Rules for Order-Sorted Completion

Equalities are ordered according to a given decorated reduction ordering $>_d$ on decorated terms (see Definition 6.4.2). Rewrite rules from R and from D are compared by the following ordering, derived from [Dershowitz et Jouannaud, 1990b]:

Definition 7.2.1 *The ordering on rewrite rules $\gg$ is defined by:*

$$l^{:S_l} \rightarrow r^{:S_r} \gg g^{:S_g} \rightarrow d^{:S_d}$$

if after first replacing sort variables by the empty set of sorts then:

1. $g^{:S_g} \lhd_d l^{:S_l}$ *(a subterm of $l^{:S_l}$ is an instance of $g^{:S_g}$ modulo sort inheritance and not conversely),*
2. *or else $l^{:S_l}$ and $g^{:S_g}$ are subsumption equivalent ($l^{:S_l} \lesssim_d g^{:S_g}$ and $g^{:S_g} \lesssim_d l^{:S_l}$) and $r^{:S_r} >_d d^{:S_d}$ in the given reduction ordering.*

The completion procedure is expressed with the set $\mathcal{OSC}$ of transformation rules given in Figure 7.2. First of all, note that $\mathcal{OSC}$ only generates valid decorated terms, rules and equalities in each (D_k, E_k, R_k).

Lemma 7.2.2 *Let $(D_0,E_0,R_0) \vdash_{\mathcal{OSC}} (D_1,E_1,R_1) \vdash_{\mathcal{OSC}} \ldots$ be a derivation starting with $D_0 = D_\mathcal{P}$, $E_0 = E_\mathcal{P}$ and $R_0 = \emptyset$.*

Then for all $k \geq 0$, all terms, all decorated and decoration rewrite rules and decorated equalities in (D_k, E_k, R_k) are valid in $\mathcal{P}$.

Proof: The proof is an induction on k. For $k = 0$, obviously all terms $t^{:S}$ in $D_\mathcal{P} \cup E_\mathcal{P}$ satisfy $(t^{:S})^{:\downarrow\emptyset} =_d t^{:S}$ and are therefore trivially valid. The equations are in one-to-one correspondence with those in $\mathcal{P}$ and therefore also valid, since the set of all $\mathcal{T}(\Sigma,\mathcal{X})$-assignments is equivalent to the set of all $\mathcal{P}$-conform substitutions (by invariant H_1 in Theorem 6.5.1) and hence the substitutivity rules in **GA** give the derivability condition needed for valid equations/rules.

For $k > 0$, we get the validity of the terms in (D_k, E_k, R_k) by the fact that they either can be reached from terms in $(D_{k-1}, E_{k-1}, R_{k-1})$ by $\leftrightarrowtail_{D_{k-1},E_{k-1},R_{k-1}}$ or result from

1. **Orient_SD**
$$\frac{D, E \cup \{p^{:S} = q^{:S'}\}, R}{D, E, R \cup \{p^{:S} \to q^{:S'}\}}$$
if $p^{:S} >_d q^{:S'}$ and $S \subsetneq S'$

2. **Orient_NSD**
$$\frac{D, E \cup \{p^{:S} = q^{:S'}\}, R}{D \cup \{(q^{:s} \to q^{:s \cup S \setminus S'} \text{ if } S \setminus S' \not\subseteq s)\}, E, R \cup \{p^{:S} \to q^{:S \cup S'}\}}$$
if $p^{:S} >_d q^{:S \cup S'}$ and $S \not\subseteq S'$ and $q \notin \mathcal{X}_\Diamond$

3. **Deduce**
$$\frac{D, E, R}{D, E \cup \{(p^{:S} = q^{:S'})\}, R}$$
if $(p^{:S} = q^{:S'}) \in CP(R, R) \cup CP(D, R)$

4. **Deduce_deco**
$$\frac{D, E, R}{D \cup \{(p^{:s} \to p^{:s \cup S} \text{ if } S \not\subseteq s)\}, E, R}$$
if $(p^{:s} = p^{:s \cup S} \text{ if } S \not\subseteq s) \in \overline{CP(R, D)} \cup \overline{CP(D, D)}$

5. **Simplify**
$$\frac{D, E \cup \{(p^{:S} = q^{:S'})\}, R}{D, E \cup \{(p''^{:S''} = q^{:S'})\}, R}$$
if $p^{:S} \rightarrowtail^{\sigma,\phi}_{D \cup R} p''^{:S''}$

6. **Delete**
$$\frac{D, E \cup \{(p^{:S} = q^{:S'})\}, R}{D, E, R}$$
if $p^{:S} \cong_d q^{:S'}$

7. **Compose**
$$\frac{D, E, R \cup \{(l^{:S_l} \to r^{:S_r})\}}{D, E, R \cup \{(l^{:S_l} \to r'^{:S_{r'}})\}}$$
if $r^{:S_r} \rightarrowtail^{\sigma,\phi}_{D \cup R} r'^{:S_{r'}}$

8. **Compose_D_deco**
$$\frac{D \cup \{(p^{:s} \to p^{:s \cup S} \text{ if } S \not\subseteq s)\}, E, R}{D \cup \{(p'^{:s} \to p'^{:s \cup S'} \text{ if } S' \not\subseteq s)\}, E, R}$$
if $p^{:S} \rightarrowtail^{\omega,\sigma,\phi}_{D} p'^{:S'}$

9. **Compose_R_deco**
$$\frac{D \cup \{(p^{:s} \to p^{:s \cup S} \text{ if } S \not\subseteq s)\}, E, R}{D \cup \{(p'^{:s} \to p'^{:s \cup S} \text{ if } S \not\subseteq s)\}, E, R}$$
if $p^{:S} \rightarrowtail^{\omega,\sigma,\phi}_{R} p'^{:S}$ and $\omega \neq \Lambda$

10. **Subsume_deco**
$$\frac{D \cup \{(p^{:s} \to p^{:s \cup S} \text{ if } c(s))\}, E, R}{D, E, R}$$
if $p^{:\emptyset} \rightarrowtail^{\Lambda,\sigma,\phi}_{D} p^{:S'}$ with $S \subsetneq S'$ and $(p^{:s} \to p^{:s \cup S} \text{ if } c(s)) \neq \phi$

11. **Collapse**
$$\frac{D, E, R \cup \{(l^{:S_l} \to r^{:S_r})\}}{D, E \cup \{(l'^{:S_{l'}} = r^{:S_r})\}, R}$$
if $l^{:S_l} \rightarrowtail^{\sigma,\phi}_{D \cup R} l'^{:S_{l'}}$ & $l^{:S_l} \to r^{:S_r} \gg \phi$

Figure 7.2: *OSC* The completion rules for decorated terms.

the application of a unifier for two valid terms calculated by **UNIF$_d$** and $\rightarrowtail_{D_{k-1} \cup R_{k-1}}$, or have a new sort in its top decoration, that was at the top of a valid term reachable by $\leftarrow\!\!\rightarrowtail_{D_{k-1}, E_{k-1}, R_{k-1}}$ in the last presentation. Consequently, these terms are also valid, because of Lemma 6.2.3, used together with the induction hypothesis, and the fact that **UNIF$_d$** is subterm conservative.

The validity of the decorated rewrite rules and equalities in (D_k, E_k, R_k) is now a consequence of **EqReplacement**, **Symmetry** and **EqSubstitutivity**. Validity of decoration rewrite rules follows immediately from the validity of all terms. □

Correctness of completion amounts to prove that any formula provable in a decorated presentation $\mathcal{P}$ obtained during the completion is equivalently provable in $\mathcal{P}_0$, the initial presentation.

Proposition 7.2.3 *The transformation rules in OSC are sound w.r.t. deduction in G-algebra. In other words, if the completion starts with* $\mathcal{P}_0 = (D_\mathcal{P}, E_\mathcal{P}, \emptyset)$ *and* $\mathcal{P}_0 \vdash_{OSC} \ldots \vdash_{OSC} \mathcal{P}_k$, *then* $Form(\mathcal{P}_0) = Form(\mathcal{P}_k)$.

Proof: We show the following property $\mathbb{H}$ for all $\mathcal{P}_k$, $k \geq 0$, by induction on k: for any terms t, t', t'',

1. $(\ t^{:\downarrow\emptyset} \leftarrow\!\!\rightarrow^*_{\mathcal{P}_k} t''^{:S'' \cup \{A\}} \leftarrow\!\!\rightarrow^*_{\mathcal{P}_k} t'^{:\downarrow\emptyset})$ iff $t = t' \in Form(\mathcal{P}_0)$,
2. $(\exists A' \leq^{syn}_{S_\Diamond} \langle A \rangle : t^{:\downarrow\emptyset} \leftarrow\!\!\rightarrow^*_{\mathcal{P}_k} t'^{:S' \cup \{A'\}})$ iff $(t : A) \in Form(\mathcal{P}_0)$,
3. $(t^{:\downarrow\emptyset} \leftarrow\!\!\rightarrow^*_{\mathcal{P}_k} t'^{:S' \cup \{A\}})$ iff $(EX\ t) \in Form(\mathcal{P}_0)$.

The base case is a consequence of Theorem 6.5.1, since $\mathcal{P}_k = \mathcal{P}_0$. The induction step from $\mathcal{P}_k$ to $\mathcal{P}_{k+1}$ assumes the equivalence for $\mathcal{P}_k$ and shows then the same equivalence for $\mathcal{P}_{k+1}$. Let $\mathcal{P}_k$ be (D_k, E_k, R_k) and $\mathcal{P}_{k+1}$ be $(D_{k+1}, E_{k+1}, R_{k+1})$.

The first direction of the equivalence ($\Rightarrow$) results from Lemma 7.2.2 that guarantees the validity of all decorated rewrite rules, equalities and decoration rewrite rules. Remember that the rewrite relation is defined using substitutions, that are valid in $\mathcal{P}$ only. This does not change, when we write $\leftarrow\!\!\rightarrow^*_{\mathcal{P}_k}$, i.e. validity is still used w.r.t. the initial G-algebra presentation. Hence, we can apply Lemma 6.2.8 in order to get $Form(\mathcal{P}_{k+1}) \subseteq Form(\mathcal{P}_0)$.

The second direction of the equivalence ($\Leftarrow$) is proved by transforming any step of $\leftarrow\!\!\rightarrowtail_{\mathcal{P}_k}$ into (possibly multiple) step(s) of $\leftarrow\!\!\rightarrowtail_{\mathcal{P}_{k+1}}$. Thus $Form(\mathcal{P}_k) \subseteq Form(\mathcal{P}_{k+1})$ and by induction hypothesis $Form(\mathcal{P}_k) = Form(\mathcal{P}_0)$, which implies $Form(\mathcal{P}_0) \subseteq Form(\mathcal{P}_{k+1})$.

This is shown using a proof reduction relation $\Longrightarrow$, given also in Appendix A.1, that is also used in Section 7.3. We give to the proof reduction rules the same name as the completion rules except that they are underlined. In the sequel of the proof, we write $0, 1$ over the $\rightarrowtail$ in order to represent one or none application of rewriting. For any of the following proof reduction rules, we assume of course the conditions of the corresponding completion rules to be fulfilled.

Let Ψ be a proof in (D_k, E_k, R_k) and Ψ' a proof in $(D_{k+1}, E_{k+1}, R_{k+1})$. Let furthermore Φ and Φ' be the proofs that are obtained from Ψ and Ψ' by omitting indices from all arrows, i.e. $\rightarrowtail^{\omega,\sigma,\phi}_{R_k}$ is replaced by $\rightarrowtail^{\omega,\sigma,\phi}_{R}$ and so on. Now, formally, the pair of proofs (Ψ, Ψ') is in the proof reduction relation $\Longrightarrow$, if Φ can be transformed into Φ' using one of the transformation rules for $\Longrightarrow$ below corresponding with the completion rules. We may then simply say that the transformation rule is applied to Ψ.

Remark that the relation $\Longrightarrow$ works independently of the current decorated presentations: A given proof Ψ reduces via $\Longrightarrow$ into a proof Ψ', ignoring whether the rules used in Ψ' actually exist in $\mathcal{P}_k$ or $\mathcal{P}_{k+1}$. However, if a completion rule **CR** is applied to $\mathcal{P}_k$, then we know by construction of the transformation CR that the rules used in the $\mathcal{P}_{k+1}$ proof Ψ', obtained from Ψ in $\mathcal{P}_k$ by application of CR, must exist.

1. Orient_(N)SD:
   If $q \in \mathcal{X}_\Diamond$ with $(q :: A) \in \mathcal{P}$, we can be sure that $S' \approx \{A\}$ (see Definition 5.2.1). Furthermore, the condition of the two completion rules guarantee that only **Orient_SD** is applicable and hence $S \subsetneq S'$, i.e. $\{A\} \approx S \cup S'$. If $\{A\} \not\approx S \cup S'$, then the equation is clearly not orientable and the completion fails. We show in Section 7.5, how we can go on in this case.
   Suppose that the equality $p^{:S} = q^{:S'}$ is applicable at some term t at occurrence ω using σ, yielding t'. Then $t_{|\omega} \cong_d \sigma(p^{:S})$ and $t'_{|\omega} \cong_d \sigma(q^{:S'})$ by Definitions 5.3.1 and 6.1.2, thus the proof reduction is defined by:

$$(t[\sigma(p^{:S})] \cong_d t \longleftarrow\!\!\!\rightarrow_E^{\omega,\sigma,p^{:S}=q^{:S'}} t' \cong_d t[\sigma(q^{:S'})]) \Longrightarrow (t \rightarrowtail_R^{\omega,\sigma,p^{:S}\to q^{:S\cup S'}} t'' \overset{0,1}{\rightleftarrows}_D^{\omega,\sigma,\phi} t'),$$

   where – ϕ is $q^{:s} \to q^{:s\cup S\setminus S'}$ if $S \setminus S' \subsetneq s$,
   – $t'' \cong_d t[\sigma(q^{:S'})^{:U}]$ (by Proposition 6.2.10) and
   – $U = Deco(\sigma(q^{:S'})) \cup (S \setminus S')$.
   Remark that the equality symbol is commutative and therefore the inverse application of a rule can be transformed analogously. Note furthermore that $Deco(\sigma(q^{:S'}))$ is not necessarily S', since q may be a variable in which case σ can extend the decoration.

2. Deduce:

$$(t' \longleftarrow\!\!\!\rightarrow_{R\cup D}^{\omega,\sigma,\phi} t \rightarrowtail_R^{\nu,\sigma',\phi'} t'') \Longrightarrow (t' \longleftarrow\!\!\!\rightarrow_E^{\nu,\tau,p^{:S}=q^{:S'}} t'')$$

   where τ is defined by $\sigma \circ \sigma'(t) \cong_d \tau \circ \psi(t)$ for some unifier ψ of the left hand sides of ϕ and ϕ' according to the definitions of $CP(R, R)$ and $CP(D, R)$.
   Remark that $\sigma \circ \sigma'$ and τ are decorated substitutions by Corollary 5.2.11 respectively Definitions 5.2.12 and 5.4.6.

3. Deduce_deco:

$$(t' \longleftarrow\!\!\!\rightarrow_{D\cup R}^{\omega,\sigma',\phi'} t \rightarrowtail_D^{\nu,\sigma,\phi} t'') \Longrightarrow (t' \rightarrowtail_D^{\nu,\tau,(p^{:s}\to p^{:s\cup S} \text{ if } S\subsetneq s)} t'_0 \overset{0,1}{\rightleftarrows}_{D\cup R}^{\omega,\sigma',\phi'} t''),$$

   where τ is defined similarly as before by $CP(R, D)$ resp. $CP(D, D)$.

4. Simplify:
   First of all $\phi \in D' \cup R'$ since $D \cup R = D' \cup R'$. Since $p^{:S} = q^{:S'}$ is applicable to t, we know that σ' (see below) is a decorated substitution for t and the application of ϕ to $p^{:S}$ guarantees us the same for σ. If $\phi : g^{:S_g} \to d^{:S_d}$, the simplifying rule, is from R, then:

$$(t[\sigma'(p^{:S}[\sigma(g^{:S_g})]_\omega)]_\nu \cong_d t[\sigma'(p^{:S})]_\nu \cong_d t \longleftarrow\!\!\!\rightarrow_E^{\nu,\sigma',p^{:S}=q^{:S'}} t' \cong_d t[\sigma'(q^{:S'})]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_R^{\nu,\omega,\sigma'\circ\sigma,\phi} t'' \longleftarrow\!\!\!\rightarrow_E^{\nu,\sigma',p''^{:S''}=q^{:S'}} t')$$

   where – $p''^{:S''} \cong_d p^{:S}[\sigma(d^{:S_d})]_\omega$ and
   – $t'' \cong_d t[\sigma'(p^{:S}[\sigma(d^{:S_d})]_\omega)]_\nu$ by Proposition 6.2.10.

Now let $\phi : (l^{:s} \to l^{:s \cup S_l}$ if $S_l \lesssim s) \in D$. We define therefore:

$$(t[\sigma'(p^{:S}[\sigma(l^{:U})]_\omega)]_\nu \cong_d t[\sigma'(p^{:S})]_\nu \cong_d t \longleftrightarrow_E^{\nu,\sigma',p^{:S}=q^{:S'}} t' \cong_d t[\sigma'(q^{:S'})]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_D^{\nu,\omega,\sigma' \circ \sigma,\phi} t'' \longleftrightarrow_E^{\nu,\sigma',p''^{:S''}=q^{:S'}} t')$$

where – $t'' \cong_d t[\sigma'(p^{:S})]_\nu[\sigma' \circ \sigma(l^{:U})^{:U \cup S_l}]_{\nu,\omega}$ (by Proposition 6.2.10) and
– $p''^{:S''} =_d p^{:S}[\sigma(l^{:U})^{:U \cup S_l}]_\omega$.

5. Delete:

$$(t \longleftrightarrow_E^{\omega,\sigma,p^{:S}=q^{:S'}} t) \Longrightarrow \Phi$$

where Φ denotes the empty proof.

6. Compose:
As in the case of **Simplify** we know that σ, σ' and $\sigma' \circ \sigma$ are decorated substitutions. Furthermore, ϕ is in $D' \cup R'$, since no rule can be composed with itself (since this would be in contradiction to the termination of the rules in $D \cup R$, which is implied by the orientation with a decorated reduction ordering). If $\phi : g^{:S_g} \to d^{:S_d}$, the simplifying rule, is from R, then:

$$(t[\sigma'(l^{:S_l})]_\nu \cong_d t \rightarrow_R^{\nu,\sigma',l^{:S_l} \to r^{:S_r}} t' \cong_d t[\sigma'(r^{:S_r})]_\nu \cong_d t[\sigma'(r^{:S_r}[\sigma(g^{:S_g})]_\omega)]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrow_R^{\nu,\sigma',l^{:S_l} \to r'^{:S_{r'}}} t'' \leftarrow_R^{\nu,\omega,\sigma' \circ \sigma,\phi} t')$$

where – $r'^{:S_{r'}} \cong_d r^{:S_r}[\sigma(d^{:S_d})]_\omega$ and
– $t'' \cong_d t[\sigma'(r^{:S_r}[\sigma(d^{:S_d})]_\omega)]_\nu$ (by Proposition 6.2.10).

If $\phi : (g^{:s} \to g^{:s \cup S_g}$ if $S_g \lesssim s) \in D$, we get:

$$(t[\sigma'(l^{:S_l})]_\nu \cong_d t \rightarrow_R^{\nu,\sigma',l^{:S_l} \to r^{:S_r}} t' \cong_d t[\sigma'(r^{:S_r})]_\nu \cong_d t[\sigma'(r^{:S_r}[\sigma(g^{:U})]_\omega)]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrow_R^{\nu,\sigma',l^{:S_l} \to r'^{:S_{r'}}} t'' \leftarrow_D^{\nu,\omega,\sigma' \circ \sigma,\phi} t')$$

where – $t'' \cong_d t[\sigma'(r^{:S_r}[\sigma(g^{:U})^{:U \cup S_g}]_\omega)]_\nu$ by Proposition 6.2.10 and
– $r'^{:S_{r'}} =_d r^{:S_r}[\sigma(g^{:U})^{:U \cup S_g}]_\omega$.

7. Compose_D_deco:
Let $\phi : (l^{:s} \to l^{:s \cup S_l}$ if $S_l \lesssim s)$, $\phi' : (p^{:s} \to p^{:s \cup S}$ if $S \lesssim s)$ and $\phi'' : (p'^{:s} \to p'^{:s \cup S'}$ if $S' \lesssim s)$. As in the **Simplify** case we can conclude that $\sigma, \sigma', \sigma' \circ \sigma$ (for the implicit definition of σ' see below) are decorated substitutions.
Case $\omega \neq \Lambda$, i.e. $S = S'$ and $p'^{:S'} =_d p^{:S}[\sigma(l^{:U})^{:U \cup S_l}]_\omega$:

$$(t[\sigma'(p^{:U_1}[\sigma(l^{:U_2})]_\omega)]_\nu \cong_d t \rightarrow_D^{\nu,\sigma',\phi'} t' \cong_d t[\sigma'(p^{:U_1}[\sigma(l^{:U_2})]_\omega)^{:U_1 \cup S}]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrow_D^{\nu,\omega,\sigma' \circ \sigma,\phi} t'' \rightarrow_D^{\nu,\sigma',\phi''} t''' \leftarrow_D^{\nu,\omega,\sigma' \circ \sigma,\phi} t')$$

where – $t'' \cong_d t[\sigma'(p^{:U_1})]_\nu[\sigma' \circ \sigma(l^{:U_2})^{:U_2 \cup S_l}]_{\nu,\omega}$ and
– $t''' \cong_d t[\sigma'(p^{:U_1})^{:U_1 \cup S}]_\nu[\sigma' \circ \sigma(l^{:U_2})^{:U_2 \cup S_l}]_{\nu,\omega}$ by Proposition 6.2.10.

Case $\omega = \wedge$, i.e. $S' = S \cup S_l$ and $p^{:\emptyset} =_d p'^{:\emptyset}$:

$$(t[\sigma'(p^{:U}[\sigma(l^{:U})]_\wedge)]_\nu \cong_d t[\sigma'(p^{:U})]_\nu \cong_d t \rightarrowtail_D^{\nu,\sigma',\phi'} t' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\wedge)^{:U\cup S}]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_D^{\nu,\sigma',\phi''} t'' \leftarrowtail_D^{\nu,\sigma',\phi} t')$$

where $t'' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\wedge)^{:U\cup S\cup S_l}]_\nu$ (by Proposition 6.2.10).

8. Compose_R_deco:
Like with **Simplify**, we can argue that σ, σ' and $\sigma \circ \sigma'$ are decorated substitutions. Let $\phi : l^{:S_l} \to r^{:S_r}$, $\phi' : (p^{:s} \to p^{:s\cup S}$ if $S \not\subseteq s)$ and $\phi'' : (p'^{:s} \to p'^{:s\cup S}$ if $S \not\subseteq s)$, then:

$$(t[\sigma'(p^{:U}[\sigma(l^{:S_l})]_\omega)]_\nu \cong_d t[\sigma'(p^{:U})]_\nu \cong_d t \rightarrowtail_D^{\nu,\sigma',\phi'} t' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:S_l})]_\omega)^{:U\cup S}]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_R^{\nu,\omega,\sigma'\circ\sigma,\phi} t'' \rightarrowtail_D^{\nu,\sigma',\phi''} t''' \leftarrowtail_R^{\nu,\omega,\sigma'\circ\sigma,\phi} t')$$

where – $p'^{:U} = p^{:U}[\sigma(r^{:S_r})]_\omega$,
– $t'' \cong_d t[\sigma'(p^{:U}[\sigma(r^{:S_r})]_\omega)]_\nu$ and
– $t''' \cong_d t[\sigma'(p^{:U}[\sigma(r^{:S_r})]_\omega)^{:U\cup S}]_\nu$ (by Proposition 6.2.10).

9. Subsume_deco:
As in the **Simplify** case we may conclude that $\sigma' \circ \sigma$ and τ in the following are decorated substitutions. Let ϕ be $l^{:s} \to l^{:s\cup S_l}$ if $S_l \not\subseteq s$ and ϕ' be $p^{:s} \to p^{:s\cup S}$ if $S \not\subseteq s$, s.t. $S_l \subsetneq S$. Then:

$$(t[\sigma'(p^{:U}[\sigma(l^{:U})]_\wedge)]_\nu \cong_d t[\sigma'(p^{:U})]_\nu \cong_d t \rightarrowtail_D^{\nu,\sigma',\phi'} t' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\wedge)^{:U\cup S}]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_D^{\nu,\sigma',\phi} t'' \underset{D}{\overset{0,1}{\rightleftarrows}}{}^{\nu,\sigma',\phi} t')$$

where $t'' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\wedge)^{:U\cup S_l}]_\nu$ (by Proposition 6.2.10).

10. Collapse:
As in the **Simplify** case we may conclude that $\sigma' \circ \sigma$ is a decorated substitution. Furthermore the condition of **Collapse** prevents us from collapsing a rule with itself and therefore we have $\phi \in D' \cup R'$.
Let $\phi : g^{:S_g} \to d^{:S_d}$ be in R. We get:

$$(t[\sigma'(l^{:S_l}[\sigma(g^{:S_g})]_\omega)]_\nu \cong_d t[\sigma'(l^{:S_l})]_\nu \cong_d t \rightarrowtail_R^{\nu,\sigma',l^{:S_l}\to r^{:S_r}} t' \cong_d t[\sigma'(r^{:S_r})]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_R^{\nu,\omega,\sigma'\circ\sigma,\phi} t'' \leftarrowtail\!\!\rightarrowtail_E^{\nu,\sigma',l^{:S_l}=r^{:S_r}} t')$$

where – $l'^{:S_{l'}} \cong_d l^{:S_l}[\sigma(d^{:S_d})]_\omega$ and
– $t'' \cong_d t[\sigma'(l^{:S_l}[\sigma(d^{:S_d})]_\omega)]_\nu$ (by Proposition 6.2.10).
Else $\phi : (g^{:s} \to g^{:s\cup S_g}$ if $S_g \not\subseteq s) \in D$ and we have:

$$(t[\sigma'(l^{:S_l})]_\nu \cong_d t \rightarrowtail_R^{\nu,\sigma',l^{:S_l}\to r^{:S_r}} t' \cong_d t[\sigma'(r^{:S_r})]_\nu)$$
$$\Longrightarrow$$
$$(t \rightarrowtail_D^{\nu,\omega,\sigma'\circ\sigma,\phi} t'' \leftarrowtail\!\!\rightarrowtail_E^{\nu,\sigma',l'^{:S_{l'}}=r^{:S_r}} t')$$

where – $t'' \cong_d t[\sigma'(l^{:S_l}[\sigma(g^{:U})^{:U\cup S_g}]_\omega)]_\nu$ (by Proposition 6.2.10) and
– $l'^{:S_{l'}} \cong_d l^{:S_l}[\sigma(g^{:U})^{:U\cup S_g}]_\omega$.

□

3 Proof Reduction and Reflection

Let us now extend the set of proof reduction rules $\Longrightarrow$ introduced in the last section (cf. Appendix A.1). The last case to consider are peaks which are implicitly reducible in any decorated presentation, i.e. peaks without critical pairs. Theorem 7.1.15 allows reducing such peaks by proof reduction rules of the form:

$$t'^{:S'} \leftarrow^{\phi'}_{D\cup R} t^{:S} \rightarrowtail^{\phi''}_{D\cup R} t''^{:S''} \Longrightarrow t'^{:S'} \overset{*}{\rightarrowtail}^{\phi''}_{D\cup R} t_0{}^{:S_0} \overset{*}{\leftarrowtail}^{\phi'}_{D\cup R} t''^{:S''},$$

called Peak without overlap and Peak with variable overlap according to the current peak type. Note that t, t', t'' denote any decorated terms in this rule. The next step is to prove that $\Longrightarrow$ is well-founded.

Lemma 7.3.1 *The proof reduction relation $\Longrightarrow$ is well-founded.*

Proof: Define the complexity measure of an elementary proof step by:

$$\begin{aligned} c(t^{:S_1} \rightarrowtail^{\phi}_{D\#} t^{:S_2}) &= (\{t^{:S_1}\}, \phi, t^{:S_2}), \\ c(t^{:S_1} \leftarrow\!\!\rightarrowtail^{\phi}_{E} t'^{:S_2}) &= (\{t^{:S_1}, t'^{:S_2}\}, \phi, -), \\ c(t^{:S_1} \rightarrowtail^{\phi}_{R} t'^{:S_2}) &= (\{t^{:S_1}\}, \phi, t'^{:S_2}). \end{aligned}$$

By convention, the complexity of the empty proof Φ is $c(\Phi) = \emptyset$. Complexities of elementary proof steps are compared using the lexicographic combination denoted $>_{ec}$ of the multiset extension $>_d^{mult}$ of $>_d$ on decorated terms, $\gg$ on formulas and again $>_d$ on decorated terms. Since $>_d$ and $\gg$ are well-founded, so is $>_{ec}$.

The complexity of a non-elementary proof is the multiset of the complexities of its proof steps. Complexities of non-elementary proofs are compared using the multiset extension $>_c$ of $>_{ec}$, which is also well-founded. Remark that $t \cong_d u$, $t' \cong_d u'$ and $t <_d t'$ implies $u <_d u'$, by Definition 6.4.2 of decorated reduction orderings. Therefore, we can work with a representative of the equivalence class modulo $\cong_d$ of a term occurring in a proof.

- Orient_(N)SD:
  Let $t \cong_d t[\sigma(p^{:S})]$ and $t' \cong_d t[\sigma(q^{:S'})]$.
  $c(t \leftarrow\!\!\rightarrowtail^{\sigma, p^{:S}=q^{:S'}}_{E} t') = \{(\{t, t'\}, p^{:S} = q^{:S'}, -)\} >_c$
  $c(t[\sigma(p^{:S})] \rightarrowtail^{\sigma', p^{:S}\to q^{:S\cup S'}}_{R} t[\sigma(q^{:S'})^{:U}] \underline{0,1}^{\sigma,\phi}_{D} t[\sigma(q^{:S'})]) =$
  $\{(\{t\}, p^{:S} \to q^{:S\cup S'}, t[\sigma(q^{:S'})^{:U}])\} \cup H,$
  where H is $\emptyset$ or $\{(\{t'\}, \phi, t[\sigma(q^{:S'})^{:U}])\}$ and ϕ is $q^{:s} \to q^{:s\cup S\setminus S'}$ if $S \setminus S' \subsetneq s$,
  since $\{t, t'\} >_d^{mult} \{t\}, \{t'\}$.
- Deduce:
  $c(t' \leftarrow^{\sigma,\phi}_{R\cup D} t \rightarrowtail^{\sigma',\phi'}_{R} t'') = \{(\{t\}, \phi, t'), (\{t\}, \phi', t'')\} >_c$
  $c(t' \leftarrow\!\!\rightarrowtail^{\nu,\tau,p^{:S}=q^{:S'}}_{E} t'') = \{(\{t', t''\}, p^{:S} = q^{:S'}, -)\}$
  just by comparing the first components, since $t >_d t'$ and $t >_d t''$.
- Deduce_deco:
  $c(t' \leftarrow^{\sigma',\phi'}_{D\cup R} t \rightarrowtail^{\sigma,\varphi}_{D} t'') = \{(\{t\}, \phi', t'), (\{t\}, \phi, t'')\} >_c$
  $c(t' \rightarrowtail^{\tau,\phi''}_{D} t'_0 \underline{0,1}^{\sigma',\phi'}_{D\cup R} t'') = \{(\{t'\}, \phi', t'_0)\} \cup H,$
  where H is $\emptyset$ or $\{(\{t''\}, \phi', t'_0)\}$ and ϕ'' is $(p^{:s} \to p^{:s\cup S}$ if $S \subsetneq s)$
  just by comparing the first components, since $t >_d t'$ and $t >_d t''$.

- Simplify:
  case $\phi \in R$:
  $c(t \leftarrow\!\!\!\rightarrow_E^{\sigma', p^{:S} = q^{:S'}} t') = \{(\{t, t'\}, p^{:S} = q^{:S'}, -)\} >_c$
  $c(t \rightarrowtail_R^{\sigma' \circ \sigma, \phi} t'' \leftarrow\!\!\!\rightarrow_E^{\sigma', p''^{:S''} = q^{:S'}} t') = \{(\{t\}, \phi, t''), (\{t'', t'\}, p''^{:S''} = q^{:S'}, -)\}$,
  where $t \cong_d t[\sigma'(p^{:S}[\sigma(g^{:S_g})])]$, $t' \cong_d t[\sigma'(q^{:S'})]$ and $t'' \cong_d t[\sigma'(p^{:S}[\sigma(d^{:S_d})])]$,
  since $\{t, t'\} >_d^{mult} \{t\}$ and $\{t, t'\} >_d^{mult} \{t'', t'\}$.
  case $\phi \in D$:
  $c(t \leftarrow\!\!\!\rightarrow_E^{\sigma', p^{:S} = q^{:S'}} t') = \{(\{t, t'\}, p^{:S} = q^{:S'}, -)\} >_c$
  $c(t \rightarrowtail_D^{\sigma' \circ \sigma, \phi} t'' \leftarrow_E^{\tau, p''^{:S''} = q^{:S'}} t') = \{(\{t\}, \phi, t''), (\{t', t''\}, p''^{:S''} = q^{:S'}, -)\}$,
  where $t \cong_d t[\sigma'(p^{:S}[\sigma(l^{:U})]_\omega)]_\nu$, $t' \cong_d t[\sigma'(q^{:S'})]_\nu$ and
  $t'' \cong_d t[\sigma'(p^{:S})]_\nu[\sigma' \circ \sigma(l^{:U})^{:U \cup S_l}]_{\nu.\omega}$.
  Clearly, $\{t, t'\} >_d^{mult} \{t\}$ and $\{t, t'\} >_d^{mult} \{t', t''\}$ gives the reduction of the complexity measure.
- Delete:
  $c(t \leftarrow\!\!\!\rightarrow_E^{p^{:S} = p^{:S}} t) = \{(\{t, t\}, p = p, -)\} >_c c(\Phi) = \emptyset$
  since $\{t, t\} >_d^{mult} \emptyset$.
- Compose:
  case $\phi \in R$:
  $c(t \rightarrowtail_R^{\sigma', l^{:S_l} \to r^{:S_r}} t') = \{(\{t\}, l^{:S_l} \to r^{:S_r}, t')\} >_c$
  $c(t \rightarrowtail_R^{\sigma', l^{:S_l} \to r'^{:S_{r'}}} t'' \leftarrow_R^{\sigma' \circ \sigma, \phi} t') = \{(\{t\}, l^{:S_l} \to r'^{:S_{r'}}, t''), (\{t'\}, \phi, t'')\}$,
  where $t \cong_d t[\sigma'(l^{:S_l})]$, $t' \cong_d t[\sigma'(r^{:S_r}[\sigma(g^{:S_g})])]$ and $t'' =_d t[\sigma'(r^{:S_r}[\sigma(d^{:S_d})])]$,
  since $r^{:S_r} >_d r'^{:S_{r'}}$ – this implies $(l^{:S_l} \to r^{:S_r}) \gg (l^{:S_l} \to r'^{:S_{r'}})$ – and $t >_d t'$.
  case $\phi \in D$:
  $c(t \rightarrowtail_R^{\sigma', l^{:S_l} \to r^{:S_r}} t') = \{(\{t\}, l^{:S_l} \to r^{:S_r}, t')\} >_c$
  $c(t \rightarrowtail_R^{\tau, l^{:S_l} \to r'^{:S_{r'}}} t'' \leftarrow_D^{\sigma' \circ \sigma, \phi} t') = \{(\{t\}, l^{:S_l} \to r'^{:S_{r'}}, t''), (\{t'\}, \phi, t'')\}$,
  where $t \cong_d t[\sigma'(l^{:S_l})]$, $t' \cong_d t[\sigma'(r^{:S_r}[\sigma(g^{:U})])]$ and $t'' \cong_d t[\sigma'(r^{:S_r}[\sigma(g^{:U})^{:U \cup S_g}])]$,
  since $r^{:S_r} >_d r'^{:S_{r'}}$ – this implies once more $(l^{:S_l} \to r^{:S_r}) \gg (l^{:S_l} \to r'^{:S_{r'}})$ – and $t >_d t'$.
- Compose_D_deco:
  case $\omega \neq \Lambda$:
  $c(t \rightarrowtail_D^{\sigma', \phi'} t') = \{(\{t\}, \phi', t')\} >_c$
  $c(t \rightarrowtail_D^{\sigma' \circ \sigma, \phi} t'' \rightarrowtail_D^{\tau, \phi''} t''' \leftarrow_D^{\sigma' \circ \sigma, \phi} t') = \{(\{t\}, \phi, t''), (\{t''\}, \phi'', t'''), (\{t'\}, \phi, t''')\}$,
  where $t \cong_d t[\sigma'(p^{:U_1}[\sigma(l^{:U_2})]_\omega)]_\nu$, $t' \cong_d t[\sigma'(p^{:U_1}[\sigma(l^{:U_2})]_\omega)^{:U_1 \cup S}]_\nu$,
  $t'' \cong_d t[\sigma'(p^{:U_1})]_\nu[\sigma' \circ \sigma(l^{:U_2})^{:U_2 \cup S_l}]_{\nu.\omega}$ and
  $t''' \cong_d t[\sigma'(p^{:U_1})^{:U_1 \cup S}]_\nu[\sigma' \circ \sigma(l^{:U_2})^{:U_2 \cup S_l}]_{\nu.\omega}$,
  since $l^{:S_\omega} \not\lhd_d p^{:\emptyset}$, where $Deco(p^{:\emptyset}{}_{|_\omega}) = S_\omega$, therefore $\phi' \gg \phi$, and trivially $t >_d t'$ and $t >_d t''$.
  case $\omega = \Lambda$:
  $c(t \rightarrowtail_D^{\sigma', \phi'} t') = \{(\{t\}, \phi', t')\} >_c c(t \rightarrowtail_D^{\sigma', \phi''} t'' \leftarrow_D^{\tau, \phi} t') = \{(\{t\}, \phi'', t''), (\{t'\}, \phi, t'')\}$,
  where $t \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\Lambda)]$, $t' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\Lambda)^{:U \cup S}]$
  and $t'' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:U})]_\Lambda)^{:U \cup S \cup S_l}]$,
  since either $l^{:\emptyset} \not\lhd_d p^{:\emptyset}$ or $S \subset S' = S \cup S_l$, therefore $\phi' \gg \phi''$, and trivially $t >_d t'$.
- Compose_R_deco:
  $c(t \rightarrowtail_D^{\sigma', \phi'} t') = \{(\{t\}, \phi', t')\} >_c$

$c(t \rightarrowtail_R^{\sigma' \circ \sigma, \phi} t'' \rightarrowtail_D^{\sigma', \phi''} t''' \leftarrowtail_R^{\sigma' \circ \sigma, \phi} t') = \{(\{t\}, \phi, t''), (\{t''\}, \phi'', t'''), (\{t'\}, \phi, t''')\}$,
where
$t \cong_d t[\sigma'(p^{:U}[\sigma(l^{:S_l})])]$, $t' \cong_d t[\sigma'(p^{:U}[\sigma(l^{:S_l})])^{:U \cup S}]$, $t'' \cong_d t[\sigma'(p^{:U}[\sigma(r^{:S_r})])]$
and $t''' \cong_d t[\sigma'(p^{:U}[\sigma(r^{:S_r})])^{:U \cup S}]$,
because of $\phi' \gg \phi$ (since $\omega \neq \Lambda$ and therefore $l^{:S_l}$ must be strictly embedded in $p^{:S}$), and finally $t >_d t'', t'$.

- Subsume_deco:
$c(t \rightarrowtail_D^{\sigma', \phi'} t') = \{(\{t\}, \phi', t')\} >_c c(t \rightarrowtail_D^{\sigma' \circ \sigma, \phi} t'' \leftarrowtail_D^{\tau, \phi} t') = \{(\{t\}, \phi, t''), (\{t'\}, \phi, t'')\}$
where $t \cong_d t^{:T}[\sigma'(p^{:U}[\sigma(l^{:U})]_\Lambda)]$, $t' \cong_d t^{:T}[\sigma'(p^{:U}[\sigma(l^{:U})]_\Lambda)^{:U \cup S}]$
and $t'' \cong_d t^{:T}[\sigma'(p^{:U}[\sigma(l^{:U})]_\Lambda)^{:U \cup S_l}]$,
because of $\phi' \gg \phi$ (since $\phi' \neq \phi$ we have either $p^{:\emptyset}$ as strict instance of $p'^{:\emptyset}$ - then we are done - or the two terms are equal modulo variable renaming implying $S \subset S'$ and therefore $p^{:S} >_d p'^{:S'}$). Finally, $t >_d t'$, since $U \not\subseteq S$.

- Collapse:
case $\phi \in R$:
$c(t \rightarrowtail_R^{\sigma', l^{:S_l} \to r^{:S_r}} t') = \{(\{t\}, l^{:S_l} \to r^{:S_r}, t')\} >_c$
$c(t \rightarrowtail_R^{\sigma' \circ \sigma, \phi} t'' \leftarrowtail\!\!\rightarrowtail_E^{\sigma', l'^{:S_{l'}} = r^{:S_r}} t') = \{(\{t\}, \phi, t''), (\{t'', t'\}, l'^{:S_{l'}} = r^{:S_r}, -)\}$
where $t \cong_d t[\sigma'(l^{:S_l}[\sigma(g^{:S_g})])]$, $t' \cong_d t[\sigma'(r^{:S_r})]$ and $t'' \cong_d t[\sigma'(l^{:S_l}[\sigma(d^{:S_d})])]$,
since $(l^{:S_l} \to r^{:S_r}) \gg \phi$ and $t >_d t', t''$.
case $\phi \in D$:
$c(t \rightarrowtail_R^{\sigma', l^{:S_l} \to r^{:S_r}} t') = \{(\{t\}, l^{:S_l} \to r^{:S_r}, t')\} >_c$
$c(t \rightarrowtail_D^{\sigma' \circ \sigma, \phi} t'' \leftarrowtail\!\!\rightarrowtail_E^{\tau, l'^{:S_{l'}} = r^{:S_r}} t') = \{(\{t\}, \phi, t''), (\{t'', t'\}, l'^{:S_{l'}} = r^{:S_r}, -)\}$,
where $t \cong_d t[\sigma'(l^{:S_l})]$, $t' \cong_d t[\sigma'(r^{:S_r})]$ and $t'' \cong_d t[\sigma'(l^{:S_l}[\sigma(g^{:U})^{:U \cup S_g}])]$,
because of similar reasons as in the case of $\phi \in R$.

- Peak without overlap:
$c(t' \leftarrowtail_{D \cup R}^{\phi} t \rightarrowtail_{D \cup R}^{\phi'} t'') = \{(\{t\}, \phi, t'), (\{t\}, \phi', t'')\} >_c$
$c(t' \rightarrowtail_{D \cup R}^{\phi'} t_1 \leftarrowtail_{D \cup R}^{\phi} t'') = \{(\{t'\}, \phi', t_1), (\{t''\}, \phi, t_1)\}$
just by comparing the first components: $t >_d t'$ and $t >_d t''$.

- Peak with variable overlap:
$c(t' \leftarrowtail_{D \cup R}^{\phi} t \rightarrowtail_{D \cup R}^{\phi'} t'') >_c c(t' \overset{*}{\rightarrowtail}_{D \cup R} t_1 \overset{*}{\leftarrowtail}_{D \cup R} t'')$
again just by comparing the first components of each elementary step appearing in these proofs.

□

The exact correspondence between the proof reduction $\Longrightarrow$ and the derivation $\vdash_{\mathcal{OSC}}$ is stated by the following results.

These proof reduction rules must *reflect* the rules of $\mathcal{OSC}$ in the following sense: at each step $\vdash_{\mathcal{OSC}}$, a given proof either does not change or is transformed into another one by $\Longrightarrow$.

Definition 7.3.2 *[Bachmair, 1991]*
The proof reduction $\Longrightarrow$ reflects $\vdash_{\mathcal{OSC}}$ *if whenever* $(D_i, E_i, R_i) \vdash_{\mathcal{OSC}} (D_{i+1}, E_{i+1}, R_{i+1})$ *and* P *is a proof in* (D_i, E_i, R_i), *then there is a proof* P' *in* $(D_{i+1}, E_{i+1}, R_{i+1})$ *such that* $P \overset{*}{\Longrightarrow} P'$.

Because the $\Longrightarrow$-rules have been built from the rules of $\mathcal{OSC}$, it is easy to verify that:

Proposition 7.3.3 $\Longrightarrow$ *reflects* $\vdash_{\mathcal{OSC}}$.

4 Fairness

The fairness hypothesis states that any proof reducible by $\Longrightarrow$ will eventually be reduced. In other words, no reducible proof is forgotten. Fairness specifies under which conditions a control for applying rules of $\mathcal{OSC}$ is correct. Fairness is again defined relatively to a subset $\mathcal{T}$ of valid terms.

Definition 7.4.1 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. A derivation $(D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ is $\mathcal{T}$-*fair *if whenever Ψ is a proof in (D_i, E_i, R_i), that uses only terms in $\mathcal{T}$ and is reducible by $\Longrightarrow$, then there is a proof Ψ' in (D_j, E_j, R_j) at some step $j \geq i$ such that $\Psi \stackrel{+}{\Longrightarrow} \Psi'$.*

When $\mathcal{T} = \mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, we drop the $\mathcal{T}$ prefix, since the notion is then obviously equivalent to classical fairness. Let us define:

$$D_* = \bigcup_{i\geq 0} D_i \qquad E_* = \bigcup_{i\geq 0} E_i \quad \text{and} \quad R_* = \bigcup_{i\geq 0} R_i$$
$$D_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} D_j \quad E_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} E_j \quad \text{and} \quad R_\infty = \bigcup_{i\geq 0} \bigcap_{j>i} R_j.$$

A sufficient condition to satisfy the fairness hypothesis can be given. Remark that a set of rewrite rules is called *interreduced* if no more simplifying completion rule can be applied.

Proposition 7.4.2 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and assume that* $\mathbf{UNIF_d}$ *(see Figures 5.4 and 5.5) is a strict_subterm_set($\mathcal{T}$)-complete unification algorithm.*

A derivation $(D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ using $\mathbf{UNIF_d}$ *for the calculation of critical pairs is $\mathcal{T}$-fair if E_∞ is empty, all critical pairs of $D_\infty \cup R_\infty$ are in $D_* \cup E_*$ and $D_\infty \cup R_\infty$ is interreduced.*

Proof: We have to prove that if Ψ is a proof in (D_i, E_i, R_i) that uses only terms in $\mathcal{T}$ and is reducible by $\Longrightarrow$, then there is Ψ' in (D_j, E_j, R_j) at some step $j \geq i$ such that $\Psi \stackrel{+}{\Longrightarrow} \Psi'$.

If one of the rules Orient_SD/NSD, Simplify, Compose_D_deco, Compose_R_deco, Delete, Compose, Subsume_deco, or Collapse applies to Ψ, then (D_i, E_i, R_i) must be different from $(D_\infty, E_\infty, R_\infty)$, since either $E_\infty \neq \emptyset$ or $D_\infty \cup R_\infty$ is not interreduced. So one of the rules of $\mathcal{OSC}$ applies.

If the transformation rule Deduce or Deduce_deco applies with a non-persisting rule then for some $j \geq i$, $(D_i, E_i, R_i) \vdash_{\mathcal{OSC}} \ldots \vdash_{\mathcal{OSC}} (D_j, E_j, R_j)$ where (D_j, E_j, R_j) does not contain this rule any more. If the transformation rule Deduce or Deduce_deco applies with persisting rules, then either the corresponding critical pair was already computed at some step $k \leq i$, in which case we can transform Ψ into Ψ' thanks to the reflection property (see Proposition 7.3.3). Remark that this justifies that we discarded the indices for the definition of $\Longrightarrow$, since otherwise we would get into trouble with the index of presentations: the $\Longrightarrow$-transformation for **Deduce** would have to consider proofs with elementary steps of different presentations.

Otherwise, by hypothesis there exists $k > i$ such that the computed critical pair is in $D_k \cup E_k$, since all terms in Ψ are in $\mathcal{T}$ and $\mathbf{UNIF_d}$ is complete for the unification problems in $CP(R, R)_{|\mathcal{T}}$, $CP(R, D)_{|\mathcal{T}}$, $CP(D, D)_{|\mathcal{T}}$ and $CP(D, R)_{|\mathcal{T}}$ as a consequence of Lemma 7.1.8. Remark, that since $E_\infty = \emptyset$ by hypothesis, then for some $j > k > i$, $(D_i, E_i, R_i) \vdash_{\mathcal{OSC}} \ldots \vdash_{\mathcal{OSC}} (D_j, E_j, R_j)$ where E_j does not contain the critical pair any more. For all these cases, since $\Longrightarrow$ reflects $\vdash_{\mathcal{OSC}}$, $\Psi \stackrel{+}{\Longrightarrow} \Psi'$.

Detect_Subsort $\frac{D,E\cup\{p:S\cup\{B\}=(x::A):S'\},R}{\bot,\bot,\bot}$ $x \in \mathcal{X}_\Diamond$ and $sort(x) = A$ and not $A \leq^{syn}_{\mathcal{S}_\Diamond} B$

Figure 7.3: Test for Equivalence of $\leq^{syn}_{\mathcal{S}}$ and $\leq^{sem}_{\mathcal{S}}$.

If either Peak without overlap, or Peak with variable overlap applies, then we know by by Theorem 7.1.15 that Ψ contains a peak that can be replaced by a rewrite proof $t' \overset{*}{\rightarrowtail} u \overset{*}{\leftarrowtail} t''$. Then $j = i$ and $\Psi \overset{+}{\Longrightarrow} \Psi'$. □

For a fair derivation, the resulting decorated presentation satisfies the property that any equational proof has a normal form which is a rewrite proof. We then get the main result of this section:

Theorem 7.4.3 *Let $\mathcal{P}_0$ be sort inheriting and $(D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ be a fair derivation. Then $\mathcal{P}_\infty$ is terminating, Church-Rosser, type complete and existentially complete on $ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. Moreover $Th(\mathcal{P}) = Form(\mathcal{P}_0) = Form(\mathcal{P}_\infty)$.*

Proof: $Th(\mathcal{P}) = Form(\mathcal{P}_0)$ is a consequence of Theorem 6.5.1. The termination property of $D_\infty \cup R_\infty$ is obvious since the test is incrementally processed w.r.t. a fixed reduction ordering $>_d$ for each rule added in $D_* \cup R_*$, thus in $D_\infty \cup R_\infty$. Furthermore, the derivation is $ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$-fair and any proof $t^{:\downarrow\emptyset} \overset{*}{\longleftrightarrow}_{D_0\cup E_0\cup R_0} t'^{:S'}$ has a rewrite proof in $\mathcal{P}_\infty$ by Proposition 7.4.2. This guarantees the Church-Rosser property, as well as type and existential completeness. Finally $Form(\mathcal{P}_0) = Form(\mathcal{P}_\infty)$ is a consequence of Proposition 7.2.3. □

It should be noticed that the presented rules for completion do not include rules for simplifying at the top occurrence decoration rules with decorated rewrite rules. This is solved by decoration critical pairs that add the simplified versions of the decoration rules without deleting their old versions. Hence, a lot of decoration rules may be generated during a fair derivation. However, including more simplification for decoration rules is yet an unsolved point.

5 Changing the Subsort Relation

The unification algorithm needed for the completion process assumes, of course, the subsort relation to stay static, but this may not be the case (see Example 5.1.3). The test for equality of $\leq^{syn}_{\mathcal{S}}$ and $\leq^{sem}_{\mathcal{S}}$ can be done using the **Detect_Subsort** rule shown in Figure 7.3, which can be added to $\mathcal{OSC}$.

The rule **Detect_Subsort** is semi-complete when it is used in $\mathcal{OSC}$ together with a fair strategy.

Proposition 7.5.1 *Let $\leq^{syn}_{\mathcal{S}}$ be a syntactic sort ordering used for the completion of $\mathcal{P}$, a sort inheriting presentation w.r.t. $\leq^{syn}_{\mathcal{S}}$. Let furthermore $(D_\mathcal{P}, E_\mathcal{P}, \emptyset) = (D_0, E_0, R_0) \vdash (D_1, E_1, R_1) \vdash \ldots$ be a derivation using $\mathcal{OSC}$ with a fair strategy if $\mathcal{P}_\infty \neq (\bot, \bot, \bot)$.*

Then, **Detect_Subsort** *applies iff $\leq^{syn}_{\mathcal{S}} \neq \leq^{sem}_{\mathcal{S}}$.*

Proof: First of all, if **Detect_Subsort** is applicable, then clearly $\leq^{syn}_{\mathcal{S}} \neq \leq^{sem}_{\mathcal{S}}$. Now, suppose $\leq^{syn}_{\mathcal{S}} \neq \leq^{sem}_{\mathcal{S}}$ and **Detect_Subsort** does not apply. Hence, there are sorts A, B, s.t. $A \leq^{sem}_{\mathcal{S}}$

B, but not $A \leq_{\mathcal{S}}^{syn} B$. Therefore Lemma 6.5.1 implies the existence of a corresponding proof $\Psi : x^{:\{A\}} \overset{*}{\leftrightarrow}_{\mathcal{P}_0} t^{:S\cup\{B'\}}$ with $B' \leq_{\mathcal{S}_\Diamond}^{syn} B$ for a variable x with $sort(x) = A$.

From fairness and termination of $\Longrightarrow$ (see Proposition 7.3.1), it follows now that there exists a rewrite proof form Ψ_k of Ψ after a finite number of steps k, since any peak must be reduced after a finite number of steps yielding a strictly smaller proof. Let $t'^{:S'}$ be the normal form of $x^{:\downarrow\emptyset} =_d x^{:\{A\}}$ in Ψ_k.

Assume $t' \notin \mathcal{X}_\Diamond$. Clearly, some $A' \leq_{\mathcal{S}_\Diamond}^{syn} A$ must be in S', since decoration and decorated rewriting can only increase the top decoration. Therefore, $\sigma = \{x^{:\{A\}} \mapsto t'^{:S'}\}$ is a decorated substitution. If $x \in \mathcal{V}ar(t'^{:S'})$, then $\sigma(x^{:\{A\}})$ is a strict subterm of $\sigma(t'^{:S'})$. Otherwise, if $x \notin \mathcal{V}ar(t'^{:S'})$, then $\sigma(x^{:\{A\}}) \cong_d \sigma(t'^{:S'})$. In both cases, we get a contradiction to the well-foundedness of $<_d$, since $\sigma(x^{:\{A\}})$ rewrites to a non-variable term that contains $\sigma(x^{:\{A\}})$. Hence, $t' \notin \mathcal{X}_\Diamond$ is impossible, not even for intermediate terms in Ψ_k.

Moreover, $t' = x$ must hold, because of $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ for any decorated rewrite rule $l \to r$ by definition. So, we can be sure that $\Psi_k : x^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_{R_k} x^{:T\cup\{B'\}}$ and any $(\phi_i : l_i^{:S_i} \to r_i^{:S'_i}) \in R_k$ used in Ψ_k must satisfy $(l_i)_{nd} = (r_i)_{nd} = x$ and $S_i \approx S'_i \approx \{A\}$ by the definition of decorated variables. But this is again in contradiction with the well-foundedness of $<_d$. Remark that decoration rewrite rules cannot be used in Ψ_k, because variable decoration rules are inapplicable due to an unsatisfiable condition.

Consequently, Ψ_k cannot exist and **Detect_Subsort** must be applicable. □

In the case where **Detect_Subsort** applies, we just proved that $\leq_{\mathcal{S}}^{syn} \subset \leq_{\mathcal{S}}^{sem}$ and the syntactical ordering on sorts should be extended before a new attempt of completion. But several situations may then occur, as in the following examples, where different kinds of extensions to the syntactical ordering are illustrated.

Example 7.5.2 *Let $\mathcal{S} = \{A, B, C, D\}$, $\leq_1$ be the initial syntactic subsort relation and $\leq_2$ be the one where the new extensions are added. We can add in $\leq_2$ a relation between :*

1. *incomparable sorts : If $\leq_1 = \emptyset$ and $\leq_2 = \{(A, B)\}$, then any solution of a unification problem of the form $(x :: A \cong_d^? y :: B)$ was incomplete. So we have to restart the critical pair computation.*

2. *comparable sorts : If $\leq_1 = \{(A, B)\}$ and $\leq_2 = \{(A, B), (B, A)\}$, then we added a cycle. To satisfy Assumption 5.1.6 again, we have to replace A or B in the last presentation by a unique representative sort, in order to continue the computations. This possibly makes some decorated rewrite rules trivial. We may need to re-orient some rules. If some condition of a decoration rewrite rule becomes unsatisfiable then this rule can of course be deleted.*

Chapitre 8

Checking Sort Inheritance

We are now left with the problem of checking the sort inheritance property of a decorated presentation. The idea is to characterise non sort inheritance on a set of decorated terms $\mathcal{T}$ by a property of the decoration rule set: D must contain a set of rules whose left-hand sides are unifiable and that produce the adjunction of sorts S without common subsort $C \leq^{syn}_{\mathcal{S}_\Diamond} S$. This characterisation is possible if D is confluent on $\mathcal{T}$ and any term of $\mathcal{T}$ is reachable by D only. A corresponding test can be realized by a rule **Detect_NonSI** shown in Figure 8.1.

Given a set of decoration rules D, a *typing proof* of a term $t^{:T}$ is a decoration rewriting proof of the form $t^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_D t^{:T}$ and terms with a typing proof are simply called *typable.* Proofs in which all terms are typable are therefore called typable, too.

The first step is to prove that our test on decoration rules is sufficient to ensure sort inheritance on $reach_D(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$, i.e. all typable terms, if D is confluent and terminating. The second step is the observation that the proofs constructed by Theorem 6.5.1 are typable.

This motivates the search of a well-founded proof reduction giving rewrite proofs as normal form and maintaining the typability property, which are thus called *typing conservative.* We thus get sort inheritance on the set of valid decorated terms $Valid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$: If $t^{:S}$ is valid and a counterexample for sort inheritance, then there is a proof Ψ in $\mathcal{P}_0$, s.t.:

$$\Psi \ : \ (t^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_{\mathcal{P}_0} t_1{}^{:S_1} \leftarrow\!\!\!\!\rightarrow^*_{\mathcal{P}_0} t^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_{\mathcal{P}_0} \ldots \leftarrow\!\!\!\!\rightarrow^*_{\mathcal{P}_0} t_n{}^{:S_n} \leftarrow\!\!\!\!\rightarrow^*_{\mathcal{P}_0} t^{:\downarrow\emptyset})$$

with $S \subseteq \bigcup_{i\in[1..n]} S_i$. Therefore, there is a proof in normal form $\Psi_{nf} \ : \ (t^{:\downarrow\emptyset} \rightarrowtail^*_{\mathcal{P}_k} t'^{:S'})$ under the proof reduction with $S \subseteq \bigcup_{i\in[1..n]} S_i \subseteq S'$, i.e. $t'^{:S'}$ is also a valid counterexample for sort inheritance, provided $\mathcal{P}_k$ is valid. Hence, by typing conservation, $t'^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_{D_k} t'^{:S'}$ and it is sufficient to test sort inheritance on all typable terms in $\mathcal{P}_k$ in order to find a counterexample.

The real difficulty is to find a typing conservative proof transformation. Unfortunately, when dealing with non-linear *decoration* rules, the proof transformation $\Longrightarrow$ is too general for this propagation, since the simplification of terms can be done at arbitrary occurrences. This means that the reduction by $\rightarrowtail_R$ could destroy the typability of a term, i.e. its reachability by $\rightarrowtail_D$.

Detect_NonSI

$$\frac{D \cup_{i\in[1..n]} \{(p_i{}^{:s} \rightarrow p_i{}^{:s\cup S_i} \text{ if } S_i \not\subseteq s)\}, E, R}{\bot, \bot, \bot} \quad \begin{array}{l}\text{if } (\exists \psi, \forall i \in [1..n-1], \psi(p_i{}^{:\emptyset}) = \psi(p_{i+1}{}^{:\emptyset})) \\ \text{and } (\exists S \subseteq \bigcup_{i\in[1..n]} S_i, \nexists C \leq^{syn}_{\mathcal{S}_\Diamond} S)\end{array}$$

Figure 8.1: Sort Inheritance Test Rule.

Let us give a simple example of the problem that occurs with non-linearity.

Example 8.0.3 *Let $\mathcal{P} = (D, \emptyset, R)$ be the following decorated presentation:*

$$\begin{aligned} D &= \{a^{:s} \to a^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s, \\ &\quad b^{:s} \to b^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s, \\ &\quad c^{:s} \to c^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s, \\ &\quad f(x^{:\{A\}}, x^{:\{A\}})^{:s} \to f(x^{:\{A\}}, x^{:\{A\}})^{:s\cup\{C\}} \text{ if } \{C\} \not\subseteq s\}, \\ R &= \{a^{:\{A\}} \to b^{:\{A\}} \\ &\quad a^{:\{A\}} \to c^{:\{A\}}\} \end{aligned}$$

Then we have the following reductions:

$$\begin{array}{ccc} f(a^{:\emptyset}, a^{:\emptyset})^{:\emptyset} & \xrightarrow{*}_D & f(a^{:\{A\}}, a^{:\{A\}})^{:\emptyset} \underset{D}{\not\to} f(a^{:\{A\}}, a^{:\{A\}})^{:C} \\ & & \downarrow_R \qquad\qquad \downarrow_R \\ f(b^{:\emptyset}, a^{:\emptyset})^{:\emptyset} & \xrightarrow{*}_D & f(b^{:\{A\}}, a^{:\{A\}})^{:\emptyset} \underset{D}{\not\to} f(b^{:\{A\}}, a^{:\{A\}})^{:C} \\ & & \downarrow_R \qquad\qquad \downarrow_R \\ f(b^{:\emptyset}, c^{:\emptyset})^{:\emptyset} & \xrightarrow{*}_D & f(b^{:\{A\}}, c^{:\{A\}})^{:\emptyset} \underset{D}{\not\to} f(b^{:\{A\}}, c^{:\{A\}})^{:C} \end{array}$$

This shows that a reduction by R may destroy the reducibility by D. Hence, $f(b^{:\{A\}}, c^{:\{A\}})^{:C}$ is no more typable, although it is irreducible.

1 Testing Sort Inheritance On D-Closed Sets

The main goal of this section is the construction of a test for sort inheritance on the set of decorated terms $\mathcal{T}$ reachable from $\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset}$ with decoration rules only. The notion of D-closure of a set of terms is needed to define this set.

Definition 8.1.1 *Let (D, E, R) be a decorated presentation. A set $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ is* D-closed *if for all $t \in \mathcal{T}$:*

1. $t^{:\downarrow\emptyset} \in \mathcal{T}$,
2. $t \xleftrightarrow{*}_D t'$ *implies* $t' \in \mathcal{T}$ *and,*
3. $t \in \mathcal{T}$ *implies* $\forall \omega \in \mathcal{O}cc(t) : t_{|\omega} \in \mathcal{T}$.

The D-closure $reach_D(\mathcal{T}^{:\downarrow\emptyset})$ of a set $\mathcal{T}^{:\downarrow\emptyset}$ of terms with empty decorations is therefore the set:

$$\{t \mid \exists t' \in \mathcal{T}^{:\downarrow\emptyset} : t' \xleftrightarrow{*}_D t'' \text{ and } \exists \omega \in \mathcal{O}cc(t'') : t =_d t''_{|\omega}\}.$$

Note that $strict\_subterm\_set(reach_D(\mathcal{T}^{:\downarrow\emptyset})) \subseteq reach_D(strict\_subterm\_set(\mathcal{T}^{:\downarrow\emptyset}))$ is an immediate consequence of the fact that decoration rewrite rules do not change the structure of a term and the conditions 1 and 3.

Lemma 8.1.2 *Let $\mathcal{P} = (D, E, R)$ be a $strict\_subterm\_set(\mathcal{T})$-sort inheriting decorated presentation, such that D is valid and confluent on $\mathcal{T}$ with $\mathcal{T} = reach_D(\mathcal{T}^{:\downarrow\emptyset})$. Then $\mathcal{P}$ is $\mathcal{T}$-sort inheriting w.r.t. $\leq_{\mathcal{S}}^{syn}$ iff for all $t \in \mathcal{T}$: $t^{:\downarrow\emptyset} \xrightarrow{*}_D t^{:T}$ implies that there is a sort $C \in \mathcal{S}$ with $\forall \langle A \ldots \rangle \in T, C \leq_{\mathcal{S}}^{syn} A$.*

Proof: $\Rightarrow$: This is obvious because of $\mathcal{T} = reach_D(\mathcal{T}^{:\downarrow\emptyset})$.

$\Leftarrow$: Since $\mathcal{P}$ is already *strict_subterm_set*($\mathcal{T}$)-sort inheriting, we only have to check top decorations. We prove that no $t'^{:T'} \in \mathcal{T}$ can be a counterexample for $\mathcal{T}$-sort inheritance w.r.t. $\leq_S^{syn}$. If $t'^{:T'} \in \mathcal{T}$, then there must be a term $t^{:\downarrow\emptyset} \in \mathcal{T}$, s.t. $t^{:\downarrow\emptyset} \leftrightarrow\!\!\!\!\rightarrow_D^* t'^{:T'}$. Since D is confluent, $t^{:\downarrow\emptyset}$ has a unique (modulo sort inheritance) D-normal form $t'''^{:T''}$ and furthermore $t'^{:T'} \rightarrow_D^* t'''^{:T''}$.

Since there is a $C \in \mathcal{S}$, s.t. $\forall\langle A \ldots\rangle \in T''$, $C \leq_S^{syn} A$, we also know that $\forall\langle A \ldots\rangle \in T'$, $C \leq_S^{syn} A$, because of $T' \subsetneq T''$. Consequently, $t'^{:T'}$ indeed cannot be a counterexample for $\mathcal{T}$-sort inheritance w.r.t. $\leq_S^{syn}$. □

In order to build a more syntactical test, a saturation process on decoration rules is designed. Let **Deduce_DD** stand for the rule **Deduce_deco** applied to decoration rules only and $\underline{Deduce\_DD}$ be the corresponding proof reduction rules in $\Longrightarrow$, including the rules for peaks.

Let furthermore $\mathcal{D}ed$ be the set of rules consisting of **Deduce_DD**, **Compose_D_deco** and **Subsume_deco**. $\underline{\mathcal{D}ed}$ is the corresponding set of proof transformation rules of $\Longrightarrow$. Analogously to Definition 7.4.1, we can define $\mathcal{T}$-fairness w.r.t. decoration rules, where $\Longrightarrow$ is restricted to the rules in $\underline{\mathcal{D}ed}$ plus the rules for overlaps at variable positions and peaks without overlap. Let $\Longrightarrow_{Ded}$ denote this restricted proof transformation.

Definition 8.1.3 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$. A derivation $(D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ is* $\mathcal{T}$-fair w.r.t. decoration rules *if whenever Ψ is a proof in (D_i, E_i, R_i), that uses only terms in $\mathcal{T}$ and is reducible by $\Longrightarrow_{Ded}$, then there is a proof Ψ' in (D_j, E_j, R_j) at some step $j \geq i$ such that $\Psi \stackrel{+}{\Longrightarrow} \Psi'$.*

Remark that using $\Psi \stackrel{+}{\Longrightarrow}_{Ded} \Psi'$ instead of $\Psi \stackrel{+}{\Longrightarrow} \Psi'$ in the above definition would lead to problems when we simplify using decorated rewrite rules, since then, $\Psi \stackrel{+}{\Longrightarrow}_{Ded} \Psi'$ can no more be guaranteed for all Ψ reducible by $\Longrightarrow_{Ded}$. Note furthermore that if a derivation is $\mathcal{T}$-fair, then it is also $\mathcal{T}$-fair w.r.t. decoration rules. The reverse direction does, however, not always hold: if a derivation is $\mathcal{T}$-fair w.r.t. decoration rules, then there might still be unsolved critical pairs with rules in R_∞.

Clearly, the proof part of Proposition 7.4.2 dealing with these rules proves that $\mathcal{T}$-fairness for decoration rules is implied by the condition that all critical pairs of D_∞ are in D_*, provided we use $\mathbf{UNIF_d}$ as unification algorithm and the latter is *strict_subterm_set*($\mathcal{T}$)-complete. Since we did not change the rules themselves, the property of reflection of $\vdash_{Ded}$ by $\underline{\mathcal{D}ed}$ is still valid.

Corollary 8.1.4 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and assume that* $\mathbf{UNIF_d}$ *(see Figures 5.4 and 5.5) is a strict_subterm_set($\mathcal{T}$)-complete unification algorithm.*

A derivation of the shape $(D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ using $\mathbf{UNIF_d}$ *for the calculation of critical pairs is $\mathcal{T}$-fair w.r.t. decoration rules if all critical pairs of D_∞ are in D_*.*

The next proposition is an analogue to Theorem 7.4.3. Remark that the derivation is not necessarily infinite nor all rules of $\mathcal{OSC}$ must be applied whenever possible.

Lemma 8.1.5 *Let $(D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ be a derivation, $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ be a D_∞-closed set of decorated terms and let the derivation be $\mathcal{T}$-fair w.r.t. decoration rules. Then D_∞ is $\mathcal{T}$-confluent.*

Proof: Remark that the $\mathcal{T}$-fairness w.r.t. decoration rules implies that for every proof using $\overset{*}{\rightarrowtail}_{D_\infty}$ with terms in $\mathcal{T}$ only has a normal form (in D_∞ and, by D_∞-closure of $\mathcal{T}$, with terms in $\mathcal{T}$ only), that is irreducible w.r.t. $\Longrightarrow_{Ded}$. It follows immediately from the proof reduction rules Deduce_deco, peak without overlap, peak with variable overlap that such a proof does not contain any peak and hence D_∞ is $\mathcal{T}$-confluent. □

Specific proofs with decoration rules, namely bottom-up decoration proofs, play a fundamental role in the test of sort inheritance.

Definition 8.1.6 *Let (D, E, R) be a decorated presentation. Then:*

$$\Psi : t_0 \rightarrowtail_D^{\omega_1} t_1 \ldots \rightarrowtail_D^{\omega_n} t_n$$

is a bottom-up decoration proof *iff for all $i, j \in [1..n]$, we have $i < j \Rightarrow \omega_i \not< \omega_j$.*

Lemma 8.1.7 *Let $\mathcal{T} \subseteq Valid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ with $\mathcal{T} = reach_D(\mathcal{T}^{:\downarrow\emptyset})$ and D be $\mathcal{T}$-confluent. Then there exists for any decoration proof:*

$$\Psi : t_0{}^{:S_0} \rightarrowtail_D^{\omega_1} t_1{}^{:S_1} \ldots \rightarrowtail_D^{\omega_n} t_n{}^{:S_n},$$

where $t_i{}^{:S_i} \in \mathcal{T}$ for all $i \in [0..n]$ and $t_n{}^{:S_n}$ is irreducible in D, a bottom-up decoration rewrite proof for $t_0{}^{:S_0} \overset{*}{\rightarrowtail}_D t_n{}^{:S_n}$.

Proof: First, we need a partial proof ordering respecting the occurrence of rule application, such that bottom-up right-to-left decoration enrichment is less complex than any other strategy.

Let $\Psi = (t_i{}^{:S_i} \rightarrowtail_D^{\omega_i,\phi_i} t_{i+1}{}^{:S_{i+1}})_{i\in[0..n]}$, $\Psi' = (t'_i{}^{:S'_i} \rightarrowtail_D^{\omega'_i,\phi'_i} t'_{i+1}{}^{:S'_{i+1}})_{i\in[0..m]}$ be two decoration rewriting proofs. Then:

$$\Psi <_o \Psi' \text{ iff } \exists k \in [0..min(m,n)] : \forall i \in [0..k-1] : \omega'_i \leq_{lex} \omega_i \text{ and } \omega'_k <_{lex} \omega_k.$$

Remark that given a fixed D, $<_o$ is well-founded on all proofs starting with the same $t_0{}^{:S_0}$. W.l.o.g. we can consider a minimal proof of $t_0 \overset{*}{\rightarrowtail}_D t_n$ w.r.t. the proof ordering $<_o$. If the proof consists of a single rule application, we are trivially done.

Otherwise, let us assume that we have:

$$\begin{array}{ll} & t_m{}^{:S_m}[\sigma_m(u_m{}^{:U_m})]_{\omega_m}[\sigma_{m+1}(u_{m+1}{}^{:U_{m+1}})]_{\omega_{m+1}} \\ \omega_m,\phi_m,\sigma_m & t_m{}^{:S_m}[\sigma_m(u_m{}^{:U_m}){}^{:U_m\cup T_m})]_{\omega_m}[\sigma_{m+1}(u_{m+1}{}^{:U_{m+1}})]_{\omega_{m+1}} \\ \omega_{m+1},\phi_{m+1},\sigma_{m+1} & t_m{}^{:S_m}[\sigma_m(u_m{}^{:U_m}){}^{:U_m\cup T_m})]_{\omega_m}[\sigma_{m+1}(u_{m+1}{}^{:U_{m+1}}){}^{:U_{m+1}\cup T_{m+1}}]_{\omega_{m+1}} \end{array}$$

with $\omega_m < \omega_{m+1}$, i.e. $\omega_{m+1} = \omega_m.\nu$. We can assume that these two rule applications follow each other, since any rewrite step at an incomparable occurrence in between could be swapped with one of the two, resulting in a smaller proof.

Obviously, we can apply ϕ_{m+1} to $t_m{}^{:S_m}[\sigma_m(u_m{}^{:U_m})]_{\omega_m}[\sigma_{m+1}(u_{m+1}{}^{:U_{m+1}})]_{\omega_{m+1}}$. The resulting proof is strictly smaller w.r.t. $<_o$ and the $\mathcal{T}$-confluence of D guarantees that $t_n{}^{:S_n}$ is still reached. Thus, there must be an equivalent bottom-up decoration proof. □

Lemma 8.1.8 *Let $\mathcal{T} \subseteq Valid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ with $\mathcal{T} = reach_D(\mathcal{T}^{:\downarrow\emptyset})$ and $\mathcal{P}' = (D, E, R)$ be a strict_subterm_set($\mathcal{T}$)-sort inheriting decorated presentation obtained from $(D_\mathcal{P}, E_\mathcal{P}, \emptyset)$ using OSC, such that D is $\mathcal{T}$-confluent. Then, the* **Detect_NonSI***-rule succeeds on D iff $\mathcal{P}'$ is not $\mathcal{T}$-sort inheriting.*

Proof: $\Leftarrow$: Lemma 8.1.2 implies, that there exists a $t^{:\downarrow\emptyset} \in \mathcal{T}$ with $\psi : t^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_D t^{:T}$, s.t. there is no sort $C \in \mathcal{S}$ with $\forall\langle A, \ldots\rangle \in T, C \leq_{\mathcal{S}}^{syn} A$. Let w.l.o.g. $t^{:T}$ be D-irreducible, i.e. T is maximal, too.

Hence, by Lemma 8.1.7 there must be a set of decorated substitutions σ_i and decoration rules $(\phi_i : p_i^{:s} \rightarrow p_i^{:s \cup S_i}$ if $S_i \not\subseteq s)$ for $i \in [1..n]$, s.t. $\sigma_i(p_i^{:\emptyset}) =_{nd} t^{:\downarrow\emptyset}$, $T \approx \cup_{i\in[1..n]} S_i$ and:

$$t^{:\downarrow\emptyset} \overset{*}{\rightarrowtail}_D t_0^{:\emptyset} \overset{\Lambda,\phi_1,\sigma_1}{\rightarrowtail}_D t_1^{:S_1} \ldots \overset{\Lambda,\phi_n,\sigma_n}{\rightarrowtail}_D t_n^{:\cup_{i\in[1..n]} S_i}.$$

Note that for all $i \in [1..n]$, $p_i \notin \mathcal{X}_\Diamond$, since the only way to introduce decoration rules for terms with a new structure is **Orient_NSD**. But the condition of **Orient_NSD** inhibits that variable decoration rules are created. Consequently, variable decoration rules are never created and t cannot be a variable, too.

Note furthermore that no application of the rules at Λ changes any decoration of strict subterms. Therefore we have $t_i^{:\emptyset} \cong_d t_{i+1}^{:\emptyset}$ for $i \in [1..n-1]$, giving us a $\mathcal{T}$-unifier of $p_i^{:\emptyset}$ and $p_{i+1}^{:\emptyset}$. Remark that $t_i^{:\emptyset}$ must be in $\mathcal{T}$ by the D-closure of $\mathcal{T}$. From Lemma 7.1.8 now follows that $\mathbf{UNIF_d}$ must succeed, since it is assumed to be *strict_subterm_set*$(\mathcal{T})$-complete and for all $i \in [1..n-1]$, $p_i^{:\emptyset} \cong_d^? p_{i+1}^{:\emptyset}$ is a unification problem in $CP(D,D)_{|\mathcal{T}}$, because of $\forall i \in [1..n], p_i \notin \mathcal{X}_\Diamond$, i.e. the conditions for the application of **Detect_NonSI** are fulfilled.

$\Rightarrow$: If **Detect_NonSI** is applicable, then take $\psi(p_1^{:\emptyset})^{:\cup_{i\in[1..n]} S_i}$ as counterexample for sort inheritance. □

Finally, we show how to achieve the $reach_D(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$-fairness for decoration rules with the **Deduce_DD** rule.

Corollary 8.1.9 *Let* $(D_\mathcal{P}, E_\mathcal{P}, \emptyset) = (D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ *be a derivation, s.t. all critical pairs of* D_∞ *are in* D_*. *Let furthermore* $\mathcal{T}^\infty$ *stand for* $reach_{D_\infty}(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$. *If* **Detect_NonSI** *cannot be applied, then:*

1. (D, E, R) *is* $\mathcal{T}^\infty$*-sort inheriting w.r.t.* $\leq_{\mathcal{S}}^{syn}$,
2. *the derivation is* $\mathcal{T}^\infty$*-fair for decoration rules and*
3. D *is* $\mathcal{T}^\infty$*-confluent.*

Proof: Let us define $\mathcal{T}^n = reach_{D_\infty}(\{t^{:\downarrow\emptyset} \mid |t| \leq n\})$ for $n \geq 0$, which implies:

$$\mathcal{T}^\infty = \bigcup_{n\geq 0} \mathcal{T}^n$$

The proof is by induction over n. We will look at the superposition of left-hand sides of decoration rewrite rules on terms in $\mathcal{T}^n$. Note that this does not only mean that these left-hand sides have to be in $\mathcal{T}^n$, but also their instances. Furthermore, if one left-hand side is in $\mathcal{T}^m$ with $m > n$, then it cannot yield an instance in $\mathcal{T}^n$, since instantiation cannot delete function symbols.

In the case $n = 0$, we have $\mathcal{T}^0$-fairness by Corollary 8.1.4, since all critical pairs of D_∞ are in D_*, *strict_subterm_set*$(\mathcal{T}^0)$ is empty and $\mathbf{UNIF_d}$ is therefore *strict_subterm_set*$(\mathcal{T}^0)$-complete. Lemmas 8.1.5 and 8.1.8 now give the $\mathcal{T}^0$-confluence of D_∞ and the $\mathcal{T}^0$-sort

inheritance, respectively. For $n > 0$ we have $strict\_subterm\_set(\mathcal{T}^n) \subseteq \mathcal{T}^{n-1}$ and therefore the induction hypothesis gives us with the $\mathcal{T}^{n-1}$-sort inheritance w.r.t. $\leq_{\mathcal{S}}^{syn}$ also the $strict\_subterm\_set(\mathcal{T}^n)$-completeness of $\mathbf{UNIF_d}$ (by Proposition 5.4.10), implying as before $\mathcal{T}^n$-confluence of D_∞ and $\mathcal{T}^n$-sort inheritance w.r.t. $\leq_{\mathcal{S}}^{syn}$.

Consequently, (D, E, R) is $\mathcal{T}^\infty$-sort inheriting w.r.t. $\leq_{\mathcal{S}}^{syn}$ and therefore it follows once more that $\mathbf{UNIF_d}$ is $\mathcal{T}^\infty$-complete, the derivation is $\mathcal{T}^\infty$-fair for decoration rules and D_∞ is $\mathcal{T}^\infty$-confluent. □

Since we now have a sufficient test for $reach_{D_\infty}(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$-sort inheritance w.r.t. $\leq_{\mathcal{S}}^{syn}$, the next step is to prove that the test is also sufficient test for $Valid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$-sort inheritance. In order to do that, the saturation process must incorporate equalities and rewrite rules.

2 Sort Inheritance on Valid Decorated Terms

The general problem in the following will be the test of sort inheritance over all valid terms. Clearly, the validity of a term depends on the provable equalities in the current presentation $\mathcal{P}$, which are decidable when we have a confluent decorated term rewriting system. This can only be achieved with a complete unification algorithm. But testing the completeness for all valid terms is exactly the problem we want to solve.

In order to break this interdependence, we try to achieve confluence with additional restrictions. Several possibilities are considered. The first and easiest one is the restriction of the allowed term declarations. However, we loose a lot of expressiveness in G-algebra when this part of the language is weakened. Another possibility is the design of another proof transformation, that normalizes proofs but keeps the typability property of terms. This leads to sophisticated completion strategies. Therefore, the expressiveness of term declarations can be expensive in practice and a compromise between complexity of completion and unrestricted term declarations is necessary.

The underlying plan for achieving confluence and testing sort inheritance will be the construction of a proof transformation that preserves the *typability* of all decorated terms in a proof, i.e. if for all decorated terms $t^{:S}$ in the proof to be transformed, $t^{:\downarrow\emptyset} \leftrightarrow\!\!\!\!\rightarrow_D^* t^{:S}$ holds, then this is also true for all terms in the transformed proof. A closer look at the proof of Theorem 6.5.1, particularily property (H_2), reveals the following Lemma.

Lemma 8.2.1 *Let* $\Psi : t_0{}^{:\downarrow\emptyset} \leftrightarrow\!\!\!\!\rightarrow_{D_0 \cup E_0} t_1{}^{:S_1} \leftrightarrow\!\!\!\!\rightarrow_{D_0 \cup E_0} \ldots t_n{}^{:S_n}$ *be the result of the proof transformation from G-algebra proofs to proofs in* $(D_\mathcal{P}, E_\mathcal{P}, \emptyset)$, *described in the proof of Theorem 6.5.1. Then*

$$\forall i \in [1..n] : t_i{}^{:\downarrow\emptyset} \rightarrow_{D_0}^* t_i{}^{:S_i}.$$

Using this observation, we can therefore test sort inheritance on all terms in the current proofs (using the results of Section 8.1) leading to the completeness of $\mathbf{UNIF_d}$ for critical pairs calculation.

We present and discuss three alternatives in the following, after proving typing proof conservation of our matching and unification algorithms. The first proposition is standard decorated rewriting and completion using flat and linear term declarations, also called function declarations. Dropping the linearity condition forces us to reduce terms in parallel, s.t. all identical subterms at the same depth are reduced simultaneously, and to prohibit decoration rewrite rule

simplification via decorated rewrite rules. This *layer rewriting* extends the set of critical pairs, but does not lead to further restrictions on the completion strategy, except that we do not allow for simplification of decoration rewrite rules by decorated rewrite rules. Last but not least, we investigate term declarations without any structural conditions. Then the proof of typability of terms in proofs is possible thanks to maximally subterm sharing rewriting, where all identical redexes have to be reduced at the same time. The principal drawback of this generality is a strong strategy for the completion procedure.

Finally, the three approaches are briefly compared and conservative extensions of the initial presentation $\mathcal{P}$ are given for the case when **Detect_NonSI** applies.

2.1 Subterm Conservation and Typing Proofs

We show here a preliminary property of typability of terms in a match or a unifier of typable terms. Remember that σ_{nf} denotes the tree-solved form corresponding to a solution σ in dag-solved form. The notation $C[\sigma(\Upsilon)]$ is used to denote the proof obtained from the proof Υ by instantiating it with the substitution σ and putting the result into a term context $C[]$. Concatenation of proofs is denoted by $+$. For any typing proof Υ, $\Upsilon_{|\omega}$ is obtained by considering subterms at position ω for all terms in Υ and restricting to rule applications on these subterms. These algebraic notations are consistent with [Bachmair, 1991]. Let us illustrate them with a simple example:

Example 8.2.2 *Let* $\Upsilon : f(x^{:\{A\}})^{:\emptyset} \rightarrowtail_D f(x^{:\{A\}})^{:\{C\}}$ *and* $\Upsilon' : a^{:\emptyset} \rightarrowtail_D a^{:\{A,B\}}$ *be two typing proofs and* $\sigma = \{x^{:\{A\}} \mapsto a^{:\{A,B\}}\}$. *Then, we get:*

$$g(c)^{:\downarrow\emptyset}[\Upsilon]_1 : \quad g(f(x^{:\{A\}})^{:\emptyset})^{:\emptyset} \rightarrowtail_D g(f(x^{:\{A\}})^{:\{C\}})^{:\emptyset}$$

$$\sigma(\Upsilon) : \quad f(a^{:\{A,B\}})^{:\emptyset} \rightarrowtail_D f(a^{:\{A,B\}})^{:\{C\}}$$

$$g(c)^{:\downarrow\emptyset}[\sigma(\Upsilon)]_1 : \quad g(f(a^{:\{A,B\}})^{:\emptyset})^{:\emptyset} \rightarrowtail_D g(f(a^{:\{A,B\}})^{:\{C\}})^{:\emptyset}$$

$$f(x^{:\{A\}})^{:\emptyset}[\Upsilon']_1 + \sigma(\Upsilon) : \quad f(a^{:\emptyset})^{:\emptyset} \rightarrowtail_D f(a^{:\{A,B\}})^{:\emptyset} \rightarrowtail_D f(a^{:\{A,B\}})^{:\{C\}}$$

$$(f(x^{:\{A\}})^{:\emptyset}[\Upsilon']_1 + \sigma(\Upsilon))_{|_1} = \Upsilon'$$

Remark that the example still works if $\rightarrowtail_D$ *is replaced by* $\leftarrowtail\!\!\rightarrowtail_D$.

Lemma 8.2.3 *Let* $\sigma = \{x_i^{:S_i} \mapsto u_i^{:U_i}\}_{i\in I}$ *be a decorated substitution in dag-solved form, s.t.* $\forall i \in I : u_i^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D u_i^{:U_i}\ (H(u_i^{:U_i}))$ *or* $u_i^{:\downarrow\emptyset} \stackrel{*}{\leftarrowtail\!\!\rightarrowtail}_D u_i^{:U_i}\ (H'(u_i^{:U_i}))$, *respectively. Then, for all* $z \in \mathcal{D}om(\sigma)$:

$$\sigma_{nf}(z)^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D \sigma_{nf}(z) \qquad (H(z))$$

(and $\sigma_{nf}(z)^{:\downarrow\emptyset} \stackrel{*}{\leftarrowtail\!\!\rightarrowtail}_D \sigma_{nf}(z)$), respectively. $(H'(z))$

Proof: The proof is by induction over the occur check ordering of the variables in $\mathcal{D}om(\sigma)$. Let therefore $oc(z)$ be the minimal $k \geq 1$, s.t. $\sigma^k(z) =_d \sigma^{k+1}(z)$. Note that $oc(z) = 0$ is obvious, since this implies that $z =_d \sigma(z) =_d \sigma_{nf}(z)$.

Else let $oc(z) = k > 1$, $\sigma(z) = u_k^{:U_k}$, $\mathcal{V}ar(\sigma(z)) = \{z_j \mid j \in J\} = V$ and O_j denote the set of occurrences of z_j in $\sigma(z)$ for all $j \in J$. We can suppose $H(z_j)$ to be given for all

$j \in J$ by the induction hypothesis. Let us denote by Υ^j the corresponding typing proof of $\sigma_{nf}(z_j)$. Then, we know that the following proof exists:

$$\sigma_{nf}(z)^{:\downarrow\emptyset}([\Upsilon^j]_{O_j})_{j\in J} \ : \ \sigma_{nf}(z)^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D \sigma_{nf}(z)^{:\downarrow\emptyset}([\sigma_{nf}(z_j)]_{O_j})_{j\in J}.$$

Now, let Υ denote the typing proof $u_k^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D u_k^{:U_k}$ given by $H(u_k^{:U_k})$, Then, substitutivity of decoration rewriting gives us:

$$(\sigma_{nf})_{|_V}(\Upsilon) \ : \ \sigma_{nf}(z)^{:\downarrow\emptyset}([\sigma_{nf}(z_j)]_{O_j})_{j\in J} \stackrel{*}{\rightarrowtail}_D \sigma_{nf}(z).$$

Concatenating $\sigma_{nf}(z)^{:\downarrow\emptyset}([\Upsilon^j]_{O_j})_{j\in J}$ with $(\sigma_{nf})_{|_V}(\Upsilon)$ now results in $H(z)$. The case $H'(z)$ is similar. □

Proposition 8.2.4 *Let $\mathcal{M}$ be a variable-disjoint matching problem, s.t. for all $t \in terms(\mathcal{M})$, $t^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D t$ and $\sigma = \{x_i^{:S_i} \mapsto u_i^{:U_i}\}_{i\in I}$ the $\mathcal{D}$-matcher calculated by* $\mathbf{MATCH_d}$. *Then:*

$$\forall i \in I \ : \ \sigma_{nf}(x_i^{:S_i})^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D \sigma_{nf}(x_i^{:S_i})$$

Proof: Clearly, the strict subterm conservation implies that the $u_i^{:U_i}$ are subterms of some $t \in terms(\mathcal{M})$. Now Lemma 8.2.3 guarantees that σ_{nf} has the desired property. □

Proposition 8.2.5 *Let $\mathcal{U}$ be a unification problem, s.t. for all $t \in terms(\mathcal{U})$, $t^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D t$ or $t^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D t$, respectively, and $\sigma = \{x_i^{:S_i} \mapsto u_i^{:U_i}\}_{i\in I}$ the $\mathcal{D}$-unifier calculated by* $\mathbf{UNIF_d}$ *in dag-solved form. Then for all $t \in terms(\mathcal{U})$:*

$$\sigma_{nf}(t)^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D \sigma_{nf}(t)) \qquad (H(t))$$

$$(\text{and } \sigma_{nf}(t)^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D \sigma_{nf}(t))), \text{ respectively.} \qquad (H'(t))$$

Proof: We know that decorated unification is subterm conservative, i.e. for all $i \in I$, $u_i^{:U_i}$ is a specialisation of a subterm of some $t \in terms(\mathcal{U})$. Let τ be the specialisation for some $u_i^{:U_i}$ with $i \in I$, i.e. for all $x^{:S_x} \in \mathcal{D}om(\tau)$, we have $\tau(x^{:S_x}) =_d z^{:S}$ for some $z \in \mathcal{X}_\Diamond$ and $sort(z) \approx S$. Clearly, $z^{:\downarrow\emptyset} =_d z^{:\{C\}} \stackrel{*}{\rightarrowtail}_D z^{:S}$, since $z^{:\{C\}} \cong_d z^{:S}$.

Hence, if $u_i^{:U_i} =_d \tau(t_{i|_{\omega_i}})$ for some $t_i \in terms(\mathcal{U})$ and Υ_i is the typing proof of t_i, then $\forall i \in I$:

$$\tau(\Upsilon_i)_{|_{\omega_i}} \ : \ (u_i^{:U_i})^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D u_i^{:U_i},$$

by substitutivity of decoration rewriting, which makes Lemma 8.2.3 applicable, giving us $H(t)$ for all $t \in terms(\mathcal{U})$ once more by substitutivity. The proof of $H'(t)$ is similar. □

2.2 Flat and Linear Term Declarations

Restricting to flat and linear term declarations, it is possible to prove that typability of terms is preserved by decorated rewriting, as defined in Chapter 6.

Lemma 8.2.6 *Let D be a set of decoration rewrite rules, that contains flat and linear decoration rules only and R be a set of decorated rewrite rules, s.t. all terms $p^{:S}$ in R satisfy $p^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D p^{:S}$. Let furthermore $t^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D t^{:T}$ and $t^{:T} \ {}_{\omega,\sigma,\phi}\!\!\rightarrow_R t'^{:T'}$, where ϕ is $l^{:S_l} \rightarrow r^{:S_r}$.*

Then $t'^{:\downarrow\emptyset} \stackrel{*}{\longleftrightarrow}_D t'^{:T'}$.

Proof: We can construct the new typing proof by the strict subterm conservation of decorated matching (see Proposition 8.2.4), the substitutivity of the typing proof for the right hand side of the rule used for simplification and at any occurrence incomparable or above the reduction, we can use the old proof unchanged. Formally, if $\Psi : t^{:\downarrow\emptyset} \leftarrow\!\!*\!\!\rightarrow_D t^{:T}$ is a typing proof for the term to be reduced, $\Psi' : r^{:\downarrow\emptyset} \leftarrow\!\!*\!\!\rightarrow_D r^{:S_r}$ and $t^{:T}[\sigma(r^{:S_r})]_\omega$ is the reduced term, then:

$$t^{:\downarrow\emptyset}[\Psi_{|_{\omega.\mathcal{VO}cc(l)}}]_{\omega.\mathcal{VO}cc(r)} + t^{:\downarrow\emptyset}[\sigma(\Psi')]_\omega : (t^{:T}[\sigma(r^{:S_r})]_\omega)^{:\downarrow\emptyset} \leftarrow\!\!*\!\!\rightarrow_D (t^{:T})^{:\downarrow\emptyset}[\sigma(r^{:S_r})]_\omega$$

where we can append all steps of Ψ that apply strictly above ω. □

Actually, the property of being flat and linear is only required for D_∞.

Lemma 8.2.7 *Let $(D_\mathcal{P}, E_\mathcal{P}, \emptyset) = (D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ be a derivation using $\mathcal{OSC}$, s.t. all decorated rewrite rules ϕ used with* **Compose_R_deco** *are* decoration preserving, *i.e. of the form $l^{:S} \to r^{:S}$.*

Then for all $k \geq 0$, all terms that are typable in D_k are also typable in D_∞, if D_∞ contains only flat, linear decoration rewrite rules.

Proof: Suppose that $\Psi_k : t^{:\downarrow\emptyset} \leftarrow\!\!*\!\!\rightarrow_{D_k} t^{:S}$ is the typing proof in D_k. Then we can first prove for each decoration rewrite rule ($\phi : l^{:s} \to l^{:s\cup S_l}$ if $S_l \not\subseteq s$) used in Ψ_k, that it is either in D_∞ or subsumed by some $\phi' \in D_\infty$ in the sense of **Subsume_deco**.

The proof is an induction on the size of $l^{:S_l}$ w.r.t. the strict part $<_d \cup \lesssim_d$ of $\leq_d \cup \lesssim_d$, where $\leq_d$ is the reduction ordering used for equation orientation and $\lesssim_d$ is decorated term subsumption modulo sort inheritance. Hence, whenever $t \lesssim_d t'$, then there exists a substitution σ with $t' \cong_d \sigma(t)$. Remark that this ordering $<_d \cup \lesssim_d$ is well-founded, since from any infinite decreasing sequence in $<_d \cup \lesssim_d$, we can extract an infinite decreasing sequence in $<_d$: Let $(t_i)_{i\in[1..\infty[}$ be such a sequence in $<_d \cup \lesssim_d$. Now, if $t_{i+2} <_d t_{i+1} \lesssim_d \sigma(t_{i+1}) = t_i$ is a subsequence, then either (a) $t_{i+2} = \sigma(t_{i+2}) <_d \sigma(t_{i+1})$ or (b) $t_{i+2} \lesssim_d \sigma(t_{i+2}) <_d \sigma(t_{i+1})$. Consequently, we get some $J \subseteq [1..\infty[$, s.t. $(t_j)_J$ is a decreasing sequence purely in $<_d$, since we can push all $<_d$-steps to the beginning. If this sequence is finite, then the sequence $(t_i)_{i\in[k..\infty[}$ with $k = max(J)$ is an infinite decreasing sequence in $\lesssim_d$, otherwise (J is infinite and) $(t_j)_{j\in J}$ is an infinite sequence. In any case, we get a contradiction with the well-foundedness of $<_d$ and $\lesssim_d$, respectively.

If $l^{:S_l}$ is irreducible by $\bigcup_{n\geq k} D_n \cup R_n$, then it must either be subsumed by some ($\phi'' : l''^{:s} \to l''^{:s\cup S_{l''}}$ if $S_{l''} \not\subseteq s$) $\in D_m$, $m \geq k$, or it is still in D_∞. In the first case we can apply the induction hypothesis on ϕ'' and get the replacement $\phi' \in D_\infty$ of ϕ'', that must also subsume ϕ, due to the fact that ϕ' is flat and linear, and a variable x in ϕ' occurring in the ith argument has to be of sort A, if $Deco(l_{|_i}) = S \approx \{A\}$. Note that such an A must exist in $\mathcal{S}_\Diamond$, since $l_{|_i}$ has to be valid (see Lemma 7.2.2). Remark that if $S_l \subsetneq S_{l''}$, then $l''^{:S_{l''}} <_d l^{:S_l}$, since $l''^{:S_{l''}} <_d l''^{:S_l}$ and $l''^{:S_l} \lesssim_d l^{:S_l}$. Otherwise, if $S_l \approx S_{l''}$, then $l''^{:S_{l''}} \lesssim_d l''^{:S_l} \lesssim_d l^{:S_l}$. Therefore, the subsumption of ϕ by ϕ'' implies that $l''^{:S_{l''}}$ is strictly smaller than $l^{:S_l}$, i.e. the use of the induction hypothesis to get ϕ' for ϕ'' is correct.

If $l^{:S_l}$ is reduced at step n with R_n, then the top symbol remains the same and therefore using the induction hypothesis on the reduced term gives a ϕ' that is flat and linear. Hence, as before, ϕ' is also subsuming ϕ, due to decoration preservation. The same argument can be applied in the case of $l^{:S_l}$ reduced with D_n. Note that in both cases $l'^{:S_{l'}} <_d l^{:S_l}$.

Now, let Ψ_∞ be the proof $t^{:\downarrow\emptyset} \leftarrow\!\!*\!\!\rightarrow_{D_\infty} t^{:S}$, where every step $\rightarrowtail^{\phi}_{D_k}$ in Ψ_k is replaced by $\rightarrowtail^{\phi'}_{D_\infty} \circ \overset{0.1\phi'}{\leftarrowtail}_{D_\infty}$, i.e. any rule is replaced by the corresponding one in D_∞. □

We get the following results, where CT^k_{DER} denotes the set of all decorated terms in $D_k \cup E_k \cup R_k$ of a decorated presentation (D_k, E_k, R_k) during completion, where sort set variables s in decoration rules are replaced by $\emptyset$.

Lemma 8.2.8 *Let $(D_\mathcal{P}, E_\mathcal{P}, \emptyset) = (D_0, E_0, R_0) \vdash_{OSC} (D_1, E_1, R_1) \vdash_{OSC} \ldots$ be a derivation using OSC, s.t. all decorated rewrite rules ϕ used with* **Compose_R_deco** *are* decoration preserving. *Then for all $k \geq 0$, all terms in CT^k_{DER} are typable in D_∞, if D_∞ only contains flat, linear decoration rewrite rules.*

Proof: The proof is an induction over the number of completion steps k. If $k = 0$, then all $t^{:S} \in CT^0_{DER}$ are equal to $t^{:\downarrow\emptyset}$ or $(t^{:\downarrow\emptyset})^{:S}$ (for terms in decoration rewrite rules) and therefore trivially typable.

For $k > 0$, we can construct a typing proof of a newly added term in any case. Remark that completion rules adding new terms to CT^k_{DER} do not change or extend the set of decoration rules. If the new term is obtained by decoration rewriting from an old one, the claim is trivial.

If it is obtained by decorated rewriting, we can use Lemma 8.2.6 in order to construct the new typing proof. If the new term is obtained by orientation of an equation, then it can be typed with the proof of the old term in the equation plus, eventually, the new decoration rule applied on top in the end.

Finally, the new term can be the result of a critical pair computation. This can be seen as the unification of two terms $t^{:S}, t'^{:S'}$ in CT^{k-1}_{DER}. The resulting unifier ψ applied to one of the terms in CT^{k-1}_{ER}, gives again a typable term, because of Proposition 8.2.5 and stability of decoration rewriting under substitutivity used for the typing proof of the initial term. The second step is the reduction of $\psi(t^{:S})$ with the lower rule, where Lemma 8.2.6 can be used.

Finally, there are completion rules manipulating decoration rules. In this case, a corresponding typing proof in D_∞ that does not change is given by Lemma 8.2.6. □

Theorem 8.2.9 *Let $\mathcal{P}_\infty \neq (\bot, \bot, \bot)$ be the presentation obtained from $(D_\mathcal{P}, E_\mathcal{P}, \emptyset)$ using OSC, s.t. all decorated rewrite rules ϕ used with* **Compose_R_deco** *are* decoration preserving *and all terms in D_∞ are flat and linear. Let furthermore $E_\infty = \emptyset$ and all critical pairs of $D_\infty \cup R_\infty$ be in $D_* \cup R_*$. Then the initial presentation $\mathcal{P}_0$ is sort inheriting on $ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and $D_\infty \cup R_\infty$ is Church-Rosser, type and existentially complete.*

Proof: The fact that all decoration rewrite rules in D_∞ are flat and linear implies that D_∞ is confluent and therefore $\mathcal{P}_\infty \neq (\bot, \bot, \bot)$ gives us sort inheritance on the set of all typable terms by Lemma 8.1.8 for $\mathcal{T} = reach_{D_\infty}(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond))$. Furthermore, any term generated by the proof reduction $\Longrightarrow$ is typable in D_∞, because of Lemmas 8.2.8, 8.2.6 and 8.2.7. Remark that any new term in a proof generated by $\Longrightarrow$ at completion step k is reachable from an old one by $\overset{*}{\rightarrow}_{D_{k+1} \cup R_{k+1}}$.

Hence, sort inheritance on the set $reach_{D_\infty}(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond))$ gives us the completeness of $\mathbf{UNIF_d}$, which is needed for peak reduction. Furthermore, the condition that all critical pairs of $D_\infty \cup R_\infty$ be in $D_* \cup R_*$ and $E_\infty = \emptyset$ proves the fairness of the derivation (see Proposition 7.4.2) and therefore $D_\infty \cup R_\infty$ has to be Church-Rosser. By Theorem 6.5.1 and Proposition 7.2.3, we know that $Form(\mathcal{P}_\infty) = Form(\mathcal{P}_0) = Th(\mathcal{P})$. Together with the Church-Rosser property, we know that all minimal sorts to which a term belongs can

be found in the top decoration of its normal form: Suppose this is not the case for $\overline{t}^{:\downarrow\emptyset}$. Then let $t^{:S}$ be its $D_\infty \cup R_\infty$-normal form, that must be typable. Let furthermore $\Psi : \overline{t}^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_{D_\infty \cup R_\infty} t'^{:S'}$ be a proof implying $t : A$ with $\nexists \langle B \rangle \in \widehat{S} : B \leq^{syn}_{\mathcal{S}} A$, i.e. w.l.o.g. $\langle A \rangle \in \widehat{S'}$. Consequently, we have a normal form Ψ_{nf} of Ψ, s.t.:

$$\Psi_{nf} : \overline{t}^{:\downarrow\emptyset} \rightarrow^*_{D_\infty \cup R_\infty} t''^{:S''} \leftarrow^*_{D_\infty \cup R_\infty} t'^{:S'},$$

where $t''^{:S''}$ is irreducible and $S' \subsetneq S''$. Now, $t^{:S} \leftarrow^*_{D_\infty \cup R_\infty} \overline{t}^{:\downarrow\emptyset} \rightarrow^*_{D_\infty \cup R_\infty} t''^{:S''}$ is a proof with typable terms only, that is reduced by $\Longrightarrow$ into a rewrite proof, i.e. $t^{:S} \cong_d t''^{:S''}$ since both terms are supposed to be irreducible. Therefore, $S \approx S''$, in contradiction to $\langle A \rangle \in \widehat{S'} \subsetneq S''$ and $\nexists \langle B \rangle \in \widehat{S} : B \leq^{syn}_{\mathcal{S}} A$.

Now, we can conclude that $reach_{D_\infty}(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond))$-sort inheritance gives us sort inheritance on all valid terms. Type and existential completeness follow thus from Theorem 7.4.3. □

2.3 Flat Term Declarations

In order to relax the restriction of linearity on term declarations, the idea is to replace rewriting by layer rewriting and to inhibit simplification by decorated rewrite rules. At the price of trickier proofs and more critical pair computations, we can prove an extension (Theorem 8.2.13) of Theorem 8.2.9.

Definition 8.2.10 *Let $t^{:S}, t'^{:S'} \in \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $\phi \in R$ and k be a natural number.*

Then $t^{:S} \rightleftharpoons_R^{\phi,\sigma,k} t'^{:S'}$, if $t^{:S} \rightharpoonup_R^{\phi,\sigma,O} t'^{:S'}$, s.t. $\forall \omega \in O$, $|\omega| = k$ and O is maximal. Given ϕ, σ and k, $O_{(t^{:S},\phi,\sigma,k)}$ stands for such an O.

Instead of $t[u]_{O_{(t[u],\phi,\sigma,k)}}$ we simply write $t[u]_{O_{(\phi,\sigma,k)}}$. Layer rewriting has the advantage that it preserves typability of terms by flat (not necessarily linear) term declarations, which correspond with the so-called semi-linear membership theories. It has the disadvantage of the maximality condition, which destroys stability by context and substitution, as the following example illustrates:

Example 8.2.11 *Let $\phi : a^{:\{A\}} \rightarrow b^{:\{B\}}$ be a decorated rewrite rule in R and id be the identity substitution, i.e. $Dom(\sigma) = id$. Then, trivially $a^{:\{A\}} \rightleftharpoons_R^{\phi,id,0} b^{:\{B\}}$, but neither:*

$$f(a^{:\{A\}}, a^{:\{A\}})^{:\emptyset} \rightleftharpoons_R^{\phi,id,1} f(a^{:\{A\}}, b^{:\{B\}})^{:\emptyset},$$

(which is a counter-example for stability by context), nor does:

$$f(x^{:\{A\}}, a^{:\{A\}})^{:\emptyset} \rightleftharpoons_R^{\phi,id,1} f(x^{:\{A\}}, b^{:\{B\}})^{:\emptyset} \text{ imply } f(a^{:\{A\}}, a^{:\{A\}})^{:\emptyset} \rightleftharpoons_R^{\phi,id,1} f(a^{:\{A\}}, b^{:\{B\}})^{:\emptyset},$$

(which is a counter-example for stability by substitution). The only correct layer reduction from $f(a^{:\{A\}}, a^{:\{A\}})^{:\emptyset}$ is:

$$f(a^{:\{A\}}, a^{:\{A\}})^{:\emptyset} \rightleftharpoons_R^{\phi,id,1} f(b^{:\{A\}}, b^{:\{A\}})^{:\emptyset}.$$

The idea behind layer rewriting is that typing with flat decoration rules is preserved. This is proved in the following Lemma:

Lemma 8.2.12 *Let D be a set of decorated rewrite rules with flat terms only and R a set of decorated rewrite rules. Let furthermore $t^{:S} \rightleftharpoons_R^{\phi,\sigma,k} t'^{:S'}$ and $(\phi : l^{:S_l} \rightarrow r^{:S_r}) \in R$, s.t. there are typing proofs $\Psi : (r^{:S_r})^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_D r^{:S_r}$ and $\Psi' : (t^{:S})^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_D t^{:S}$.*

Then there is a proof $\Upsilon : (t'^{:S'})^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^*_D t'^{:S'}$.

Proof: First, notice that we can assume Ψ, Ψ' to be bottom-up w.l.o.g., since there are no decoration critical pairs. Clearly, $t'^{:S'} \cong_d t^{:S}[\sigma(r^{:S_r})]_{O_{(\phi,\sigma,k)}}$. Furthermore, Lemma 8.2.4 guarantees us the typability of the terms in $\mathcal{I}m(\sigma)$ and therefore we get by substitutivity and context stability of decoration rewriting, by abuse of our proof syntax:

$$(t^{:s})^{:\downarrow\emptyset}[\sigma(r^{:S_r})^{:\downarrow\emptyset}[\Psi'_{|_{O_{(\phi,\sigma,k)}.\mathcal{VOcc}(l^{:S_l})}}]_{\mathcal{VOcc}(r^{:S_r})} + \sigma(\Psi)]_{O_{(\phi,\sigma,k)}} :$$
$$(t'^{:S'})^{:\downarrow\emptyset} \leftarrow\!\!\!\!\rightarrow^{*}_{D} (t^{:S})^{:\downarrow\emptyset}[\sigma(r^{:S_r})]_{O_{(\phi,\sigma,k)}}$$

Remark that we implicitly understand that typing proofs for different variable images are distinguished correctly when we write $[\Psi'_{|_{O_{(\phi,\sigma,k)}.\mathcal{VOcc}(l^{:S_l})}}]_{\mathcal{VOcc}(r^{:S_r})}$. Now we can append also all decoration rule applications at occurrences incomparable to $O_{(\phi,\sigma,k)}$ in Ψ', since their substitutions do not change. For all rule applications above some $\omega \in O_{(\phi,\sigma,k)}$, we only have to adapt the substitutions, since the rules are flat and $O_{(\phi,\sigma,k)}$ is maximal. Consequently, Υ exists. □

Using this property, we can give a critical pair lemma preserving typing proofs, using a calculus $\mathcal{SLC}$ which is essentially $\mathcal{OSC}$ without simplification by decorated rewrite rules. This leads us to the following theorem:

Theorem 8.2.13 *Let $\mathcal{P}_\infty \neq (\perp, \perp, \perp)$ be the presentation obtained from $(D_\mathcal{P}, E_\mathcal{P}, \emptyset)$ using $\mathcal{SLC}$, s.t. all terms in D_* are flat. Let furthermore $E_\infty = \emptyset$ and all critical pairs of $D_\infty \cup R_\infty$ be in $D_* \cup R_*$. Then the initial presentation $\mathcal{P}_0$ is sort inheriting on $\mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and $D_\infty \cup R_\infty$ is Church-Rosser, type and existentially complete.*

The proof is left out here, since it is rather technical due to the loss of stability by context of the rewrite relation. All proofs and lemmas can be found in Appendix B.

2.4 General Term Declarations

Let us first define some needed notions and assumptions, in order to shorten lemmas and proofs in the following. The subsumption of sets of decoration rewrite rules is a property that can be seen as combination of set subsumption combined with the rule **Subsume_deco**, i.e. D subsumes D', written $D' \subseteq_D D$, if for all ϕ' in D', there is a ϕ in D that subsumes ϕ in the sense of **Subsume_deco** with $S \approx S'$. Let $\phi' : l^{:S_l} \to r^{:S_r}$ and $\phi'' : g^{:S_g} \to d^{:S_d}$ be decorated rewrite rules in R in the sequel.

Definition and Simple Properties

Working without restrictions on term declarations compels us to define a new proof reduction. Typability of terms in the transformed proof is obtained by proving that each new term comes from an old typable one by decorated rewriting in a maximally subterm sharing way.

Definition 8.2.14 *A term $t^{:S}$* rewrites in a maximally subterm sharing way *(or MSS rewrites for short) into $t'^{:S'}$ using a decorated rewrite rule $\phi : l^{:S_l} \to r^{:S_r}$ and a decorated substitution σ if:*

1. $t^{:S} \rightarrowtail_R^{O,\sigma,\phi} t'^{:S'}$
2. *and* $O = \{\omega \in Occ(t^{:S}) \mid t^{:S}{}_{|_\omega} \cong_d \sigma(l^{:S_l})\}$.

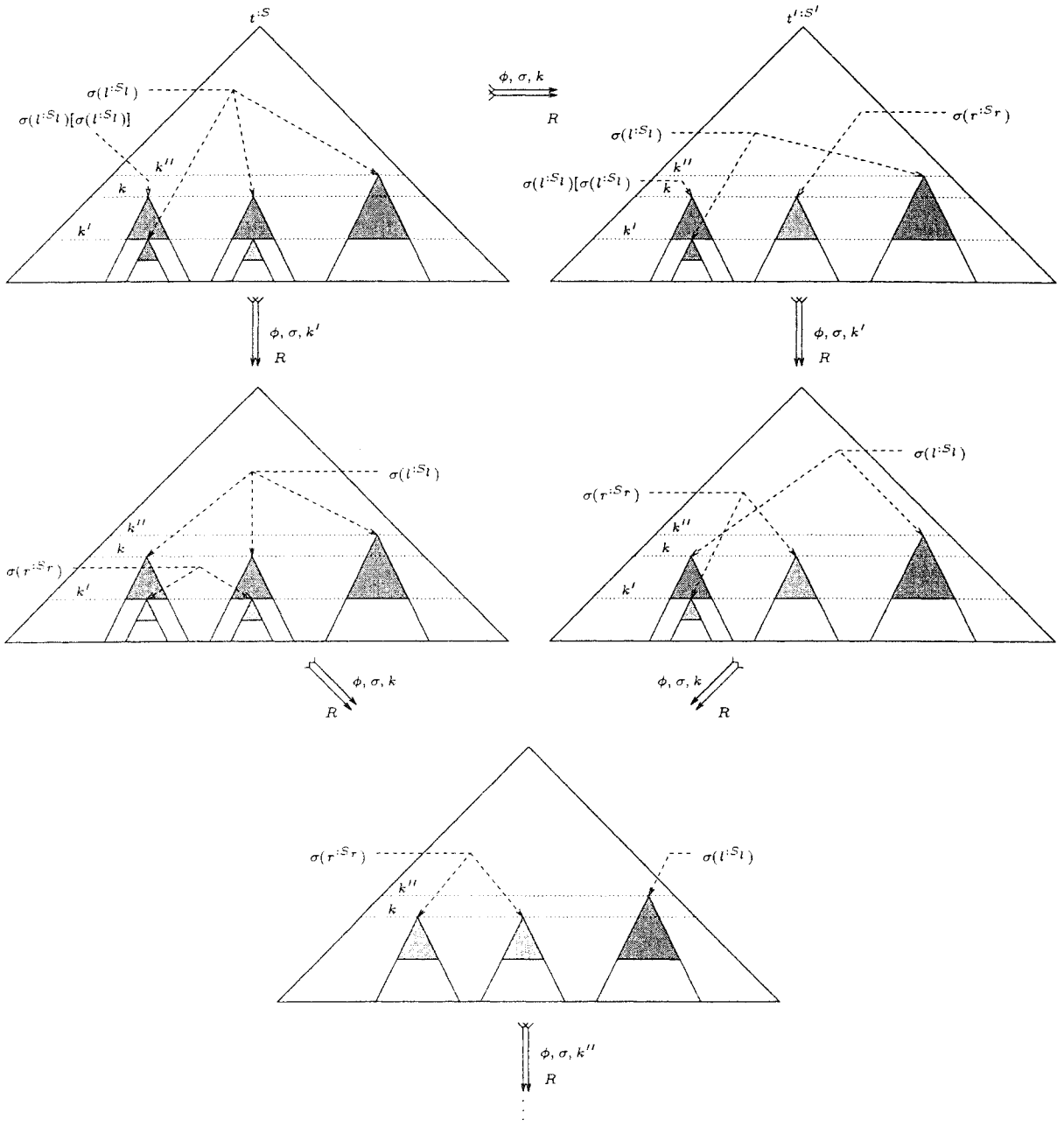


Figure 8.2: Bottom-up Layer Rewriting.

This is written $t^{:S} \twoheadrightarrow_R^{\sigma,\phi} t'^{:S'}$. *The set* O *of redex occurrences is written* $O_{(t^{:S},\sigma,\phi)}$.

Instead of $t[u]_{O_{(t[u],\phi,\sigma)}}$ we may simply write $t[u]_{O_{(\phi,\sigma)}}$. The pair (ϕ,σ) will also be called a *radical*. Clearly, the maximality condition destroys once more stability by context and substitution. The use of a different decorated rewrite relation also forces a redefinition of $CP(R,R)$ and $CP(R,D)$.

Definition 8.2.15 MSS decorated critical pairs
(obtained by MSS superposition *into* decorated rules)
Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $g^{:S_g} \to d^{:S_d}$ *and* $l^{:S_l} \to r^{:S_r}$ *be rewrite rules in* R *with disjoint sets of variables.*

The two rules overlap if there exists a position ω *in the set of non-variable positions of* $g^{:S_g}$, *such that the decorated terms* $g^{:S_g}{}_{|_\omega}$ *and* $l^{:S_l}$ *have the* $\mathcal{T}$*-complete, sound, non-empty set* Ψ *of strict decorated unifiers. Let* $O = \{\omega_i\}_{i\in[1..n]}$ *be the set of such positions and* Ψ_i *the corresponding unifiers.*

Then for any combination ψ *of some subset of* $\{\sigma \mid \exists i \in [1..n] : \sigma \in \Psi_i\}$ *corresponding with the set of overlap positions* $\overline{O} \subseteq O$, *s.t.* $\overline{O} \neq \emptyset$ *and* $\overline{O} = O_{(\psi(g^{:S_g}),l^{:S_l}\to r^{:S_r},\psi)}$, *the overlap produces the* $\mathcal{T}$*-MSS decorated critical pair (*$\mathcal{T}$*-MSSCP(R,R))* $(p^{:S_1} = q^{:S_2})$ *where* $q^{:S_2} =_d \psi(d^{:S_d})$ *and* $p^{:S_1} =_d \psi(g^{:S_g}[r^{:S_r}]_{\overline{O}})$.

Remark that the condition $\overline{O} = O_{(\psi(g^{:S_g}),l^{:S_l}\to r^{:S_r},\psi)}$ guarantees that the term $\psi(g^{:S_g}[r^{:S_r}]_{\overline{O}})$ is reachable from $\psi(g^{:S_g})$ by maximally subterm sharing rewriting using the radical $(l^{:S_l} \to r^{:S_r}, \psi)$.

Definition 8.2.16 MSS decoration critical pairs
(obtained by MSS superposition *into* decoration rules)
Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $g^{:s} \to g^{:s\cup S_g}$ if $S_g \not\subseteq s$ *and* $l^{:S_l} \to r^{:S_r}$ *be in* D *and* R, *respectively, with disjoint sets of variables.*

The two rules overlap if there exists a position ω *in the set of non-variable positions of* $g^{:\emptyset}$, *such that the decorated terms* $g^{:\emptyset}{}_{|_\omega}$ *and* $l^{:S_l}$ *have the* $\mathcal{T}$*-complete, sound, non-empty set* Ψ *of strict decorated unifiers. Let* $O = \{\omega_i\}_{i\in[1..n]}$ *be the set of such positions and* Ψ_i *the corresponding unifiers.*

Then for any combination ψ *of some subset of* $\{\sigma \mid \exists i \in [1..n] : \sigma \in \Psi_i\}$ *corresponding with the set of overlap positions* $\overline{O} \subseteq O$, *s.t.* $\overline{O} \neq \emptyset$ *and all occurrences in* $\overline{O}$ *are incomparable, the overlap produces the* $\mathcal{T}$*-MSS decoration critical pair (*$\mathcal{T}$*-MSSCP(R,D))* $(p^{:s} = q^{:s\cup S}$ if $S_g \not\subseteq s)$ *where* $q^{:s\cup S} =_d \psi(g^{:s\cup S_g})$ *and* $p^{:s} =_d \psi(g[r^{:S_r}]_{\overline{O}})^{:s}$.

Remark that the condition $\overline{O} = O_{(\psi(g^{:\emptyset}),l^{:S_l}\to r^{:S_r},\psi)}$ is missing, since we need more decoration rewrite rules for typing proof propagation. As with $CP(R,D)$, we can restrict our attention to superposition at occurrences $\omega \neq \Lambda$ in the last definition, since superpositions at the top occurrence are already handled in the definition of $CP(D,R)$.

Let $MSSCP(R,R)_{|_\mathcal{T}}$ and $MSSCP(R,D)_{|_\mathcal{T}}$, respectively, denote the set of such MSS critical pairs. Similarly to $\overline{CP(R,D)}$, we define decoration rewrite rules corresponding with decoration critical pairs for MSS rewriting. However, the fact that a condition of the shape $\overline{O} = O_{(\psi(g^{:\emptyset}),l^{:S_l}\to r^{:S_r},\psi)}$ is missing for decoration critical pairs results in the possibility to create untypable decoration rewrite rules, as the following example illustrates:

Example 8.2.17 *Let* ϕ *and* ϕ' *be defined as follows, respectively:*
$h(g(f(x^{:A})^{:A}, f(x^{:A})^{:A})^{:B})^{:s} \to h(g(f(x^{:A})^{:A}, f(x^{:A})^{:A})^{:B})^{:s\cup\{C\}}$ if $\{C\} \not\subseteq s$,
$g(x^{:A}, x^{:A})^{:s} \to g(x^{:A}, x^{:A})^{:s\cup\{B\}}$ if $\{B\} \not\subseteq s$.

Clearly, the decoration B in ϕ can be added using ϕ'. Assume now, we have a decorated rewrite rule ϕ'' : $f(x^{:A})^{:A} \to x^{:A}$. Hence, calculating the overlap of ϕ'' into ϕ at position 1.1 *results in a rule of the shape:*

$$\psi \ : \ h(g(x^{:A}, f(x^{:A})^{:A})^{:B})^{:s} \to h(g(x^{:A}, f(x^{:A})^{:A})^{:B})^{:s\cup\{C\}} \text{ if } \{C\} \lesssim s.$$

Clearly, ϕ' can no more be used to add B due to its non-linearity. Hence, the new rule may no more be typable. We solve this problem by adding another decoration rule allowing to add B:

$$\psi' \ : \ g(x^{:A}, f(x^{:A})^{:\emptyset})^{:s} \to g(x^{:A}, f(x^{:A})^{:\emptyset})^{:s\cup\{B\}} \text{ if } \{B\} \lesssim s.$$

This motivates to define the replacement $\overline{MSSCP(R,D)}$ of $\overline{CP(R,D)}$ as follows, assuming all symbols to be defined as in Definition 8.2.16:

$$\{(p_{|\omega}{}^{:s} \to p_{|\omega}{}^{:s\cup S_\omega} \text{ if } S_\omega \lesssim s) \mid \begin{array}{l}(p^{:s} = q^{:s\cup S} \text{ if } S \lesssim s) \in MSSCP(R,D),\\ \omega \in \mathcal{NVO}cc(p^{:S}),\\ Deco(p_{|\omega}{}^{:S}) = S_\omega\\ \text{and if } O = O_{(\psi(g^{:\emptyset}),l^{:S_l}\to r^{:S_r},\psi)} \text{ then } \omega = \Lambda\}\end{array}$$

Let us give two examples for MSSCP computations:

Example 8.2.18 *Let the following two rules be in D and R, respectively:*
$(\psi : g(f(x^{:A})^{:A}, f(a^{:A})^{:A})^{:s} \to g(f(x^{:A})^{:A}, f(a^{:A})^{:A})^{:s\cup\{B\}} \text{ if } \{B\} \lesssim s)$,
$(\psi' : f(a^{:A})^{:A} \to a^{:A})$.
Then there are the following MSS decoration critical pairs $(p_i{}^{:s} = q_i{}^{:s\cup S_i} \text{ if } S_i \lesssim s)$, $i \in [1..3]$, of ψ' and ψ, s.t.:

$$\begin{array}{ll} p_1{}^{:s} =_d g(f(x^{:A})^{:A}, a^{:A})^{:s}, & q_1{}^{:s\cup S_1} =_d g(f(x^{:A})^{:A}, f(a^{:A})^{:A})^{:s\cup\{B\}},\\ p_2{}^{:s} =_d g(a^{:A}, f(a^{:A})^{:A})^{:s}, & q_2{}^{:s\cup S_2} =_d g(f(a^{:A})^{:A}, f(a^{:A})^{:A})^{:s\cup\{B\}} \text{and}\\ p_3{}^{:s} =_d g(a^{:A}, a^{:A})^{:s}, & q_3{}^{:s\cup S_3} =_d g(f(a^{:A})^{:A}, f(a^{:A})^{:A})^{:s\cup\{B\}}. \end{array}$$

Consequently, $\overline{MSSCP(R,D)}$ results in:

$$\begin{array}{l}(g(f(x^{:A})^{:A}, a^{:A})^{:s} \to g(f(x^{:A})^{:A}, a^{:A})^{:s\cup\{B\}} \text{ if } \{B\} \lesssim s),\\ (g(a^{:A}, f(a^{:A})^{:A})^{:s} \to g(a^{:A}, f(a^{:A})^{:A})^{:s\cup\{B\}} \text{ if } \{B\} \lesssim s),\\ (f(a^{:A})^{:s} \to f(a^{:A})^{:s\cup\{A\}} \text{ if } \{A\} \lesssim s),\\ (a^{:s} \to a^{:s\cup A} \text{ if } \{A\} \lesssim s),\\ (g(a^{:A}, a^{:A})^{:s} \to g(a^{:A}, a^{:A})^{:s\cup\{A\}} \text{ if } \{A\} \lesssim s).\end{array}$$

Remark that $(g(a^{:A}, f(a^{:A})^{:A})^{:s} \to g(a^{:A}, f(a^{:A})^{:A})^{:s\cup\{A\}} \text{ if } \{A\} \lesssim s)$ does not fulfill the condition $\overline{O} = O_{(\psi(g^{:\emptyset}),l^{:S_l}\to r^{:S_r},\psi)}$, i.e. we have to add the typing rules for a and $f(a)$.

The second example handles the same rules as Example B.0.4.

Example 8.2.19 *Let us define two decorated rewrite rules:*
$(\psi : g(f(x^{:A}, a^{:A})^{:A}, f(f(y^{:A}, a^{:A})^{:A}, y'^{:A})^{:A})^{:B} \to g(x^{:A}, y^{:A})^{:B})$ *and*
$(\psi' : f(z^{:A}, a^{:A})^{:A} \to z^{:A})$.

Clearly, ψ' overlaps into ψ at occurrences 1, 2 *and* 2.1 *with unifiers $\sigma_1 = \{z^{:A} \mapsto x^{:A}\}, \sigma_2 = \{z^{:A} \mapsto f(y^{:A}, a^{:A})^{:A}, y'^{:A} \mapsto a^{:A}\}$ and $\sigma_{2.1} = \{z^{:A} \mapsto y^{:A}\}$, respectively. Then the $MSSCP$s are:*

1. *overlap at* 1: $g(x^{:A}, f(f(y^{:A}, a^{:A})^{:A}, y'^{:A})^{:B} = g(x^{:A}, y^{:A})^{:B}$,

2. *overlap at* 2: $g(f(x^{:A}, a^{:A})^{:A}, f(y^{:A}, a^{:A})^{:A})^{:B} = g(x^{:A}, y^{:A})^{:B}$

3. *overlap at* 2.1:
$g(f(x^{:A}, a^{:A})^{:A}, f(y^{:A}, y'^{:A})^{:A})^{:B} = g(x^{:A}, y^{:A})^{:B}$,

4. *Combining* 1 *and* 2:
$g(f(y^{:A}, a^{:A})^{:A}, f(y^{:A}, a^{:A})^{:A})^{:B} = g(f(y^{:A}, a^{:A})^{:A}, y^{:A})^{:B}$

5. *Combining* 1 *and* 2.1:
$g(x^{:A}, f(x^{:A}, y'^{:A})^{:A})^{:B} = g(x^{:A}, x^{:A})^{:B}$,

The redexes 2 *and* 2.1 *cannot be combined, since they are comparable.*

In the following, we call a term bottom-up typable if there is a typing proof in $\stackrel{*}{\rightarrowtail}_D$, that is bottom-up. The next lemma plays a central role for the typing conservation by the proof transformation used with MSS rewriting.

Lemma 8.2.20 *Let* $\mathcal{T} = reach_D(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$, (D, E, R) *to be a decorated presentation, s.t.* D *is confluent,* $\overline{MSSCP(R,D)}_{|_\mathcal{T}} \subseteq_D D$ *and* $t^{:S}$ *be a bottom-up typable decorated term, as well as all terms in* CT_{DER}.

Let $t^{:S} \stackrel{\phi,\sigma}{\rightarrowtail}_R t'^{:S'}$. *Then* $t'^{:S'}$ *is also bottom-up typable.*

Proof: For the terms in $\mathcal{I}n(\sigma)$, we can use the corresponding parts of the typing proof of $t^{:S}$, since decorated matching is subterm conservative (see Lemma 8.2.4). The same is true for all occurrences being incomparable with the redex positions. For the right-hand side of ϕ, we can take the instantiation of the old proof by σ. Remain the occurrences ω above the redexes. There, we can use the critical pairs from ϕ into $\phi' \in D$ in case of $O_{(t^{:S},\phi,\sigma)} \cap \omega.\mathcal{NVO}cc(lhs(\phi')) \neq \emptyset$ (critical overlap), that are assumed to be included in D, and the decoration rewrite rules for $t^{:S}$ in case of $O_{(t^{:S},\phi,\sigma)} \cap \omega.\mathcal{VO}cc(lhs(\phi')).\omega' \neq \emptyset$ for some ω' (variable overlap). In the last case, the used substitution can obviously be adapted for $t'^{:S'}$, since we reduced all identical subterms at the same time.

More formally, if $\mathcal{T} = reach_D(\mathcal{V}alid\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$, Ψ_i is the bottom-up typing proof of $\sigma(x_i^{:S_i})$, where $\mathcal{D}om(\sigma) = \{x_i^{:S_i} \mid i \in [1..n]\}$, and Ψ is the bottom-up typing proof of $r^{:S_r}$, then there exists a $t''^{:S''}$, s.t.:

$$t^{:\downarrow\emptyset}[\sigma(r^{:S_r})^{:\downarrow\emptyset}[(\Psi_i)_{i\in[1..n]}]_{\mathcal{VO}cc(r^{:S_r})} + \sigma(\Psi)]_{O_{(t^{:S},\phi,\sigma)}} : t'^{:\downarrow\emptyset} \stackrel{*}{\rightarrowtail}_D t^{:\downarrow\emptyset}[\sigma(r^{:S_r})]_{O_{(t^{:S},\phi,\sigma)}},$$

and $t^{:\downarrow\emptyset}[\sigma(r^{:S_r})]_{O_{(t^{:S},\phi,\sigma)}} \stackrel{*}{\rightarrowtail}_{\overline{MSSCP(\phi,D)}_{|_\mathcal{T}}} t'^{:S'}$. Now, every $\stackrel{*}{\rightarrowtail}_{\overline{MSSCP(\phi,D)}}$-step can be replaced by some step in D, since $\overline{MSSCP(R,D)}_{|_\mathcal{T}} \subseteq_D D$ and the decorated terms $t^{:S}, lhs(\phi)$ and $rhs(\phi)$ are all in $\mathcal{T}$. □

Using the same arguments as in the proof of Lemma 7.1.12 we get a Critical Pairs Lemma for MSS-rewriting:

Lemma 8.2.21 *Let* $t^{:V}, t'^{:S'}, t''^{:S''} \in \mathcal{T}$ *be decorated terms such that:*

$$t'^{:S'} \stackrel{\alpha,\phi'}{\leftarrowtail}_R t^{:V} \stackrel{\beta,\phi''}{\rightarrowtail}_R t''^{:S''},$$

with $O_{(\phi',\alpha)} \cap O_{(\phi,\beta)} \neq \emptyset$. *Then, there exists a* $\mathcal{T}$*-MSS decorated critical pair:*

$$(p^{:S_p} = q^{:S_q} \text{ if } S_q \not\gtrsim s)$$

Moreover, there is a decorated substitution ψ *in a* $\mathcal{T}$*-complete set of decorated unifiers according to Definition 8.2.15, such that* $\beta\alpha \gtrsim_d^W \psi$ *with* $W = Var(g) \cup Var(l)$ *and there exists a decorated substitution* τ, *such that* $t'^{:S'}|_{O_{(t^{:S},\phi',\alpha)}} \cong_d \tau(p^{:S_p})$ *and* $t''^{:S''}|_{O_{(t^{:S},\phi'',\beta)}} \cong_d \tau(q^{:S_q})$. *If:*

$$t'^{:S'} \leftarrow_R^{\alpha,\phi',O'} t^{:V} \rightarrowtail_D^{\Lambda,\beta,\phi''} t''^{:S''},$$

with $O' \cap \mathcal{NVO}cc(lhs(\phi'')) \neq \emptyset$, *then there exists a* $\mathcal{T}$*-MSS decoration critical pair:*

$$(p^{:s} = q^{:s \cup S_q} \text{ if } S_q \not\gtrsim s)$$

of the rule ϕ' *on the rule* ϕ''.

Moreover, there is a decorated substitution ψ *in a* $\mathcal{T}$*-complete set of decorated unifiers according to Definition 8.2.16, such that* $\beta\alpha \gtrsim_d^W \psi$ *with* $W = Var(g) \cup Var(l)$ *and there exists a decorated substitution* τ, *such that* $t'^{:S'}|_{O'} \cong_d \tau(p^{:U})$ *and* $t''^{:S''} \cong_d \tau(p^{:S_q \cup U})$ *for some* U, *s.t.* $S_q \not\gtrsim U$.

Furthermore, we can extend the result of Lemma 8.2.20 in the following way:

Lemma 8.2.22 *Let* $\mathcal{T} = reach_D(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\downarrow\emptyset})$, (D, E, R) *to be a decorated presentation, s.t.* D *is confluent,* $\overline{MSSCP(R,D)}|_{\mathcal{T}} \subseteq_D D$ *and* $t^{:S}$ *be a bottom-up typable decorated term, as well as all terms in* CT_{DER}.

Let $\phi' : l^{:S_l} \rightarrow r^{:S_r}$ *and* $\phi'' : g^{:S_g} \rightarrow d^{:S_d}$ *be decorated rewrite rules and* $t^{:S} \twoheadrightarrow_R^{\phi',\sigma'} t'^{:S'} \rightarrow_R^{\phi'',\sigma'',\overline{O}} t''^{:S''}$, *s.t.:*

1. $O' = O_{(t^{:S},\phi',\sigma')}$,
2. $O'' = O_{(t^{:S},\phi'',\sigma'')}$,
3. $\overline{O} = O'' \setminus \{\omega'' \mid \exists \omega \in O'.\mathcal{NVO}cc(r^{:S_r}), \omega'' < \omega\}$ *and*
4. $O' \asymp O''$ *or* $\sigma''(g^{:S_g})$ *is a subterm of* $\sigma'(l^{:S_l})$.

Then $t'^{:S'}$ *and* $t''^{:S''}$ *are also bottom-up typable.*

Proof: The bottom-up typability of $t'^{:S'}$ follows immediately from Lemma 8.2.20. For $t''^{:S''}$ we have to construct such a proof from the one of $t^{:S}$. Remark that the case $\sigma'(l^{:S_l}) \cong_d \sigma''(g^{:S_g})$ is excluded, since this implies that $\overline{O}$ is the empty set.

Let Ψ be the assumed bottom-up typing proof for $t^{:S}$. Remember that the bottom-up typing proof Ψ' of $t'^{:S'}$ was constructed by concatenation of $\Psi|_{\omega \asymp O'}$, $\Psi|_{O'.\mathcal{VO}cc(l^{:S_l})}$, the bottom-up typing proof of $r^{:S_r}$ and $\overline{\Psi'}$, which is the result of decoration critical pairs computation using overlaps of ϕ' into all decoration rewrite rules that are applied above O' in Ψ.

Now, the bottom-up decoration proof Ψ'' for $t''^{:S''}$ can be constructed in almost the same way, as concatenation of $\Psi'|_{\omega \asymp \overline{O}}$, $\Psi'|_{\overline{O}.\mathcal{VO}cc(g^{:S_g})}$, the bottom-up typing proof of $d^{:S_d}$ and $\overline{\Psi''}$,

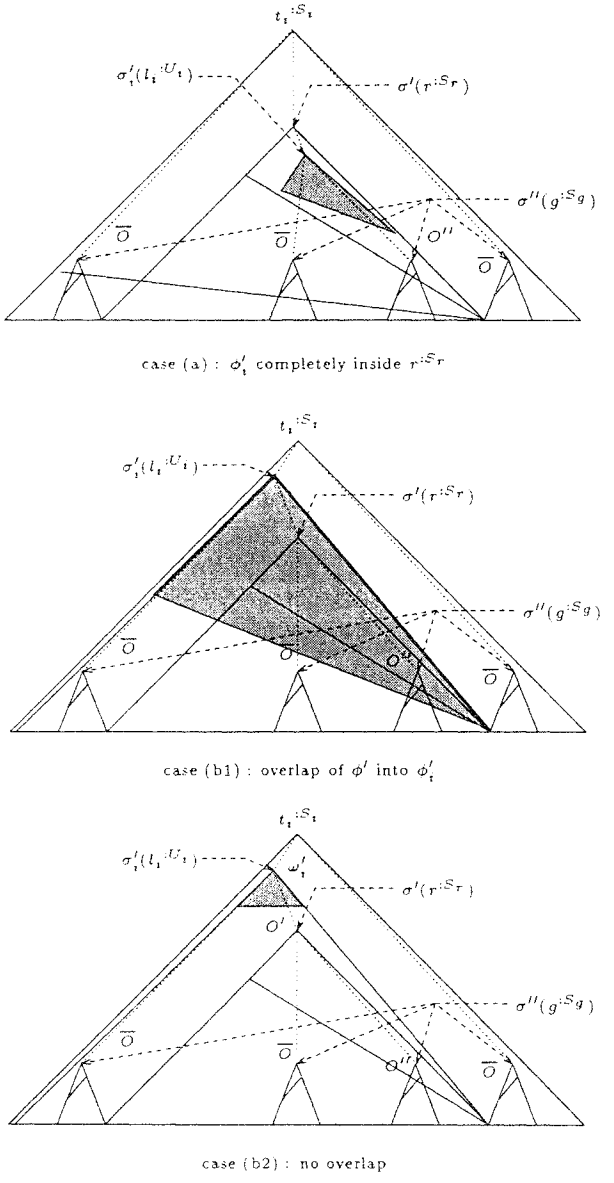


Figure 8.3: The Proof Cases for Extended Type Propagation

which is defined in the following. Note that the only remaining decorations are those above the occurrences where ϕ'' was applied.

Let $\Psi'_{|_{\omega<\overline{o}}} : t'_0{}^{:S'_0} \rightarrowtail_D^{\phi'_1,\sigma'_1,\omega'_1} t'_1{}^{:S'_1} \rightarrowtail_D^{\phi'_2,\sigma'_2,\omega'_2} \ldots \rightarrowtail_D^{\phi'_n,\sigma'_n,\omega'_n} t'_n{}^{:S'_n} \cong_d t^{:S}$.

We prove for all $i \in [1..n]$ the existence of ϕ''_i, σ''_i corresponding with ϕ'_i, σ'_i, s.t. we get a proof $\Psi'' : t''_0{}^{:S''_0} \rightarrowtail_D^{\phi''_1,\sigma''_1,\omega'_1} t''_1{}^{:S''_1} \rightarrowtail_D^{\phi''_2,\sigma''_2,\omega'_2} \ldots \rightarrowtail_D^{\phi''_i,\sigma''_i,\omega'_i} t''_n{}^{:S''_n}$, which provides $Deco((t''^{:S''_i})_{|_{\omega'_i}}) \approx Deco((t'^{:S'_i})_{|_{\omega'_i}})$, $t'_j{}^{:S'_j} \rightarrowtail_R^{\phi'',\sigma'',\overline{O}} t''_j{}^{:S''_j}$ for all $j \in [0..n]$.

The same construction as in the proof of Lemma 8.2.20 can also be used to construct Ψ'', except for the cases where ϕ'_i is non-linear, where we have to prove that the construction of σ''_i from σ'_i is consistent. Assume this is not the case. Then, there are two occurrences, one in $O'' \setminus \overline{O}$ and one in $\overline{O}$, but both in equal relative position to a non-linear variable x in ϕ'_i, s.t. the definition in Lemma 8.2.20 gives $\sigma''_i(x^{:\downarrow\emptyset}) = u^{:U}[\sigma''(g^{:S_g})]_\omega$ and $\sigma''_i(x^{:\downarrow\emptyset}) = u^{:U}[\sigma''(d^{:S_d})]_\omega$. Now, there are two possibilities: Either ϕ'_i is applied in $r^{:S_r}$, i.e. phi'_i is by construction of Ψ' a rule entirely inside $r^{:S_r}$, or it is applied above $r^{:S_r}$.

In the first case, the non-linearity of ϕ'_i ensures that both subterms below x in $r^{:S_r}$ have to be identical. Hence, the ambiguity for $\sigma''_i(x^{:\downarrow\emptyset})$ cannot exist. In the second case, we have again two subcases: either ϕ' overlaps into ϕ'_i or it does not. In the first sub-case, we know by construction of Ψ' that ϕ'_i is the result of a critical pair of ϕ' into some ϕ_i. But this is in contradiction to the fact that ϕ'_i is applied strictly above ϕ' and hence this case is impossible. In the second sub-case, $r^{:S_r}$ is completely below x. But all $r^{:S_r}$ are changed uniformly by the application of ϕ'' in contradiction to the ambiguity of $\sigma''_i(x^{:\downarrow\emptyset})$. This proves the lemma. □

Remark that the use of $\overline{O}$ instead of O'' in the last lemma makes it necessary to calculate MSS decoration critical pairs without the restriction $\overline{O} = O_{(\psi(g^{:\emptyset}),l^{:S_l}\rightarrow r^{:S_r},\psi)}$ in Definition 8.2.16.

Peak Reduction

The reduction of peaks between two decoration rule applications and between a decoration rewrite rule and a decorated rewrite rule are as usual. Note that the peaks corresponding with critical pairs in $MSSCP(R,D)$ differ slightly, due to the multiple occurrences decorated rewriting relation, but the application of a decoration rewrite rule strictly above a decorated rewrite rule ϕ using σ cannot introduce a new redex for the radical (ϕ,σ). Therefore, the proof reduction remains essentially the same. However, the peaks between two decorated rewrite rules applications are reduced in a different way.

We assume in the whole section that (D,E,R) is a decorated presentation, s.t. D is confluent, $\overline{MSSCP(R,D)}_{|_\mathcal{T}} \subseteq_D D$ for $\mathcal{T} = reach_D(\mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond)^{:\downarrow\emptyset})$ and $t^{:S}$ is a bottom-up typable decorated term, as well as all terms in CT_{DER}. The first case is where there are disjoint sets of redexes for the radicals (ϕ',σ') and (ϕ'',σ'').

Lemma 8.2.23 *Let* $t'^{:S'} \leftarrowtail_R^{\phi',\sigma',O'} t^{:S} \rightarrowtail_R^{\phi'',\sigma'',O''} t''^{:S''}$, *s.t.* $O_{(t^{:S},\phi',\sigma')} \asymp O_{(t^{:S},\phi'',\sigma'')}$.
Then there exists $\Psi : t'^{:S'} \underline{0.1}_R \circ \rightarrowtail_R \circ \leftarrowtail_R \circ \underline{0.1}_R\, t''^{:S''}$, *s.t. all new terms are typable.*

Proof: Let $\overline{t'^{:S'}}, \overline{t''^{:S''}}$ be decorated terms, s.t. $\overline{t'^{:S'}} \twoheadleftarrow_R^{\phi',\sigma'} t^{:S} \twoheadrightarrow_R^{\phi'',\sigma''} \overline{t''^{:S''}}$.

Therefore, $t'^{:S'} \underline{0.1}_R^{\phi',\sigma',\overline{O'}} \overline{t'^{:S'}}$ and $t''^{:S''} \underline{0.1}_R^{\phi'',\sigma'',\overline{O''}} \overline{t''^{:S''}}$, where $O' \cup \overline{O'} = O_{(t^{:S},\phi',\sigma')}$ and $O'' \cup \overline{O''} = O_{(t^{:S},\phi'',\sigma'')}$. The resulting peak is illustrated in Figure 8.4.

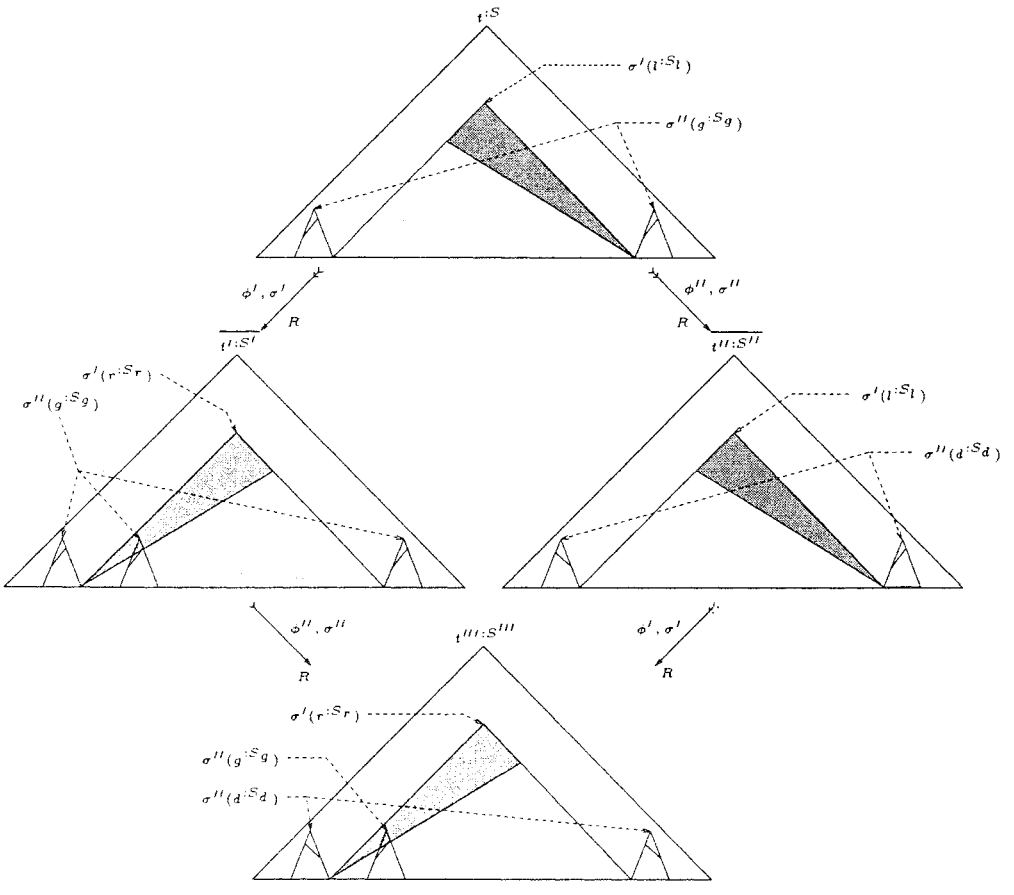


Figure 8.4: No Overlap Case

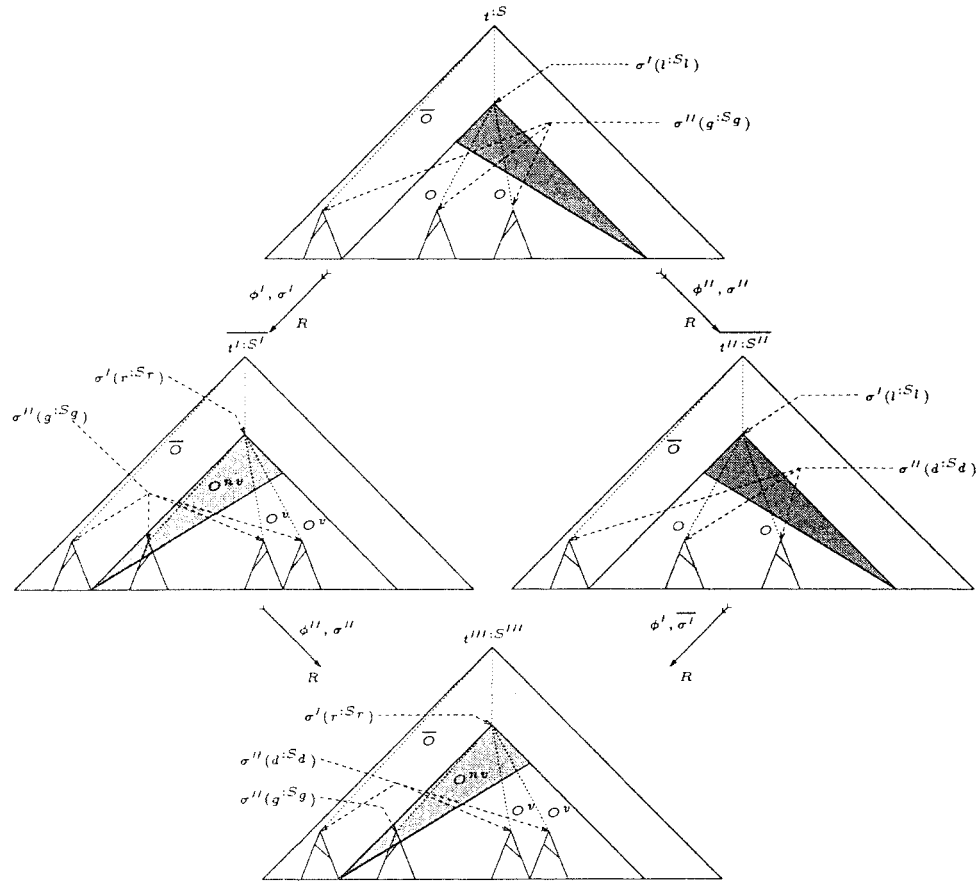


Figure 8.5: Variable Overlap Case

Let w.l.o.g. $t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(\phi',\sigma')}}[\sigma''(g^{:S_g})]_{O_{(\phi'',\sigma'')}}$,

i.e. $\overline{t'^{:S'}} \cong_d t^{:S}[\sigma'(r^{:S_r})]_{O_{(t^{:S},\phi',\sigma')}}[\sigma''(g^{:S_g})]_{O_{(t^{:S},\phi'',\sigma'')}}$,

and $\overline{t''^{:S''}} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(t^{:S},\phi',\sigma')}}[\sigma''(d^{:S_d})]_{O_{(t^{:S},\phi'',\sigma'')}}$,

since $\sigma''(g^{:S_g})$ cannot contain $\sigma'(l^{:S_l})$ and vice versa.

First of all, remark that $\overline{t'^{:S'}}$ and $\overline{t''^{:S''}}$ are bottom-up typable by Lemma 8.2.20. Now, we can reach convergence of the peak:

$$\overline{t'^{:S'}} \underset{R}{\longrightarrow}^{\phi'',\sigma'',O_{(t^{:S},\phi'',\sigma'')}} t'''^{:S'''} \underset{R}{\longleftarrow}^{\phi',\sigma'} \overline{t''^{:S''}}$$

Remark that w.l.o.g. $O_{(t^{:S},\phi',\sigma')} = O_{(\overline{t''^{:S''}},\phi',\sigma')}$, since only one of the radicals can introduce a new radical for the other one, because of well-foundedness of $\rightarrowtail_R$. The typability of $t'''^{:S'''}$ can be obtained by Lemma 8.2.22. □

In the second case, there are only variable overlaps and no non-variable overlaps.

Lemma 8.2.24 *Let* $t'^{:S'} \underset{R}{\longleftarrow}^{\phi',\sigma',O'} t^{:S} \underset{R}{\longrightarrow}^{\phi'',\sigma'',O''} t''^{:S''}$, *where:*

$$O_{(t^{:S},\phi',\sigma')}.\mathcal{NV}Occ(l^{:S_l}) \cap O_{(t^{:S},\phi'',\sigma'')} = \emptyset$$

and there are ω, ω', *s.t.* $\omega \in O_{(t^{:S},\phi',\sigma')}.\mathcal{V}Occ(l^{:S_l})$ *and* $\omega.\omega' \in O_{(t^{:S},\phi'',\sigma'')}$.

Then $t'^{:S'} \underset{R}{\overset{0,1}{\longrightarrow}} \circ \rightarrowtail_R \circ \leftarrowtail_R \circ \underset{R}{\overset{0,1}{\longleftarrow}} t''^{:S''}$, *where all new terms are typable.*

Proof: Let $\overline{t'^{:S'}}, \overline{t''^{:S''}}$ be defined as in the proof of Lemma 8.2.23. The resulting peak is illustrated in Figure 8.5. The typability of $\overline{t'^{:S'}}$ and $\overline{t''^{:S''}}$ follows from Lemma 8.2.20. Hence, we can assume w.l.o.g.:

$t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_O]_{O_{(\phi',\sigma')}}[\sigma''(g^{:S_g})]_{\overline{O}}$,

$\overline{t'^{:S'}} \cong_d t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_{O^v}[\sigma''(g^{:S_g})]_{O^{nv}}]_{O_{(t^{:S},\phi',\sigma')}}[\sigma''(g^{:S_g})]_{\overline{O}}$,

$\overline{t''^{:S''}} \cong_d t^{:S}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma')}}[\sigma''(d^{:S_d})]_{\overline{O}}$.

where $O_{(t^{:S},\phi',\sigma')}.O \uplus \overline{O} = O_{(t^{:S},\phi'',\sigma'')}$, O^{nv} stands for the redexes introduced by the radical (ϕ', σ'), which overlap into non-variable positions of $\sigma'(r^{:S_r})$, and O^v for those under variable positions.

Note that all redexes at $O_{(t^{:S},\phi',\sigma')}.O^v$ in $\overline{t'^{:S'}}$ correspond with some redex at $O_{(t^{:S},\phi',\sigma')}.O$ in $t^{:S}$, since $\mathcal{V}ar(l^{:S_l}) \supseteq \mathcal{V}ar(r^{:S_r})$. The peak converges with:

$$\overline{t'^{:S'}} \underset{R}{\rightarrow}^{\phi'',\sigma'',O_{(t^{:S},\phi',\sigma')}.O^v \cup \overline{O}} t'''^{:S'''} \underset{R}{\leftarrow}^{\phi',\overline{\sigma'}} \overline{t''^{:S''}},$$

where $\overline{\sigma'}$ is defined as follows, provided $O_{\sigma''(g^{:S_g})}$ is the set of occurrences of $\sigma''(g^{:S_g})$ in $x^{:S}$:

$$\overline{\sigma'(x^{:S})} = \sigma'(x^{:S})[\sigma''(d^{:S_d})]_{O_{\sigma''(g^{:S_g})}}$$

Remark that $O_{(t^{:S},\phi',\sigma')} = O_{(\overline{t''^{:S''}},\phi',\overline{\sigma'})}$. The bottom-up typability of $\overline{t'^{:S'}}, \overline{t''^{:S''}}$ follows from the one of $t^{:S}$ together with Lemma 8.2.20. For the typability of $t'''^{:S'''}$ we can still use Lemma 8.2.22. □

The last case covers peaks where there is at least one non-variable overlap (and possibly other variable overlaps).

Lemma 8.2.25 MSS Critical Pair Lemma
Let $t'^{:S'} \underset{R}{\leftarrow}^{\phi',\sigma',O'} t^{:S} \underset{R}{\rightarrow}^{\phi'',\sigma'',O''} t''^{:S''}$, *s.t.* $O_{(t^{:S},\phi',\sigma')}.\mathcal{NVO}cc(l^{:S_l}) \cap O_{(t^{:S},\phi'',\sigma'')} \neq \emptyset$.
Then $t'^{:S'} \overset{0,1}{\rightarrow}_R \circ \overset{0,1}{\rightarrow}_R \circ \leftarrow\!\!\rightarrow_{MSSCP(\phi',\phi'')_{|\mathcal{T}}} \circ \overset{0,1}{\leftarrow}_R t''^{:S''}$, *where* $\mathcal{T}$ *is the set of decorated terms* $reach_D(\mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)^{:\Downarrow\emptyset})$ *and all new terms are bottom-up typable.*

Proof: Let once more $\overline{t'^{:S'}}, \overline{t''^{:S''}}$ be defined as in the proof of Lemma 8.2.23. The resulting peak is illustrated in Figure 8.6. The typability of $\overline{t'^{:S'}}$ and $\overline{t''^{:S''}}$ follows from Lemma 8.2.20. We can assume w.l.o.g.:

$t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_O]_{O_{(\phi',\sigma')}}[\sigma''(g^{:S_g})]_{\overline{O}}$,
$\overline{t'^{:S'}} \cong_d t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_{O^v}[\sigma''(g^{:S_g})]_{O^{nv}}]_{O_{(t^{:S},\phi',\sigma')}}[\sigma''(g^{:S_g})]_{\overline{O}}$ and
$\overline{t''^{:S''}} \cong_d t^{:S}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma')}}[\sigma''(d^{:S_d})]_{\overline{O}}$, s.t.

1. $O^v = \{\omega \mid \exists \omega', \omega'' : \omega \geq \omega''$ with
   (a) $\omega' \in O_{(t^{:S},\phi',\sigma')}$,
   (b) $\omega'' \in \mathcal{VO}cc(r^{:S_r})$ and
   (c) $\omega'.\omega \in O_{(\overline{t'^{:S'}},\phi'',\sigma'')}\}$,
2. $O^{nv} = O_{(\sigma'(r^{:S_r}),\phi'',\sigma'')} \setminus O^v$,
3. $O_{(t^{:S},\phi',\sigma')}.O \cup \overline{O} = O_{(t^{:S},\phi'',\sigma'')}$.

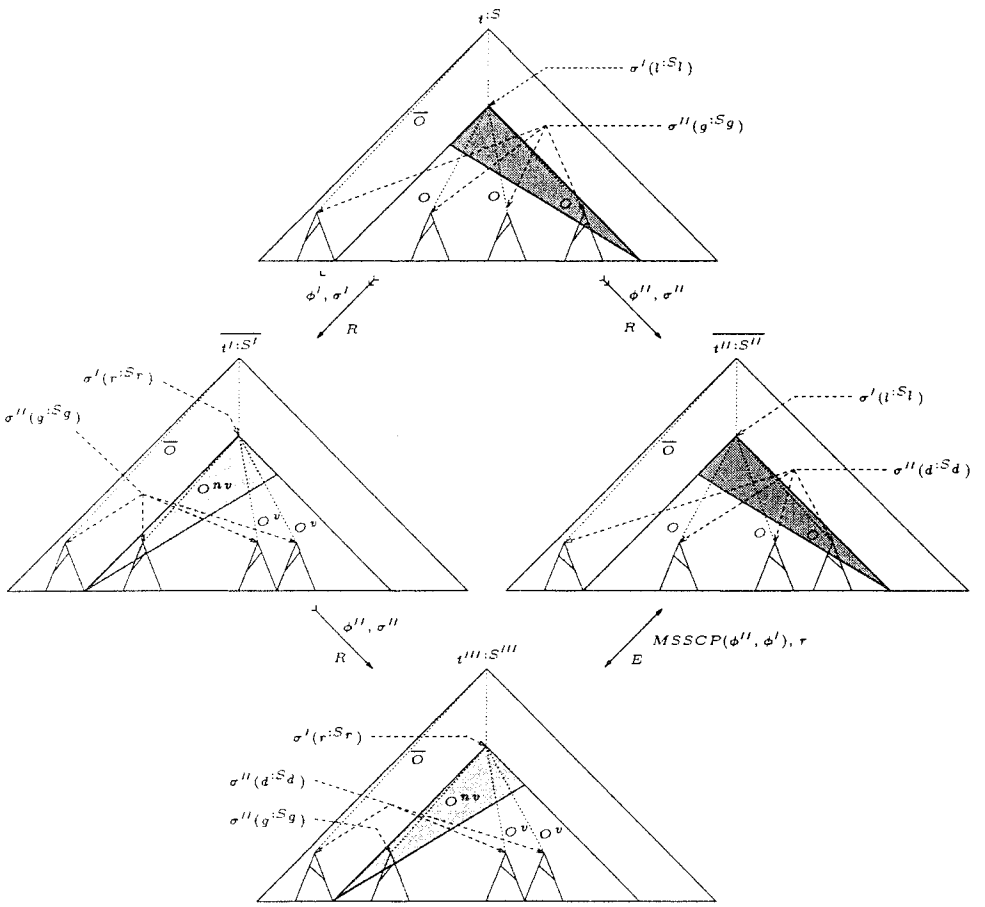


Figure 8.6: Critical Overlap Case

Remark that $O^v \cup \overline{O} \neq O_{\overline{(t'^{:S'},\phi'',\sigma'')}}$, since there might be redexes for the radical (ϕ'', σ'') above $O_{(t^{:S},\phi',\sigma')}.\mathcal{VO}cc(r^{:S_r})$, i.e. possibly $O^{nv} \neq \emptyset$. We can let the peak converge as follows:

$$\overline{t'^{:S'}} \underset{R}{0,1}^{\phi',\sigma',O_{(t^{:S},\phi',\sigma')}.O^v \cup \overline{O}} t'''^{:S'''} \hookleftarrow\!\!\!\rightarrow^{\tau',O_{(t^{:S},\phi',\sigma')}}_{MSSCP(\phi',\phi'')_{|_\mathcal{T}}} \overline{t''^{:S''}},$$

where τ' is defined via the unifier ψ used for the critical pairs computation: if τ is defined by $\tau \circ \psi(t^{:S}) \cong_d \sigma' \circ \sigma''(t^{:S})$, then τ' is defined as follows, provided that $O_{\sigma''(g^{:S_g})}$ is the set of occurrences of $\sigma''(g^{:S_g})$ in $x^{:S}$:

$$\tau'(x^{:S}) = \tau(x^{:S})[\sigma''(d^{:S_d})]_{O_{\sigma''(g^{:S_g})}}.$$

The first reduction with ϕ' is superfluous in the case $\sigma'(l^{:S_l}) \cong_d \sigma''(g^{:S_g})$. The typing proof of $t'''^{:S'''}$ can be obtained from Lemma 8.2.22. □

Orientation and Simplification

Lemma 8.2.26 *Let* $t^{:S} \hookleftarrow\!\!\!\rightarrow_E^{p^{:S}=q^{:S'},\sigma,O} t'^{:S'}$, $(\phi : p^{:S} \to q^{:S\cup S'})$ *be in* R *and* $(\phi' : q^{:s} \to q^{:s\cup S\setminus S'}$ *if* $s \underset{\sim}{\subsetneq} S \setminus S') \in D$, *s.t.* $sort(q) \approx S \cup S'$ *if* $q \in \mathcal{X}_\Diamond$.
Then $t^{:S} \underset{R}{\rightarrowtail}^{\phi,\sigma,O} \circ \underset{D}{0,1}^{\phi',\sigma,O} t'^{:S'}$.

Proof: The proof reduction is identical with the one for $\Longrightarrow$. □

Overlaps with decoration rewrite rules can be treated as usual, since typing conservation is obviously satisfied. Simplification rules for decorated rewrite rules are an unsolved problem.

Confluence

Let $\mathcal{MSSC}$ be the set of completion rules shown in Figure 8.7 plus **Delete**, **Orient_SD**, **Orient_NSD** of $\mathcal{OSC}$ and the failure rules **Detect_NonSI**, **Detect_Subsort**, that do not change. To guarantee bottom-up typability of terms, additional strategy assumptions on the completion process are needed. We call *first-level (completion) rules* those completion rules that are dealing with decorated equalities and rewrite rules only.

General Assumption 8.2.27 *We suppose for the strategy of* $\mathcal{MSSC}$ *the following points:*

1. *Sort inheritance on typable terms is tested whenever no more critical pairs in* $CP(D, D)$ *can be computed.*
2. *Critical pairs in* $MSSCP(R, D)$ *are only computed when* D *is confluent.*
3. *First-level completion rules are only used if no other rules apply.*
4. *Decoration rules are not composed.*
5. *Subsumption of decoration rules is tested in a strict way (i.e.* $S \approx S'$ *in* **Subsume_deco**).
6. *Critical pairs in* $MSSCP(R, D)$ *using one* $\phi \in R$ *are computed block-wise, i.e.* ϕ *is overlapped in all rules in* D *and all the corresponding decoration rewrite rules are added simultaneously.*

1. **Deduce_deco_MSS**
$$\frac{D, E, R}{D \cup \{(p^{:s} \to p^{:s \cup S} \text{ if } S \not\subseteq s)\}, E, R} \quad \text{if } (p^{:s} = p^{:s \cup S} \text{ if } S \not\subseteq s) \in \overline{MSSCP(R, D)} \cup \overline{CP(D, D)}$$

2. **Subsume_deco_MSS**
$$\frac{D \cup \{(p^{:s} \to p^{:s \cup S} \text{ if } c(s))\}, E, R}{D, E, R} \quad \text{if } p^{:\emptyset} \rightarrowtail_D^{\Lambda, \sigma, \phi} p^{:S} \text{ and } (p^{:s} \to p^{:s \cup S} \text{ if } c(s)) \neq \phi$$

3. **Deduce_MSS**
$$\frac{D, E, R}{D, E \cup \{(p^{:S} = q^{:S'})\}, R} \quad \text{if } (p^{:S} = q^{:S'}) \in MSSCP(R, R) \cup CP(D, R)$$

4. **Simplify_MSS_D**
$$\frac{D, E \cup \{(p^{:S} = q^{:S'})\}, R}{D, E \cup \{(p''^{:S''} = q^{:S'})\}, R} \quad \text{if } p^{:S} \rightarrowtail_D^{\sigma, \phi} p''^{:S''}$$

5. **Compose_MSS_D**
$$\frac{D, E, R \cup \{(l^{:S_l} \to r^{:S_r})\}}{D, E, R \cup \{(l^{:S_l} \to r'^{:S_{r'}})\}} \quad \text{if } r^{:S_r} \rightarrowtail_D^{\sigma, \phi} r'^{:S_{r'}}$$

6. **Collapse_MSS_D**
$$\frac{D, E, R \cup \{(l^{:S_l} \to r^{:S_r})\}}{D, E \cup \{(l'^{:S_{l'}} = r^{:S_r})\}, R} \quad \text{if } l^{:S_l} \rightarrowtail_D^{\sigma, \phi} l'^{:S_{l'}} \ \& \ l^{:S_l} \to r^{:S_r} \gg \phi$$

Figure 8.7: $\mathcal{MSSC}$: Specific Completion Rules for Maximally Subterm Sharing Rewriting.

The requirements of General Assumption 8.2.27 can be fulfilled by the strategy shown in Figure 8.8, using the syntax of ELAN [Kirchner *et al.*, 1993]. We clarify our syntax before starting with the propositions. The first level completion rules in $\mathcal{MSSC}$ are limited to **Deduce_MSS** for $MSSCP(R, R)$. The simplification rules will be followed by $!D$, since $\phi \in R$ is not allowed. Furthermore, **Deduce_MSS_RR** stands for the case $MSSCP(R, R)$ and **Deduce_MSS_DR** consequently for $CP(D, R)$. Analogously, **Deduce_deco_MSS** with suffix **_DD** stands for the case $CP(D, D)$ and with suffix **_RD** for the case $MSSCP(R, D)$.

In order to prove the typability of all terms generated by the proof reduction from the proofs constructed in the Birkhoff Theorem 6.5.1, we need the typablility of all terms occurring in intermediate presentations.

Lemma 8.2.28 *Let* $(D_\mathcal{P}, E_\mathcal{P}, \emptyset) = (D_0, E_0, R_0) \vdash (D_1, E_1, R_1) \vdash \ldots$ *be a derivation using* $\mathcal{MSSC}$, *satisfying General Assumption 8.2.27. Then for all* $k \geq 0$, *all terms in* CT^k_{DER} *are typable in* D_k.

Proof: The proof is an induction over the number of completion steps k. If $k = 0$, then all $t^{:S} \in CT^0_{DER}$ are equal to $t^{:\downarrow\emptyset}$ resp. $(t^{:\downarrow\emptyset})^{:S}$ for terms in decoration rewrite rules and therefore trivially typable.

If $k > 0$, then we distinguish different cases for the last applied completion rule:

1. **Deduce_deco_MSS**:
In the case of $CP(D, D)$, the typing proofs are transformed and the new term is

```
strategy Deco_Norm
 repeat dont know choose(Deduce_MSS_DD; Subsume_deco_MSS)
 endrepeat
 dont know choose(Detect_NonSI; Detect_Subsort)
end of strategy

strategy Weakening
 repeat
  repeat dont know choose(Simplify_MSS_D; Compose_MSS_D; Collapse_MSS_D)
  endrepeat,
  dont care choose(Deduce_MSS_DR,
   repeat dont care choose(Delete) endrepeat),
 endrepeat
end of strategy

strategy Propagate
 repeat
  Deco_Norm, Weakening,
  try(Deduce_MSS_RD),
 endrepeat

strategy DecoratedRewrite
 dont know choose(Deduce_MSS_RR; Orient_NSD; Orient_SD),
 try(Detect_Subsort, Detect_NonSI)
end of strategy

strategy MSSC
 repeat dont know choose(Deco_Norm; (Deco_Norm, Weakening);
     Propagate; (Propagate, DecoratedRewrite))
 endrepeat
end of strategy
```

Figure 8.8: A Sample MSSC Strategy

typable by Lemma 8.2.5, since it is only the instantiation of a term in CT^{k-1}_{DER}, that has a decoration enriched via $\rightarrowtail_{D_{k-1}}$.
In the case of $CP(R,D)$, the new terms added are trivially typable, since General Assumption 8.2.27 guarantees the presence of all needed decoration rewrite rules and $\overline{MSSCP(R,D)}$ contains the rules needed for typing subterms in deduced decoration rules.

2. **Subsume_deco_MSS:**
$CT^k_{DER} = CT^{k-1}_{DER}$ and the typing proofs are only transformed.
3. **Simplify_MSS_D, Compose_MSS_D, Collapse_MSS_D:**
Every $p^{:S} \in CT^k_{DER} \setminus CT^{k-1}_{DER}$ stems from a $p'^{:S'}$ in CT^{k-1}_{DER} via $\rightarrowtail^{\phi,\sigma,\omega}_D$, i.e. the typing proof of $p^{:S}$ is the one of $p'^{:S'}$ followed by the decoration rewriting step $\rightarrowtail^{\phi,\sigma,\omega}_D$.
4. **Delete, Orient_SD:**
$CT^k_{DER} \subset CT^{k-1}_{DER}$.
5. **Orient_NSD:**
Like **Simplify_D**.
6. **Deduce_MSS_DR, Deduce_MSS_RR:**
In both cases, the new terms are either an instantiation of a term in CT^{k-1}_{DER} or a reduced instantiation. In either case, the typability of the instantiation follows from Lemma 8.2.5 and the fact that the unifiers are calculated with $\mathbf{UNIF_d}$. If the instantiation is reduced via $\rightarrowtail_D$, then the typability follows immediately from the one of the instantiation.
If it is reduced via $\rightarrowtail_R$, then its typability is a consequence of Lemma 8.2.20. Remark that D_{k-1} must be confluent on all typable terms, since **Deduce_deco_MSS** is no more applicable, due to assumption 8.2.27, implying also:

$$MSSCP(R_{k-1}, D_{k-1})_{|_{reach_{D_{k-1}}(\mathcal{T}_d(\mathcal{S}_\Diamond,\mathcal{F},\mathcal{X}_\Diamond):\downarrow\emptyset)}} \subseteq_D D_{k-1}$$

Furthermore, the term to be reduced must be bottom-up typable in D_{k-1}, since D_{k-1} is confluent and we do not compose or simplify decoration rewrite rules.

□

Now, we can define a typing proof preserving proof reduction $\Rrightarrow$ according to the peak reduction Lemmas. The set of reduction rules can be found in Appendix A.2.

Lemma 8.2.29 *The proof reduction $\Rrightarrow$ is well-founded.*

Proof: This can be verified with the complexity measure c of Lemma 7.3.1. □

Theorem 8.2.30 *Let $\mathcal{P}_\infty \neq (\perp,\perp,\perp)$ be the presentation obtained from $(D_\mathcal{P}, E_\mathcal{P}, \emptyset)$ using $\mathcal{MSSC}$, s.t. General Assumption 8.2.27 is fulfilled. Let furthermore $E_\infty = \emptyset$ and all critical pairs of $D_\infty \cup R_\infty$ be in $D_* \cup R_*$. Then the initial presentation $\mathcal{P}_0$ is sort inheriting on $ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ and $D_\infty \cup R_\infty$ is Church-Rosser, type and existentially complete.*

Proof: All peak reductions have the property, that they only introduce typable new terms, provided $\overline{MSSCP(R,D)} \subseteq_D D$ and the term at the top of the peak is bottom-up typable if it is a peak between two decorated rewrite rules. Bottom-up typability results from

typability at each step k, when D_k is confluent on all typable terms, since all terms in the proof to be transformed are typable by induction hypothesis and we do not simplify nor compose decoration rewrite rules (see Lemma 8.1.7). Hence, peaks between decorated rewrite rules with variable overlap or without overlap can be reduced at any such k if additionally $\overline{MSSCP(R,D)} \subseteq_D D$ holds, i.e. especially in D_∞. Peaks with critical overlap between decorated rewrite rules are reduced when the corresponding decorated critical pairs $MSSCP(R,R)$ are calculated. Here we can be sure, due to General Assumption 8.2.27, that all decoration critical pairs are calculated and that $\overline{MSSCP(R,D)} \subseteq_D D$, giving us the needed confluence of D_k.

If we orient a rule, the only new term can be reached via decoration rewriting from an old one, i.e. all terms in the new proof are trivially typable. Deleting an equation leads to a proof with less terms and preserves therefore also typability. All other proof reductions have the property, that any new term can be reached from an old one by decoration rewrite steps. Consequently, typability of these terms follows as in the orientation case.

Now that we know the typability of all terms in proofs in $\mathcal{P}_\infty$, we can reach sort inheritance, the Church-Rosser property, type and existential completeness as in the proof of Theorem 8.2.9. □

The restriction on the completion strategies, especially the one on priority for $CP(D,D)$, may often lead to divergency. However, there are some theories for which this part of the completion terminates, as the following example illustrates:

Example 8.2.31 *Let* $\mathcal{S} = \{A, clist0, clist1, clist2, clist3\}$, $x :: A$, $l :: clist0$, $l' :: clist1$ *and:*

$$\mathcal{P} = \left\{ \begin{array}{l} nil : clist0, \\ cons1(x,l) : clist1, \\ cons2(x, cons1(x,l)) : clist2, \\ cons3(x, cons2(x,l')) : clist3 \end{array} \right\}$$

After transformation into decoration rewrite rules we get the following set of decoration rewrite rules:

$$D_0 = \left\{ \begin{array}{l} (nil^{:s} \to nil^{:s \cup \{clist0\}} \text{ if } \{clist0\} \not\subseteq s), \\ (cons1(x^{:A}, l^{:\{clist0\}})^{:s} \\ \quad \to cons1(x^{:A}, l^{:\{clist0\}})^{:s \cup \{clist1\}} \text{ if } \{clist1\} \not\subseteq s), \\ (cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\emptyset})^{:s} \\ \quad \to cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\emptyset})^{:s \cup \{clist2\}} \text{ if } \{clist2\} \not\subseteq s), \\ (cons3(x^{:A}, cons2(x^{:A}, l'^{:\{clist1\}})^{:\emptyset})^{:s} \\ \quad \to cons3(x^{:A}, cons2(x^{:A}, l'^{:\{clist1\}})^{:\emptyset})^{:s \cup \{clist3\}} \text{ if } \{clist3\} \not\subseteq s) \end{array} \right\}$$

After calculation of all critical pairs in $CP(D,D)$, *we get* $D_\infty = D_0 \cup D_1$, *where* D_1 *is defined as the following set:*

$$\left\{ \begin{array}{l} (cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\{clist1\}})^{:s} \\ \quad \to cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\{clist1\}})^{:s \cup \{clist2\}} \text{ if } \{clist2\} \not\subseteq s), \\ (cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\emptyset})^{:\{clist2\}})^{:s} \\ \quad \to cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\emptyset})^{:\{clist2\}})^{:s \cup \{clist3\}} \\ \quad \text{if } \{clist3\} \not\subseteq s), \\ (cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\{clist1\}})^{:\{clist2\}})^{:s} \\ \quad \to cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, l^{:\{clist0\}})^{:\{clist1\}})^{:\{clist2\}})^{:s \cup \{clist3\}} \\ \quad \text{if } \{clist3\} \not\subseteq s) \end{array} \right\}$$

The sort inheritance test does not apply to D_∞ and therefore the presentation $\mathcal{P}$ is sort inheriting. Consider $\mathcal{P}' = \mathcal{P} \cup \{cons3(x, cons2(x, cons1(x, nil))) : clist2\}$. Hence, the transformation into decoration rewrite rules yields $D_0' = D_0 \cup \overline{D}_0$, where $\overline{D}_0$ is :

$$\left\{ \begin{array}{l} (cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, nil^{:\emptyset})^{:\emptyset})^{:\emptyset})^{:s} \\ \quad \rightarrow cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, nil^{:\emptyset})^{:\emptyset})^{:\emptyset})^{:s \cup \{clist2\}} \text{ if } \{clist2\} \not\subseteq s) \end{array} \right\}$$

Calculating critical pairs yields now the following rule:

$$\begin{array}{c} (cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, nil^{:\{cons0\}})^{:\{cons1\}})^{:\emptyset})^{:s} \\ \rightarrow cons3(x^{:A}, cons2(x^{:A}, cons1(x^{:A}, nil^{:\{cons0\}})^{:\{cons1\}})^{:\emptyset})^{:s \cup \{clist2\}} \\ \text{if } \{clist2\} \not\subseteq s) \end{array}$$

Now, **Detect_NonSI** *can be applied, stating that $\mathcal{P}'$ is not sort inheriting, since the term $cons3(x, cons2(x, cons1(x, nil)))$ is of sort cons2 and cons3 at the same time.*

But, as already mentioned, there are also presentations, where this decoration critical pair computation does unfortunately not terminate.

Example 8.2.32 *Let $\mathcal{S} = \{Even, Odd\}$, x::Even, y::Odd, and:*

$$\mathcal{P} = \left\{ \begin{array}{l} 0 : Even, \\ succ(0) : Odd, \\ succ(succ(x)) : Even, \\ succ(succ(y)) : Odd \end{array} \right\}$$

Then, there is an infinite number of uncomparable decoration critical pairs of the shape:

$$\begin{array}{l} succ(succ(\ldots(succ(0^{:\{Even\}})^{:\{Odd\}})\ldots)^{:\{Even\}})^{:s} \\ \rightarrow succ(succ(\ldots(succ(0^{:\{Even\}})^{:\{Odd\}})\ldots)^{:\{Even\}})^{:s \cup \{Odd\}} \\ succ(succ(\ldots(succ(0^{:\{Even\}})^{:\{Odd\}})\ldots)^{:\{Odd\}})^{:s} \\ \rightarrow succ(succ(\ldots(succ(0^{:\{Even\}})^{:\{Odd\}})\ldots)^{:\{Odd\}})^{:s \cup \{Even\}} \end{array}$$

This can be explained by the syntactical nature of decorations, which are considered to be a hard *part of the syntax of decorated terms, similar to function symbols. Hence, differences in decorations alone can lead to divergency of the completion procedure.*

A different point of view is taken in Chapter 10, where we consider decorations as assertions that have an optional character (cf. Section 4.3.3).

3 Comparison

Each of the three cases treated in the last sections offer several advantages and drawbacks:

1. The standard rewriting technique gives us the compatibility with classical order-sorted completion procedures as described in [Gnaedig *et al.*, 1990]. Furthermore, the sort inheritance test can be postponed until the end of completion.

2. The layer rewriting technique gives us a similar property, but simplification by decorated rewrite rules is inhibited and critical pairs computation got quite complicated.

3. The maximally subterm sharing rewriting technique has mainly proof purposes and does not support simplification for decorated rewrite rules for the moment. In contrast to layer rewriting, it haqs additional restrictions on decoration rule subsumption and a very strict strategy, e.g. all decoration critical pairs have to be computed eagerly. In some cases, this may result in divergence of the completion procedure and therefore the loss of fairness.

4. Anyway, if the proof of sort inheritance succeeds with one of these techniques, the resulting decorated presentation may be used in order to continue with $\mathcal{OSC}$, since all deductions were correct as a consequence of the fact that any layer rewriting or maximally subterm sharing step can be simulated by standard decorated rewriting.

5. We conjecture that the three techniques can be combined into one strategy, where standard rewriting is used as long as all terms in decoration rewrite rules are flat and linear, layer rewriting for the flat case and parallel rewriting on occurrence sets as long as there are non-flat decoration rules. However, sort inheritance on typable terms must be tested each time when non-flat decoration rules are introduced.

   We conjecture that this is sound, since all terms in rules and equations stay typable as well as all terms in the transformed proofs. Furthermore, each step in a rewriting strategy can be simulated by standard rewriting sequences or one parallel rewriting step at multiple occurrences. A standard rewriting or parallel rewriting step can be replaced by two converging bottom-up layer rewriting sequences, due to Lemma B.0.1. Remark also, that we can mix the proof transformations, since their termination is provable with the same complexity measure.

Further extensions of these techniques might be the definition of simplification rules for maximally subterm sharing rewriting and an extension of the results on layer rewriting to non-flat, semi-linear terms in decoration rules, where non-linear variables may only occur in identical subterms.

4 Changing the Membership Relation

The application of the **Detect_NonSI** rule indicates the detection of a counter-example for the sort inheritance w.r.t. $\leq_{\mathcal{S}}^{syn}$, i.e. there exists a term belonging to a set of incomparable sorts $A_1, \ldots, A_n$ without a common subsort. Therefore the intersection of $A_1, \ldots, A_n$, which must be computed for the unification of n variables of sorts $A_1 \ldots A_n$, respectively, cannot be described as a sort. One solution for this problem is the restriction in membership formulas to shallow terms (cf. [Comon *et al.*, 1992]) or semi-linear ones [Uribe, 1992]. But this implies that all terms in decoration rules – also those introduced by **Orient** – must have this property. Clearly, this restriction is not possible with dynamic sorts as in G-algebras and the other frameworks considered in this thesis.

Assume that the valid decorated term $t^{:S}$ with $A_1, \ldots, A_n$ (as before) ib S, is a detected counter-example for the sort inheritance of $\mathcal{P}$ w.r.t. $\leq_{\mathcal{S}}^{syn}$. Then the definition of models in G-algebra forces the interpretation of this term to be in both denotations of A and B. Hence, we can add without changing equivalence classes a new sort $C \notin \mathcal{S}$ and the membership formula $(t' : C)$, s.t. $t' = \alpha(t^{:S})_{nd}$ for some $\mathcal{T}(\Sigma, \mathcal{X})$-assignment α, together with a variable definition $x :: C$ and two declarations $(x : A)$ and $(x : B)$, standing for $C \leq_{\mathcal{S}}^{syn} A$ and $C \leq_{\mathcal{S}}^{syn} B$ respectively. I.e. we add a new subsort and therefore have to restart the completion.

Results analogous to Proposition 7.5.1 for the detection of differences between $\leq_{\mathcal{S}}^{syn}$ and $\leq_{\mathcal{S}}^{sem}$ are Theorems 8.2.9, 8.2.13 and 8.2.30. Remark that there is no condition like $\leq_{\mathcal{S}}^{syn} = \leq_{\mathcal{S}}^{sem}$ and therefore we can be sure that $\mathcal{P}_\infty = (\bot, \bot, \bot)$ whenever $\leq_{\mathcal{S}}^{syn} \neq \leq_{\mathcal{S}}^{sem}$ and $\mathcal{P}$ is not sort inheriting at the same time.

Chapitre 9

Related Work, Discussion and Conclusion

In this section, we compare our approach with similar approaches trying to overcome the problems of regularity, sort-decreasingness and dynamic typing. For this purpose, we distinguish between syntactic and semantic sort approaches, where semantic sorts means that equal terms are of equal sorts by definition and syntactic sorts means that membership in a sort is a purely syntactic property. After discussion of the different alternatives and several examples, we conclude from our results on decorated completion for G-algebras.

1 Syntactic Sort Approaches

We first come back in this section on the relation between sort inheritance and regularity, then we compare our own completion approach with several others. Then, we prove that the completion presented here subsumes the completion described in [Gnaedig *et al.*, 1990] for OBJ-3 specifications. Before we close the section mentioning other approaches, we also compare with the signature extension approach from [Chen et Hsiang, 1991].

1.1 Sort Inheritance vs. Regularity

Sort inheritance can be interpreted as an extension of regularity, which means that each valid term has a unique least sort. In fact, after the extension of $\Sigma = (\mathcal{S}, \mathcal{F})$ to $\Sigma_\Diamond = (\mathcal{S}_\Diamond, \mathcal{F})$, sort inheritance is nothing else than 'semantic' regularity for $\Sigma_\Diamond$, i.e. regularity w.r.t. semantic sorts (see [Werner, 1993]). However, this depends heavily on the completing sorts in $\Sigma_\Diamond$ and therefore should not be mixed up with semantic Σ-regularity.

The reason why we need something like regularity is hidden in the unification used for critical pair calculation. When we try to resolve a peak in a proof based on rewriting steps, we need to unify two rules left-hand sides and introduce a new equality based on the right-hand sides of the two rules instantiated by the unifier, which should solve the peak, as in the classical case. The problem arises here when we unify two variables with incomparable sorts having a common subsort. This yields a new variable in the substitution image, as the following example illustrates :

Example 9.1.1 *Let* $\phi_1 : f(x^{:\{A\}})^{:\emptyset} \to g(x^{:\{A\}})^{:\emptyset}$ *and* $\phi_2 : f(y^{:\{B\}})^{:\emptyset} \to y^{:\{B\}}$ *be two decorated rewrite rules in* R. *Then* $\mathcal{U} = (f(x^{:\{A\}})^{:\emptyset} \cong_d^? f(y^{:\{B\}})^{:\emptyset})$ *has the following principal solution*

$\sigma = \{x^{:\{A\}} \mapsto z^{:\{(A,B)\}}, y^{:\{B\}} \mapsto z^{:\{(A,B)\}}\}$, *assuming* $A \bowtie_{\mathcal{S}}^{syn} B$, $\exists C \in \mathcal{S} : C \leq_{\mathcal{S}}^{syn} A, B$ *in the current presentation* $\mathcal{P}$.

Therefore the equality

$$\phi^* \; : \; g(z^{:\{(A,B)\}})^{:\emptyset} = z^{:\{(A,B)\}}$$

should allow the peak $g(t^{:\{A,B\}})^{:\emptyset} \hookleftarrow_{\phi_1} f(t^{:\{A,B\}})^{:\emptyset} \hookrightarrow_{\phi_2} t^{:\{A,B\}}$ *to converge, i.e. it should make the left and the right terms equal. But without completing sorts, neither* $\{z^{:\{(A,B)\}} \mapsto t^{:\{A,B\}}\}$ *nor anything equivalent can be expressed.*

Furthermore, completing the sort structure also implies that decorated unification with $\mathbf{UNIF_d}$ becomes unitary. However, the sort inheritance notion is more general than syntactical regularity discussed in [Schmidt-Schauß, 1987], [Smolka *et al.*, 1989] or [Waldmann, 1989b]:

1. Sort inheritance takes equalities into account and so may be called a 'semantical' property. Thus, we have undecidability in general.

2. There are non-regular, but sort inheriting specifications like

   $$((\mathcal{S} = \{A, B, C\}, \mathcal{F} = \{a, c\}), \mathcal{P} = \{z :: C, z : A, z : B, c : C, a : A, a : B\})$$

   with $ar(a) = ar(c) = 0$, where a is a counterexample for regularity.

3. The construction of $\mathcal{S}_\Diamond$ is very close to the transformation of non-regular signatures into regular ones (see [Schmidt-Schauß, 1987]). However, due to the fact that sort inheritance is semantical, the transformation cannot be completely performed a priori.

At the semantic level, sort inheritance and completing sorts can be seen as a restriction of models of the current specification. Every $\Sigma_\Diamond$-algebra $\mathcal{A}$ is required to interpret $\langle T \cup T' \rangle \in \mathcal{S}_\Diamond$ as intersection of the interpretations of $\langle T \rangle$ and $\langle T' \rangle$. Homomorphisms are restricted accordingly to agree with sort intersection. Hence, we refine the notion of algebras :

Definition 9.1.2 *A* $\Sigma_\Diamond$-algebra *is a pair* $(|\mathcal{A}|, .^{\mathcal{A}})$ *of a domain* $|\mathcal{A}|$ *and an interpretation function* $.^{\mathcal{A}}$. $.^{\mathcal{A}}$ *itself is composed of interpretations for each symbol in* $\mathcal{S}$ *and* $\mathcal{F}$, *such that :*

1. $\forall A \in \mathcal{S}$, *the interpretation of* A, $A^{\mathcal{A}}$ *is a non-empty set, s.t.* $\Omega^{\mathcal{A}} = |\mathcal{A}|$

2. $\forall f \in \mathcal{F}$, *the interpretation of* f, $f^{\mathcal{A}}$ *is a partial function* $f^{\mathcal{A}} : |\mathcal{A}|^{ar(f)} \rightarrow |\mathcal{A}|$

The interpretation of sorts in $\mathcal{S}_\Diamond$ *follows from the one of* $\mathcal{S}$ *in the following way :* $\forall S \in \mathcal{S}_\Diamond$: $S^{\mathcal{A}} = \bigcap_{A \in S} A^{\mathcal{A}}$.

The definition of models does not change. Remark that non-emptiness of sorts in $\mathcal{S}$ implies also the non-emptiness of sorts in $\mathcal{S}_\Diamond$, since for every sort $S \in \mathcal{S}_\Diamond$, we know that there is a $A \in \mathcal{S}$, s.t. $\langle A \rangle \leq_{\mathcal{S}_\Diamond}^{syn} S$.

A $\Sigma_\Diamond$-*homomorphism* is a G-algebra Σ-homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$ satisfying $\forall \langle S \rangle \in \mathcal{S}_\Diamond$: $h(S^{\mathcal{A}}) = \bigcap_{A \in S} h(A^{\mathcal{A}})$.

It can easily be shown that there exists a free construction for all G-algebras to $\Sigma_\Diamond$-algebras with completing sorts and therefore initial as well as free models are preserved. Adding these completing sorts is rather easy. Proving the emptiness of all other possible sort intersections is the duty of the **Detect_NonSI** rule in our completion process. Since we deal with semantical sorts, we cannot expect a decidable test.

1.2 Retracts are Superfluous

As a consequence of working with semantical sorts, the retracts defined in [Goguen et Diaconescu, 1992; Jouannaud *et al.*, 1992] in order to handle syntactically ill-formed terms are superfluous. Clearly, our results guarantee that the needed sort will appear eventually in the decoration of the syntactically ill-formed term, if and only if it is semantically well-formed. The following example issued from [Goguen et Diaconescu, 1992] illustrates this:

Example 9.1.3 *Let $\mathcal{P}$ be:*

$$\left\{\begin{array}{llll} x :: Nat, & y :: NeStack, & z :: Stack, & y : Stack, \\ empty(z) : Stack, & push(x,z) : NeStack, & top(y) : Nat, & pop(y) : Stack, \\ top(push(x,z)) = x, & pop(push(x,z)) = z & & \end{array}\right\}$$

Then $\mathcal{P}$ will be translated into:

$$\begin{array}{rcll} empty(z^{:\{Stack\}})^{:s} & \rightarrow & empty(z^{:\{Stack\}})^{:s\cup\{Stack\}} & \text{if } \{Stack\} \not\subseteq s \\ push(x^{:\{Nat\}}, z^{:\{Stack\}})^{:s} & \rightarrow & push(x^{:\{Nat\}}, z^{:\{Stack\}})^{:s\cup\{NeStack\}} & \text{if } \{NeStack\} \not\subseteq s \\ top(y^{:\{NeStack\}})^{:s} & \rightarrow & top(y^{:\{NeStack\}})^{:s\cup\{Nat\}} & \text{if } \{Nat\} \not\subseteq s \\ pop(y^{:\{NeStack\}})^{:s} & \rightarrow & pop(y^{:\{NeStack\}})^{:s\cup\{Stack\}} & \text{if } \{Stack\} \not\subseteq s \end{array}$$

$$\begin{array}{rcl} top(push(x^{:\{Nat\}}, z^{:\{Stack\}})^{:\emptyset})^{:\emptyset} & = & x^{:\{Nat\}} \\ pop(push(x^{:\{Nat\}}, z^{:\{Stack\}})^{:\emptyset})^{:\emptyset} & = & z^{:\{Stack\}} \end{array}$$

Simplifying the equalities with decoration rules and orienting them yields:

$$\begin{array}{rcl} top(push(x^{:\{Nat\}}, z^{:\{Stack\}})^{:\{NeStack\}})^{:\{Nat\}} & \rightarrow & x^{:\{Nat\}} \\ pop(push(x^{:\{Nat\}}, z^{:\{Stack\}})^{:\{Stack\}})^{:\{Stack\}} & \rightarrow & z^{:\{Stack\}} \end{array}$$

The completion process stops at this point and, e.g. the term

$$top(pop(push(2^{:\emptyset}, push(1^{:\emptyset}, empty^{:\emptyset})^{:\emptyset})^{:\emptyset})^{:\emptyset})^{:\emptyset}$$

cannot be typed statically on top. The normalisation with decoration rules only gives:

$$top(pop(push(2^{:\{Nat\}}, push(1^{:\{Nat\}}, empty^{:\{Stack\}})^{:\{NeStack\}})^{:\{NeStack\}})^{:\{Stack\}})^{:\emptyset}$$

But after two reduction steps we get $push(1^{:\{Nat\}}, empty^{:\{Stack\}})^{:\{Stack\}}$, which has a top decoration and is therefore meaningful in any G-algebra satisfying $\mathcal{P}$.

However, when using retracts, $top(pop(push(1^{:\emptyset}, empty^{:\emptyset})^{:\emptyset})^{:\emptyset})^{:\emptyset}$ is statically typable with retracts but non-sense in the quotient algebra of $\mathcal{P}$. Normalising it yields $top(empty^{:\{Stack\}})^{:\emptyset}$, proving that there is an algebra, in which the term does not make sense, since the top decoration is empty.

1.3 Subsumption of Sort-Decreasing Rules Approach

We now show that our decorated completion is a conservative extension of the procedure in [Gnaedig *et al.*, 1990]. In this framework, $\mathcal{S}$ is finite and function declarations translate to flat, linear decoration rules. Syntactic regularity of the signature is assumed and implies that every term has a least sort computed by an algorithm using the static signature. This algorithm is imitated in our approach by a bottom-up normalisation process with decoration rules only.

Furthermore, the procedure in [Gnaedig *et al.*, 1990] only orients rules if they are sort decreasing, which means that for every instance of a rule, the least sort of the right-hand side is smaller than the least sort of the left-hand side. Therefore, semantical and syntactical regularity coincide.

Proposition 9.1.4 *Let $\mathcal{P}$ be a syntactically regular, subsort unique presentation using only flat, linear term declarations, in a signature Σ with a finite set of sorts $\mathcal{S}$.*

Any finite completion derivation of $\mathcal{P}$, using the order-sorted extension of the classical Knuth-Bendix algorithm (cf. [Gnaedig et al., *1990]) with sort-decreasing rules, can be transformed into a finite decorated derivation without failure.*

Proof: The sort decreasingness hypothesis implies that there are no decoration critical pairs at all. Furthermore, simplifying eagerly with decoration rewrite rules allows for orientation with **Orient_SD** only, i.e. **Orient_NSD** gets superfluous. Consequently, there are no new decoration rules added during the whole decorated completion.

The critical pairs in $CP(D, R)$ are used to compute explicitely weakenings (restricted forms of decorated rewrite rules obtained by specialisation of its variables), in order to compute the same critical pairs between rules in R. Since the number of decoration rewrite rules is finite, there can only be a finite number of such weakenings for each rule. As for simplification with decoration rules, we have to assume the computation of weakenings to be performed eagerly. Remark that a weakening can be simplified if and only if the decorated rewrite rule it stems from can be simplified, since variables are not specialised during undecorated matching.

Furthermore, no new $CP(R, R)$ critical pairs are generated by this process, since all unifiers of instantiations are covered by unifiers of the originals. In fact, the existence of a new $CP(R, R)$ critical pair would imply that we forgot one in the original deduction. The eager computation of weakenings is also the reason why there are no additional collapses due to decoration rules.

Hence, we can imitate such a completion procedure in linear time with a factor limited by m^k, where m is the maximal number of leaves occurring in a left-hand side of an equation during the unsorted completion and k is the maximum of overloads of operators. This factor may be minimised, if rules and specialisations are treated as classes: undecorated terms in [Gnaedig *et al.*, 1990] are in fact representatives of their class of specialisations.

Finally, remark that sort-decreasingness implies both sort inheritance and the fact that $\leq_{\mathcal{S}}^{syn} = \leq_{\mathcal{S}}^{sem}$. $\mathcal{P}_\infty = (\perp, \perp, \perp)$ would be in contradiction with Theorem 8.2.9 and Proposition 7.5.1. □

Therefore we can say that $\mathcal{OSC}$ with the results of Section 8.2.2 corresponds with sort-decreasing completion with flat, linear term declarations, although we allow for temporary presence of non-flat, non-linear term declarations.

1.4 The Signature Extension Approach

In [Chen et Hsiang, 1991], H. Chen and J. Hsiang present an order-sorted rewriting approach that allows for ill-sorted terms obtained from well-sorted ones by application of rewrite rules using syntactically well-sorted substitutions. Together with a condition called *sort-convergence*, a critical pair lemma can be obtained.

Their completion procedure constructs a sort convergent specification via *sort enrichment*, i.e. adding new sorts and function symbols when needed. The following small example, which is taken from [Gnaedig *et al.*, 1990], illustrates this:

Example 9.1.5 *Let* $\mathcal{P} = \{a : A, b : B, c : C, y :: B, z :: C, y : A, z : A, a = b, a = c\}$.

Then orienting the equalities into $R = \{a \to b, a \to c\}$ *results in the critical pair* $b = c$, *which is irreducible and cannot be oriented. The completion algorithm in [Chen et Hsiang, 1991] now adds a new constant* d *together with the rules*

$$b \to d,\ c \to d,$$

s.t. the final rule set is

$$\{a \to d, b \to d, c \to d\}.$$

The obtained system allows for equational proofs on the initial signature, but the normal form of a term, for instance d, *may have no meaning from a computing point of view. Of course, there exists an interpretation for* d *which validates the equalities provable in the new presentation, but the models must be extended and it might not be obvious how to interprete new function symbols.*

In our approach, we get the following set of initial rules, after simplification of the equations with decoration rules and final orientation:

$$\begin{array}{lcll} a^{:s} & \to & a^{:s \cup \{A\}} & \text{if } \{A\} \not\subseteq s \\ b^{:s} & \to & b^{:s \cup \{B\}} & \text{if } \{B\} \not\subseteq s \\ c^{:s} & \to & c^{:s \cup \{C\}} & \text{if } \{C\} \not\subseteq s \\ a^{:\{A\}} & \to & b^{:\{B\}} & \\ a^{:\{A\}} & \to & c^{:\{C\}} & \end{array}$$

The obvious critical pair is $c^{:\{C\}} = b^{:\{B\}}$, *that may be oriented using a decorated recursive path ordering based on the precedence* $a > b > c$, *yielding:*

$$\begin{array}{lcll} b^{:\{B\}} & \to & c^{:\{B,C\}} & \\ c^{:s} & \to & c^{:s \cup \{B\}} & \text{if } \{B\} \not\subseteq s \end{array}$$

We can now apply **Detect_NonSI** *on the two decoration rewrite rules for* c. *Therefore, we have to add a new sort* D *to* $\mathcal{S}$ *with the term declaration* $c : D$ *and hence also* $\langle B, C\rangle$ *to* $\mathcal{S}_\Diamond$, *s.t.* $D \leq^{syn}_{\mathcal{S}_\Diamond} \langle B, C\rangle \leq^{syn}_{\mathcal{S}_\Diamond} B, C$. *Since there was no variable unification necessary for the whole completion up to this point, we can simply continue with the new sorts, after a translation of the new term declaration into a decoration rewrite rule, giving:*

$$c^{:s} \to c^{:s \cup \{D\}} \quad \text{if } \{D\} \not\subseteq s$$

Finally, the new decoration rewrite rule subsumes the two other decoration rewrite rules for c, *finishing therefore the completion without new function symbol.*

Due to this adjunction of function symbols, Hsiang's and Chen's approach looks more appropriate for proving the truth of equivalences than for functional computation. However, adding new function symbols in the described way also results in a dynamic proof reduction ordering, which may, in case of an infinite number of function symbols added during completion, not be well-founded. The authors do not give any answer to this problem, which must be solved in order to do inductive proofs using completion (like e.g. in [Bachmair, 1988]).

The following example gives a base for the comparison on a bigger specification, that can be solved in both approaches, with the difference that we don't need any new function symbols.

Example 9.1.6 *Let* $\mathcal{P}$ = {
$a : A, x :: A, x_1 :: A, x_2 :: A,$
$b : B, y :: B, y : A,$
$c : C, z :: C, z : A,$
$d : D, y' :: D, y' : B, y' : C$
$e : E, z' :: E, z' : C, z' : B$
$f(x_1, x_2) : A, g(x_1, x_2) : A, h(x_1, x_2) : A,$
$g(x, z) : D, h(x, z) : C,$
$g(y, x) : B, h(y, x) : E,$
$g(y', z') : D, h(y', z') : E,$
$f(x_1, x_2) = g(x_1, x_2), f(x_1, x_2) = h(x_1, x_2)$
}
be the initial presentation. In $\mathcal{S}_\Diamond$, *we have to add a new sort* $\langle B, C\rangle$. *Furthermore, we have to add a variable* $u :: \langle B, C\rangle$ *and the term declarations* $u : B, u : C, y' : \langle B, C\rangle, z' : \langle B, C\rangle$.

After some initial applications of **Simplify_D** *and* **Orient**, *the decorated presentation has the following structure:*

$$
\begin{array}{rcll}
f(x_1^{:\{A\}}, x_2^{:\{A\}})^{:s} & \to & f(x_1^{:\{A\}}, x_2^{:\{A\}})^{:s\cup\{A\}} & \text{if } \{A\} \not\subseteq s \\
g(x_1^{:\{A\}}, x_2^{:\{A\}})^{:s} & \to & g(x_1^{:\{A\}}, x_2^{:\{A\}})^{:s\cup\{A\}} & \text{if } \{A\} \not\subseteq s \\
h(x_1^{:\{A\}}, x_2^{:\{A\}})^{:s} & \to & h(x_1^{:\{A\}}, x_2^{:\{A\}})^{:s\cup\{A\}} & \text{if } \{A\} \not\subseteq s \\
g(x^{:\{A\}}, z^{:\{C\}})^{:s} & \to & g(x^{:\{A\}}, z^{:\{C\}})^{:s\cup\{D\}} & \text{if } \{D\} \not\subseteq s \\
h(x^{:\{A\}}, z^{:\{C\}})^{:s} & \to & h(x^{:\{A\}}, z^{:\{C\}})^{:s\cup\{C\}} & \text{if } \{C\} \not\subseteq s \\
g(y^{:\{B\}}, x^{:\{A\}})^{:s} & \to & g(y^{:\{B\}}, x^{:\{A\}})^{:s\cup\{B\}} & \text{if } \{B\} \not\subseteq s \\
h(y^{:\{B\}}, x^{:\{A\}})^{:s} & \to & h(y^{:\{B\}}, x^{:\{A\}})^{:s\cup\{E\}} & \text{if } \{E\} \not\subseteq s \\
g(y'^{:\{D\}}, z'^{:\{E\}})^{:s} & \to & g(y'^{:\{D\}}, z'^{:\{E\}})^{:s\cup\{D\}} & \text{if } \{D\} \not\subseteq s \\
h(y'^{:\{D\}}, z'^{:\{E\}})^{:s} & \to & h(y'^{:\{D\}}, z'^{:\{E\}})^{:s\cup\{E\}} & \text{if } \{E\} \not\subseteq s \\
f(x_1^{:\{A\}}, x_2^{:\{A\}})^{:\{A\}} & \to & g(x_1^{:\{A\}}, x_2^{:\{A\}})^{:\{A\}} & \\
f(x_1^{:\{A\}}, x_2^{:\{A\}})^{:\{A\}} & \to & h(x_1^{:\{A\}}, x_2^{:\{A\}})^{:\{A\}} &
\end{array}
$$

Remark that there are neither critical pairs in $CP(D, D)$, *nor* $CP(R, D)$ *(since all rules in* D *are flat) or* $CP(D, R)$ *and all rules are interreduced. The only completion rule applicable is* **Deduce_RR** *for superposing the two decorated rewrite rules, yielding after orientation and simplification with decoration rewrite rules:*

$$g(x_1^{:\{A\}}, x_2^{:\{A\}})^{:\{A\}} \to h(x_1^{:\{A\}}, x_2^{:\{A\}})^{:\{A\}}$$

The application of **Deduce_DR** *followed by* **Simplify_D**, **Deco_Norm** *and final orientation gives:*

$$
\begin{array}{rcll}
g(x^{:\{A\}}, z^{:\{C\}})^{:\{A,C,D\}} & \to & h(x^{:\{A\}}, z^{:\{C\}})^{:\{A,C,D\}} & \\
h(x^{:\{A\}}, z^{:\{C\}})^{:s} & \to & h(x^{:\{A\}}, z^{:\{C\}})^{:s\cup\{D\}} & \text{if } \{D\} \not\subseteq s \\
g(y^{:\{B\}}, x^{:\{A\}})^{:\{B\}} & \to & h(y^{:\{B\}}, x^{:\{A\}})^{:\{B,E\}} & \\
g(y'^{:\{D\}}, z'^{:\{E\}})^{:\{D\}} & \to & h(y'^{:\{D\}}, z'^{:\{E\}})^{:\{D,E\}} & \\
h(y'^{:\{D\}}, z'^{:\{E\}})^{:s} & \to & h(y'^{:\{D\}}, z'^{:\{E\}})^{:s\cup\{D\}} & \text{if } \{D\} \not\subseteq s
\end{array}
$$

Now, **Detect_NonSI** *gets applicable to the two decoration rules for* $h(y', z')$, *since* D *and* E *do not have a common subsort, but* $h(y', z')$ *is provable to be in the intersection. Hence, a*

new sort G has to be introduced, s.t. $G \leq_S^{syn} D, E$ and $h(y', z') : G$, resulting also in a new sort $\langle D, E\rangle$ in $\mathcal{S}_\Diamond$, with $G \leq_{\mathcal{S}_\Diamond}^{syn} \langle D, E\rangle \leq_{\mathcal{S}_\Diamond}^{syn} D, E$. However, the recalculation of all critical pairs doesn't change anything, since no variable of sort D was unified with one of sort E.

1.5 Other Syntactic Sort Approaches

Apart from the transformation of order-sorted equational theories into conditional unsorted theories, called relativisation, as in [Oberschelp, 1962], we would like to mention the completion of order-sorted equational specifications by transformation into many-sorted conditional specifications as proposed by H. Ganzinger in [Ganzinger, 1991b]. The used transformation is sketched in Example 3.2.3. This is a serious alternative for coping with sort-decreasingness during completion. However, as point of criticism one might mention that the examples in [Ganzinger, 1991b] show, the transformed specifications are hard to read. The absence of partial functions avoid the use with G-algebras. We do not go further into details here, since we think that this approach is more related to superposition calculi for order-sorted theories, which are the topic of Chapter 10. The discussion at the end of that chapter sheds also some light on the relationship between decorated completion and completion by in the many-sorted way.

2 Semantic Sort Approaches

In this section, we compare with the tree automata approach of [Comon, 1992]. Furthermore we discuss the relation with the T-contact method in [Werner, 1993] and the approaches of L. With (cf. [With, 1992b]), as well as the one of P. Watson and J. Dick (see [Watson et Dick, 1989]).

2.1 The Tree Automata Approach

Concurrently with the development of this work, H. Comon designed the completion of rewrite systems with membership constraints (see [Comon, 1992]). Also motivated by the failure of the critical pair lemma, his approach of the problem is to provide new deduction rules and to compute critical pairs in a fragment of second-order logic. In order to argue that our approach is not subsumed by the completion with membership constraints of [Comon, 1992], we exhibit an example that does terminate in our approach but not in the other one. Actually, this is the example already used by [Chen et Hsiang, 1991].

Example 9.2.1 *Given the rewrite system R*

$$\begin{array}{lll} y \in S & : & f(g(y)) \to b \\ & & h(x) \to l(x) \end{array}$$

where $S = \{h^i(a)\}$, the completion procedure in [Comon, 1992] generates the infinite set of rules

$$\{X \in h^i(.) \wedge x \in S \ : \ f(g(X(l^j(x)))) \to b\}_{j=0}^{\infty}.$$

(see [Chen et Hsiang, 1991])

Using our decorated approach, we transform the system into

$$\begin{array}{lcl}
a^{:s} & \to & a^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s \\
b^{:s} & \to & b^{:s\cup\{B\}} \text{ if } \{B\} \not\subseteq s \\
h(x^{:\{A\}})^{:s} & \to & h(x^{:\{A\}})^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s \\
h(y^{:\{B\}})^{:s} & \to & h(y^{:\{B\}})^{:s\cup\{B\}} \text{ if } \{B\} \not\subseteq s \\
g(y^{:\{B\}})^{:s} & \to & g(y^{:\{B\}})^{:s\cup\{B\}} \text{ if } \{B\} \not\subseteq s \\
f(y^{:\{B\}})^{:s} & \to & f(y^{:\{B\}})^{:s\cup\{B\}} \text{ if } \{B\} \not\subseteq s \\
l(y^{:\{B\}})^{:s} & \to & l(y^{:\{B\}})^{:s\cup\{B\}} \text{ if } \{B\} \not\subseteq s \\
f(g(x^{:\{A\}})^{:\emptyset})^{:\emptyset} & = & b^{:\emptyset} \\
h(y^{:\{B\}})^{:\emptyset} & = & l(y^{:\{B\}})^{:\emptyset}
\end{array}$$

Remark that $A \leq_S^{syn} B$ *and* $h > l$, $f, g > b$ *in the precedence of the used decorated recursive path ordering (see Definition 6.4.2 on how to get the decorated version)* $>_d$. *After some applications of* **Simplify** *using decoration rules and final orientations, the last two equalities become decorated rewrite rules :*

$$\begin{array}{lcl}
f(g(x^{:\{A\}})^{:\{B\}})^{:\{B\}} & \to & b^{:\{B\}} \\
h(y^{:\{B\}})^{:\{B\}} & \to & l(y^{:\{B\}})^{:\{B\}}
\end{array}$$

The computation of $CP(D, R)$ *yields now the following new decorated equation:*

$$h(x^{:\{A\}})^{:\{A,B\}} = l(x^{:\{A\}})^{:\{B\}},$$

that can be oriented as before giving an additional decoration rule:

$$\begin{array}{lcl}
h(x^{:\{A\}})^{:\{A,B\}} & \to & l(x^{:\{A\}})^{:\{A,B\}} \\
l(x^{:\{A\}})^{:s} & \to & l(x^{:\{A\}})^{:s\cup\{A\}} \text{ if } \{A\} \not\subseteq s
\end{array}$$

This results in a confluent decorated term rewriting system.

The following peak:

$$b \leftarrow f(g(h(h(a)))) \to f(g(l(l(a)))).$$

given in [Chen et Hsiang, 1991] is therefore confluent. This is because $f(g(l(l(a))))$ *is normalized using the decoration rules into*

$$f(g(l(l(a^{:\{A\}})^{:\{B\}})^{:\{B\}})^{:\{B\}})^{:\{B\}}$$

that can be rewritten into

$$f(g(l(l(a^{:\{A\}})^{:\{A,B\}})^{:\{A,B\}})^{:\{B\}})^{:\{B\}}$$

using $l(x^{:\{A\}})^{:s} \to l(x^{:\{A\}})^{:s\cup\{A,B\}}$ if $\{A, B\} \not\subseteq s$ *twice and finally into* $b^{:\{B\}}$ *using:*

$$f(g(x^{:\{A\}})^{:\{B\}})^{:\{B\}} \to b^{:\{B\}},$$

which is now applicable.

Nevertheless, a strictly more powerful language – due to the second order monadic logic fragment – is used in [Comon, 1992]. This power is needed for the specification of equality schemata, which is indeed not possible in our first-order approach. Let us consider the map function as an example:

Example 9.2.2 *Let* $\perp$ *be an independent constant,* $\mathcal{F}_1$ *be the set of unary functions in the signature and* $\overline{\mathcal{F}_1}$ *be the corresponding disjunction of contexts of the form* $f[.]_1$ *for all* $f \in \mathcal{F}_1$, *then using H. Comon's syntax yields:*

$$\begin{array}{rcl}
X \in \overline{\mathcal{F}_1} : & map(X(\perp), cons(x, l)) & = cons(X(x), map(X, l)) \\
& map(X, nil) & = nil
\end{array}$$

This is indeed not expressible in our language since it is first order. A more order-sorted specific example could not be found yet, but it is possible that due to the integrated term schematization facilities, H. Comon can transform signatures in a more sophisticated way than we could. However, we did not find such an example yet, as well as we could not show that starting from a specification using first-order terms only, we could transform such a derivation into one of our approach. The latter might be possible, since all schematizations are also expressible as flat, non-linear term declarations using new sorts.

Using the syntax of H. Comon, we can express our $CP(R, D)$ critical pairs as:

Deduce 2' $\dfrac{(x \in q \wedge \phi \ : \ l \to r) \qquad (\psi \ : \ g \to d)}{(x \in q' \wedge \phi' \ : \ l \to r)}$ if $q_{|_p} = g \wedge \phi \wedge \psi \overset{*}{\to} \phi' \wedge \sigma$ and $q' = q[\sigma(d)]_p$

This can be seen as replacement for the rule **Deduce2** in [Comon, 1992]. As this rule does not introduce new second order variables, there is no more need for **Deduce3**, if we start with a first order equality set. This is the reason why we can stay in a first-order framework while treating the same kind of problems.

However, using **Deduce2'** can obviously result in non-linear (resp. non-semi-linear) regular tree expressions, i.e. we can no more decide the intersection of two sorts. But this is necessary for constraint simplification and therefore for the decidability of unification. Consequently, we can interpret our decoration critical pairs as a semi-decision procedure for the intersection of membership constraints.

In general, one can state that due to the undecidability of intersection in non-(semi-)linear membership constraints, H. Comon has to push all $CP(R, D)$-critical pairs into new rewrite rules, calculated by **Deduce2** and **Deduce3**, instead of creating new membership constraints, which corresponds with our decoration rewrite rules. Since the variable overlaps used in [Comon, 1992] seem to cause lots of rules, we also conjecture that our approach converges more frequently. Indeed, it is less the way that critical pairs are calculated than how constraints are solved, what makes this comparison so difficult.

To conclude, the approach in [Comon, 1992] is very interesting for the term schematization facilities that it incorporates. Indeed we may wonder if seeing sorts as term schemata is the best way to cope with non-sort-decreasing rules, although it clearly allows to get rid of regularity. Nevertheless, we feel that combining the decorated rewriting approach with term schematizations can still increase expressiveness and is worth being considered.

Finally, we treat an example due to H. Comon, that comes from [Comon, 1992].

Example 9.2.3 *Let us first give the specification:*

Sorts :	A	*translated*	$u::A$
	B		$v::B$
	C		$x::C$
	D		$y::D$
			$z::D$
Subsorts :	$A \leq D$	*translated :*	$u:D$
	$B \leq D$		$v:D$
	$C \leq D$		$x:D$
Operators :	$a :\ \rightarrow A$	*translated*	$a\ :A$
	$f :\ A \rightarrow A$		$f(u)\ :A$
	$f :\ B \rightarrow C$		$f(v)\ :C$
	$f :\ C \rightarrow C$		$f(x)\ :C$
	$f :\ D \rightarrow D$		$f(y)\ :D$
	$g :\ A \rightarrow B$		$g(u)\ :B$
	$g :\ D \rightarrow D$		$g(y)\ :D$
	$h :\ D, D \rightarrow D$		$h(y,z):D$
Rules :	$f(x) \rightarrow a$	*translated*	$f(x) \rightarrow a$
	$g(y) \rightarrow h(y,y)$		$g(y) \rightarrow h(y,y)$

The decoration rules:

u	$:D$ *give*	$u^{:s}$	$\rightarrow u^{:s \cup \{D\}}$	if $\{D\} \not\subseteq s$	(1)
v	$:D$	$v^{:s}$	$\rightarrow v^{:s \cup \{D\}}$	if $\{D\} \not\subseteq s$	(2)
x	$:D$	$x^{:s}$	$\rightarrow x^{:s \cup \{D\}}$	if $\{D\} \not\subseteq s$	(3)
a	$:A$	$a^{:s}$	$\rightarrow a^{:s \cup \{A\}}$	if $\{A\} \not\subseteq s$	(4)
$f(u)$	$:A$	$f(u^{:\{A\}})^{:s}$	$\rightarrow f(u^{:\{A\}})^{:s \cup \{A\}}$	if $\{A\} \not\subseteq s$	(5)
$f(v)$	$:C$	$f(v^{:\{B\}})^{:s}$	$\rightarrow f(v^{:\{B\}})^{:s \cup \{C\}}$	if $\{C\} \not\subseteq s$	(6)
$f(x)$	$:C$	$f(x^{:\{C\}})^{:s}$	$\rightarrow f(x^{:\{C\}})^{:s \cup \{C\}}$	if $\{C\} \not\subseteq s$	(7)
$f(y)$	$:D$	$f(y^{:\{D\}})^{:s}$	$\rightarrow f(y^{:\{D\}})^{:s \cup \{D\}}$	if $\{D\} \not\subseteq s$	(8)
$g(u)$	$:B$	$g(u^{:\{A\}})^{:s}$	$\rightarrow g(u^{:\{A\}})^{:s \cup \{B\}}$	if $\{B\} \not\subseteq s$	(9)
$g(y)$	$:D$	$g(y^{:\{D\}})^{:s}$	$\rightarrow g(y^{:\{D\}})^{:s \cup \{D\}}$	if $\{D\} \not\subseteq s$	(10)
$h(y,z)$	$:D$	$h(y^{:\{D\}}, z^{:\{D\}})^{:s}$	$\rightarrow h(y^{:\{D\}}, z^{:\{D\}})^{:s \cup \{D\}}$	if $\{D\} \not\subseteq s$	(11)

The decorated rules:

$f(x) \rightarrow a$	*give*	$f(x^{:\{C\}})^{:\emptyset} \rightarrow a^{:\emptyset}$	(12)
$g(y) \rightarrow h(y,y)$		$g(y^{:\{D\}})^{:\emptyset} \rightarrow h(y^{:\{D\}}, y^{:\{D\}})^{:\emptyset}$	(13)

They become after some **Collapse**, **Simplify** *and finally* **Orient**-*steps:*

$$f(x^{:\{C\}})^{:\{C\}} \rightarrow a^{:\{A,C\}} \quad (14)$$
$$g(y^{:\{D\}})^{:\{D\}} \rightarrow h(y^{:\{D\}}, y^{:\{D\}})^{:\{D\}} \quad (15)$$

together with the decoration rule

$$a^{:s} \to a^{:s\cup\{A,C\}} \text{ if } \{A,C\} \not\subseteq s \quad (16)$$

Now **Detect_NonSI** *can be applied and shows that a is in the intersection of A and C, although there is no common subsort. Therefore the specification was shown to be non-sort inheriting. But introducing a new sort E with a new variable $u' :: E$ and the term declarations $a : E$, $u' : A$, $u' : C$ (giving $E \leq^{syn}_{\mathcal{S}} A, C$). This results in a new $\mathcal{S}_\Diamond$ containing additionally $\langle A, C\rangle$ as sort with $E \leq^{syn}_{\mathcal{S}_\Diamond} \langle A, C\rangle \leq^{syn}_{\mathcal{S}_\Diamond} A, C$ and another new variable $u'' :: \langle A, C\rangle$ and the declarations $u'' : A$, $u'' : C$ and $u' : \langle A, C\rangle$.*

Hence, we have to restart the critical pairs computation. Applying **Deduce** *to (5) and (14) gives the following critical pair:*

$$f(u''^{:\{\langle A,C\rangle\}})^{:\{A,C\}} = a^{:\{A,C\}} \quad (17)$$

This can be oriented into:

$$f(u''^{:\{\langle A,C\rangle\}})^{:\{A,C\}} \to a^{:\{A,C\}} \quad (18)$$

Applying once more **Deduce** *at (9) and (15) yields:*

$$g(u^{:\{A\}})^{:\{B,D\}} = h(u^{:\{A\}}, u^{:\{A\}})^{:\{D\}}$$

This equation can be oriented into:

$$\begin{aligned} g(u^{:\{A\}})^{:\{B\}} &\to h(u^{:\{A\}}, u^{:\{A\}})^{:\{B\}} \\ h(u^{:\{A\}}, u^{:\{A\}})^{:s} &\to h(u^{:\{A\}}, u^{:\{A\}})^{:s\cup\{B\}} \quad \text{if } \{B\} \not\subseteq s \end{aligned}$$

This finishes the completion. Remark that the last steps were independent from the introduced sorts. Consider now the peak from [Comon, 1992]:

$$f(f(h(a,a))) \leftarrow f(f(g(a))) \to a$$

This corresponds with

$$f(f(h(a^{:\{A,C\}}, a^{:\{A,C\}})^{:\{B\}})^{:\{C\}})^{:\{C\}} \leftarrow f(f(g(a^{:\{A,C\}})^{:\{B\}})^{:\{C\}})^{:\{C\}} \rightarrowtail a^{:\{A,C\}}$$

Using our approach, we get a confluent rewrite system using "only" first-order unification. The peak becomes confluent, since we can apply rule (14) in the saturated system:

$$f(f(h(a^{:\{A,C\}}, a^{:\{A,C\}})^{:\{B\}})^{:\{C\}})^{:\{C\}} \rightarrowtail a^{:\{A,C\}}.$$

Remark that the sort $\langle A, C\rangle$ only appears in the decoration of u'', a variable of this sort. Indeed, sorts in $\mathcal{S}_\Diamond \setminus \{\langle A\rangle \mid A \in \mathcal{S}\}$ cannot be added to any decorations. They only serve for evaluating conditions during matching, unification and evaluation of the condition of decoration rewrite rules. We may conclude from our comparison that instead of considering all possible term schematisation in a certain tree language, we introduce new sort constants during completion if necessary. This can be interpreted as second-order in the broadest sense.

2.2 The T-contact Method

Another approach for the case of flat linear function declarations is described by A. Werner in [Werner, 1993][21]. Using the concept of semantical sorts, A. Werner proves a critical pair lemma, that allows for syntactically ill-sorted terms. This leads to a decision procedure for order-sorted equalities over extended terms, i.e. not necessarily well-formed terms, if the rewriting system is weakly sort-decreasing. The latter is the extension of sort decreasingness to multiple rewrite steps. More formally, for all semantically well-formed terms t, t', $t : A$ (syntactically) and $t \rightarrow_R t'$ implies that there exists some t'' with $t' \stackrel{*}{\rightarrow}_R t''$ and $t'' : A$ (syntactically). Clearly, this property solves the problem of retracts.

However, if weak sort-decreasingness does not hold, the decidability cannot be obtained for all true equalities. This is due to the fact, that in this case the rewrite relation becomes undecidable, since it is restricted to semantically well-formed terms and semantical sorts are proven undecidable in general, even in confluent rewrite systems. This is not in contradiction with our results, as Example 9.2.4 shows.

Furthermore, the completion procedure given in [Werner, 1993], transforms weakly sort-decreasing rewrite systems into sort-decreasing ones, since in interreduced systems any rule that is not sort-decreasing gives a counter-example for weak sort-decreasingness in form of its left-hand side. Let us discuss this completion in full detail. Before the start of the completion procedure, any specification has to transformed into a *range unique* one, where any function symbol can only be declared to belong to exactly one range sort, but may have several different domains.

Similarly to [Chen et Hsiang, 1991], the rewriting relation is also defined for terms derived from syntactically typable terms under the rewrite relation, which uses so-called *T-substitutions*, which test only the coarity of the top symbols of images, instead of the classical order-sorted substitutions of [Gnaedig *et al.*, 1990; Schmidt-Schauß, 1987; Smolka *et al.*, 1989].

The main difference is due to the range-uniqueness of the used signature: any image of a variable $x :: A$ must be either a variable of a subsort of A or a semantically well-sorted term with a top symbol with a coarity which is a subsort of A. This allows for a critical pair lemma using also *T-contacts* of two rule's left hand sides at variable positions, but not strictly under them as in [Comon, 1992], if the replacement at the variable position is not a T-substitution.

As there are usually many variable positions in a rule's left hand side, this approach may lead to much inefficiency or even divergence. The example given in the introduction of [Werner, 1993], recalled and extended below, seems to us a better reason for term declarations than for variable overlaps.

Example 9.2.4 *Given* $\mathcal{P} = \{o : N,\ x :: N,\ x : Z,\ s(x) : N,\ y :: Z,\ z :: Z,\ sq(y) : N,\ |y| : N,\ y * z : Z,\ sqrt(x) : N,\ opp(y) : Z,\ |x| = x,\ sq(y) = y * y,\ opp(y) * opp(y) = y * y\}$. *When the equalities are oriented into:*

$$|x| \rightarrow x,\ sq(y) \rightarrow y * y,\ opp(y) * opp(y) \rightarrow y * y,$$

we can calculate a T-overlap:

$$(sq(y), |y * y|),$$

which is oriented into $sq(y) \rightarrow |y * y|$. *There are no more critical pairs or T-overlaps and consequently the four rewrite rules are confluent. Hence, the following peaks (for* $n \geq 1$*):*

$$s^n(sq(y)) \leftarrow |s^n(sq(y))| \rightarrow |s^n(y * y)|$$

[21] We know meanwhile of a more recent version, but the results are essentially the same.

can be solved by T-rewriting, yielding $s^n(y * y)$.

However, the rule $sq(y) \to y*y$ *is not sort decreasing and the term* $sq(y)$ *is a counter example for weak sort decreasingness, i.e. there are equalities like* $sqrt(y*y) = sqrt(opp(y)*opp(y))$ *with a common, unique normal form* $sqrt(y*y)$ *for both sides, but none of the three terms is syntactically typable. Clearly, the equality holds in the initial model of* $\mathcal{P}$, *since* $sqrt(sq(y)) = sqrt(y * y)$ *and the first term is syntactically typable. Unfortunately, the existence of such a term is in general undecidable.*

Remark that $y * y : N$ *is a semi-linear function declaration, that is derived automatically using our approach, when the equality* $sq(y) = y * y$ *is oriented after typing the two members using* **Simplify** *with decoration rules. In fact, we can complete* $\mathcal{P}$ *in our approach and obtain therefore the decidability of the theory of* $\mathcal{P}$.

In our approach, the initial decoration rules are:

$$\begin{array}{rll}
o^{:s} & \to o^{:s\cup\{N\}} & \text{if } \{N\} \not\subseteq s \\
s(x^{:\{N\}})^{:s'} & \to s(x^{:\{N\}})^{:s'\cup\{N\}} & \text{if } \{N\} \not\subseteq s' \\
opp(y^{:\{Z\}})^{:s'} & \to opp(y^{:\{Z\}})^{:s'\cup\{Z\}} & \text{if } \{Z\} \not\subseteq s' \\
sq(y^{:\{Z\}})^{:s} & \to sq(y^{:\{Z\}})^{:s\cup\{N\}} & \text{if } \{N\} \not\subseteq s \\
sqrt(x^{:\{N\}})^{:s} & \to sqrt(x^{:\{N\}})^{:s\cup\{N\}} & \text{if } \{N\} \not\subseteq s \\
|y^{:\{Z\}}|^{:s} & \to |y^{:\{Z\}}|^{:s\cup\{N\}} & \text{if } \{N\} \not\subseteq s \\
(y^{:\{Z\}} * z^{:\{Z\}})^{:s} & \to (y^{:\{Z\}} * z^{:\{Z\}})^{:s\cup\{Z\}} & \text{if } \{Z\} \not\subseteq s
\end{array}$$

The initial decorated equalities:

$$\begin{aligned}
|x^{:\{N\}}|^{:\emptyset} &= x^{:\{N\}} \\
sq(y^{:\{Z\}})^{:\emptyset} &= (y^{:\{Z\}} * y^{:\{Z\}})^{:\emptyset} \\
(opp(y^{:\{Z\}})^{:\emptyset} * opp(y^{:\{Z\}})^{:\emptyset})^{:\emptyset} &= (y^{:\{Z\}} * y^{:\{Z\}})^{:\emptyset}
\end{aligned}$$

After **Simplify_D**, *we get:*

$$\begin{aligned}
|x^{:\{N\}}|^{:\{N\}} &= x^{:\{N\}} \\
sq(y^{:\{Z\}})^{:\{N\}} &= (y^{:\{Z\}} * y^{:\{Z\}})^{:\{Z\}} \\
(opp(y^{:\{Z\}})^{:\{Z\}} * opp(y^{:\{Z\}})^{:\{Z\}})^{:\{Z\}} &= (y^{:\{Z\}} * y^{:\{Z\}})^{:\{Z\}}
\end{aligned}$$

Now, decorating and orienting the equalities yields:

$$\begin{array}{rll}
|x^{:\{N\}}|^{:\{N\}} & \to x^{:\{N\}} & \\
sq(y^{:\{Z\}})^{:\{N\}} & \to (y^{:\{Z\}} * y^{:\{Z\}})^{:\{N,Z\}} & \\
(y^{:\{Z\}} * y^{:\{Z\}})^{:s} & \to (y^{:\{Z\}} * y^{:\{Z\}})^{:s\cup\{N\}} & \text{if } \{N\} \not\subseteq s \\
(opp(y^{:\{Z\}})^{:\{Z\}} * opp(y^{:\{Z\}})^{:\{Z\}})^{:\{Z\}} & \to (y^{:\{Z\}} * y^{:\{Z\}})^{:\{N,Z\}} &
\end{array}$$

This is already the final presentation, since no more completion rule is applicable. Remark that top superposition decoration critical pairs are always solved.

The equality $sqrt(y * y) = sqrt(opp(y) * opp(y))$ *can be proven, since:*

$$sqrt(sq(y))^{:\downarrow\emptyset} \downarrow_{D\cup R} =_d sqrt((y^{:\{Z\}} * y^{:\{Z\}})^{:\{N,Z\}})^{:\{N\}} =_d sqrt(opp(y) * opp(y))^{:\downarrow\emptyset} \downarrow_{D\cup R} .$$

The set of all true equalities in the initial model of $\mathcal{P}$ *is decidable by Theorem 8.2.13.*

As the procedure in [Werner, 1993] doesn't construct weakly sort decreasing rewriting systems by adding new function or term declarations, this remains to be checked after completion.

Otherwise, obtaining a normal form t_{nf} of a term t that is not syntactically typable is no more equivalent to the property that t does not make sense, i.e. there might be meaningful terms with an untypable normal form. [Werner, 1993] actually contains such a test. But if this test fails, we do not have a decision procedure for the word problem even if the system is completed and therefore confluent and terminating, as in the above example. However, if this final problem could be solved, the T-contact method might be a serious alternative.

2.3 Other Semantic Sort Approaches

An approach really similar to ours is developed in [With, 1992b]. L. With gives a lemma of decidability of order-sorted unification with term declarations, based on the assumption that all Σ-*critical overlaps between term declarations* are solved, i.e. covered by already existing term declarations. Our completion procedure can be interpreted as a test for this property.

The used unification algorithm of [Schmidt-Schauß, 1987] is only complete for regular signatures, which the author obtains via a signature transformation based on a minimal complete set of unifiers. But this kind of sets is undecidable in general (in the presence of term declarations). Hence the signature transformation is not computable and testing if all Σ-critical overlaps between term declarations of an arbitrary signature Σ are solved becomes undecidable. However, if the signature is known to be regular, an extended version of the unification algorithm of [Schmidt-Schauß, 1987], working with a marking process that guarantees termination, is sufficient to decide if all Σ-critical overlaps between term declarations are solved.

Remark that our approach does not give a computable transformation either, because the completion used to check sort inheritance might not terminate. Nevertheless, if the procedure terminates, maybe after some conservative extensions of the signature (as described in Section 7.5), the signature is known to be semantically regular and the unification is therefore complete.

A last considered approach is worked out by P. Watson and J. Dick (cf. [Watson et Dick, 1989]). In order to approximate semantical sorts during completion, P. Watson and J. Dick rely on instances of equalities already generated by the completion procedure to propagate sorts. The data structures for the realization of approximated least semantical sorts, the unification algorithm and the handling of all corresponding undecidability problems are left open but we feel that they can be solved as in our approach, since the propagation of sort information can be compared to our $CP(R, D)$-critical pair computation. Furthermore, P. Watson and J. Dick propose adding intersection sorts if equal terms happen to belong to incomparable sorts, but do not give a practical algorithm for doing this.

3 Conclusion from Decorated Completion

The first contribution of this work is to give an operational semantics for G-algebra and equational deduction in an order-sorted framework. The need for retracts and sort-decreasingness disappears in our approach, which is more powerful than previous approaches such as [Gnaedig *et al.*, 1990; Waldmann, 1992], in the sense that every completion process that terminates in these frameworks also terminates with ours.

The second interest of our approach is to formalize the notion of decorations. Decorated terms appear to be quite adequate to deal with membership declarations coming either from variable declarations, or term declarations. In fact decorations are exactly what is needed at run time for recording sort updates. The operations of matching and unification proposed in this paper are limited as much as possible to a local use of this decoration information. As a

consequence, this gives a theoretical model for the implementation of dynamic types in a quite efficient way via for example the use of jungle (cf. [Hoffmann et Plump, 1988]) rewriting to implement decorated terms.

The third contribution is to provide a relation with the notion of deduction with constraints (see e.g. [Kirchner *et al.*, 1990; Nieuwenhuis et Rubio, 1992a]). In the same vein, H. Comon designed the completion of rewrite systems with membership constraints [Comon, 1992]. We feel that the notion of decorated terms should provide another attractive alternative, while keeping the interesting notion of sorts as constraints.

A promising direction for further research is hence to extend the computations on decorated terms to a more powerful language on decorations, as outlined in the next chapter.

Chapitre 10

A Decorated Superposition Calculus

In this chapter, we extend the results achieved for decorated calculi with G-algebra semantics in Chapters 5 to 9 to a more expressive framework. The goal is to reuse decorated rewriting in the case of a fragment of G^1-logics in order to give another operational semantics to the basic paramodulation calculus for cased clauses.

G-algebras provided us with partial functions and dynamic (sub-)typing. This gives us already a quite comfortable formal language for specifying and programming. Nevertheless, the set of sorts is finite in G-algebra and the used language is strictly first-order. The first restriction disallows parametric types, which are very common in existing imperative programming language like EIFFEL, C++ and in a very restricted form in ADA. The second restriction avoids functions and sorts as parameters, which is possible in much simpler languages as the ones just mentioned, e.g. in C. Both restriction are dropped by the use of G^1-logic. In this chapter we concentrate on types as parameters. Although G^1-logic is second-order, we focus more on the handling of parametric types in the following, leaving functions as arguments undiscussed. The latter represents an interesting direction for future work.

Unfortunately, subsorting calculi for parametric types are easily undecidable. We therefore treat this problem as independent issue that is beyond the scope of this thesis. We assume instead the existence of a constraint solver for conjunctions of sort inclusions of the shape $\bigwedge_{i\in[1..n]} s_i \subseteq^? t_i$, where $\subseteq$ stands for sort inclusion, which dhould not to be confused with matching equations. Such solvers can be found, e.g., in [Smolka, 1989; Hill et Topor, 1990; Fuh et Mishra, 1988]. More general subtyping calculi may be derived using saturation calculi for transitive relations [Bachmair et Ganzinger, 1994b].[22]

A typical program that we would like to handle is as follows:

Example 10.0.1 *Assume we want to define a hash function for lists. Hence, we start with a short axiomatisation of lists built with* $\mathbf{F}_{list} = \{cons, nil\}$ *over elements in a set* z^1*:*

$$nil(z^1) \in list(z^1)$$
$$cons(x^0, y^0) \in list(z^1) \quad \Leftarrow x^0 \in z^1 \wedge y^0 \in list(z^1)$$

Remark that giving z^1 *as an argument to nil may avoid erroneous use of empty lists. Let natural numbers (set Nat) be given as the set of terms* $s^n(0)$*, together with addition (plus) and modulo (mod) function. Next, we define the hash function hash as element-wise sum modulo a natural number k, assuming a hash function hash returning natural numbers to be given for every set s that is not a list. We assume that this requirement is checked syntactically in advance. Another*

[22] This is in fact an on-going PhD thesis on its own at MPI Saarbrücken.

(partial) version of this is to use a membership condition instead to state $hash(z^1) \in Nat$ *each time hash is defined for all members of some set* z^1*, preferably when* z^1 *is defined. Hence,* $hash(z^1) \in Nat$ *can then be used as condition whenever it is required for the soundness of an axiom like the first one for hash below:*

$$\begin{aligned} &hash(y^0) \in Nat && \Leftarrow y^0 \in list(z^1) \\ &hash(nil(z^1)) = 0 \\ &hash(cons(x^0, y^0)) = x'^0 && \Leftarrow x^0 \in z^1 \wedge y^0 \in list(z^1) \wedge x'^0 \in Nat \\ & && \quad \wedge mod(plus(hash(x^0), hash(y^0)), k) = x'^0 \end{aligned}$$

Now, it is possible to use this definition with lists of boolean values, rationals, characters, etc., without changing a line of it, provided the hash function is defined for the elements.

Parametric subsorting comes into role, when we define arrays as lists by a subset declaration $array(z^1) \subseteq list(z^1)$ *(cf. Example 2.3.14) plus specific operations on arrays. Then, it is no more necessary to define hash on arrays, since it is inherited from lists.*

A first goal for this chapter is to achieve the same result as Theorem 4.3.13 for the cased clause calculus: refutational completeness. We start with the construction of an ordering on set terms, before we come to decorated terms and clauses, for which we give transformations to the cased clause calculus and back, leading to a decorated saturation calculus. A reduction of decorated clauses to cased clauses is defined, in order to simplify the proof of refutational completeness. Next we look at term rewriting systems that can be extracted from saturated presentations. These systems help us then to give a completeness result for a sort inheritance test. We close the chapter with examples. In what follows now, we require the following:

General Assumption 10.0.2 *Let* $\mathbf{P}$ *be a* G^1*-presentation.*

1. *All terms* t *of sort* s_1 *exist, i.e.* $\mathbf{P} \models_{G^1} EX\ t$.
2. $\mathbf{P}$ *defines a* subset relation *through the clauses:*

$$x^1 \subseteq y^1 \Leftarrow choose(x^1) \in y^1,$$
$$z'^0 \in y^1 \Leftarrow z'^0 \in x^1 \wedge x^1 \subseteq y^1.$$

   No other clauses contain a variable of sort s_0'.
3. *All clauses in* $\mathbf{P}$ *containing* $\subseteq$ *are of the shape* $(t^1 \subseteq s^1 \Leftarrow \Gamma)$*, where* Γ *contains only atoms of the shape* $A \subseteq B$*, i.e. the subset calculus is independent from the rest in* $\mathbf{P}$.
4. *There are no equations nor references nor dereference operators on set terms and references to set terms, respectively.*
5. *There are no set valued operators taking individual terms as arguments.*

The main purpose of the first restrictions is to avoid decorations for set terms in decorations. The second and third force an independent subset calculus, which we push into the constraint solver. The fourth is there to inhibit the need of equational reasoning inside decorations. In a many-sorted framework, these deductions would not create any problems, however, since we have subsets in G^1-logic, we have to avoid such interferences of rewriting and subset computations. Hence, we can separate this problem from the more general one of saturating clauses with decorated terms.

Syntax. Due to the number of different theories and calculi that we already presented we are getting into conflicts concerning the used symbols, e.g., S is used as a set of sorts symbol in G-algebra and the corresponding decorated completion, but in Section 4.3.3 we also used it for assertions, which are close but not exactly the same. To be concise, we adapt the following naming convention. We use A, B for set terms, $s, t, s', t', \ldots$ for individual terms, u, p for occurrences (or positions), C for clauses, $D, D', D'', \ldots$ for decorations, $L, L', L'', \ldots$ for atoms, Γ for multisets of atoms, $\mathbf{P}, \mathbf{P}', P_0, P_1, \ldots, \mathcal{P}_0, \mathcal{P}_1, \ldots$ for sets of clauses, S, S', S'' for assertions, T, T', T'' for constraints, $\vdash_K$ for a step of inference in the calculus K and $\models_{Log}$ for the satisfaction/entailment relation in the logic Log. Furthermore, we redefine $\mathcal{S}$:

Definition 10.0.3 *The set of ground terms in s_1 be written $\mathcal{S}$ and the $\mathcal{S}(\mathbf{X})$ be the set of all set terms in s_1.*

Remark that $\mathcal{S}$ is similar to G-algebra sorts and not to confuse with the set of sorts $\mathbf{S}$ in the R^n-/G^n-type system. Note also that we do not use capital letters for set variables.

1 Subsets, Sort Inheritance and Decorated Terms

Extraction of a subset structure from P. As for G-algebras, we assume the subset ordering $\subseteq$ on the set terms (which take over the role of G-algebra sorts) to be extracted from the initial presentation $\mathbf{P}$. In what follows we assume that this extraction has already taken place and the used clauses are discarded from $\mathbf{P}$. In contrast to Chapters 5 to 9, we want to handle parametric ordered sorts in the style of [Hanus, 1991], i.e. we do not fix the class of sort structures but assume the existence of a constraint solver for formulas of the shape $A \subseteq^? B$.

General Assumption 10.1.1 *Let SNF denote the skolemization function, which replaces free variables by new constants.*

- *The ordering $\subseteq$ on set terms does not contain cycles of length more than one, i.e. if $x_1 \subseteq x_2 \subseteq \ldots \subseteq x_n = x_1$, then $n = 1$.*
- *For all set terms $A_i, B_i \in \mathcal{S}(\mathbf{X})$, $i \in [1..m]$, there is a solver for constraints of the shape $\tilde{\exists}(\bigwedge_{i\in[1..m]} A_i \subseteq^? B_i)$, where $\tilde{\exists}$ is the existential closure over all (set) variables of the formula, which is implicit in the following.*
- *For all finite sets $D \subseteq \mathcal{S}(\mathbf{X})$ of set terms, it is decidable whether all $B \in D$ have a common subsort for all variable assignments in all models, i.e. there is a solver for $\exists A \in \mathcal{S}(\mathbf{X}), (\tilde{\forall})_{Var(D)}(A \subseteq^? B)_{B\in D}$, where $(\tilde{\forall})_{Var(D)}$ is the universal closure over all (set) variables in D. We abbreviate this constraint by $(A \subseteq^? SNF(B))_{B\in D}$, where SNF stands for* Skolem normal form. *Let $ncs(D)$ be true if there is no common subset A for all (universally closed) B with $B \in D$, i.e. the last constraint has no solution. This corresponds with the decidability of the satisfiability problem in the $\exists\forall$-fragment of the subset theory and is therefore stronger than the last requirement.*
- *Entailment for subset constraints is decidable, i.e. we can decide if the solutions of the constraint $[\![\bigvee_{i\in[1..m]} \bigwedge_{j\in J_i} A_j \subseteq^? B_j]\!]$ include the solutions of $[\![\bigvee_{i\in[1..m']} \bigwedge_{j\in J'_i} A'_j \subseteq^? B'_j]\!]$, where both constraints may possibly contain new Skolem constants, i.e. implication in the $\exists\forall$-fragment of the subset theory is decidable and therefore also equivalence and strict implication (where the inverse does not hold in all models).*

Remark that the first point corresponds with the absence of equations on set terms. If $\mathcal{S}$ is finite, then there is no problem with the requirements above. Otherwise, more sophisticated restrictions have to be developed (cf. e.g. [Smolka, 1989; Hill et Topor, 1990; Fuh et Mishra, 1988]).

Semantic set inclusion. Defining a semantic set inclusion $\subseteq^{sem}$ (cf. Chapter 5) in an arbitrary G^1-logical program $\mathbf{P}$ as the relation $s \subseteq^{sem} t$ whenever $\mathbf{P} \models_{G^1} x \in s \Leftarrow x \in t$, results in general in undecidability of the subset relation. However, our general assumptions and the shape of G^1-logic formulas avoid these problems, since variables of sort s'_0 may not occur in equations and therefore e.g. $x : B \simeq tt$, $f(x) : A \simeq tt$ and $f(x) \simeq x$ do not entail $B \subseteq A$, although we can derive $x : A \simeq tt$. Note that x ranges only over s_0, not over s'_0. Consequently, the non-standard elements of B may not be included in A.

Expressing a subset relation $s \subseteq t$ in our framework is therefore equivalent to give a clause of the form $x \in t \Leftarrow x \in s$, where x is of sort s'_0. The identity of $\subseteq$ and $\subseteq^{sem}$ is then forced by condition 3 of General Assumption 10.0.2, since variables of sort s'_0 are not allowed in equations in the head of clauses. Hence, atoms of the form $x \in s$ with x of sort s'_0 are only provable using clauses of the form $x \in s' \leftarrow x \in t$, s.t. s is an instance of s', which can be seen easily from the deduction rules of G^1-logics. Therefore, we conclude that the set theoretic axioms are not needed in this restricted fragment of G^1-logics:

Corollary 10.1.2 *If General Assumption 10.0.2 is satisfied, then:*

- **Extensionality** *is subsumed by* **PartialReflex**.
- **Ref** *and* **Deref** *can never be applied to ground references to set terms and ground set terms, respectively.*
- **Choice** *can only be followed by applications of the subset rules, i.e. clauses defining* $\subseteq$, *using* **Cut**.

Proof: The first and the last point follow from the fact hat $\subseteq$ is cycle-free and the restriction on variables of sort s'_0. The second one is a consequence of the absence of reference and dereference operators on set terms and references to set terms, respectively. □

Hence, since we are not interested in theorems of the form $choose(A) \in B$ (this is by Corollary 10.1.2 equivalent to $A \subseteq B$ under General Assumption 10.0.2), we can discard all set theoretic axioms. Remark that the set theoretic axioms were introduced mainly for semantic reasons, in particular to get intuitively easy class of models w.r.t. which we can define completeness in the style of universal algebra. The axioms do not contribute to the operational semantics, which is given by the equational rules, but they serve for model theoretic issues.

Decorated Terms and Sort Inheritance. Similar to Chapter 5, we have to overcome the difficulty of semantic sorts, in order to compute effectively in G^1-logics. As with G-algebra, we opt for sort-inheritance (SI for short), in order to ensure decidability of unification and therefore feasibility of completion. We call a G^1-presentation $\mathbf{P}$ sort inheriting (w.r.t. the extracted subset ordering $\subseteq$), if for any individual term t in s_0, $\forall D \subseteq \mathcal{S}(\mathbf{X})$:

$$(\forall A \in D, \mathbf{P} \vdash_{G^1} t : A) \Rightarrow (\exists C \in \mathcal{S}(\mathbf{X}), (\tilde{\forall})_{Var(D)}(C \subseteq A)_{A \in D})$$

As we outlined above, $\subseteq$ is the same as $\subseteq^{sem}$ under our general assumptions. Hence, it is sufficient to test SI w.r.t. $\subseteq$, the decidable relation. This motivates to write SI without precising the used relation, which implicitly means SI w.r.t. $\subseteq$. Sort inheritance is undecidable in general, but a constructive test computing terms that destroy SI is developed later. Simple restrictions such as syntactic regularity and sort-decreasingness of [Smolka *et al.*, 1989; Gnaedig *et al.*, 1990; Waldmann, 1992] are sufficient to ensure SI.

Decorated terms are defined as in Chapter 5, except that we have now a sort and a set attached to an individual variable, and a sort only for set variables. We give the refined definition in order to be clear in the following. In fact, this definition is all we need from Chapter 5.

Definition 10.1.3 *Let $\Sigma = (\mathbf{S}, \leq_\mathbf{S}, \mathbf{F}, \mathbf{R})$ be a G^1-signature, $\mathbf{X}$ be a denumerably infinite set of $\mathbf{S}$-sorted variables and $\mathbf{V}$ be a set of variables ranging over finite subsets of $\mathcal{S}(\mathbf{X})$. Let furthermore $\mathcal{X}$ be a $\mathcal{S}(\mathbf{X})$-family of variable sets partitioning $\mathbf{X}_{s_0}$ and for all $x \in \mathbf{X}_{s_0}$, $set(x)$ stand for the ground set term denoting the partition x belongs to.*

A decoration *is either a finite subset of $\mathcal{S}(\mathbf{X})$, called basic decoration, or the union of a basic decoration and a variable v in $\mathbf{V}$, representing a basic decoration. Then, a decorated $(\Sigma, \mathcal{X})$-term, or* decorated term *for short if $(\Sigma, \mathcal{X})$ is clear from the context, is:*

1. *either a pair $(x, \emptyset)$ for a set variable $x \in \mathbf{X}_{s_1}$, written $x^{:\emptyset}$ or x,*
2. *or a pair (x, D) for a variable $x \in \mathcal{X}$ and a decoration $D = \{set(x)\}$ (also called decorated variable), written $x^{:D}$,*
3. *or of the form $(f, D)((t_1, D_1), \ldots, (t_n, D_n))$, if $(t_1, D_1), \ldots, (t_n, D_n)$ are decorated $(\Sigma, \mathcal{X})$-terms, $f \in \mathbf{F}$ with $ar(f) = n$ and D is a decoration. Such a term is written $f(t_1{}^{:D_1}, \ldots, t_n{}^{:D_n})^{:D}$.*

$Var(t^{:D})$ stands for the set of variables without their decorations contained in $t^{:D}$ and as in Chapter 5, $Deco(t^{:D})$ stands for D.

For proof technical purposes it is practical to define the set of G^1-membership formulas associated with the decorations of a term, analogously to Definition 5.2.3.

Definition 10.1.4 *The* set of assertions *of a decorated term $t^{:D}$, written $assert(t^{:D})$, is:*

$$\{((t^{:D}{}_{|_p})_{nd} : A \simeq tt) \mid A \in Deco(t^{:D}{}_{|_p}) \text{ and } p \in \mathcal{O}cc(t^{:D})\}$$

An extension of the subset structure by set intersections is not performed in contrast to Chapters 5 to 9. In the rather raw transformation in the following, the same effect is achieved by additional variables used in the condition of clauses.

2 Decorated Clause Transformation

In this section, we present two transformations that convert cased clauses into decorated ones and vice versa, respectively.

From Cased to Condeco Clauses. The idea in the following is to replace any cased clause C of the form $[\![S]\!]\ L \Leftarrow \Gamma\ [\![T]\!]$ in[23] $\Xi(\mathbf{P})$ (cf. Section 4.3.2) by a constrained, decorated (for short *condeco-*) clause $\Delta(L) \Leftarrow \Delta(\Gamma)\ [\![\Delta(T)]\!]$.

Because of Assumption 10.0.2, we are sure that assertions about membership of set terms are not necessary. For membership conditions, we try to get close to the way such conditions were transformed into decorated rewriting in Theorem 6.5.1. We therefore introduce a new kind of condition $t^{:D} \urcorner A$ standing intuitively for "$t^{:D}$ must evaluate to some $t'^{:D'}$ with some $A' \in D'$, such that for all instances $A\sigma$ of A, there is some $A' \in D'$ with an instance $A'\sigma'$, s.t. $A'\sigma'$ is a subset of $A\sigma$", i.e. rewriting decorated terms plays again the role of typing.

We start with the transformation of terms, where we choose non-deterministically a decoration that is covered by the assumption of the cased clause. For individual variables, we choose additionally a unique $A \in \{B \mid (x : B \simeq tt) \in \Gamma\}$, if this set is non-empty, otherwise let A be Ω_0. Let by definition $set(x) = A$. Note that A may be a non-ground set term. Remark in the following that if $\{B \mid (x : B \simeq tt) \in \Gamma\} \neq \emptyset$, then this leads to a special condition $x :: set(x)$ for which the intuititively means that $x^{:\{A\}}$ must be instantiated by a term belonging to A. Remark that this condition is also present in the decoration of the variable, as long as it occurs in the rest of the clause. However, when the variable is discarded from the rest of the clause due to simplification, then we still have to prove that $set(x)$ is non-empty, whenever we want to use the simplified clause. This is reminded by $x :: A$.

$$
\begin{array}{rl}
\Delta(f(t_1,\ldots,t_m)) = & f(\Delta(t_1),\ldots,\Delta(t_m))^{:D}, \text{ s.t. } D \subseteq \{A \mid (f(t_1,\ldots,t_m) : A \simeq tt) \in S\} \\
\Delta(A) = & A^{:\downarrow\emptyset} \text{ if } A \text{ is a set term} \\
\Delta(f_p(t_1,\ldots,t_m)) = & f_p(\Delta(t_1),\ldots,\Delta(t_m)), \text{ if } f_p \text{ is a propositional function introduced by } .^{=} \\
\Delta(x) = & x^{:\{A\}}, \text{ such that } set(x) = A \\
\Delta(x) = & x^{:\emptyset}, x \text{ is a set variable} \\
\Delta(tt) = & tt
\end{array}
$$

To clearify this, assume $f(x) = x \Leftarrow x : A = tt, x : B = tt$ is the clause to be transformed, where x is an individual variable and f is an individual function and A, B are set constants. Then, we may choose $\Delta(x) = x^{:\{A\}}$, i.e. $set(x) = A$, leading to $\Delta(f(x)) = f(x^{:\{A\}})^{:\emptyset}$, $\Delta(A) = A^{:\emptyset}$ and $\Delta(B) = B^{:\emptyset}$. If L' is an atom in the body of the cased clause, then let $\Delta(L')$ be defined by:

$$
\Delta(L') = \begin{cases}
\Delta(s) \simeq \Delta(t) & \text{if } L' = (s \simeq t) \text{ but not } L' = (t' : A \simeq tt) \text{ for some } t', A \\
\Delta(t) \urcorner A & \text{if } L' = (t : A \simeq tt), t \notin \mathcal{X} \text{ and } A \neq set(t) \\
\Delta(x) \urcorner A & \text{if } L' = (x : A \simeq tt), x \in \mathcal{X}, set(x) = B \neq A \\
x :: A & \text{if } L' = (x : A \simeq tt), x \in \mathcal{X}, set(x) = A
\end{cases}
$$

The transformation of the head literal L is:

$$
\Delta(L) = \begin{cases}
\Delta(t)^{:v} \to \Delta(t)^{:v \cup \{A\}} & \text{if } L = (t : A \simeq tt) \\
\Delta(s) \simeq \Delta(t) & \text{if } L = (s \simeq t) \text{ but not } L = (t' : A \simeq tt) \text{ for some } t', A
\end{cases}
$$

Remark that $\Delta(t)^{:v} \to \Delta(t)^{:v \cup \{A\}}$ is just a shorthand for $\Delta(t)^{:v} \to \Delta(t)^{:v \cup \{A\}}$ if $\{A\} \not\subseteq v$, since the condition can be deduced from the added decoration on the right hand side. Remark also that L cannot be of the form $(x : A \simeq tt)$, because General Assumption 10.0.2 is satisfied and since we discard clauses used for the definition of $\subseteq$. Finally, the transformation $\Delta(T)$ of the constraint T contains $\delta(s) =^? \delta(t)$ for each $s =^? t$.

[23] Remark that we do not need $\mathbf{P}^+$, because of Corollary 10.1.2 above.

For multisets Γ of atoms, Δ gives the multiset of transformed atoms, and for equational or ordering constraints in T, Δ just distributes over all terms. Clearly, this direction of the transformation does not yield a unique result, since we may forget information present in the assertion S. However, if $S = \emptyset$, then Δ is unambiguous.

Note also that set terms obtain the empty decoration, since typing of set terms would be trivial (always Ω_1). A particularly interesting example is the translation of individual terms containing variables ranging over set terms:

$$\Delta(create(x) : list(x) \simeq tt) = (create(x^{:\emptyset})^{:v} \to create(x^{:\emptyset})^{:v \cup \{list(x)\}})$$

Remark here that set term variables in decorations are not decorated. Another interesting example is the translation of parametric set membership, as e.g. for polymorphic lists:

$$\Delta(cons(x, y) : list(z) \simeq tt \Leftarrow x : z \simeq tt, y : list(z) \simeq tt) =$$
$$(cons(x^{:z}, y^{:list(z)})^{:v} \to cons(x^{:z}, y^{:list(z)})^{:v \cup \{list(z)\}}) \Leftarrow x :: z, y :: list(z)$$

From Condeco to Cased Clauses. In the other direction, which is more interesting for the soundness proof of our calculus, we can define an injective mapping. Let $L \Leftarrow \Gamma \; [\![T]\!]$ be a condeco clause. Then let the transformation Ψ be defined as:

$$\Psi(L \Leftarrow \Gamma \; [\![T]\!]) = ([\![S]\!] \; \Psi(L) \Leftarrow \Psi(\Gamma) \; [\![\Psi(T)]\!]),$$

where S is defined by:

$$S = \{(t_{nd} : A \simeq tt) \mid t^{:\{A\} \cup D} \text{ occurs as non-variable subterm in } L \Leftarrow \Gamma \text{ or } T\}$$

$\Psi(L')$ for atoms L' is defined as follows:

$$\Psi(L') = \begin{cases} (t_{nd} : A \simeq tt) & \text{if } L' = (t^{:v} \to t^{:v \cup \{A\}}) \\ & \text{or } L' = (t^{:D} \urcorner A) \\ (x : A \simeq tt) & \text{if } L' = (x :: A) \\ L'_{nd} & \text{otherwise} \end{cases}$$

$\Psi(T)$ is defined as $\{t_{nd} =^? s_{nd} \mid (t^{:D} \simeq s^{:D'}) \in T\}$.

Corollary 10.2.1 *For all cased clauses C with empty constraints and assertions $\Psi(\Delta(C)) = C$.*

As an example, remember the condeco clause from above:

$$(cons(x^{:z}, y^{:list(z)})^{:v} \to cons(x^{:z}, y^{:list(z)})^{:v \cup \{list(z)\}}) \Leftarrow x :: z, y :: list(z)$$

Retransforming it via Ψ gives us indeed $(cons(x, y) : list(z) \simeq tt) \Leftarrow x : z \simeq tt, y : list(z) \simeq tt$.

3 The Initial Presentation, or How to Test Sort Inheritance

Given a G^1-presentation $\mathbf{P}$, the goal of this section is to define the initial set of clauses with decorated terms and the corresponding set of cased clauses, which gives the semantics to the former through the transformations Δ and Ψ. The main issue here is how to integrate a test of sort inheritance. A naive approach for a detection test for individual terms violating sort inheritance (on a set D of set terms) could be adding clauses of the form $(\Leftarrow \wedge_{A \in D \setminus \{B\}} x^{:\{B\}} \simeq$

$z^{:\{A\}}$) to $\mathbf{P}$, where B is an arbitrary element in $D \subseteq \mathcal{S}$. This basically says that there must not be an element in the intersection of all sets $A \in D$, i.e. if the goal succeeds during saturation, then there is at least one non-empty intersection. Remark that we could post-pone this test until the end of the saturation. However, $\mathcal{S}$ may be denumerably infinite, i.e. this test is as such not effective, since the number of clauses may also be infinite.

However, on the cased clause level, we may use a similar definition to give implicitly a semantics to our decorated calculus by the transformation Ψ. The saturation process together with the decorated variables will help us then to schematise the test, i.e. to reduce the number of goals to a finite subset. For the following definition of the initial clause set for the decorated clause calculus and the cased clause set giving a semantics to it, recall that we identify cased clauses without assertions nor constraints with ordinary clauses. Recall also the definition of $.^{=}$ and its inverse transformation of presentations $.^{rel}$ from Section 4.2.4.

Definition 10.3.1 *Let* $\mathbf{P}$ *be a* G^1*-presentation satisfying Assumptions 10.0.2 and 10.1.1,* $\Xi(\mathbf{P})^{=}$ *be defined as in Section 4.2 and* $\mathbf{P}_{\setminus\subseteq}$ *be* $\Xi(\mathbf{P})^{=}$ *after extraction of the set term ordering* $\subseteq$ *and discarding all literals of the form* $(t : \Omega_1 \simeq tt)$ *and clauses with a conclusion of the form* $(t : \Omega_1 \simeq tt)$ *or* $(x^0 : \Omega_0 \simeq tt)$*, where* $x^0 \in \mathbf{X}$*. Then let* P_δ *be defined as follows, assuming that* x'^0 *is a variable of sort* s'_0 *and* x^1 *is a variable of sort* s_1*:*

$$\begin{aligned} P_\delta = \quad & \{(f_\subseteq(B, A) \simeq tt) \mid \mathbf{P} \models_{G^1} B \subseteq A\} \\ \cup \quad & \{z'^0 : A \simeq tt \Leftarrow z'^0 : B \simeq tt \mid \mathbf{P} \models_{G^1} B \subseteq A \text{ and } \mathbf{P} \not\models_{G^1} A \subseteq B\} \\ \cup \quad & \{(\Leftarrow (x : A \simeq tt)_{A \in D}) \mid ncs(D), D \subseteq \mathcal{S}(\mathbf{X})\} \\ \cup \quad & \{(x^1 : \Omega_1 \simeq tt)\} \\ \cup \quad & \{tt \simeq tt\} \end{aligned}$$

The initial presentation P_0 *representing* $\mathbf{P}$ *plus sort inheritance is* $P_0 = \mathbf{P}_{\setminus\subseteq} \cup P_\delta$.
The initial decorated presentation *is defined as* $\mathcal{P}_0 = \Delta(\mathbf{P}_{\setminus\subseteq})$.

Remark that P_δ has, apart from $\{tt \simeq tt\}$, three components: $\{(f_\subseteq(B, A) \simeq tt) \mid \mathbf{P} \models_{G^1} B \subseteq A\}$ and $\{z'^0 : A \simeq tt \Leftarrow z'^0 : B \simeq tt \mid \mathbf{P} \models_{G^1} B \subseteq A \text{ and } \mathbf{P} \not\models_{G^1} A \subseteq B\}$ are clauses defining the subset relation, $\{(\Leftarrow (x : A \simeq tt)_{A \in D}) \mid ncs(D), D \subseteq \mathcal{S}(\mathbf{X})\}$ represents the sort inheritance property and $\{(x^1 : \Omega_1 \simeq tt)\}$ the existence of all set terms, as required in General Assumption 10.0.2. To illustrate this, consider the following simple example:

Example 10.3.2 *Let* $\mathbf{P}$ *be the following* G^1*-presentation:*

$$\{(EX\ x^1),\ (x^1 \subseteq y^1 \Leftarrow choose(x^1) \in y^1),\ (z'^0 \in y^1 \Leftarrow z'^0 \in x^1 \wedge x^1 \subseteq y^1),\ (A \subseteq B),\ (f(x^0) = x^0 \Leftarrow x^0 \in A)\}.$$

Remark that $\mathbf{P}$ *obviously fulfils General Assumption 10.0.2. After transformation by* Ξ*, we get:*

$$\{(x^1 \in \Omega_1),\ (x^1 \subseteq y^1 \Leftarrow choose(x^1) \in y^1),\ (z'^0 \in y^1 \Leftarrow z'^0 \in x^1 \wedge x^1 \subseteq y^1),\ (A \subseteq B),\ (A \in \Omega_1),\ (B \in \Omega_1),\ (f(x^0) = x^0 \Leftarrow x^0 \in A),\ (f(x^0) \in \Omega_0 \Leftarrow x^0 \in A)\}.$$

Note also that the clause $z'^0 \in \Omega_0 \Leftarrow z'^0 \in x^1 \wedge x^1 \subseteq y^1$ *gives* $choose(x^1) \subseteq \Omega_0$ *by* $x^1 \subseteq y^0 \Leftarrow choose(x^1) \in y^1$ *and therefore* $x^1 \subseteq \Omega_0$ *holds. Hence,* Ω_0 *is interpreted as superset of all other sets represented as terms. Now, we discard all literals of the shape* $t^1 \in \Omega_1$ *and clauses with heads of the form* $t^1 \in \Omega_1$ *or* $x^0 \in \Omega_0$*. We discard furthermore all subset rules. There remain:*

$$\{(f(x^0) = x^0 \Leftarrow x^0 \in A), (f(x^0) \in \Omega_0 \Leftarrow x^0 \in A)\}.$$

Applying $.^{=}$ *to these clauses gives:*

$$\{(f(x^0) \simeq x^0 \Leftarrow x^0 : A \simeq tt), (f(x^0) : \Omega_0 \simeq tt \Leftarrow x^0 : A \simeq tt)\}.$$

This gives us $\mathbf{P}_{\setminus\subseteq}$. *Hence,* P_0 *is:*

$$\begin{array}{ll} & \{(f(x^0) \simeq x^0 \Leftarrow x^0 : A \simeq tt),\ (f(x^0) : \Omega_0 \simeq tt \Leftarrow x^0 : A \simeq tt)\} \\ \cup & \{(f_{\subseteq}(A,B) \simeq tt),\ (f_{\subseteq}(A,\Omega_0) \simeq tt),\ (f_{\subseteq}(B,\Omega_0) \simeq tt)\} \\ \cup & \{(z'^0 : B \simeq tt \Leftarrow z'^0 : A \simeq tt),\ (z'^0 : \Omega_0 \simeq tt \Leftarrow z'^0 : A \simeq tt),\ (z'^0 : \Omega_0 \simeq tt \Leftarrow z'^0 : B \simeq tt)\} \\ \cup & \{(x^1 : \Omega_1 \simeq tt)\} \\ \cup & \{(tt \simeq tt)\} \end{array}$$

Remark that $ncs(D)$ *is not satisfied for any* $D \subseteq \{A, B\}$. $\mathcal{P}_0$ *is:*

$$\{(f(x^{:\{A\}})^{:\emptyset} = x^{:\{A\}} \Leftarrow x :: A),\ ((f(x^{:\{A\}})^{:v} \to f(x^{:\{A\}})^{:v\cup\{\Omega_0\}}) \Leftarrow x :: A)\}.$$

Now, we come to the inverse direction, from condeco clause sets to cased or G^1-presentations.

Definition 10.3.3 *The reduction* Π *of any set of condeco clauses to the corresponding set of cased clauses by* $\Pi(\mathcal{P}) = \Psi(\mathcal{P}) \cup P_\delta$, *i.e. by first transforming condeco into cased clauses with* Ψ, *then adding subsorting, sort inheritance and set term existence formulas (all in* P_δ*). Single condeco clauses are transformed by* Ψ *into cased clauses, so there is no need to define* Π *for it.*

The reduction of any set of condeco clauses to the corresponding G^1*-presentation is given by the transformation* Υ, *defined as* $\Upsilon(\mathcal{P}) = \Theta(\Pi(\mathcal{P})^{rel}$. *i.e. by first transforming it via* Π, *then retransforming propositional functions into relation symbols with* $.^{rel}$, *and finally exchanging membership in* Ω_0, Ω_1 *by existence formulas with* Θ. *Analogously, let the reduction* Υ_C *of single condeco clauses to* G^1*-clauses be given by* $\Upsilon_C(L \Leftarrow \Gamma\ [\![T]\!]) = (\Theta(\Psi(L \Leftarrow \Gamma\ [\![T]\!])^{rel}))$.

Since we have now a lot of different transformations, we think that it is the right place to give an overview: Figure 10.1 shows their relationship.

Remark that the set $\{(\Leftarrow (x : A \simeq tt)_{A\in D}) \mid ncs(D)\}$ may again be infinite, because there may be denumerably infinitely many set terms. Since the decorated calculus should be an effective calculus, i.e. working with finite[24] sets of clauses only, we do not add corresponding clauses to $\mathcal{P}_0$. Instead, we postpone the test until the end, where we may define a corresponding test on decorated rules with heads of the form $l^{:v} \to l^{:v\cup\{A\}}$. Let us state the following immediate conclusion of the definition:

Corollary 10.3.4 *For all ground atoms* L:

$$\Upsilon(\mathcal{P}_0) \models_{G^1} L \text{ iff } \mathbf{P} \cup \{(\Leftarrow (x : A \simeq tt)_{A\in D}) \mid ncs(D)\} \models_{G^1} L$$

Proof: Definition 10.3.1 gives $\Upsilon(\mathcal{P}_0) = \Theta((\Psi(\Delta(\mathbf{P}_{\setminus\subseteq})) \cup P_\delta)^{rel}) = \Theta(\Psi(\Delta(\mathbf{P}_{\setminus\subseteq}))^{rel}) \cup \Theta(P_\delta^{rel})$. Now, by Corollary 10.2.1, we know that $\Theta(\mathbf{P}_{\setminus\subseteq}^{rel})$ and $\Theta(\Psi(\Delta(\mathbf{P}_{\setminus\subseteq}))^{rel})$ are identical and $\Theta(\mathbf{P}_{\setminus\subseteq}^{rel}) \cup \Theta(P_\delta^{rel}) = \Theta((\mathbf{P}_{\setminus\subseteq}^{rel} \cup P_\delta)^{rel})$ entails, by Lemma 4.2.3, Theorem 4.2.5 and the definition of $\mathbf{P}_{\setminus\subseteq}$ and P_δ, the same ground literals as $\mathbf{P} \cup \{(\Leftarrow (x : A \simeq tt)_{A\in D}) \mid ncs(D)\}$. □

[24] We ignore schematisations here.

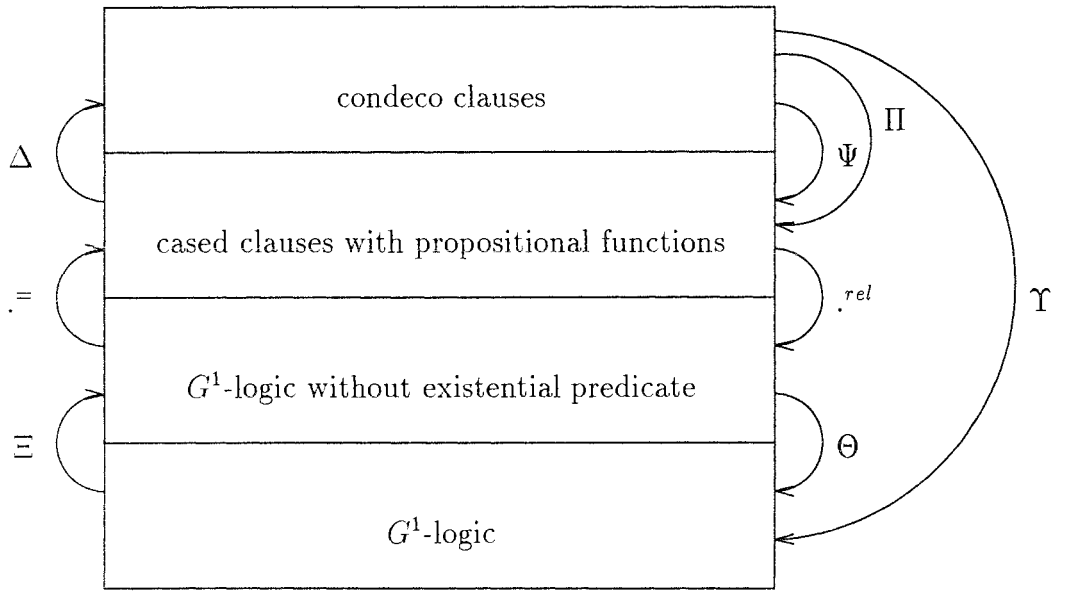


Figure 10.1: Clause (Set) Transformations in this Chapter

4 Decorated Unification and Saturation

The next step is to replace the inference rules from Section 4.3.3 by ones that work directly with decorated terms. The main problem here is the consistency of the decorations, i.e. replacing one term by another may lead to several occurrences of the same subterm, but with different cache informations. However, this does not result in any difficulties if we do not completely exploit the information in the assertions as this is the case with Δ. Remark that the use of assertions is facultative, i.e. the cache memory rules, especially *read cache*, are not applied eagerly, which would result in an optimal but very expensive strategy.

Decorated Substitutions, Matching and Unification. In the following we speak of valid decorations in a similar way as in Chapter 5, i.e. $t^{:D}$ is valid whenever $\mathbf{P} \models_{G^1} assert(t^{:D})$. We start with the adaption of decorated substitutions, matchers and unifiers, in order to show the difficulties arising with a semantics making the use of decorations optional (as assertions) and to discuss these notions w.r.t. those in Chapter 5.

Definition 10.4.1 *A* decorated substitution σ *is a mapping of set variables* σ_S *to set terms, plus a mapping* σ_D *of decorated variables to decorated terms and a subset constraint* $[\![T_\sigma]\!]$, *s.t.* $Dom(\sigma) = \{x^{:\{A\}} \mid \sigma_D(x^{:\{A\}}) \neq x^{:\{A\}}\} \cup \{y \mid \sigma_S(y) \neq y\}$ *is finite. Furthermore, if* $\sigma_D = \{x_i^{:\{A_i\}} \mapsto t_i^{:D_i}\}_{i\in[1..m]}$, *then the associated constraint* $[\![T_\sigma]\!] = [\![\bigwedge_{i\in[1..m]} \bigvee_{A'\in D_i} A' \subseteq^? \sigma_S(A_i)]\!]$ *where all unconstrained variables in* D_i *are renamed due to universal quantification, is satisfiable. Moreover,* σ_S *and* σ_{nd} *(the undecorated substitution corresponding with* σ_D*) are* Σ*-substitutions and* $ncs(Deco(\sigma(x^{:\downarrow\emptyset})))$ *is false for all* $x \in Dom(\sigma)$. *Decorated substitutions are identified with their homeomorphic extension on decorated terms.*

A decorated matching problem *is a conjunction of ordered pairs of decorated terms* $s^{:D}$ *and* $t^{:D'}$, *written* $\bigwedge_{i\in[1..m]} s_i^{:D_i} \leq^? t_i^{:D'_i}$. *A* decorated matcher *of this problem is a decorated substitution* σ, *s.t.* $\sigma(s_i^{:D_i})_{nd} = (t_i^{:D'_i})_{nd}$ *for all* $i \in [1..m]$ *and for all variables* $x^{:D''}$ *occurring*

in some $s_i^{:D_i}$, $Deco(\sigma(x^{:D''})) = \bigcup_{i\in[1..m]} \bigcup_{\omega\in O} Deco((t_i^{:D_i})_{|\omega})$, *where* O *is the set of occurrences of the variable* $x^{:D''}$ *in* $s_i^{:D_i}$.

A decorated unification problem *is a conjunction of unordered pairs of decorated terms* $s^{:D}$ *and* $t^{:D'}$, *written* $\bigwedge_{i\in[1..m]} s_i^{:D_i} =^? t_i^{:D'_i}$. *A* decorated unifier *of this problem is a decorated substitution* σ, *s.t. for all* $i \in [1..m]$, $\sigma(s_i^{:D_i})_{nd} = \sigma(t_i^{:D'_i})_{nd}$.

We start with examples to illustrate these notions:

Example 10.4.2 *Let* $\mathbf{P}_{\setminus\subseteq} = \{a : A, b : B, x :: A\}$, $\subseteq$ *be the identity on* $\mathcal{S}(\mathbf{X})$ *and:*

$$\begin{aligned}
\sigma_1 &= \{x^{:\{A\}} \mapsto a^{:\{A\}}\}\\
\sigma_2 &= \{x^{:\{A\}} \mapsto a^{:\{A,B\}}\}\\
\sigma_3 &= \{x^{:\{A\}} \mapsto b^{:\{B\}}\}\\
\sigma_4 &= \{x^{:\{A\}} \mapsto b^{:\{A\}}\}\\
\sigma_5 &= \{x^{:\{A\}} \mapsto b^{:\emptyset}\}
\end{aligned}$$

Then, σ_1 *and* σ_4 *(although* $\sigma_4(x^{:\{A\}})$ *is not valid) are decorated substitutions, but the other ones are not. Remark that* $ncs(A, B)$ *holds. Now, let* $\mathbf{P}_{\setminus\subseteq} = \{a : list(B), b : B, x :: list(y)\}$, *where* x *is an individual variable and* y *is a set variable. Let furthermore:*

$$\begin{aligned}
\sigma_1 &= \{x^{:\{list(y)\}} \mapsto a^{:\{list(B)\}}\}\\
\sigma_2 &= \{x^{:\{list(y)\}} \mapsto a^{:\{list(B)\}}, y \mapsto B\}\\
\sigma_3 &= \{x^{:\{list(y)\}} \mapsto a^{:\{list(y')\}}\}\\
\sigma_4 &= \{x^{:\{list(B)\}} \mapsto a^{:\{list(y')\}}\}\\
\sigma_5 &= \{x^{:\{list(y)\}} \mapsto b^{:\{B\}}\}
\end{aligned}$$

Then, $\sigma_1, \sigma_2, \sigma_3$ *and* σ_4 *are decorated substitutions (since the axiomatisation of* $\subseteq$ *satisfies reflexivity), but* σ_5 *only if* $[\![T_\sigma]\!]$ *is satisfiable, i.e. if there is a set term* y *such that* $B \subseteq list(y)$, *which is in general not the case, especially when* $\subseteq$ *is the identity relation.*

Note that there are differences w.r.t. Chapter 5, in that decorations of non-variable terms only play a role if they are top decorations of variable images. Another condition that was dropped is validity of variable images - it has now moved to a higher level of deduction. Instead, the only condition imposed is sort inheritance through the *ncs* condition. Let us continue our discussion with decorated matchers.

Example 10.4.3 *Let* $\mathbf{P}_{\setminus\subseteq}, \sigma_1, \ldots, \sigma_5$ *be as in the first part of the last example. Let furthermore:*

$$\begin{aligned}
\mathcal{M}_1 &= (f(x^{:\{A\}}, x^{:\{A\}})^{:\emptyset} \leq^? f(a^{:\{A\}}, a^{:\emptyset})^{:\{B\}})\\
\mathcal{M}_2 &= (f(x^{:\{A\}}, x^{:\{A\}})^{:\emptyset} \leq^? f(a^{:\{A\}}, a^{:\{A,B\}})^{:\{B\}})\\
\mathcal{M}_3 &= (f(x^{:\{A\}}, x^{:\{A\}})^{:\emptyset} \leq^? f(a^{:\{A\}}, b^{:\{B\}})^{:\{B\}})\\
\mathcal{M}_4 &= (f(x^{:\{A\}}, x^{:\{A\}})^{:\emptyset} \leq^? f(b^{:\{A\}}, b^{:\{A\}})^{:\{B\}})
\end{aligned}$$

Then, $\mathcal{M}_1$ *has the decorated matcher* σ_1, *but neither* $\mathcal{M}_2$ *has a solution, since* $ncs(A, B)$ *holds, nor* $\mathcal{M}_3$, *since there is no matcher for the corresponding undecorated matching problem.* $\mathcal{M}_4$ *has* σ_4 *as solution, again without caring about validity of the variable image. Let us also consider the parametric case from the second part of the last example, assuming* $\mathbf{P}_{\setminus\subseteq}, \sigma_1, \ldots, \sigma_5$ *to be defined as there and* $\subseteq$ *to be the identity relation:*

$$\begin{aligned}
\mathcal{M}_1 &= (x^{:\{list(y)\}} \leq^? a^{:\{list(B)\}})\\
\mathcal{M}_2 &= (x^{:\{list(y)\}} \leq^? a^{:\{list(y')\}})\\
\mathcal{M}_3 &= (x^{:\{list(B)\}} \leq^? a^{:\{list(y')\}})\\
\mathcal{M}_4 &= (x^{:\{list(B)\}} \leq^? b^{:\{B\}})
\end{aligned}$$

Then, σ_1 and σ_2 are decorated matchers for $\mathcal{M}_1$, σ_3 is a decorated matcher for $\mathcal{M}_2$ and σ_4 is one for $\mathcal{M}_3$. $\mathcal{M}_4$ does not have any decorated matcher, since $[\![list(B) \subseteq^? B]\!]$ is not satisfiable.

Remark that decorated matching as defined above is decidable by the upper bound on decorations given in the last definition. It is then in fact a version of the *optimised decorated matching* algorithm of [Hintermeier, 1992]. Note furthermore that in the parametric case $\mathcal{M}_1$, the matcher is not unique. We may choose different sorts in the decoration a.

The loss of validity in our definition of decorated unifiers allows for almost all undecorated solutions, since we may calculate the undecorated unifier and then choose the decoration $set(x)$ of a variable x to be also the decoration of the image of the variable, say t, without proving that t actually belongs to $set(x)$, i.e. the so decorated t may be invalid (cf. σ_4 in Example 10.4.2). This is one more reason to talk of soundness only in the context of an actual use of a unification algorithm in a theorem prover and not independently.

In [Hintermeier, 1992], we also gave a similar notion for optimised decorated unifiers, but since this definition is rather complex and does not give further advantages, we dropped the upper bound for decorations of variable images in unifiers completely. Hence, decorated unification has become like undecorated unification, except that it has more failure possibilities due to sort inheritance, but it is more general than the decorated unification algorithm of Chapter 5 in that it allows invalid decorations as mentioned above.

The idea is therefore to let decorated unification behave like undecorated unification except that it fails whenever a calculated unifier with valid decorations would contradict sort inheritance, where the validity of decorations in the input unification problem is a property ensured by the whole theorem prover. Failure may happen when we want to unify variables, but also when we try to unify variables and non-variable terms, as the following discussion shows. In the case of unification of a decorated variable $x^{:\{A\}}$ with a non-variable decorated term $t^{:D}$, we have three cases:

1. If there is a solution to the subset constraint $[\![(A \subseteq^? B)_{B \in D}]\!]$, then the constraint is satisfiable. Remark that decorations assert membership, i.e. a decoration $list(z)$ for a term t means that t belongs to all possible instantiations of $list(z)$. Intuitively, one may think of the intersection of all the sets represented by these instantiations.

2. If the constraint of the last case is unsatisfiable and $ncs(D \cup \{A\})$ holds, then the constraint is unsatisfiable due to sort inheritance.

3. Otherwise, there is not enough information in the decoration in order to decide the constraint using the definition of Chapter 5 and therefore the equality constraint has to be covered by the condition $x :: A$ of the clause, i.e. we may try to solve it by basic paramodulation (see *propagate decoration constraint* below). We say in this case for pragmatic reasons that the constraint is **DU**-satisfiable.

We conclude from this discussion that a decorated unification algorithm in this setting has to be sound and complete w.r.t. undecorated unification, except that it may use the information in valid decorations to fail due to sort inheritance. Remark that demanding sort inheritance and therefore this failure is completely facultative, but it is very useful to restrict the search space.

This results in the constraint solver **DU** for decorated unification associated with our decorated superposition calculus, which is shown in Figure 10.2. Remark that $|t^{:D}|$ stands for the norm of a decorated term giving a similar total ordering on all terms over the current signature.

Delete

$$\frac{[\![t^{:D} =^? t^{:D}, T]\!]}{[\![T]\!]}$$

Decompose

$$\frac{[\![f^{:D}(t_1{}^{:D_1}, \ldots, t_m{}^{:D_m}) =^? f^{:D'}(t'_1{}^{:D'_1}, \ldots, t'_m{}^{:D'_m}), T]\!]}{[\![t_1{}^{:D_1} =^? t'_1{}^{:D'_1}, \ldots, t_m{}^{:D_m} =^? t'_m{}^{:D'_m}, T]\!]}$$

Conflict

$$\frac{[\![f^{:D}(t_1{}^{:D_1}, \ldots, t_m{}^{:D_m}) =^? g^{:D'}(t'_1{}^{:D'_1}, \ldots, t'_{m'}{}^{:D'_{m'}}), T]\!]}{\bot}$$

if $f \neq g$ or $m \neq m'$.

Coalesce

$$\frac{[\![x^{:D} =^? y^{:D'}, T]\!]}{[\![x^{:D} =^? y^{:D'}, T\{x^{:D} \mapsto y^{:D'}\}]\!]}$$

if $x, y \in \mathcal{V}ar(T)$

Check*

$$\frac{[\![x_1{}^{:D_1} =^? t_1[x_2{}^{:D_2}]_{p_1}, \ldots, x_n{}^{:D_n} \cong_d^? t_n[x_1{}^{:D_1}]_{p_n}, T]\!]}{\bot}$$

if $p_i \neq \Lambda$ for some $i \in [1..n]$

Merge

$$\frac{[\![x^{:D} =^? t'^{:D'}, x^{:D} =^? t''^{:D''}, T]\!]}{[\![x^{:D} =^? t'^{:D' \cup D''}, t'^{:D'} =^? t''^{:D''}, T]\!]}$$

if $0 < |t'^{:D'}| \leq |t''^{:D''}|$.

DecoClash

$$\frac{[\![x^{:\{A\}} =^? t^{:D}, T]\!]}{\bot}$$

if $ncs(\{A\} \cup D)$

VarClash

$$\frac{[\![x_1{}^{:\{A_1\}} =^? x_m{}^{:\{A_m\}}, (x_i{}^{:\{A_i\}} =^? x_{i+1}{}^{:\{A_{i+1}\}})_{i \in [1..m-1]}, T]\!]}{\bot}$$

if $ncs(\{A_1, \ldots, A_m\})$

SortClash

$$\frac{[\![x^{:D} =^? t^{:D'}, T]\!]}{\bot}$$

if $sort(x) \neq sort(t_{nd})$

Figure 10.2: **DU** : A constraint solver for decorated unification

Again for technical reasons, let $<_V$ stand for a total ordering relation on set variables compatible with $\subseteq$ (cf. Section 5.4.1).

Remark that this is not the strict decorated unification from Chapter 5, since decorations are ignored during decomposition and variable/non-variable term unification does not fail in case of insufficient decorations. This can be explained by the fact that we do not interpret decorations in the same way as function symbols, as we did in Chapter 5. Instead, decorations get now the status of formulas with assertional character. Hence, **Delete**, **Decompose**, **Conflict**, **Coalesce**, **Check*** and **Merge** ignore decorations and behave as the corresponding undecorated rules. Remark also that $\{x^{:D} \mapsto y^{:D'}\}$ in **Coalesce** may not be a decorated substitution, due to inappropriate decorations. We ignore this issue here, since we are mainly interested in undecorated unifiers and give a purely syntactical replacement semantics to $\{x^{:D} \mapsto y^{:D'}\}$.

Note furthermore that there is no subsort rule adding new variables. This rule has been omitted mainly for technical reasons, in order to avoid the introduction of new variables and the associated set intersections - this simplifies the reduction of the decorated calculus to the cased clause calculus. As a consequence, the **VarClash** rule has become more complex. Note also that **SortClash** is used for preventing set variables from being instantiated by individual terms and individual variables from being instantiated by set terms.

Lemma 10.4.4 *Let T be a unification problem with valid decorations only. If the constraint solver* **DU** *fails due to* **DecoClash** *or then there exists a counter-example for sort inheritance. If* **DU** *fails due to* **VarClash**, *then variables belonging to sets without intersection according to sort inheritance are to be unified. Otherwise, it is sound and complete w.r.t. undecorated unification.*

Proof: The only rule changing decorations is **Merge**, but the resulting term $t'^{:D' \cup D''}$ has valid decorations in the context of the additional unification equation $t'^{:D'} =^? t''^{:D''}$. Hence, validity of decorations is preserved for any derivation using **DU**. Consequently, failure due to **DecoClash** or **VarClash** effectively detects a counter-example for sort-inheritance in the form of t_{nd} and **VarClash** detects variables belonging to sets without intersection according to sort inheritance, which are to be unified.

Otherwise, assume that the two rules are not applicable. Clearly, if we discard all decorations, the rules are identical with those of undecorated unifications. Hence, the lemma holds. □

In this sense, we have to define solutions and solvability of constraints, in order to make our saturation rules well-defined.

Definition 10.4.5 *A decorated equality constraint T is* unsatisfiable *if $T \vdash_{DU} \bot$. If T is* **DU**-*satisfiable, i.e. it has a normal form $\bigwedge_{i \in [1..m]} x_i^{:D_i} =^? t_i^{:D'_i}$, then $\sigma = \{x_i^{:D_i} \mapsto t_i^{:D'_i}\}$ is a* **DU**-solution *of the constraint, although it may not be a decorated substitution (due to the conditions on decorations), but σ_{nd} is an undecorated substitution.*

Soundness of the calculus w.r.t. G^1-logic is proved together with the soundness of the whole decorated superposition calculus by reduction to the cased clause calculus. As should be clear from [Hintermeier, 1992], we cannot prove completeness w.r.t. G^1-logic, since it is undecidable. We circumvent this problem by proving that the whole saturation calculus is sound and refutationally complete.

A matching algorithm may be derived by dropping the commutativity of equations and prohibiting instantiation of variables in the right hand side of a matching equation. This algorithm is then sound and complete w.r.t. Definition 10.4.1. Remark that $x^{:D}$ may represent a set variable, in contrast to Chapter 5. In this case D is the empty set by convention.

Decorated Superposition. Since the transformation Ψ of condeco clauses into cased clauses is performed atom by atom, independently, we may reuse the multiset expressions of equations as defined in Section 4.3.2. This allows us to simplify the conditions of the calculus considerably. Hence, let the multiset expressions of $t^{:v} \to t^{:v\cup\{A\}}, t^{:D} \triangleleft A$ and $t :: A$ be the one of $t_{nd} : A \simeq tt$. For decorated equations of the shape $t^{:D} \simeq t'^{:D'}$ let it be the one of $t_{nd} \simeq t'_{nd}$. The ordering on clauses can then be defined as in Section 4.3.2. Furthermore, when we write $s^{:D} \succ t^{:D'}$, we implicitly mean $s_{nd} \succ t_{nd}$.

Concerning the superposition rules, we may observe that membership formulas, which appear as equations of the form $s : t \simeq tt$ in the cased clause calculus of Section 4.3.3, only allow for top superposition, since the colon operator is only used at top. Hence, the result of basic superpositions into a conclusion clearly is a clause of the form $tt \simeq tt \Leftarrow \Gamma$, which is a tautology and thus redundant. Consequently, we may observe that critical pairs resulting from a decoration rule into any other kind of rule (called $CP(D, D)$ and $CP(D, R)$ in Chapter 5 is always redundant.

However, the same is not true for basic superposition into the premise. Furthermore, the upper bound in *write cache* concerning the size of the conditions that may be used to add membership assertions to the cache forces us to discard decorations above the overlay positions. In the concerned rules, we write $t^{:\downarrow_u\emptyset}[s^{:D'}]_u$ in order to indicate that the subterm of $t^{:D''}$ at occurrence u has been replaced by $s^{:D'}$ and all decorations strictly above u (or all decorations from top down to u, depending on the point of view) in $t^{:\downarrow_u\emptyset}[s^{:D'}]_u$ are empty. All other decorations not below or at u are unaffected, i.e. still those of $t^{:D''}$. Formally, this can be defined by:

$$
\begin{array}{ll}
(t^{:\downarrow_u\emptyset}[s^{:D'}]_u)_{nd} = (t^{:D''})_{nd}[(s^{:D'})_{nd}]_u, & \\
Deco((t^{:\downarrow_u\emptyset}[s^{:D'}]_u)_{|_p}) = Deco(t^{:D''}{}_{|_p}), & \text{if } p \text{ is uncomparable with } u, \\
Deco((t^{:\downarrow_u\emptyset}[s^{:D'}]_u)_{|_{u.p}}) = Deco(s^{:D'}{}_{|_p}), & \\
Deco((t^{:\downarrow_u\emptyset}[s^{:D'}]_u)_{|_p}) = \emptyset, & \text{otherwise, i.e. if } p < u.
\end{array}
$$

Altogether, we get the following inference rules (distinct cases for decorated and decoration rewrite rules).

1. *strict basic decorated superposition into decorated conclusion* (formerly $CP(R, R)$) :

$$\frac{s^{:D} \simeq s'^{:D'} \Leftarrow \Gamma' \; [\![T']\!] \qquad t^{:D''} \simeq t'^{:D'''} \Leftarrow \Gamma \; [\![T]\!]}{t^{:\downarrow_u\emptyset}[s^{:D'}]_u \simeq t'^{:D'''} \Leftarrow \Gamma, \Gamma' \; [\![T, T', t^{:D''}|_u =^? s^{:D}]\!]}$$

where $t^{:D''}|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t^{:D''}|_u = s^{:D}]\!]$ has a **DU**-solution σ, s.t.:

(a) $\sigma(t^{:D''}) \succ \sigma(t'^{:D'''}), \sigma(s^{:D}) \succ \sigma(s'^{:D'})$ and $\sigma(t^{:D''}) \simeq \sigma(t'^{:D'''}) \succ_E \sigma(s^{:D}) \simeq \sigma(s'^{:D'})$,
(b) $\sigma(s^{:D}) \simeq \sigma(s'^{:D'})$ is strictly maximal in $(s^{:D} \simeq s'^{:D'} \Leftarrow \Gamma')\sigma$ and
(c) $\sigma(t^{:D''}) \simeq \sigma(t'^{:D'''})$ is strictly maximal in $(t^{:D''} \simeq t'^{:D'''} \Leftarrow \Gamma)\sigma$.

2. *strict basic decorated superposition into decoration conclusion* (formerly $CP(R, D)$) :

$$\frac{s^{:D} \simeq s'^{:D'} \Leftarrow \Gamma' \; [\![T']\!] \qquad (t^{:v} \to t^{:v\cup\{A\}}) \Leftarrow \Gamma \; [\![T]\!]}{((t^{:\downarrow_u\emptyset}[s^{:D'}]_u)^{:v} \to (t^{:\downarrow_u\emptyset}[s^{:D'}]_u)^{:v\cup\{A\}}) \Leftarrow \Gamma, \Gamma' \; [\![T, T', t^{:D}|_u =^? s^{:D}]\!]}$$

where $t^{:D''}|_u \notin \mathcal{V}ar(t)$ and $[\![T, T', t^{:D}|_u =^? s^{:D}]\!]$ has a **DU**-solution σ satisfying:

(a) $\sigma(s^{:D}) \succ \sigma(s'^{:D'})$ and $(t^{:v} \to t^{:v\cup\{A\}})\sigma \succ_E \sigma(s^{:D}) \simeq \sigma(s'^{:D'})$,
(b) $\sigma(s^{:D}) \simeq \sigma(s'^{:D'})$ is strictly maximal in $(s^{:D} \simeq s'^{:D'} \Leftarrow \Gamma')\sigma$ and
(c) $(t^{:v} \to t^{:v\cup\{A\}})\sigma$ is strictly maximal in $((t^{:v} \to t^{:v\cup\{A\}}) \Leftarrow \Gamma)\sigma$.

3. *strict basic decorated superposition into premise* :

$$\frac{s^{:D} \simeq s'^{:D'} \Leftarrow \Gamma' \ [\![T']\!] \qquad L \Leftarrow \Gamma, L'[t^{:D''}]_u \ [\![T]\!]}{L \Leftarrow L'^{:\downarrow_u \emptyset}[s'^{:D'}]_u, \Gamma, \Gamma' \ [\![T, T', t^{:D''} =^? s^{:D}]\!]}$$

where $t \notin \mathcal{X}$ and $[\![T, T', t^{:D''} =^? s^{:D}]\!]$ has a **DU**-solution σ satisfying:

(a) $\sigma(s^{:D}) \succ \sigma(s'^{:D'})$ and $\sigma(t') \succ \sigma(t'')$ if L' is of the shape $t' \simeq t''$ and u is inside t',
(b) $\sigma(s^{:D}) \simeq \sigma(s'^{:D'})$ is strictly maximal in $(s^{:D} \simeq s'^{:D'} \Leftarrow \Gamma')\sigma$ and
(c) $\sigma(L'[t^{:D''}]_u)$ is maximal in $(L \Leftarrow \Gamma, L'[t^{:D''}]_u)\sigma$.

4. *strict basic decoration superposition into premise* :

$$\frac{(s^{:v} \to s^{:v\cup\{A\}} \Leftarrow \Gamma' \ [\![T']\!] \qquad L \Leftarrow \Gamma, t^{:D} \urcorner A' \ [\![T]\!]}{L \Leftarrow \Gamma, \Gamma' \ [\![T, T', t^{:D} =^? s^{:D}, A \subseteq^? A']\!]}$$

where $[\![T, T', t^{:D} =^? s^{:D}, A \subseteq^? A']\!]$ has a **DU**-solution σ satisfying:

(a) $(s^{:v} \to s^{:v\cup\{A\}})\sigma$ is strictly maximal in $((s^{:v} \to s^{:v\cup\{A\}} \Leftarrow \Gamma')\sigma$ and
(b) $(t^{:D} \urcorner A)\sigma$ is maximal in $(L \Leftarrow \Gamma, t^{:D} \urcorner A)\sigma$.

5. *strict basic variable decoration superposition into premise* :

$$\frac{(s^{:v} \to s^{:v\cup\{A\}}) \Leftarrow \Gamma' \ [\![T']\!] \qquad L \Leftarrow \Gamma, x :: A' \ [\![T]\!]}{(L \Leftarrow \Gamma, \Gamma' \ [\![T, T', A \subseteq^? A']\!])\{x^{:\{A'\}} \mapsto s^{:\{A\}}\}}$$

where $[\![T, T', A \subseteq^? A', x^{:\{A'\}} =^? s^{:\{A\}}]\!]$ has a **DU**-solution σ satisfying:

(a) $(s^{:v} \to s^{:v\cup\{A\}})\sigma$ is strictly maximal in $((s^{:v} \to s^{:v\cup\{A\}}) \Leftarrow \Gamma')\sigma$ and
(b) $(x :: A)\sigma$ is maximal in $(L \Leftarrow \Gamma, x :: A)\sigma$.

6. *decorate term* :

$$\frac{(s^{:v} \to s^{:v\cup\{A\}}) \Leftarrow \Gamma' \ [\![T']\!] \qquad C[t^{:D}] \ [\![T]\!]}{C[t^{:D\cup\{\rho(A)\}}] \ [\![T, T_\rho]\!]}$$

if we can find a part Γ'' of the condition of C that contains only literals that are strictly smaller than the one containing $t^{:D}$ and that covers the condition of the first clause, formally $C[t^{:D}]$ is of the form $(L''[t^{:D}] \Leftarrow \Gamma, \Gamma'')$ or $(L \Leftarrow \Gamma, L''[t^{:D}], \Gamma'')$, s.t.:

(a) $(t^{:D} \urcorner A)$ is strictly maximal in $(t^{:D} \urcorner A), \Gamma''$,

(b) for all ground substitutions σ satisfying T, there is a decorated matcher ρ for the matching equation $s^{:D} \leq^? \sigma(t^{:D})$, s.t. $T'\rho$ is true and for all $L' \in \Gamma'$, which are not of the shape $x :: B$, $(\Psi(L')\rho)_{nd} \in (\Psi(\Gamma'')\sigma)_{nd}$.

(c) the solutions for x in $[\![T, \bigvee_{A' \in D} A' \subseteq^? x]\!]$ are strictly subsumed by those for x in $[\![T, T_\rho, \rho(A) \subseteq^? x]\!]$, where x is a new variable and all unconstrained variables stemming from $C[t^{:D}]$ are renamed due to universal quantification, s.t. they are disjoint from $\mathcal{V}ar(T)$.

7. *generate well definedness rule* :

$$\frac{t^{:D} \simeq t'^{:D'} \Leftarrow \Gamma' \; [\![T']\!]}{t^{:v} \to t^{:v \cup \{\Omega_0\}} \Leftarrow \Gamma' \; [\![T', t^{:D} =^? t'^{:D'}]\!]}$$

where $[\![T', t^{:D} =^? t'^{:D'}]\!]$ has a **DU**-solution σ,
s.t. $\sigma(t^{:D}) \simeq \sigma(t'^{:D'})$ is maximal in $(t^{:D} \simeq t'^{:D'} \Leftarrow \Gamma')\sigma$.

8. *decorated equality resolution in premise* :

$$\frac{L \Leftarrow t^{:D} \simeq t'^{:D'}, \Gamma' \; [\![T']\!]}{L \Leftarrow t^{:D} \urcorner \Omega_0, \Gamma' \; [\![T', t^{:D} =^? t'^{:D'}]\!]}$$

where $[\![T', t^{:D} =^? t'^{:D'}]\!]$ has a **DU**-solution σ,
s.t. $\sigma(t^{:D}) \simeq \sigma(t'^{:D'})$ is maximal in $(L \Leftarrow t^{:D} \simeq t'^{:D'}, \Gamma')\sigma$.

9. *membership resolution in premise* :

$$\frac{L \Leftarrow t^{:D} \urcorner A, \Gamma' \; [\![T']\!]}{L \Leftarrow \Gamma' \; [\![T', (\bigvee_{A' \in D} (A' \subseteq^? A))]\!]}$$

if $[\![T', (\bigvee_{A' \in D}(A' \subseteq^? A))]\!]$ is **DU**-satisfiable. Remark that all unconstrained variables in D have to be renamed before the new constraint part is built, since they are universally quantified.

10. *variable membership resolution in premise* :

$$\frac{L \Leftarrow x :: A, \ldots, x :: A, \Gamma' \; [\![T', x^{:\{A\}} \simeq t^{:D}]\!]}{L \Leftarrow \Gamma'\{x^{:\downarrow\emptyset} \mapsto t^{:D}\} \; [\![T'\{x^{:\{A\}} \mapsto t^{:D}\}, (\bigvee_{A' \in D} (A' \subseteq^? A))]\!]}$$

if $[\![T', x^{:\{A\}} \simeq t^{:D}, (\bigvee_{A' \in D}(A' \subseteq^? A))]\!]$ is **DU**-satisfiable. Remark again that all unconstrained variables in D have to be renamed before the new constraint part is built, since they are universally quantified.

11. *decorated tautology resolution in conclusion* :

$$\underline{tt \simeq tt \Leftarrow \Gamma' \; [\![T']\!]}$$

12. *decorated tautology resolution in premise* :

$$\frac{L \Leftarrow tt \simeq tt, \Gamma' \; [\![T']\!]}{L \Leftarrow \Gamma' \; [\![T']\!]}$$

Let the condeco clause calculus above be called $\mathcal{GCD}$. In order to get an intuition for these rules, remark that the first two superposition rules correspond with *strict basic cased superposition into conclusion*, the following three with *strict basic cased superposition into premise*, all combined with *discard cache*. The *decorate term* rule is a special cased of *write cache*. The next two rules correspond with *cased G^1-equality resolution in conclusion/premise*. The *membership resolution in premise* and *variable membership resolution in premise* rules correspond with *read cache*. Finally, the last two rules correspond with *G^1-tautology resolution in premise/conclusion*.

For the soundness and refutational completeness proof, we have to add further general assumption on the used reduction ordering $\succ$ and $\mathcal{GC}3$, the cased clause calculus for G^n-logics:

General Assumption 10.4.6 *Let the simplification ordering $\succ$ underlying the saturation process be compatible with $\supseteq$, i.e. $A \supseteq B$ implies $A \succ B$ for all ground set terms A, B, and $\Omega_0 \succ A$ holds for all set terms $A \neq \Omega_0$. Let furthermore $\mathcal{GC}3e$ be $\mathcal{GC}3$ plus* variable elimination.

Now we can explain why we use $(t^{:D} \urcorner A)\sigma$ in the condition of *strict basic decoration superposition into premise* instead of $(t^{:D} \urcorner A')\sigma$: the main reason is that we want to find $\mathcal{GC}3e$-steps corresponding with $\mathcal{GCD}$-steps. Since by the above assumption $A'\sigma \succ A$, the second atom is maximal in a clause whenever the first is. This allows us to use clauses defining $\subseteq$ for superposition into $(\sigma(t^{:D})_{nd} : \sigma(A')_{nd} \simeq tt)$ such that we get $(\sigma(t^{:D''})_{nd} : A)_{nd} \simeq tt)$. Then we can overlap with $\sigma(s)_{nd} : \sigma(A) \simeq tt \Leftarrow \Psi(\Gamma'\sigma)$. Remark that both is possible using *strict basic cased superposition into premise* of $\mathcal{GC}3e$. This is helpful in order to prove that $\mathcal{GCD}$ can be reduced to $\mathcal{GC}3e$, after transformation of the clauses using Ψ.

We start with proving the soundness of $\mathcal{GCD}$. First, this is done for the decorated equational constraint solver **DU** in Figure 10.2. We have to do this, since the notion of a unifier does not guarantee the validity of decorations. We assume in the following all used symbols to be in accordance with Definition 10.3.1. Before we start, we define how $\mathcal{GCD}$ is used leading to the formal meaning of redundancy in the context of decorated deduction.

Definition 10.4.7 *Let $\mathcal{P}$ be a set of condeco clauses and $C_1 \; [\![T_1]\!] \; (, C_2 \; [\![T_2]\!]) \in \mathcal{P}$.*

*A condeco clause $C \; [\![T]\!]$ is (Log-)*redundant *in $\mathcal{P}$ if for every ground instance $C\sigma$ of it with σ_{nd} irreducible w.r.t. $R_{\Upsilon(\mathcal{P}_\infty)}$, there exist instances D_i in $irred_{R_{\Upsilon(\mathcal{P}_\infty)}}(\mathcal{P})$, for all $i \in [1..m]$, s.t. $C\sigma \succ_C D_i$ and:*

$$R_{\Upsilon(\mathcal{P}_\infty)} \cup \{D_1, \ldots, D_m\} \models_{G^1} \Upsilon(C\sigma)$$

*If one of the rules in $\mathcal{GCD}$ can be applied to $C_1 \; [\![T_1]\!] \; (, C_2 \; [\![T_2]\!])$, giving $C \; [\![T]\!]$, then $\mathcal{P} \vdash_{GCD} \mathcal{P} \cup \{C \; [\![T]\!]\}$ is a $\mathcal{GCD}$-*step or -inference.

*Let π be a $\mathcal{GCD}$-inference with premises $C_1 \; [\![T_1]\!] (, C_2 \; [\![T_2]\!])$, and with conclusion $C \; [\![T]\!]$. Then, π is (G^1-)*redundant *in a set of condeco clauses $\mathcal{P}'$ if for every ground instance $\pi\sigma$ of π with σ_{nd} irreducible w.r.t. $R_{\Upsilon(\mathcal{P}_\infty)}$, there exist instances D_i in $irred_{R_{\Upsilon(\mathcal{P}_\infty)}}(\Upsilon(\mathcal{P}'))$, $i \in [1..k]$, s.t.:*

- $max(C_1\sigma(, C_2\sigma)) \succ_C D_i$ *and*
- $\Theta((R_{\Upsilon(\mathcal{P}_\infty)} \cup \{D_1, \ldots, D_k\})^{rel}) \models_{G^1} \Upsilon(C\sigma)$.

To summarise, a sequence of $\mathcal{GCD}$-steps *is a sequence* $\mathcal{P}_0, \mathcal{P}_1, \ldots$ *of sets of condeco clauses, s.t.:*

$$\mathcal{P}_i = \mathcal{P}_{i-1} \cup \{C\ [\![T]\!]\}, \quad \text{where } \mathcal{P}_{i-1} \vdash_{GCD} \mathcal{P}_i, \text{ or}$$
$$\mathcal{P}_i = \mathcal{P}_{i-1} \setminus \{C\ [\![T]\!]\}, \quad \text{if } C\ [\![T]\!] \text{ is } G^1\text{-redundant in } \mathcal{P}_{i-1}.$$

A sequence of $\mathcal{GCD}$*-steps* $\mathcal{P}_0, \mathcal{P}_1, \ldots$ *is a* condeco theorem proving derivation, *whenever the transformed sequence* $\Pi(\mathcal{P}_0), \Pi(\mathcal{P}_1), \ldots$ *is a cased* G^1*-theorem proving derivation.*

Let $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ *be a condeco theorem proving derivation, s.t. all* $\mathcal{GCD}$*-inferences with premises in* $\mathcal{P}_\infty$ *are* G^1*-redundant in some* $\mathcal{P}_j$*. Then, the derivation is a* fair condeco theorem proving derivation.

Hence, we work with the redundancy of G^1-logics after transformation with Υ. Consequently, a condeco clause C is G^1-redundant in a set of condeco clauses $\mathcal{P}$ iff $\Psi(C)$ is G^1-redundant in $\Pi(\mathcal{P})$ iff $\Upsilon(C)$ is G^1-redundant in $\Upsilon(C)$. In the following we prefer to use Ψ and Π, i.e. the cased clause level, since it is closest to the condeco level. Intuitively, we just have to discard all decorations and to apply $.^{rel}$ and Θ. in order to test redundancy in the $\mathcal{P}$ transformed in the same way and extended by P_δ. Remark also the following invariant:

Lemma 10.4.8 *An atom of the shape* $x :: A$*, where* $A \neq \Omega_0$*, is in the condition* Γ' *of a condeco clause* $L \Leftarrow \Gamma'\ [\![T]\!]$ *occurring in a sequence of* $\mathcal{GCD}$*-steps* $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_k$*, whenever* $x^{:\{A\}}$ *occurs in it.*

Proof: For all $A \neq \Omega_0$, we know by definition of $\mathcal{P}_0$ and the transformation Δ that a decorated variable has A as decoration, if and only if $(x : A \simeq tt)$ occurs in the premise of the corresponding clause, which is translated into $x :: A$ by Δ. For all other $\mathcal{P}_i$, $i > 0$, we can observe that no new decorated variable is introduced by any inference rule, **DU** included. Furthermore, whenever a condition of the shape $x :: A$ is eliminated (cf. *strict basic variable decoration superposition into premise* and *variable membership resolution in premise*), then the decorated variable is replaced [25] also. Hence, the lemma follows by induction on the number of inference steps. □

Lemma 10.4.9 *The constraint solver* **DU** *for decorated unification is sound and complete for* G^1*-deduction from* P_0*.*

Formally, let $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_k$ *be a condeco theorem proving derivation, counting constraint solving steps as theorem proving steps. Assume that* $\mathcal{P}_{k+1}$ *is obtained from* $\mathcal{P}_k$ *by applying a rule of* **DU** *to the constraint* T *of a clause* $L \Leftarrow \Gamma'\ [\![T]\!]$ *in* $\mathcal{P}_k$ *giving* T'*, s.t.* T' *is* **DU***-satisfiable, and by adding the obtained clause* $L \Leftarrow \Gamma'\ [\![T']\!]$*. Then:*

$$\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k, \mathcal{P}_{k+1}$$

is a G^1*-theorem proving sequence and* $L \Leftarrow \Gamma'\ [\![T]\!]$ *is* G^1*-redundant in* $\mathcal{P}_{k+1}$*.*

Proof: Before we start the proof, let us clarify what we have to prove. Since $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ is a condeco theorem proving sequence, we know that the following holds for all condeco clauses $L \Leftarrow \Gamma'\ [\![T]\!]$ in $\mathcal{P}_k$, assuming $\Psi(L \Leftarrow \Gamma'\ [\![T]\!])$ to be of the form $[\![S]\!]\ L'' \Leftarrow \Gamma''\ [\![T_{nd}]\!]$. For all $(s : t \simeq tt) \in S$ and all ground substitutions σ satisfying T_{nd}, there are instances $L_1, \ldots, L_k \in \Gamma''\sigma$, s.t.:

- $\mathbf{P} \models_{G^1} \Theta((L''\sigma)^{rel})$ whenever $\mathbf{P} \models_{G^1} \Theta(\Gamma''\sigma)$,

[25] Remark that an alternative for this replacement would be to simply forget the decoration and keeping everything else except the conditions $x :: A$.

- $\mathbf{P} \models_{G^1} (s\sigma : t\sigma \simeq tt)$ whenever $\mathbf{P} \models_{G^1} \Theta(L_1^{rel}, \ldots, L_k^{rel})$ and
- $(s\sigma : t\sigma \simeq tt) \succ_C L_1, \ldots, L_k$.

We have to prove two things: First, that these three points also hold for $\Psi(L \Leftarrow \Gamma' \; [\![T']\!])$ and second, the redundancy of the old clause.

Since $\Psi(T) = T_{nd}$ for all constraints T, one can observe that **Delete**, **Decompose**, **Conflict**, **Coalesce**, **Check*** and **SortClash** do not change the solutions of the corresponding undecorated constraints. Consequently, these rules fulfill the redundancy claim, since the instances of $\Psi(L \Leftarrow \Gamma' \; [\![T]\!])$ and $\Psi(L \Leftarrow \Gamma' \; [\![T']\!])$ are the same. They also fulfill the condeco theorem proving derivation requirement, because they do not change any decoration, although they may discard decorations, but discarding the corresponding assertions may also be achieved by *discard cache* from $\mathcal{GC}3e$. The proof for the lemma is then covered by Theorem 4.3.11.

Merge also provides the same solution sets for T_{nd} and T'_{nd}. Furthermore, it does not add nor discard any variable completely. Hence, we get the same instances for $\Psi(L \Leftarrow \Gamma' \; [\![T]\!])$ and $\Psi(L \Leftarrow \Gamma' \; [\![T']\!])$, since they do not depend on assertions, i.e. $\Psi(L \Leftarrow \Gamma' \; [\![T]\!])^{red}$ is G^1-redundant in $\Pi(\mathcal{P}_{k+1})^{red}$. But it is the only rule changing the decoration of a subterm occurring in the constraint: $t'^{:D' \cup D''}$ is possibly a decorated subterm that does not occur in the constraint to be transformed.

If **Merge** is applicable at T, then we know by definition of Ψ that all assertions corresponding with $t'^{:D'}$ and $t''^{:D''}$ are in the assertions of $\Psi(L \Leftarrow \Gamma \; [\![T]\!])$. Furthermore, we know by the introducing remarks that there are instances $L_1, \ldots, L_k \in \Gamma''\sigma$, s.t. the conditions of *write cache* are fulfilled for $\Psi(L \Leftarrow \Gamma \; [\![T]\!])$ and all $A \in D''$, s.t. $(t'_{nd} : A \simeq tt)$ is not already in S, since $\sigma(t'_{nd}) = \sigma(t''_{nd})$ is guaranteed for each solution of T_{nd}. Hence, $\mathbf{P} \models_{G^1} (s\sigma : t\sigma \simeq tt)$ whenever $\mathbf{P} \models_{G^1} \Theta(L_1^{rel}, \ldots, L_k^{rel})$ is guaranteed by Theorem 4.3.11, i.e. the sequence up to $k+1$ is a condeco theorem proving derivation.

Consider now the **VarClash**-rule. Let $L \Leftarrow \Gamma \; [\![T]\!]$ be a condeco clause, s.t. **VarClash** is applicable at T. Adding the obtained clause is trivially sound, since the constraint is unsatisfiable. Furthermore, $\Psi(L \Leftarrow \Gamma \; [\![T]\!])$ is subsumed by one of the clauses in $\{(\Leftarrow (x : A \simeq tt)_{A \in D}) \mid ncs(D)\} \subseteq P_\delta$, i.e. $\Upsilon(L \Leftarrow \Gamma \; [\![T]\!]) = \Theta(\Psi(L \Leftarrow \Gamma \; [\![T]\!])^{rel})$ is G^1-redundant in $\Upsilon(\mathcal{P}_{k+1}) = \Theta((\Psi(\mathcal{P}_{k+1}) \cup P_\delta)^{rel})$ and therefore the sequence up to $k+1$ is again a condeco theorem proving derivation.

There remains the **DecoClash** rule. Adding the obtained clause is again trivially sound. We know for all $B \in D$, the decoration of t, that $\Theta(t_{nd} : B \simeq tt)$ is a sound G^1-consequence of strictly smaller literals in the premise of $\Psi(L \Leftarrow \Gamma \; [\![T]\!])$. Furthermore, since $x^{:\{A\}}$ occurs in the condeco clause, we know that its condition contains $x :: A$, which leads to a precondition of the shape $(x : A \simeq tt)$ in the same clause transformed by Ψ, and T_{nd} contains $x =^? t_{nd}$. Consequently, any instance of $\Psi(\Leftarrow \Gamma \; [\![T]\!])$ by a ground substitution σ (which has to satisfy $\Psi(T) = T_{nd}$) entails $(\sigma(x) : A \simeq tt), (\sigma(x) : B \simeq tt)_{B \in D}$, since $\sigma(x) = \sigma(t_{nd})$, i.e. all instances of $\Psi(L \Leftarrow \Gamma \; [\![T]\!])$ are G^1-redundant in $\Pi(\mathcal{P}_{k+1})^{red}$ and therefore the sequence up to $k+1$ is once again a condeco G^1-theorem proving derivation. □

Now, we come to the soundness of the inference rules for superposition. We prove it by reduction to $\mathcal{GC}3e$ for all ground instances with decorated terms.

Lemma 10.4.10 *$\mathcal{GCD}$ is sound for deduction from* **P** *assuming sort inheritance and any sequence $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots \mathcal{P}_k$ of $\mathcal{GCD}$-steps, satisfies that $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ is a condeco theorem proving derivation.*

Proof: By induction on the number of inference steps in $\mathcal{GCD}$. Since we know that any sequence of $\mathcal{GC}3e$-steps is a cased G^1-theorem proving derivation, we can prove the lemma by reduction of any $\mathcal{GCD}$-step in $\mathcal{P}_i$ to possibly multiple $\mathcal{GC}3e$-steps in $\Pi(\mathcal{P}_i)$. In detail, we prove for all condeco clauses $C_1\ [\![T_1]\!], C_2\ [\![T_2]\!]$, s.t. π is the inference:

$$C_1\ [\![T_1]\!], C_2\ [\![T_2]\!] \vdash_{GCD} C\ [\![T]\!],$$

and $\pi\sigma$ is a ground instance of this inference, that the following diagram commutes, where $\vdash^1$ stands for a one step inference:

$$\begin{array}{ccc} C_1\sigma(, C_2\sigma) & \xrightarrow{\vdash^1_{GCD}} & C\sigma \\ \Big\downarrow \Psi & & \Big\downarrow \Psi \\ \Psi(C_1\sigma)(, \Psi(C_2\sigma)) & \xrightarrow[\vdash^*_{GC3e}]{} & \Psi(C\sigma) \end{array}$$

For steps in the constraint solver **DU**, refer to Lemma 10.4.9. Remark that the base case follows from Corollary 10.3.4. Note also that σ **DU**-satisfies T, T_1 and T_2. We distinguish the last applied rule:

- *strict basic decorated superposition into decorated conclusion*:
  Since $\Psi(\sigma(s^{:D}) \simeq \sigma(s'^{:D'})) = (\sigma(s)_{nd} \simeq \sigma(s')_{nd})$ and $\Psi(\sigma(t^{:D''}) \simeq \sigma(t'^{:D'''})) = (\sigma(t)_{nd} \simeq \sigma(t')_{nd})$, the conditions for the application of *strict basic superposition into conclusion* of $\mathcal{GC}3e$ to $\Psi(\sigma(s^{:D}) \simeq \sigma(s'^{:D'}) \Leftarrow \Gamma'\sigma)$ and $\Psi(\sigma(t^{:D''}) \simeq \sigma(t'^{:D'''}) \Leftarrow \Gamma\sigma)$ are entailed by those of *strict basic decorated superposition into decorated conclusion*. We still have to check that we may reach $\Psi(C\sigma)$, i.e.:

  $$\Psi(\sigma(t^{:\downarrow_u\emptyset}[s'^{:D'}]_u) \simeq \sigma(t'^{:D'''}) \Leftarrow \Gamma\sigma, \Gamma'\sigma),$$

  through $\mathcal{GC}3e$-steps from the result of the application,

  $$[\![S, S']\!]\ \sigma(t[s']_u)_{nd} \simeq \sigma(t')_{nd} \Leftarrow \Psi(\Gamma\sigma), \Psi(\Gamma'\sigma).$$

  $\Psi(C\sigma)$ is by definition of Ψ equal to:

  $$[\![S'''']\!]\ \sigma(t[s']_u)_{nd} \simeq \sigma(t')_{nd} \Leftarrow \Psi(\Gamma\sigma), \Psi(\Gamma'\sigma),$$

  where S'''' is defined accordingly to Ψ. Hence, we have to prove that we can come from (S, S') to S'''' by $\mathcal{GC}3e$. Since we removed all decorations above u, we know that $S'''' \subseteq (S, S')$. Hence, we may apply *discard cache* to forget those assertions in $(S, S') \setminus S''''$ until $S'''' = (S, S')$.

- *strict basic decorated superposition into decoration conclusion*:
  Completely analogous.
- *strict basic decorated superposition into premise*:
  We get with the same arguments as before the applicability of *strict basic superposition into premise* at the premises transformed by Ψ and the reachability of $\Psi(C\sigma)$ through *discard cache*. Consequently, the diagram commutes.
- *strict basic decoration superposition into premise*:
  Once again, we get the applicability of *strict basic superposition into premise* at the premises transformed by Ψ, possibly using rules defining $\subseteq$ first. Remark that the superposition has to be on top, since $\Psi(\sigma(t^{:D} \neg A'))$ has the shape $(\sigma(t)_{nd} : \sigma(A') \simeq tt)$. Using the subset rules, we get $(\sigma(t)_{nd} : \sigma(A) \simeq tt)$, because σ is a **DU**-solution of $A \subseteq A'$, then we may use C_1. Hence, *strict basic superposition into premise* yields a clause equal to $\Psi(C\sigma)$, except that we may have less assertions. We can once more use *discard cache* to forget superfluous assertions until we have $\Psi(C\sigma)$. Consequently, the diagram commutes.
- *strict basic variable decoration superposition into premise*:
  Idem.
- *decorate term*:
  If $(\sigma(t)_{nd} : \sigma(A) \simeq tt)$ is already in the assertions of $\Psi(\sigma(C[t^{:D}]))$, then the soundness of *decorate term* follows from the induction hypothesis on the premise of the rule. Otherwise, $(a), (c)$ imply the corresponding conditions (b) and (c) of *write cache* for the application to the premise transformed by Ψ. There remains to prove the implication of condition (a) of *write cache* by (b) of *decorate term* modulo Ψ. Let:

$$\begin{aligned}&\Psi((s\sigma^{:v} \to s\sigma^{:v \cup \{A\sigma\}}) \Leftarrow \Gamma'\sigma)\\ &= ([\![S]\!]\ (\sigma(s)_{nd} : A\sigma \simeq tt) \Leftarrow \Psi(\Gamma'\sigma))\end{aligned}$$

  From the induction hypothesis we know that $\Pi(\mathcal{P}_0), \Pi(\mathcal{P}_1), \Pi(\mathcal{P}_2), \ldots, \Pi(\mathcal{P}_{k-1})$ is a cased G^1-theorem proving sequence. Hence, using the properties of assertions in Definition 4.3.10 for $\Psi((s^{:v} \to s^{:v \cup \{A\}}) \Leftarrow \Gamma' [\![T']\!]) \in \Pi(\mathcal{P}_{k-1})$, we know that whenever $\mathbf{P} \models_{G^1} L'$, for all instantiations L' of atoms in $\Upsilon(\Gamma'\sigma)$, then $\mathbf{P} \models_{G^1} (\sigma(s)_{nd} \in A\sigma)$. Since σ is a ground instance of the whole inference, we know that $\rho_{nd} =^{\mathcal{V}ar(C_1)} \sigma_{nd}$ for ρ from condition (b) of *decorate term*. There remains to prove that $\mathbf{P} \models_{G^1} \rho_{nd}(L')$ for all instances L' of atoms in $\Psi(\Gamma')$. This is clearly covered by the condition $(\Psi(\Gamma')\rho)_{nd} \subseteq (\Psi(\Gamma'')\sigma)_{nd}$. Consequently, *write cache* is applicable for $(\sigma(t)_{nd} : A\sigma \simeq tt)$ to be added to S in $\Psi(\sigma(C[t^{:D}]))$, since $\rho(s)_{nd} = \sigma(t)_{nd}$. Hence, the diagram commutes.
- *generate well definedness rule*:
  Applying G^1-*equality resolution in conclusion* to the premise transformed by Ψ gives exactly the conclusion transformed by Ψ. Hence, this rule is also sound w.r.t. Ψ and the diagram commutes.
- *decorated equality resolution in premise*:
  Idem for G^1-*equality resolution to in premise*.
- *membership resolution in premise*:
  By the induction hypothesis we know that for all $A' \in D$ and for all σ satisfying T'_{nd}, there are instances in $\Gamma'\sigma$, that are strictly smaller than $(\sigma(t)_{nd} : \sigma(A') \simeq tt)$,

s.t. $\mathbf{P} \models_{G^1} \sigma(t)_{nd} : \sigma(A') \simeq tt$ holds whenever this is true for all these instances. Furthermore, there is an $A' \in D$, s.t. $\sigma(A') \subseteq \sigma(A)$ and therefore also $\mathbf{P} \models_{G^1} \sigma(t)_{nd} : \sigma(A) \simeq tt$. Furthermore, $(\sigma(t)_{nd} : \sigma(A) \simeq tt) \succ (\sigma(t)_{nd} : \sigma(A') \simeq tt)$, since $\sigma(A) \succ \sigma(A')$ by our assumptions. Hence, we may use the same instances to apply *write cache* to add $(\sigma(t)_{nd} : \sigma(A) \simeq tt)$ to the assertions of the clause transformed by Ψ, followed by *read cache* to eliminate the same equation from the condition and *discard cache* to forget $(\sigma(t)_{nd} : \sigma(A) \simeq tt)$ again. The result is $\Psi(C\sigma)$.

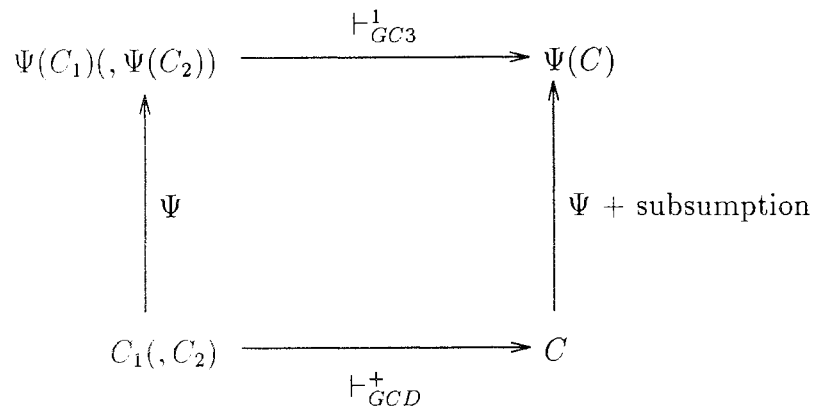
- *membership resolution in premise*:
  Completely analogous to the last case for the elimination of $x :: A$, since this translates by Ψ into $x : A \simeq tt$. The application of the decorated substitution is then covered by *variable elimination* and eventually *discard cache* for the assertions corresponding with decorations of $t^{:D}$. Again, the result is $\Psi(C\sigma)$.
- *decorated tautology resolution in conclusion*:
  Apply G^1*-tautology resolution in conclusion.*
- *decorated tautology resolution in premise*:
  Apply G^1*-tautology resolution in premise.*

□

We go on with the implication of cased saturation by condeco saturation.

Lemma 10.4.11 *Let $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ be a fair condeco theorem proving sequence.*
If every $\mathcal{GCD}$-inference is redundant in $\mathcal{P}_\infty$, then every $\mathcal{GC}3$-inference from $\Pi(\mathcal{P}_\infty)$ is redundant in $\Pi(\mathcal{P}_\infty)$, except for overlaps of subset rules into premises of non-subset rules and overlaps into clauses representing sort inheritance.

Proof: We actually prove the following more formal statement: For all $C_1', C_2' \in \Pi(\mathcal{P}_\infty)$, if $C_1', C_2' \vdash_{GC3e} C'$, then C' is G^1-redundant in $\Pi(\mathcal{P}_\infty)$, except for overlaps of subset rules into premises of non-subset rules and overlaps into clauses representing sort inheritance. If both C_1' and C_2', are in $\Psi(\mathcal{P}_\infty)$, i.e. there are C_1, C_2, s.t. $C_1' = \Psi(C_1)$ and $C_2' = \Psi(C_2)$, then there exists a C, s.t. the following diagram commutes, where $\vdash^+$ stands for one or more inference steps:



Assume that there is a $\mathcal{GC}3$-inference in $\Pi(P_\infty)$ that is not redundant in $\Pi(P_\infty)$. We distinguish the different cases for the last applied rules in order to get a contradiction with

the fact that all $\mathcal{GCD}$-inferences in $\mathcal{P}_\infty$ are redundant in $\mathcal{P}_\infty$. We start with a discussion concerning overlaps into rules of P_δ.

Overlapping into $(x^1 : \Omega_1 \simeq tt)$ results in $tt \simeq tt$, which is a redundant inference, since $tt \simeq tt$ is by definition in $\Pi(P_\infty)$.

Overlaps into subset rules are only possible in the head, since $\succ$ is compatible with $\subseteq$ and $\subseteq$ is cycle-free. Therefore, the premise of the clause is always strictly smaller than the head. The result of such an overlap is a trivial clause with head $tt \simeq tt$, i.e. it is a redundant inference as before. The same is true for overlaps of subset rules into the head of another clause.

There remains the case of superpositions with a subset rule into the premise of a clause. Clearly, after these superpositions, which replace literals of the form $t : A \simeq tt$ by $tt \simeq tt, t : B \simeq tt$ if $B \subseteq A$, a superposition with another (non-subset) clause $\Psi(C)$ at $t : B \simeq tt$ might be possible. This is the reason why we have to prove also the redundancy of such *coerced* superpositions below in the case of superpositions on top of a literal $t : A \simeq tt$ in a premise.

There remain the inferences from $C_1'(, C_2')$, s.t. there are C_1, C_2 with $\Psi(C_1) = C_1'$ (and $\Psi(C_2) = C_2'$. We distinguish the different rules in $\mathcal{GC}3$:

- *strict basic superposition into conclusion*:
  If C_1, s.t. $\Psi(C_1) = (s \simeq s' \Leftarrow \Gamma' \; [\![T']\!])$, is of the form $(s''^{:D''} \simeq s'''^{:D'''}) \Leftarrow \Gamma''' \; [\![T''']\!]$, then the conditions of *strict basic superposition into conclusion* imply those of *strict basic decorated superposition into decorated conclusion* or *strict basic decorated superposition into decoration conclusion*, depending on the nature of $t \simeq t'$. If the resulting decorated equality constraint is unsatisfiable, then the result of *strict basic superposition into conclusion* is subsumed by one of the clauses for detecting non-SI and therefore redundant in $\Pi(\mathcal{P}_\infty)$, in contradiction to our assumption.
  Otherwise, we know that the conclusion, say C, of superposing C_1 into C_2 is redundant in $\mathcal{P}_\infty$, i.e. all ground instances of $\Psi(C)$ are redundant in $\Pi(\mathcal{P}_\infty)$. We have to prove that all ground instances of the $\mathcal{GC}3$-inference on $\Psi(C_1), \Psi(C_2)$ are also redundant in $\Pi(\mathcal{P}_\infty)$. But these are by definition of Ψ exactly those of C, since the only difference (cf. Lemma 10.4.10) is the assertion part, which is ignored in the definition of ground instances.
  If C_1 is of the form $(s'''^{:v} \simeq s'''^{:v \cup \{A\}}) \Leftarrow \Gamma''' \; [\![T''']\!]$, then $\Psi(C_1)$ is of the form $(s'''_{nd} : A \simeq tt) \Leftarrow \Gamma' \; [\![T']\!]$) and $\Psi(C_2)$ has to be of the form $(t''' : A \simeq tt) \Leftarrow \Gamma \; [\![T]\!]$, i.e. u is the top position. Since tt is the smallest function symbol in the precedence, we get the result $tt \simeq tt \Leftarrow \Gamma, \Gamma' \; [\![T, T', T'']\!]$. This inference is redundant, because $tt \simeq tt$ is in $\Pi(\mathcal{P}_\infty)$.
- *strict basic superposition into premise*:
  (*i*) If C_1, s.t. $\Psi(C_1) = (s \simeq s' \Leftarrow \Gamma' \; [\![T']\!])$, is of the form $(s''^{:D''} \simeq s'''^{:D'''}) \Leftarrow \Gamma''' \; [\![T''']\!]$, then the conditions of *strict basic superposition into premise* imply those of *strict basic decorated superposition into premise*. If the resulting constraint is unsatisfiable, we have again subsumption of the result by a clause representing sort inheritance. Since the result of applying the latter rule to C_1, C_2, say C, transformed by Ψ is the same as the result of *strict basic superposition into premise* applied to $\Psi(C_1), \Psi(C_2)$, we know that the latter inference is redundant in $\Pi(\mathcal{P}_\infty)$.
  (*ii*) Otherwise, if C_1 is of the form $(s'''^{:v} \to s'''^{:v \cup \{A\}}) \Leftarrow \Gamma''' \; [\![T''']\!]$, then $\Psi(C_1)$ is of the form $[\![S']\!] \; (s'''_{nd} : A \simeq tt) \Leftarrow \Gamma' \; [\![T']\!]$ and only top superpositions are possible. Here

we have also to respect coerced superpositions, i.e. those that might be possible after superpositions with subset rules into C_2, as mentioned in the beginning of the proof. Now, if C_2 is of the form $L'''' \Leftarrow \Gamma'''', L' \; [\![T'''']\!]$, with L' of the shape $t''''^{:D} \lhd A'$, i.e. $\Psi(L') = (t''''_{nd} : A' \simeq tt)$ with $A'\sigma \subseteq A\sigma$ for some ground substitution σ satisfying T'''', then *strict basic decoration superposition into premise* is applicable. This results in C of the form $L'''' \Leftarrow \Gamma'''' \; [\![T'''', t''''^{:D} =^? s'''^{:D}]\!]$, which subsumes after transformation with Ψ the result of *strict basic superposition into premise* applied to C'_1, C'_2, say C', which has the shape $L''''_{nd} \Leftarrow \Gamma''''_{nd}, tt \simeq tt \; [\![T''''_{nd}, t''''_{nd} =^? s'''_{nd}]\!]$. Since C is redundant in $\mathcal{P}_\infty$, we know that C' is, too.

If L' is of the shape $x :: A'$, then *strict basic variable decoration superposition into premise* is applicable and the resulting clause C has the shape:

$$(L'''' \Leftarrow \Gamma'''' \; [\![T'''']\!])\{x^{:\{A'\}} \mapsto s^{:D}\}$$

Following the same argumentation as before, we can prove that this inference is redundant in $\mathcal{P}_\infty$ and that the instances of $\Psi(C)$ subsume those of C'.

- *G^1-equality resolution in conclusion*:
  Remark that $t \simeq t'$ cannot be of the form $t'' : A \simeq tt$. Consequently, C_1 has to be of the form $t''^{:D''} \simeq t'''^{:D'''} \Leftarrow \Gamma''' \; [\![T''']\!]$ and we can apply *generate well definedness rule* in order to get $t''^{:v} \to t''^{:v \cup \{\Omega_0\}} \Leftarrow \Gamma''' \; [\![T''', t''^{:D''} =^? t'''^{:D'''}]\!]$, which is, after transformation by Ψ, syntactically the same as $t : \Omega_0 \simeq tt \Leftarrow \Gamma' \; [\![T', T'']\!]$, the conclusion of *$G^1$-equality resolution in conclusion*. Hence, the latter inference is redundant in $\Pi(\mathcal{P}_\infty)$, since the former is in $\mathcal{P}_\infty$.
- *G^1-equality resolution in premise*:
  By the condition of *G^1-equality resolution in premise*, $t \simeq t'$ cannot be of the form $t'' : A \simeq tt$. Consequently, C_1 has to be of the form $L''' \Leftarrow t''^{:D''} \simeq t'''^{:D'''}, \Gamma''' \; [\![T''']\!]$ and we can apply *decorated equality resolution in premise* yielding $L''' \Leftarrow t''^{:D''} \lhd \Omega_0, \Gamma''' \; [\![T''']\!]$, which gives after transformation by Ψ a clause C, which is syntactically the same as $[\![S]\!] \; L \Leftarrow t : \Omega_0 \simeq tt, \Gamma' \; [\![T', t = t']\!]$, which is also the result of *G^1-equality resolution in premise* applied to $\Psi(C_1)$. Hence, the latter inference is redundant in $\Pi(\mathcal{P}_\infty)$, since the former is in $\mathcal{P}_\infty$.
- *G^1-tautology resolution in conclusion*, *G^1-tautology resolution in premise*:
  Apply *decorated tautology resolution in conclusion* or *decorated tautology resolution in premise*, respectively.

□

Hence, we can conclude the following from Theorem 4.3.13. Remark that $\mathcal{P}_0$ does not include sort inheritance. The only-if direction is missing, since we may have an empty clause due to sort-inheritance tests. Testing sort inheritance is subject of Section 10.6.

Corollary 10.4.12 *If every $\mathcal{GCD}$-inference is G^1-redundant in $\mathcal{P}_\infty$, then $\Pi(\mathcal{P}_\infty)$ is saturated up to superpositions into premises of non-subset clauses using subset clauses and superpositions into clauses representing sort-inheritance.*

5 Decorated Rewriting

The goal in this section is now to extend the results on ground confluence and completeness of saturated clause sets which are quasi-reductive SDTRSs (cf. Theorem 4.1.17) to the decorated

calculus. The restriction on ground terms will be the same for individual terms, but not for set terms. This corresponds with polymorphic types like $\forall z^1, list(z^1)$. As example, we may say that $nil \in list(z^1)$, where z^1 remains uninstantiated. Hence, we may have $nil^{:list(z)}$ as part of a decorated term to be reduced. The interpretation of decorations remains very intuitive: *nil* belongs to all instantiations of $list(z)$, analogous to the polymorphic type $\forall z^1, list(z^1)$. Remark that the corresponding cased clause $(nil : list(z^1) \simeq tt)$ is operational, i.e. the right-hand side does not contain any new variable. Hence, condeco clauses invert the direction of deduction for membership formulas: instead of resolving typing proofs lazily or by need, we accumulate them in the decoration. We prove in this section that this strategy has only a minor influence on completeness for deduction.

Hence, we say that a set of condeco clauses $\mathcal{P}$ is an SDTRS, if $\Psi(\mathcal{P})$ fulfils the conditions of an SDTRS. The conditions for the quasi-reductivity property are also translated through Ψ. Therefore, set terms A in membership conditions of the form $x :: A$ or $t^{:D} \uparrow A$ may contain *new* variables. This is harmless, since we do not have any equations on set terms and furthermore, we assume the existence of a subset constraint solver. Consequently, non-ground set terms lead us to consider constraint rewriting. Remark that we assume condeco clauses $C\ [\![T]\!]$ to be transformed into conditional rewrite rules by eliminating equality constraints, i.e. we get $C\sigma\ [\![T'\sigma]\!]$, where σ is a most general solution for the equational part of T and T' is the remaining part containing all subset constraints. We call the result then rule instead of clause.

Let $\cong_d$ be structural equality on decorated terms, lexicographically combined with equality modulo SI-closure, i.e. in this Chapter simply including all supersorts of the decoration in each term position (cf. Chapter 5). Alternatively, one may define it as syntactic equality of the SI-closure of the two terms to be compared. We start with the definition of the rewrite relation induced by decorated SDTRSs. The intuition behind this definition is to accumulate subset constraints during rule application and to evaluate conditions as with unconditional SDTRSs, except for typing conditions, where we add new constraints like with *(variable) membership resolution in premise*. Remark that the empty disjunction $\bigvee_\emptyset$ is as usual equivalent to falsehood.

Definition 10.5.1 *Let the set $\mathcal{P}$ of condeco clauses be a quasi-reductive SDTRS, $t^{:D}$ be a decorated ground term and $C = (L \Leftarrow (L_1, \ldots, L_m)\ [\![T'']\!])$ be a rule from $\mathcal{P}$, s.t. the atoms $\Psi(L_1), \ldots, \Psi(L_m)$ are in decreasing order as in Definition 3.2.4. We distinguish the different cases for L:*

- *Let $L = (u^{:D_u} \to u'^{:D_{u'}})$. Then, $t^{:D}\ [\![T]\!]$, where T contains subset constraints only,* rewrites *to $t'^{:D'}\ [\![T']\!]$ by C at position $p \in \mathcal{O}cc(t^{:D})$, written $t^{:D}\ [\![T]\!] \rightarrowtail_{\mathcal{P}}^{p,\sigma,C} t'^{:D'}\ [\![T']\!]$, if σ is a decorated matcher of $t^{:D}{}_{|p} \geq^? u^{:D_u}$, s.t.:*
  - *$t'^{:D'} = t^{:D}[\sigma_{m+1}(u'^{:D_{u'}})]_p$ and $T' = T_{m+1}$, where*
  - *$\sigma_1 = \sigma$, $T_1 = T \cup T'' \cup T_\sigma$,*
  - *for $i \in [1..m]$:*
    - *If L_i is of the shape $u_i{}^{:D_i} \simeq u_i'^{:D_i'}$ and $\Psi(L_i)$ (as rule condition) is of the shape $(u_i)_{nd} \xrightarrow{*} (u_i')_{nd}$, then:*
      - *there is a $u_i''^{:D_i''}$, s.t. $\sigma_i(u_i{}^{:D_i}) \rightarrowtail_{\mathcal{P}}^* u_i''^{:D_i''}$, and either $D_i'' \neq \emptyset$ or u_i'' is tt,*
      - *σ_{i+1} is $\tau \circ \sigma_i$, where τ is a solution of the matching equation $u_i''^{:D_i''} \geq^? \sigma_i(u_i'^{:D_i'})$ and*
      - *$T_{i+1} = T_i \cup T_\tau$, where all unconstrained set variables in T_τ stemming from $u_i''^{:D_i''}$ are renamed.*

Delete	$\mathcal{M} \wedge t^{:D} \leq^? t'^{:D'}$	$\Longrightarrow \mathcal{M}$ if $t^{:D} =_{nd} t'^{:D'}$
Decompose	$\mathcal{M} \wedge f(t_1^{:D_1}, \ldots, t_n^{:D_n})^{:D} \leq^? f(t'^{:D'_1}_1, \ldots, t'^{:D'_n}_n)^{:D'}$	$\Longrightarrow \mathcal{M} \wedge \bigwedge_{i=1,\ldots,n}(t_i^{:D_i} \leq^? t'^{:D'_i}_i)$
Conflict	$\mathcal{M} \wedge f(t_1^{:D_1}, \ldots, t_n^{:D_n})^{:D} \leq^? g(t'^{:D'_1}_1, \ldots, t'^{:D'_m}_m)^{:D'}$	$\Longrightarrow \mathbb{F}$ if $f \neq g$ or $m \neq n$
Merge	$\mathcal{M} \wedge x^{:\downarrow\emptyset} \leq^? t^{:D} \wedge x^{:\downarrow\emptyset} \leq^? t'^{:D'}$	$\Longrightarrow \mathcal{M} \wedge x^{:\downarrow\emptyset} \leq^? t^{:D \cup D'}$ if $x \in \mathcal{X}_\Diamond$ and $t^{:D} =_{nd} t'^{:D'}$
MergeClash	$\mathcal{M} \wedge x^{:\downarrow\emptyset} \leq^? t^{:D} \wedge x^{:\downarrow\emptyset} \leq^? t'^{:D'}$	$\Longrightarrow \mathbb{F}$ if $x \in \mathcal{X}_\Diamond$ and $t^{:D} \neq_{nd} t'^{:D'}$
VariableClash	$\mathcal{M} \wedge f(t_1^{:D_1}, \ldots, t_n^{:D_n})^{:D} \leq^? x^{:\downarrow\emptyset}$	$\Longrightarrow \mathbb{F}$ if $x \in \mathcal{X}_\Diamond$

MATCHsub

Figure 10.3: Rules for Decorated Matching with Subset Constraints

- *If L_i is of the shape $u_i^{:D_i} \simeq u'^{:D'_i}_i$ and $\Psi(L_i)$ is of the shape $(u_i)_{nd} \simeq (u'_i)_{nd}$, then:*
  - *there is a $u''^{:D''_i}_i$, s.t. $\sigma_i(u_i^{:D_i}) \rightarrowtail_{\mathcal{P}}^* u''^{:D''_i}_i =_{nd} \circ \leftarrowtail_{\mathcal{P}}^* \sigma_i(u'^{:D'_i}_i)$, and either $D''_i \neq \emptyset$ or u''_i is tt,*
  - *σ_{i+1} is σ_i and*
  - *$T_{i+1} = T_i$.*
- *If L_i is of the shape $u_i^{:D_i} \neg A$ or $u_i^{:D_i} :: A$, then:*
  - *there is a $u'^{:D_i}_i$, s.t. $\sigma_i(u_i^{:D_i}) \rightarrowtail_{\mathcal{P}}^* u'^{:D'_i}_i$,*
  - *$T_{i+1} = [\![T_i, \bigvee_{A' \in D'_i}(A' \subseteq^? A)]\!]$ is satisfiable and*
  - *σ_{i+1} is σ_i.*

- *Let L be $(u^{:v} \rightarrow u^{:v \cup \{A\}})$, s.t. u is not a variable. Then $t^{:D}\ [\![T]\!]$ rewrites to $t'^{:D'}\ [\![T']\!]$ by C at position $p \in Occ(t^{:D})$, written $t^{:D}\ [\![T]\!] \rightarrowtail_{\mathcal{P}}^{p,\sigma,C} t'^{:D'}\ [\![T']\!]$, if σ is a decorated matcher of $t^{:D}{}_{|_p} \geq^? u^{:D_p}$, where $D_p = Deco(t^{:D}{}_{|_p})$, s.t. the conditions L_i for $i \in [1..m]$, are satisfied as in the previous case, defining T', σ_{m+1} and hence $t'^{:D'}$ through $t^{:D}[\sigma(u^{:D_p \cup \{\sigma(A)\}})]_p$. Furthermore, the solutions for x in $[\![T, \bigvee_{A' \in D} A' \subseteq^? x]\!]$ have to be strictly subsumed by those of $[\![T, T_\sigma, \sigma(A) \subseteq^? x]\!]$, where all unconstrained set variables in T_σ stemming from $t^{:D}{}_{|_p}$ have to be renamed due to universal quantification.*

Remark that the conditions for the decorated matcher σ in the definition imply that all conditions of the shape $x :: set(x)$ in the condition of a decorated rewrite rule are automatically satisfied, if x occurs in the left hand side of the head.

A corresponding matching algorithm is shown in Figure 10.3. Let $u'^{:D'_i}_i \leq^? u''^{:D''_i}_i$ be the initial matching equation and $[\![T_i]\!]$ be the corresponding constraint in the last definition. If $\mathbb{F}$

is derived, then there is no matcher. Otherwise, let $\bigwedge_{j\in[1..m]} x_j{:}^{D_j} \leq^? t_j'''{:}^{D_j'''}$ be the obtained normal form. Now, let $\sigma = \{x_j{:}^{D_j} \mapsto t_j'''{:}^{D_j'''} \mid j \in [1..m]\}$ and σ_S be the part concerning set variables (i.e. those with $D_j = \emptyset$) and σ_D be the part concerning individual (decorated) variables, i.e. with $D_j = \{A_j\}$ for some $A_j \in \mathcal{S}(\mathbf{X})$, and let $I \subseteq [1..m]$ stand for the corresponding indices.

If $[\![T_\sigma]\!] = [\![T_i, \bigwedge_{j\in I} \bigvee_{A_j'\in D_j'''}(A_j' \subseteq^? \sigma_S(A_j))]\!]$, where all unconstrained variables in D_j''' are renamed, is satisfiable, then σ is a decorated matcher of $u_i'{:}^{D_i'} \leq^? u_i''{:}^{D_i''}$ $[\![T_i]\!]$.

Remark that the main difference to undecorated rewriting apart from constraints is now that rules are no more applicable if the sort of a variable in the left hand side is not included in the top decoration of its image. Moreover, decorated matching does not give a unique solution, as the disjunction in the constraint shows. Before we start with the properties of decorated rewriting, let us give an example.

Example 10.5.2 *Assume Int, the set of integers, is given as superset of Nat using additionally p as predecessor operation. Let furthermore polymorphic lists be defined as in the last example. Hence, $p(0)$ denotes -1 and so on. Now, let $Nat \subseteq Int$ and:*

$$\begin{aligned}\mathcal{P}_\infty = \ & \{(0^{:v} \to 0^{:v\cup\{Nat\}}),\\ & (p(0)^{:v} \to p(0)^{:v\cup\{Int\}}),\\ & (nil^{:v} \to nil^{:v\cup\{List(x)\}}),\\ & ((cons(x^{:\{z\}}, y^{:\{List(z)\}})^{:v} \to cons(x^{:\{z\}}, y^{:\{List(z)\}})^{:v\cup\{List(z)\}} \Leftarrow x :: z, y :: List(z))\end{aligned}$$

Consider the following sample reduction for $cons(p(0), cons(0, nil))$:

$$\begin{aligned}& cons(p(0), cons(0, nil))^{:\downarrow\emptyset}\\ \rightarrowtail_{\mathcal{P}_\infty}\ & cons(p(0^{:\emptyset})^{:\emptyset}, cons(0^{:\emptyset}, nil^{:\{List(x)\}})^{:\emptyset})^{:\emptyset}\\ \rightarrowtail_{\mathcal{P}_\infty}\ & cons(p(0^{:\emptyset})^{:\emptyset}, cons(0^{:\{Nat\}}, nil^{:\{List(x)\}})^{:\emptyset})^{:\emptyset}\\ \rightarrowtail_{\mathcal{P}_\infty}\ & cons(p(0^{:\{Nat\}})^{:\emptyset}, cons(0^{:\{Nat\}}, nil^{:\{List(x)\}})^{:\emptyset})^{:\emptyset}\\ \rightarrowtail_{\mathcal{P}_\infty}\ & cons(p(0^{:\{Nat\}})^{:\{Int\}}, cons(0^{:\{Nat\}}, nil^{:\{List(x)\}})^{:\emptyset})^{:\emptyset}\\ \rightarrowtail_{\mathcal{P}_\infty}\ & cons(p(0^{:\{Nat\}})^{:\{Int\}}, cons(0^{:\{Nat\}}, nil^{:\{List(x)\}})^{:\{List(z)\}})^{:\emptyset}\\ & \qquad [\![Nat \subseteq^? z, List(x') \subseteq^? List(z)]\!]\\ \rightarrowtail_{\mathcal{P}_\infty}\ & cons(p(0^{:\{Nat\}})^{:\{Int\}}, cons(0^{:\{Nat\}}, nil^{:\{List(x)\}})^{:\{List(z)\}})^{:\{List(z')\}}\\ & \qquad [\![Nat \subseteq^? z, List(x') \subseteq^? List(z), Int \subseteq^? z', List(z) \subseteq^? List(z')]\!]\end{aligned}$$

If the set operator List is declared to be monotone, which is expressible by $List(x) \subseteq List(y) \Leftarrow x \subseteq y$, then the final constraint can be simplified into $[\![Nat \subseteq^? z, x' \subseteq^? z, Int \subseteq^? z', z \subseteq^? z']\!]$. This constraint is satisfiable, since there is the trivial solution mapping all variables to Ω_0. A closer look on how we use set terms in decorations for decorated rewriting reveals that we only use them as lower bound for further computations or constraints. Hence, if all set operators are monotone, we may eliminate variables like z in the last constraint, leading to $[\![x' \subseteq^? Nat, Int \subseteq^? z', Nat \subseteq^? z']\!]$ and $List(Nat)$ as decoration instead of $List(z)$. Using similar arguments for z' we get $[\![x' \subseteq^? Nat]\!]$ and $List(Int)$ instead of $List(z')$. The remaining constraint can now be discarded, since it is satisfiable and x' does not occur anywhere else. Similarly, we hope that other properties like regularity may help us simplifying the calculus.

One difficulty among others is the handling of set terms as arguments and decorations at the same time. To illustrate this, assume nil takes an argument indicating the type of elements for a list. The decoration rule for nil is then:

$$nil(x)^{:v} \to nil(x)^{:v\cup\{x\}}$$

Decorating $cons(p(0), cons(0, nil(Nat)))^{:\downarrow\emptyset}$ *results hence in:*

$$cons(p(0^{:\{Nat\}})^{:\{Int\}}, cons(0^{:\{Nat\}}, nil(Nat)^{:\{List(Nat)\}})^{:\{List(z)\}})^{:\{List(z')\}}$$
$$[\![Nat \subseteq^? z, List(Nat) \subseteq List(z), List(z) \subseteq^? List(z'), Int \subseteq^? z']\!]$$

This can with monotonicity be simplified into $[\![Nat \subseteq^? z, z \subseteq^? z', Int \subseteq^? z']\!]$. *Hence, disambiguation of decorations can be achieved through additional arguments, similar to polymorphic* λ*-calculi, where type abstraction has a binder on its own. Remark that the use of constraints may be an advantage when one does not want to do rewriting but logic programming (cf. [Hanus, 1991]). The same arguments hold for decorating during saturation.*

Now, we come to the proof of soundness for G^1-deduction with rewrite relation of decorated SDTRSs. We therefore show that proofs by decorated rewriting, interpreted as for the evaluation of conditions in the definition of the rewrite relation, correspond with condeco theorem proving derivations that can be constructed using $\mathcal{GCD}$. Remark that $\Leftarrow [\![T]\!]$ is the empty clause, whenever T is a satisfiable constraint.

Lemma 10.5.3 *Let* $\mathcal{P}$ *be a quasi-reductive SDTRS and* $t^{:D}, t'^{:D'}$ *be valid decorated terms being ground, except for set variables in decorations,* T, T' *be satisfiable constraints and* A *be in* $\mathcal{S}(\mathbf{X})$. *Then:*

- *If* $t^{:D}\ [\![T]\!] \rightarrowtail^!_{\mathcal{P}} t''^{:D''}\ [\![T'']\!]$ *and* $t'^{:D'}\ [\![T']\!] \rightarrowtail^!_{\mathcal{P}} t'''^{:D'''}\ [\![T''']\!]$, *s.t.* $[\![T'', T''']\!]$ *is satisfiable,* $t''^{:D''} =_{nd} t'''^{:D'''}$ *and either* $D'' \neq \emptyset$ *or* $t'' =_{nd} t''' =_{nd} tt$, *then* $\mathcal{P} \cup \{\Leftarrow t^{:D} \simeq t'^{:D'}\ [\![T, T']\!]\} \vdash_{GCD} (\Leftarrow [\![T'''']\!])$, *s.t.* T'''' *is satisfiable.*
- *If* $t^{:D}\ [\![T]\!] \rightarrowtail^!_{\mathcal{P}} t''^{:D''}\ [\![T'']\!]$, $(D'' \neq \emptyset$ *or* $t'' = tt)$ *and* $t'''^{:D'''}$ *is strongly irreducible, s.t.* $T'', t''^{:D''} =^? t'''^{:D'''}$ *is satisfiable, then* $\mathcal{P} \cup \{\Leftarrow t^{:D} \simeq t'''^{:D'''}\ [\![T]\!]\} \vdash_{GCD} (\Leftarrow [\![T''']\!])$, *s.t.* T''' *is satisfiable and for all ground substitutions* σ *satisfying* T''', $t''^{:D''} =_{nd} \sigma(t'''^{:D'''})$.
- *If* $t^{:D}\ [\![T]\!] \rightarrowtail^!_{\mathcal{P}} t''^{:D''}\ [\![T'']\!]$, *s.t.* $[\![T'', \bigvee_{A' \in D''}(A' \subseteq^? A)]\!]$ *is satisfiable, then* $\mathcal{P} \cup \{\Leftarrow t^{:D} \neg A\ [\![T]\!]\} \vdash_{GCD} (\Leftarrow [\![T''']\!])$, *s.t.* T''' *is satisfiable.*
- *If* $t^{:D}\ [\![T]\!] \rightarrowtail^!_{\mathcal{P}} t''^{:D''}\ [\![T'']\!]$, *s.t.* $[\![T'', \bigvee_{A' \in D''}(A' \subseteq^? A)]\!]$ *is satisfiable, then* $\mathcal{P} \cup \{\Leftarrow x :: A, x^{:\{A\}} \simeq t^{:D}\ [\![T]\!]\} \vdash_{GCD} (\Leftarrow [\![T''']\!])$, *s.t.* T''' *is satisfiable.*

Hence, the rewrite relation for decorated SDTRSs is sound for G^1*-deduction.*

Proof: The proof is by induction on $\succ_C$, where $\succ$ is the ordering used to prove quasi-reductivity of $\mathcal{P}$, over the multiset expression of goal clauses, i.e. clauses of the form $(\Leftarrow \Gamma)$. Remark that we discard decorations for computing these multi-set expressions.

If t_{nd} (and t'_{nd}) are minimal, i.e. irreducible, then t_{nd} is identical with t''_{nd} (and t'_{nd} with t'''_{nd} in the first case). Then, in the first case, one of the rules *decorated tautology resolution in premise* or *decorated equality resolution in premise* is applicable. If the former is applied, then the lemma is trivial, since T is identical with T''''. If the latter is applied, then we get a strictly smaller goal clause, say $C_1\ [\![T_1]\!]$. Using *variable membership resolution in premise*, we may eliminate all individual variables in C_1. Hence, we can apply the induction hypothesis to get the result.

In the second case, we may proceed similarly. The constraint T''' gets $T, t^{:D} =^? t'''^{:D'''}$, which gives us together with $t_{nd} = t''_{nd}$ that $t''^{:D''} =_{nd} \sigma(t'''^{:D'''})$ for all σ satisfying T'''.

Remark that $t_{nd} = t''_{nd}$ follows from the minimality of t_{nd}. Hence, this gives us the base case.

In the third case, *membership resolution in premise* can be applied, leading directly to the desired result. In the fourth case, we can use *equality resolution in premise* to obtain $\Leftarrow x :: A, t^{:D} \urcorner \Omega_0 \; [\![T, t^{:D} =^? x^{:A}]\!]$. Now, *membership resolution in premise* can be applied to eliminate $t^{:D} \urcorner \Omega_0$ and *variable membership resolution in premise* to eliminate $x :: A$. Hence, we get the empty clause with a satisfiable constraint.

Otherwise, assume that the lemma holds for all strictly smaller clauses. We distinguish the four cases, as given in the lemma:

- If $t_{nd} = t'_{nd}$, then we can apply *decorated tautology resolution in premise* or *decorated equality resolution in premise* and as above to get a strictly smaller clause (cf. Lemma 4.3.7, rule *G^n-equality resolution in premise*). Otherwise, let w.l.o.g. $t^{:D}$ be the maximal term of the two w.r.t. $\succ$. Let furthermore $(s^{:D_s} \to s'^{:D_{s'}}) \Leftarrow L_1, \ldots, L_m$ be the rule applied first in $t^{:D} \rightarrowtail_{\mathcal{P}} t_1{}^{:D_1} \rightarrowtail^*_{\mathcal{P}} t'''^{:D''}$. Hence, we may apply *strict basic decorated superposition into decorated conclusion*, which gives us $(\Leftarrow t^{:D}[s'^{:D_{s'}}]_u \simeq t'^{:D'}, L_1, \ldots, L_m \; [\![T, t^{:D}|_u =^? s'^{:D_{s'}}]\!])$. Since the constraint represents a matching problem ($t^{:D}|_u$ is ground), we can solve it completely giving a matcher σ, that we can apply to the clause using *variable membership resolution in premise*. Now, we may apply the induction hypothesis to $\Leftarrow \sigma(L_1)$, giving us σ_2 through the definition of the rewrite relation for SDTRSs, with which we can instantiate L_2. Going on like this, we eventually get a clause of the shape $(\Leftarrow t^{:D}[s'^{:D_{s'}}]_u \simeq t'^{:D'} \; [\![T, T_{\sigma_{m+1}}]\!])$, where $T_{\sigma_{m+1}}$ defines σ_{m+1}. We can finally use *variable membership resolution in premise*, to eliminate all variables, introducing new subset constraints, which are satisfiable together with T by our definition of the rewrite relation. After this, we may still have some additional constraints defining the image of the variables occurring in the left but not in the right hand side of the applied rule. These may be discarded, since these variables do not occur in the corresponding clause. Hence, we get $(\Leftarrow t_1{}^{:D_1} \simeq t'^{:D'} \; [\![T_1]\!])$, which is once again ground and T_1 is satisfiable. Hence, we can apply the induction hypothesis on this clause.
- The case $\{\Leftarrow t^{:D} \simeq t'''^{:D'''}\}$ is completely analogous.
- In the case of $\{\Leftarrow t^{:D} \urcorner A\}$, either $[\![T'', \bigvee_{A' \in D''}(A' \subseteq^? A)]\!]$ is satisfiable, i.e. we can apply *membership resolution in premise* as in the base case. Otherwise, we can use *strict basic decoration superposition into premise* and continue as in the first case. Remark that the atom $t^{:D} \urcorner A$ is eliminated from this first inference step onwards, i.e. following the lines of the first case, we do not need to apply the induction hypothesis in the end, since we already have the empty clause.
- In the case $\{\Leftarrow x :: A, x^{:\{A\}} \simeq t^{:D}\}$, if $[\![T'', \bigvee_{A' \in D''}(A' \subseteq^? A)]\!]$ is satisfiable, then we may apply *decorated equality resolution in premise*, followed by *variable membership resolution in premise* and *membership resolution in premise* (for $t^{:D} \urcorner \Omega_0$), in order to get the result. Otherwise, we may follow the first case in order to reduce $t^{:D}$ to $t_1{}^{:D_1}$, which is again a ground term. The resulting clause is therefore of the shape $\Leftarrow x :: A, x^{:\{A\}} \simeq t_1{}^{:D_1} \; [\![T_1]\!]$, where T_1 is again satisfiable. The result follows then from the induction hypothesis.

□

Before we come to the next theorem, recall that undecorated quasi-reductive SDTRSs are terminating. Since decorated rules may add decorations without changing the structure of the term, it is not obvious that the decorated version terminates, too. Quite on the contrary, in the general case, decorated rewriting can add in infinite number of sorts into the decoration of the same term node, as the following example illustrates:

Example 10.5.4 *Let* $\mathbf{P}_\infty$ *be the following set of decorated rewrite rules:*

$$\left\{ \begin{array}{l} (a^{:v} \to a^{:\{A\}}), \\ (a^{:v} \to a^{:\{list(x)\}}) \Leftarrow a^{:\emptyset} \urcorner x \end{array} \right\}$$

Hence,
$$\begin{array}{lll} a^{:\emptyset} & \rightarrowtail^*_{\mathcal{P}_\infty} & a^{:\{A\}} \\ & \rightarrowtail^*_{\mathcal{P}_\infty} & a^{:\{A,list(x)\}} \; [\![A \subseteq^? x]\!] \\ & \rightarrowtail^*_{\mathcal{P}_\infty} & a^{:\{A,list(x),list(x')\}} \; [\![A \subseteq^? x, list(x) \subseteq^? x']\!] \\ & \vdots & \end{array}$$

This motivates the following property, which provides a restriction implying termination of decorated rewriting with SDTRSs having this property:

Definition 10.5.5 *Let* $\mathcal{P}$ *be a set of condeco clauses that forms an SDTRS.* $\mathcal{P}$ *is* structurally decorating, *if every clause of the shape* $(l^{:v} \to l^{:v \cup D}) \Leftarrow \Gamma$ *in* $\mathcal{P}$ *is structurally decorating, i.e. it satisfies the following condition:*

If Γ' *is the condition obtained from* Γ *by eliminating all conditions of the shape* $x :: A$*, s.t.* $x^{:\{A\}}$ *occurs in* l*, then* $Var(D) \cap Var(\Gamma') = \emptyset$*.*

We illustrate the definition with an example:

Example 10.5.6 *Let* $\mathcal{P}$ *be defined as follows:*

$$\left\{ \begin{array}{l} (f(x^{:\{z\}}, y^{:\{L(z)\}})^{:v} \to f(x^{:\{z\}}, y^{:\{L(z)\}})^{:v \cup \{L(z)\}}) \Leftarrow x :: z, y :: L(z) \\ (g(x^{:\{A\}}, y^{:\{B\}})^{:v} \to g(x^{:\{A\}}, y^{:\{B\}})^{:v \cup \{C\}}) \Leftarrow x :: A, y :: B, x < y \end{array} \right\}$$

Then, $\mathcal{P}$ *is structurally decorating. Now, let* $\mathcal{P}'$ *be:*

$$\left\{ (f(x^{:\{z\}}, y^{:\{L(z)\}})^{:v} \to f(x^{:\{z\}}, y^{:\{L(z)\}})^{:v \cup \{L(z)\}}) \Leftarrow x :: z, y :: L(z), y \urcorner z \right\}$$

$\mathcal{P}'$ *is not structurally decorating, since* z *occurs in the set term to be added to the decoration and in the condition* $y \urcorner z$*.*

This allows us to formulate the following theorem:

Theorem 10.5.7 *Let* $\mathcal{P}_\infty$ *be saturated.*

If $\mathcal{P}_\infty$ *is a finite quasi-reductive and structurally decorating SDTRS, then* $\rightarrowtail_{\mathcal{P}_\infty}$ *is well-founded.*

Proof: Rules whose head has the shape $l^{:D_l} \to r^{:D_r}$ change the structure of the term and hence decrease the size of a term w.r.t. $\succ$, the ordering used to prove quasi-reductivity of $\mathcal{P}_\infty$. Hence, there cannot be an infinite number of steps with these rules in any decorated rewrite sequence using $\mathcal{P}_\infty$, since this would contradict the well-foundedness of $\succ$.

There remain those with a head of the form $l^{:v} \to l^{:v \cup \{A\}}$. Because of our general assumptions, l cannot be a variable. Furthermore, the set terms constraining the variables in A by the matcher must come from strict subterms, since $\mathcal{P}_\infty$ is structurally decorating.

Hence, there is an upper bound on the size of decorations that can be added using $\mathcal{P}_\infty$, namely the number m of rules in $\mathcal{P}_\infty$. Consequently, if $\Subset$ is an ordering comparing terms with the same structure and decorations up to m, which judges more decorated terms to be smaller (cf. Chapter 5), then the lexicographic combination of $\succ$ with $\Subset$ is well-founded and proves the termination of $\mathcal{P}_\infty$. □

There remains the completeness of decorated rewriting. Recall that if $\mathcal{P}_\infty$ be saturated, s.t. it is a quasi-reductive SDTRS, then $\Psi(\mathcal{P}_\infty)$ is ground confluent (cf. Theorem 4.1.17).

Lemma 10.5.8 *Let $\mathcal{P}_\infty$ be saturated, s.t. it is a finite quasi-reductive SDTRS and P be:*

$$\begin{aligned}\Psi(\mathcal{P}_\infty) \quad & \cup\{(f_{\subseteq}(B, A) \to tt) \mid \mathbf{P} \models_{G^1} B \subseteq A\} \\ & \cup\{(z'^{0} : A \to tt \Leftarrow z'^{0} : B \to tt) \mid \mathbf{P} \models_{G^1} B \subseteq A \text{ and } \mathbf{P} \not\models_{G^1} A \subseteq B\} \\ & \cup\{(x^1 : \Omega_1 \to tt)\}\end{aligned}$$

Then P is ground confluent.

Proof: The rules defining the subset relation are of the form $(z' : A \to tt) \Leftarrow (z' : B \to tt)$, s.t. A, B are set terms with $A \succ B$. Due to our general assumptions, there are no individual subterms in A, B and no equations for set terms. Hence, A and B are irreducible and whenever one of these rules is applicable, they always give tt after one step. Since all other rules being applicable on instances of non-variable positions in $x : A$ must have a head of the form $t : C \to tt$, they also yield tt as result. The same is true for $(x^1 : \Omega_1 \to tt)$. This gives us local confluence for P, which gives us together with the termination property of quasi-reductive SDTRSs also confluence of the combination by the critical pair lemma from [Avenhaus et Loría-Sáenz, 1994]. □

Now, we come to the main theorem of this section, which states the completeness of decorated rewriting for quasi-reductive, structurally decorating SDTRSs. Remark that in the second case of the theorem, the constraint $[\![T, \bigvee_{A' \in D''} A' \subseteq^? SNF(A)]\!]$ illustrates how decorated rewriting schematises typing proofs: the finite set D'' of set terms covers all sets (represented by ground set terms A) the decorated term belongs to. This qualifies the condeco rewriting calculus as real calculus with parametric types, since the set of such A's may be infinite.

Theorem 10.5.9 *Let $\mathcal{P}_\infty$ be saturated, s.t. it is a finite quasi-reductive, structurally decorating SDTRS and P be as in Lemma 10.5.8.*

- *For all Σ-ground terms t of sort s'_0:*

$$\begin{aligned}& t \rightarrowtail^!_{\Psi(\mathcal{P}_\infty)} t' \text{ implies } \exists t''^{:D''} [\![T]\!], \\ & \text{s.t. } t^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t''^{:D''} [\![T]\!], \text{ and } (t''^{:D''})_{nd} = t' \text{ and } T \text{ is satisfiable.}\end{aligned}$$

- *For all Σ-ground terms t of sort s'_0 and A (not necessarily ground) of sort s'_1:*

$$\begin{aligned}& t : A \rightarrowtail^!_{\Psi(\mathcal{P}_\infty)} tt \text{ implies } \exists t''^{:D''} [\![T]\!], \\ & \text{s.t. } t^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t''^{:D''} [\![T]\!], \text{ and } [\![T, \bigvee_{A' \in D''} A' \subseteq^? SNF(A)]\!] \text{ is satisfiable.}\end{aligned}$$

- *For all Σ-terms of the shape $f_p(t_1, \ldots, t_m$, such that f_p is a propositional function (with range sort s'_2) introduced by $.^=$ different from : and $f_\subseteq$, where all subterms t of sort s'_0 are ground:*

$$f_p(t_1, \ldots, t_m \rightarrowtail^!_{\Psi(\mathcal{P}_\infty)} tt \text{ implies } \exists T,$$
$$\text{s.t. } f_p(t_1, \ldots, t_m{}^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} tt \; [\![T]\!], \text{ and } [\![T]\!] \text{ is satisfiable.}$$

- *For all Σ-terms A, B (not necessarily ground) of sort s'_1: if $f_\subseteq(A, B) \rightarrowtail^!_{\Psi(\mathcal{P}_\infty)} tt$, then $[\![SNF(A) \subseteq^? SNF(B)]\!]$ is true.*

Hence, $\mathcal{P}_\infty$ is ground confluent and has structurally the same normal forms as P. Additionally, decorated terms in $\mathcal{P}_\infty$-normal form are maximally decorated.

Proof: We proceed by simultaneous induction on $\succ$ over t and $t : A$, respectively. If t or $t : A$ is irreducible in $\Psi(\mathcal{P}_\infty)$, i.e. we are in the base case, then $t' = t$ in the first case and the implication holds, since we are allowed to do zero steps with decorated rewriting. In the second case, since $t : A$ is irreducible, it cannot reduce to tt and the implication holds, since the premise is not satisfied.

Assume that t is reducible by some rule $\Psi(L \Leftarrow L_1, \ldots, L_m \; [\![T]\!])$ in $\Psi(\mathcal{P}_\infty)$. Let Γ be $L_1, \ldots, L_m$. We can assume w.l.o.g. that all reductions in P are bottom-up, since P is ground confluent

Since the evaluation of $\Psi(\Gamma)$ only involves strictly smaller terms than the instantiated left hand side of $\Psi(L)$, we may apply the induction hypothesis to get the satisfaction of Γ in the decorated case:

- If the condition has the shape $u_k{}^{:D_{u_k}} \stackrel{*}{\rightarrow} u'_k{}^{:D_{u'_k}}$, then we know that the corresponding condition $(u_k)_{nd} \stackrel{*}{\rightarrow} (u'_k)_{nd}$ leads to a matcher σ_{k+1}, s.t. $\sigma_{k+1}((u'_k)_{nd}) = (\sigma_k((u_k)_{nd})) \downarrow_P$. Furthermore, for all conditions $(x : B \stackrel{*}{\rightarrow} tt)$ generated by Ψ from conditions $x :: B$, we know that $\sigma_{k+1}(x : B) \rightarrowtail^!_P tt$. Hence, by the induction hypothesis $\sigma_{k+1}(x)^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t_x{}^{:D_x} \; [\![T_x]\!]$ with $[\![T_x, \bigvee_{B' \in D_x} B' \subseteq^? SNF(B)]\!]$ satisfiable and $t_x{}^{:D_x} =_{nd} \sigma_{k+1}(x)^{:\downarrow\emptyset}$, since $\sigma_{k+1}(x)$ is irreducible, which is a consequence of the way conditions are evaluated for SDTRSs (first, we compute $\sigma_k((u_k)_{nd}) \rightarrowtail^!_P u'$, i.e. u' is irreducible and σ_{k+1} is the combination of σ_k and the matcher of $\sigma_k(u_{k+1}) \leq^? u'$, which has therefore only irreducible terms in the image).

  Consequently, σ_{k+1} can be *lifted* to a decorated substitution σ'_{k+1} with $\sigma'_{k+1}(x^{:\{B\}}) = t_x{}^{:D_x}$ for all x in $(u_k)_{nd}, (u'_k)_{nd}$, s.t. $set(x) = B$. Note that the same argumentation allows us to lift the matcher σ_1 of the left hand side of $\Psi(L)$ to a decorated matcher σ'_1, since we assumed bottom-up evaluation, which provides us with the necessary irreducibility. Remark also that by the induction hypothesis $\sigma_k((u_k)_{nd})^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t_k{}^{:D_k} \; [\![T_k]\!]$, such that $t_k{}^{:D_k}$ is without decorations equal to $\sigma_{k+1}((u'_k)_{nd})$ and T_k is satisfiable.

  Finally, we know by the definition of the rewrite relation of SDTRSs in G^n-logics (cf. Section 4.3.2), that $\sigma_{k+1}((u'_k)_{nd}) : \Omega_0 \rightarrowtail^!_{\Psi(\mathcal{P}_\infty)} tt$. Again by induction hypothesis this implies that $t_k{}^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t'_k{}^{:D'_k} \; [\![T'_k]\!]$, such that $[\![T'_k, \bigvee_{A' \in D'_k} A' \subseteq^? \Omega_0]\!]$ is satisfiable. The same derivation, up to some rule applications that are superfluous due to the decorations in $t_k{}^{:D_k}$, may be applied to $t_k{}^{:D_k}$, since this term has the same structure as $t_k{}^{:\downarrow\emptyset}$. Hence, $D_k \neq \emptyset$ and this condition is satisfied.

- If the condition L_k in Γ has the shape $u_k{}^{:D_{u_k}} = u'_k{}^{:D_{u'_k}}$, then σ_k can be lifted as above and the induction hypothesis guarantees that $\sigma_k((u_k)_{nd})^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} \circ =_{nd} \circ \leftarrowtail^!_{\mathcal{P}_\infty} \sigma_k((u'_k)_{nd})^{:\downarrow\emptyset}$ and therefore also $\sigma'_k(u_k{}^{:D_{u_k}}) \rightarrowtail^!_{\mathcal{P}_\infty} \circ =_{nd} \circ \leftarrowtail^!_{\mathcal{P}_\infty} \sigma'_k(u'_k{}^{:D_{u'_k}})$, since adding decorations does not change decorated rewriting as defined in this section.
- If the condition L_k in Γ has the shape $u_k{}^{:D_{u_k}} \curlywedge A$, then $\Psi(L_k) = ((u_k)_{nd} : A \simeq tt)$. The substitution σ_k is lifted as before to σ'_k and $\sigma_k((u_k)_{nd})^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t_k{}^{:D_k} [\![T_k]\!]$, s.t. $[\![T_k, \bigvee_{A' \in D_k} A' \subseteq^? SNF(A)]\!]$ is satisfiable. Hence, the condition is also fulfilled in the decorated setting.
- If the condition L_k in Γ is of the form $x :: A$, then we can use the same argumentation as in the last case.

Hence, the condition Γ is satisfied. Remark that there are no problems to combine the constraints of the different conditions, since there cannot be the same variables in two decorations occurring in two different conditions.

Since t is of sort s'_0 and since individual terms cannot contain propositional functions by the arity restrictions, we know that L has the shape $l^{:D_l} \rightarrow r^{:D_r}$, i.e. $\Psi(L)$ is $(l^{:D_l})_{nd} \rightarrow (r^{:D_r})_{nd}$. Furthermore, by construction, $(\sigma'_{m+1})_{nd} = \sigma_{m+1}$. Hence, we also know that $\sigma'_{m+1}(r^{:D_r})_{nd} = \sigma_{m+1}((r^{:D_r})_{nd})$. Therefore, the result of the application of the decorated rule, say $t'''^{:D'''}$, is identical to the one of the undecorated rule up to decorations. Furthermore, since P is a quasi-reductive SDTRS, the result $(t'''^{:D'''})_{nd}$ of the application of the undecorated rule is strictly smaller than $t^{:D}$, i.e. we can apply the induction hypothesis for the remaining derivation. This gives us a derivation from $t'''^{:\downarrow\emptyset}$ to $t''^{:D''}$, which we can also reuse as a derivation starting with $t'''^{:D'''}$, since decorated rewriting as defined in this section is stable under adding decorations – this is a consequence of the redefinition of decorated matchers, which does not take decorations as static part of the syntax, but as optional information.

Next, consider the case of $t : A$. If the first reduction is in t, then we can reuse the argumentation above. By the ground confluence of P, we may then assume that t is irreducible. Otherwise, the first reduction step using P has to be on top, since there are no equations on set terms and set operators cannot take individual terms as arguments. If the applied rule is a rule defining $\subseteq$, then we can apply the induction hypothesis on the result of the application (remember that $\succ$ is compatible with $\subseteq$), which gives us the theorem immediately.

Otherwise, there must again be a rule $\Psi(L \Leftarrow \Gamma)$ in $\Psi(\mathcal{P}_\infty)$, whose condition is satisfied as before. Now, L has to be of the form $l^{:v} \rightarrow l^{:v \cup \{B\}}$, and $\Psi(L)$ has the shape $l_{nd} : A \simeq tt$. Again, we get that the matcher σ_1 of $t : A$ against $l_{nd} : B$ can be lifted to a decorated matcher σ'_1 of $l^{:\emptyset}$ against $t'^{:\emptyset}$, which is t completely decorated on strict subterms. Hence, we can add $\tau(B)$ with $\tau \leq \sigma_{m+1}$ to the decoration of $t'^{:\emptyset}$. Remark that $\sigma_{m+1}(B) = A$ due to the absence of individual subterms in set terms. Since t is irreducible and $\mathcal{P}_\infty$ shares the termination ordering concerning undecorated terms, we know that reducing $t'^{:\emptyset}$ with $\mathcal{P}_\infty$ cannot change the structure of the term, only the decorations. Furthermore, by definition of decorated rewriting we can only add decorations, i.e. $\tau(B)$ remains in the decoration.

In the case of $f_p(t_1, \ldots, t_m)$, we can apply the induction hypothesis to all subterms $t_1, \ldots, t_n$ and to all conditions in a rule reducing $f_p(t_1, \ldots, t_m)$ in $\Psi(\mathcal{P}_\infty)$. This gives us as in the $t : A$-case the applicability of the corresponding rule in $\mathcal{P}_\infty$. Since no applicable rule yields an unsatisfiable constraint, we get the result.

The last case is trivially satisfied due to our general assumptions – it was only given to state the completeness of deduction. This concludes the theorem. □

Before we can come to a satisfying conclusion, we still need to test sort inheritance. This is the topic of the next section.

6 Completeness for the SI-test

Now that we have proved decorated rewriting in $\mathcal{P}_\infty$ to be complete for term reduction w.r.t. the undecorated case, we turn around and pick up the SI-test of Chapter 8 again, which was formulated in the undecorated case as a possibly infinite set of goal clauses added to the initial presentation, all of which were of the following shape:

$$(\Leftarrow (x : A \simeq tt)_{A \in D}), \text{s.t. } ncs(D)$$

The idea is now to use Theorem 10.5.9 in order to test SI with a finite set of decorated clauses. This is in fact the remaining part in order to get logical equivalence of P_0 with $\mathcal{P}_0$ w.r.t. consistency – recall that we added the above clauses only to P_0, the undecorated initial presentation, and not to $\mathcal{P}_0$, the decorated one.

Theorem 10.6.1 *Let $\mathcal{P}_\infty$ be saturated, s.t. it is a (finite) quasi-reductive, structurally decorating SDTRS. Let furthermore $\mathcal{S}'$ be the (finite) set of set terms A, s.t. a rule of the form $(l^{:v} \to l^{:v \cup \{A\}}) \Leftarrow \Gamma'$ is in $\mathcal{P}_\infty$. Then:*

P, *the G^1-logical presentation used to construct P_0 and $\mathcal{P}_0$, is SI,*
iff
P_0 is consistent,
iff
$\mathcal{P}_\infty$ plus the following set of clauses is consistent:

$$\{(\Leftarrow \Gamma_1, \ldots, \Gamma_m)\ [\![T_1, \ldots, T_m, l_1{}^{:\emptyset} =^? l_2{}^{:\emptyset}, \ldots l_1{}^{:\emptyset} =^? l_m{}^{:\emptyset}]\!] \mid \forall i \in [1..m], ((l_i{}^{:v} \to l_i{}^{:v \cup \{A_i\}}) \Leftarrow \Gamma_i\ [\![T_i]\!]) \in \mathcal{P}_\infty \text{and } ncs(\{A_1, \ldots, A_m\})\}$$

Furthermore, if there is inconsistency, then a witness in form of a ground individual term destroying SI in **P** *can be constructed.*

Proof: Assume there is a ground term t witnessing non-SI of **P**, i.e. t belongs to some finite number of sets represented by ground set terms, say $B_1, \ldots, B_{m'}$. W.l.o.g., we can assume t to be minimal w.r.t. $\succ$, i.e. also irreducible w.r.t. $\mathcal{P}_\infty$, since we know that in G^1-logics equal terms belong to equal sets. Hence, we get by Theorem 10.5.9 that $t^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t'^{:D'}\ [\![T]\!]$, s.t. $t_{nd} = t'_{nd}$ and $[\![T, \bigwedge_{i \in [1..m']} \bigvee_{B' \in D'} (B' \subseteq^? SNF(B_i))]\!]$ is satisfiable. Hence, there must be a set of rules $\phi_1, \ldots, \phi_{m'}$ with left hand sides $l_i{}^{:v}$, $i \in [1..m']$, which add these decorations to the top node and there are corresponding decorated matchers $\sigma_1, \ldots, \sigma_{m'}$, s.t:

$$\forall i \in [1..m'], t'^{:\emptyset} =_{nd} \sigma_i(l_i{}^{:\emptyset})$$

Hence, the left hand sides of the rules have to be unifiable assuming SI, since no strict subterms of t can be a witness for non-SI, due to the minimality of t, and since all rules adding decorations have non-variable left hand sides, due to the definition of these rules. This proves the completeness of the above test.

For the soundness, assume that there is a solution to one of the above goals, i.e. we can derive the empty clause from it. Then, clearly there is a term t at which all of the rules $\phi_1, \ldots, \phi_{m'}$ used to construct the goal can be applied: just unify their left hand sides and discard the decorations.

Hence, all free individual variables x in t correspond with conditions of the form $x :: A$ in the rules $\phi_1, \ldots, \phi_{m'}$. Since we have derived the empty clause, all these variables have to be instantiated by ground terms. Assume w.l.o.g. that *variable elimination* was used exhaustively during the derivation. Therefore, we are sure that the composition of all substitutions used with *variable elimination* during the derivation of the empty clause applied to t gives a term t', s.t. t'_{nd} is a witness for non-SI of $\mathbf{P}$ by the soundness of decorated rewriting. □

Remark that the set of clauses can be reduced to those combinations of decoration rules with the same top function symbol on the left hand side of the head.

Corollary 10.6.2 *Let $\mathcal{P}_\infty$ be saturated, s.t. it is a finite quasi-reductive, structurally decorating SDTRS and the finite set of goals defined in the last theorem do not have a solution. Then, $\mathbf{P}$ is SI and using $\mathcal{P}_\infty$ as SDTRS gives a sound and complete operational semantics on ground equations in $\mathbf{P}$, i.e. for all ground individual terms t, t':*

$$\Theta(P_0^{rel}) \models_{G^1} t = t' \text{ iff } t^{:\downarrow\emptyset} \rightarrowtail^!_{\mathcal{P}_\infty} t''^{:D''} \; [\![T]\!] =_{nd} t'''^{:D'''} \rightarrowtail^!_{\mathcal{P}_\infty} t'^{:\downarrow\emptyset},$$
$$\text{s.t. } D'' \neq \emptyset \text{ and } [\![T]\!] \text{ is satisfiable}$$

And for all ground individual terms t, all set terms A:

$$\Theta(P_0^{rel}) \models_{G^1} t \in A \text{ iff } \exists t'^{:D'}, t^{:\downarrow\emptyset} \rightarrowtail^*_{\mathcal{P}_\infty} t'^{:D'} \; [\![T]\!],$$
$$\text{s.t. } [\![T, \bigvee_{A' \in D'} A' \subseteq^? SNF(A)]\!] \text{ is satisfiable,}$$

where $t'^{:D'}$ is a decorated term. Furthermore, for all ground individual terms t:

$$\Theta(P_0^{rel}) \models_{G^1} EX\ t \text{ iff } \exists t'^{:D'}, t^{:\downarrow\emptyset} \rightarrowtail^*_{\mathcal{P}_\infty} t'^{:D'} \; [\![T]\!], \text{s.t. } D' \neq \emptyset \text{ and } [\![T]\!] \text{ is satisfiable,}$$

Proof: Lemma 10.5.3 and Theorems 10.5.7, 10.5.9 prove soundness, termination and completeness of decorated rewriting for ground equations in $\mathbf{P}$, assuming SI. Only the last claim needs a little explanation: it follows from the definition of Θ saying that $t \in \Omega_0$ after Ξ (the quasi-inverse of Θ) and $EX\ t$ are equivalent. The claim then follows from the one on membership formulas and $A \subseteq \Omega_0$ for all $A \in \mathcal{S}$. Finally, Theorem 10.6.1 guarantees us that $\mathbf{P}$ is SI iff the goals defined there have a solution. Since we assumed that they do not have a solution, we get the required results. □

7 Looking for Standard Elements in Sets

In this section, we suggest a test for standard elements, i.e. individuals of sort s_0, in sets during saturation. Remark that $choose(A)$ is not of this sort, since it is non-standard – it was added only for model theoretical issues. in the following, we suggest a procedure collecting set terms A, for which we can find an inhabitance witness. If A is in this collection, then we may omit not only the variable membership conditions $x :: A$ for variables x occurring in the head of a

clause when we use decorated rewriting for SDTRSs. We can moreover omit those $x :: A$, where x occurs only in variable membership conditions and not in the constraint.

This serves mainly for getting the same effect as in "Schubert's Steamroller" [Walther, 1985], which is based on the fact that order-sorted resolution and paramodulation stop when the remaining goal to solve results in a variable unification problem. This is possible, since all sets in [Walther, 1985] (there called sorts) are known to be non-empty. Our procedure as described so far would try to continue with the proof of non-emptiness of the sets A for variable membership conditions $x :: A$ corresponding with the variables x to be unified, similar to Weidenbach's approach [Weidenbach et Ohlbach, 1990; Weidenbach, 1991], where non-emptiness is added in form of literals to the premise of a clause. Hence, the goal is to perform this proof a priori.

The idea is simply to use all syntactically decorating rules in order to test if their variable membership conditions are satisfied. These rules are formally defined as follows:

Definition 10.7.1 *Let* $(l^{:v} \to l^{:v \cup \{B\}}) \Leftarrow \Gamma$ *be a condeco clause and* Γ' *be* Γ *without the variable membership conditions* $x :: A$, *s.t.* $x^{:\{A\}}$ *occurs in* l. *If* $\Gamma' = \emptyset$, *then this clause is* syntactically decorating.

Hence, every syntactically decorating clause has only variable membership conditions for variables occurring in l in the premise. Assume $(l^{:v} \to l^{:v \cup \{B\}}) \Leftarrow (x_i :: A_i)_{i \in [1..m]}$ is syntactically decorating. Then, we may add the set term B to our collection, if all A_i, $i \in [1..m]$, are subsumed by some set term in our collection. More formally, let $NES_0 = \emptyset$, where NES stands for non-empty sets, and NES_i be defined by the following equality:

$$NES_{i+1} = NES_i \cup \{B \mid ((l^{:v} \to l^{:v \cup \{B\}}) \Leftarrow (x_j :: A_j)_{j \in [1..m]}) \in \mathcal{P}_i \text{ is syntactically decorating and } \forall j \in [1..m], \exists A'_j \in NES_i, \text{ s.t. } [\![A'_j \subseteq^? SNF(A_j)]\!] \text{ is satisfiable}\}$$

Lemma 10.7.2 *If* $A \in NES_i$, *then* $\exists t, \mathbf{P} \models_{G^1} t \in A$, *where* t *is ground.*

Proof: By Lemma 10.4.8, we know that during a saturation with GCD starting with $\mathcal{P}_0$, $x :: A$ is in the premise, if $x^{:\{A\}}$ occurs in the corresponding clause. Hence, we can associate, by induction on i, a ground term t to each $B \in NES_i$. This is possible, since we may replace any x_j in l with $x_j :: A_j$ in the premise of the clause by the witness for A_j, which has to exist since there is a $A'_j \in NES_i$, s.t. $[\![A'_j \subseteq^? SNF(A_j)]\!]$ is satisfiable. □

A possible extension of this would be to drop SNF in $[\![A'_j \subseteq^? SNF(A_j)]\!]$, but then we can only add $\sigma(B)$ for all solutions of $[\![A'_j \subseteq^? A_j]\!]$. Furthermore, it is then reasonable to replace $\forall j \in [1..m], \exists A'_j \in NES_i$ by $\forall j \in [1..m], \exists A'_j \in NES_{i+1}$, while mentioning that NES_{i+1} is the smallest such set. Even if we assume that the just mentioned set of substitutions σ can be finitely schematised, we still can get problems with termination. Consider the following example:

Example 10.7.3 *Let* $(f(x^{:\{z\}})^{:v} \to f(x^{:\{z\}})^{:v \cup \{L(z)\}} \Leftarrow x :: z)$ *and* $(n^{:v} \to n^{:v \cup \{A\}})$ *be in* $\mathcal{P}_0$. *Then, we get immediately that* A *is in* NES_1. *Furthermore, using* $[\![A'_j \subseteq^? A_j]\!]$, *we would get that* $L(A)$ *is also in* NES_1 *and then* $L(L(A)), L(L(L(A)))$ *and so on.*

Hence, the fact that we do not allow the set variables in the A_j to be instantiated provides us with a practical collection of non-empty sets in the standard sense. Remark that in the case of a finite set of set terms, this results in the procedure suggested for G-algebra and $\mathcal{OSE}$-logic.

Let us now come to the use of these sets. A simple inference rule eliminating isolated variable membership conditions is as follows, where NES stands for the current NES_i:

variable membership elimination:

$$\frac{L \Leftarrow \Gamma, x :: A \; [\![T]\!]}{L \Leftarrow \Gamma \; [\![T]\!]} \quad \begin{array}{l}\text{if } \exists A' \in NES, [\![A' \subseteq^? SNF(A)]\!] \text{ is satisfiable} \\ \text{and } x \notin \mathcal{V}ar(L, \Gamma, T)\end{array}$$

Lemma 10.7.4 *The rule* variable membership elimination *is a simplification rule.*

Proof: Let $wit(A)$ stands for the non-emptiness witness of A for all $A \in NES$. We have to prove two things: first, the soundness of the rule and second, that whenever $C \vdash C'$ using the rule, then C is subsumed by C'. Soundness follows from the fact that $x \notin \mathcal{V}ar(L, \Gamma, T)$ and that $\mathbf{P} \models_{G^1} wit(A) : A$, since there is an $A' \in NES$ with this property and for each instance of A, there is an instance of A', that is a subset of A. Subsumption of C by C' is obvious. □

Although this allows us already to get rid of isolated variables, that disappeared from the rest of the clause, e.g., by simplification, we still do not get the effect of Schubert's Steamroller. There, we need to preview also equational constraints on variables. Consider therefore the following extended rule:

extended variable membership elimination:

$$\frac{L \Leftarrow \Gamma, (x_i :: A_i)_{i\in[1..m]} \; [\![(x_1 =^? x_i)_{i\in[2..m]}, T]\!]}{L \Leftarrow \Gamma \; [\![(x_1 =^? x_i)_{i\in[2..m]}, T]\!]} \quad \begin{array}{l}\text{if } \exists A' \in NES, \text{ s.t.} \\ [\![\bigwedge_{i\in[1..m]} A' \subseteq^? SNF(A_i)]\!] \text{ is satisfiable} \\ \text{and } \forall i \in [1..m], x_i \notin \mathcal{V}ar(L, \Gamma, T)\end{array}$$

Although this seems to be expensive to test, it allows for example to stop superposition into goal clauses, when they only contain variable membership conditions and their satisfiability is already known. This is in the spirit of order-sorted unification and the example of Schubert's Steamroller, where the efficiency of deduction is raised by order-sorted variable unification instead of the enumeration of all ground solutions. Hence, this corresponds with the schematisation of term sets. Since the construction of NES is a conservative extension of the non-empty sort tests for classical order-sorted logics, we know that Schubert's Steamroller also works with $\mathcal{GCD}$. As before, we get:

Lemma 10.7.5 *The rule* extended variable membership elimination *is a simplification rule.*

Proof: As before. □

8 Examples

In this section we will give an intuition and compare the decorated calculus with the one of Chapters 5 to 9 using some examples.

The first example illustrates the detection of non-SI:

Example 10.8.1 *Let* $\mathbf{P} = \{a \in A,\ a \in B,\ f(x^0) \in A \Leftarrow x^0 \in A,\ f(y^0) \in B \Leftarrow x^0 \in B\}$. *Hence* $\mathcal{P}_0$ *is the following set of decorated clauses:*

$$\left\{ \begin{array}{l} (f(x^{:\{A\}})^{:v} \to f(x^{:\{A\}})^{:v\cup\{A\}}) \Leftarrow x :: A, \\ (f(y^{:\{B\}})^{:v} \to f(y^{:\{B\}})^{:v\cup\{B\}}) \Leftarrow y :: B, \\ (a^{:v} \to a^{:v\cup\{A\}}), \\ (a^{:v} \to a^{:v\cup\{B\}}) \end{array} \right\}$$

This set is trivially saturated, since we do not have overlaps with decoration rules. In order to test SI – remark that A and B are incomparable – we have to add two clauses concerning the SI-test:

$$\left\{ \begin{array}{l} \Leftarrow x :: A, y :: B\ [\![f(x^{:\{A\}})^{:\emptyset} =^? f(y^{:\{B\}})^{:\emptyset}]\!] \\ \Leftarrow [\![a^{:\emptyset} =^? a^{:\emptyset}]\!] \end{array} \right\}$$

Clearly, the constraint of the second (empty) clause is satisfiable and hence the SI test gives us a as witness for non-SI.

This was a rather trivial example to start with. Let us consider next an example where the approach using decorations as part of the syntax failed due to divergency (cf. Example 8.2.32):

Example 10.8.2 *Let:*

$$\mathcal{P} = \left\{ \begin{array}{l} 0 \in Even, \\ succ(0) \in Odd, \\ succ(succ(x)) \in Even \Leftarrow x :: Even, \\ succ(succ(y)) \in Odd \Leftarrow y :: Odd \end{array} \right\}$$

Hence $\mathcal{P}_0$ *is the following set of decorated clauses:*

$$\left\{ \begin{array}{l} (0^{:v} \to 0^{:v\cup\{Even\}}), \\ (succ(0^{:\emptyset})^{:v} \to succ(0^{:\emptyset})^{:v\cup\{Odd\}}) \\ (succ(succ(x^{:\{Even\}})^{:\emptyset})^{:v} \to succ(succ(x^{:\{Even\}})^{:\emptyset})^{:v\cup\{Even\}}) \Leftarrow x :: Even, \\ (succ(succ(y^{:\{Odd\}})^{:\emptyset})^{:v} \to succ(succ(y^{:\{Odd\}})^{:\emptyset})^{:v\cup\{Odd\}} \Leftarrow y :: Odd, \end{array} \right\}$$

This set is again trivially saturated, since we do not have overlaps with decoration rules. Since Odd and Even are incomparable, we must try to add a clause concerning the SI-test:

$$\left\{ \Leftarrow x :: Even, y :: Odd\ [\![succ(succ(x^{:\{Even\}})^{:\emptyset})^{:\emptyset} =^? succ(succ(y^{:\{Odd\}})^{:\emptyset})^{:\emptyset}]\!] \right\}$$

But the constraint of this clause is unsatisfiable due to **VarClash** *and hence* **P** *is proved to be sort inheriting.*

Remark that we just proved an inductive theorem by consistency:

$$\forall t \in \mathbf{T}_{s_0}, \neg(t \in Odd \wedge t \in Even)$$

This extends the usual techniques of proving inductive theorems by consistency in the sense that we may have partial functions involved.

Next, let us consider the presentation from Example 4.3.5, including sort constraints.

Example 10.8.3 *The corresponding initial presentation $\mathcal{P}_0$ is, assuming the subset axioms to be implicit:*

$$
\left\{
\begin{array}{lll}
(0^{:v} \to 0^{:v\cup\{Nat\}}), & & \\
(s(x^{:\{Nat\}})^{:v} \to s(x^{:\{Nat\}})^{:v\cup\{Nat\}}) & \Leftarrow & x :: Nat, \\
\hline
(\leq (0^{:\emptyset}, x^{:\{Nat\}}) \to tt) & \Leftarrow & x :: Nat, \\
(\leq (s(x^{:\{Nat\}})^{:\emptyset}, s(y^{:\{Nat\}})^{:\emptyset}) \to tt) & \Leftarrow & x :: Nat, y :: Nat, \\
(< (0^{:\emptyset}, s(x^{:\{Nat\}})^{:\emptyset}) \to tt & \Leftarrow & x :: Nat, \\
(< (s(x^{:\{Nat\}})^{:\emptyset}, s(y^{:\{Nat\}})^{:\emptyset}) \to tt) & \Leftarrow & < (x^{:\{Nat\}}, y^{:\{Nat\}}) \xrightarrow{*} tt, \\
 & & x :: Nat, y :: Nat, \\
\hline
(Omset \subseteq List), & & \\
(Oset \subseteq Omset), & & \\
\hline
(\epsilon^{:v} \to \epsilon^{:v\cup\{Oset\}}), & & \\
((x^{:\{Nat\}}.\epsilon^{:\emptyset})^{:v} \to (x^{:\{Nat\}}.\epsilon^{:\emptyset})^{:v\cup\{Oset\}}) & \Leftarrow & x :: Nat, \\
((x^{:\{Nat\}}.(x'^{:\{Nat\}}.y^{:\{Oset\}})^{:\{\emptyset\}})^{:v} \to & & \\
(x^{:\{Nat\}}.(x'^{:\{Nat\}}.y^{:\{Oset\}})^{:\{\emptyset\}})^{:v\cup\{Oset\}}) & \Leftarrow & < (x^{:\{Nat\}}, x'^{:\{Nat\}}) \xrightarrow{*} tt, \\
 & & (x'^{:\{Nat\}}.y^{:\{Oset\}})^{:\emptyset} \curlywedge Oset, \\
 & & x :: Nat, x' :: Nat, y :: Oset), \\
\hline
((x^{:\{Nat\}}.(x'^{:\{Nat\}}.y^{:\{Omset\}})^{:\{\emptyset\}})^{:v} \to & & \\
(x^{:\{Nat\}}.(x'^{:\{Nat\}}.y^{:\{Omset\}})^{:\{\emptyset\}})^{:v\cup\{Omset\}}) & \Leftarrow & \leq (x^{:\{Nat\}}, x'^{:\{Nat\}}) \xrightarrow{*} tt, \\
 & & (x'^{:\{Nat\}}.y^{:\{Oset\}})^{:\emptyset} \curlywedge Omset, \\
 & & x :: Nat, x' :: Nat, y :: Oset), \\
\hline
((x^{:\{Nat\}}.y^{:\{List\}})^{:v} \to (x^{:\{Nat\}}.y^{:\{List\}})^{:v\cup\{List\}}) & \Leftarrow & x :: Nat, y :: List)
\end{array}
\right\}
$$

This set is saturated using the ordering of Example 4.3.5. The resulting decorated term rewriting system is trivially sort-inheriting, since there is no clause set for the test is empty - remark that $Omset, Oset$ and $List$ are comparable w.r.t. $\subseteq$.

Let us now consider the quick-sort-algorithm of Section 3.2, using polymorphic lists this time and illustrating inheritance in an object-oriented style for arrays and stacks implemented by lists:

Example 10.8.4 *We start with the definition of natural numbers with comparison, reusing the sets Odd and Even from the last example:*

$$
\begin{array}{lll}
x^0 & \in Nat & \Leftarrow x^0 \in Odd \\
x^0 & \in Nat & \Leftarrow x^0 \in Even \\
leq & \in Cmp & \\
gtr & \in Cmp & \\
cmp(0, x^0) & = leq & \Leftarrow x^0 \in Nat \\
cmp(s(x^0), 0) & = gtr & \Leftarrow x^0 \in Nat \\
cmp(s(x^0), s(y^0)) & = cmp(x^0, y^0) & \Leftarrow x^0 \in Nat, y^0 \in Nat
\end{array}
$$

This is translated into:

$$
\begin{array}{ll}
Odd \subseteq Nat & \\
Even \subseteq Nat & \\
(leq^{:v} \to leq^{:v \cup \{Cmp\}}) & \\
(gtr^{:v} \to gtr^{:v \cup \{Cmp\}}) & \\
(cmp(0^{:\emptyset}, x^{:\{Nat\}})^{:\emptyset} \to leq^{:\emptyset}) & \Leftarrow x :: Nat \\
(cmp(s(x^{:\{Nat\}})^{:\emptyset}, 0^{:\emptyset})^{:\emptyset} \to gtr^{:\emptyset}) & \Leftarrow x :: Nat \\
(cmp(s(x^{:\{Nat\}})^{:\emptyset}, s(y^{:\{Nat\}})^{:\emptyset})^{:\emptyset} \to & \\
\qquad cmp(x^{:\{Nat\}}, y^{:\{Nat\}})^{:\emptyset}) & \Leftarrow x :: Nat, y :: Nat
\end{array}
$$

Remark that this part alone is saturated for $\succ$ being an LPO generated by the precedence $cmp \succ gtr \succ leq \succ Cmp \succ s \succ 0 \succ Nat \succ Odd \succ Even$, since there are no superpositions possible. It is also SI for the same reason. Next, we define and polymorphic lists:

$$
\begin{array}{lll}
nil & \in List(x^1) & \\
cons(x^0, y^0) & \in List(z^1) & \Leftarrow x^0 \in z^1, y^0 \in List(z^1) \\
car(cons(x^0, y^0)) & = x^0 & \Leftarrow x^0 \in z^1, y^0 \in List(z^1) \\
cdr(cons(x^0, y^0)) & = y^0 & \Leftarrow x^0 \in z^1, y^0 \in List(z^1) \\
append(nil, y^0) & = y^0 & \Leftarrow y^0 \in List(z^1) \\
append(cons(x^0, y_1^0), y_2^0) & = & \\
\qquad cons(x^0, append(y_1^0, y_2^0)) & \Leftarrow x^0 \in z^1, y_1^0 \in List(z^1), y_2^0 \in List(z^1) &
\end{array}
$$

This becomes:

$$
\begin{array}{ll}
(nil^{:v} \to nil^{:v \cup \{List(x)\}}) & \\
(cons(x^{:\{z\}}, y^{:\{List(z)\}})^{:v} \to & \\
\qquad cons(x^{:\{z\}}, y^{:\{List(z)\}})^{:v \cup \{List(z)\}}) & \Leftarrow x :: z, y :: List(z) \\
(car(cons(x^{:\{z\}}, y^{:\{List(z)\}})^{:\emptyset})^{:\emptyset} \to x^{:\{z\}}) & \Leftarrow x :: z, y :: List(z) \\
(cdr(cons(x^{:\{z\}}, y^{:\{List(z)\}})^{:\emptyset})^{:\emptyset} \to y^{:\{List(z)\}}) & \Leftarrow x :: z, y :: List(z) \\
(append(nil^{:\emptyset}, y^{:\{List(z)\}})^{:\emptyset} \to y^{:\{List(z)\}}) & \Leftarrow y :: List(z) \\
(append(cons(x^{:\{z\}}, y_1^{:\{List(z)\}})^{:\emptyset}, y_2^{:\{List(z)\}})^{:\emptyset} \to & \\
\quad cons(x^{:\{z\}}, append(y_1^{:\{List(z)\}}, y_2^{:\{List(z)\}})^{:\emptyset})^{:\emptyset}) & \Leftarrow x :: z, y_1 :: List(z), y_2 :: List(z)
\end{array}
$$

Remark that now, our definition of multiset expressions for equations allows us to consider set variables in conditions, However, for the third to the last axiom, we have a set variable z in the body of the clause that does not occur in the head. Hence, all set terms need to be smaller than individual terms. We can achieve this using $append \succ car \succ cdr \succ cons \succ nil \succ List$ as precedence, which can be merged with the one for natural numbers, s.t. set constructors remain all below individual term constructors. Next, we may define polymorphic stacks as subsets of lists with particular properties:

$$
\begin{array}{lll}
x^0 & \in List(y^1) & \Leftarrow x^0 \in Stack(y^1) \\
mt\_stack & \in Stack(x^1) & \\
push(x^0, y^0) & \in Stack(z^1) & \Leftarrow x^0 \in z^1, y^0 \in Stack(z^1) \\
mt\_stack & = nil & \\
push(x^0, y^0) & = cons(x^0, y^0) & \Leftarrow x^0 \in z^1, y^0 \in Stack(z^1) \\
top(push(x^0, y^0)) & = x^0 & \Leftarrow x^0 \in z^1, y^0 \in Stack(z^1) \\
pop(push(x^0, y^0)) & = y^0 & \Leftarrow x^0 \in z^1, y^0 \in Stack(z^1)
\end{array}
$$

This is translated into:

$$
\begin{array}{ll}
(Stack(y) \subseteq List(y^1)) & \\
(mt\_stack^{:v} \to mt\_stack^{:v \cup \{Stack(x^1)\}}) & \\
(push(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:v} \to & \\
\qquad push(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:v \cup \{Stack(z)\}}) & \Leftarrow x :: z, y :: Stack(z) \\
mt\_stack^{:\emptyset} \to nil^{:\emptyset} & \\
(push(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:\emptyset} \to & \\
\qquad cons(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:\emptyset}) & \Leftarrow x :: z, y :: Stack(z) \\
(top(push(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:\emptyset})^{:\emptyset} \to x^{:\{z\}}) & \Leftarrow x :: z, y :: Stack(z) \\
(pop(push(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:\emptyset})^{:\emptyset} \to y^{:\{Stack(z)\}}) & \Leftarrow x :: z, y :: Stack(z)
\end{array}
$$

The same remarks concerning termination are true here: we may prove decreasingness using any LPO containing $mt\_stack \succ nil$ *and* $push \succ cons$ *in its precedence and inserting List in the set term part. Similarly, we define polymorphic arrays with direct access functions implemented as lists :*

$$
\begin{array}{lll}
x^0 & \in List(y^1) & \Leftarrow x^0 \in Array(y^1) \\
mt\_array & \in Array(x^1) & \\
column(x^0, y^0) & \in Array(z^1) & \Leftarrow x^0 \in z^1, y^0 \in Array(z^1) \\
mt\_array & = nil & \\
column(x^0, y^0) & = cons(x^0, y^0) & \Leftarrow x^0 \in z^1, y^0 \in Array(z^1) \\
first(column(x^0, y^0)) & = x^0 & \Leftarrow x^0 \in z^1, y^0 \in Array(z^1) \\
rest(column(x^0, y^0)) & = y^0 & \Leftarrow x^0 \in z^1, y^0 \in Array(z^1) \\
create\_array(x^0, y^0) & \in Array(z^1) & \Leftarrow x^0 \in Nat, y^0 \in z^1 \\
create\_array(0, y^0) & = mt\_array & \\
create\_array(s(x^0), y^0) & = & \\
\qquad column(y^0, create\_array(x^0, y^0)) & & \Leftarrow x^0 \in Nat \\
get(0, column(x^0, y^0)) & = x^0 & \Leftarrow x^0 \in z^1, y^0 \in Array(z^1) \\
get(s(x^0), column(y^0, z^0)) & = get(x^0, z^0) & \Leftarrow x^0 \in Nat, y^0 \in z^1, z^0 \in Array(z^1)
\end{array}
$$

These become:

$$
\begin{array}{ll}
(Array(y) \subseteq List(y)) & \\
(mt\_array^{:v} \to mt\_array^{:v \cup \{Array(x)\}}) & \\
(column(x^{:\{z\}}, y^{:\{Array(z)\}})^{:v} \to & \\
\qquad column(x^{:\{z\}}, y^{:\{Array(z)\}})^{:v \cup \{Array(z)\}}) & \Leftarrow x :: z, y :: Array(z) \\
(mt\_array^{:\emptyset} \to nil^{:\emptyset}) & \\
(column(x^{:\{z\}}, y^{:\{Array\}})^{:\emptyset} \to cons(x^{:\{z\}}, y^{:\{Array\}})^{:\emptyset}) & \Leftarrow x :: z, y :: Array(z) \\
(first(column(x^{:\{z\}}, y^{:\{Array(z)\}})^{:\emptyset})^{:\emptyset} \to x^{:\{z\}}) & \Leftarrow x :: z, y :: Array(z) \\
(rest(column(x^{:\{z\}}, y^{:\{Array(z)\}})^{:\emptyset})^{:\emptyset} \to y^{:\{Array(z)\}}) & \Leftarrow x :: z, y :: Array(z) \\
(create\_array(x^{:\{z\}}, y^{:\{z\}})^{:v} \to & \\
\qquad create\_array(x^{:\{Nat\}}, y^{:\{z\}})^{:v \cup \{Array(z)\}}) & \Leftarrow x :: Nat, y :: z \\
(create\_array(0^{:\emptyset}, y^{:\{z\}})^{:\emptyset} \to mt\_array^{:\emptyset}) & \Leftarrow y :: z \\
(create\_array(s(x^{:\{Nat\}})^{:\emptyset}, y^{:\{z\}})^{:\emptyset} \to & \\
\qquad column(y^{:\{z\}}, create\_array(x^{:\{Nat\}}, y^{:\{z\}})^{:\emptyset})^{:\emptyset}) & \Leftarrow x :: Nat, y :: z \\
(get(0^{:\emptyset}, column(x^{:\{z\}}, y^{:\{Array(z)\}})^{:\emptyset})^{:\emptyset} \to x^{:\{z\}}) & \Leftarrow x :: z, y :: Array(z) \\
(get(s(x^{:\{Nat\}})^{:\emptyset}, column(y^{:\{z_1\}}, z_0^{:\{Array(z_1)\}})^{:\emptyset})^{:\emptyset} \to & \\
\qquad get(x^{:\{z_1\}}, z_0^{:\{Array(z_1)\}})^{:\emptyset}) & \Leftarrow x :: Nat, y :: z_1, z_0 :: Array(z_1)
\end{array}
$$

Again, we may discriminate the set term constructor Array as being lower than all individual operators and use an LPO with $create\_array \succ mt\_array, column, s$ *to prove decreasingness.*

Furthermore, all of the given presentations so far do not allow for overlaps, except replacing mt_stack and mt_array, respectively, by nil, and push, column by cons. This gives us the following additional rules:

$$
\begin{array}{ll}
(nil^{:v} \to nil^{:v \cup \{Stack(x^1)\}}) & \\
(cons(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:v} \to & \\
(cons(x^{:\{z\}}, y^{:\{Stack(z)\}})^{:v \cup \{Stack(z)\}} & \Leftarrow x :: z, y :: Stack(z) \\
(nil^{:v} \to nil^{:v \cup \{Array(x)\}}) & \\
(cons(x^{:\{z\}}, y^{:\{Array(z)\}})^{:v} \to & \\
cons(x^{:\{z\}}, y^{:\{Array(z)\}})^{:v \cup \{Array(z)\}}) & \Leftarrow x :: z, y :: Array(z)
\end{array}
$$

Hence, after these inferences, the whole presentation is saturated and we may try to test sort inheritance. The only clause generated stems from an overlap of the two newly generated clauses for nil:

$$(\Leftarrow [\![nil^{:\emptyset} =^? nil^{:\emptyset}]\!])$$

Remark that overlapping the other two clauses leads to unification of a variable of sort $Array(z)$ and one of $Stack(z)$, which fails due to the sort-inheritance assumption. The constraint of this empty clause is satisfiable and hence we have found a counter-example for sort-inheritance: nil. Clearly, this constant belongs to $Stack(z)$ and $Array(z')$, for all instantiations of z and z', respectively. Hence, we have to introduce a new subset for these sets. Let Nil be this set and $Nil \subseteq Stack(z), Nil \subseteq Array(z'), nil \in Nil$ be added to the presentation. The extended set of clauses is still saturated. Testing again SI gives no more clauses for the test and hence the resulting presentation is SI.

Now, we can define quick-sort for polymorphic lists, which can then also be used to sort stacks and arrays, i.e. we have inheritance.

$$
\begin{array}{lll}
qsort(nil) & = nil & \\
qsort(cons(x^0, y^0)) & = z^0 & \Leftarrow cons(x^0, y^0) \in list(z^1) \\
 & & split(y^0, x^0) = pair(y_1^0, y_2^0), \\
 & & qsort(y_1^0) = z_1^0, \\
 & & qsort(y_2^0) = z_2^0, \\
 & & append(z_1^0, cons(x^0, z_2^0)) = z^0 \\
split(nil, x^0) & = nil & \\
split(cons(x_1^0, y^0), x_2^0) & = pair(cons(x_1^0, y_1^0), y_2^0) & \Leftarrow x_2^0 \in z^1, \\
 & & cons(x_1^0, y^0) \in list(z^1), \\
 & & cmp(x_1^0, x_2^0) = leq, \\
 & & split(y^0, x_2^0) = pair(y_1^0, y_2^0) \\
split(cons(x_1^0, y^0), x_2^0) & = pair(y_1^0, cons(x_1^0, y_2^0)) & \Leftarrow x_2^0 \in z^1, \\
 & & cons(x_1^0, y^0) \in list(z^1), \\
 & & cmp(x_1^0, x_2^0) = leq, \\
 & & split(y^0, x_2^0) = pair(y_1^0, y_2^0)
\end{array}
$$

Remark that this represents a SDTRS, at least semantically. Finding a total ordering on ground terms proving this is another problem. This part hence shows the limits of our approach so far, which mainly depends on the termination orderings that we can use. The totality requirement is very strong. One may use a weakening[26] of the calculus adapted to any simplification ordering [Bachmair et Ganzinger, 1994a].

[26] In the sense that there are typically more critical pairs to be computed.

9 Conclusion from Decorated Clause Theorem Proving with Constraints

We have developed a saturation calculus for (the second-order) G^1-logics, with non-standard models including partial functions and proved refutational completeness, using decorated terms. The approach can be seen as extension of the one in Chapters 5 to 9, in that it uses the same ideas from a proof theoretic point of view: existential formulas become typing, typing becomes decorating terms and equational reasoning can be done using decorated unifiers for replacing equal by equal. However, there is a significant difference between the two approaches: in the first for G-algebras, we used decoration as a fixed part of the syntax. In the second for G^1-logics, we used decorations as optional information. The examples in this chapter have shown that this makes the SI test terminate in some cases, where the old test diverged, due to useless superpositions.

Additionally, we can now manage infinite numbers of sets of individuals, corresponding with sorts in G-algebra, due to the possibility of parameterisation. Furthermore, sort constraints [Jouannaud *et al.*, 1992], i.e. conditional typing rules, can be handled in a uniform way with the other typing rules. Set terms can be taken as arguments to functions, which gives us the possibility to add real higher-order features in the future. However, we are not able to add equations on set terms for the moment, since this would result in problems for the parametric subsort calculus needed to solve our constraints. We may conclude that the calculus looks very promising and that there is still much work to be done.

The following questions, that we did not investigate in detail, since they appeared to be independent from the decorated term principle, are particular interesting:

- Is it possible to develop a non-emptiness test for all instantiations of set terms A, s.t. a condition of the shape $x :: A$ belongs to a clause in the saturated presentation ? This would allow us to discard this condition. For variables occurring in the left hand side of the head literal or an oriented condition, this is possible without problems. For all other cases, non-emptiness has to be proved each time the condition is evaluated. A test similar to the SI test but with skolemization seems to be possible.

- The efficiency of the decorated rewrite relation of this chapter seems to depend mostly on the number of different set terms that are possible at the same time in a decoration. Regularity would give us a way to limit decorations to one set term. Unfortunately, the axiomatisation of regularity behaves worse than the one of sort inheritance: we may try to start with a similar set of clauses as for the SI test, except that we also need those clause combinations that do not validate *ncs*. But then, we need to prove that all ground instances of the common overlap of decoration rules belong to a common subsort of all the sorts added by the decoration rules. I.e. either we use a full inductive theorem prover for this or we try to show the existence of this common subsort not only for ground instances, but for all terms. The latter may be solved by skolemization of the common overlap, but there remain still open questions, like for the decorations of the Skolem constants.

- Are intersection sorts a solution to the last point ? We already mentioned in the introduction that G^n-logics allow for partial intersection. Provided that the known subset constraint solvers are able to cope with this extension, the remaining question is if there is an efficient encoding for parametric subsets.

- How far can decoration rules be used to perform completion-time program analysis, as this is usually done with static typing ?

- How can we further simplify or even eliminate the subset constraints ? We already saw in the discussion of Example 10.5.2 that monotonicity of all set constructors is one way to do this. The next question were then if it is possible to localise these constraints, just like decorations, in each term node. It seems that subset constraints concerning the decoration of the top node of a term get their lower bounds, i.e. the set terms at the left of $\subseteq^?$, from strict subterms only.

- Is there a way to combine parameterised sets and conditional membership formulas in the same specification ? The first one often needs the fact that all individual terms are strictly greater than set terms, the second needs to mix both kinds of operators in the precedence. Are ordering constraints restricting the arguments of a set operator a realistic solution ?

- How can ordering constraints be used to prove quasi-reductivity ? The published solvers seem to handle only satisfiability tests, but not entailment, which would be necessary for proving this property. The extensions discussed in [Avenhaus et Loría-Sáenz, 1994] seem to be a good starting point.

- How far may we integrate individual terms as arguments to set functions ? Constructors seem to be possible, but are the subset constraint solvers able to cope with such an extension ?

- How easy is the extension to so called *kinds*, i.e. types of types or sets of sets of individuals in our setting ? Kinds attached to sets by 'syntactic typing' seem easily realisable.

- How can we integrate higher-order functions best ? Kinds seem to be a key to distinguish between sets used as types and sets used as function graphs. This might be useful to keep the separation of the subset calculus, which would then be used only for sets representing types.

- To which point may we include finite sets with non-deterministic choices ? The main problem here is the type system that we used for G^n-logics and how to keep sort preservation for extensionality.

Chapitre 11

Conclusion

Dans cette thèse, nous avons présenté nos contributions au domaine des techniques de développement formel de logiciels. Nous avons commencé par la définition de deux logiques de clauses de Horn avec relation d'appartenance et sémantique ensembliste, pour lesquelles nous avons prouvé l'existence d'un modèle initial. Ces deux théories sont restées presque inexploitées. Il semble que l'absence de l'abstraction de variables, qui représente la différence majeure avec des lambda-calculs, admet de restreindre l'espace de fonction axiomatisé à une limite raisonnable. Nous avons démontré à notre avis, que ces deux logiques ont néanmoins une grande expressivité, qui suffit largement pour programmer d'une façon bien-structuré avec des sous-types paramétrisés et des fonctions partielles. Nous sommes optimistes que cette restriction nous fait gagner dans le domaine du raisonnement automatique à cause d'un espace de recherche restreint. Notre travail futur va être basé surtout sur ces deux théories, où bien des extensions mineures.

Puis, nous avons étudié des prouveurs de théorèmes réfutationnellement complets, utilisant des techniques de réécriture, pour ces deux théories. Pour ce but, nous avons étendu les techniques existants pour des fonctions totales à l'utilisation avec des fonctions partielles. Les prouveurs résultants sont réfutationnellement complets et offrent une technique simple mais puissant pour prouver des théorèmes équationnels, d'appartenance et existentiels dans ces théories. Nous avons finalement étendu ces deux prouveurs de théorèmes par des assertions, qui offrent un mécanisme de mémoire cachée locale aux clauses. L'idée est de stocker des conséquences de prémisse comme assertion, qui peut en conséquence être utilisée pour éliminer des prémisses superflues. Ce mécanisme est très général et pourrait aussi être utilisé pour d'autres techniques de simplification.

Dans la deuxième partie de cette thèse, nous avons étudié une structure de termes stockant l'information de typage dans chaque nœud d'un terme, ce qui nous avons appelés un terme décoré. Notre premier approche a considéré les décorations comme partie fixe de la syntaxe et a restreint filtrage et unification d'une façon correspondante. Ceci nous a permis de définir la réécriture décorée, où nous avons distingué entre les règles destinées au raisonnement équationnel, appelées des règles décorées, et les règles de décoration. Nous avons donné un lemme de paires critiques et une procédure de complétion à la Knuth-Bendix pour des systèmes de réécriture décorés. Cette procédure de complétion a prouvée qu'elle pouvait surmonter quelques difficultés présentes dans les approches précédentes, comme des règles incrementant la sorte minimale d'un terme réécrit, des fonctions partielles et des sortes sémantiques. Une comparaison avec d'autres procédures de complétion proposées dans la littérature à montré que cette procédure se comporte bien dans la plupart des cas. Cependant, la présence de déclarations de terme la laisse diverger facilement, à cause de problèmes techniques avec l'unification.

Afin d'étendre l'expressivité intuitive pour ce langage formel, nous avons essayé de passer aux clause de Horn équationnelles avec une relation d'appartenance prédéfinie. En utilisant la structure de termes décorée, nous avons adapté la méthode de démonstration réfutationnelle pour des clauses de Horn équationnelles au cas des clauses décorées. La procédure résultante nous permet de construire des systèmes de réécriture qui calculent effectivement dans ces théories. En plus, nous savons maintenant manier des types paramétrés et des règles de décoration conditionnelles.

Il faut remarquer que nous avons ignoré les problèmes temporels, de communication, de combinaison et de parallélisme, car la plupart des logiciels existants en ce moment fait surtout la gestion de données ou du calcul arithmétique. Le pourcentage de code source qui traite des aspects temporels, de communication ou de parallélisme est relativement bas par rapport aux parties du logiciel qui devrait vraiment calculer quelque chose – les aspects mentionnés ci-dessus peuvent être vus comme barrières ultimes d'approches non-formelles et ad-hoc à la programmation et sont donc très interéssant pour des études supplémentaires. La combinaison de logiciels est essentielle pour la réutilisation de logiciels et de spécifications. Elle peut-être située au-dessus du travail présenté ici. Nous pensons que les points mentionnés plus les aspects d'ordre supérieur dépassent tout simplement le cadre de cette thèse, mais nous espérons continuer à travailler sur de telles extensions.

Appendix A

Proof Transformations

1 Transformation Rules for Completeness of $\mathcal{OSC}$ Completion

For the definition of the decorated terms t, t', t'', t''' and rules ϕ, ϕ', ϕ'', see Proposition 7.2.3.

Orient_(N)SD :
$$(t \overset{\nu,\sigma,p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t' \Longrightarrow (t \overset{\nu,\sigma,p:S\to q:S\cup S'}{\underset{R\#}{\rightarrowtail}} t'' \overset{0,1}{\underset{}{\leftrightarrows}}{}^{\nu,\sigma,\phi}_{D\#} t')$$

Delete :
$$(t \overset{\omega,\sigma,p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t) \Longrightarrow \Phi$$

Deduce :
$$(t' \overset{\omega,\sigma,\phi}{\underset{R\#\cup D\#}{\longleftarrow}} t \overset{\nu,\sigma',\phi'}{\underset{R\#}{\rightarrowtail}} t'') \Longrightarrow (t' \overset{\nu,\tau,p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t'')$$

Deduce_deco :
$$(t' \overset{\omega,\sigma',\phi'}{\underset{D\#\cup R\#}{\longleftarrow}} t \overset{\nu,\sigma,\phi}{\underset{D\#}{\rightarrowtail}} t'') \Longrightarrow (t' \overset{\nu,\tau,(p:s\to p:s\cup S \text{ if } S\not\subseteq s)}{\underset{D\#}{\rightarrowtail}} t'_0 \overset{0,1}{\underset{}{\leftrightarrows}}{}^{\omega,\sigma',\phi'}_{D\#\cup R\#} t'')$$

Simplify_R :
$$(t \overset{\nu,\sigma',p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t') \Longrightarrow (t \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{R\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',p'':S''=q:S'}{\underset{E\#}{\longleftrightarrow}} t')$$

Simplify_D :
$$(t \overset{\nu,\sigma',p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t') \Longrightarrow (t \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{D\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',p'':S''=q:S'}{\underset{E\#}{\longleftrightarrow}} t')$$

Compose_R :
$$(t \overset{\nu,\sigma',l:S_l\to r:S_r}{\underset{R\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu,\sigma',l:S_l\to r':S_{r'}}{\underset{R\#}{\rightarrowtail}} t'' \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{R\#}{\leftarrowtail}} t')$$

Compose_D :
$$(t \overset{\nu,\sigma',l:S_l\to r:S_r}{\underset{R\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu,\sigma',l:S_l\to r':S_{r'}}{\underset{R\#}{\rightarrowtail}} t'' \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{D\#}{\leftarrowtail}} t')$$

Compose_D_deco ($\omega \neq \Lambda$) :
$$(t \overset{\nu,\sigma',\phi'}{\underset{D\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{D\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',\phi''}{\underset{D\#}{\rightarrowtail}} t''' \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{D\#}{\leftarrowtail}} t')$$

Compose_D_deco ($\omega = \Lambda$) :
$$(t \overset{\nu,\sigma',\phi'}{\underset{D\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu,\sigma',\phi''}{\underset{D\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',\phi}{\underset{D\#}{\leftarrowtail}} t')$$

Compose_R_deco :
$$(t \overset{\nu,\sigma',\phi'}{\underset{D\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{R\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',\phi''}{\underset{D\#}{\rightarrowtail}} t''' \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{R\#}{\leftarrowtail}} t')$$

Subsume_deco :
$$(t \overset{\nu,\sigma',\phi'}{\underset{D\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu,\sigma',\phi}{\underset{D\#}{\rightarrowtail}} t'' \overset{0,1}{\underset{}{\leftrightarrows}}{}^{\nu,\sigma',\phi}_{D\#} t')$$

Collapse_R :
$$(t \overset{\nu,\sigma',l:S_l\to r:S_r}{\underset{R\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{R\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',l':S_{l'}=r:S_r}{\underset{E\#}{\longleftrightarrow}} t')$$

Collapse_D :
$$(t \overset{\nu,\sigma',l:S_l\to r:S_r}{\underset{R\#}{\rightarrowtail}} t') \Longrightarrow (t \overset{\nu.\omega,\sigma'\circ\sigma,\phi}{\underset{D\#}{\rightarrowtail}} t'' \overset{\nu,\sigma',l':S_{l'}=r:S_r}{\underset{E\#}{\longleftrightarrow}} t')$$

2 Transformation Rules for MSS Rewriting

For the definition of the decorated terms t, t', t'', t''' and rules ϕ, ϕ', ϕ'', see Section 8.2.4.

Orient_(N)SD :
$$(t \overset{\sigma,p:S=q:S',O}{\underset{E\#}{\longleftrightarrow}} t') \Rrightarrow (t \overset{\sigma',p:S\to q:S\cup S',O}{\underset{R\#}{\rightarrowtail}} t'' \overset{0,1\sigma,\phi,O}{\underset{D\#}{\hookleftarrow}} t')$$

Delete :
$$(t \overset{p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t) \Rrightarrow \Phi$$

Deduce_MSS :
$$(t' \overset{\sigma',\phi',O'}{\underset{R\#\cup D\#}{\longleftarrow}} t \overset{\sigma'',\phi'',O''}{\underset{R\#}{\rightarrowtail}} t'') \Rrightarrow (t' \overset{0,1\sigma,\phi,\overline{O'}}{\underset{R\#}{\hookleftarrow}} \overline{t'} \overset{\tau,p:S=q:S'}{\underset{E\#}{\longleftrightarrow}} t''' \overset{\sigma,\phi}{\underset{R\#\cup D\#}{\longleftarrow}} \overline{t''} \overset{0,1\sigma'',\phi'',\overline{O''}}{\underset{R\#}{\hookleftarrow}} t'')$$

Simplify_MSS_D :
$$(t \overset{\sigma',p:S=q:S',O'}{\underset{E\#}{\longleftrightarrow}} t') \Rrightarrow (t \overset{\sigma'\circ\sigma,\phi,O'.O}{\underset{D\#}{\rightarrowtail}} t'' \overset{\sigma'\circ\sigma,p'':S''=q:S',O'}{\underset{E\#}{\longleftrightarrow}} t')$$

Compose_MSS_D :
$$(t \overset{\sigma',l:S_l\to r:S_r,O'}{\underset{R\#}{\rightarrowtail}} t') \Rrightarrow (t \overset{\sigma'\circ\sigma,l:S_l\to r':S_{r'},O'}{\underset{R\#}{\rightarrowtail}} t'' \overset{\sigma'\circ\sigma,\phi,O'.O}{\underset{D\#}{\longleftarrow}} t')$$

Collapse_MSS_D :
$$(t \overset{\sigma',l:S_l\to r:S_r,O'}{\underset{R\#}{\rightarrowtail}} t') \Rrightarrow (t \overset{\sigma'\circ\sigma,\phi,O'.O}{\underset{D\#}{\rightarrowtail}} t'' \overset{\sigma'\circ\sigma,l'':S_{l'}=r:S_r,O'}{\underset{E\#}{\longleftrightarrow}} t')$$

Deduce_deco_MSS :
$$(t' \overset{\sigma',\phi',O'\cup O''.O}{\underset{D\#\cup R\#}{\longleftarrow}} t \overset{\sigma,\phi,O''}{\underset{D\#}{\rightarrowtail}} t'') \Rrightarrow (t' \overset{\tau,(p:s\to p:s\cup S \text{ if } S\subsetneq s),O''}{\underset{D\#}{\rightarrowtail}} t'_0 \overset{\sigma',\phi',O'}{\underset{D\#\cup R\#}{\longleftarrow}} t'')$$

Subsume_deco_MSS :
$$(t \overset{\sigma',\phi',O}{\underset{D\#}{\rightarrowtail}} t') \Rrightarrow (t \overset{\sigma'\circ\sigma,\phi,O}{\underset{D\#}{\rightarrowtail}} t')$$

Peak w/o ovlp :
$$(t'^{:S'} \overset{\phi''}{\underset{D\#\cup R\#}{\longleftarrow}} t^{:S} \overset{\phi'}{\underset{D\#\cup R\#}{\rightarrowtail}} t''^{:S''}) \Rrightarrow (t'^{:S'} \overset{0,1\phi'}{\underset{R\#}{\hookleftarrow}} \overline{t'^{:S'}} \overset{\phi''}{\underset{D\#\cup R\#}{\rightarrowtail}} t_0{}^{:S_0} \overset{\phi'}{\underset{D\#\cup R\#}{\longleftarrow}} \overline{t''^{:S''}} \overset{0,1\phi''}{\underset{R\#}{\hookleftarrow}} t''^{:S''})$$

Peak w/ var ovlp :
$$(t'^{:S'} \overset{\phi'}{\underset{R\#\cup D\#}{\longleftarrow}} t^{:S} \overset{\phi''}{\underset{R\#\cup D\#}{\rightarrowtail}} t''^{:S''}) \Rrightarrow (t'^{:S'} \overset{0,1\phi'}{\underset{R\#}{\hookleftarrow}} \overline{t'^{:S'}} \overset{\phi''}{\underset{D\#\cup R\#}{\rightarrowtail}} t_0{}^{:S_0} \overset{\phi'}{\underset{D\#\cup R\#}{\longleftarrow}} \overline{t''^{:S''}} \overset{0,1\phi''}{\underset{R\#}{\hookleftarrow}} t''^{:S''})$$

Appendix B

Testing SI for Flat Term Declarations

In this section, we prove Theorem 8.2.13. Another property is needed in order to get unique reduced forms.

Lemma B.0.1 *Let $\phi \in R$ be terminating and $t^{:S} \rightrightarrows_R^{\phi,\sigma,k} t'^{:S'}$. Then*

$$t^{:S} \overset{*}{\rightrightarrows}_R^{\phi,\sigma} t''^{:S''} \overset{*}{\leftleftarrows}_R^{\phi,\sigma} t'^{:S'},$$

where the two rewrite sequences are normalising for all layers $k'' \leq k$.

Proof: Remark first of all that normalising sequences with $\rightrightarrows_R^{\phi,\sigma}$ are always bottom-up, since the substitution does not change. This guarantees furthermore, that the result is unique, although R as a whole is not necessarily confluent. The result follows immediately from the fact that a layer rewrite step at layer k only introduces redexes for the same pair (ϕ, σ) that are situated on a layer l with $l < k$ in the reduced term. This is illustrated in Figure 8.2, where we assume ϕ to have the shape $l^{:S_l} \to r^{:S_r}$. Consequently, the subterms at the ascending layers k' are identical in the two rewrite sequences, as long as $k' < k$. At k, the left rewrite sequence reduces the redexes that were also reduced in the step $t^{:S} \rightrightarrows_R^{\phi,\sigma,k} t'^{:S'}$ plus those generated by the steps in deeper layers l, i.e. $l > k$, whereas the right rewrite sequence reduces only the latter. Remark that there are no additional redexes for (ϕ, σ) resulting from $t^{:S} \rightrightarrows_R^{\phi,\sigma,k} t'^{:S'}$, since these have to be on a layer l with $l < k$. Therefore, the left and the right rewrite sequence are identical for all layers $k' < k$ up to k''. □

In the following definition, we need the notion of ascending narrowing (in R). This denotes the 'classical' narrowing steps along the occurrences of the narrowed term s.t. no step occurs after another one at lower or equal position. Furthermore, we need to combine two unifiers σ_1, σ_2. This is done by retransforming each unifier $\sigma_i = \{x_{ij}{}^{:S_{ij}} \mapsto t_{ij}{}^{:S'_{ij}}\}_{j \in J_i}$ for $i \in [1,2]$ into a unification problem $(\sigma_i)_= = \bigwedge_{j \in J_i}(x_{ij}{}^{:S_{ij}} \cong_d^? t_{ij}{}^{:S'_{ij}})$. Then, the result of the combination is the $\mathcal{D}$-solution ψ for the problem $(\sigma_1)_= \wedge (\sigma_2)_=$, if it exists. This is written $\psi = \sigma_1 + \sigma_2$. Otherwise, there is no combination.

Definition B.0.2 *Let $\mathcal{T} \subseteq ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $\phi : l^{:S_l} \to r^{:S_r}$ be a decorated rewrite rule and $t^{:S} \in \mathcal{T}$, s.t. $Var(\phi) \cap Var(t^{:S}) = \emptyset$. Then, $t^{:S}$ ϕ-*narrows *in $\mathcal{T}$ to $t'^{:S'}$ at occurrence ω, if:*

1. $\omega \in \mathcal{NVO}cc(t^{:S})$,

2. *there is a decorated unifier* σ *in a sound and* $\mathcal{T}$*-complete set of unifiers of* $(t^{:S}|_\omega \cong_d^? l^{:S_l})$) *and*

3. $\sigma(t^{:S}) \rightarrowtail_R^{\phi,\sigma,\omega} t'^{:S'}$.

This ϕ*-narrowing step* yields *the substitution* σ*. If this* ϕ*-step is followed by a sequence of* ϕ*-steps yielding* σ'*, where* ϕ *always has the same variables, then the whole sequence yields the combination* $\sigma+\sigma'$*, if it exists. Otherwise, the sequence is by definition not correct. A sequence of narrowing steps is* strictly ascending*, if it is the empty sequence or a strictly ascending sequence followed by a step at* ω*, s.t. all preceding steps take place at incomparable or strictly greater occurrences than* ω*.*

We get the usual results [Hullot, 1980] (without proof here) that narrowing sequences starting with $t^{:S}$, ending with $t'^{:S'}$ and yielding σ correspond one-to-one with decorated rewrite sequences using $\rightarrowtail_R^\phi$ from $\sigma(t^{:S})$ to $t'^{:S'}$. In the other direction, given a normalising rewrite sequence using $\rightarrowtail_R^{\sigma,\phi}$, we get a corresponding narrowing sequence by the confluence and termination of the decorated term rewrite rule set $R = \{\sigma(\phi)\}$, where $\sigma(\phi)$ is the instantiation of ϕ by σ, s.t. the variables in $\sigma(\phi)$ may not be instantiated. Hence, $\sigma(\phi)$ behaves like a ground rewrite rule.

Remark that any strictly ascending sequence of narrowing steps has to terminate, since eventually $\wedge$ is reached. These narrowing steps are necessary in order to adjust substitution images in the forthcoming Critical Pair Lemma B.0.6, since decorated rewrite rules are not necessarily flat or linear. Therefore, reducing all redexes for some $\phi \in R$ in a term t in a layer below a redex for some $\phi' \in R$ might change the image of a variable x in the domain of σ, the matcher needed to apply ϕ'. Hence, there may be no σ' that could be used to apply ϕ' on the changed term t'. So we need to equalise the subterms that were equal in t but are no more equal in t', in order to find such a σ'. But these may be situated on different layers. In order to get a unique reduced form, we use a bottom-up strategy for the equalisation together with Lemma B.0.1.

In some cases, it will be necessary to mount up to the left hand side l of ϕ', since some non-variable parts of l may be situated on deeper layers as the variable x. Consequently, critical pairs have to take care of such bottom-up reductions, s.t. ascending reductions correspond with ascending narrowing steps.

Definition B.0.3 Layer critical pairs (obtained by layered superposition *into* decorated rules)
Let $\mathcal{T} \subseteq ValidT_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$, $\phi : g^{:S_g} \rightarrow d^{:S_d}$ *and* $\phi' : l^{:S_l} \rightarrow r^{:S_r}$ *be rewrite rules in* R *with disjoint sets of variables.*

The two rules overlap if there exists an occurrence ω*. s.t.* $g^{:S_g}$ ϕ'*-narrows at* ω*, A* $\mathcal{T}$-layer critical pair *(for short LCP) is a decorated equation* $p^{:S_p} = q^{:S_q}$*, s.t.* $p^{:S_p}$ *is obtained by strictly ascending* ϕ'*-narrowing in* $\mathcal{T}$ *from* $g^{:S_g}$ *yielding* σ_1*,* $q^{:S_q}$ *is obtained in the same way from* $d^{:S_d}$ *yielding* σ_2*,* $\sigma = \sigma_1 + \sigma_2$ *exists and both* $p^{:S_p}$ *and* $q^{:S_q}$ *are* $\rightarrowtail_R^{\phi,\sigma}$*-normalised, which does not mean that we use normal narrowing here - it has to be understood as an additional condition preventing us from skipping necessary steps.*

Let $LCP(R,R)|_\mathcal{T}$ denote the set of such layer critical pairs. Clearly, these sets of critical pairs are finite, since strictly ascending ϕ'-narrowing terminates. Remark that for every $p^{:S_p}$ and $q^{:S_q}$ in an LCP, we know that it can be reached from $\sigma(g^{:S_g})$ or $\sigma(d^{:S_d})$, respectively, by layer rewriting, for σ being the substitution yield by the LCP-calculation, since rewriting with $\rightarrowtail_R^{\phi,\sigma}$ is confluent and $\rightrightarrows_R^{\phi,\sigma}$ can be seen as a special normalisation strategy for $\rightarrowtail_R^{\phi,\sigma}$.

Example B.0.4 *Let the decorated rewrite rules ϕ and ϕ' be defined as follows:*

$$(\phi : g(f(x^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, f(f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, y'^{:\langle A\rangle})^{:\langle A\rangle})^{:\{B\}} \to g(x^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\langle B\rangle})$$
$$(\phi' : f(z^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle} \to z^{:\langle A\rangle})$$

For simplicity, we calculate with $\mathbf{UNIF_d}$ *in the following. Clearly, ϕ' overlaps into ϕ at occurrences 1, 2 and 2.1 with unifiers:*

$$\begin{aligned}\sigma_1 &= \{z^{:\langle A\rangle} \mapsto x^{:\langle A\rangle}\},\\ \sigma_2 &= \{z^{:\langle A\rangle} \mapsto f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, y'^{:\langle A\rangle} \mapsto a^{:\langle A\rangle}\},\\ \sigma_{2.1} &= \{z^{:\langle A\rangle} \mapsto y^{:\langle A\rangle}\}.\end{aligned}$$

Then the LCPs are:

- *overlap at 1 gives σ_1 and:*
  $g(x^{:\langle A\rangle}, f(f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, y'^{:\langle A\rangle})^{:\langle A\rangle})^{:\{B\}} = g(x^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\langle B\rangle})$, *(LCP 1)*
  Narrowing at 2 in (LCP 1) with ϕ' gives $\sigma_1 \cup \{x^{:\langle A\rangle} \mapsto f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}\}$ and:
  $g(f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle})^{:\{B\}} = g(f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\langle B\rangle}$, *(LCP 2)*
  Narrowing at 2.1 in (LCP 1) gives $\sigma_1 \cup \{y^{:\{A\}} \mapsto x^{:\langle A\rangle}\}$ and:
  $g(x^{:\langle A\rangle}, f(x^{:\langle A\rangle}, y'^{:\langle A\rangle})^{:\langle A\rangle})^{:\{B\}} = g(x^{:\langle A\rangle}, x^{:\langle A\rangle})^{:\langle B\rangle})$, *(LCP 3)*
  Now, we can narrow at 2 in (LCP 3) giving $\sigma_1 \cup \{y^{:\{A\}} \mapsto x^{:\langle A\rangle}, y'^{:\{A\}} \mapsto a^{:\langle A\rangle}\}$ and:
  $g(x^{:\langle A\rangle}, x^{:\langle A\rangle})^{:\{B\}} = g(x^{:\langle A\rangle}, x^{:\langle A\rangle})^{:\langle B\rangle}$, *(LCP 4)*
- *overlap at 2 gives σ_2 and:*
  $g(f(x^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, f(y^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle})^{:\{B\}} = g(x^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\langle B\rangle}$ *(LCP 5)*
- *overlap at 2.1 gives $\sigma_{2.1}$ and:*
  $g(f(x^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, f(y^{:\langle A\rangle}, y'^{:\langle A\rangle})^{:\langle A\rangle})^{:\{B\}} = g(x^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\langle B\rangle}$, *(LCP 6)*
  Now, we can narrow at 2 in (LCP 6), giving $\sigma_{2.1} \cup \{y'^{:\{A\}} \mapsto a^{:\langle A\rangle}\}$ and:
  $g(f(x^{:\langle A\rangle}, a^{:\langle A\rangle})^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\{B\}} = g(x^{:\langle A\rangle}, y^{:\langle A\rangle})^{:\langle B\rangle}$. *(LCP 7)*

Peak Reduction

In this section we prove results allowing us to reduce variable, critical and disjoint peaks.

Lemma B.0.5 *Let $t'^{:S'} \leftrightarrows_R^{\phi',\sigma',k'} t^{:S} \rightrightarrows_R^{\phi'',\sigma'',k''} t''^{:S''}$, s.t. $O_{(t^{:S},\phi',\sigma',k')} \asymp O_{(t^{:S},\phi'',\sigma'',k'')}$. Then $t'^{:S'} \overset{*}{\leftrightarrows}_R \circ \overset{*}{\rightrightarrows}_R t''^{:S''}$.*

Proof: Let $\phi' : l^{:S_l} \to r^{:S_r}$, $\phi'' : g^{:S_g} \to d^{:S_d}$ and $t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(\phi',\sigma',k')}}[\sigma''(g^{:S_g})]_{O_{(\phi'',\sigma'',k'')}}$. W.l.o.g. we can assume $k' \leq k''$.

There may be redexes for ϕ', σ' that are created by reductions with ϕ'', σ''. Therefore, we have to assume $\sigma'(l^{:S_l}) \cong_d \sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}$ with $|\omega''| = k''$ for all $\omega'' \in O''$, O'' is maximal and $t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_{O''}]_{O'}[\sigma''(g^{:S_g})]_{\overline{O''}}$, s.t. $O'' \uplus \overline{O''} = O_{(t^{:S},\phi'',\sigma'',k'')}$.
Consequently, $t'^{:S'} \cong_d t^{:S}[\sigma'(r^{:S_r})]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_{O''}]_{O'}[\sigma''(g^{:S_g})]_{\overline{O''}}$ and $t''^{:S''} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}}$.

Furthermore, the replacement constructed by ϕ' may generate a redex for ϕ'', σ'', i.e.:

$$\begin{aligned}t'^{:S'} &\cong_d t^{:S}[\sigma'(r^{:S_r})]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_{O''}]_{O'}[\sigma''(g^{:S_g})]_{\overline{O''}}\\ &\cong_d t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_{O''}]_{O'}[\sigma''(g^{:S_g})]_{\overline{O''}}\end{aligned}$$

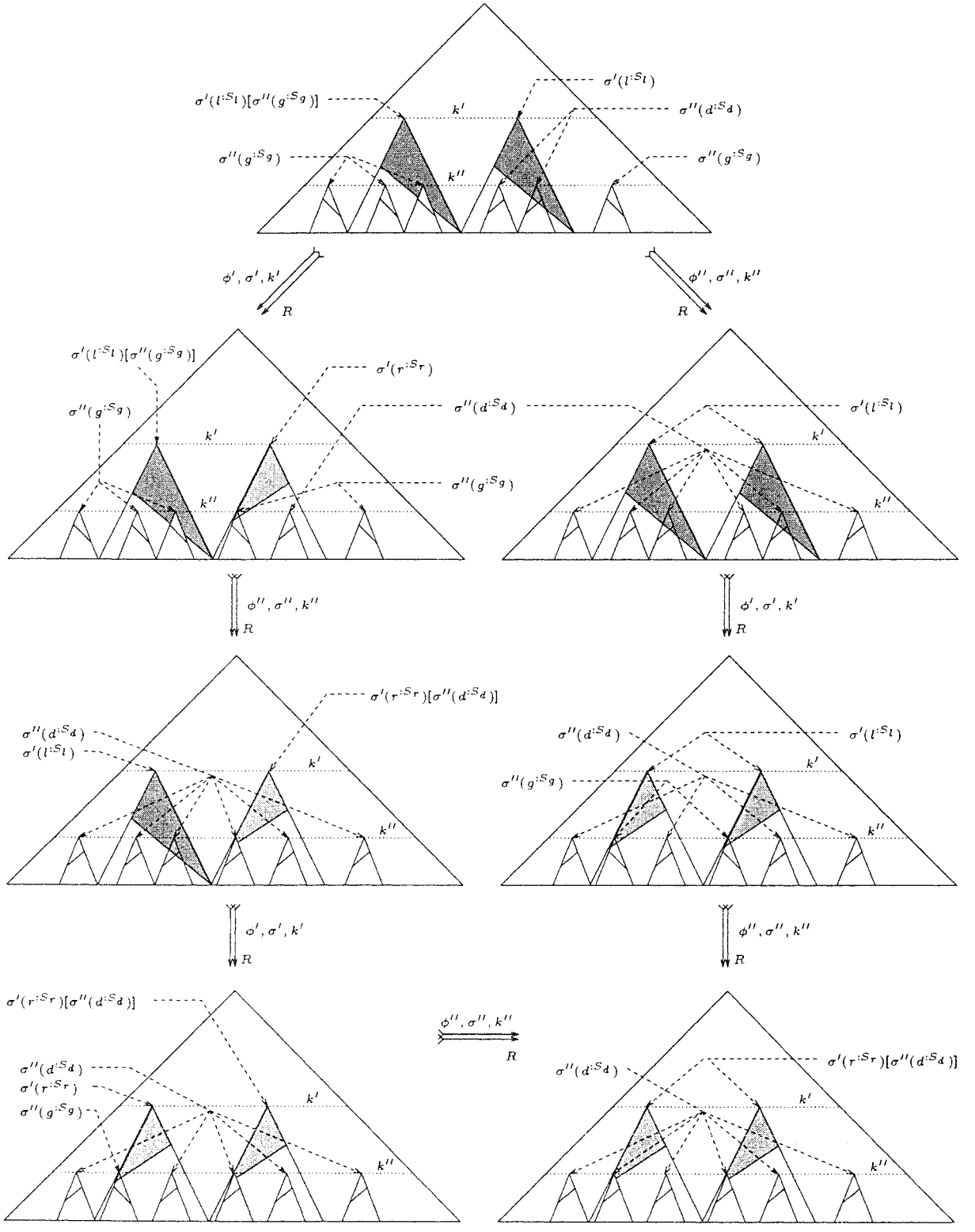


Figure B.1: No Overlap Case

Now, we are sure that $O \uplus O'' \uplus \overline{O''} = O_{(t^{:S'},\phi'',\sigma'',k'')}$ and $O_{(t^{:S},\phi',\sigma',k')} \uplus O' = O_{(t''^{:S''},\phi',\sigma',k')}$. The following peak reduction is illustrated in Figure B.1:

$$
\begin{array}{rl}
t'^{:S'} \cong_d & t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_{O''}]_{O'}[\sigma''(g^{:S_g})]_{\overline{O''}} \\
\rightrightarrows_R^{\phi'',\sigma'',k''} & t^{:S}[\sigma'(r^{:S_r})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}} \\
=_d & t^{:S}[\sigma'(r^{:S_r})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}} \\
\overset{0,1}{\rightrightarrows}_R^{\phi',\sigma',k'} & t^{:S}[\sigma'(r^{:S_r})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}} \\
\overset{0,1}{\rightrightarrows}_R^{\phi'',\sigma'',k''} & t^{:S}[\sigma'(r^{:S_r})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(r^{:S_r})[\sigma''(d^{:S_d})]_O]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}} \\
\overset{0,1}{\rightrightarrows}_R^{\phi'',\sigma'',k''} & t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}} \\
\rightrightarrows_R^{\phi',\sigma',k'} & t^{:S}[\sigma'(l^{:S_l})]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O'}[\sigma''(d^{:S_d})]_{\overline{O''}}. \\
\cong_d & t''^{:S''}
\end{array}
$$

The (optional) 0, 1-steps are due to the cases where either no new redex for the radical ϕ', σ' is generated using ϕ'', σ'' or $k' = k''$, i.e. possibly $\sigma''(d^{:S_d}) \cong_d \sigma'(l^{:S_l})$. □

Lemma B.0.6 Layer Critical Pair Lemma
Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ be downward closed w.r.t. $\rightrightarrows_R$ and $t^{:S}, t'^{:S'}, t''^{:S''} \in \mathcal{T}$.
Let furthermore $\phi' : l^{:S_l} \to r^{:S_r}$, $\phi'' : g^{:S_g} \to d^{:S_d}$ and $t'^{:S'} \leftrightarrows_R^{\phi',\sigma',k'} t^{:S} \rightrightarrows_R^{\phi'',\sigma'',k''} t''^{:S''}$, s.t. $k' \leq k''$, $O_{(t^{:S},\phi',\sigma',k')}.\mathcal{NVO}cc(l^{:S_l}) \cap O_{(t^{:S},\phi'',\sigma'',k'')} \neq \emptyset$.
Then $t'^{:S'} \overset{*}{\rightrightarrows}_R \circ \longleftarrow\!\!\!\shortrightarrow_{LCP(\phi',\phi'')_{|\mathcal{T}}} \circ \overset{*}{\leftrightarrows}_R t''^{:S''}$.

Proof: Let w.l.o.g. $t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(\phi',\sigma',k')}}[\sigma''(g^{:S_g})]_{O_{(\phi'',\sigma'',k'')}}$,
$t'^{:S'} \cong_d t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma''(g^{:S_g})]_{O'}$,
s.t. $O_{(t^{:S},\phi',\sigma',k')}.O \uplus O' = \bigcup_{i \geq k'} O_{(t'^{:S'},\phi'',\sigma'',i)}$ and
$t''^{:S''} \cong_d t^{:S}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma''(d^{:S_d})]_{O'''}[\sigma''(g^{:S_g})]_{O' \setminus O'''}$,
where $O_{(t^{:S},\phi',\sigma',k')}.O'' \uplus O''' = O_{(t^{:S},\phi'',\sigma'',k'')}$. Hence, O' stands for the redexes of ϕ'' with σ'' 'outside of' $\sigma'(l^{:S_l})$.

Then we can bottom-up reduce up to k':

$$t'^{:S'} \overset{*}{\rightrightarrows}_R^{\phi'',\sigma''} t^{:S}[\sigma'(r^{:S_r})[\sigma''(d^{:S_d})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma''(d^{:S_d})]_{O'}$$

The same is possible for $t''^{:S''}$:

$$
\begin{array}{rll}
t''^{:S''} & \overset{*}{\rightrightarrows}_R^{\phi'',\sigma''} & t^{:S}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma''(d^{:S_d})]_{O'''}[\sigma''(d^{:S_d})]_{O' \setminus O'''} \\
& =_d & t^{:S}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma''(d^{:S_d})]_{O'}
\end{array}
$$

Since we may have variable overlaps with $l^{:S_l}$, we need to continue with the bottom-up reduction until k'. Remark that in both branches, we have $\sigma''(d^{:S_d})$ at the occurences O', i.e. we can omit these details in the following. Moreover, the terms $\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}$ and $\sigma'(r^{:S_r})[d^{:S_d}]_O$ correspond by construction with a layer critical pair.

Therefore, there exists an LCP $p^{:S_p} = q^{:S_q}$ that can be applied to the result of the bottom-up reduction until k' of the left and the right reduction sequence.

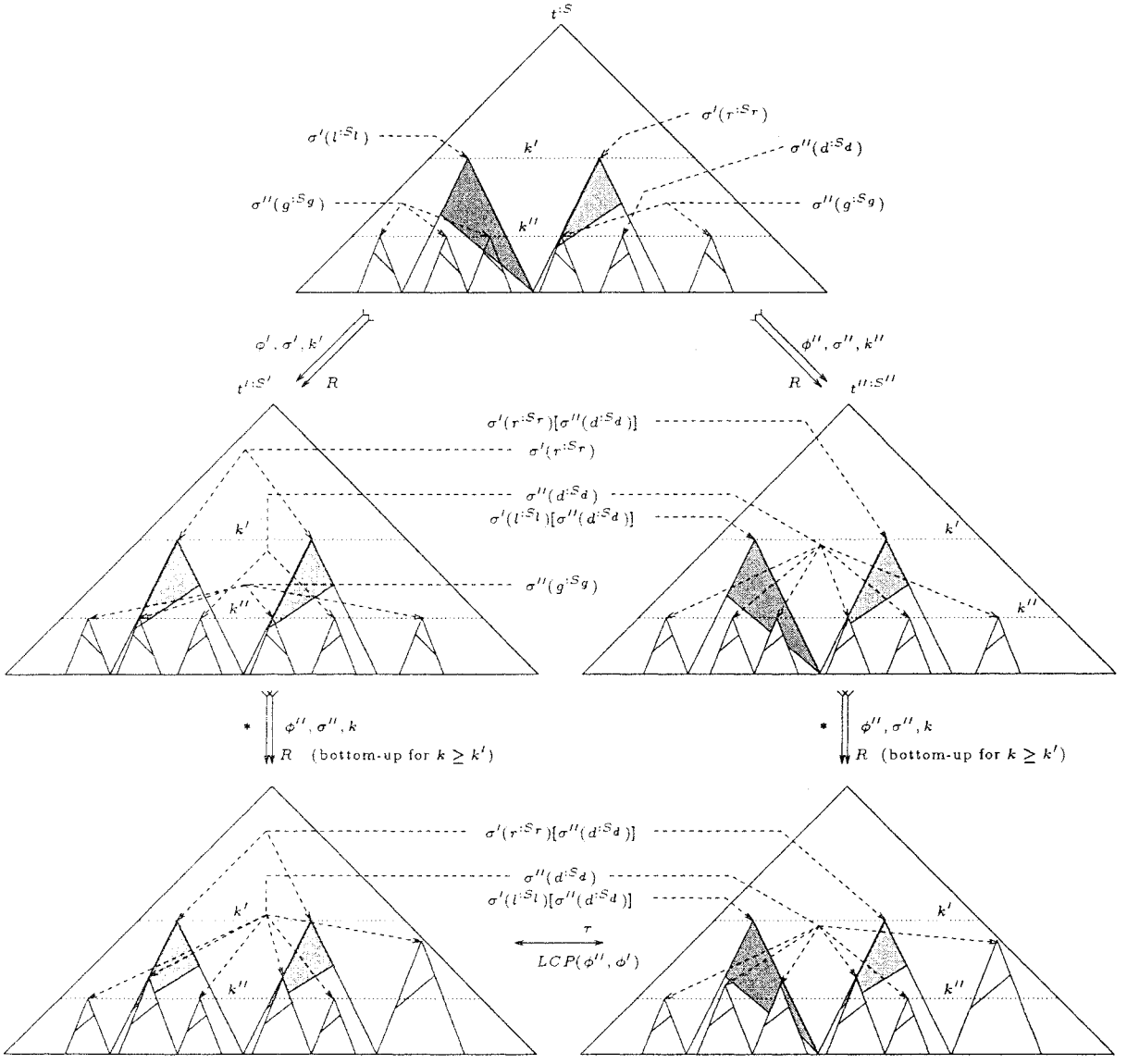


Figure B.2: Critical Overlap Case

The LCP is actually applicable, since all parts of the unifier ψ used to compute the overlap correspond with strict subterms[27] of the redex $\sigma''(g^{:S_g})$, since we only compute non-variable overlaps. Consequently, we do not need to overlap recursively into the substitution part of $\psi(l^{:S_l})$, which might lead to infinite sets of critical pairs.

□

Only now we are able to prove the variable overlap case, since we may need layer critical pairs in order to solve them.

Lemma B.0.7 *Let $\mathcal{T} \subseteq \mathcal{T}_d(\mathcal{S}_\Diamond, \mathcal{F}, \mathcal{X}_\Diamond)$ be downward closed w.r.t. $\rightrightarrows_R$ and $t^{:S}, t'^{:S'}, t''^{:S''} \in \mathcal{T}$. Let furthermore $\phi' : l^{:S_l} \to r^{:S_r}$, $\phi'' : g^{:S_g} \to d^{:S_d}$ and $t'^{:S'} \leftrightarrows_R^{\phi',\sigma',k'} t^{:S} \rightrightarrows_R^{\phi'',\sigma'',k''} t''^{:S''}$, where $k' \geq k''$, $O_{(t^{:S},\phi',\sigma',k')}.\mathcal{NVO}cc(l^{:S_l}) \cap O_{(t^{:S},\phi'',\sigma'',k'')} = \emptyset$ and there is an occurrence $\omega \in O_{(t^{:S},\phi',\sigma',k')}.\mathcal{VO}cc(l^{:S_l})$, s.t. there exists a ω' with $\omega.\omega' \in O_{(t^{:S},\phi'',\sigma'',k'')}$.*

Then either $t'^{:S'} \overset{*}{\rightrightarrows}_R \circ \overset{}{\hookleftarrow\!\!\!\rightarrow}_{LCP(\phi',\phi'')_{|_T}} \circ \overset{*}{\leftrightarrows}_R t''^{:S''}$ or $t'^{:S'} \overset{*}{\rightrightarrows}_R \circ \overset{*}{\leftrightarrows}_R t''^{:S''}$.

Proof: Let w.l.o.g. $t^{:S} \cong_d t^{:S}[\sigma'(l^{:S_l})]_{O_{(\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_Q]_{k'}[\sigma''(g^{:S_g})]_{O_{(\phi'',\sigma'',k'')}}$,
$t'^{:S'} \cong_d t^{:S}[\sigma'(r^{:S_r})[\sigma''(g^{:S_g})]_O]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(g^{:S_g})]_Q]_{k'}[\sigma''(g^{:S_g})]_{O'}$,
s.t. $O_{(t^{:S},\phi',\sigma',k')}.O \uplus O' = \bigcup_{i \geq k'} O_{(t'^{:S'},\phi'',\sigma'',i)}$ and $t''^{:S''} \cong_d$
$t^{:S}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_{O''}]_{O_{(t^{:S},\phi',\sigma',k')}}[\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_Q]_{k'}[\sigma''(d^{:S_d})]_{O'''}[\sigma''(g^{:S_g})]_{O' \setminus O'''}$,
where $O_{(t^{:S},\phi',\sigma',k')}.O'' \cup O''' = O_{(t''^{:S''},\phi'',\sigma'',k'')}$. Hence, O' stands for the redexes of ϕ'' using σ'' outside of $\sigma'(l^{:S_l})$.

First, we can bottom-up reduce the terms $t'^{:S'}$ and $t''^{:S''}$ up to k' with (ϕ'', σ''). Let $\overline{t'^{:S'}}$ and $\overline{t''^{:S''}}$ be the result of this reduction in the left resp. right sequence. If there is a reduction at an occurrence $\omega \in O_{(t^{:S},\phi',\sigma',k')}.\mathcal{NVO}cc(l^{:S_l})$ in the right reduction sequence, then there is also a corresponding layer critical pair, giving us the convergence like in Lemma B.0.6.

Otherwise, we are sure that there exists a substitution $\overline{\sigma'}$, such that $O_{(\overline{t''^{:S''}},\phi',\overline{\sigma'},k')}$ subsumes $O_{(t^{:S},\phi',\sigma',k')}$ and $\sigma'(l^{:S_l})[\sigma''(d^{:S_d})]_Q \cong_d \overline{\sigma'}(l^{:S_l})$

This case is illustrated in Figure B.3. Therefore,

$$\overline{t'^{:S'}} \rightrightarrows_R^{\phi',\overline{\sigma'},k'} t^{:S}[\overline{\sigma}(r^{:S_r})[\sigma''(d^{:S_d})]_{Q'}[\sigma''(g^{:S_g})]_{Q''}]_{O_{(\overline{t'^{:S'}},\phi',\overline{\sigma'},k')}} \leftrightarrows_R^{\phi',\overline{\sigma'},k'} \overline{t''^{:S''}},$$

because of $O_{(\overline{t'^{:S'}},\phi',\overline{\sigma'},k')} \uplus O_{(t^{:S},\phi',\sigma',k')} = O_{(\overline{t''^{:S''}},\phi',\overline{\sigma'},k')}$ and Lemma B.0.1. □

Orientation and Simplification

Lemma B.0.8 *Let $t^{:S} \overset{}{\hookleftarrow\!\!\!\rightarrow}_E^{p^{:S}=q^{:S'},\sigma,O} t'^{:S'}$, $(\phi : p^{:S} \to q^{:S \cup S'})$ be in R and $(\phi' : q^{:s} \to q^{:s \cup S \setminus S'}$ if $s \subsetneq S \setminus S') \in D$, s.t. $sort(q) \approx S \cup S'$ if $q \in \mathcal{X}_\Diamond$.*

Then $t^{:S} \overset{*}{\rightrightarrows}_R^{\phi} \circ \overset{*}{\leftrightarrows}_R^{\phi} \circ \overset{*}{\leftarrow}_D^{\phi'} t'^{:S'}$.

Proof: Let $K = \{k \mid \exists \omega \in O : |\omega| = k\}$ and $k_1, \ldots k_n$ be the elements of K sorted , s.t. $k_i > k_{i+1}$. Then $t^{:S} \rightrightarrows_R^{\phi,\sigma,k_1} \circ \ldots \circ \rightrightarrows_R^{\phi,\sigma,k_n} t''^{:S''} \leftrightarrows_R^{\phi,\sigma,k_n} \circ \ldots \circ \leftrightarrows_R^{\phi,\sigma,k_1} t'''^{:S'''} \underset{D}{0,1} \phi',\sigma,O$ $t'^{:S'}$,
where $t'^{:S'} \cong_d t^{:S}[\sigma(q^{:S'})]_O$, $t''^{:S''} \cong_d t^{:S}[\sigma(q^{:S \cup S'})]_{\overline{O}}$ and $t'''^{:S'''} \cong_d t^{:S}[\sigma(q^{:S'})^{:S \cup S'}]_O$ s.t. for

[27] Remark that the used substitution for ϕ'' is always σ'' !

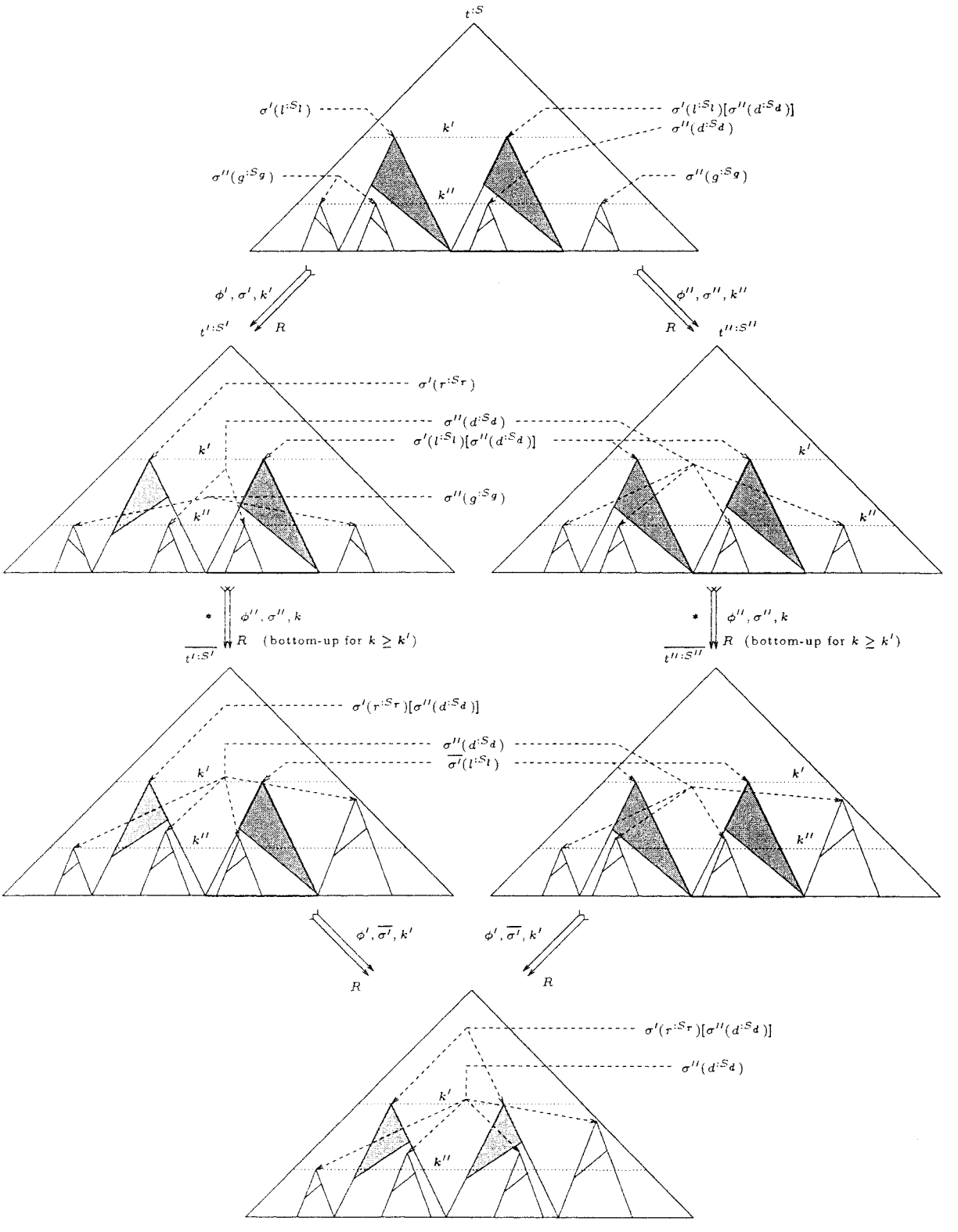


Figure B.3: Variable Overlap Case, Simple Part

all $\omega \in O_{(t:S,\phi,\sigma,k_i)}$, $i \in [1..n]$, there is a $\omega' \in \overline{O}$ with $\omega' \leq \omega$. Remark that this works since bottom-up layer rewriting always yields a unique term (see Lemma B.0.1).

□

The same technique of bottom-up layer rewriting is possible for the proof reduction of **Compose**. However, for **Simplify** and **Collapse**, we have to restrict the application to linear rules/equations, as the following example shows:

Example B.0.9 *Let* $\phi : a^{:\{A\}} \to b^{:\{A\}}$ *be a decorated rewrite rule and* ϕ' *be a decorated rewrite rule or equation of the shape* $f(x^{:\{A\}}, a^{:\{A\}}, g(x^{:\{A\}})^{:\{A\}})^{:\{A\}} \to f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}}$ *or* $f(x^{:\{A\}}, a^{:\{A\}}, g(x^{:\{A\}})^{:\{A\}})^{:\{A\}} = f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}}$, *respectively.*

Then the simplification of the term $f(x^{:\{A\}}, a^{:\{A\}}, g(x^{:\{A\}})^{:\{A\}})^{:\{A\}}$ *by* ϕ *as used in* **Collapse** *or* **Simplify**, *respectively, yields* $f(x^{:\{A\}}, b^{:\{A\}}, g(x^{:\{A\}})^{:\{A\}})^{:\{A\}}$. *However, using this simplification it is no more possible to prove the following equality with typable terms only:*

$$h(f(a^{:\{A\}}, a^{:\{A\}}, g(a^{:\{A\}})^{:\{A\}})^{:\{A\}}, f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}})^{:\{A\}}$$
$$=$$
$$h(f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}}, f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}})^{:\{A\}}$$

This is so, since the application of the simplified equation makes it necessary to reduce $f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}}$ *to* $f(f(b^{:\{A\}})^{:\{A\}})^{:\{A\}}$. *Then, using the simplified equation, we get the term:*

$$h(f(f(a^{:\{A\}})^{:\{A\}})^{:\{A\}}, f(f(b^{:\{A\}})^{:\{A\}})^{:\{A\}})^{:\{A\}},$$

which may be no more typable, because it does not stem from a typable term by layer rewriting (imagine a non-linear typing rule for h*).*

This might be avoided by the use of an adapted simplification rule.

Another possibility would be to use second-order context variables in order to cope with variable overlaps. However, this can be seen as an option, since we only want to prove sort inheritance and do not plan to use the resulting set of decorated and decoration rewrite rules as operational version of the G-algebra $\mathcal{P}$.

Simplification with decoration rewrite rules does not change w.r.t. standard rewriting, since their application strategy did not change.

Confluence

Let $\Rrightarrow$ be the proof reduction for layer rewriting and $\mathcal{SLC}$ be completion rules that can be easily obtained as special case of $\mathcal{OSC}$ by using layer rewriting for R instead of standard rewriting and inhibiting simplification by decorated rewrite rules. This is justified by Lemmas B.0.5, B.0.7, B.0.8 and B.0.6, together with the remarks of the last section concerning the simplification rules. Remark that the reduction rules dealing with decoration rewrite rules can be taken from $\mathcal{OSC}$ without any changes and that all newly introduced terms at step k must be reachable from old ones using $\stackrel{*}{\rightharpoonup}_{D_k} \cup \stackrel{*}{\rightrightarrows}_{R_k}$, as this is the case for the proof reductions in the last two sections. The proof of well-foundedness for $\Rrightarrow$, at least for the part without simplification rules, is straightforward using the complexity measure c of Lemma 7.3.1.

In order to achieve confluence for layer rewriting, we first have to prove the conservation of the typability property for all proofs stemming from one constructed by Theorem 6.5.1. This

is done in two steps. First, we prove the typability of all terms in decorated rewrite rules and equations. Then, we prove the typability of all terms in proofs stemming from typable proofs.

Lemma B.0.10 *Let* $(D_{\mathcal{P}}, E_{\mathcal{P}}, \emptyset) = (D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ *be a derivation using* $\mathcal{SLC}$. *Then for all* $k \geq 0$, *all terms that are typable in* D_k *are also typable in* D_∞, *if* D_* *only contains flat decoration rewrite rules, i.e.* **Compose_R_deco** *is not used during the derivation and* D_∞ *only contains flat decoration rewrite rules.*

Proof: The proof is an induction over $<_d \cup \lesssim_d$ analogous to Lemma 8.2.7. We show that any $\phi \in D_k$ is subsumed by some $\phi'' \in D_\infty$.

If all decoration rules are flat in D_*, the only way to make some $\phi \in D_k$ disappear is the subsumption rule **Subsume_deco**. Let $\phi' \in D_k$ subsume ϕ. Therefore, the induction hypothesis gives us a $\phi'' \in D_\infty$ subsuming ϕ', which also subsumes ϕ by transitivity of subsumption. □

Lemma B.0.11 *Let* $(D_{\mathcal{P}}, E_{\mathcal{P}}, \emptyset) = (D_0, E_0, R_0) \vdash_{\mathcal{OSC}} (D_1, E_1, R_1) \vdash_{\mathcal{OSC}} \ldots$ *be a derivation using* $\mathcal{SLC}$. *Then for all* $k \geq 0$, *all terms in* CT^k_{DER} *are typable in* D_∞, *if* D_* *(and therefore* D_∞*) only contains flat decoration rewrite rules.*

Proof: The proof is identical with the one of Lemma 8.2.8. Only the reference to Lemma 8.2.6 has to be replaced by one to Lemma 8.2.12. □

Now, the proof of Theorem 8.2.13 is once more identical to the one of Section 8.2.2, where the references to Lemmas 8.2.8, 8.2.6 and 8.2.7 have to replaced by those to Lemmas B.0.11, 8.2.12 and B.0.10, respectively.

Appendix C

Calculi, Transformations and Symbols

1 List of Calculi and Formula Transformations

1.1 Logic Calculi

name	description	section
GA	G-algebra deduction	2.1
OSL	$\mathcal{OSE}$-logic deduction	2.2
OSGL	G^n-logic deduction after EX elimination	2.2
RNL	R^n-logic deduction	2.3
GNL	G^n-logic deduction	2.4

1.2 Superposition Calculi

name	description	section
$\mathcal{NR}$	"classical" superposition (from [Nieuwenhuis et Rubio, 1992a])	4.1.3
$\mathcal{G}$	G^n-logic superposition	4.3.2
$\mathcal{RC}3$	cased R^n-logic superposition	4.3.3
$\mathcal{GC}3$	cased G^n-logic superposition	4.3.3
$\mathcal{OSC}$	G-algebra decorated completion	7.2
$\mathcal{GCD}$	G^1-logic decorated saturation	10.4

1.3 Formula Transformations

name	description	section
$.^+$	addition of set theoretic clauses	4.2.1,4.2.2
Ξ	EX elimination	4.2.3
Θ	quasi inverse of Ξ	4.2.3
$.^=$	replaces relations by propositional functions	4.2.4
$.^{rel}$	inverse of $.^=$	4.2.4
Δ	cased G^n-clauses to condeco clauses	10.2
Ψ	inverse of Δ	10.2
Π	condeco clause sets to cased G^n-clauses with sort inheritance axioms	10.2
Υ	condeco clause sets to G^n-clauses with sort inheritance axioms	10.2

2 List of Used Symbols

2.1 Model/Proof Theoretic Relations

name	description
$\models_{Log}$	satisfaction relation in logic Log
$\vdash_K$	derivability relation in calculus K

2.2 Orderings

name	description	section
$\geq^V$	substitution ordering	3.2.1
$\geq$	occurrence prefix ordering	5.2
$\geq_{lex}$	lexicographic occurrence ordering	5.2
$\lesssim_d$	decorated term subsumption modulo SI	5.2
$\underset{\sim}{\subset}$	set subsumption modulo SI	5.2
$\lesssim_d^V$	decorated substitution subsumption modulo SI	5.2
$\succ$	simplification ordering for saturation	4.1.2
$\Subset$	less decorated	6.4
$>_d$	reduction ordering for completion	6.4

2.3 Arrows

name	description	section
$\rightarrowtail$	rewrite relation	3.2.1
$\rightarrowtail_D$	decoration rewrite relation	6.1
$\rightarrowtail_R$	decorated rewrite relation	6.1
$\leftarrowtail\!\!\rightarrowtail_E$	decorated equality relation	6.1

2.4 Equation Symbols

name	description	section
$=_d$	syntactical equality (decorations included)	5.2
$=_{nd}$	syntactical equality without decorations	5.2
$\cong_d$	equality (on terms) modulo sort inheritance	5.2
$\approx$	equality (on sets) modulo sort inheritance	5.2

2.5 Matching/Unification Equations

name	description	section
$\leq^?$	matching equation	3.2.1
$\lesssim_d^?$	decorated matching equation	5.3
$\cong_d^?$	decorated unification equation	5.4

2.6 Constraints

name	description	chapter/section
$=^?$	equality constraint	4.1.1
$\subseteq^?$	subset constraint	10

Index

absolutely deterministic, 88
absolutely irreducible, 88
algebra
    G-, 45
    $\Sigma_{\Diamond}$-, 236
algorithm, 84
assertions of a decorated term, 255
atom
    $\mathcal{OSE}$-logic, 50

bottom-up decoration proof, 206

cased clause, 124
clause
    $\mathcal{OSE}$-logic, 50
    condeco, 256
closure
    sort inheritance , 146
complete, 84
completion rules
    first level, 226
conclusion of a clause, 102
confluent, 85
    locally on $\mathcal{T}$, 182
    on $\mathcal{T}$, 182
conform, 46
    G^n-logic, 72
constraints, 102
critical pair
    $\mathcal{T}$-layer, 304
    decorated, 184
    decoration, 184
critrical pair
    solved, 187
cycle, 143
    minimal set of, 143
cyclic, 143

D-closed, 204
decorated term
    G-algebra, 147
    G^1-logic, 255
decoration
    G-algebra, 147
    G^1-logic, 255
decoration formulas, 148
deterministic
    absolutely, 88
    strongly, 88
downward closed, 175

entailment relation, 102
equality
    decorated, 167
extra variables, 87

fairness
    condeco derivation, 269
    on $\mathcal{T}$, 200
    saturation, 104
    w.r.t. decoration rules, 205
formula
    saturation calculus, 102
free Σ-term interpretation
    $\mathcal{OSE}$-logic, 49

generates, 103
goal
    $\mathcal{OSE}$-logic, 50
ground instance
    cased clause, 124
    clause, 103
    inference, 104
ground term interpretation
    $\mathcal{OSE}$-logic, 49

homomorphism
    G^n-logic, 71
    $\mathcal{OSE}$-logic, 48

inference step, 102
    $\mathcal{GCD}$, 268

initial decorated presentation
condeco clauses, 258
initial presentation
condeco clauses, 258
instance
$\mathcal{OSE}$-logic, 51
interpretation
G^n-logic, 70
HCL logic, 103
$\mathcal{OSE}$-logic, 48
irreducible
absutely, 88
strongly, 88

less decorated, 175

matcher, 83
decorated
G^1-logic, 260
strict, 153
matching equation
decorated
G-algebra, 152
matching problem, 83
decorated
G-algebra, 152
G^1-logic, 260
variable disjoint, 83
model
G-algebra, 46
R^n-logic, 61
$\mathcal{OSE}$-logic, 51
multiset expression of an equation, 103
G^n-, 119

narrowing
strictly ascending, 304
narrows
with ϕ in $\mathcal{T}$, 303

occurrence ordering
lexicographic, 148
prefix, 148
occurrences, 83
non-variable , 148
variable, 148
one-step-equal, 167
oriented condition
operational, 87
partially, 87
oriented equation
unconditional, 87
orthogonal, 85

persistent, 104
premise of a clause, 102
presentation, 45
G^n-logic, 70
R^n-logic, 59
$\mathcal{OSE}$-logic, 50

quasi-reductive, 87

reduction ordering
decorated, 175
reduction relation, 85
redundant, 104, 268
condeco clause, 268
reflects, 199
regularity, 85
replacement, 83
rewrite relation, 84, 88
condeco clause, 276
decorated
G-algebra, 168
decoration, 168
layer, 213
maximally subterm sharing, 214
order-sorted, 85
rewrite rule
conditional, 86
conditional normal, 86
decorated, 168
decoration, 168
rewrite system
decoration, 168

satisfaction relation, 102
saturated, 104
sequence of $\mathcal{GCD}$-steps, 269
signature
G-algebra, 45
G^n-logic, 70
R^n-logic, 59
$\mathcal{OSE}$-logic, 47
solution
DU-, 264
strict decorated unification, 158

solved, 188
solved form
  decorated matching problem, 152
  decorated unification problem, 157
sort
  completing, 145
  syntactically non-empty, 144
sort inheritance , 141
  closure, 149
  on $\mathcal{T}$, 149
sort ordering
  semantic, 140
  syntactic
    G-algebra, 139
sort preserving
  $\mathcal{OSE}$-logic, 54
sort structure
  completed, 145
sort-decreasing, 85
sortal behavior
  $\mathcal{OSE}$-logic, 54
sound, 84
specification
  G-algebra, 45
strongly deterministic, 88
strongly irreducible, 88
structurally decorating, 281
subset relation, 252
subsort
  G-algebra, 139
    strict, 139
substitution, 83
  $\mathcal{OSE}$-logic, 51
  decorated, 150
    G^1-logic, 260
    equality, 151
    of $\mathcal{T}_X$ into $\mathcal{T}$, 150
    ordering, 151
  irreducible, 103
subsumed modulo sort inheritance , 150
subterm conservative, 165
  strictly, 165
syntactically decorating, 287

term interpretation
  $\mathcal{OSE}$-logic, 49
term projection, 148
term rewrite rule, 84
  deterministic, 87
  order-sorted, 85
term rewrite system
  decorated, 168
  deterministic, 87
  strongly deterministic, 88
  unconditional, 84
terms, 83
  G-algebra, 45
  $\mathcal{OSE}$-logic, 47
theorem proving derivation, 104
  cased clauses, 127
  condeco, 269
transformation rules, 84
typable, 203
typing conservative, 203
typing proof, 203

unification equation
  decorated
    G-algebra, 157
unification problem
  decorated
    G-algebra, 157
    G^1-logic, 261
unifier, 83
  decorated
    G^1-logic, 261
    strict, 158
  more general, 83
unsatisfiable, 264
upward closure, 140

valid
  G-algebra, 148
  $\mathcal{OSE}$-logic, 51
  cased clauses, 127
  decorated term
    G-algebra, 169
variable assignment
  G-algebra, 46
  G^n-logic, 71
  $\mathcal{OSE}$-logic, 48
variable disjoint
  matching problem, 152

Bibliography

[Abadi *et al.*, 1991] M. Abadi, L. Cardelli, B. Pierce, et G. Plotkin. Dynamic typing in a statically typed language. *ACM Transactions on Programming Languages and Systems*, 13(2):237–268, Avril 1991.

[Abadi *et al.*, 1994] M. Abadi, L. Cardelli, B. Pierce, et D. Rémy. Dynamic typing in polymorphic languages. Rapport Technique 120, DEC Systems Research Center (Palo Alto/CA, USA), Janvier 1994.

[Aida *et al.*, 1990] H. Aida, G. Goguen, et J. Meseguer. Compiling concurrent rewriting onto the rewrite rule machine. Dans *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems, Montreal (Canada)*, rédacteurs S. Kaplan et M. Okada, volume 516 de *Lecture Notes in Computer Science*, pages 320–332. Springer-Verlag, Juin 1990.

[Ait-Kaci *et al.*, 1994] H. Ait-Kaci, A. Podelski, et G. Smolka. A feature constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(?):263–283, 1994.

[Aït-Kaci et Nasr, 1985] H. Aït-Kaci et R. Nasr. A logic programming language with built-in inheritance. Rapport Technique AI-068-85, MCC, Juillet 1985.

[Avenhaus et Loría-Sáenz, 1994] J. Avenhaus et C. Loría-Sáenz. On conditional rewrite systems with extra-variables and deterministic logic programs. Dans *Proceedings of the 3rd International Conference on Logic Programming and Automated Reasoning, Kiev (Ukraine)*, rédacteur F. Pfenning, volume 822 de *Lecture Notes in Artificial Intelligence*, pages 215–229. Springer-Verlag, Juillet 1994.

[Bachmair *et al.*, 1992] L. Bachmair, H. Ganzinger, C. Lynch, et W. Snyder. Basic paramodulation and superposition. Dans *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, pages 462–476, 1992.

[Bachmair et Ganzinger, 1990] L. Bachmair et H. Ganzinger. On restrictions of ordered paramodulation with simplification. Dans *Proceedings 10th International Conference on Automated Deduction, Kaiserslautern (Germany)*, rédacteur M. E. Stickel, volume 449 de *Lecture Notes in Computer Science*, pages 427–441. Springer-Verlag, Juillet 1990.

[Bachmair et Ganzinger, 1991] L. Bachmair et H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. Rapport Technique MPI-I-91-208, Max-Planck Institut für Informatik, Saarbrücken, 1991.

[Bachmair et Ganzinger, 1994a] L. Bachmair et H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):1–31, 1994.

[Bachmair et Ganzinger, 1994b] L. Bachmair et H. Ganzinger. Rewrite techniques for transitive relations. Dans *Proc. 9th IEEE Symposium on Logic in Computer Science*, page ?? IEEE Computer Society Press, 1994. Short version of technical report MPI-I-93-249.

[Bachmair, 1988] L. Bachmair. Proof by consistency in equational theories. Dans *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 228–233, 1988.

[Bachmair, 1991] L. Bachmair. *Canonical equational proofs.* Computer Science Logic, Progress in Theoretical Computer Science. Birkhäuser Verlag AG, 1991.

[Bergstra et Klop, 1986] J. A. Bergstra et J. W. Klop. Conditional rewrite rules: Confluency and termination. *Journal of Computer and System Sciences*, 32(3):323–362, 1986.

[Bergstra et Tucker, 1987] J. A. Bergstra et J. V. Tucker. Algebraic specifications of computable and semicomputable data structures. *Theoretical Computer Science*, 50:137–181, 1987.

[Bidoit *et al.*, 1990] rédacteurs M. Bidoit, P. Lescanne, H. J. Kreowski, F. Orejas, et D. Sanella. *Algebraic System Specification and Development: A Survey and Annotated Bibliography*, volume 501 de *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[Birkhoff, 1935] G. Birkhoff. On the structure of abstract algebras. *Proceedings Cambridge Phil. Soc.*, 31:433–454, 1935.

[Bogaert et Tison, 1992] B. Bogaert et S. Tison. Equality and disequality constraints on direct subterms in tree automata. Dans *Proceedings of the 9th Symposium on Theoretical Computer Science*, volume 577 de *Lecture Notes in Computer Science*, pages 161–172, 1992.

[Bosco *et al.*, 1989] P.G. Bosco, C. Cecchi, et C. Moiso. An extension of WAM for K-LEAF: a WAM-based compilation of conditional narrowing. Dans *Proceedings Sixth International Conference on Logic Programming, Lisboa (Portugal)*, pages 325–339. The MIT press, 1989.

[Bouhoula, 1994] A. Bouhoula. *Preuves automatiques par récurrence dans les théories conditionnelles.* Thèse de Doctorat d'Université, Université de Nancy 1, Mars 1994.

[Boyer *et al.*, 1986] R. Boyer, E. Lusk, W. McCune, R. Overbeek, M. Stickel, et L. Wos. Set theory in first-order logic: Clauses for Gödel's axioms. *Journal of Automated Reasoning*, 2(3):287–327, 1986.

[Burris, 1992] S. Burris. Discriminator varieties and symbolic computation. *Journal of Symbolic Computation*, 13(2):175–208, Février 1992.

[Cardelli *et al.*, 1991] Luca Cardelli, Simone Martini, John C. Mitchell, et Andre Scedrov. An extension of System F with subtyping. Rapport Technique 80, Digital Systems Research Center, Palo Alto, California, USA, Décembre 1991.

[Cardelli, 1986] L. Cardelli. Amber. Dans *Combinators and Functional Programming Languages*, rédacteurs G. Cousineau, P.-L. Curien, et B. Robinet, volume 242 de *Lecture Notes in Computer Science.* Springer-Verlag, 1986.

[Cardelli, 1992] Luca Cardelli. Extensible records in a pure calculus of subtyping. Rapport Technique 81, Digital Systems Research Center, Palo Alto, California, USA, Janvier 1992.

[Chen et Hsiang, 1991] H. Chen et J. Hsiang. Order-sorted equational specification and completion. Rapport technique, State University of New York at Stony Brook, Novembre 1991.

[Church, 1940] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Comon *et al.*, 1992] H. Comon, M. Haberstrau, et J.-P. Jouannaud. Decidable problems in shallow equational theories. Dans *Proceedings of LICS'92*, Santa-Cruz (California, USA), Juin 1992.

[Comon, 1990] H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Sciences*, 1(4):387–411, 1990.

[Comon, 1991] H. Comon. Equational formulae with membership constraints. Research Report 649, CNRS-LRI, Mars 1991.

[Comon, 1992] H. Comon. Completion of rewrite systems with membership constraints. Dans *Proceedings of ICALP 92*, rédacteur W. Kuich, volume 623 de *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[Curien et Ghelli, 1990] P.-L. Curien et G. Ghelli. Coherence of subsumption. Dans *Proceedings of CAAP'90, Copenhagen (Denmark)*, rédacteur A. Arnold, volume 431 de *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[Curien et Ghelli, 1991] Pierre-Louis Curien et Giorgio Ghelli. Subtyping + extensionality: Confluence of $\beta\eta$top reduction in $F_{\leq}$. Dans *Theoretical Aspects of Computer Software*, rédacteurs T. Ito et A. R. Meyer, volume 526 de *Lecture Notes in Computer Science*, pages 731–749. Springer-Verlag, Septembre 1991.

[Dauchet, 1989] M. Dauchet. Simulation of Turing machines by a left-linear rewrite rule. Dans *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (N.C., USA)*, rédacteur N. Dershowitz, volume 355 de *Lecture Notes in Computer Science*, pages 109–120. Springer-Verlag, Avril 1989.

[Dershowitz et Jouannaud, 1990a] N. Dershowitz et J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapitre 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[Dershowitz et Jouannaud, 1990b] N. Dershowitz et J.-P. Jouannaud. Rewrite Systems. Dans *Handbook of Theoretical Computer Science*, rédacteur J. van Leeuwen, chapitre 6, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990.

[Dershowitz et Okada, 1990] N. Dershowitz et M. Okada. A rationale for conditional equational programming. *Theoretical Computer Science*, 75:111–138, 1990.

[Dershowitz et Plaisted, 1988] N. Dershowitz et D. A. Plaisted. *Equational Programming*, pages 21–56. J. Richards, Oxford, 1988. Machine Intelligence 11: The Logic and Acquisition of knowledge.

[Ehrig et Mahr, 1985] H. Ehrig et B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and initial semantics*, volume 6 de *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Engberg *et al.*, 1994] rédacteurs U.H. Engberg, K.G. Larsen, et P.D. Mosses. *Proceedings of the 6th Nordic Workshop in Programming Theory*. Basic Reasearch Institute in Computer Science (Aarhus, DK), 1994. Available as BRICS Note NS-94-6.

[Fourman et Scott, 1977] M.P. Fourman et D.S. Scott. Sheaves and logic. Dans *Applications of Sheaves*, rédacteurs M. P. Fourman et C. J. Mulvey, volume 753 de *Lecture Notes in Mathematics*, pages 302–401. Springer-Verlag, 1977.

[Frege, 1884] G. Frege. *Die Grundlagen der Arithmetik.* Breslau, 1884. English translation in [Frege, 1980].

[Frege, 1980] G. Frege. *The Foundations of Arithmetic: A logico-mathematical Enquiry into the concept of number.* Blackwell, Oxford, 1980.

[Fuh et Mishra, 1988] Y.-C. Fuh et P. Mishra. Type inference with subtypes. Dans *Proceedings of ESOP'88*, rédacteur H. Ganzinger, volume 300 de *Lecture Notes in Computer Science*, pages 94–114. Springer-Verlag, Mars 1988.

[Ganzinger, 1987] H. Ganzinger. A completion procedure for conditional equations. Dans *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, rédacteurs S. Kaplan et J.-P. Jouannaud, volume 308 de *Lecture Notes in Computer Science*, pages 62–83. Springer-Verlag, Juillet 1987.

[Ganzinger, 1989] H. Ganzinger. Order-sorted completion: the many-sorted way. Dans *Proceedings International Joint Conference on Theory and Practice of Software Development: Colloquium on Software Engineering*, Lecture Notes in Computer Science. Springer-Verlag, 1989.

[Ganzinger, 1991a] H. Ganzinger. A completion procedure for conditional equations. *Journal of Symbolic Computation*, 11:51–81, 1991.

[Ganzinger, 1991b] H. Ganzinger. Order-sorted completion: the many-sorted way. *Theoretical Computer Science*, 89(1):3–32, 1991.

[Giovannetti et Moisi, 1987] E. Giovannetti et C. Moisi. Notes on the elimination of conditions. Dans *Proceedings 1st International Workshop on Conditional Term Rewriting Systems, Orsay (France)*, rédacteurs J.-P. Jouannaud et S. Kaplan, volume 308 de *Lecture Notes in Computer Science*. Springer-Verlag, Juillet 1987.

[Girard, 1972] J.-Y. Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, Juin 1972.

[Gnaedig *et al.*, 1990] I. Gnaedig, Claude Kirchner, et Hélène Kirchner. Equational completion in order-sorted algebras. *Theoretical Computer Science*, 72:169–202, 1990.

[Goguen *et al.*, 1978] J. A. Goguen, J. W. Thatcher, et E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Dans *Current Trends in Programming methodology IV: Data structuring*, rédacteur R. Yeh, pages 80–144. Prentice Hall, 1978.

[Goguen *et al.*, 1985] J. A. Goguen, J.-P. Jouannaud, et J. Meseguer. Operational semantics for order-sorted algebra. Dans *Proceeding of the 12th International Colloquium on Automata, Languages and Programming, Nafplion (Greece)*. rédacteur W. Brauer, volume 194 de *Lecture Notes in Computer Science*, pages 221–231. Springer-Verlag, 1985.

[Goguen et Diaconescu, 1992] J. A. Goguen et R. Diaconescu. A short survey of order-sorted algebra. *EATCS Bulletin*, 49:121–133, 1992.

[Goguen et Meseguer, 1988] J. A. Goguen et J. Meseguer. Order-sorted algebra I: Partial and overloaded operations, errors and inheritance. Rapport technique, SRI International, Computer Science Lab, 1988. Given as lecture at a Seminar on Types, Carnegie-Mellon University, June 1983.

[Goguen et Meseguer, 1992] Joseph A. Goguen et José Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 2(105):217–273, 1992.

[Goguen, 1978] J. A. Goguen. Order sorted algebras: Exceptions and error sorts, coercions and overloaded operators. Semantics and Theory of Computation Report 14, Computer Science Dept, UCLA, Décembre 1978.

[Gordon, 1993] M. J. C. Gordon. *Introduction to HOL: A Theorem Proving Environment.* Cambridge University Press, 1993.

[Hanus, 1989] M. Hanus. Horn clause programs with polymorphic types: Semantics and resolution. Dans *Proceedings of the TAPSOFT '89*, 1989.

[Hanus, 1991] M. Hanus. Parametric order-sorted types in logic programming. Dans *Proc. of the TAPSOFT '91*, volume 494 de *Lecture Notes in Computer Science*, pages 181–200, 1991.

[Henkin, 1950] L. Henkin. Completeness in the theory of types. *The Journal of Symbolic Logic*, 15(2):81–91, Juin 1950.

[Herbrand, 1930] J. Herbrand. Recherches sur la théorie de la démonstration. *Travaux de la Soc. des Sciences et des Lettres de Varsovie, Classe III*, 33(128), 1930.

[Herbrand, 1971] J. Herbrand. Recherches sur la théorie de la démonstration–thesis 1930. *Logical Writings*, 1971.

[Hill et Topor, 1990] P.M. Hill et R.W. Topor. A semantics for typed logic programs. Rapport Technique TR-90-11, University of Bristol, 1990.

[Hintermeier *et al.*, 1992] C. Hintermeier, Claude Kirchner, et Hélène Kirchner. Strict matchings and unifications in G-algebras. Dans *Proceedings of the 9th WADT - 4th Compass Workshop*, 1992. Centre de Recherche en Informatique de Nancy, 92-R-268.

[Hintermeier *et al.*, 1994] C. Hintermeier, C. Kirchner, et H. Kirchner. Dynamically-typed computations for order-sorted equational presentations –extended abstract–. Dans *Proc. 21st International Colloquium on Automata, Languages, and Programming*, rédacteurs S. Abiteboul et E. Shamir, volume 820 de *Lecture Notes in Computer Science*, pages 450–461. Springer-Verlag, 1994.

[Hintermeier *et al.*, 1995a] C. Hintermeier, C. Kirchner, et H. Kirchner. Sort inheritance for order-sorted equational presentations. Dans *Recent Trends in Data Types Specification*, volume 906 de *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[Hintermeier *et al.*, 1995b] C. Hintermeier, H. Kirchner, et P. Mosses. Specifying λ-calculi in $\mathcal{ЯL}$. Rapport technique, Centre de Recherche en Informatique de Nancy, 1995.

[Hintermeier, 1992] C. Hintermeier. Matchings and unifications in G-algebras. Rapport de DEA, Université de Nancy 1, 1992.

[Hintermeier, 1994] C. Hintermeier. How to transform canonical decreasing ctrss into equivalent canonical trss. Dans *Proceedings of the 4th International Workshop on Conditional Term Rewriting Systems, Jerusalem (Israel)*, Juin 1994.

[Hoffmann et Plump, 1988] B. Hoffmann et D. Plump. Jungle evaluation for efficient term rewriting. Dans *Proceedings 1st International Workshop on Algebraic and Logic Programming*, rédacteurs J. Grabowski, P. Lescanne, et W. Wechler, numéro 343 dans Lecture Notes in Computer Science. Springer-Verlag, 1988.

[Hsiang et Rusinowitch, 1987] J. Hsiang et M. Rusinowitch. On word problem in equational theories. Dans *Proceedings of 14th International Colloquium on Automata, Languages and Programming, Karlsruhe (Germany)*, rédacteur Th. Ottmann, volume 267 de *Lecture Notes in Computer Science*, pages 54–71. Springer-Verlag, 1987.

[Hsiang et Rusinowitch, 1991] J. Hsiang et M. Rusinowitch. Proving refutational completeness of theorem proving strategies: The transfinite semantic tree method. *Journal of the ACM*, 38(3):559–587, Juillet 1991.

[Huet, 1976] G. Huet. *Résolution d'equations dans les langages d'ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[Huet, 1980] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, Octobre 1980. Preliminary version in 18th Symposium on Foundations of Computer Science, IEEE, 1977.

[Hullot, 1980] J.-M. Hullot. Canonical forms and unification. Dans *Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, pages 318–334, Juillet 1980.

[Jech, 1978] T. Jech. *Set Theory*. Pure and Applied Mathematics. Academic Press, 1978.

[Jenks et Sutor, 1992] R. D. Jenks et R. S. Sutor. *Axiom: The scientific Computation System*. Springer-Verlag, Oxford, 1992.

[Josephson et Dershowitz, 1989] N. Alan Josephson et Nachum Dershowitz. An implementation of narrowing. *Journal of Logic Programming*, 6(1&2):57–77, Mars 1989.

[Jouannaud *et al.*, 1992] J.-P. Jouannaud, Claude Kirchner, Hélène Kirchner, et A. Mégrelis. Programming with equalities, subsorts, overloading and parameterization in OBJ. *Journal of Logic Programming*, 12(3):257–280, Février 1992.

[Jouannaud et Kirchner, 1986] J.-P. Jouannaud et Hélène Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. Preliminary version in Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City (USA), 1984.

[Jouannaud et Kirchner, 1990] J.-P. Jouannaud et Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. Research report, Centre de Recherche en Informatique de Nancy, 1990.

[Jouannaud et Kirchner, 1991] J.-P. Jouannaud et Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. Dans *Computational Logic. Essays in honor of Alan Robinson*, rédacteurs Jean-Louis Lassez et G. Plotkin, chapitre 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991.

[Jouannaud et Waldmann, 1986] J.-P. Jouannaud et B. Waldmann. Reductive conditional term rewriting systems. Dans *Proceedings of 3rd IFIP Conf. on Formal description of Programming Concepts*, rédacteur M. Wirsing, Ebberup (Denmark), 1986. Elsevier Science Publishers B. V. (North-Holland).

[Kaplan, 1987] S. Kaplan. A compiler for conditional term rewriting systems. Dans *Proceedings 2nd Conference on Rewriting Techniques and Applications, Bordeaux (France)*, rédacteur P. Lescanne, volume 256 de *Lecture Notes in Computer Science*, pages 25–41, Bordeaux (France), Mai 1987. Springer-Verlag.

[Kirchner *et al.*, 1988a] Claude Kirchner, Hélène Kirchner, et J. Meseguer. Operational semantics of obj-3. Rapport technique, Centre de Recherche en Informatique de Nancy, BP 239, Vandoeuvre les Nancy cedex, France, Octobre 1988.

[Kirchner *et al.*, 1988b] Claude Kirchner, Hélène Kirchner, et J. Meseguer. Operational semantics of OBJ-3. Dans *Proceedings of 15th International Colloquium on Automata, Languages and Programming*, volume 317 de *Lecture Notes in Computer Science*, pages 287–301. Springer-Verlag, 1988.

[Kirchner *et al.*, 1990] Claude Kirchner, Hélène Kirchner, et M. Rusinowitch. Deduction with symbolic constraints. *Revue d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on Automatic Deduction.

[Kirchner *et al.*, 1993] Claude Kirchner, Hélène Kirchner, et M. Vittek. Implementing computational systems with constraints. Dans *Proceedings of the first Workshop on Principles and Practice of Constraint Programming, Providence (R.I., USA)*, rédacteurs Paris Kanellakis, Jean-Louis Lassez, et Vijay Saraswat, pages 166–175. Brown University, 1993.

[Knuth et Bendix, 1970] Donald E. Knuth et P. B. Bendix. Simple word problems in universal algebras. Dans *Computational Problems in Abstract Algebra*, rédacteur J. Leech, pages 263–297. Pergamon Press, Oxford, 1970.

[Knuth, 1981] Donald E. Knuth. *The art of Computer Programming*, volume 2. Addison Wesley, Reading Massachusetts, second édition, 1981.

[Kounalis et Rusinowitch, 1988] E. Kounalis et M. Rusinowitch. On word problems in Horn theories. Dans *Proceedings 9th International Conference on Automated Deduction, Argonne (Ill., USA)*, rédacteurs E. Lusk et R. Overbeek, volume 310 de *Lecture Notes in Computer Science*, pages 527–537. Springer-Verlag, 1988.

[Loveland, 1978] D. Loveland. *Automatic Theorem Proving.* Elsevier Science Publishers B. V. (North-Holland), 1978.

[Lysne et Piris, 1995] O. Lysne et J. Piris. A termination ordering for higher-order rewrite systems. Dans *Proceedings 6th Conference on Rewriting Techniques and Applications, Kaiserslautern (Germany)*, rédacteur J. Hsiang, volume 914 de *Lecture Notes in Computer Science.* Springer-Verlag, 1995.

[Manca *et al.*, 1990] V. Manca, A. Salibra, et G. Scollo. Equational type logic. *Theoretical Computer Science*, 77(1-2):131–159, 1990.

[Mégrelis, 1990] Aristide Mégrelis. *Algèbre galactique — Un procédé de calcul formel, relatif aux semi-fonctions, à l'inclusion et à l'égalité.* Thèse de Doctorat d'Université, Université de Nancy 1, 1990.

[Meinke, 1992] K. Meinke. Universal algebra in higher types. *Theoretical Computer Science*, 100:385–417, 1992.

[Meseguer, 1992] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[Middeldorp et Hamoen, 1994] A. Middeldorp et E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computation*, 5(3 & 4):213ff, 1994.

[Milner et Tofte, 1991] Robin Milner et Mads Tofte. *The definition of Standard ML.* The MIT press, Cambridge, Massachusetts and London, England, 1991.

[Milner, 1993] R. Milner. Elements of interaction. *Communications of the ACM*, 36(1):78–89, Janvier 1993.

[Miola, 1993] rédacteur Alfonso Miola. *Design and Implementation of Symbolic Computation Systems.* Springer-Verlag, Septembre 1993.

[Mosses, 1989] Peter D. Mosses. Unified algebras and institutions. Dans *Proceedings 4th IEEE Symposium on Logic in Computer Science, Pacific Grove*, pages 304–312, 1989.

[Nieuwenhuis et Rubio, 1992a] R. Nieuwenhuis et A. Rubio. Basic superposition is complete. Dans *Proceedings of ESOP'92*, rédacteur B. Krieg-Brückner, volume 582 de *Lecture Notes in Computer Science*, pages 371–389. Springer-Verlag, 1992.

[Nieuwenhuis et Rubio, 1992b] R. Nieuwenhuis et A. Rubio. Theorem proving with ordering constrained clauses. Dans *Proceedings of CADE-11*, rédacteur D. Kapur, volume 607 de *Lecture Notes in Computer Science*, pages 477–491. Springer-Verlag, 1992.

[Nivela et Nieuwenhuis, 1993] P. Nivela et R. Nieuwenhuis. Saturation of first-order (constrained) clauses with the *saturate* system. Dans *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, rédacteur C. Kirchner, volume 690 de *Lecture Notes in Computer Science*, pages 436–440. Springer-Verlag, 1993.

[Oberschelp, 1962] A. Oberschelp. Untersuchungen zur mehrsortigen Quantorenlogik. *Math. Annalen*, 145(1):297–333, 1962.

[Odersky, 1994] M. Odersky. A functional theory of local names. Dans *Proceedings of the 21st Annual ACM Symposium on Principles Of Programming Languages, Portland (Or., USA)*, Janvier 1994.

[Paulson, 1989] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, Septembre 1989.

[Poigné, 1988] A. Poigné. Partial algebras, subsorting and dependent types. prerequisites of error handling in algebraic specifications. Dans *Proceedings of Workshop on Abstract Data Types*, volume 332 de *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[Qian, 1991] Z. Qian. *Extensions of Order-Sorted Algebraic Specifications: Parameterisation, Higher-Order Functions and Polymorphism.* Phd thesis, Universität Bremen (Germany), Mars 1991.

[Quaife, 1992] Art Quaife. Automated deduction in von Neumann-Bernays-Gödel set theory. *Journal of Automated Reasoning*, 8(1):91–147, Février 1992.

[Quine, 1969] W.V.O. Quine. *Set Theory and Its Logic.* The Belknap Press of Harvard University Press, Cambridge (MA/USA), 1969.

[Robinson et Wos, 1969] G. A. Robinson et L. T. Wos. Paramodulation and first-order theorem proving. Dans *Machine Intelligence 4*, rédacteurs B. Meltzer et D. Mitchie, pages 135–150. Edinburgh University Press, 1969.

[Rusinowitch, 1988] M. Rusinowitch. Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure to a complete set of inference rules. Dans *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1988. See also the extended version published in Journal of Symbolic Computation, number 1&2, 1991.

[Russell, 1908] B. Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30:222–262, 1908.

[Schmidt-Schauß, 1987] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* PhD thesis, Universität Kaiserslautern (Germany), 1987.

[Schmidt-Schauß, 1989] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 de *Lecture Notes in Computer Science.* Springer-Verlag, 1989.

[Schmidt, 1938] A. Schmidt. Über deduktive Theorien mit mehreren sorten von Grunddingen. *Math. Annalen*, 115:485–506, 1938.

[Scott, 1976] D. S. Scott. Data types as lattices. *SIAM Journal of Computing*, 5(3):522–587, 1976.

[Scott, 1977] D.S. Scott. Identity and existence in intuitionistic logic. Dans *Applications of Sheaves*, rédacteurs M. P. Fourman et C. J. Mulvey, volume 753 de *Lecture Notes in Mathematics*, pages 660–696. Springer-Verlag, 1977.

[Sieg, 1988] Sieg. Hilbert's program sixty years later. *Journal of Symbolic Logic*, 53, 1988.

[Siekmann, 1989] J. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7(3 & 4):207–274, 1989. Special issue on unification. Part one.

[Sivakumar, 1989] G. Sivakumar. *Proofs and computations in conditional equational theories.* Thèse de Doctorat d'Université, University of Illinois, Urbana-Champaign (IL/USA), 1989.

[Smolka *et al.*, 1989] G. Smolka, W. Nutt, J. A. Goguen, et J. Meseguer. Order-sorted equational computation. Dans *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, rédacteurs H. Aït-Kaci et M. Nivat, pages 297–367. Academic Press, 1989.

[Smolka, 1989] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types.* PhD thesis, FB Informatik, Universität Kaiserslautern, Germany, 1989.

[Spivey, 1988] J.M. Spivey. *Understanding Z: a specification language and its formal semantics.* Cambridge University Press, 1988.

[Takeuti, 1975] G. Takeuti. *Proof Theory.* North-Holland Publishing Co., 1975.

[Thatcher et Wagner, 1976] J. W. Thatcher et E. G. Wagner. Specification of abstract data types using conditional axioms. *IBM T.J. Watson Res Center Rep RC-6214*, 1976.

[Uribe, 1992] T. E. Uribe. Sorted unification using set constraints. Dans *Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA)*, rédacteur D. Kapur, volume 607 de *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Juin 1992.

[Vorobyov, 1994] S. Vorobyov. $F_{\leq}$: Bounded Quantification is NOT Essentially Undecidable. Rapport interne, Centre de Recherche en Informatique de Nancy, Vandœuvre-lès-Nancy, 1994.

[Waldmann, 1989a] U. Waldmann. Semantics of order-sorted specifications. Rapport Technique 297, Universität Dortmund (Germany), 1989.

[Waldmann, 1989b] U. Waldmann. Unification in order-sorted signatures. Rapport Technique 298, Universität Dortmund (Germany), 1989.

[Waldmann, 1992] U. Waldmann. Semantics of order-sorted specifications. *Theoretical Computer Science*, 94(1):1–33, 1992.

[Walther, 1983] C. Walther. Many-sorted calculus based on resolution and paramodulation. Dans *Proceedings of 8th IJCAI, Karlsruhe*, pages 882–891, 1983.

[Walther, 1985] Christoph Walther. A mechanical solution of Schubert's steamroller problem by many-sorted resolution. *Artificial Intelligencer*, 26(2):217–214, 1985.

[Watson et Dick, 1989] P. Watson et J. Dick. Least sorts in order-sorted term rewriting. Rapport technique, Royal Holloway and Bedford New College, University of London, 1989.

[Weidenbach et Ohlbach, 1990] Christoph Weidenbach et Hans-Jürgen Ohlbach. A resolution calculus with dynamic sort structures and partial functions. Dans *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 688–693. Pitman Publishing, London, August 1990.

[Weidenbach, 1991] Christoph Weidenbach. A sorted logic using dynamic sorts. Technical Report MPI-I-91-218, Max-Planck-Institut fuer Informatik, Saarbruecken, 1991.

[Weis *et al.*, 1989] P. Weis, M. V. Aponte, A. Laville, M. Mauny, et A. Suárez. The CAML reference manual. Rapport technique, Projet Formel, INRIA-ENS, 1989. Version 2.6.

[Werner, 1993] A. Werner. A semantic approach to order-sorted rewriting. Dans *Proceedings 5th Conference on Rewriting Techniques and Applications, Montreal (Canada)*, rédacteur C. Kirchner, volume 690 de *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 1993.

[Whitehead et Russell, 1925] A. N. Whitehead et B. Russell. *Principia Mathematica*, volume 1. Cambridge University Press, Cambridge, MA, 1925.

[Whitehead, 1898] A. N. Whitehead. *A Treatise on Universal Algebra, with Applications, I.* Cambridge, 1898. Reprinted in 1960.

[With, 1992a] L. With. Completeness and confluence of order-sorted term rewriting. Dans *Proceedings 3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson (France)*, rédacteurs M. Rusinowitch et J.-L. Rémy, numéro 656 dans Lecture Notes in Computer Science, pages 393–407. Springer-Verlag, Juillet 1992.

[With, 1992b] L. With. Completeness and confluence of order-sorted term rewriting. Dans *Proceedings 3rd International Workshop on Conditional Term Rewriting Systems, Pont-à-Mousson (France)*, rédacteurs M. Rusinowitch et J.-L. Rémy, numéro 656 dans Lecture Notes in Computer Science. Springer-Verlag, Juillet 1992.

Nom : **HINTERMEIER**

Prénom : **Claus**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 12 OCT. 1995 n° 350

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ, NANCY 1 — LE PRÉSIDENT

J.P. FINANCE

3C-F9

UNIVERSITÉ HENRI POINCARÉ - NANCY 1
24-30, rue Lionnois B.P.3069 54013 Nancy cedex
Tél : 83 32 81 81 Fax : 83 32 95 90
Adresse électronique : _____ @presn1.u-nancy.fr

Résumé

Cette thèse est dédiée au développement de langages de spécification formels, de leur sémantique opérationnelle sous forme de systèmes de réécriture décorés et des outils de preuves associés. Nous définissons les logiques R^n et G^n, qui sont des logiques à clauses de Horn égalitaires avec prédicat d'appartenance, adaptées à la spécification et à la programmation polymorphique d'ordre supérieur avec sous-typage dynamique et paramétrique, des fonctions strictes et partielles et une sémantique initiale basée sur la théorie des ensembles. Puis, nous introduisons les termes décorés, qui représentent une structure de données bien adaptée aux spécifications et programmes équationnels typés. Ils permettent d'intégrer le raisonnement égalitaire, les fonctions partielles et le typage dynamique de termes.

Mots-clés: spécifications formelles, logique équationelle, théorie ensembliste, sortes ordonnées, sous-typage dynamique et paramétré, fonctions partielles strictes, réécriture, contraintes, complétion, calculs de superposition, prouveurs de théorèmes automatiques.