Order-Sorted Equational Computation

Gert Smolka, Werner Nutt,
Joseph A. Goguen and José Meseguer

SEKI-Report SR-87-14

SEKI Report SR-87-14, Universität Kaiserslautern, West Germany, December 1987

Order-Sorted Equational Computation

Gert Smolka and Werner Nutt
FB Informatik, Universität Kaiserslautern

Joseph A. Goguen and José Meseguer
SRI International and CSLI

Abstract

The expressive power of many-sorted equational logic can be greatly enhanced by allowing for subsorts and multiple function declarations. In this paper we study some computational aspects of such a logic. We start with a self-contained introduction to order-sorted equational logic including initial algebra semantics and deduction rules. We then present a theory of order-sorted term rewriting and show that the key results for unsorted rewriting extend to sort decreasing rewriting. We continue with a review of order-sorted unification and prove the basic results.

In the second part of the paper we study hierarchical order-sorted specifications with strict partial functions. We define the appropriate homomorphisms for strict algebras and show that every strict algebra is base isomorphic to a strict algebra with at most one error element. For strict specifications, we show that their categories of strict algebras have initial objects. We validate our approach to partial functions by proving that completely defined total functions can be defined as partial without changing the initial algebra semantics. Finally, we provide decidable sufficient criteria for the consistency and strictness of ground confluent rewriting systems.

Keywords: Order-Sorted Equational Logic, Algebraic Specification, Initial Algebra Semantics, Partial Functions, Rewriting, Unification, Subsorts, Inheritance, Logic Programming.

Acknowledgments: We are grateful to Manfred Schmidt-Schauß for many discussions. The research reported in this paper has been supported in part by the Bundesminister für Forschung und Technologie, contract ITR8501A, the Office of Naval Research, contracts N00014-82-C-0333, N00014-85-C-0417 and N00014-86-C-0450, and a gift from the System Development Foundation.

To appear in:
H. Ait-Kaci and M. Nivat,
Resolution of Equations in Algebraic Structures, Academic Press.

Contents

1 Introduction

Many-sorted equational logic is the basis for algebraic specifications [Goguen et al. 78, Meseguer/Goguen 85a, Ehrig/Mahr 85], rewriting techniques [Huet 80, Huet/Oppen 80], unification theory [Siekmann 86], and equational programming [Futatsugi et al. 85, O'Donnell 85, Goguen/Meseguer 86]. In the standard approach, sorts are unrelated and can be thought of as denoting disjoint sets. Order-sorted equational logic, which originated with Goguen [78], improves the expressivity of many-sorted equational logic by adding the notion of subsorts. The standard example of an abstract data type, stacks of natural numbers, can be specified in order-sorted equational logic as follows:

variables: $N\colon \mathbf{nat}$, $\quad S\colon \mathbf{stack}$

$o\colon \rightarrow \mathbf{nat}$, $\quad s\colon \mathbf{nat} \rightarrow \mathbf{nat}$

$\mathbf{empty\_stack} < \mathbf{stack}$, $\quad \mathbf{nonempty\_stack} < \mathbf{stack}$
$estack\colon \rightarrow \mathbf{empty\_stack}$, $\quad push\colon \mathbf{nat} \times \mathbf{stack} \rightarrow \mathbf{nonempty\_stack}$

$top\colon \mathbf{nonempty\_stack} \rightarrow \mathbf{nat}$ $\qquad pop\colon \mathbf{nonempty\_stack} \rightarrow \mathbf{stack}$
$\quad top(push(N, S)) \doteq N$ $\qquad pop(push(N, S)) \doteq S$

The sorts **empty_stack** and **nonempty_stack** are declared as subsorts of **stack**. Semantically, declaring ξ as a subsort of η means that the denotation of ξ must be a subset of the denotation of η. The important point of the example is that with the subsort **nonempty_stack** the correct domains of the selectors *top* and *pop* can be specified. In many-sorted equational logic without subsorts, one has to introduce two error elements for **nat** and **stack** and seven (!) equations to properly extend s, *push*, *top*, and *pop*, thus arriving at a very awkward specification of a very simple thing.

Our second example, a specification of the integers shown in Figure 1.1, illustrates the second key feature of order-sorted equational logic: functions can have more than one declaration. A model satisfies a declaration for a function symbol f if the domain of the denotation of f includes the declared *variables:* I, I': **int**, $\quad$ *Nat*: **nat**, $\quad$ *Negint*: **negint**

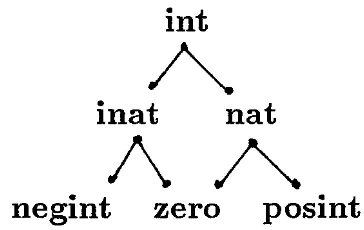
negint $<$ **inat**, $\quad$ **zero** $<$ **inat**, $\quad$ **inat** $<$ **int**
zero $<$ **nat**, $\quad$ **posint** $<$ **nat**, $\quad$ **nat** $<$ **int**

$o\colon \to$ **zero**

$s\colon$ **nat** $\to$ **posint**, $\quad s\colon$ **int** $\to$ **int**

$p\colon$ **inat** $\to$ **negint**, $\quad p\colon$ **int** $\to$ **int**

$s(p(I)) \doteq I, \quad p(s(I)) \doteq I$



$+\colon$ **int** $\times$ **int** $\to$ **int**
$+\colon$ **posint** $\times$ **nat** $\to$ **posint**
$+\colon$ **nat** $\times$ **posint** $\to$ **posint**
$+\colon$ **nat** $\times$ **nat** $\to$ **nat**

$$p(I) + I' \doteq p(I + I')$$
$$o + I \doteq I$$
$$s(I) + I' \doteq s(I + I')$$

true: $\to$ **bool**
false: $\to$ **bool**

$\leq\colon$ **int** $\times$ **int** $\to$ **bool**

$$p(I) \leq I' \doteq I \leq s(I')$$
$$o \leq Nat \doteq true$$
$$o \leq Negint \doteq false$$
$$s(I) \leq I' \doteq I \leq p(I')$$

Figure 1.1. A specification in order-sorted equational logic. Every integer can be represented by a ground term built from o, the successor function s, and the predecessor function p. The elements of the sort **inat** are the negatives of the natual numbers (including zero).

domain and the denotation of f maps every element of the declared domain to an element of the declared codomain.

In the example, the declarations $p\colon$ **inat** $\to$ **negint** and $s\colon$ **nat** $\to$ **posint** generate the elements of the subsorts **inat** and **nat**, while the declarations $p\colon$ **int** $\to$ **int** and $s\colon$ **int** $\to$ **int** extend p and s to all integers. Deleting the declaration $s\colon \textbf{nat} \to \textbf{posint}$ from the specification would make **posint** empty and collapse **nat** to **zero**. Deleting $s\colon \textbf{int} \to \textbf{int}$ would make all equations containing s ill-sorted. Keeping only the declaration $+\colon \textbf{int} \times \textbf{int} \to \textbf{int}$ for $+$ wouldn't change the initial model but results in a less expressive sort discipline. On the other hand, we could make the sort discipline more expressive—without changing the initial model—by adding the declarations

$$+\colon \textbf{negint} \times \textbf{inat} \to \textbf{negint}, \quad +\colon \textbf{inat} \times \textbf{negint} \to \textbf{negint},$$
$$+\colon \textbf{inat} \times \textbf{inat} \to \textbf{inat}, \quad +\colon \textbf{zero} \times \textbf{zero} \to \textbf{zero}.$$

The subspecification

$$\textbf{negint} < \textbf{inat}, \quad \textbf{zero} < \textbf{inat}, \quad \textbf{inat} < \textbf{int}$$
$$\textbf{zero} < \textbf{nat}, \quad \textbf{posint} < \textbf{nat}, \quad \textbf{nat} < \textbf{int}$$

$$o\colon \to \textbf{zero}, \quad s\colon \textbf{nat} \to \textbf{posint}, \quad p\colon \textbf{inat} \to \textbf{negint}$$

of the specification in Figure 1.1 is an equation-free specification of the integers. Giving an equation-free specification of the integers in many-sorted equational logic without subsorts is a rather tedious exercise.

The definition of the less or equal test for integers in Figure 1.1 is by induction over the term structure of the first argument, where the base cases make use of the subsorts **nat** and **negint**. It is known that defining a less or equal test for integers with unconditional equations not using subsorts is complicated: one has to introduce an auxiliary function and an auxiliary sort. These complications disappear if one uses conditional equations [Kaplan 84], but verification methods for the confluence of conditional rewriting systems are complicated and in most cases not practical. On the other hand, as we will show in this paper, the verification methods for confluence extend nicely to order-sorted unconditional rewriting systems.

The examples illustrate several respects in which order-sorted equational logic is more expressive than many-sorted equational logic:

- In many cases the correct domain of a function can be specified by defining the appropriate subsorts. For instance, the subsort **nonempty_stack** of **stack** is the correct domain of the selectors *top* and *pop*. See [Goguen/Meseguer 87b] for a thorough analysis of the constructor-selector problem.

- The use of subsorts makes it easier to give equation-free specifications of types. For instance, while it is easy to give an equation-free specification of the integers in order-sorted equational logic, giving an equation-free specification of the integers in many-sorted equational logic without subsorts is a tedious exercise.

- The use of subsorts often helps to avoid auxiliary functions or conditional equations. This is illustrated by the specification of the less or equal test for integers shown in Figure 1.1.

Research in automated theorem proving [Cohn 83 and 85, Irani/Shin 85, Walther 83 and 85, Schmidt-Schauß 85a] emphasizes another benefit obtained from subsorts, which applies as well to typed logic programming: employing order-sorted unification can drastically reduce the search spaces that come with resolution, paramodulation, and narrowing.

Order-sorted equational logic and extensions of it are the basis of several specification and programming languages. OBJ [Goguen 79, Futatsugi et al. 85] employs order-sorted conditional term rewriting modulo equations and has a powerful generic module system. Eqlog [Goguen/Meseguer 86] extends OBJ to include relational programming à la Prolog. FOOPS [Goguen/Meseguer 87d] extends OBJ to accommodate object-oriented programming. TEL [Smolka 87] integrates equational and relational programming and has an expressive type system combining subsorts with parametric polymorphism à la ML.

LOGIN [Aït-Kaci/Nasr 86] is a Prolog-like language, where ordinary terms are replaced with so-called ψ-terms that are unified with respect to a subsort lattice. Recent work [Smolka/Aït-Kaci 87] shows that LOGIN's type structure can be expressed in order-sorted equational logic by so-called inheritance hierarchies and provides an initial algebra semantics for LOGIN. Inheritance hierarchies provide record notation and a taxonomic data organization scheme.

Unification in these hierarchies [Smolka/Aït-Kaci 87] combines order-sorted unification with ψ-term unification [Aït-Kaci 86, Aït-Kaci/Nasr 86], the operational key ingredient of the unification grammars used in computational linguistics.

The paper is organized in two parts and is aimed at readers familiar with the basic notions of algebraic specification and term rewriting.

The first part starts with a self-contained and compact development of order-sorted equational logic including deduction rules and initial algebra semantics. We then present a theory of order-sorted term rewriting and show that the key results for unsorted rewriting extend to sort decreasing rewriting. Finally, we give a review of order-sorted unification and prove the basic results. Except for parts of the section on order-sorted rewriting, all of the results presented in the first part of the paper have been published before, but not in a single paper and not in a uniform notation. Every section of the first part ends with a subsection sketching the historical development and giving the relevant references. We hope that this comprehensive presentation makes life easier for newcomers and shows that order-sorted equational logic is a quite simple formalism.

The second part of the paper presents a theory of hierarchical order-sorted specifications with strict partial functions. We define the appropriate homomorphisms for strict algebras and show that every strict algebra is isomorphic to a strict algebra with at most one error element. For strict specifications, we show that their categories of strict algebras have initial objects. We validate our approach to partial functions by proving that completely defined total functions can also be defined as partial functions without changing the initial algebra semantics. Finally, we provide decidable sufficient criteria for the consistency and strictness of ground confluent rewriting systems.

The extension of algebraic specification techniques to partial functions is not a new idea [Reichel 80 and 87, Broy/Wirsing 82, Kamin/Archer 84]. However, our approach, which builds on ideas in [Goguen/Meseguer 87c], is particularly simple and has the additional advantage of incorporating subsorts, which allow modelling functions like *pop* or *top* as total.

2 Order-Sorted Equational Logic

This section presents a self-contained development of order-sorted equational logic, which should be easy to follow for readers familiar with the basic notions of equational logic.

2.1 Syntax

Technically, it is convenient to stipulate the following pairwise disjoint and countably infinite sets of symbols:

Sort Symbols (ξ, η, ζ). We use $\vec{\xi}$, $\vec{\eta}$ and $\vec{\zeta}$ to denote possibly empty strings of sort symbols.

Function Symbols (f, g, h). Every function symbol f comes with an **arity** $|f|$ specifying the number of arguments it takes. Function symbols having arity zero are called **constant symbols**.

Variables (x, y, z). Every variable x comes with a **sort** σx, which is a sort symbol. For every sort symbol there exist infinitely many variables having this sort.

A **subsort declaration** has the form $\xi < \eta$, where ξ and η are sort symbols.

A **function declaration** has the form $f\colon \xi_1 \cdots \xi_n \to \xi$, where n is the arity of f and $\xi_1, \ldots, \xi_n$ and ξ are sort symbols.

A **signature** Σ is a set of subsort and function declarations. We say that a sort or function symbol is a **Σ-symbol** if it occurs in a declaration of Σ. A variable is a **Σ-variable** if its sort is a Σ-symbol.

The **subsort order** "$\xi \leq_\Sigma \eta$" of Σ is the least quasi-order $\leq_\Sigma$ on the sort symbols of Σ such that $\xi \leq_\Sigma \eta$ if $(\xi < \eta) \in \Sigma$. The subsort order is extended componentwise to strings of sort symbols. If the signature is clear from the context, we will drop the index Σ in $\xi \leq_\Sigma \eta$.

Let Σ be a signature.

A **Σ-term of sort** ξ is either a variable x such that $\sigma x \leq_\Sigma \xi$, or has the form $f(s_1, \ldots, s_n)$, where there is a declaration $(f : \eta_1 \cdots \eta_n \to \eta) \in \Sigma$ such that $\eta \leq_\Sigma \xi$ and s_i is a Σ-term of sort η_i for $i = 1, \ldots, n$. The letters s, t, u and v will always denote terms.

A **Σ-equation** is an ordered pair of Σ-terms written as $s \doteq t$.

A **syntactic Σ-object** is a Σ-term or a Σ-equation. A syntactic object is called **ground** if it contains no variables. We use $\mathcal{V}(O)$ to denote the set of variables occurring in a syntactic object O. If V is a set of Σ-variables, then a syntactic Σ-object O is called a syntactic **(Σ, V)-object** if $\mathcal{V}(O) \subseteq V$.

A **Σ-substitution** is a function from Σ-terms to Σ-terms such that

1. if s is a Σ-term of sort ξ, then θs is a Σ-term of sort ξ
2. $\theta f(s_1, \ldots, s_n) = f(\theta s_1, \ldots, \theta s_n)$
3. $\mathcal{D}\theta := \{x \mid \theta x \neq x\}$ is finite.

Following the usual abuse of notation, we call $\mathcal{D}\theta$ the **domain of** θ. The letters θ, ψ, and ϕ will always range over substitutions. The composition of Σ-substitutions is again a Σ-substitution. Σ-substitutions are extended to Σ-equations as one would expect.

Proposition 2.1. *Let θ and ψ be Σ-substitutions. Then $\theta = \psi$ if and only if $\mathcal{D}\theta = \mathcal{D}\psi$ and $\theta x = \psi x$ for all $x \in \mathcal{D}\theta$.*

A Σ-term s is called a **Σ-instance** of a Σ-term t if there exists a Σ-substitution θ such that $s = \theta t$. Note that, if t is a term of sort ξ, every instance of t is a term of sort ξ.

A **specification** $\mathcal{S} = (\Sigma, \mathcal{E})$ consists of a signature Σ and a set $\mathcal{E}$ of Σ-equations, called the **axioms** of $\mathcal{S}$. We don't require that Σ or $\mathcal{E}$ are finite since most definitions and results apply to infinite specifications as well. Given a specification $\mathcal{S} = (\Sigma, \mathcal{E})$, it is convenient to call Σ-objects $\mathcal{S}$-objects and Σ-instances $\mathcal{S}$-instances.

2.2 Semantics

Let Σ be a signature. A **Σ-algebra** $\mathcal{A}$ consists of **denotations** $\xi^{\mathcal{A}}$ and $f^{\mathcal{A}}$ for the sort and function symbols of Σ such that:

1. $\xi^{\mathcal{A}}$ is a set

2. if $(\xi < \eta) \in \Sigma$ then $\xi^{\mathcal{A}} \subseteq \eta^{\mathcal{A}}$

3. $\mathrm{C}_{\mathcal{A}} := \bigcup \{\xi^{\mathcal{A}} \mid \xi$ is a sort symbol of $\Sigma\}$ is called the **carrier of $\mathcal{A}$**

4. $f^{\mathcal{A}}$ is a mapping $\mathrm{D}_f^{\mathcal{A}} \to \mathrm{C}_{\mathcal{A}}$ whose domain $\mathrm{D}_f^{\mathcal{A}}$ is a subset of $\mathrm{C}_{\mathcal{A}}^{|f|}$

5. if $(f : \xi_1 \dots \xi_n \to \xi) \in \Sigma$ and $a_i \in \xi_i^{\mathcal{A}}$ for $i = 1, \dots, n$, then $(a_1, \dots, a_n) \in \mathrm{D}_f^{\mathcal{A}}$ and $f^{\mathcal{A}}(a_1, \dots, a_n) \in \xi^{\mathcal{A}}$.

$\mathrm{C}_{\mathcal{A}}^{|f|}$ denotes the cartesian product $\mathrm{C}_{\mathcal{A}} \times \cdots \times \mathrm{C}_{\mathcal{A}}$ having one factor for every argument of f. Note that a function symbol has only one denotation although there can be several declarations for it in the signature. Thus having several declarations for a function symbol does not mean that the function symbol is overloaded.

Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras. A mapping $\gamma\colon \mathrm{C}_{\mathcal{A}} \to \mathrm{C}_{\mathcal{B}}$ is called a **homomorphism** $\mathcal{A} \to \mathcal{B}$ if

1. $\gamma(\xi^{\mathcal{A}}) \subseteq \xi^{\mathcal{B}}$ for every Σ-sort symbol ξ

2. $\gamma(\mathrm{D}_f^{\mathcal{A}}) \subseteq \mathcal{D}_f^{\mathcal{B}}$ for every Σ-function symbol f

3. $\gamma(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(\gamma(a_1), \dots, \gamma(a_n))$ for every Σ-function symbol f and every tuple $(a_1, \dots, a_n) \in \mathrm{D}_f^{\mathcal{A}}$.

Proposition 2.2. *Let Σ be a signature. Then the Σ-algebras together with their homomorphisms comprise a category.*

A homomorphism $\gamma: \mathcal{A} \to \mathcal{B}$ is called an **isomorphism** if there exists a homomorphism $\gamma': \mathcal{B} \to \mathcal{A}$ such that $\gamma\gamma' = \mathrm{id}_{\mathrm{C}_\mathcal{A}}$ and $\gamma'\gamma = \mathrm{id}_{\mathrm{C}_\mathcal{B}}$. Two Σ-algebras are called **isomorphic** if there exists an isomorphism from one to the other.

Example 2.3. A bijective homomorphism is not necessarily an isomorphism. To see this, consider the signature

$$\Sigma = \{a: \to \mathbf{S},\ b: \to \mathbf{T}\},$$

the Σ-algebra $\mathcal{A}$

$$\mathbf{S}^\mathcal{A} = \{a\},\ \mathbf{T}^\mathcal{A} = \{b\},\ a^\mathcal{A} = a,\ b^\mathcal{A} = b,$$

the Σ-algebra $\mathcal{B}$

$$\mathbf{S}^\mathcal{B} = \{a, b\},\ \mathbf{T}^\mathcal{B} = \{a, b\},\ a^\mathcal{B} = a,\ b^\mathcal{B} = b,$$

and the bijective homomorphism $\gamma: \mathcal{A} \to \mathcal{B}$

$$\gamma(a) = a,\ \gamma(b) = b.$$

The inverse mapping γ^{-1} is not a homomorphism $\mathcal{B} \to \mathcal{A}$ since, for instance, $\gamma^{-1}(\mathbf{S}^\mathcal{B}) \not\subseteq \mathbf{S}^\mathcal{A}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be Σ-algebras. We say that a homomorphism $\gamma: \mathcal{A} \to \mathcal{B}$ is a **covering** $\mathcal{A} \to \mathcal{B}$ if the following two conditions are satisfied:

1. if ξ is a Σ-sort symbol and $b \in \xi^\mathcal{B}$, then there exists $a \in \xi^\mathcal{A}$ such that $\gamma(a) = b$
2. if f is a Σ-function symbol and $(b_1, \ldots, b_n) \in \mathrm{D}_f^\mathcal{B}$, then there exists $(a_1, \ldots, a_n) \in \mathrm{D}_f^\mathcal{A}$ such that $\gamma(a_i) = b_i$ for $i = 1, \ldots, n$.

Proposition 2.4. *An injective homomorphism is an isomorphism if and only if it is a covering.*

Construction 2.5. (Term Algebra $\mathcal{T}_{\Sigma,V}$) Let Σ be a signature and V be a set of Σ-variables. Then the following defines a Σ-algebra $\mathcal{T}_{\Sigma,V}$:

- $\xi^{\mathcal{T}_{\Sigma,V}} := \{s \mid s \text{ is a } (\Sigma, V)\text{-term of sort } \xi\}$
- $\mathrm{D}_f^{\mathcal{T}_{\Sigma,V}} := \{(s_1, \ldots, s_n) \mid f(s_1, \ldots, s_n) \text{ is a } (\Sigma, V)\text{-term}\}$
- $f^{\mathcal{T}_{\Sigma,V}}(s_1, \ldots, s_n) := f(s_1, \ldots, s_n)$.

Proposition 2.6. *Let Σ be a signature and V and W be sets of Σ-variables. If θ is a Σ-substitution such that $\mathcal{V}(\theta x) \subseteq W$ for all $x \in V$, then the restriction of θ to (Σ, V)-terms is a homomorphism $\mathcal{T}_{\Sigma,V} \to \mathcal{T}_{\Sigma,W}$. Furthermore, if V is finite and γ is a homomorphism $\mathcal{T}_{\Sigma,V} \to \mathcal{T}_{\Sigma,W}$, then there exists a Σ-substitution θ that agrees with γ on all (Σ, V)-terms.*

Let $\mathcal{A}$ be a Σ-algebra and V be a set of Σ-variables. A **$(V, \mathcal{A})$-assignment** is a mapping $\alpha\colon V \to \mathrm{C}_{\mathcal{A}}$ such that $\alpha(x) \in (\sigma x)^{\mathcal{A}}$ for all variables $x \in V$. Given a $(V, \mathcal{A})$-assignment α and a (Σ, V)-term s, the **denotation $[\![s]\!]_\alpha$ of s in $\mathcal{A}$ under α** is defined as follows:

$$\begin{aligned}[\![x]\!]_\alpha &= \alpha(x)\\ [\![f(s_1, \ldots, s_n)]\!]_\alpha &= f^{\mathcal{A}}([\![s_1]\!]_\alpha, \ldots, [\![s_n]\!]_\alpha).\end{aligned}$$

If s is ground, we write $[\![s]\!]_{\mathcal{A}}$ rather than $[\![s]\!]_\alpha$ since then the denotation only depends on $\mathcal{A}$.

Validity of Σ-equations in a Σ-algebra $\mathcal{A}$ is defined as follows:

$$\mathcal{A} \models s \doteq t \quad :\Longleftrightarrow \quad \forall\, (\mathcal{V}(s \doteq t), \mathcal{A})\text{-assignment } \alpha.\ [\![s]\!]_\alpha = [\![t]\!]_\alpha.$$

If $\mathcal{A} \models s \doteq t$, we say that $s \doteq t$ **is valid in** $\mathcal{A}$ or that $\mathcal{A}$ **satisfies** $s \doteq t$.

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and $\mathcal{A}$ be a Σ-algebra. We say that $\mathcal{A}$ **is an $\mathcal{S}$-algebra** or that $\mathcal{A}$ **is a model of** $\mathcal{S}$ if $\mathcal{A}$ satisfies every equation of $\mathcal{E}$. We say that a Σ-equation $s \doteq t$ **is valid in** $\mathcal{S}$ or that $\mathcal{S}$ **satisfies** $s \doteq t$ if $s \doteq t$ is valid in every $\mathcal{S}$-algebra. We write

$$\mathcal{S} \models s \doteq t \quad \text{or} \quad s =_{\mathcal{S}} t$$

if $\mathcal{S}$ satisfies $s \doteq t$.

Proposition 2.7. *Let $s \doteq t$ be a Σ-equation. Then the equation-free specification $(\Sigma, \emptyset)$ satisfies $s \doteq t$ if and only if $s = t$.*

Proof. One direction is obvious. To see the other direction, suppose $s \doteq t$ is a (Σ, V)-equation that is valid in $(\Sigma, \emptyset)$. Then $s \doteq t$ is valid in $\mathcal{T}_{\Sigma,V}$ since $\mathcal{T}_{\Sigma,V}$ is a $(\Sigma, \emptyset)$-algebra. Since the identity id on V is a $(V, \mathcal{T}_{\Sigma,V})$-assignment, we have $s = [\![s]\!]_{\mathrm{id}} = [\![t]\!]_{\mathrm{id}} = t$. $\square$

Proposition 2.8. *Let $s \doteq t$ be a (Σ, V)-equation that is valid in a Σ-algebra $\mathcal{A}$. Then $[\![s]\!]_\alpha = [\![t]\!]_\alpha$ for every $(V, \mathcal{A})$-assignment α.*

Example 2.9. The converse of the proposition *does not hold* since the denotation of a sort ξ in a model can be empty if there is no ground term of sort ξ. To see this, consider the specification $\mathcal{S}$

$$
\begin{gathered}
\mathit{true}\colon \to \mathbf{bool},\ \mathit{false}\colon \to \mathbf{bool},\ \mathit{foo}\colon \mathbf{void} \to \mathbf{bool} \\
\mathit{foo}(x_{\mathbf{void}}) \doteq \mathit{true},\ \mathit{foo}(x_{\mathbf{void}}) \doteq \mathit{false}
\end{gathered}
$$

where $x_{\mathbf{void}}$ is a variable having the sort **void**. Since the denotation of **void** in $\mathcal{T}_{\Sigma,\emptyset}$ is empty, there exists no $(\{x_{\mathbf{void}}\}, \mathcal{T}_{\Sigma,\emptyset})$-assignment. Hence $[\![\mathit{true}]\!]_\alpha = [\![\mathit{false}]\!]_\alpha$ for every $(\{x_{\mathbf{void}}\}, \mathcal{T}_{\Sigma,\emptyset})$-assignment α, although $\mathit{true} \doteq \mathit{false}$ is not valid in $\mathcal{T}_{\Sigma,\emptyset}$.

Another important consequence of the fact that sorts can be empty is that $s =_{\mathcal{S}} t$, in general, is not transitive. In the specification $\mathcal{S}$ above, for instance, $\mathit{true} =_{\mathcal{S}} \mathit{foo}(x_{\mathbf{void}})$ and $\mathit{foo}(x_{\mathbf{void}}) =_{\mathcal{S}} \mathit{false}$ hold, but $\mathit{true} =_{\mathcal{S}} \mathit{false}$ does not hold.

We say that a sort symbol ξ of a signature Σ is **inhabited** if there is at least one ground Σ-term of sort ξ. A signature is called **fully inhabited** if each of its sort symbols is inhabited.

Proposition 2.10. *Let Σ be a fully inhabited signature and $\mathcal{A}$ be a Σ-algebra. Then a (Σ, V)-equation $s \doteq t$ is valid in $\mathcal{A}$ if and only if $[\![s]\!]_\alpha = [\![t]\!]_\alpha$ for every $(V, \mathcal{A})$-assignment α.*

Proposition 2.11. *Let $\mathcal{S}$ be a specification whose signature is fully inhabited. Then "$s =_{\mathcal{S}} t$" is an equivalence relation.*

Theorem 2.12. (Denotation) *Let $\mathcal{A}$ be a Σ-algebra, V be a set of Σ-variables, and α be a $(V, \mathcal{A})$-assignment. Then:*

- *the denotation function $[\![\cdot]\!]_\alpha$ is a homomorphism $\mathcal{T}_{\Sigma,V} \to \mathcal{A}$*
- *if γ is a homomorphism $\mathcal{T}_{\Sigma,V} \to \mathcal{A}$, then the restriction of γ to V is a $(V, \mathcal{A})$-assignment*
- *if γ is a homomorphism $\mathcal{T}_{\Sigma,V} \to \mathcal{A}$ such that γ agrees with α on V, then $\gamma = [\![\cdot]\!]_\alpha$.*

Corollary 2.13. *Let Σ be a signature. Then the term algebra $\mathcal{T}_{\Sigma,\emptyset}$ is an initial object in the category comprised of the Σ-algebras with their homomorphisms.*

Corollary 2.14. *Let $\mathcal{A}$ and $\mathcal{B}$ be two Σ-algebras and γ be a homomorphism $\mathcal{A} \to \mathcal{B}$. Then every ground Σ-equation that is valid in $\mathcal{A}$ is valid in $\mathcal{B}$.*

Corollary 2.15. *Let $\mathcal{A}$ and $\mathcal{B}$ be two isomorphic Σ-algebras. Then a Σ-equation is valid in $\mathcal{A}$ if and only if it is valid in $\mathcal{B}$.*

Corollary 2.16. *A Σ-algebra $\mathcal{A}$ satisfies a Σ-equation $s \doteq t$ if and only if $\mathcal{A}$ satisfies every Σ-instance of $s \doteq t$.*

Corollary 2.17. *A specification $\mathcal{S}$ satisfies an $\mathcal{S}$-equation $s \doteq t$ if and only if it satisfies every $\mathcal{S}$-instance of $s \doteq t$.*

Let $\mathcal{A}$ be a Σ-algebra. An equivalence relation $\sim$ on the carrier of $\mathcal{A}$ is called a **congruence on $\mathcal{A}$** if for every Σ-function symbol f

$$a_1 \sim b_1 \ \wedge \ \ldots \ \wedge \ a_n \sim b_n \quad \Rightarrow \quad f^{\mathcal{A}}(a_1, \ldots, a_n) \sim f^{\mathcal{A}}(b_1, \ldots, b_n)$$

provided $(a_1, \ldots, a_n) \in \mathrm{D}^{\mathcal{A}}_f$ and $(b_1, \ldots, b_n) \in \mathrm{D}^{\mathcal{A}}_f$.

Construction 2.18. (Quotient Algebra $\mathcal{A}/\!\sim$) Let $\sim$ be a congruence on a Σ-algebra $\mathcal{A}$ and let $\overline{a}$ denote the equivalence class of $a \in \mathrm{C}_{\mathcal{A}}$ with respect to $\sim$. Then the following defines a Σ-algebra $\mathcal{A}/\!\sim$:

- $\xi^{\mathcal{A}/\sim} := \{\overline{a} \mid a \in \xi_{\mathcal{A}}\}$
- $\mathrm{D}_f^{\mathcal{A}/\sim} := \{(\overline{a_1}, \ldots, \overline{a_n}) \mid (a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}}\}$
- $f^{\mathcal{A}/\sim}(\overline{a_1}, \ldots, \overline{a_n}) := \overline{f^{\mathcal{A}}(a_1, \ldots, a_n)}$ if $(a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}}$.

Proposition 2.19. *Let $\sim$ be a congruence on a Σ-algebra $\mathcal{A}$. Then the quotient algebra $\mathcal{A}/\!\sim$ is a Σ-algebra and $\kappa(a) := \overline{a}$ defines a covering $\mathcal{A} \to \mathcal{A}/\!\sim$ called the* **canonical covering** $\mathcal{A} \to \mathcal{A}/\!\sim$.

Let γ be a homomorphism from a Σ-algebra $\mathcal{A}$ to a Σ-algebra $\mathcal{B}$. Then

$$a \sim_\gamma a' \quad :\Longleftrightarrow \quad \gamma(a) = \gamma(a')$$

defines a congruence $\sim_\gamma$ on $\mathcal{A}$ called the **congruence induced by** γ.

Proposition 2.20. *Let γ be a homomorphism $\mathcal{A} \to \mathcal{B}$ and let $\sim$ be a congruence on $\mathcal{A}$ such that $\sim\subseteq\sim_\gamma$. Then there exists a unique homomorphism $\bar{\gamma}: \mathcal{A}/\!\sim\, \to \mathcal{B}$ such that $\gamma = \bar{\gamma}\kappa$. Furthermore,*

1. *if $\sim=\sim_\gamma$, then $\bar{\gamma}$ is injective*
2. *if $\sim=\sim_\gamma$ and γ is a covering, then $\bar{\gamma}$ is an isomorphism $\mathcal{A}/\!\sim\, \to \mathcal{B}$.*

2.3 Deduction and Initial Algebras

Let Σ be a signature and V be a set of variables. We will show that the deduction rules in Figure 2.1 are sound and complete for order-sorted equational logic. The rules are similar to the rules for unsorted equational logic, but there is a subtle difference: the transitivity rule needs to restrict the involved variables because sorts can be empty. Since empty sorts are also possible in

$$(R)\quad \frac{}{\vdash_{\Sigma,V} s \doteq s} \quad \text{if } s \text{ is a } \Sigma\text{-term}$$

$$(S)\quad \frac{\vdash_{\Sigma,V} s \doteq t}{\vdash_{\Sigma,V} t \doteq s}$$

$$(T)\quad \frac{\vdash_{\Sigma,V} s \doteq t \quad \vdash_{\Sigma,V} t \doteq u}{\vdash_{\Sigma,V} s \doteq u} \quad \text{if } \mathcal{V}(t) \subseteq V$$

$$(C)\quad \frac{\vdash_{\Sigma,V} s_1 \doteq t_1 \quad \cdots \quad \vdash_{\Sigma,V} s_n \doteq t_n}{\vdash_{\Sigma,V} f(s_1,\ldots,s_n) \doteq f(t_1,\ldots,t_n)} \quad \begin{array}{l}\text{if } f(s_1,\ldots,s_n) \text{ and} \\ f(t_1,\ldots,t_n) \text{ are } \Sigma\text{-terms}\end{array}$$

$$(I)\quad \frac{\vdash_{\Sigma,V} s \doteq t}{\vdash_{\Sigma,V} \theta s \doteq \theta t} \quad \text{if } \theta \text{ is } \Sigma\text{-substitution}$$

Figure 2.1. The deduction rules for order-sorted equational logic.

many-sorted equational logic, there is no explicit difference between the deduction rules for many-sorted and order-sorted equational logic.

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification. Then we write

$$\mathcal{S} \vdash s \doteq t \quad :\Longleftrightarrow \quad \mathcal{E} \vdash_{\Sigma,\mathcal{V}(s \doteq t)} s \doteq t.$$

We say that an equation $s \doteq t$ **is deducible in** $\mathcal{S}$ if $\mathcal{S} \vdash s \doteq t$.

Proposition 2.21. *Let Σ be a signature, V be a set of Σ-variables, and $\mathcal{E}$ be a set of Σ-equations. Then $s \doteq t$ is a Σ-equation if $\mathcal{E} \vdash_{\Sigma,V} s \doteq t$.*

Theorem 2.22. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification, $\mathcal{A}$ be an $\mathcal{S}$-algebra, and $\mathcal{E} \vdash_{\Sigma,V} s \doteq t$, where V is a set of Σ-variables. Then $[\![s]\!]_\alpha = [\![t]\!]_\alpha$ for every $(V \cup \mathcal{V}(s \doteq t), \mathcal{A})$-assignment α.*

Proof. By induction on the structure of the derivation $\mathcal{E} \vdash_{\Sigma,V} s \doteq t$. □

Corollary 2.23. **(Soundness)** *If $\mathcal{S} \vdash s \doteq t$, then $\mathcal{S} \models s \doteq t$.*

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and V be a set of Σ-variables. Then

$$s \sim_{\mathcal{S},V} t \quad :\Longleftrightarrow \qquad s \text{ and } t \text{ are } (\Sigma, V)\text{-terms and } \mathcal{E} \vdash_{\Sigma,V} s \doteq t$$

defines a congruence on $\mathcal{T}_{\Sigma,V}$. The quotient $\mathcal{T}_{\mathcal{S},V} := \mathcal{T}_{\Sigma,V}/\sim_{\mathcal{S},V}$ is called the **quotient term algebra of $\mathcal{S}$ and V.**

Proposition 2.24. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification, $\mathcal{A}$ be an $\mathcal{S}$-algebra, and γ be a Σ-homomorphism $\mathcal{T}_{\Sigma,V} \to \mathcal{A}$. Then $\sim_{\mathcal{S},V} \subseteq \sim_\gamma$.*

Proof. Let $s \sim_{\mathcal{S},V} t$. Since the restriction of γ to V is a $(V, \mathcal{A})$-assignment, we know by the Denotation Theorem and Theorem 2.22 that $\gamma(s) = [\![s]\!]_\gamma = [\![t]\!]_\gamma = \gamma(t)$. Thus $s \sim_\gamma t$. $\square$

Theorem 2.25. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and V be a set of Σ-variables. Then $\mathcal{T}_{\mathcal{S},V}$ is an $\mathcal{S}$-algebra.*

Proof. Let $s \doteq t$ be an axiom of $\mathcal{S}$ and α be a $(\mathcal{V}(s \doteq t), \mathcal{T}_{\mathcal{S},V})$-assignment. We have to show that $[\![s]\!]_\alpha = [\![t]\!]_\alpha$. Let β be a $(\mathcal{V}(s \doteq t), \mathcal{T}_{\Sigma,V})$-assignment such that $\alpha(x) = \kappa(\beta(x))$ and $\beta(x)$ is a (Σ, V)-term of sort σx, where κ is the canonical covering $\mathcal{T}_{\Sigma,V} \to \mathcal{T}_{\mathcal{S},V}$. By the Denotation Theorem we know that $[\![\cdot]\!]_\alpha = \kappa[\![\cdot]\!]_\beta$. Since $[\![\cdot]\!]_\beta$ is a homomorphism $\mathcal{T}_{\Sigma,\mathcal{V}(s \doteq t)} \to \mathcal{T}_{\Sigma,V}$, we know by Proposition 2.6 that there exists a Σ-substitution θ that agrees with $[\![\cdot]\!]_\beta$ on all $(\Sigma, \mathcal{V}(s \doteq t))$-terms. Since $s \doteq t$ is an axiom of $\mathcal{S}$, we know that $\mathcal{E} \vdash_{\Sigma,V} \theta s \doteq \theta t$. Hence $\kappa(\theta s) = \kappa(\theta t)$ and $[\![s]\!]_\alpha = \kappa([\![s]\!]_\beta) = \kappa(\theta s) = \kappa(\theta t) = \kappa([\![t]\!]_\beta) = [\![t]\!]_\alpha$. $\square$

Theorem 2.26. **(Initiality)** *Let $\mathcal{S}$ be a specification. Then $\mathcal{I}_\mathcal{S} := \mathcal{T}_{\mathcal{S},\emptyset}$ is an initial object in the category comprised of all $\mathcal{S}$-algebras and their homomorphisms.*

Proof. The claim follows from the fact that $\mathcal{T}_{\Sigma,\emptyset}$ is an initial object in the category of Σ-algebras, Proposition 2.24 and Proposition 2.20. $\square$

Theorem 2.27. (Soundness and Completeness) *Let $\mathcal{S}$ be a specification. Then*

$$\mathcal{S} \models s \doteq t \quad \Longleftrightarrow \quad \mathcal{S} \vdash s \doteq t$$

for every $\mathcal{S}$-equation $s \doteq t$.

Proof. The soundness direction has been already established. To show the completeness direction, let $\mathcal{S} \models s \doteq t$. By the preceding theorem we know that $s \doteq t$ is valid in $\mathcal{T}_{\mathcal{S},\mathcal{V}(s \doteq t)}$. Since the restriction of the canonical covering $\kappa: \mathcal{T}_{\Sigma,\mathcal{V}(s \doteq t)} \to \mathcal{T}_{\mathcal{S},\mathcal{V}(s \doteq t)}$ to $\mathcal{V}(s \doteq t)$ is a $(\mathcal{V}(s \doteq t), \mathcal{T}_{\mathcal{S},\mathcal{V}(s \doteq t)})$-assignment, we know that $\kappa(s) = [\![s]\!]_\kappa = [\![t]\!]_\kappa = \kappa(t)$. Hence $\mathcal{E} \vdash_{\Sigma,\mathcal{V}(s \doteq t)} s \doteq t$, which implies $\mathcal{S} \vdash s \doteq t$ by definition. $\square$

When order-sorted equational logic is used as a specification or programming language, a specification is written such that its initial algebra formalizes a given intuition. In short, a specification "specifies" its initial algebra. For instance, the specification in Figure 1.1 in fact specifies the integers.

To support your intuition on initial algebras, we give an explicit construction of $\mathcal{I}_\mathcal{S}$.

Construction 2.28. (Initial Algebra $\mathcal{I}_\mathcal{S}$) Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification. Then the initial $\mathcal{S}$-algebra $\mathcal{I}_\mathcal{S}$ can be obtained as follows:

- $\xi^{\mathcal{I}_\mathcal{S}} := \{\overline{s} \mid s \text{ is a ground } \Sigma\text{-term of sort } \xi\}$
- $\mathrm{D}_f^{\mathcal{I}_\mathcal{S}} := \{(\overline{s_1}, \ldots, \overline{s_n}) \mid f(s_1, \ldots, s_n) \text{ is a ground } \Sigma\text{-term}\}$
- $f^{\mathcal{I}_\mathcal{S}}(\overline{s_1}, \ldots, \overline{s_n}) := \overline{f(s_1, \ldots, s_n)}$ if $f(s_1, \ldots, s_n)$ is a ground Σ-term

where

$$\overline{s} := \{t \mid \mathcal{E} \vdash_{\Sigma,\emptyset} s \doteq t \text{ and } t \text{ is a ground } \Sigma\text{-term}\}$$

for every ground Σ-term s.

Theorem 2.29. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification. A Σ-algebra $\mathcal{I}$ is an initial object in the category of the $\mathcal{S}$-algebras if and only if*

- *$\mathcal{I}$ has* **no junk**, *that is, the denotation homomorphism $[\![\cdot]\!]_{\mathcal{I}}$ is a covering $\mathcal{T}_{\Sigma,\emptyset} \to \mathcal{I}$*
- *$\mathcal{I}$ has* **no confusion**, *that is, a ground Σ-equation is valid in $\mathcal{I}$ if and only if it is deducible in $\mathcal{S}$.*

Proof. By the Denotation Theorem we know that the denotation homomorphism $[\![\cdot]\!]_{\mathcal{I}_\mathcal{S}}$ is the canonical covering $\kappa: \mathcal{T}_{\Sigma,\emptyset} \to \mathcal{I}_\mathcal{S}$. Thus $\mathcal{I}_\mathcal{S}$ has no junk. By the construction of $\mathcal{I}_\mathcal{S}$ it is clear that $\mathcal{I}_\mathcal{S}$ has no confusion.

To show the other direction, let $\mathcal{I}$ be a Σ-algebra without junk and confusion. It suffices to show that $\mathcal{I}$ and $\mathcal{I}_\mathcal{S}$ are isomorphic. Since $\mathcal{I}$ has no junk, we know that the denotation homomorphism $[\![\cdot]\!]_{\mathcal{I}}: \mathcal{T}_{\Sigma,\emptyset} \to \mathcal{I}$ is a covering. Since $\mathcal{I}$ has no confusion, we know that $\sim_\kappa = \sim_{\mathcal{S},\emptyset} = \sim_{[\![\cdot]\!]_{\mathcal{I}}}$, where κ is the canonical covering $\mathcal{T}_{\Sigma,\emptyset} \to \mathcal{I}_\mathcal{S}$. Hence we know by Proposition 2.20 that there exists an isomorphism $\gamma: \mathcal{I}_\mathcal{S} \to \mathcal{I}$ such that $[\![\cdot]\!]_{\mathcal{I}} = \gamma\kappa$. □

Theorem 2.30. (Structural Induction) *Let $\mathcal{S}$ be a specification and $s \doteq t$ be an $\mathcal{S}$-equation. Then $s \doteq t$ is valid in the initial algebra $\mathcal{I}_\mathcal{S}$ if and only if every ground $\mathcal{S}$-instance of $s \doteq t$ is deducible in $\mathcal{S}$.*

Proof. One direction follows by Corollary 2.16 and the no confusion part of Theorem 2.29. To show the other direction, suppose that $\mathcal{I}_\mathcal{S}$ satisfies every ground $\mathcal{S}$-instance of $s \doteq t$ and let α be a $(\mathcal{V}(s \doteq t), \mathcal{I}_\mathcal{S})$-assignment. Since $\mathcal{I}_\mathcal{S}$ has no junk, there exists an $\mathcal{S}$-substitution θ such that $\theta s \doteq \theta t$ is ground and $\alpha(x) = [\![\theta x]\!]_{\mathcal{I}_\mathcal{S}}$ for every $x \in \mathcal{V}(s \doteq t)$. By our assumption we know that $[\![\theta s]\!]_{\mathcal{I}_\mathcal{S}} = [\![\theta t]\!]_{\mathcal{I}_\mathcal{S}}$. By Proposition 2.6 we know that the restriction of θ to $(\Sigma, \mathcal{V}(s \doteq t))$-terms is a homomorphism $\mathcal{T}_{\Sigma,\mathcal{V}(s \doteq t)} \to \mathcal{T}_{\Sigma,\emptyset}$. Furthermore, we know that the composition $[\![\cdot]\!]_{\mathcal{I}_\mathcal{S}}\theta$ agrees with α on $\mathcal{V}(s \doteq t)$. Hence we know by the Denotation Theorem that $[\![\cdot]\!]_\alpha = [\![\cdot]\!]_{\mathcal{I}_\mathcal{S}}\theta$. Thus $[\![s]\!]_\alpha = [\![\theta s]\!]_{\mathcal{I}_\mathcal{S}} = [\![\theta t]\!]_{\mathcal{I}_\mathcal{S}} = [\![t]\!]_\alpha$. □

2.4 Remarks and References

Order-sorted algebra originated with [Goguen 78]. This paper shows that order-sorted algebras are just the right solution for algebraic specification and proves many basic results, including the existence of initial algebras. However, its approach is more complicated than necessary. Gogolla [83, 86] improves and simplifies the approach of [Goguen 78] and studies several methods for error handling with subsorts. Poigné [84] discusses subsorts in the context of parameterized specifications. Independently, Oberschelp [62] argues for order-sorted logic as a more natural logical language for expressing mathematics and presents models and deduction for an order-sorted predicate logic.

Goguen and Meseguer [87c] give a broad development of order-sorted equational logic including conditional equations and a reduction to many-sorted equational logic. The algebras in [Goguen/Meseguer 87c] differ from the algebras in this paper in that functions with several declarations are overloaded, that is, every function declaration of the signature has a separate denotation in the algebra. Smolka [86] proves that nonoverloaded semantics as in this paper and overloaded semantics as in [Goguen/Meseguer 87c] define the same notion of validity.

Smolka [86] and Goguen and Meseguer [87a] present order-sorted definite clause logics with equations and relations. Schmidt-Schauß [87] develops a generalized order-sorted logic that allows for so-called term declarations, for instance, $x_{\mathbf{nat}} + x_{\mathbf{nat}}$: **evennat**. Term declarations originated with [Goguen 78] and are generalized to sort constraints in [Goguen et al. 85].

3 Order-Sorted Rewriting

In this section we generalize the most important notations and results of term rewriting [Huet 80, Huet/Oppen 80] to order-sorted equational logic. It turns out that, in general, the key results for rewriting do not carry over to order-sorted rewriting. However, for the class of sort decreasing rewriting systems, all notations and results from unsorted rewriting generalize nicely.

3.1 Basic Definitions

The **positions** or **occurrences** of a term are defined as usual as finite sequences of postive integers. We use s/π to denote the **subterm of** s **at position** π, and $s[\pi \leftarrow t]$ to denote the term obtained from s by replacing the subterm at position π with t. Note that, for Σ-terms s and t and a position π of s, $s[\pi \leftarrow t]$ is not necessarily a Σ-term since sort constraints might be violated. This is one important difference from unsorted rewriting.

A **Σ-rewrite rule** $s \longrightarrow t$ is a Σ-equation $s \doteq t$ such that s is not a variable and every variable occurring in the right-hand side t occurs in the left-hand side s. A **rewriting system** is a specification $\mathcal{R} = (\Sigma, \mathcal{E})$ such that every equation in $\mathcal{E}$ is a rewrite rule. A rewriting system $\mathcal{R} = (\Sigma, \mathcal{E})$ defines a binary relation $\longrightarrow_{\mathcal{R}}$ called **rewriting relation** on the set of all Σ-terms as follows: $s \longrightarrow_{\mathcal{R}} t$ if and only if there exists a position π of s and a Σ-instance $u \longrightarrow v$ of a rule of $\mathcal{R}$ such that $s/\pi = u$ and $t = s[\pi \leftarrow v]$. We use $\longrightarrow_{\mathcal{R}}^*$ to denote the reflexive and transitive closure of $\longrightarrow_{\mathcal{R}}$ on the set of all Σ-terms. Note that we defined $s \longrightarrow_{\mathcal{R}}^* t$ such that s and t must be Σ-terms.

Proposition 3.1. (Stability) *Let $\mathcal{R}$ be a rewriting system. Then the rewriting relation $\longrightarrow_{\mathcal{R}}$ is* **stable**, *that is,*

$$s \longrightarrow_{\mathcal{R}} t \quad \Rightarrow \quad \theta s \longrightarrow_{\mathcal{R}} \theta t$$

if θ is an $\mathcal{R}$-substitution.

Proposition 3.2. (Soundness) *Let $\mathcal{R}$ be a rewriting system. Then*

$$s \longrightarrow_{\mathcal{R}}^* t \quad \Rightarrow \quad s =_{\mathcal{R}} t.$$

Proof. The claim can be proved by induction on the length of the derivation $s \rightarrow^*_{\mathcal{R}} t$ using the fact that rewriting does not introduce new variables. $\square$

To discuss order-sorted rewriting further, we need some notations and results for binary relations from [Huet 80]. Let $\rightarrow$ be a binary relation on some set. We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$, and $\leftrightarrow^*$ to denote the reflexive, symmetric, and transitive closure of $\rightarrow$. We write $x \downarrow y$ (read "x and y **converge**") if there exists a z such that $x \rightarrow^* z$ and $y \rightarrow^* z$. The relation $\rightarrow$ is called **locally confluent** if $x \rightarrow y$ and $x \rightarrow z$ always implies $y \downarrow z$. The relation $\rightarrow$ is called **confluent**, if $x \rightarrow^* y$ and $x \rightarrow^* z$ always implies $y \downarrow z$. We say that the relation $\rightarrow$ is **terminating** if there are no infinite chains $x_1 \rightarrow x_2 \rightarrow \cdots$. An element x is called **$\rightarrow$-normal** if there is no y such $x \rightarrow y$. An element x is called **$\rightarrow$-reducible** if there exists an element y such that $x \rightarrow y$. We say that y is a **$\rightarrow$-normal form of** x if $x \rightarrow^* y$ and y is $\rightarrow$-normal. The following theorems are proven in [Huet 80].

Proposition 3.3. *Let $\rightarrow$ be a confluent relation. Then no element has more than one $\rightarrow$-normal form. Furthermore, if $\rightarrow$ is confluent and terminating, then every element has exactly one $\rightarrow$-normal form.*

Theorem 3.4. *Let $\rightarrow$ be a confluent relation. Then $x \leftrightarrow^* y$ if and only if $x \downarrow y$.*

Theorem 3.5. *A relation is confluent if it is locally confluent and terminating.*

The specification in Figure 1.1 is a confluent and terminating rewriting system.

3.2 Compatibility and the Completeness Theorem

Example 3.6. A key result for unsorted rewriting states that $s =_{\mathcal{R}} t$ if and only if $s \leftrightarrow^*_{\mathcal{R}} t$. In general, this result does not hold for order-sorted rewriting.

To see this, consider the following confluent and terminating rewriting system $\mathcal{R}$

$$\mathbf{A} < \mathbf{B}, \quad a\colon \to \mathbf{A}, \quad a'\colon \to \mathbf{A}, \quad b\colon \to \mathbf{B}, \quad f\colon \mathbf{A} \to \mathbf{A}$$
$$a \to b, \quad a' \to b,$$

for which $f(a) =_{\mathcal{R}} f(a')$ holds but $f(a) \leftrightarrow^*_{\mathcal{R}} f(a')$ does not hold. The incompleteness of $\leftrightarrow^*_{\mathcal{R}}$ stems from the fact that $f(b)$ is not an $\mathcal{R}$-term.

We say that a rewriting system $\mathcal{R}$ is **compatible** if, for every $\mathcal{R}$-term s, every position π of s, and every $\mathcal{R}$-instance $u \to v$ of a rule of $\mathcal{R}$ such that $s/\pi = u$, we have that $s[\pi \leftarrow v]$ is an $\mathcal{R}$-term. For a compatible rewriting system, the applicability of a rule to a term s doesn't depend on the overall structure of s, but solely on the existence of a subterm in s that is an instance of the left hand side of the rule.

The rewriting system $\mathcal{R}$ in the preceding example isn't compatible since $f(b)$ isn't an $\mathcal{R}$-term. The rewriting system in Figure 1.1 is compatible.

Proposition 3.7. (Compatibility) *Let $\mathcal{R}$ be a compatible rewriting system, s and t be $\mathcal{R}$-terms, and π be an position of s. Then:*

$$s/\pi \to^*_{\mathcal{R}} t \quad \Rightarrow \quad s \to^*_{\mathcal{R}} s[\pi \leftarrow t].$$

Proof. By induction on the depth of s and the length of $s/\pi \to^*_{\mathcal{R}} t$. □

Example 3.8. The relation $\leftrightarrow^*_{\mathcal{R}}$ can be unsound if there are empty sorts and $\mathcal{R}$ isn't confluent. Let $\mathcal{R}$ be the nonconfluent and compatible rewriting system

$$\mathit{true}\colon \to \mathbf{bool}, \quad \mathit{false}\colon \to \mathbf{bool}, \quad \mathit{foo}\colon \mathbf{void} \to \mathbf{bool}$$
$$\mathit{foo}(x_{\mathbf{void}}) \doteq \mathit{true}, \quad \mathit{foo}(x_{\mathbf{void}}) \doteq \mathit{false}.$$

Then we have $\mathit{true} \leftrightarrow^*_{\mathcal{R}} \mathit{false}$, but $\mathit{true} \doteq \mathit{false}$ isn't valid in the initial algebra of $\mathcal{R}$.

Theorem 3.9. (Soundness and Completeness) *Let $\mathcal{R}$ be a compatible and confluent rewriting system. Then*

$$s =_{\mathcal{R}} t \quad \Longleftrightarrow \quad s \downarrow_{\mathcal{R}} t \quad \Longleftrightarrow \quad s \leftrightarrow^*_{\mathcal{R}} t$$

for every $\mathcal{R}$-equation $s \doteq t$.

Proof. Let $\mathcal{R} = (\Sigma, \mathcal{E})$ be a compatible and confluent rewriting system. Since the second equivalence holds for every confluent relation, it suffices to show the first equivalence.

1. "$\Leftarrow$". Let $s \downarrow_{\mathcal{R}} t$. Then there exists a term u such that $s \rightarrow^*_{\mathcal{R}} u$ and $t \rightarrow^*_{\mathcal{R}} u$. By Proposition 3.2 we know that $s =_{\mathcal{R}} u$ and $t =_{\mathcal{R}} u$. Hence $s =_{\mathcal{R}} t$ since $\mathcal{V}(u) \subseteq \mathcal{V}(s)$.

2. "$\Rightarrow$". Let V be a set of Σ-variables and $\mathcal{E} \vdash_{\Sigma,V} s \doteq t$. Since order-sorted equational deduction is complete, it suffices to show that $s \downarrow_{\mathcal{R}} t$. We prove this claim by induction on the size of the derivation $\mathcal{E} \vdash_{\Sigma,V} s \doteq t$. If $s \doteq t$ is in $\mathcal{E}$, then $s \rightarrow_{\mathcal{R}} t$. If $s \doteq t$ is obtained by the reflexivity rule, then $s \rightarrow^*_{\mathcal{R}} t$. If $s \doteq t$ is obtained by the symmetry rule, we know by the induction hypothesis that $t \downarrow_{\mathcal{R}} s$.

If $s \doteq t$ is obtained by the transitivity rule, there exists a term u such that $\mathcal{E} \vdash_{\Sigma,V} s \doteq u$ and $\mathcal{E} \vdash_{\Sigma,V} u \doteq t$. By the induction hypothesis we know that $s \downarrow_{\mathcal{R}} u$ and $u \downarrow_{\mathcal{R}} t$. Hence there exist two terms v_1 and v_2 such that $s \rightarrow^*_{\mathcal{R}} v_1$, $u \rightarrow^*_{\mathcal{R}} v_1$, $u \rightarrow^*_{\mathcal{R}} v_2$, and $t \rightarrow^*_{\mathcal{R}} v_2$. Since $\mathcal{R}$ is confluent, there exists a term v_3 such that $v_1 \rightarrow^*_{\mathcal{R}} v_3$, and $v_2 \rightarrow^*_{\mathcal{R}} v_3$. Hence we have $s \downarrow_{\mathcal{R}} t$.

If $s \doteq t$ is obtained by the congruence rule, there exist terms $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ such that $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ for some function symbol f and $\mathcal{E} \vdash_{\Sigma,V} s_i \doteq t_i$ for $i = 1, \ldots, n$. By the induction hypothesis we know that $s_i \downarrow_{\mathcal{R}} t_i$ for $i = 1, \ldots, n$. Hence $f(s_1, \ldots, s_n) \downarrow_{\mathcal{R}} f(t_1, \ldots, t_n)$ since $\mathcal{R}$ is compatible.

If $s \doteq t$ is obtained by the instantiation rule, $s \downarrow_{\mathcal{R}} t$ follows from the induction hypothesis by the Stability Proposition 3.1. $\square$

Since we are mainly interested in initial algebra semantics, confluence is actually an unnecessarily strong requirement. We call a rewriting system $\mathcal{R}$ **ground confluent** if the restriction of the rewriting relation $\to_{\mathcal{R}}$ to ground $\mathcal{R}$-terms is confluent.

Theorem 3.10. (Soundness and Completeness) *Let $\mathcal{R}$ be a compatible and ground confluent rewriting system. Then*

$$s =_{\mathcal{R}} t \quad \iff \quad s \downarrow_{\mathcal{R}} t \quad \iff \quad s \leftrightarrow^*_{\mathcal{R}} t$$

for every ground $\mathcal{R}$-equation $s \doteq t$.

Proof. Analogous to the proof of the preceding theorem. □

In Section 7 we will show that for every rewriting system one can construct a semantically equivalent rewriting system that is compatible. Thus, compatibility is no problem in practice.

3.3 Sort Decreasingness and the Critical Pair Theorem

The second key theorem for unsorted term rewriting says that $\to_{\mathcal{R}}$ is locally confluent if all critical pairs of $\mathcal{R}$ converge. We will see that this result, in general, does not hold for order-sorted rewriting, even if $\mathcal{R}$ is compatible. We start by defining overlaps and critical pairs.

We say that a syntactical Σ-object O' is a **variant** of a Σ-object O if O' is obtainable from O by consistent variable renaming, that is, there exist Σ-substitutions θ and ψ such that $O' = \theta O$ and $O = \psi O'$.

An **overlap** of a rewriting system $\mathcal{R}$ is a triple $(s \to t, \pi, s' \to t')$ such that

1. $s \to t$ and $s' \to t'$ are variable disjoint variants of rules of $\mathcal{R}$ and π is an position of s such that s/π is not a variable
2. if $s/\pi = s$, then $s \to t$ is not a variant of $s' \to t'$
3. there exist an $\mathcal{R}$-substitution θ such that $(\theta s)/\pi = \theta s'$.

We say that an overlap $(s \to t, \pi, s' \to t')$ is a **variant** of an overlap $(u \to v, \pi, u' \to v')$ if $u \to v$ is a variant of $s \to t$ and $u' \to v'$ is a variant of $s' \to t'$.

Proposition 3.11. *A finite rewriting system has only finitely many overlaps up to variants.*

A **critical pair of an overlap** $(s \to t, \pi, s' \to t')$ of $\mathcal{R}$ is a pair $(\theta t, \theta(s[\pi \leftarrow t']))$ such that $(\theta s)/\pi = \theta s'$, $\theta(s[\pi \leftarrow t'])$ is an $\mathcal{R}$-term, and θ is an $\mathcal{R}$-substitution. We say that a pair (s, t) is **$\mathcal{R}$-critical** if (s, t) is a critical pair of an overlap of $\mathcal{R}$. We say that a pair (s, t) of $\mathcal{R}$-terms **converges in $\mathcal{R}$** if $s \downarrow_{\mathcal{R}} t$.

Proposition 3.12. *Let $\mathcal{R}$ be a rewriting system. Every instance of an $\mathcal{R}$-critical pair is an $\mathcal{R}$-critical pair.*

A set C of $\mathcal{R}$-critical pairs is called **complete for $\mathcal{R}$** if every $\mathcal{R}$-critical pair is an $\mathcal{R}$-instance of a pair in C.

Proposition 3.13. *Let $\mathcal{R}$ be a rewriting system and C be a complete set of $\mathcal{R}$-critical pairs. Then every $\mathcal{R}$-critical pair converges in $\mathcal{R}$ if every pair in C converges in $\mathcal{R}$.*

For many-sorted rewriting without subsorts, it is well-known that all critical pairs of an overlap can be represented by a single "most general" pair. This isn't true, in general, for order-sorted rewriting, as we will see in the section on unification. However, under mild restrictions, every overlap has still a finite complete set of critical pairs, and such a complete set can be computed using an order-sorted unification algorithm.

The overlaps of the rewriting system in Figure 1.1 are:

$$\begin{array}{llll}
(s(p(I)) \to I, & 1, & p(s(J)) \to J) & (s(J),\ s(J)) \\
(p(s(I)) \to I, & 1, & s(p(J)) \to J) & (p(J),\ p(J)) \\
(p(I) + I' \to p(I + I'), & 1, & p(s(J)) \to J) & (p(s(J) + I'),\ J + I') \\
(s(I) + I' \to s(I + I'), & 1, & s(p(J)) \to J) & (s(p(J) + I'),\ J + I') \\
(p(I) \le I' \to I \le s(I'), & 1, & p(s(J)) \to J) & (s(J) \le s(I'),\ J \le I') \\
(s(I) \le I' \to I \le p(I'), & 1, & s(p(J)) \to J) & (p(J) \le p(I'),\ J \le I').
\end{array}$$

The critical pairs to the right of the overlaps all converge and are complete for $\mathcal{R}$.

Example 3.14. Consider the following compatible rewriting system $\mathcal{R}$:

$$\begin{array}{l}
\mathbf{A} < \mathbf{B}, \quad a\colon \to \mathbf{A}, \quad b\colon \to \mathbf{B}, \quad f\colon \mathbf{A} \to \mathbf{A}, \quad f\colon \mathbf{B} \to \mathbf{B} \\
a \to b, \quad f(x_{\mathbf{A}}) \to x_{\mathbf{A}}.
\end{array}$$

Since $\mathcal{R}$ doesn't have an overlap, every $\mathcal{R}$-critical pair converges. However, $\mathcal{R}$ is not locally confluent since $f(a) \to_{\mathcal{R}} a$, $f(a) \to_{\mathcal{R}} f(b)$, and a and $f(b)$ do not converge.

Fortunately, there is a large class of order-sorted rewriting systems for which a critical pair theorem holds. We say that a rewriting system $\mathcal{R}$ is **sort decreasing** if, for every $\mathcal{R}$-term s of sort ξ, $s \to_{\mathcal{R}} t$ implies that t is an $\mathcal{R}$-term of sort ξ. The rewriting system in the preceding example is not sort decreasing since, for instance, a has sort $\mathbf{A}$, $a \to_{\mathcal{R}} b$ and b has not sort $\mathbf{A}$.

Proposition 3.15. *Every sort decreasing rewriting system is compatible.*

Theorem 3.16. (Critical Pairs) *Let $\mathcal{R}$ be a sort decreasing rewriting system. Then $\mathcal{R}$ is locally confluent if and only if all critical pairs of $\mathcal{R}$ converge.*

Proof. The proof is identical to the proof of Lemma 3.1 in [Huet 80], where the sort decreasingness of $\mathcal{R}$ validates the used arguments. □

The rewriting system in Figure 1.1 is sort decreasing. However, if we add the declaration p: **posint** $\rightarrow$ **nat**, which is valid in the initial algebra, we obtain a system that isn't sort decreasing. The trouble is caused by the instance $p(x_{\mathbf{posint}})+y_{\mathbf{posint}} \rightarrow p(x_{\mathbf{posint}}+y_{\mathbf{posint}})$ of the rule $p(I)+I' \rightarrow p(I+I')$ and cannot be avoided by adding further declarations that are valid in the initial algebra.

Next we outline a procedure for deciding whether a finite rewriting system is sort decreasing.

A Σ-rewrite rule $s \rightarrow t$ is **sort decreasing** if, for every Σ-substitution θ and every Σ-sort symbol ξ, θt has sort ξ if θs has sort ξ.

Proposition 3.17. *A rewriting system is sort decreasing if and only if each of its rules is sort decreasing.*

Let V be a set of Σ-variables. A **sort assignment for** V is a function τ mapping V to the sort symbols of Σ such that $\tau(x) \leq \sigma x$ for all $x \in V$.

Proposition 3.18. *Let Σ be a finite signature. Then there exist only finitely many sort assignments for a set V of Σ-variables.*

Let τ be a sort assignment for a set V of Σ-variables. A **τ-weakening** is a Σ-substitution θ such that, for all $x \in V$, θx is a variable and $\sigma(\theta x) = \tau(x)$.

Proposition 3.19. *Let τ be a sort assignment for a set V of Σ-variables and let θ and θ' be τ-weakenings. Then*

$$\theta s \text{ has sort } \xi \quad \Longleftrightarrow \quad \theta' s \text{ has sort } \xi$$

for every (Σ, V)-term s and every Σ-sort symbol ξ.

Proposition 3.20. *A (Σ, V)-rewrite rule $s \rightarrow t$ is sort decreasing if and only if*

$$\theta s \text{ has sort } \xi \quad \Rightarrow \quad \theta t \text{ has sort } \xi$$

for every Σ-sort symbol ξ, every sort assignment τ for V, and some τ-weakening θ.

Proposition 3.21. *It is decidable whether a finite rewriting system is sort decreasing.*

A Knuth-Bendix completion procedure for order-sorted rewriting systems [Gnaedig et al. 87] has to orient equations with respect to termination and sort decreasingness. This adds an additional difficulty since an orientation that respects the termination order might not be sort decreasing.

3.4 Optimizing Sort Tests

An interpreter for a sort decreasing rewriting system works like an interpreter for an unsorted rewriting system, except that it has to test that a term s has sort σx before it can bind the variable x to s. In general, such a sort test can be rather expensive since a complete bottom up inspection of s might be necessary. Hence the optimization of sort tests is important for practical applications. Here we discuss two straightforward optimizations.

Let Σ be a signature. A Σ-sort symbol ξ is **singular in** Σ if for every Σ-function symbol f Σ contains at most one declaration $f{:}\, \eta_1 \cdots \eta_n \to \eta$ such that $\eta \leq \xi$.

Proposition 3.22. *Let ξ be a singular sort symbol in Σ. Then a Σ-term $f(s_1, \ldots, s_n)$ has sort ξ if and only if Σ contains a declaration $f{:}\, \eta_1 \cdots \eta_n \to \eta$ such that $\eta \leq \xi$.*

A variable $x \in \mathcal{V}(s)$ is **most general in a Σ-term** s if

$$t = \theta s \quad \Rightarrow \quad \theta x \text{ has sort } \sigma x$$

for every Σ-term t and every substitution θ (not necessarily a Σ-substitution).

Proposition 3.23. *If the sort of a variable is maximal, then it is most general in every term.*

To apply a rewrite rule $s \rightarrow t$, it suffices to have a sort test only for those variables that aren't most general in s.

In the rewriting system in Figure 1.1 all variables that aren't most general have singular sorts.

3.5 Remarks and References

Goguen et al. [85] give an operational semantics for order-sorted equational logic by compiling order-sorted rewriting systems into many-sorted rewriting systems without subsorts. This approach requires that the order-sorted rewriting system is sort decreasing and regular (regularity will be defined in the next section) and also handles sort constraints. Meseguer and Goguen [85b] study many-sorted rewriting without subsorts. Gogolla [86] gives a soundness and completeness theorem for sort decreasing rewriting systems. Schmidt-Schauß [87] studies rewriting in an order-sorted logic with term declarations.

Kirchner et al. [87] study order-sorted rewriting modulo equations and give criteria for soundness and completeness. They devise a method for the efficient implementation of sort tests and develop conditions for the separate compilation of rewrite rules in different modules. Their approach is the theoretical foundation for the implementation of OBJ3. Gnaedig et al. [87] study completion procedures for order-sorted rewriting systems.

Cunningham and Dick [85] investigate rewriting methods in a lattice of sorts that is a special case of regular signatures. They don't give a model theoretic semantics for their systems. Since they don't consider any equivalent of compatibility or sort decreasingness, their approach is at least incomplete.

4 Order-Sorted Unification

Term unification in order-sorted signatures is quite different from unification in unsorted signatures. In fact, unification in order-sorted signatures has many properties in common with unsorted unification modulo equations [Fages/Huet 86, Siekmann 86]. There are pathological order-sorted signatures in which infinitely many most general unifiers are needed to represent the unifiers of two terms. In regular signatures, a class that excludes pathological cases, finitely many most general unifiers suffice to represent the unifiers of two terms. Even in signatures in which the unifiers of two terms can be represented by a single most general unifier, the most general unifier may necessarily involve auxiliary variables.

4.1 Regularity

A signature Σ is called **regular** if

1. the subsort order of Σ is a partial order
2. every Σ-term s has a least sort σs, that is, there is a unique function σ from the set of all Σ-terms into the set of sort symbols such that

   2.1 if s is a Σ-term, then s is a term of sort σs

   2.2 if s is a Σ-term of sort ξ, then $\sigma s \leq \xi$.

The requirement that the subsort order is a partial order eases the notation but is not really essential. If Σ is a regular signature, we use σs to denote the least sort of a Σ-term s. We call a specification or a rewriting system **regular** if its signature is regular.

The signatures of all examples discussed so far are regular. An example for a nonregular signature is

$$\{a: \rightarrow \mathbf{A}, \quad a: \rightarrow \mathbf{B}\}.$$

It seems that there are no natural examples of nonregular signatures. The class of regular signatures is important since unification in nonregular signatures is infinitary, while unification in regular signatures is finitary.

Theorem 4.1. *A signature Σ whose subsort order is anti-symmetric is regular if and only if for every Σ-function symbol f and every string $\vec{\xi}$ of Σ-sort symbols the set $\{\eta \mid (f\colon \vec{\eta} \to \eta) \in \Sigma$ and $\vec{\xi} \leq_\Sigma \vec{\eta}\}$ is either empty or has a minimum with respect to the subsort order of Σ.*

Proof. *1.* Let Σ be a regular signature, f be a Σ-function symbol, and $\xi_1, \ldots, \xi_n$ be Σ-sort symbols such that Σ contains a declaration $f\colon \eta_1 \cdots \eta_n \to \eta$ such that $\xi_i \leq \eta_i$ for $i = 1, \ldots, n$. Furthermore, let $x_1, \ldots, x_n$ be variables such that $\sigma x_i = \xi_i$ for $i = 1, \ldots, n$. Then $f(x_1, \ldots, x_n)$ is a Σ-term. Since Σ is regular, Σ contains a declaration $f\colon \zeta_1 \cdots \zeta_n \to \zeta$ such that $\xi_i \leq \zeta_i$ for $i = 1, \ldots, n$ and $\zeta = \sigma f(x_1, \ldots, x_n)$, where $\sigma f(x_1, \ldots, x_n)$ is the least sort of $f(x_1, \ldots, x_n)$ in Σ. Hence $\zeta = \min\{\eta \mid (f\colon \vec{\eta} \to \eta) \in \Sigma \text{ and } \xi_1 \cdots \xi_n \leq \vec{\eta}\}$.

2. Let Σ be a signature whose subsort order is a partial order such that for every function symbol f and every string $\vec{\xi}$ the set $\{\eta \mid (f\colon \vec{\eta} \to \eta) \in \Sigma$ and $\vec{\xi} \leq \vec{\eta}\}$ is either empty or has a minimum. Then its easy to verify that

$$\begin{aligned} \tau(x) &:= \sigma x \\ \tau(f(s_1, \ldots, s_n)) &:= \min\{\eta \mid (f\colon \vec{\eta} \to \eta) \in \Sigma \text{ and } \tau(s_1) \cdots \tau(s_n) \leq \vec{\eta}\} \end{aligned}$$

defines a unique least sort function τ on the set of all Σ-terms. ☐

This theorem is also proved in [Goguen/Meseguer 87c], where regularity is called preregularity and regularity is defined as a slightly stronger condition.

Corollary 4.2. *Regularity of finite signatures is decidable.*

Corollary 4.3. *Every signature without multiple function declarations is regular.*

Proposition 4.4. *Let $\mathcal{R}$ be a sort decreasing, confluent and regular rewriting system. Then*

$$\sigma(s\downarrow_{\mathcal{R}}) = \min\{\sigma t \mid s =_{\mathcal{R}} t\}$$

for every $\mathcal{R}$-term s having an $\mathcal{R}$-normal form $s\downarrow_{\mathcal{R}}$.

Proposition 4.5. *Let $\mathcal{R}$ be a sort decreasing, confluent and regular rewriting system and let $\mathcal{I}_{\mathcal{R}}$ be the initial algebra of $\mathcal{R}$. Then:*

- *if s is a ground $\mathcal{R}$-term with the $\mathcal{R}$-normal form $s\downarrow_{\mathcal{R}}$, then $[\![s]\!]_{\mathcal{I}_{\mathcal{R}}} \in \xi^{\mathcal{I}_{\mathcal{R}}}$ if and only if $\sigma(s\downarrow_{\mathcal{R}}) \leq \xi$*
- *if $a \in \xi^{\mathcal{I}_{\mathcal{R}}} \cap \eta^{\mathcal{I}_{\mathcal{R}}}$, then there exists a common subsort ζ of ξ and η such that $a \in \zeta^{\mathcal{I}_{\mathcal{R}}}$.*

4.2 Basic Definitions and Counterexamples

A **Σ-equation system** is either the **empty equation system** $\emptyset$ or has the form $s_1 \doteq t_1 \ \& \ \cdots \ \& \ s_n \doteq t_n$, where $s_i \doteq t_i$ is a Σ-equation for $i = 1, \ldots, n$. For convenience, we assume that the conjunction operator $\&$ is associative, commutative and satisfies $\emptyset \ \& \ E = E$ for every equation system E. For instance, $(a \doteq b \ \& \ a \doteq c)$ and $(\emptyset \ \& \ a \doteq c \ \& \ \emptyset \ \& \ a \doteq b)$ denote the same equation system. An equation $s \doteq t$ is **trivial** if $s = t$. An equation system is **trivial** if each of its equations is trivial. The letter E will always range over equation systems.

The **Σ-unifiers** of a Σ-equation system E are

$$\mathrm{U}_{\Sigma}(E) := \{\theta \in \mathrm{SUB}_{\Sigma} \mid \theta E \text{ is trivial}\},$$

where SUB_{Σ} is the set of all Σ-substitutions. A Σ-equation system is called **Σ-unifiable** if it has at least one Σ-unifier.

Let θ be a Σ-substitution. Then $\mathcal{C}\theta := \{\theta x \mid x \in \mathcal{D}\theta\}$ is called the **codomain of θ** and $\mathcal{I}\theta := \mathcal{V}(\mathcal{C}\theta)$ is called the set of variables introduced by θ. A substitution θ is called **idempotent** if $\theta\theta = \theta$. Note that θ is idempotent if and only if $\mathcal{D}\theta$ and $\mathcal{I}\theta$ are disjoint.

Let θ be a Σ-substitution. The **equational representation of θ** is the Σ-equation system

$$[\theta] := (x_1 \doteq \theta x_1 \ \& \ \cdots \ \& \ x_n \doteq \theta x_n),$$

where $\{x_1, \ldots, x_n\} = \mathcal{D}\theta$. Two Σ-substitutions are equal if and only if their equational representations are equal.

Proposition 4.6. *Let θ and ψ be Σ-substitutions. If θ is idempotent, then*

$$\psi \in \mathrm{U}_\Sigma([\theta]) \iff \psi[\theta] \textit{ is trivial} \iff (\Sigma, \emptyset) \models \psi[\theta]$$
$$\iff \psi = \psi\theta \iff \exists\, \phi \in \mathrm{SUB}_\Sigma.\ \psi = \phi\theta$$

and $\mathrm{U}_\Sigma([\theta]) = \mathrm{SUB}_\Sigma \cdot \theta := \{\psi\theta \mid \psi \in \mathrm{SUB}_\Sigma\}$.

Let Σ be a signature. The **unsorted signature corresponding to** Σ is

$$\bar{\Sigma} := \Sigma \cup \{(\xi < \eta) \mid \xi \text{ and } \eta \text{ are } \Sigma\text{-sort symbols}\}.$$

Every Σ-term, equation or equation system is a $\bar{\Sigma}$-term, equation or equation system, respectively.

Proposition 4.7. *Let E be a Σ-equation system. Then*

$$\mathrm{U}_\Sigma(E) = \mathrm{U}_{\bar{\Sigma}}(E) \cap \mathrm{SUB}_\Sigma.$$

This proposition says that the order-sorted unifiers of E are exactly the unsorted unifiers of E that are order-sorted substitutions.

Theorem 4.8. (Unsorted Unification) *Let Σ be a signature and E be a $\bar{\Sigma}$-equation system that is $\bar{\Sigma}$-unifiable. Then there exists an idempotent $\bar{\Sigma}$-substitution θ such that $\mathrm{U}_{\bar{\Sigma}}(E) = \mathrm{U}_{\bar{\Sigma}}([\theta])$ and $\mathcal{V}([\theta]) \subseteq \mathcal{V}(E)$.*

This theorem is just a reformulation of Robinson's [65] unification theorem. A substitution θ such that $\mathrm{U}_{\bar{\Sigma}}(E) = \mathrm{U}_{\bar{\Sigma}}([\theta])$ is usually called a **most general unifier** of E.

Let V be a set of Σ-variables. The (Σ, V)**-unifiers** of a Σ-equation system E are

$$\mathrm{U}_\Sigma^V(E) := \{\theta|_V \mid \theta \in \mathrm{SUB}_\Sigma \ \wedge\ \theta E \text{ is trivial}\}.$$

where the restriction of θ to V is the substitution $\theta|_V$ satisfying

$$\theta|_V(x) = \begin{cases} \theta x & \text{if } x \in V \\ x & \text{otherwise.} \end{cases}$$

A signature Σ is called

- **unitary unifying** if for every Σ-unifiable Σ-equation system E there exists an idempotent Σ-substitution θ such that $\mathrm{U}_\Sigma^{\mathcal{V}(E)}(E) = \mathrm{U}_\Sigma^{\mathcal{V}(E)}([\theta])$
- **finitary unifying** if for every Σ-unifiable Σ-equation system E there exist idempotent Σ-substitutions $\theta_1, \ldots, \theta_n$ such that

$$\mathrm{U}_\Sigma^{\mathcal{V}(E)}(E) = \mathrm{U}_\Sigma^{\mathcal{V}(E)}([\theta_1]) \cup \cdots \cup \mathrm{U}_\Sigma^{\mathcal{V}(E)}([\theta_n]).$$

In practice, (Σ, V)-unifiers suffice and Σ-unifiers are not really needed. This is very fortunate since the above definition of unitary unifying applies to many more order-sorted signatures than an analogous definition requiring $\mathrm{U}_\Sigma(E) = \mathrm{U}_\Sigma([\theta])$. The need to restrict unifiers to a set of "interesting" variables also exists for unsorted unification modulo equations [Fages/Huet 86].

Example 4.9. Let Σ be the nonregular signature

$$o{:} \to \mathbf{A}, \quad o{:} \to \mathbf{B}, \quad s{:}\, \mathbf{A} \to \mathbf{A}, \quad s{:}\, \mathbf{B} \to \mathbf{B}.$$

The Σ-equation $x_\mathbf{A} = y_\mathbf{B}$ has the infinitely many Σ-unifiers with the equational representations

$$\begin{gathered} (x_\mathbf{A} \doteq o \;\; \& \;\; x_\mathbf{B} \doteq o) \\ (x_\mathbf{A} \doteq s(o) \;\; \& \;\; x_\mathbf{B} \doteq s(o)) \\ (x_\mathbf{A} \doteq s(s(o)) \;\; \& \;\; x_\mathbf{B} \doteq s(s(o))) \\ \cdots . \end{gathered}$$

Since the terms $o, s(o), \ldots$ don't have a least sort, $\mathrm{U}_\Sigma^{\{x_\mathbf{A}, y_\mathbf{B}\}}(x_\mathbf{A} = y_\mathbf{B})$ cannot be represented by finitely many idempotent substitutions. Hence Σ is not finitary unifying.

Proposition 4.10. *There is a finite nonregular signature that is not finitary unifying.*

Example 4.11. Let Σ be the regular signature of the example in Figure 1.1. Then

$$\begin{aligned}\mathrm{U}_\Sigma^V(x_{\mathbf{posint}} \doteq y_{\mathbf{nat}} + z_{\mathbf{nat}}) = \\ &\mathrm{U}_\Sigma^V(x_{\mathbf{posint}} \doteq y'_{\mathbf{posint}} + z_{\mathbf{nat}} \mathbin{\&} y_{\mathbf{nat}} \doteq y'_{\mathbf{posint}}) \\ &\cup \mathrm{U}_\Sigma^V(x_{\mathbf{posint}} \doteq y_{\mathbf{nat}} + z'_{\mathbf{posint}} \mathbin{\&} z_{\mathbf{nat}} \doteq z'_{\mathbf{posint}})\end{aligned}$$

for every set V of Σ-variables not containing the auxiliary variables $y'_{\mathbf{posint}}$ and $z'_{\mathbf{posint}}$. Obviously, there exists no idempotent Σ-substitution θ such that $\mathrm{U}_\Sigma^V(x_{\mathbf{posint}} \doteq y_{\mathbf{nat}} + z_{\mathbf{nat}}) = \mathrm{U}_\Sigma^V([\theta])$ for $V = \{x_{\mathbf{posint}}, y_{\mathbf{nat}}, z_{\mathbf{nat}}\}$.

Proposition 4.12. *There exists a finite regular signature with multiple function declarations that is not unitary unifying.*

Example 4.13. This example demonstrates that, for order-sorted unification, it is crucial to consider (Σ, V)-unifiers rather than Σ-unifiers. Let Σ be the regular signature

$$\mathbf{AB} < \mathbf{A}, \quad \mathbf{AB} < \mathbf{B}, \quad a\colon \to \mathbf{AB}.$$

Then

$$\mathrm{U}_\Sigma^V(x_{\mathbf{A}} \doteq y_{\mathbf{B}}) = \mathrm{U}_\Sigma^V(x_{\mathbf{A}} \doteq z_{\mathbf{AB}} \mathbin{\&} y_{\mathbf{B}} \doteq z_{\mathbf{AB}})$$

for every set V of Σ-variables not containing the auxiliary variable $z_{\mathbf{AB}}$. However,

$$\mathrm{U}_\Sigma(x_{\mathbf{A}} \doteq y_{\mathbf{B}}) \neq \mathrm{U}_\Sigma(x_{\mathbf{A}} \doteq z_{\mathbf{AB}} \mathbin{\&} y_{\mathbf{B}} \doteq z_{\mathbf{AB}})$$

since, for instance, $(x_{\mathbf{A}} \doteq a \mathbin{\&} y_{\mathbf{B}} \doteq a)$ is a Σ-unifier of $x_{\mathbf{A}} \doteq y_{\mathbf{B}}$ but not of $(x_{\mathbf{A}} \doteq z_{\mathbf{AB}} \mathbin{\&} y_{\mathbf{B}} \doteq z_{\mathbf{AB}})$. Obviously, there exists no idempotent Σ-substitution θ such that $\mathrm{U}_\Sigma(x_{\mathbf{A}} \doteq y_{\mathbf{B}}) = \mathrm{U}_\Sigma([\theta])$. Nevertheless, as follows from a general result to be proved later, Σ is unitary unifying.

4.3 Computing Order-Sorted Unifiers from Unsorted Unifiers

Given a signature Σ and a Σ-equation system E, we can compute with an unsorted unification algorithm an idempotent $\bar{\Sigma}$-substitution θ such that

$$\mathrm{U}_\Sigma(E) = \mathrm{U}_{\bar{\Sigma}}([\theta]) \cap \mathrm{SUB}_\Sigma.$$

We will now show how one can compute finitely many idempotent Σ-subsitutions $\theta_1, \ldots, \theta_n$ from the unsorted most general unifier θ such that

$$\mathrm{U}_\Sigma^{\mathcal{V}(E)}(E) = \mathrm{U}_\Sigma^{\mathcal{V}(E)}([\theta_1]) \cup \cdots \cup \mathrm{U}_\Sigma^{\mathcal{V}(E)}([\theta_n]),$$

provided that Σ is finite and regular.

A **Σ-containment** $s{:}\,\xi$ is a pair consisting of a $\bar{\Sigma}$-term s and a Σ-sort symbol ξ. A **Σ-containment system** is a possibly empty, finite bag of Σ-containments. A **disjunctive Σ-containment system** is a possibly empty, finite bag of Σ-containment systems.

A Σ-substitution θ **satisfies a Σ-containment** $s{:}\,\xi$ if θs is a Σ-term of sort ξ. A Σ-substitution **satisfies a Σ-containment system** if it satisfies each of its containments. A Σ-substitution **satisfies a disjunctive Σ-containment system** if it satisfies at least one of its containment systems.

Lemma 4.14. *Let Σ be a signature and θ be an idempotent $\bar{\Sigma}$-substitution. Then*

$$\mathrm{U}_{\bar{\Sigma}}([\theta]) \cap \mathrm{SUB}_\Sigma = \{\psi\theta \mid \psi \in \mathrm{SUB}_\Sigma \text{ and } \psi \text{ satisfies } \{\theta x{:}\,\sigma x \mid x \in \mathcal{D}\theta\}\}.$$

Proof. "$\subseteq$". Let ϕ be a Σ-substitution such that $\phi[\theta]$ is trivial. Then $\phi = \phi\theta$. Hence $\phi\theta x$ is a Σ-term of sort σx for every $x \in \mathcal{D}\theta$. Thus ϕ satisfies the containment system $\{\theta x{:}\,\sigma x \mid x \in \mathcal{D}\theta\}$.

"$\supseteq$". Let ψ be a Σ-substitution satisfying $\{\theta x{:}\,\sigma x \mid x \in \mathcal{D}\theta\}$. Then $\psi\theta x$ is a Σ-term of sort σx for all $x \in \mathcal{D}\theta$. Hence $\psi\theta$ is a Σ-substitution. Furthermore, $\psi\theta = \psi\theta\theta$ since θ is idempotent. Thus $\psi\theta \in \mathrm{U}_{\bar{\Sigma}}([\theta])$. $\square$

A Σ-containment $x\colon\xi$ is **solved** if its left-hand side is a variable and $\xi \leq_\Sigma \sigma x$. A containment system is **solved** if each of its containments is solved and no variable occurs in more than one of its containments. A disjunctive containment system is **solved** if each of its containment systems is solved.

Let Σ be a finite and regular signature and let maxel_Σ be the function that yields the maximal elements (with respect to the subsort order of Σ) of a set of Σ-sort symbols or strings of Σ-sort symbols. We define the following reduction rules for disjunctive Σ-containment systems:

(1) $D \cup \{\{x\colon\xi\} \cup C\} \quad \to_\Sigma \quad D \cup \{\{x\colon\zeta\} \cup C \mid \zeta \in X\}$

if $\xi \not\leq_\Sigma \sigma x$ and
$X := \mathrm{maxel}_\Sigma\{\zeta \mid \zeta \leq_\Sigma \sigma x \text{ and } \zeta \leq_\Sigma \xi\}$ is nonempty

(2) $D \cup \{\{x\colon\xi, x\colon\eta\} \cup C\} \quad \to_\Sigma \quad D \cup \{\{x\colon\zeta\} \cup C \mid \zeta \in X\}$

if $X := \mathrm{maxel}_\Sigma\{\zeta \mid \zeta \leq_\Sigma \xi \text{ and } \zeta \leq_\Sigma \eta\}$ is nonempty

(3) $D \cup \{\{f(s_1, \ldots, s_n)\colon\xi\} \cup C\} \quad \to_\Sigma$
$D \cup \{\{s_1\colon\eta_1, \ldots, s_n\colon\eta_n\} \cup C \mid (\eta_1, \ldots, \eta_n) \in X\}$

if $X := \mathrm{maxel}_\Sigma\{(\eta_1, \ldots, \eta_n) \mid (f\colon\eta_1 \cdots \eta_n \to \eta) \in \Sigma \text{ and } \eta \leq_\Sigma \xi\}$
is nonempty

(4) $D \quad \to_\Sigma \quad \{\emptyset\} \qquad \text{if } \emptyset \in D \neq \{\emptyset\}$

(5) $D \cup \{C\} \quad \to_\Sigma \quad D$

if C is not solved and none of the rules above apply to C

Proposition 4.15. *Let Σ be a finite and regular signature. Then:*

- *there are no infinite chains $D_1 \to_\Sigma D_2 \to_\Sigma \cdots$ issuing from a disjunctive Σ-containment system D_1*

- *if D is a disjunctive Σ-containment system and $D \to_\Sigma D'$, then D' is a disjunctive Σ-containment system and a Σ-substitution satisfies D if and only if it satisfies D'*

- *for every disjunctive Σ-containment system D there exists a solved disjunctive Σ-containment system D' such that $D \rightarrow_{\Sigma}^{*} D'$.*

The proposition says that the reduction rules for disjunctive containment systems constitute a solution algorithm.

Proposition 4.16. *Let Σ be a finite and regular signature and let θ be an idempotent $\bar{\Sigma}$-substitution. Then*

$$\mathrm{U}_{\bar{\Sigma}}\left([\theta]\right) \cap \mathrm{SUB}_{\Sigma} = \bigcup_{C \in D} \{\psi\theta \mid \psi \text{ is a } \Sigma\text{-substitution satisfying } C\}$$

if D is disjunctive containment system such that $\{\{\theta x\colon \sigma x \mid x \in \mathcal{D}\theta\}\} \rightarrow_{\Sigma}^{*} D$.

Let Σ be a regular signature, C be a solved Σ-containment system, and V be a set of Σ-variables. A **weakening for C away from V** is a Σ-substitution ω such that

- $\mathcal{D}\omega \subseteq \mathcal{V}(C)$ and ω is injective on $\mathcal{D}\omega$
- if $x \in \mathcal{D}\omega$, then $\omega(x)$ is a variable not contained in V
- if $(x\colon \xi) \in C$, then $\sigma(\omega x) = \xi$.

Proposition 4.17. *Let Σ be a regular signature, C be a solved Σ-containment system, and V be a finite set of Σ-variables. Then there exists a weakening for C away from V.*

The following theorem is closely related to theorems in [Schmidt-Schauß 85a] and [Meseguer et al. 87].

Theorem 4.18. (Order-Sorted Unification) *Let Σ be a finite and regular signature, E be a Σ-unifiable Σ-equation system, and V be a finite set of Σ-variables such that $\mathcal{V}(E) \subseteq V$. Then there exists an idempotent $\bar{\Sigma}$-substitution θ, which can be computed by an unsorted unification algorithm, such that $\mathcal{V}([\theta]) \subseteq \mathcal{V}(E)$ and*

$$\mathrm{U}_{\Sigma}^{V}(E) = \mathrm{U}_{\bar{\Sigma}}^{V}([\theta]) \cap \mathrm{SUB}_{\Sigma}.$$

Furthermore, one can compute solved Σ-containment systems $C_1, \ldots, C_n$, $n \geq 1$, such that

$$\{\{\theta x : \sigma x \mid x \in \mathcal{D}\theta\}\} \to_\Sigma^* \{C_1, \ldots, C_n\}.$$

Then

$$\mathrm{U}_\Sigma^V(E) = \bigcup_{i=1}^{n} \mathrm{U}_\Sigma^V([\omega_i\theta]) \text{ and } \mathcal{D}(\omega_i\theta) \subseteq \mathcal{V}(E) \text{ for } i = 1, \ldots, n$$

if ω_i is a weakening for C_i away from V for $i = 1, \ldots, n$.

Proof. The first claim follows from the Unsorted Unification Theorem and Proposition 4.7. The second claim follows from Proposition 4.15. To show the third claim, let $C_1, \ldots, C_n$ be solved Σ-containment systems such that $\{\{\theta x : \sigma x \mid x \in \mathcal{D}\theta\}\} \to_\Sigma^* \{C_1, \ldots, C_n\}$, and let ω_i be a weakening for C_i away from V for $i = 1, \ldots, n$.

"$\subseteq$". Let $\psi \in \mathrm{U}_\Sigma^V(E)$. Then ψ is a Σ-substitution such that $\psi = \psi\theta$. By Lemma 4.14 we know that ψ satisfies $\{\{\theta x : \sigma x \mid x \in \mathcal{D}\theta\}\}$. Hence ψ satisfies some C_j. Thus the following defines a Σ-substitution ϕ

$$\phi x := \begin{cases} \psi y & \text{if } x \in \mathcal{I}\omega_j \text{ and } \omega_j y = x \\ \psi x & \text{otherwise.} \end{cases}$$

since, if $\omega_j y = x$ and $x \in \mathcal{I}\omega_j$, there exists a containment $(y : \eta) \in C_j$ such that $\sigma(\psi y) \leq \eta = \sigma(\omega_j y) = \sigma x$. It's easy to verify that $\phi|_V = \psi = (\phi\omega_j)|_V$. To show that $\phi = \phi\omega_j\theta$, we distinguish two cases. If $x \notin V$, then $\phi\omega_j\theta x = \phi x$ since $x \notin \mathcal{D}\theta$ and $x \notin \mathcal{D}\omega_j$. If $x \in V$, then $\phi\omega_j\theta x = \psi\theta x = \psi x = \phi x$ since $\mathcal{V}(\theta x) \subseteq V$, $(\phi\omega_j)|_V = \psi$, $\psi\theta = \psi$, and $\psi = \phi|_V$. Hence $\psi = \phi|_V \in \mathrm{U}_\Sigma^V([\omega_j\theta])$.

"$\supseteq$". Let ψ be a Σ-substitution such that $\psi = \psi\omega_j\theta$ for some j. Then $\psi\theta = \psi\omega_j\theta\theta = \psi\omega_j\theta = \psi$ since θ is idempotent. Hence $\psi|_V \in \mathrm{U}_\Sigma^V([\theta]) \cap \mathrm{SUB}_\Sigma = \mathrm{U}_\Sigma^V(E)$. □

Corollary 4.19. *Every finite and regular signature is finitary unifying.*

The following theorem was first proved by Schmidt-Schauß [87].

Theorem 4.20. *For finite and regular signatures, deciding whether a containment $s{:}\,\xi$ is satisfiable is an NP-complete problem.*

Proof. Let Σ be the regular signature

$$\mathbf{T} < \mathbf{bool}, \quad \mathbf{F} < \mathbf{bool},$$

$$\mathit{and}{:}\,\mathbf{T} \times \mathbf{T} \to \mathbf{T}, \quad \mathit{and}{:}\,\mathbf{F} \times \mathbf{bool} \to \mathbf{F}, \quad \mathit{and}{:}\,\mathbf{bool} \times \mathbf{F} \to \mathbf{F},$$

$$\mathit{or}{:}\,\mathbf{T} \times \mathbf{bool} \to \mathbf{T}, \quad \mathit{or}{:}\,\mathbf{bool} \times \mathbf{T} \to \mathbf{T}, \quad \mathit{or}{:}\,\mathbf{F} \times \mathbf{F} \to \mathbf{F},$$

$$\mathit{not}{:}\,\mathbf{T} \to \mathbf{F}, \quad \mathit{not}{:}\,\mathbf{F} \to \mathbf{T}.$$

Obviously, every $\bar{\Sigma}$-term represents a boolean formula, where variables of sort $\mathbf{T}$ represent the truth value true, variables of sort $\mathbf{F}$ represent the truth value false, and variables of sort **bool** represent boolean variables. Vice versa, every boolean formula can be represented as a $\bar{\Sigma}$-term. Hence, deciding whether a Σ-containment $s{:}\,\mathbf{T}$ is satisfiable is equivalent to deciding whether the boolean formula represented by s is satisfiable. □

Since unsorted unification has linear complexity [Paterson/Wegman 78], we have:

Corollary 4.21. *For finite and regular signatures, deciding whether an equation is unifiable is an NP-complete problem.*

A signature Σ is called **coregular** if, for every Σ-function symbol f and every Σ-sort symbol ξ, the set

$$\mathrm{maxel}_\Sigma\{\eta_1 \cdots \eta_n \mid (f{:}\,\eta_1 \cdots \eta_n \to \eta) \in \Sigma \;\;\wedge\;\; \eta \le \xi\}$$

has at most one element.

Proposition 4.22. *Every signature without multiple function declarations is coregular.*

A signature Σ is called **downward complete** if, for every two Σ-sort symbols ξ and η, the set $\mathrm{maxel}_\Sigma\{\zeta \mid \zeta \le \xi \;\;\wedge\;\; \zeta \le \eta\}$ of all common subsorts of ξ and η has at most one element.

Proposition 4.23. *Let Σ be a finite, regular, coregular and downward complete signature and let D be a disjunctive Σ-containment system. Then there exists a solved Σ-containment system C such that $D \rightarrow_\Sigma^* \{C\}$.*

The next two theorems were first proven in [Meseguer et al. 87].

Theorem 4.24. *Every finite, regular, coregular and downward complete signature is unitary unifying.*

Proof. Follows immediately from the Order-Sorted Unification Theorem and the preceding proposition. □

Theorem 4.25. *Unification in finite, regular, coregular and downward complete signatures has quasi-linear complexity.*

This theorem is proved in [Meseguer et al. 87] by giving an extension of Martelli and Montanari's [82] quasi-linear unification algorithm to order-sorted signatures with the listed properties. Note that in signatures with the listed properties the solved disjunctive containment system of the unsorted unifier can be computed in time linear to the size of the unsorted unifier.

4.4 Computing Complete Sets of Critical Pairs

We now outline how an order-sorted Knuth-Bendix completion procedure can compute finite and complete sets of critical pairs.

Proposition 4.26. *Let $\mathcal{R} = (\Sigma, \mathcal{E})$ be a rewriting system, $s \rightarrow t$ and $u \rightarrow v$ be rules of $\mathcal{R}$, π be a position of s such that s/π is not a variable and $s \rightarrow t$ is not a variant of $u \rightarrow v$ if $s/\pi = s$. Furthermore, let $u' \rightarrow v'$ be a variant of $u \rightarrow v$ such that s and u' don't have variables in common. Then $(s \rightarrow t, \pi, u' \rightarrow v')$ is an overlap of $\mathcal{R}$ if and only if $s/\pi = u'$ is Σ-unifiable.*

Proposition 4.27. *Let $(s \rightarrow t, \pi, u \rightarrow v)$ be an overlap of a rewriting system $\mathcal{R} = (\Sigma, \mathcal{E})$ and let $\theta_1, \ldots, \theta_n$ be Σ-substitutions such that*

$$\mathrm{U}_\Sigma^V(s/\pi \doteq u) = \mathrm{U}_\Sigma^V([\theta_1]) \cup \cdots \cup \mathrm{U}_\Sigma^V([\theta_n]),$$

where $V = \mathcal{V}(s \doteq u)$. *Then* $\{\theta_i(t, s[\pi \leftarrow v])\}_{i=1}^n$ *is a complete set of critical pairs for the overlap* $(s \rightarrow t, \pi, u \rightarrow v)$.

Now the following two theorems are obvious.

Theorem 4.28. *For every regular and finite rewriting system* $\mathcal{R}$ *a finite, complete set of* $\mathcal{R}$*-critical pairs can be computed using an order-sorted unification algorithm.*

Theorem 4.29. *It is decidable whether a finite, regular, sort decreasing and terminating rewriting system is confluent.*

4.5 Remarks and References

Walther [83, 84, 85, 86, 87] was the first to investigate order-sorted unification. He studies signatures without multiple function declarations and gives unification algorithms for them. He proves that resolution and paramodulation with order-sorted unification are refutation complete for an order-sorted predicate logic with equality and without multiple function declarations. Walther [85] and others [Cohn 83 and 85, Irani/Shin 85] observe that the use of order-sorted unification can drastically reduce the search space of a resolution-based theorem prover.

Schmidt-Schauß [85a] extended Walther's work to multiple function declarations. He gave the first unification algorithm for signatures with multiple function declarations and proved that regular signatures are finitary unifying. Schmidt-Schauß [85b] also showed that in order-sorted signatures with term declarations the unifiability of two terms is undecidable. Furthermore, Schmidt-Schauß [86] studied order-sorted unification modulo equations and showed that, for a certain class of equational theories, the solution of the order-sorted problem can be obtained from the solution of the unsorted problem in the same way as in the absence of equations.

Meseguer et al. [87] give a categorical treatment of order-sorted unification modulo equations. They were the first to observe that unification in coregular and downward complete signatures is unitary. They extend the results of [Schmidt-Schauß 86] by a decidable sufficient condition for when order-sorted equational unification can be related to unsorted equational unification. For the case where order-sorted equational unification can be related to unsorted equational unification, they give characterization theorems for when order-sorted unification has unitary, finitary and minimal families of most general unifiers.

Feature Unification [Smolka/Aït-Kaci 87] combines order-sorted unification with ψ-term unification [Aït-Kaci 86, Aït-Kaci/Nasr 86] and is oriented towards applications in knowledge representation and computational linguistics.

5 Hierarchical Specifications and Partial Functions

For algebraic specification to be a practical tool, it is necessary to structure specifications. Hierarchical organization is about the most simple form of structuring specifications. For instance, a specification can be organized in two layers as follows: a *basic layer* defining a collection of data types and an *extending layer* defining functions on the basic types. While this hierarchical organization scheme is built-in in programming languages such as Pascal or ML, it isn't explicitly present in the equational specifications we have considered so far.

A **hierarchical specification** is a pair $(\mathcal{B}, \mathcal{S})$ consisting of two specifications $\mathcal{B}$, called the **basic specification**, and $\mathcal{S}$, called the **full specification**, such that $\mathcal{B} \subseteq \mathcal{S}$, that is, the signature of $\mathcal{B}$ is a subset of the signature of $\mathcal{S}$ and the axioms of $\mathcal{B}$ are a subset of the axioms of $\mathcal{S}$. Following [Ehrig/Mahr 85], we call a hierarchical specification $(\mathcal{B}, \mathcal{S})$

- **consistent** if every ground $\mathcal{B}$-equation is valid in $\mathcal{B}$ if and only if it is valid in $\mathcal{S}$
- **complete** if for every $\mathcal{B}$-sort symbol ξ and every ground $\mathcal{S}$-term s of sort ξ there exists a ground $\mathcal{B}$-term t of sort ξ such that $s =_{\mathcal{S}} t$
- **conservative** if it is consistent and complete.

Consistency means that the initial algebra of the extension doesn't *collapse* basic sorts, that is, doesn't identify elements of basic sorts. Completeness means that the initial algebra of the extension doesn't blow up basic sorts, that is, doesn't add new elements to basic sorts. Consistency and completeness of hierarchical specifications correspond to the no confusion and no junk requirements for initial algebras.

The specification in Figure 1.1 can be organized as a conservative hierarchical specification by considering $+$ and $\leq$ as extending functions.

Figure 5.1 provides some examples of inconsistent and incomplete hierarchical specifications. The specification $\mathcal{B}$ gives an equation-free definition of

$$\textit{variables}: \quad M, N\colon \mathbf{nat}$$

$$\mathcal{B} \begin{cases} \textit{true}\colon \to \mathbf{bool}, \quad \textit{false}\colon \to \mathbf{bool} \\ o\colon \to \mathbf{nat}, \quad s\colon \mathbf{nat} \to \mathbf{nat} \end{cases}$$

$$\mathcal{E}_1 \begin{cases} \textit{le}\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{bool} \\ \quad \textit{le}(o, N) \doteq \textit{true} \\ \quad \textit{le}(s(N), 0) \doteq \textit{false} \\ \quad \textit{le}(s(N), s(M)) \doteq \textit{le}(N, M) \\ \quad \textit{le}(N, N) \doteq \textit{true} \\ \quad \textit{le}(s(N), N) \doteq \textit{false} \end{cases}$$

$$\mathcal{E}_2 \begin{cases} \textit{foo}\colon \mathbf{nat} \to \mathbf{nat} \\ \quad \textit{foo}(N) \doteq s(\textit{foo}(N)) \end{cases}$$

Figure 5.1. Consistency and completeness of hierarchical specifications: $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_1)$ is consistent and complete, $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_2)$ is consistent but not complete, and $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2)$ is both incomplete and inconsistent.

bool and **nat** and serves as basic layer. The extension $\mathcal{E}_1$ defines a less or equal test on **nat**. The last two equations of *le* are actually redundant since, in the initial model, they are consequences of the first three equations; one may think of them as sound optimizations. The hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_1)$ is consistent and complete. The hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_2)$ is consistent but not complete: for instance, there is no $\mathcal{B}$-term that equals $\textit{foo}(o)$ in $\mathcal{B} \cup \mathcal{E}_2$. The hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2)$ is both incomplete and inconsistent. To see the inconsistency, verify that

$$\textit{true} \doteq \textit{le}(\textit{foo}(o), \textit{foo}(o)) \doteq \textit{le}(s(\textit{foo}(o)), \textit{foo}(o)) \doteq \textit{false}$$

is valid in $\mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2$. Since *true* and *false* are distinct normal forms of $\textit{le}(\textit{foo}(o), \textit{foo}(o))$, $\mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2$ is not a confluent rewriting system. However, $\mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2$ is a locally confluent rewriting system since it is sort decreasing and each of its critical pairs converges.

$$
\begin{array}{ll}
 & \textit{variables}: \quad M, N: \mathbf{nat} \\
\mathcal{B} & \left\{ \begin{array}{l} \textit{true}: \to \mathbf{bool}, \quad \textit{false}: \to \mathbf{bool} \\ o: \to \mathbf{nat}, \quad s: \mathbf{nat} \to \mathbf{nat} \end{array} \right. \\
\mathcal{E} & \left\{ \begin{array}{l} \mathbf{bool} < ?\mathbf{bool}, \quad \mathbf{nat} < ?\mathbf{nat} \\ s: ?\mathbf{nat} \to ?\mathbf{nat} \\ \\ \textit{le}: ?\mathbf{nat} \times ?\mathbf{nat} \to ?\mathbf{bool} \\ \quad \textit{le}(o, N) \doteq \textit{true} \\ \quad \textit{le}(s(N), 0) \doteq \textit{false} \\ \quad \textit{le}(s(N), s(M)) \doteq \textit{le}(N, M) \\ \quad \textit{le}(N, N) \doteq \textit{true} \\ \quad \textit{le}(s(N), N) \doteq \textit{false} \\ \\ \textit{foo}: ?\mathbf{nat} \to ?\mathbf{nat} \\ \quad \textit{foo}(N) \doteq s(\textit{foo}(N)) \end{array} \right.
\end{array}
$$

Figure 5.2. The stratification of the specification in Figure 5.1, where *le* and *foo* are taken as extending functions. The hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E})$ is consistent and complete.

Conservative hierarchical specifications in many-sorted equational logic without subsorts have a well-known flaw: it is not possible to define partial functions, for instance, an interpreter of a programming language or a theorem prover, since a partial function $f: \xi \to \eta$ would blow up its codomain η by adding "error elements" for the elements of its domain ξ for which it is not "defined". One approach to overcome this limitation is to generalize many-sorted equational logic to partial algebras [Reichel 80 and 87, Broy/Wirsing 82, Kamin/Archer 84].

In order-sorted equational logic, however, it is straightforward to accommodate partial functions. The key idea [Goguen/Meseguer 87c] is to equip every basic sort ξ with an *error supersort* $?\xi$ having its base sort ξ as a subsort. Furthermore, every declaration $f: \xi \to \eta$ of a possibly partial extending function f is replaced with its *lifting* $f: ?\xi \to ?\eta$ to the corresponding error supersorts. With that we accomplish that the elements of ξ for which f is not "defined" are mapped to error elements in $?\xi - \xi$, that is, to elements outside of the basic sort ξ.

Figure 5.2 shows how this method, which we like to call *stratification*, applies to the specification in Figure 5.1. All difficulties caused by the partial function *foo* disappear: the hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E})$ is consistent and complete and the initial algebra of $\mathcal{B} \cup \mathcal{E}$ is in fact what we wanted to specify in the first place. Note that the rewriting system $\mathcal{B} \cup \mathcal{E}$ is sort decreasing and confluent although $\mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2$ is not confluent.

5.1 Strict Specifications

To accomodate partial functions, we assume from now on that the set of sort symbols is partitioned into two disjoint and infinite classes whose elements are called **basic sort symbols** and **error sort symbols.**

Let Σ be a signature. A Σ-term s is called

- **admissible** if every variable occurring in s has a basic sort
- **basic in** Σ if there exists a basic sort symbol ξ such that s has sort ξ

A function symbol f is called **basic in** Σ if Σ contains a declaration $f\colon \xi_1 \cdots \xi_n \to \xi$ such that ξ is basic. A Σ-equation $s \doteq t$ is called **basic in** Σ if s and t are basic in Σ.

A signature Σ is called **strict** if the following conditions are satisfied:

- if $(\xi < \eta) \in \Sigma$ and η is basic, then ξ is basic
- if $(f\colon \xi_1 \cdots \xi_n \to \xi) \in \Sigma$ and ξ is basic, then $\xi_1, \ldots, \xi_n$ are basic.

Proposition 5.1. *Let Σ be a strict signature. Then:*

- *every subterm of an admissible Σ-term is an admissible Σ-term*
- *every subterm of a basic Σ-term is a basic Σ-term*
- *every function symbol occurring in a basic Σ-term is basic in Σ*

- *every Σ-instance of an admissible Σ-term is an admissible Σ-term*
- *every Σ-instance of a basic Σ-term is a basic Σ-term.*

A specification is called **admissible** if its signature is strict and all its axioms are admissible. A rewriting system is called **admissible** if it is an admissible specification. In the rest of the paper, we will only consider admissible specifications and admissible equations. Note that every specification that doesn't contain error sort symbols is admissible.

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be an admissible specification. Then the **base** $\mathcal{S}_\mathrm{B} = (\Sigma_\mathrm{B}, \mathcal{E}_\mathrm{B})$ **of** $\mathcal{S}$ is the specification defined as follows:

- $\Sigma_\mathrm{B} \subseteq \Sigma$ is the set of all declarations of Σ that don't contain error sort symbols
- $\mathcal{E}_\mathrm{B} \subseteq \mathcal{E}$ is the set of all axioms of $\mathcal{S}$ that don't contain function symbols that are nonbasic in Σ.

Proposition 5.2. *Let $\mathcal{S}$ be an admissible specification. Then $(\mathcal{S}_\mathrm{B}, \mathcal{S})$ is a complete hierarchical specification. Furthermore, $\mathcal{S}_\mathrm{B}$ is regular if $\mathcal{S}$ is regular.*

We call an admissible specification $\mathcal{S}$ **consistent** if $(\mathcal{S}_\mathrm{B}, \mathcal{S})$ is a consistent hierarchical specification.

Let $\mathcal{S}$ be an admissible specification. An $\mathcal{S}$-term s is called **sensible in $\mathcal{S}$** if, for every ground $\mathcal{S}$-instance t of s, there exists a ground and basic $\mathcal{S}$-term u such that $t =_\mathcal{S} u$.

Proposition 5.3. *Let $\mathcal{S}$ be an admissible specification. Then an $\mathcal{S}$-term s is sensible in $\mathcal{S}$ if and only if every $\mathcal{S}$-instance of s is sensible in $\mathcal{S}$.*

An admissible specification $\mathcal{S}$ is called **strict** if every subterm of a sensible $\mathcal{S}$-term is sensible in $\mathcal{S}$.

Proposition 5.4. *Let $\mathcal{S}$ be an admissible specification. Then $\mathcal{S}$ is strict if and only if every subterm of a ground and sensible $\mathcal{S}$-term is sensible in $\mathcal{S}$.*

Proposition 5.5. *Let $\mathcal{S}$ be a specification that doesn't contain error sort symbols. Then $\mathcal{S}$ is a consistent and strict specification and every $\mathcal{S}$-term is sensible in $\mathcal{S}$.*

It is of course undecidable whether an admissible specification is strict. However, in Section 7 we will give a decidable sufficient condition for the strictness of ground confluent rewriting systems.

5.2 Stratification

In practice, it is quite inconvenient to introduce the error supersorts needed to accomodate partial functions by hand. A convenient alternative is to declare partial functions as such and to not write any error supersorts at all. Such a sugared specification can then be translated automatically into an admissible specification with error supersorts where the functions declared as partial are lifted accordingly. We now give such a translation method, called stratification, that preserves the sort discipline and yields an admissible specification.

Let $\mathcal{S}$ be a specification not containing error sort symbols and let F be a set of function symbols. The functions in F are supposed to be the partial functions of $\mathcal{S}$. A **stratification of $\mathcal{S} = (\Sigma, \mathcal{E})$ with repect to F** is a specification $S' = (\Sigma', \mathcal{E})$ whose signature Σ' can be constructed as follows:

1. if ξ is a Σ-sort symbol, then put $\xi <?\xi$ into Σ', where $?\xi$ is a new error sort symbol called the **error supersort** of ξ
2. if $\xi < \eta$ is a declaration of Σ, then put $\xi < \eta$ and $?\xi <?\eta$ into Σ'
3. if $f\colon \xi_1 \cdots \xi_n \to \xi$ is a declaration of Σ and $f \in F$, then put the **lifted declaration** $f\colon ?\xi_1 \cdots ?\xi_n \to ?\xi$ into Σ'
4. if $f\colon \xi_1 \cdots \xi_n \to \xi$ is a declaration of Σ and $f \notin F$, then put $f\colon \xi_1 \cdots \xi_n \to \xi$ into Σ'

5. if $f: \xi_1 \cdots \xi_n \to \xi$ is a declaration of Σ, $f \notin F$, and $n > 0$, then put the **lifted declaration** $f: ?\xi_1 \cdots ?\xi_n \to ?\xi$ into Σ'.

We say that $\mathcal{S}'$ **is a stratification of** $\mathcal{S}$ if there exists a set F of function symbols such that $\mathcal{S}'$ is a stratification of $\mathcal{S}$ with respect to F. In the following we will tacitly assume that stratification is only applied to specifications not containing error sorts.

The specification $\mathcal{B} \cup \mathcal{E}$ in Figure 5.2 is a stratification of the specification $\mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2$ in Figure 5.1 with respect to *le* and *foo*. Stratication of the inconsistent and incomplete hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E}_1 \cup \mathcal{E}_2)$ thus yields the conservative hierarchical specification $(\mathcal{B}, \mathcal{B} \cup \mathcal{E})$. Note also that $\mathcal{B} \cup \mathcal{E}$ is a sort decreasing and confluent rewriting system.

Proposition 5.6. *Let $\mathcal{S}'$ be a stratification of $\mathcal{S}$. Then $\mathcal{S}'$ is an admissible specification and $(\mathcal{S}'_{\mathrm{B}}, \mathcal{S}')$ is a complete hierarchical specification. Furthermore, $\mathcal{S}'$ is regular if and only if $\mathcal{S}$ is regular.*

Proposition 5.7. *Let $\mathcal{S}'$ be a stratification of $\mathcal{S}$. Then:*

$$\begin{array}{llllll} \xi \leq \eta \text{ in } \mathcal{S} & \iff & \xi \leq \eta \text{ in } \mathcal{S}'_{\mathrm{B}} & \iff & \xi \leq \eta \text{ in } \mathcal{S}' \\ & \iff & ?\xi \leq ?\eta \text{ in } \mathcal{S}' & \iff & \xi \leq ?\eta \text{ in } \mathcal{S}' \end{array}$$

if ξ and η are $\mathcal{S}$-sort symbols.

Theorem 5.8. *Let $\mathcal{S}'$ be a stratification of $\mathcal{S}$. Then $\mathcal{S}$ and $\mathcal{S}'$ have equivalent sort checking disciplines:*

1. *if s is an $\mathcal{S}$-term of sort ξ, then s is an admissible $\mathcal{S}'$-term of sort $?\xi$*

2. *if s is an admissible $\mathcal{S}'$-term of sort $?\xi$, then s is an $\mathcal{S}$-term of sort ξ.*

Proof. 1. Let s be an $\mathcal{S}$-term of sort ξ. We show by induction on the term structure of s that s is an $\mathcal{S}'$-term of sort $?\xi$. If $s = x$, then $\sigma x \leq \xi$ in $\mathcal{S}$. Hence we know that $\sigma x \leq \xi \leq ?\xi$ in $\mathcal{S}'$. Thus s is an $\mathcal{S}'$-term of sort $?\xi$. If $s = f(s_1, \ldots, s_n)$, then $\mathcal{S}$ contains a declaration $f: \eta_1 \cdots \eta_n \to \eta$ such that

$\eta \leq \xi$ in $\mathcal{S}$ and s_i is an $\mathcal{S}$-term of sort η_i for $i = 1, \ldots, n$. Hence we know by the induction hypothesis that s_i is an $\mathcal{S}'$-term of sort $?\eta_i$ for $i = 1, \ldots, n$. If $n = 0$, then $\mathcal{S}'$ contains either $f\colon \rightarrow \eta$ or $f\colon \rightarrow ?\eta$. Hence $s = f$ is an $\mathcal{S}'$-term of sort $?\eta$. If $n > 0$, then $\mathcal{S}'$ contains the declaration $f\colon ?\eta_1 \cdots ?\eta_n \rightarrow ?\eta$. Hence $s = f(s_1, \ldots, s_n)$ is an $\mathcal{S}'$-term of sort $?\eta$. Since $\eta \leq \xi$ in $\mathcal{S}$, we know that $?\eta \leq ?\xi$ in $\mathcal{S}'$. Thus s is an $\mathcal{S}'$-term of sort $?\xi$.

2. Let s be an admissible $\mathcal{S}'$-term of sort $?\xi$, where $?\xi$ is the error supersort symbol of the $\mathcal{S}$-sort symbol ξ. We prove by induction on the term structure of s that s is an $\mathcal{S}$-term of sort ξ. If $s = x$, then s is an $\mathcal{S}'$-term of sort σx and $\sigma x \leq ?\xi$ in $\mathcal{S}'$. Since s is assumed to be admissible, σx is an $\mathcal{S}$-sort symbol. Hence s is an $\mathcal{S}$-term of sort σx. Since $\sigma x \leq \xi$ in $\mathcal{S}$, s is an $\mathcal{S}$-term of sort ξ. If $s = f$ and $f \notin F$, then $\mathcal{S}'$ contains a declaration $f\colon \rightarrow \eta$ such that $\eta \leq ?\xi$ and η is an $\mathcal{S}$-sort symbol. Hence s is an $\mathcal{S}$-term of sort η. Since $\eta \leq \xi$ in $\mathcal{S}$, s is an $\mathcal{S}$-term of sort ξ. If $s = f(s_1, \ldots, s_n)$ and $n > 0$, then $\mathcal{S}'$ contains a declaration $f\colon ?\eta_1 \cdots ?\eta_n \rightarrow ?\eta$ such that $?\eta \leq ?\xi$ in $\mathcal{S}'$, s_i is an $\mathcal{S}'$-term of sort $?\eta$ for $i = 1, \ldots, n$, and $\eta_1, \ldots, \eta_n$ and η are $\mathcal{S}$-sort symbols. Hence we know by the induction hypothesis that s_i is an $\mathcal{S}$-term of sort η for $i = 1, \ldots, n$. Since $\mathcal{S}$ contains the declaration $f\colon \eta_1 \cdots \eta_n \rightarrow \eta$, we know that $s = f(s_1, \ldots, s_n)$ is an $\mathcal{S}$-term of sort η. Since $\eta \leq \xi$ in $\mathcal{S}$, s is an $\mathcal{S}$-term of sort ξ. □

Stratification is a method to accommodate partial functions that don't add new data elements. Since total functions that don't add new data elements are also partial functions, one could also stratify with respect to them. We will prove in Section 7 that stratification with respect to total functions that don't add new data elements doesn't change the initial algebra semantics of a specification.

5.3 Example: Algebraic Semantics for Programming Languages

Figures 5.3 and 5.4 show an algebraic specification of the semantics of a very simple but Turing-complete imperative programming language. Since the language allows for nonterminating programs, the specified interpreter *evalp* is a partial function.

$nat := \{o,\ s\colon\ nat\}$
$var := \{v\colon\ nat\}$
$exp := nat \mathbin{++} var \mathbin{++} \{le\colon\ exp \times exp\} \mathbin{++} \{inc\colon\ exp\}$
$stat := assignment \mathbin{++} conditional \mathbin{++} loop$
$assignment := \{\text{':='}\colon\ var \times exp\}$
$conditional := \{if\colon\ exp \times prog \times prog\}$
$loop := \{while\colon\ exp \times prog\}$
$prog := stat \mathbin{++} \{\text{';'}\colon\ stat \times prog\}$
$configuration := \{empty,\ c\colon\ var \times nat \times configuration\}$

$bool := \{true,\ false\}$

$==\colon\ nat \times nat \rightarrow bool$
$==\colon\ var \times var \rightarrow bool$

$(N{:}nat == N'{:}nat) = \textbf{if } N \leq N' \textbf{ then } N' \leq N \textbf{ else } false \textbf{ fi}$
$(v(N) == v(N')) \quad = (N == N')$

$\leq\colon\ nat \times nat \rightarrow bool$

$o \leq N \qquad\quad = true$
$s(N) \leq o \qquad = false$
$s(N) \leq s(N') = N \leq N'$

Figure 5.3. The data structures and auxiliary functions of an abstract interpreter for a simple imperative programming language.

The example is written in sugared syntax. The actual specification is obtained by applying stratification with respect to the functions declared as partial with the symbol ~>.

The signature equation

$nat := \{o,\ s\colon\ nat\}$

$$
\begin{array}{l}
\textit{evalp: prog}\times\textit{configuration} \rightsquigarrow\!> \textit{configuration}\\[1ex]
\quad \begin{array}{ll}
\textit{evalp}(X{:=}E,\ C) & = \textit{update}(C,\ X,\ \textit{evale}(E,C))\\
\textit{evalp}(\textit{if}(E,P,P'),\ C) & = \textbf{if } \textit{evale}(E,C){=}{=}s(o) \textbf{ then } \textit{evalp}(P,C)\\
& \quad \textbf{else } \textit{evalp}(P',C) \textbf{ fi}\\
\textit{evalp}(\textit{while}(E,P),\ C) & = \textbf{if } \textit{evale}(E,C){=}{=}s(o)\\
& \quad \textbf{then } \textit{evalp}((P;\textit{while}(E,P)),\ C)\\
& \quad \textbf{else } C \textbf{ fi}\\
\textit{evalp}(S;P,\ C) & = \textit{evalp}(P,\ \textit{evalp}(S,C))
\end{array}\\[2ex]
\textit{evale: exp}\times\textit{configuration} \rightsquigarrow\!> \textit{nat}\\[1ex]
\quad \begin{array}{ll}
\textit{evale}(N{:}\textit{nat},\ C) & = N\\
\textit{evale}(V{:}\textit{var},\ c(V',N,C)) & = \textbf{if } V{=}{=}V' \textbf{ then } N\\
& \quad \textbf{else } \textit{evale}(V,C) \textbf{ fi}\\
\textit{evale}(\textit{le}(E,E'),\ C) & = \textbf{if } \textit{evale}(E,C) \leq \textit{evale}(E',C) \textbf{ then } s(o)\\
& \quad \textbf{else } o \textbf{ fi}\\
\textit{evale}(\textit{inc}(E),\ C) & = s(\textit{evale}(E,C))
\end{array}\\[2ex]
\textit{update: configuration}\times\textit{var}\times\textit{nat} \rightarrow \textit{configuration}\\[1ex]
\quad \begin{array}{ll}
\textit{update}(\textit{empty},\ V,\ N) & = c(V,\ N,\ \textit{empty})\\
\textit{update}(c(V,N,C),\ V',\ N') & = \textbf{if } V{=}{=}V' \textbf{ then } c(V,\ N',\ C)\\
& \quad \textbf{else } c(V,\ N,\ \textit{update}(C,\ V',\ N')) \textbf{ fi}
\end{array}
\end{array}
$$

Figure 5.4. An abstract interpreter for a simple imperative programming language.

stands for the constructor declarations

$$o\colon \rightarrow \textit{nat}, \quad s\colon \textit{nat} \rightarrow \textit{nat}$$

and asserts that the sort *nat* is completely specified by its two constructors.

The signature equation

$$exp := nat \;{+}{+}\; var \;{+}{+}\; \{le\!: exp \times exp\} \;{+}{+}\; \{inc\!: exp\}$$

stands for the declarations

$$nat < exp, \quad var < exp, \quad le\!: exp \times exp \rightarrow exp, \quad inc\!: exp \rightarrow exp$$

and asserts that the sort *exp* is completely specified by the subtypes *nat* and *var* and the free constructors *le* and *inc*.

The function == defines an equality test for the types *nat* and *var*. The equation

$$(N{:}nat == N'{:}nat) \quad = \quad \textbf{if } N \leq N' \textbf{ then } N' \leq N \textbf{ else } \mathit{false} \textbf{ fi}$$

is syntactic sugar for the equation

$$(N == N') \quad = \quad foo(N \leq N', N, N')$$

where N and N' are variables of sort *nat* and *foo* is an automatically introduced auxiliary function defined as follows:

$$foo\!: bool \times nat \times nat \sim> bool$$
$$foo(true, N, N') = N' \leq N$$
$$foo(false, N, N') = false.$$

Here there is actually no need to declare the auxiliary function *foo* as partial. However, the auxiliary functions needed for the conditionals in the definitions of *evalp* and *evale* are in fact partial.

Note that the standard solution for defining conditionals

$$if\!: bool \times bool \times bool \rightarrow bool$$
$$if(true, B{:}bool, B'{:}bool) = B$$
$$if(false, B{:}bool, B'{:}bool) = B'$$

doesn't work here since stratification would turn *if* into a strict function. However, our translation of conditionals preserves their nonstrictness.

6 Strict Algebras and Base Homomorphisms

So far our approach to strict specifications with partial functions has been of a syntactic nature. We will now define strict algebras and show that a specification is strict if and only if its initial algebra is strict. We also will discuss strict equality, which is an appropriate equality relation for strict algebras.

6.1 Strict Algebras and Strict Equality

The **base** $\mathrm{B}_{\mathcal{A}}$ of a Σ-algebra $\mathcal{A}$ is

$$\mathrm{B}_{\mathcal{A}} := \bigcup\{\xi^{\mathcal{A}} \mid \xi \text{ is a basic sort symbol of } \Sigma\}.$$

The elements of $\mathrm{B}_{\mathcal{A}}$ are called the **base elements** and the elements of $\mathrm{C}_{\mathcal{A}} - \mathrm{B}_{\mathcal{A}}$ are called the **error elements** of $\mathcal{A}$.

A Σ-algebra $\mathcal{A}$ is called **strict** if for every Σ-function symbol f and every tuple $(a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}}$

$$f^{\mathcal{A}}(a_1, \ldots, a_n) \in \mathrm{B}_{\mathcal{A}} \quad \Rightarrow \quad a_1 \in \mathrm{B}_{\mathcal{A}} \ \wedge \ \ldots \ \wedge \ a_n \in \mathrm{B}_{\mathcal{A}}.$$

In a strict algebra error elements are always mapped to error elements. A Σ-algebra $\mathcal{A}$ is called **total** if $\mathrm{B}_{\mathcal{A}} = \mathrm{C}_{\mathcal{A}}$, and it is called **partial** if $\mathrm{B}_{\mathcal{A}} \neq \mathrm{C}_{\mathcal{A}}$. Note that a total algebra is always strict.

Proposition 6.1. *Let Σ be a signature not containing error sort symbols. Then every Σ-algebra is strict and total.*

Proposition 6.2. *Let Σ be a strict signature and V be a set of Σ-variables. Then the term algebra $\mathcal{T}_{\Sigma,V}$ is strict.*

Proposition 6.3. *Let $\mathcal{A}$ be a strict Σ-algebra, s be a (Σ, V)-term and α be a $(V, \mathcal{A})$-assignment. Then*

$$[\![s]\!]_\alpha \in \mathrm{B}_{\mathcal{A}} \text{ and } t \text{ is a subterm of } s \quad \Rightarrow \quad [\![t]\!]_\alpha \in \mathrm{B}_{\mathcal{A}}.$$

Theorem 6.4. *The initial algebra $\mathcal{I}_\mathcal{S}$ of an admissible specification $\mathcal{S}$ is strict if and only if $\mathcal{S}$ is a strict specification.*

Proof. 1. Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a strict specification and let $f(s_1, \ldots, s_n)$ be a ground Σ-term such that $\overline{f(s_1, \ldots, s_n)}$ contains a basic Σ-term u, where $\overline{f(s_1, \ldots, s_n)}$ is defined as in the construction of $\mathcal{I}_\mathcal{S}$. To prove that $\mathcal{I}_\mathcal{S}$ is strict, it suffices to show that $\overline{s_1}, \ldots, \overline{s_n}$ contain basic Σ-terms. Since $\mathcal{S} \vdash u \doteq f(s_1, \ldots, s_n)$ and u is basic, we know that $f(s_1, \ldots, s_n)$ is sensible in $\mathcal{S}$. Since $\mathcal{S}$ is strict, we thus know that $s_1, \ldots, s_n$ are sensible in $\mathcal{S}$. Hence, there exist basic and ground Σ-terms $v_1, \ldots, v_n$ such that $\mathcal{S} \vdash v_i \doteq s_i$ for $i = 1, \ldots, n$. Thus $\overline{s_1}, \ldots, \overline{s_n}$ contain basic Σ-terms.

2. Let $\mathcal{S}$ be an admissible specification whose initial algebra $\mathcal{I}_\mathcal{S}$ is strict, s be a ground and sensible $\mathcal{S}$-term, and u be a subterm of s. We have to show that there exists a ground and basic $\mathcal{S}$-term v such that $u =_\mathcal{S} v$. Since s is sensible, there exists a ground and basic $\mathcal{S}$-term t such that $s =_\mathcal{S} t$. Hence there is a basic $\mathcal{S}$-sort symbol ξ such that $[\![s]\!]_{\mathcal{I}_\mathcal{S}} \in \xi^{\mathcal{I}_\mathcal{S}}$. Thus we know by Proposition 6.3 that $[\![u]\!]_{\mathcal{I}_\mathcal{S}}$ is a base element of $\mathcal{I}_\mathcal{S}$. Hence there exists a ground and basic $\mathcal{S}$-term v such that $\mathcal{S} \vdash u \doteq v$. $\square$

Corollary 6.5. *Let $\mathcal{S}$ be a strict specification. Then $\mathcal{I}_\mathcal{S}$ is an initial object in the category comprised of the strict $\mathcal{S}$-algebras and their homomorphisms.*

In strict algebras, we are not interested in equality between terms that denote error elements. Furthermore, we are only interested in equality between admissible terms, that is, terms all of whose variables range over basic sorts. Thus we actually need a three-valued logic where the the truthvalue of an equation can be "true", "false", and "undefined".

The **truth value** $[\![s \doteq t]\!]_\mathcal{A}$ of a Σ-equation $s \doteq t$ in a Σ-algebra $\mathcal{A}$ is defined as follows:

- $[\![s \doteq t]\!]_\mathcal{A} := \mathbf{tt}$ if $[\![s]\!]_\alpha = [\![t]\!]_\alpha \in \mathrm{B}_\mathcal{A}$ for every $(\mathcal{V}(s \doteq t), \mathcal{A})$-assignment α
- $[\![s \doteq t]\!]_\mathcal{A} := \mathbf{ff}$ if $[\![s]\!]_\beta \neq [\![t]\!]_\beta$ for some $(\mathcal{V}(s \doteq t), \mathcal{A})$-assignment β and $[\![s]\!]_\alpha, [\![t]\!]_\alpha \in \mathrm{B}_\mathcal{A}$ for every $(\mathcal{V}(s \doteq t), \mathcal{A})$-assignment α

- $[\![s \doteq t]\!]_{\mathcal{A}} := \mathbf{uu}$ otherwise.

Proposition 6.6. *Let $\mathcal{A}$ be a Σ-algebra and $s \doteq t$ be a Σ-equation. Then:*

- *if $[\![s \doteq t]\!]_{\mathcal{A}} = \mathbf{tt}$, then $s \doteq t$ is valid in $\mathcal{A}$*
- *if $[\![s \doteq t]\!]_{\mathcal{A}} = \mathbf{ff}$, then $s \doteq t$ is not valid in $\mathcal{A}$*
- *if $[\![s \doteq t]\!]_{\mathcal{A}} \neq \mathbf{uu}$, then $s \doteq t$ is valid in $\mathcal{A}$ if and only if $[\![s \doteq t]\!]_{\mathcal{A}} = \mathbf{tt}$, and $s \doteq t$ is not valid in $\mathcal{A}$ if and only if $[\![s \doteq t]\!]_{\mathcal{A}} = \mathbf{ff}$.*

Proposition 6.7. *Let $\mathcal{A}$ be a Σ-algebra. Then $[\![s \doteq t]\!]_{\mathcal{A}} \neq \mathbf{uu}$ if s and t are basic Σ-terms.*

Proposition 6.8. *Let $\mathcal{S}$ be an admissible specification and $\mathcal{A}$ be an $\mathcal{S}$-algebra. Then $[\![s \doteq t]\!]_{\mathcal{A}} \neq \mathbf{uu}$ if s and t are ground sensible $\mathcal{S}$-terms.*

Proposition 6.9. *Let $\mathcal{S}$ be an admissible specification and $\mathcal{A}$ be an $\mathcal{S}$-algebra without junk. Then $[\![s \doteq t]\!]_{\mathcal{A}} \neq \mathbf{uu}$ if s and t are sensible $\mathcal{S}$-terms.*

6.2 Base Homomorphisms

Base homomorphisms generalize ordinary homomorphisms in that they only relate nonerror elements. We will show that base homomorphisms are the appropriate homomorphisms for strict algebras since (1) strict base isomorphic algebras agree with respect to strict equality and (2) the initial algebra of a strict specification $\mathcal{S}$ is an initial object in the category comprised of the strict algebras and their base homomorphisms.

Furthermore, base homomorphisms allow relating algebras with different signatures, and base isomorphisms will provide the right notion of semantic equivalence for the signature transformations presented in the next section.

A **presignature** is a set of function and basic sort symbols. If Σ is a signature, the **presignature $|\Sigma|$ of Σ** is the set of all function and basic sort symbols occurring in the declarations of Σ. If Π is a presignature, a Σ-algebra is called a Π-algebra if Π is the presignature of Σ. The letter Π will always range over presignatures.

The **basic domain** $\mathrm{BD}_f^{\mathcal{A}}$ of a Σ-function symbol f in a Σ-algebra $\mathcal{A}$ is

$$\mathrm{BD}_f^{\mathcal{A}} := \{(a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}} \cap (\mathrm{B}_{\mathcal{A}})^{|f|} \mid f^{\mathcal{A}}(a_1, \ldots, a_n) \in \mathrm{B}_{\mathcal{A}}\}.$$

A **base homomorphism** from a Π-algebra $\mathcal{A}$ to a Π-algebra $\mathcal{B}$ is a mapping $\gamma: \mathrm{B}_{\mathcal{A}} \to \mathrm{B}_{\mathcal{B}}$ such that

1. if ξ is a basic sort symbol of Π, then $\gamma(\xi_{\mathcal{A}}) \subseteq \xi_{\mathcal{B}}$
2. if f is a function symbol of Π, then

   2.1 $\gamma(\mathrm{BD}_f^{\mathcal{A}}) \subseteq \mathrm{BD}_f^{\mathcal{B}}$

   2.2 if $(a_1, \ldots, a_n) \in \mathrm{BD}_f^{\mathcal{A}}$, then $\gamma(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(\gamma(a_1), \ldots, \gamma(a_n))$.

A base homomorphism $\gamma: \mathcal{A} \to \mathcal{B}$ is a **base isomorphism** if there exists a base homomorphism $\gamma': \mathcal{B} \to \mathcal{A}$ such that $\gamma\gamma' = \mathrm{id}_{\mathrm{B}_{\mathcal{A}}}$ and $\gamma'\gamma = \mathrm{id}_{\mathrm{B}_{\mathcal{B}}}$. Two Π-algebras $\mathcal{A}$ and $\mathcal{B}$ are called **base isomorphic** if there exists a base isomorphism $\mathcal{A} \to \mathcal{B}$.

Proposition 6.10. *Let Π be a presignature. Then the Π-algebras together with their base homomorphisms form a category.*

Proposition 6.11. *The restriction of a homomorphism $\mathcal{A} \to \mathcal{B}$ to the base of $\mathcal{A}$ is a base homomorphism $\mathcal{A} \to \mathcal{B}$.*

Proposition 6.12. *Let Σ be a signature not containing error sort symbols. Then a base homomorphism from a Σ-algebra $\mathcal{A}$ to a Σ-algebra $\mathcal{B}$ is a homomorphism $\mathcal{A} \to \mathcal{B}$.*

Theorem 6.13. *Let $\mathcal{S}$ be a specification. Then $\mathcal{I}_{\mathcal{S}}$ is an initial object in the category comprised of the $\mathcal{S}$-algebras and their base homomorphisms.*

Proof. Since $\mathcal{I}_{\mathcal{S}}$ is an initial object in the category comprised of the $\mathcal{S}$-algebras and their homomorphisms and the restriction of a homomorphism

$\mathcal{I_S} \to \mathcal{A}$ to the base of $\mathcal{I_S}$ is a base homomorphism $\mathcal{I_S} \to \mathcal{A}$, we know that there exists a base homomorphism $\mathcal{I_S} \to \mathcal{A}$ for every $\mathcal{S}$-algebra $\mathcal{A}$.

Let $\mathcal{A}$ be an $\mathcal{S}$-algebra and let γ and γ' be two base homomorphisms $\mathcal{I_S} \to \mathcal{A}$. We have to show that $\gamma = \gamma'$. Let s be a ground and basic Σ-term. It suffices to show that $\gamma(\llbracket s \rrbracket_{\mathcal{I_S}}) = \gamma'(\llbracket s \rrbracket_{\mathcal{I_S}})$, which we prove by induction on the term structure of s. Let $s = f(s_1, \ldots, s_n)$. Then

$$\llbracket s \rrbracket_{\mathcal{I_S}} = \llbracket f(s_1, \ldots, s_n) \rrbracket_{\mathcal{I_S}} = f^{\mathcal{I_S}}(\llbracket s_1 \rrbracket_{\mathcal{I_S}}, \ldots, \llbracket s_n \rrbracket_{\mathcal{I_S}})$$

and $(\llbracket s_1 \rrbracket_{\mathcal{I_S}}, \ldots, \llbracket s_n \rrbracket_{\mathcal{I_S}}) \in \mathrm{BD}_f^{\mathcal{I_S}}$. Hence $\gamma(\llbracket s \rrbracket_{\mathcal{I_S}}) = f^{\mathcal{A}}(\gamma(\llbracket s_1 \rrbracket_{\mathcal{I_S}}), \ldots, \gamma(\llbracket s_n \rrbracket_{\mathcal{I_S}}))$ and $\gamma'(\llbracket s \rrbracket_{\mathcal{I_S}}) = f^{\mathcal{A}}(\gamma'(\llbracket s_1 \rrbracket_{\mathcal{I_S}}), \ldots, \gamma'(\llbracket s_n \rrbracket_{\mathcal{I_S}}))$. Thus we have by the induction hypothesis that $\gamma(\llbracket s \rrbracket_{\mathcal{I_S}}) = \gamma'(\llbracket s \rrbracket_{\mathcal{I_S}})$. □

Corollary 6.14. *Let $\mathcal{S}$ be a strict specification. Then $\mathcal{I_S}$ is an initial object in the category comprised of the strict $\mathcal{S}$-algebras with their base homomorphisms.*

Lemma 6.15. *Let Σ and Σ' be two signatures such that $|\Sigma| = |\Sigma'|$, $\mathcal{A}$ be a strict Σ-algebra, $\mathcal{B}$ be a Σ'-algebra, and γ be a base homomorphism $\mathcal{A} \to \mathcal{B}$. Furthermore, let V be a set of basic Σ-variables and α be a $(V, \mathcal{A})$-assignment. Then $\gamma\alpha$ is a $(V, \mathcal{B})$-assignment and*

$$\llbracket s \rrbracket_\alpha \in \mathrm{B}_{\mathcal{A}} \quad \Rightarrow \quad \llbracket s \rrbracket_{\gamma\alpha} = \gamma(\llbracket s \rrbracket_\alpha) \in \mathrm{B}_{\mathcal{B}}$$

for every (Σ, V)-term s such that s is a Σ'-term.

Proof. Let s be a (Σ, V)-term such that s is a Σ'-term and $\llbracket s \rrbracket_\alpha \in \mathrm{B}_{\mathcal{A}}$. We prove by induction on the term structure of s that $\llbracket s \rrbracket_{\gamma\alpha} = \gamma(\llbracket s \rrbracket_\alpha) \in \mathrm{B}_{\mathcal{B}}$. If $s = x$, then $\llbracket s \rrbracket_{\gamma\alpha} = \llbracket x \rrbracket_{\gamma\alpha} = \gamma(\alpha(x)) = \gamma(\llbracket x \rrbracket_\alpha) = \gamma(\llbracket s \rrbracket_\alpha) \in \mathrm{B}_{\mathcal{B}}$. If $s = f(s_1, \ldots, s_n)$, then $\llbracket f(s_1, \ldots, s_n) \rrbracket_\alpha = f^{\mathcal{A}}(\llbracket s_1 \rrbracket_\alpha, \ldots, \llbracket s_n \rrbracket_\alpha) \in \mathrm{B}_{\mathcal{A}}$. Since $\mathcal{A}$ is strict, we know that $(\llbracket s_1 \rrbracket_\alpha, \ldots, \llbracket s_n \rrbracket_\alpha) \in \mathrm{BD}_f^{\mathcal{A}}$. Hence we know by the induction hypothesis that $\llbracket s_i \rrbracket_{\gamma\alpha} = \gamma(\llbracket s_i \rrbracket_\alpha) \in \mathrm{B}_{\mathcal{B}}$ for $i = 1, \ldots, n$. Hence $\llbracket s \rrbracket_{\gamma\alpha} = \llbracket f(s_1, \ldots, s_n) \rrbracket_{\gamma\alpha} = f^{\mathcal{B}}(\llbracket s_1 \rrbracket_{\gamma\alpha}, \ldots, \llbracket s_n \rrbracket_{\gamma\alpha}) = f^{\mathcal{B}}(\gamma(\llbracket s_1 \rrbracket_\alpha), \ldots, \gamma(\llbracket s_n \rrbracket_\alpha)) = \gamma(f^{\mathcal{A}}(\llbracket s_1 \rrbracket_\alpha, \ldots, \llbracket s_n \rrbracket_\alpha)) = \gamma(\llbracket f(s_1, \ldots, s_n) \rrbracket_\alpha) = \gamma(\llbracket s \rrbracket_\alpha)$. □

Theorem 6.16. (Invariance of Strict Equality) *Let $\mathcal{A}$ be a strict Σ-algebra, $\mathcal{B}$ be a strict Σ'-algebra, and $|\Sigma| = |\Sigma'|$. Then $[\![s \doteq t]\!]_{\mathcal{A}} = [\![s \doteq t]\!]_{\mathcal{B}}$ if $\mathcal{A}$ and $\mathcal{B}$ are base isomorphic and $s \doteq t$ is an admissible Σ- and Σ'-equation.*

Proof. Let γ be a base isomorphism $\mathcal{A} \to \mathcal{B}$. Furthermore, let V be a set of basic Σ-variables and s be a (Σ, V)-term such that s is also a Σ'-term and $[\![s]\!]_\alpha \in \mathrm{B}_{\mathcal{A}}$ for every $(V, \mathcal{A})$-assignment α. By the preceding lemma we know that it suffices to show that $[\![s]\!]_\beta \in \mathrm{B}_{\mathcal{B}}$ for every $(V, \mathcal{B})$-assignment β. Let β be a $(V, \mathcal{B})$-assignment. Then $\gamma^{-1}\beta$ is a $(V, \mathcal{A})$-assignment. Hence $[\![s]\!]_{\gamma^{-1}\beta} \in \mathrm{B}_{\mathcal{A}}$ by our assumption. Since $\gamma\gamma^{-1}\beta = \beta$, we have by the preceding lemma that $[\![s]\!]_\beta = [\![s]\!]_{\gamma(\gamma^{-1}\beta)} \in \mathrm{B}_{\mathcal{B}}$. □

We now give a construction that transforms a strict $\mathcal{S}$-algebra into a base isomorphic strict $\mathcal{S}$-algebra with at most one error element.

Construction 6.17. **($\mathcal{A}^\perp$)** Let $\mathcal{A}$ be a strict Σ-algebra, $\perp$ be a symbol not occurring in the base of $\mathcal{A}$, and let

$$|a| := \begin{cases} a & \text{if } a \in \mathrm{B}_{\mathcal{A}} \\ \perp & \text{otherwise.} \end{cases}$$

Then the Σ-algebra $\mathcal{A}^\perp$ is defined as follows:

- $\xi^{\mathcal{A}^\perp} := \{|a| \mid a \in \xi^{\mathcal{A}}\}$
- $\mathrm{D}_f^{\mathcal{A}^\perp} := \{(|a_1|, \ldots, |a_n|) \mid (a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}}\}$
- $f^{\mathcal{A}^\perp}(a_1, \ldots, a_n) := |f^{\mathcal{A}}(a_1, \ldots, a_n)|$.

Proposition 6.18. *Let $\mathcal{A}$ be a strict $\mathcal{S}$-algebra. Then $\mathcal{A}^\perp$ is a strict $\mathcal{S}$-algebra containing at most one error element and $\mathrm{id}_{\mathrm{B}_{\mathcal{A}}}$ is a base isomorphism $\mathcal{A} \to \mathcal{A}^\perp$. Furthermore, every $\mathcal{S}$-equation that is valid in $\mathcal{A}$ is valid in $\mathcal{A}^\perp$.*

7 Changing the Sort Discipline

In this section we will attack three problems that are specific to order-sorted logic:

- meaningful terms can be ill-sorted; for instance, $\mathit{fac}(8-7)$ is ill-sorted under the declarations $\mathbf{nat} < \mathbf{int}$, $7\colon \to \mathbf{nat}$, $8\colon \to \mathbf{nat}$, $\mathit{fac}\colon \mathbf{nat} \to \mathbf{nat}$, and $-\colon \mathbf{int} \times \mathbf{int} \to \mathbf{int}$, where *fac* is the factorial function
- rewriting systems may not be compatible and thus the Completeness Theorem may not apply
- rewriting systems may not be sort decreasing and thus the Critical Pair Theorem may not apply.

For the first two problems we will provide perfect solutions, while for the third problem we can only offer a partial solution. The tool for solving these sort problems are signature transformations that keep the semantics of a specification in a sufficiently strong sense invariant.

We will also validate our approach to partial functions by proving that defining completely defined total functions as partial functions does not change the inital algebra semantics. Furthermore, we will provide decidable sufficient criteria for the consistency and strictness of ground confluent rewriting systems.

7.1 Compatibility by Construction

From now on we assume that $\top$ is an error sort symbol.

Construction 7.1. **($\Sigma^\top$ and $\mathcal{S}^\top$)** Let Σ be a signature. Recall that we defined the base signature Σ_B of Σ as the set of all declarations of Σ that don't contain error sort symbols. The **compatible signature** $\Sigma^\top$ is defined as follows:

$$\begin{aligned}\Sigma^\top :=& \Sigma_B \\ &\cup \{\xi < \top \mid \xi \text{ is a basic } \Sigma\text{-sort symbol}\} \\ &\cup \{f\colon \top \cdots \top \to \top \mid f \text{ is a } \Sigma\text{-function symbol}\}.\end{aligned}$$

If $\mathcal{S} = (\Sigma, \mathcal{E})$ is a specification, then $\mathcal{S}^\top := (\Sigma^\top, \mathcal{E})$.

The construction of $\Sigma^\top$ identifies all error sorts and and extends all functions to the *top error sort* $\top$. Thus every term consisting of Σ-variables and Σ-function symbols is a $\Sigma^\top$-term if it just equips every function symbol with the appropriate number of arguments. The title of this subsection is motivated by the fact that $\mathcal{R}^\top$ is a compatible rewriting system if $\mathcal{R}$ is a rewriting system. Note that the construction of $\Sigma^\top$ is idempotent, that is, $(\Sigma^\top)^\top = \Sigma^\top$. Similar constructions that do not identify existing error sorts can be found in [Goguen/Meseguer 87c] and [Schmidt-Schauß 87].

We will show that $\mathcal{S}$ and $\mathcal{S}^\top$ have equivalent semantics, provided $\mathcal{S}$ is an admissible specification. The construction of $\mathcal{S}^\top$ is an important tool for proofs but is not intended for practical applications. However, if one changes an admissible specification $\mathcal{S}$ to $\mathcal{S}'$ by deleting and adding declarations not containing basic sorts, then $\mathcal{S}$ and $\mathcal{S}'$ will still have equivalent semantics since $\mathcal{S}^\top = \mathcal{S}'^\top$. In other words, the semantical equivalence of $\mathcal{S}$ and $\mathcal{S}^\top$ says that the error sort structure of an admissible specification is semantically irrelevant.

Proposition 7.2. *Let Σ be a strict signature. Then $\Sigma^\top$ is a strict signature and*

1. *$\Sigma^\top$ is more permissive than Σ, that is, every Σ-term is a $\Sigma^\top$-term*
2. *Σ and $\Sigma^\top$ have the same basic terms, that is, every basic Σ-term of sort ξ is a basic $\Sigma^\top$-term of sort ξ, and every basic $\Sigma^\top$-term of sort ξ is a basic Σ-term of sort ξ*
3. *Σ and $\Sigma^\top$ define the same instances for admissible Σ-terms, that is, every Σ-instance of an admissible Σ-term s is a $\Sigma^\top$-instance of s, and every $\Sigma^\top$-instance of an admissible Σ-term s is a Σ-instance of s.*

Lemma 7.3. *Let $\mathcal{A}$ be a Σ-algebra. Then there exists a $\Sigma^\top$-algebra $\mathcal{B}$ and a base homomorphism $\gamma: \mathcal{B} \rightarrow \mathcal{A}$ such that an admissible Σ-equation is valid in $\mathcal{A}$ if and only if it is valid in $\mathcal{B}$.*

Proof. We construct a $\Sigma^{\top}$-algebra $\mathcal{B}$ as follows:

- $\xi^{\mathcal{B}} := \xi^{\mathcal{A}}$ if ξ is a basic Σ-sort symbol
- $\top^{\mathcal{B}} := \mathrm{C}_{\mathcal{A}} \cup \{\natural\}$, where $\natural \notin \mathrm{C}_{\mathcal{A}}$
- $\mathrm{D}_f^{\mathcal{B}} := (\mathrm{C}_{\mathcal{B}})^{|f|}$
- $f^{\mathcal{B}}(a_1, \ldots, a_n) := \begin{cases} f^{\mathcal{A}}(a_1, \ldots, a_n) & \text{if } (a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}} \\ \natural & \text{otherwise.} \end{cases}$

It is obvious that $\mathrm{id}_{\mathrm{B}_{\mathcal{B}}} = \mathrm{id}_{\mathrm{B}_{\mathcal{A}}}$ is a base homomorphism $\mathcal{B} \to \mathcal{A}$. Furthermore, it's easy to verify that an admissible Σ-equation is valid in $\mathcal{A}$ if and only if it is valid in $\mathcal{B}$. □

Lemma 7.4. *Let Σ be a strict signature and $\mathcal{A}$ be a $\Sigma^{\top}$-algebra. Then there exists a Σ-algebra $\mathcal{B}$ and a base homomorphism $\gamma\colon \mathcal{B} \to \mathcal{A}$ such that an admissible Σ-equation is valid in $\mathcal{A}$ if and only if it is valid in $\mathcal{B}$.*

Proof. We construct a Σ-algebra $\mathcal{B}$ as follows:

- $\xi^{\mathcal{B}} := \xi^{\mathcal{A}}$ if ξ is a basic Σ-sort symbol
- $\xi^{\mathcal{B}} := \top^{\mathcal{A}}$ if ξ is an error Σ-sort symbol
- $\mathrm{D}_f^{\mathcal{B}} := \mathrm{D}_f^{\mathcal{A}}$
- $f^{\mathcal{B}}(a_1, \ldots, a_n) := f^{\mathcal{A}}(a_1, \ldots, a_n)$.

It is obvious that $\mathrm{id}_{\mathrm{B}_{\mathcal{B}}} = \mathrm{id}_{\mathrm{B}_{\mathcal{A}}}$ is a base homomorphism $\mathcal{A} \to \mathcal{B}$. Furthermore, it's easy to verify that an admissible Σ-equation is valid in $\mathcal{A}$ if and only if it is valid in $\mathcal{B}$. □

Theorem 7.5. (Equivalence of Validity) *Let $\mathcal{S}$ be an admissible specification. Then an admissible $\mathcal{S}$-equation is valid in $\mathcal{S}$ if and only if it is valid in $\mathcal{S}^{\top}$.*

Proof. Let $s \doteq t$ be an admissible $\mathcal{S}$-equation that is valid in $\mathcal{S}$ and let $\mathcal{A}$ be an $\mathcal{S}^{\top}$-algebra. We have to show that $s \doteq t$ is valid in $\mathcal{A}$. By the preceding lemma we know that there exists an $\mathcal{S}$-algebra $\mathcal{B}$ such that $s \doteq t$ is valid in $\mathcal{A}$ if and only if it is valid in $\mathcal{B}$. Since $s \doteq t$ is valid in $\mathcal{S}$, we know that $s \doteq t$ is valid in $\mathcal{B}$. Hence $s \doteq t$ is valid in $\mathcal{A}$. The converse direction is shown analogously. □

Similar theorems can be found in [Goguen/Meseguer 87c] and [Schmidt-Schauß 87].

Corollary 7.6. (Equivalence of Initial Validity) *Let $\mathcal{S}$ be an admissible specification. Then an admissible $\mathcal{S}$-equation is valid in $\mathcal{I}_{\mathcal{S}}$ if and only if it is valid in $\mathcal{I}_{\mathcal{S}^\top}$.*

Proof. Follows from the Structural Induction Theorem (2.30), the preceding theorem, and Proposition 7.2. □

Corollary 7.7. *Let $\mathcal{S}$ be a specification not containing error sort symbols and let $\mathcal{S}'$ be a stratification of $\mathcal{S}$ with respect to $F = \emptyset$. Then an $\mathcal{S}$-equation is valid in $\mathcal{S}$ if and only if it is valid in $\mathcal{S}'$.*

Proof. Follows from the fact that $\mathcal{S}^\top = \mathcal{S}'^\top$. □

Next we will show that the initial algebras of $\mathcal{S}$ and $\mathcal{S}^\top$ are base isomorphic.

We call two specifications **base equivalent** if they have the same presignature and their initial algebras are base isomorphic.

Lemma 7.8. *Let $\mathcal{S}$ and $\mathcal{S}'$ be specifications that have the same presignature. Furthermore, let the following conditions be satisfied:*

- *there exists an $\mathcal{S}'$-algebra $\mathcal{A}$ and a base homomorphism $\alpha: \mathcal{A} \to \mathcal{I}_{\mathcal{S}}$*
- *there exists an $\mathcal{S}$-algebra $\mathcal{B}$ and a base homomorphism $\beta: \mathcal{B} \to \mathcal{I}_{\mathcal{S}'}$.*

Then $\mathcal{S}$ and $\mathcal{S}'$ are base equivalent.

Proof. Let all assumptions be satisfied. Because of the initiality of $\mathcal{I}_\mathcal{S}$ and $\mathcal{I}_{\mathcal{S}'}$, there exist base homomorphisms $\gamma: \mathcal{I}_\mathcal{S} \to \mathcal{B}$ and $\delta: \mathcal{I}_{\mathcal{S}'} \to \mathcal{A}$. The initiality of $\mathcal{I}_\mathcal{S}$ and $\mathcal{I}_{\mathcal{S}'}$ furthermore yields that $\mathrm{id}_{\mathcal{I}_\mathcal{S}} = (\alpha\delta)(\beta\gamma)$ and $\mathrm{id}_{\mathcal{I}_{\mathcal{S}'}} = (\beta\gamma)(\alpha\delta)$. Hence $\beta\gamma$ is a base isomorphism $\mathcal{I}_\mathcal{S} \to \mathcal{I}_{\mathcal{S}'}$. □

Theorem 7.9. (Base Equivalence) *If $\mathcal{S}$ is an admissible specification, then $\mathcal{S}$ and $\mathcal{S}^\top$ are base equivalent.*

Proof. Follows immediately from lemmas 7.3, 7.4 and 7.8. □

The equivalence theorems entitle us to consider $\mathcal{S}^\top$ rather than $\mathcal{S}$, which can be quite advantageous for deduction and rewriting. Furthermore, in $\mathcal{S}^\top$ meaningful terms like *fac*$(8-7)$ don't cause problems anymore since they are always well-sorted. Of course, $\mathcal{S}^\top$ is too permissive for the user interface of a specification or programming system since one would loose all benefits of sort checking. A practical solution preserving the benefits of sort checking is to allow the programmer writing expressions like *fac*$((8-7)\colon \mathbf{nat})$. With the assertion $(8-7)\colon \mathbf{nat}$ the programmer states that he knows by reasoning that is beyond the possibilities of the sort checker that $8-7$ in fact denotes an element in **nat**.

7.2 Lifting Completely Defined Functions

We now discuss a signature transformation that combines stratification with the construction of $\Sigma^\top$. This transformation is needed for a theorem that validates our approach to partial functions. It is also useful for practical applications since, when applied to rewriting systems, it can make rules sort decreasing.

Construction 7.10. (Σ^F and $\mathcal{S}^F$) Let Σ be a strict signature and F be a set of Σ-function symbols. The **lifted signature** Σ^F is defined as

$$\Sigma^F := \Sigma^\top - \{D \in \Sigma_\mathrm{B} \mid D \text{ contains a symbol of } F\}.$$

Furthermore, if $\mathcal{S} = (\Sigma, \mathcal{E})$ is an admissible specification, then $\mathcal{S}^F := (\Sigma^F, \mathcal{E})$.

Figure 7.1 shows an example for the lifting construction. Intuitively, lifting a function means to delete its sort declarations. We will prove that lifting completely defined functions does not change the initial algebra semantics. This means that the sort declarations of completely defined functions are redundant. In other words, completely defined functions are already completely defined by the equations of a specification.

Proposition 7.11. *Let $\mathcal{S}$ be a specification not containing error sort symbols, F be a set of $\mathcal{S}$-function symbols, and $\mathcal{S}'$ be a stratification of $\mathcal{S}$ with respect to F. Then $\mathcal{S}^F = \mathcal{S}'^\top \subseteq \mathcal{S}^\top$.*

Proposition 7.12. *Let $\mathcal{S}$ be an admissible specification and F be a set of $\mathcal{S}$-function symbols. Then $\mathcal{S}^F$ is an admissible specification and*

- *every $\mathcal{S}^F$-term of a basic sort ξ is an $\mathcal{S}$-term of sort ξ*
- *every $\mathcal{S}$-term of a basic sort ξ not containing symbols of F is an $\mathcal{S}^F$-term of sort ξ*
- *every $\mathcal{S}^\top$-algebra is an $\mathcal{S}^F$-algebra*
- *every admissible equation that is valid in $\mathcal{S}^F$ is valid in $\mathcal{S}^\top$, and every admissible $\mathcal{S}$-equation that is valid in $\mathcal{S}^F$ is valid in $\mathcal{S}$.*

Let $\mathcal{S}$ be an admissible specification and F be a set of $\mathcal{S}$-function symbols. We say that F is **completely defined in $\mathcal{S}$** if

for every declaration $f: \xi_1 \cdots \xi_n \to \xi$ of $\mathcal{S}$ such that $f \in F$ and ξ is basic, and

for every tuple $s_1, \ldots, s_n$ of ground $\mathcal{S}$-terms not containing symbols of F such that s_i has sort ξ_i for $i = 1, \ldots, n$

there exists a ground $\mathcal{S}$-term t of sort ξ not containing symbols of F such that $f(s_1, \ldots, s_n) =_{\mathcal{S}^F} t$.

This definition captures our intuition of what it means for a function to be completely defined by equations. Our goal is to show that specifying completely defined functions as partial functions (that is, stratifying with respect

Specification $\mathcal{S}$:

Variables: N, N': **nat**

$o\colon \to \mathbf{nat}$

$s\colon \mathbf{nat} \to \mathbf{nat}$

$+\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}$

$$o + N \doteq N$$
$$s(N) + N' \doteq s(N + N')$$

$true\colon \to \mathbf{bool}$

$false\colon \to \mathbf{bool}$

$\leq\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{bool}$

$$o \leq N \doteq true$$
$$s(N) \leq o \doteq false$$
$$s(N) \leq s(N') \doteq N \leq N'$$

Specification $\mathcal{S}^F$, where $F = \{+, \leq\}$:

Variables: N, N': **nat**

$\mathbf{nat} < \top$

$o\colon \to \mathbf{nat},\ o\colon \to \top$

$s\colon \mathbf{nat} \to \mathbf{nat},\ s\colon \top \to \top$

$+\colon \top \times \top \to \top$

$$o + N \doteq N$$
$$s(N) + N' \doteq s(N + N')$$

$\mathbf{bool} < \top$

$true\colon \to \mathbf{bool},\ true\colon \to \top$

$false\colon \to \mathbf{bool},\ false\colon \to \top$

$\leq\colon \top \times \top \to \top$

$$o \leq N \doteq true$$
$$s(N) \leq o \doteq false$$
$$s(N) \leq s(N') \doteq N \leq N'$$

Figure 7.1. Lifting Completely Defined Functions. The declarations $o\colon \to\ \top$, $true\colon \to\ \top$ and $false\colon \to\ \top$ are actually redundant; however, the declaration $s\colon \top \to \top$ is needed to extend s to $\top$.

to them) does not change the initial algebra semantics of a specification. This result will be the major validation of the error supersorts approach to partial functions.

Lemma 7.13. *Let F be completely defined in an admissible specification $\mathcal{S}$.*

Then the initial algebra $\mathcal{I}_{\mathcal{S}^F}$ is an $\mathcal{S}^\top$-algebra.

Proof. Let $f\colon \xi_1 \cdots \xi_n \to \xi$ be a declaration of $\mathcal{S}$ such that $f \in F$ and $\xi_1, \ldots, \xi_n$ and ξ are basic. It suffices to show that $\mathcal{I}_{\mathcal{S}^F}$ satisfies the declaration $f\colon \xi_1 \cdots \xi_n \to \xi$. Let $a_i \in \xi_i^{\mathcal{I}_{\mathcal{S}^F}}$ for $i = 1, \ldots, n$. Since $\mathcal{S}^F$ contains the declarations $f\colon \top \cdots \top \to \top$ and $\xi_1 < \top, \ldots, \xi_n < \top$, we know that $(a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{I}_{\mathcal{S}^F}}$. Since $\mathcal{I}_{\mathcal{S}^F}$ has no junk, there exist ground $\mathcal{S}^F$-terms $s_1, \ldots, s_n$ such that $[\![s_i]\!]_{\mathcal{I}_{\mathcal{S}^F}} = a_i$ and s_i has sort ξ_i for $i = 1, \ldots, n$. Since $\xi_1, \ldots, \xi_n$ are basic, s_i is a ground $\mathcal{S}$-term of sort ξ_i for $i = 1, \ldots, n$. Since F is completely defined in $\mathcal{S}$, we know that there exists a ground $\mathcal{S}$-term t of sort ξ not containing symbols of F such that $f(s_1, \ldots, s_n) =_{\mathcal{S}^F} t$. Since t doesn't contain symbols of F, t is also an $\mathcal{S}^F$-term of sort ξ. Hence we have $f^{\mathcal{I}_{\mathcal{S}^F}}(a_1, \ldots, a_n) = f^{\mathcal{I}_{\mathcal{S}^F}}([\![s_1]\!]_{\mathcal{I}_{\mathcal{S}^F}}, \ldots, [\![s_n]\!]_{\mathcal{I}_{\mathcal{S}^F}}) = [\![f(s_1, \ldots, s_n)]\!]_{\mathcal{I}_{\mathcal{S}^F}} = [\![t]\!]_{\mathcal{I}_{\mathcal{S}^F}} \in \xi^{\mathcal{I}_{\mathcal{S}^F}}$. □

Theorem 7.14. (Base Equivalence) *Let F be completely defined in an admissible specification $\mathcal{S}$. Then $\mathcal{S}$, $\mathcal{S}^\top$ and $\mathcal{S}^F$ are base equivalent.*

Proof. Since every $\mathcal{S}^\top$-algebra is an $\mathcal{S}^F$-algebra, we know that $\mathcal{I}_{\mathcal{S}^\top}$ is an $\mathcal{S}^F$-algebra. Furthermore, $\mathrm{id}_{\mathcal{I}_{\mathcal{S}^\top}}$ is a base homomorphism $\mathcal{I}_{\mathcal{S}^\top} \to \mathcal{I}_{\mathcal{S}^\top}$. By the preceding lemma we know that $\mathcal{I}_{\mathcal{S}^F}$ is an $\mathcal{S}^\top$-algebra. Furthermore, $\mathrm{id}_{\mathcal{I}_{\mathcal{S}^F}}$ is a base homomorphism $\mathcal{I}_{\mathcal{S}^F} \to \mathcal{I}_{\mathcal{S}^F}$. Hence we know by Lemma 7.8 that $\mathcal{S}^\top$ and $\mathcal{S}^F$ are base equivalent. Since we know by Theorem 7.9 that $\mathcal{S}$ and $\mathcal{S}^\top$ are base equivalent, we know that $\mathcal{S}$ and $\mathcal{S}^F$ are base equivalent. □

Corollary 7.15. *Let $\mathcal{S}$ be a specification not containing error sort symbols and let F be completely defined in $\mathcal{S}$. Furthermore, let $\mathcal{S}'$ be a stratification of $\mathcal{S}$ with respect to F. Then $\mathcal{S}$, $\mathcal{S}'$, $\mathcal{S}^\top$, and $\mathcal{S}^F$ are base equivalent.*

Proof. Follows from the preceding theorem and the fact that $\mathcal{S}^F = \mathcal{S}'^\top$. □

Theorem 7.16. *Let F be completely defined in an admissible specification*

$\mathcal{S}$. Then, for every ground $\mathcal{S}$-term s of a basic sort ξ, there exists a ground $\mathcal{S}$-term t of sort ξ not containing symbols of F such that $s =_{\mathcal{S}^F} t$ and $s =_{\mathcal{S}} t$.

Proof. Let $s = f(s_1, \ldots, s_n)$ be a ground $\mathcal{S}$-term of a basic sort ξ. We prove by induction on the term structure of s that there exists a ground $\mathcal{S}$-term t of sort ξ not containing symbols of F such that $s =_{\mathcal{S}^F} t$. If s doesn't contain a symbol of F, then the claim is trivial. Otherwise, $\mathcal{S}$ contains a declaration $f\colon \eta_1 \cdots \eta_n \to \eta$ such that $\eta \leq \xi$ and s_i is an $\mathcal{S}$-term of sort η_i for $i = 1, \ldots, n$. Since $\mathcal{S}$ is admissible and ξ is basic, we know that $\eta_1, \ldots, \eta_n$ are basic. Hence we know by the induction hypothesis that there exist ground $\mathcal{S}$-terms $t_1, \ldots, t_n$ not containing symbols of F such that $s_i =_{\mathcal{S}^F} t_i$ and t_i has sort η_i in $\mathcal{S}$ for $i = 1, \ldots, n$. Hence $f(t_1, \ldots, t_n)$ is an $\mathcal{S}$-term of sort ξ such that $s =_{\mathcal{S}^F} f(t_1, \ldots, t_n)$. If $f \notin F$, we have the claim immediately. If $f \in F$, then we have the claim since F is completely defined in $\mathcal{S}$. Furthermore, by Proposition 7.12 we know that every admissible $\mathcal{S}$-equation that is valid in $\mathcal{S}^F$ is valid in $\mathcal{S}$. □

Theorem 7.17. (Equivalence of Initial Validity) *Let F be completely defined in an admissible specification $\mathcal{S}$. Then*

$$\mathcal{I}_{\mathcal{S}^\top} \models s \doteq t \quad \iff \quad \mathcal{I}_{\mathcal{S}^F} \models s \doteq t$$

for every admissible $\mathcal{S}^\top$-equation $s \doteq t$ and

$$\mathcal{I}_{\mathcal{S}} \models s \doteq t \quad \iff \quad \mathcal{I}_{\mathcal{S}^F} \models s \doteq t$$

for every admissible $\mathcal{S}$-equation $s \doteq t$.

Proof. 1. Let $s \doteq t$ be an $\mathcal{S}^\top$-equation that is valid in $\mathcal{I}_{\mathcal{S}^\top}$. To show that $s \doteq t$ is valid in $\mathcal{I}_{\mathcal{S}^F}$, it suffices to show that every ground $\mathcal{S}^F$-instance of $s \doteq t$ is valid in $\mathcal{I}_{\mathcal{S}^F}$. Let $u \doteq v$ be a ground $\mathcal{S}^F$-instance of $s \doteq t$. Since $u \doteq v$ is also an $\mathcal{S}^\top$-instance of $s \doteq t$ and we assumed $s \doteq t$ to be valid in $\mathcal{I}_{\mathcal{S}^\top}$, $u \doteq v$ is valid in $\mathcal{I}_{\mathcal{S}^\top}$. Hence we know that $u \doteq v$ is valid in $\mathcal{S}^\top$ since $u \doteq v$ is ground. Since we know by Lemma 7.13 that $\mathcal{I}_{\mathcal{S}^F}$ is an $\mathcal{S}^\top$-algebra, we have that $u \doteq v$ is valid in $\mathcal{I}_{\mathcal{S}^F}$.

2. Let $s \doteq t$ be an admissible $\mathcal{S}^{\top}$-equation that is valid in $\mathcal{I}_{\mathcal{S}^F}$. To show that $s \doteq t$ is valid in $\mathcal{I}_{\mathcal{S}^{\top}}$, it suffices to show that every ground $\mathcal{S}^{\top}$-instance of $s \doteq t$ is valid in $\mathcal{I}_{\mathcal{S}^{\top}}$. Let $\theta s \doteq \theta t$ be a ground $\mathcal{S}^{\top}$-instance of $s \doteq t$. By the preceding theorem we know that there exists a substitution ψ such that $\psi s \doteq \psi t$ is a ground $\mathcal{S}^F$-instance of $s \doteq t$ and $\theta x =_{\mathcal{S}^{\top}} \psi x$ for all $x \in \mathcal{V}(s \doteq t)$. We know that $\psi s \doteq \psi t$ is valid in $\mathcal{I}_{\mathcal{S}^F}$ since $s \doteq t$ is valid in $\mathcal{I}_{\mathcal{S}^F}$. Hence we know that $\psi s \doteq \psi t$ is valid in $\mathcal{S}^F$ since $\psi s \doteq \psi t$ is ground. Thus we know by Proposition 7.12 that $\psi s \doteq \psi t$ is valid in $\mathcal{S}^{\top}$. Since we already know that $\theta x =_{\mathcal{S}^{\top}} \psi x$ for all $x \in \mathcal{V}(s \doteq t)$, we know that $\theta s \doteq \theta t$ is valid in $\mathcal{S}^{\top}$. Hence $\theta s \doteq \theta t$ is valid in $\mathcal{I}_{\mathcal{S}^{\top}}$.

3. The second equivalence follows from the first equivalence and Corollary 7.6. □

Corollary 7.18. *Let $\mathcal{S}$ be a specification not containing error sort symbols and let F be completely defined in $\mathcal{S}$. Furthermore, let $\mathcal{S}'$ be a stratification of $\mathcal{S}$ with respect to F. Then an $\mathcal{S}$-equation is valid in $\mathcal{I}_{\mathcal{S}}$ if and only if it is valid in $\mathcal{I}_{\mathcal{S}'}$.*

Proof. Follows from the preceding theorem and the fact that $\mathcal{S}^F = \mathcal{S}'^{\top}$. □

From the example in Figure 7.1 one can see that the semantic equivalence of $\mathcal{S}$ and $\mathcal{S}^F$ is a rather powerful result. It says that declarations of *constructors*, that is, functions that generate data elements (for instance, $s\colon \mathbf{nat} \to \mathbf{nat}$), contribute to the structure of the initial algebra, while declarations of *extending functions*, that is, completely defined or partial functions (for instance, $+\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}$), are semantically redundant. The purpose of the declarations for extending functions is to set up an appropriate sort checking discipline. Furthermore, they are consistency constraints for the specification, that is, they must be satisfied by the initial algebra if the specification is "correct". Checking whether functions that don't contribute data elements are completely defined is an important validation of a specification. See Comon [86] for a method for automatically checking complete definedness (in a framework without subsorts) and for further references to this topic. In general, of course, complete definedness is undecidable.

7.3 Order-Sorted Rewriting Revisited

The signature transformations discussed in the preceding subsections are very useful for order-sorted rewriting systems. First of all, we don't have to worry about compatibility anymore since $\mathcal{R}^\top$ is always compatible and equivalent to $\mathcal{R}$. Furthermore, the transformation $\mathcal{R}^F$ provides a partial solution for the sort decreasingness problem sufficing for many practical applications.

Proposition 7.19. *Let $\mathcal{R}$ be an admissible rewriting system. Then $\mathcal{R}^\top$ is a compatible rewriting system having the same overlaps as $\mathcal{R}$ and every critical pair of $\mathcal{R}$ is a critical pair of $\mathcal{R}^\top$. Furthermore, if $s \to_{\mathcal{R}}^* t$, then $s \to_{\mathcal{R}^\top}^* t$.*

Example 7.20. Let $\mathcal{R}$ be the rewriting system

$$\mathbf{A} < \mathbf{B}, \quad a{:} \to \mathbf{A}, \quad f{:}\, \mathbf{A} \to \mathbf{A}, \quad b{:} \to \mathbf{B}$$
$$a \to b, \quad f(a) \to b.$$

Then $(f(a) \to b,\ 1,\ a \to b)$ is an overlap of $\mathcal{R}$ and $\mathcal{R}^\top$. In $\mathcal{R}$ this overlap has no critical pair since $f(b)$ is not an $\mathcal{R}$-term, while in $\mathcal{R}^\top$ this overlap has the critical pair $(b, f(b))$.

Proposition 7.21. *Let $\mathcal{R}$ be an admissible rewriting system. Then $\mathcal{R}^\top$ is sort decreasing if $\mathcal{R}$ is sort decreasing. Furthermore, $\mathcal{R}^\top$ is sort decreasing if and only if every rule of $\mathcal{R}$ whose left-hand side doesn't contain a nonbasic function symbol is sort decreasing in $\mathcal{R}^\top$.*

Proposition 7.22. *Let F be completely defined in an admissible rewriting system $\mathcal{R}$ and let P be the set of the nonbasic function symbols of $\mathcal{R}$. Then $\mathcal{R}^F$ is sort decreasing if*

1. *every rule of $\mathcal{R}$ that does not contain a function symbol of F or P is sort decreasing in $\mathcal{R}$*

2. *every rule of $\mathcal{R}$ that contains a function symbol of F or P contains a function symbol of F or P in its left-hand side.*

In practice it turns out that $\mathcal{R}^F$ is often sort decreasing although $\mathcal{R}$ cannot be made sort decreasing. Sort decreasingness is important since it is needed to automatically check the confluence of a specification. Below we will show that $\mathcal{R}^\top$ is ground confluent if $\mathcal{R}^F$ is ground confluent. Thus, if this is preferable, one can rewrite in $\mathcal{R}^\top$ and use $\mathcal{R}^F$ just for checking confluence.

Of course, checking the confluence of $\mathcal{R}^F$ to establish the ground confluence of $\mathcal{R}^\top$ is only possibly if one puts in the knowledge that F is completely defined, a property that is undecidable in general. However, complete definedness is a semantic property that is independent of the particular equations used in a specification, while ground confluence is an operational property depending on the particular equations used in a specification. Hence $\mathcal{R}^F$ allows us trading the automatic verification of an operational property for the assertion of a semantic property.

Example 7.23. Kirchner et al. [87] give a specification of the complex rational numbers as an order-sorted rewriting system. They define the square of the absolute value of a complex number by

$$\begin{aligned}&f\colon \mathbf{complex} \to \mathbf{rational}\\ &\quad f(C) \doteq C * \mathit{conjugate}(C),\end{aligned}$$

where **rational** is a subsort of **complex** and $*$ and *conjugate* come with the declarations

$$\begin{aligned}&*\colon \mathbf{complex} \times \mathbf{complex} \to \mathbf{complex}\\ &\mathit{conjugate}\colon \mathbf{complex} \to \mathbf{complex}.\end{aligned}$$

It is obvious that the rewrite rule defining f cannot be made sort decreasing by adding further declarations. However, since f is completely defined, the rule can be made sort decreasing by lifting f to $\top$.

Proposition 7.24. *Let $\mathcal{R}$ be an admissible rewriting system and F be a set of $\mathcal{R}$-function symbols. Then $s \rightarrow^*_{\mathcal{R}^\top} t$ if $s \rightarrow^*_{\mathcal{R}^F} t$. Furthermore, every critical pair of $\mathcal{R}^F$ is a critical pair of $\mathcal{R}^\top$.*

Although $\mathcal{R}^F$ and $\mathcal{R}^\top$ have the same rules, the converse of the proposition doesn't hold since in $\mathcal{R}^F$-derivations the variables of a rewrite rule cannot be instantiated to terms containing function symbols of F because such terms have only the sort $\top$ and the variables in the rewrite rules range over basic sorts. Thus the lifted system $\mathcal{R}^F$ admits only those $\mathcal{R}^\top$-derivations that are, roughly speaking, innermost with respect to F. This is of pratical interest since innermost rewriting can be implemented more efficiently than general rewriting. Furthermore, narrowing with the lifted system $\mathcal{R}^F$ has a smaller search space than narrowing with $\mathcal{R}^\top$ since the rewrite rules have fewer instances in $\mathcal{R}^F$ than they have in $\mathcal{R}^\top$.

Nevertheless, the following theorem tells us that once we have established the ground confluence of $\mathcal{R}^F$ we can as well rewrite in $\mathcal{R}^\top$.

Theorem 7.25. *Let F be completely defined in an admissible rewriting system $\mathcal{R}$. Then $\mathcal{R}^\top$ is ground confluent if $\mathcal{R}^F$ is ground confluent.*

Proof. Let $\mathcal{R}^F$ be ground confluent. To show that $\mathcal{R}^\top$ is ground confluent, suppose that $s \rightarrow^*_{\mathcal{R}^\top} u$ and $s \rightarrow^*_{\mathcal{R}^\top} v$, where s, u and v are ground terms. By Theorem 7.17 we know that the equations $s \doteq u$ and $s \doteq v$ are valid in $\mathcal{R}^F$. Since $\mathcal{R}^F$ is ground confluent and compatible, we know $s{\downarrow_{\mathcal{R}^F}}u$ and $s{\downarrow_{\mathcal{R}^F}}v$. Hence $u{\downarrow_{\mathcal{R}^F}}v$ by the ground confluence of $\mathcal{R}^F$. Thus $u \downarrow_{\mathcal{R}^\top} v$ since $\mathcal{R}^F \subseteq \mathcal{R}^\top$. □

7.4 Consistency and Strictness of Rewriting Systems

Here we give decidable criteria for the consistency and strictness (defined in Subsection 5.1) of rewriting system with partial functions.

Theorem 7.26. (Consistency) *Let $\mathcal{R}$ be an admissible rewriting system such that*

1. *the left-hand side of every nonbasic rule of $\mathcal{R}$ is nonbasic in $\mathcal{R}$*

2. *$\mathcal{R}^{\top}$ is ground confluent.*

Then $\mathcal{R}$ and $\mathcal{R}^{\top}$ are consistent.

Proof. Let $s \doteq t$ be a ground $\mathcal{R}_{\mathrm{B}}$-equation that is valid in $\mathcal{R}$. By Theorem 7.5 we know that $s \doteq t$ is valid in $\mathcal{R}^{\top}$. Since $\mathcal{R}^{\top}$ is ground confluent and compatible, we know that there exists an $\mathcal{R}^{\top}$-term u such that $s \to^{*}_{\mathcal{R}^{\top}} u$ and $t \to^{*}_{\mathcal{R}^{\top}} u$. Since s and t are basic $\mathcal{R}$-terms and the left-hand side of every nonbasic rule is nonbasic, we know that $s \to^{*}_{\mathcal{R}^{\top}_{\mathrm{B}}} u$ and $t \to^{*}_{\mathcal{R}^{\top}_{\mathrm{B}}} u$. Thus $s \doteq t$ is valid in $\mathcal{R}^{\top}_{\mathrm{B}}$. Hence we know by Theorem 7.5 that $s \doteq t$ is valid in $\mathcal{R}_{\mathrm{B}}$. □

Proposition 7.27. *Let $\mathcal{R}$ be an admissible rewriting system. Then an admissible $\mathcal{R}$-term is sensible in $\mathcal{R}$ if and only if it is sensible in $\mathcal{R}^{\top}$. Furthermore, $\mathcal{R}$ is a strict specification if $\mathcal{R}^{\top}$ is a strict specification.*

Proof. Follows from Propositon 7.2 and Theorem 7.5. □

Let Σ be a strict signature. We call a Σ-term **simple in** Σ if it has the form $f(s_1, \ldots, s_n)$, where f is nonbasic and $s_1, \ldots, s_n$ are basic in Σ. Every Σ-instance of a simple Σ-term is simple in Σ.

We call an admissible rewriting system $\mathcal{R}$ **respectful** if the left-hand side of every nonbasic rule of $\mathcal{R}$ is simple in $\mathcal{R}$.

Lemma 7.28. *Let $\mathcal{R}$ be a respectful rewriting system. Furthermore, let s be a ground $\mathcal{R}^{\top}$-term and t be a basic $\mathcal{R}^{\top}$-term such that $s \to^{*}_{\mathcal{R}^{\top}} t$. Then, for every subterm s/π of s, there exists a basic $\mathcal{R}^{\top}$-term u such that $s/\pi \to^{*}_{\mathcal{R}^{\top}} u$.*

Proof. Let s/π be a subterm of s. We show by induction on the length of the derivation $s \to^{*}_{\mathcal{R}^{\top}} t$ that there exists a basic $\mathcal{R}^{\top}$-term u such that $s/\pi \to^{*}_{\mathcal{R}^{\top}} u$. If $s = t$, then the claim is trivial. Otherwise, there exists a term s' such that $s \to_{\mathcal{R}^{\top}} s' \to^{*}_{\mathcal{R}^{\top}} u$, where $s \to_{\mathcal{R}^{\top}} s'$ by a rewrite step at position π' of s. If π is below π', then s/π is basic since s/π' is simple because $\mathcal{R}$ is respectful. Otherwise, there exists a subterm s'' of s' such that $s/\pi \rightarrow_{\mathcal{R}^\top} s''$. By the induction hypothesis we know that there exists a basic $\mathcal{R}^\top$-term u such that $s'' \rightarrow^*_{\mathcal{R}^\top} u$. Hence $s/\pi \rightarrow^*_{\mathcal{R}^\top} u$. □

Theorem 7.29. **(Strictness)** *Let $\mathcal{R}$ be a respectful rewriting system such that $\mathcal{R}^\top$ is ground confluent and every basic rule of $\mathcal{R}$ is sort decreasing. Then $\mathcal{R}^\top$ and $\mathcal{R}$ are strict specifications.*

Proof. Because of Proposition 7.27 it suffices to show that $\mathcal{R}^\top$ is a strict specification. Let s be a ground and sensible $\mathcal{R}^\top$-term. We have to show that every subterm of s is sensible in $\mathcal{R}^\top$. Since s is sensible, there exists a ground and basic $\mathcal{R}^\top$-term t such that $s =_{\mathcal{R}^\top} t$. Since $\mathcal{R}^\top$ is ground confluent, there exists an $\mathcal{R}^\top$-term u such that $s \rightarrow^*_{\mathcal{R}^\top} u$ and $t \rightarrow^*_{\mathcal{R}^\top} u$. Since t is basic in $\mathcal{R}^\top$ and every basic rule of $\mathcal{R}$ is sort decreasing, we know that u is a basic $\mathcal{R}^\top$-term. Hence we know by the preceding lemma that every subterm of s is sensible in $\mathcal{R}^\top$. □

Example 7.30. Let $\mathcal{R}$ be a stratification of the specification in Figures 5.3 and 5.4 with respect to the functions declared as partial (including the auxiliary functions for the conditionals). Then $\mathcal{R}$ is obviously a sort decreasing and respectful rewriting system. Furthermore, we know by a theorem in [Huet 80] that $\mathcal{R}^\top$ is confluent since it has no overlaps and all left-hand sides are linear (that is, no variable occurs more than once). (Of course, the theorem in [Huet 80] is only proven for unsorted rewriting; so you have to believe us that is also holds for sort decreasing rewriting.) Thus $\mathcal{R}$ is a consistent and strict specification. Since $==$, $\leq$ and *update* are completely defined, one could in addition stratify with respect to them without changing the initial algebra semantics. In this case one would obtain an equation-free base specification just consisting of the signature equations in Figure 5.3.

References

H. Aït-Kaci, An Algebraic Semantics Approach to the Effective Resolution of Type Equations. Theoretical Computer Science 45, 1986, 293–351.

H. Aït-Kaci and R. Nasr, LOGIN: A Logic Programming Language with Built-In Inheritance. The Journal of Logic Programming, 1986, 3, 185–215.

M. Broy and M. Wirsing, Partial Abstract Types. Acta Informatica 18, 1982, 47–64.

A. G. Cohn, Improving the Expressiveness of Many-Sorted Logic. Proc. of the National Conference on Artificial Intelligence, 1983, W. Kaufmann, 84–87.

A. G. Cohn, On the Solution of Schubert's Steamroller in Many-Sorted Logic. Proc. 9th International Joint Conference on Artificial Intelligence, 1985, W. Kaufmann, 1169–1174.

H. Comon, Sufficient Completeness, Term Rewriting Systems and "Anti-Unification". Proc. 8th International Conference on Automated Deduction, 1986, Springer LNCS 230, 128–140.

R.J. Cunningham and A.J.J. Dick, Rewrite Systems on a Lattice of Types. Acta Informatica 22, 1985, 149–169.

H. Ehrig and B. Mahr, Fundamentals of Algebraic Specification 1, Equations and Initial Semantics. Springer Verlag, 1985.

F. Fages and G. Huet, Complete sets of Unifiers and Matchers in Equational Theories. Theoretical Computer Science 43 (2,3), 1986, 189–200.

K. Futatsugi, J.A. Goguen, J.-P. Jouannaud and J. Meseguer, Principles of OBJ2. Proc. POPL 1985, 52–66.

I. Gnaedig, C. Kirchner and H. Kirchner, Equational Completion in Order-sorted Algebras. Report 87R086, CRIN, Vandoeuvre les Nancy, France, 1987.

M. Gogolla, Algebraic Specifications with Partially Ordered Sorts and Declarations. Forschungsbericht 169, Informatik, Universität Dortmund, West Germany, 1983.

M. Gogolla, Über partiell geordnete Sortenmengen und deren Anwendung zur Fehlerbehandlung in abstrakten Datentypen. PhD Thesis, FB Informatik, Universität Braunschweig, West Germany, 1986.

J.A. Goguen, J.W. Thatcher, and E.G. Wagner, An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types. In R.T. Yeh (ed.), Current Trends in Programming Methodology, Volume IV, Data Structuring; Prentice Hall, 1978, 80–149.

J.A. Goguen, Order Sorted Algebra. Semantics and Theory of Computation Report No. 14, UCLA Computer Science Department, 1978.

J.A. Goguen, Some Design Principles and Theory of OBJ0, a Language for Expressing and Executing Algebraic Specifications of Programs. Proc. International Conference on Mathematical Studies of Information Processing, Kyoto, Japan, Springer LNCS 75, 1979, 425–473.

J.A. Goguen, J.-P. Jouannaud and J. Meseguer, Operational Semantics of Order-Sorted Algebra. Proc. ICALP 1985, Springer LNCS 194, 221–231.

J.A. Goguen and J. Meseguer, Eqlog: Equality, Types, and Generic Modules for Logic Programming. In D. DeGroot and G. Lindstrom (eds.), Logic Programming, Functions, Relations, and Equations; Prentice Hall 1986.

J.A. Goguen and J. Meseguer, Models and Equality for Logic Programming. TAPSOFT '87, Pisa, Springer LNCS 250, 1987a, 1–22.

J.A. Goguen and J. Meseguer, Order-Sorted Algebra Solves the Constructor-Selector, Multiple Representation and Coercion Problems. Second IEEE Symposium on Logic in Computer Science, Ithaca, 1987b, 18–29.

J.A. Goguen and J. Meseguer, Order-Sorted Algebra I: Partial and Overloaded Operators, Errors and Inheritance. To appear, Computer Science Lab, SRI International, 1987c.

J.A. Goguen and J. Meseguer, Unifying Functional, Object-Oriented and Relational Programming with Logical Semantics. In B. Shriver and P. Wegner (eds.), Research Directions in Object-Oriented Programming; MIT Press, 1987d, 417-477.

G. Huet, Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. Journal of the ACM 27,4, 1980, 797–821.

G. Huet and D.C. Oppen, Equations and Rewrite Rules: A Survey. In R. Book (ed.), Formal Languagues: Perspectives and Open Problems; Academic Press, 1980, 349–405.

K.B. Irani and D.G. Shin, A Many-Sorted Resolution Based on an Extension of a First-Order Language. Proc. 9th International Joint Conference on Artificial Intelligence, 1985, W. Kaufmann, 1175–1177.

S. Kamin and M. Archer, Partial Implementations of Abstract Data Types: A Dissenting View on Errors. In G. Kahn, D. MacQueen and G. Plotkin (eds.), Semantics of Data Types; Springer Verlag, LNCS 173, 1984, 317–336.

S. Kaplan, Conditional Rewrite Rules. Theoretical Computer Science 33, 1984, 175–193.

C. Kirchner, H. Kirchner, J. Meseguer, Operational Semantics of OBJ3. Report 87R87, CRIN, Vandoeuvre les Nancy, France, 1987.

A. Martelli and U. Montanari, An Efficient Unification Algorithm. ACM Transactions on Programming Languagues and Systems 4,2, 1982, 258–282.

J. Meseguer and J.A. Goguen, Initiality, Induction, and Computability. In M. Nivat and J.C. Reynolds (eds.), Algebraic Methods in Semantics; Cambridge University Press, 1985a.

J. Meseguer and J.A. Goguen, Deduction with Many-Sorted Rewrite Rules. To appear in Theoretical Computer Science. Report CSLI-85-42, Center for the Study of Language and Information, Stanford University, 1985b.

J. Meseguer, J.A. Goguen, and G. Smolka, Order-Sorted Unification. Report CSLI-87-86, Center for the Study of Language and Information, Stanford University, 1987.

M.J. O'Donnell, Equational Logic as a Programming Language. The MIT Press, 1985.

A. Oberschelp, Untersuchungen zur mehrsortigen Quantorenlogik. Math. Annalen 145, 1962, 297–333.

M.S. Paterson and M.N. Wegman, Linear Unification. Journal of Computer and System Science 16, 1978, 158–167.

A Poigné, Another Look at Parameterization Using Algebraic Specifications with Subsorts. Proc. of the Conference on Mathematical Foundations of Computer Science, Praha, Springer Verlag, LNCS 176, 1984.

H. Reichel, Initially Restricting Algebraic Theories. In P. Dembinski (ed.), Proc. of the Conference on Mathematical Foundations of Computer Science; Springer Verlag, LNCS 88, 1980, 504-514.

H. Reichel, Initial Computability, Algebraic Specifications, and Partial Algebras. Akademie-Verlag Berlin and Oxford University Press, 1987.

J.A. Robinson, A Machine-Oriented Logic Based on the Resolution Principle. Journal of the ACM 12, 1965, 23–41.

M. Schmidt-Schauß, A Many-Sorted Calculus with Polymorphic Functions Based on Resolution and Paramodulation. Proc. 9th International Joint Conference on Artificial Intelligence, 1985a, W. Kaufmann, 1162–1168.

M. Schmidt-Schauß, Unification in a Many-Sorted Calculus with Declarations. Proc. of German Workshop on Artificial Intelligence, 1985b, Springer Informatik Fachberichte 118, 118–132.

M. Schmidt-Schauß, Unification in Many-Sorted Equational Theories. Proc. 8th International Conference on Automated Deduction, 1986, Springer LNCS 230, 538–552.

M. Schmidt-Schauß, Computational Aspects of an Order-Sorted Logic with Term Declarations. PhD Thesis, FB Informatik, Universität Kaiserslautern, West Germany, 1987.

J.H. Siekmann, Unification Theory. Proc. 7th European Conference on Artificial Intelligence, 1986, vi–xxxv.

G. Smolka, Order-Sorted Horn Logic: Semantics and Deduction. SEKI Report

SR-86-17, FB Informatik, Universität Kaiserslautern, West Germany, 1986.

G. Smolka and H. Aït-Kaci, Inheritance Hierarchies: Semantics and Unification. MCC Report AI-057-87, MCC, Austin, Texas, 1987.

G. Smolka, TEL (Version 0.9), Report and User Manual. SEKI Report SR-87-11, FB Informatik, Universität Kaiserslautern, West Germany, 1987.

C. Walther, A Many-Sorted Calculus Based on Resolution and Paramodulation. Proc. 8th International Joint Conference on Artificial Intelligence, 1983, W. Kaufmann, 882–891.

C. Walther, Unification in Many-Sorted Theories. Proc. 6th European Conference on Artificial Intelligence, 1984, North-Holland, 383–392.

C. Walther, A Mechanical Solution of Schubert's Steamroller by Many-Sorted Resolution. Artificial Intelligence 26,2, 1985, 217–224.

C. Walther, A Classification of Many-Sorted Unification Problems. Proc. 8th International Conference on Automated Deduction, 1986, Springer LNCS 230, 525–537.

C. Walther, A Many-sorted Calculus Based on Resolution and Paramodulation. Pitman and Morgan Kaufman Publishers, Research Notes in Artificial Intelligence, 1987.