Logic Programming over Polymorphically Order-Sorted Types

Gert Smolka

German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
smolka@dfki.uni-sb.de

Dissertation
Fachbereich Informatik
Universität Kaiserslautern
Kaiserslautern
Germany

3 May 1989

At the end, we had something complete
that made everything obvious, that made
us realize how we should have attacked
the problem. It takes a long time to realize
the right form.

John E. Hopcroft

From K. A. Frenkel, An Interview with the 1986 A.M. Turing Award Recipients—John E. Hopcroft and Robert E. Tarjan. Communications of the ACM 30, 3, 1987, 214–222.

Abstract

This thesis presents the foundations for relational logic programming over polymorphically order-sorted data types. This type discipline combines the notion of parametric polymorphism, which has been developed for higher-order functional programming, with the notion of order-sorted typing, which has been developed for equational first-order specification and programming. Polymorphically order-sorted types are obtained as canonical models of a class of specifications in a suitable logic accommodating sort functions. Algorithms for constraint solving, type checking and type inference are given and proven correct.

Acknowledgements

I would like to thank: Jörg Siekmann for his enthusiasm, openness, unfailing support and trust over many years; once more Jörg for writing all these grant proposals whose success was the base for the exciting, challenging and inspiring research environment my colleagues and I enjoyed in Kaiserslautern (my colleagues still do); Joseph Goguen and José Meseguer for getting me involved with order-sorted algebra and OBJ, which have provided technical guidance and inspiration for the research reported in this thesis; once more Joseph and José for two great summers at SRI International that opened up so many things for me; Werner Nutt for working with me on Order-Sorted Logic and implementing our programming language TEL; once more Werner for reading and discussing some of this thesis under great time pressure; Markus Höhfeld for working with me on constraint logic programming; Hans-Jürgen Bürckert, Richard Göbel and Manfred Schmidt-Schauß for the many inspiring and challenging discussions we had in Kaiserslautern on unification and rewriting; Alan Demers and Bob Constable for putting up with a student who stubbornly insisted on doing his own research; my colleagues in the LILOG and Protos projects at IBM in Stuttgart for providing the pleasant and inspiring environment in which I finally finished this thesis; and last not least Harald Ganzinger, Jean-Pierre Jouannaud and Jörg Siekmann for giving their expert opinions on this thesis.

Contents

1 Introduction **1**

1.1 The Problem Solved 1

1.2 Related Work 4

1.3 Overview 6

2 Logic Programming over Constraint Languages **7**

2.1 Constraint Languages 10

2.2 Canonical Extensions 13

2.3 Definite Clauses 15

2.4 Operational Semantics 17

2.5 A Type Discipline 21

2.6 Type Inference 23

3 POS-Logic **27**

3.1 The Category of POS-Algebras 28

3.2 POS-Constraints 30

3.3 The Substitution Theorem 32

3.4 Sort Rewriting 34

3.5 Quasi-Extensional Algebras 34

3.6 Simple Specifications 36

4 Sort Rewriting Systems **45**

4.1 Shallow Rewriting Systems 46

4.2 Upper Matchers and Suprema 51

4.3 Lower Matchers and Infima 55

4.4 Modes 59

5 POS-Types **63**

5.1 Type Specifications 64
5.2 Inhabitation 69
5.3 Unifiers and Solution Schemata 75

6 POS-Constraint Solving 81
6.1 The Constraint Solver 83
6.1.1 Approximations 83
6.1.2 The Constraint Solving Rules 85
6.1.3 Pyramids 87
6.1.4 The Hauptsatz 89
6.2 Well-Typedness in Pyramids 90
6.3 Approximations 94
6.4 P-Infima and Mergings 97
6.5 Solving PM-Systems 100
6.6 Solving PE-Systems 106
6.7 Proof of the Hauptsatz 111

7 Logic Programming over POS-Types 113
7.1 POS-Programs 114
7.2 The Interpreter 117
7.3 Type Inference 123

A Mathematical Preliminaries 129

Chapter 1

Introduction

1.1 The Problem Solved
1.2 Related Work
1.3 Overview

1.1 The Problem Solved

This thesis presents the foundations for relational logic programming over polymorphically order-sorted data types (called POS-types, hereafter). This type discipline combines the notion of parametric polymorphism [Mil78, DM82], which has been developed for higher-order functional programming [HMM86], with the notion of order-sorted typing [Gog78, GM87a, SNGM89], which has been developed for equational first-order specification and programming [FGJM85]. Both notions are important for practical reasons. With parametric polymorphism one avoids the need for redefining lists and other parametric data types for every type they are used with. Subsorts not only provide for more natural type specifications, but also yield more computational power: variables can be constrained to sorts rather than to single values and typed unification computes directly with sort constraints, thus reducing the need for expensive backtracking.

Figure 1.1 shows some examples of sort and relation definitions in our language. Sort constants and sort functions are defined by equations, and relations are defined by a declaration and a collection of definite clauses. The sort bool has two elements, which are given by the value constants true and false. The sort int is defined as the union of its subsorts inat and nat. The elements of the sort posint are obtained by applying the value function

$$\mathsf{s}\colon \mathsf{nat} \to \mathsf{posint}$$

$$\mathsf{bool} \;:=\; \mathsf{true}{:}\,[] \;\sqcup\; \mathsf{false}{:}\,[]$$

$$\mathsf{int} \;:=\; \mathsf{inat} \;\sqcup\; \mathsf{nat}$$
$$\mathsf{inat} \;:=\; \mathsf{zero} \;\sqcup\; \mathsf{negint}$$
$$\mathsf{negint} \;:=\; \mathsf{p}{:}\,\mathsf{inat}$$
$$\mathsf{zero} \;:=\; \mathsf{o}{:}\,[]$$
$$\mathsf{nat} \;:=\; \mathsf{zero} \;\sqcup\; \mathsf{posint}$$
$$\mathsf{posint} \;:=\; \mathsf{s}{:}\,\mathsf{nat}$$

$$\mathsf{le}{:}\,\mathsf{int} \times \mathsf{int} \times \mathsf{bool}$$
$$\mathsf{le}(\mathsf{p}(\mathsf{I}), \mathsf{p}(\mathsf{J}), \mathsf{B}) \;\leftarrow\; \mathsf{le}(\mathsf{I}, \mathsf{J}, \mathsf{B})$$
$$\mathsf{le}(\mathsf{s}(\mathsf{I}), \mathsf{s}(\mathsf{J}), \mathsf{B}) \;\leftarrow\; \mathsf{le}(\mathsf{I}, \mathsf{J}, \mathsf{B})$$
$$\mathsf{le}(\mathsf{o}, \mathsf{I}, \mathsf{true}) \;\leftarrow\; \mathsf{I} : \mathsf{nat}$$
$$\mathsf{le}(\mathsf{o}, \mathsf{I}, \mathsf{false}) \;\leftarrow\; \mathsf{I} : \mathsf{negint}$$
$$\mathsf{le}(\mathsf{I}, \mathsf{o}, \mathsf{true}) \;\leftarrow\; \mathsf{I} : \mathsf{inat}$$
$$\mathsf{le}(\mathsf{I}, \mathsf{o}, \mathsf{false}) \;\leftarrow\; \mathsf{I} : \mathsf{posint}$$

$$\mathsf{list}(\mathsf{T}) \;:=\; \mathsf{elist} \;\sqcup\; \mathsf{nelist}(\mathsf{T})$$
$$\mathsf{elist} \;:=\; \mathsf{nil}{:}\,[]$$
$$\mathsf{nelist}(\mathsf{T}) \;:=\; \mathsf{cons}{:}\,\mathsf{T} \times \mathsf{list}(\mathsf{T})$$
$$\mathsf{pair}(\mathsf{S}, \mathsf{T}) \;:=\; \mathsf{cp}{:}\,\mathsf{S} \times \mathsf{T}$$
$$\mathsf{difflist}(\mathsf{T}) \;:=\; \mathsf{pair}(\mathsf{list}(\mathsf{T}), \mathsf{list}(\mathsf{T}))$$

$$\mathsf{error_or_list}(\mathsf{E}, \mathsf{T}) \;:=\; \mathsf{errormsg}(\mathsf{E}) \;\sqcup\; \mathsf{list}(\mathsf{T})$$
$$\mathsf{errormsg}(\mathsf{E}) \;:=\; \mathsf{error}{:}\,\mathsf{nat} \times \mathsf{list}(\mathsf{pair}(\mathsf{nat}, \mathsf{E}))$$

$$\mathsf{append}{:}\,\mathsf{list}(\mathsf{T}) \times \mathsf{list}(\mathsf{T}) \times \mathsf{list}(\mathsf{T})$$
$$\mathsf{append}(\mathsf{nil}, \mathsf{L}, \mathsf{L})$$
$$\mathsf{append}(\mathsf{cons}(\mathsf{H}, \mathsf{R}), \mathsf{L}, \mathsf{cons}(\mathsf{H}, \mathsf{RL})) \;\leftarrow\; \mathsf{append}(\mathsf{R}, \mathsf{L}, \mathsf{RL})$$

Figure 1.1: A POS-program.

to elements of nat. Since nat is defined as the union of its subsorts zero and posint, the elements of nat are o, s(o), s(s(o)) and so forth.

The value functions of POS-types are free constructors: they are injective, and distinct value functions never yield the same element. Defining sorts with free constructors is common in functional programming languages and can be traced back to Landin [Lan64], Burstall [Bur69] and Hoare [Hoa75]. Types defined with sorts and free constructors are a special case of algebraic types [GTW78, NR85, EM85], which are much more expressive since they provide for equations between constructors. The POS-types investigated in this thesis are restricted to free constructors and hence their specifications cannot employ equations between value terms.

The relation le is a less or equal test on the elements of int. Some of the variables occurring in the clauses defining le are explicitly constrained to sorts. For the variables that aren't explicitly constrained, most general sorts are automatically derived by a type inference algorithm. This yields

$$\mathsf{I{:}\,inat}, \quad \mathsf{J{:}\,inat}, \quad \mathsf{B{:}\,bool}$$

for the first clause of le and

$$\mathsf{I{:}\,nat}, \quad \mathsf{J{:}\,nat}, \quad \mathsf{B{:}\,bool}$$

for the second clause of le.

Sort functions are defined analogously to sort constants, except that the defining equation is parameterized with respect to sort variables (one for every argument of the sort function to be defined), which range over the set of all sorts specified by the program. Programs with sort functions specify infinitely many sorts, for instance, list(nat), list(list(nat)) and so forth.

A sort is simply the set of its elements. Hence sorts are partially ordered by set inclusion. Sort functions are monotone with respect to the inclusion order. Consequently, list(nat) is a subset of list(int) since nat is a subset of int (recall that int is defined as the union of nat and inat).

From the sort equations

$$\mathsf{list(T)} \;:=\; \mathsf{elist} \;\sqcup\; \mathsf{nelist(T)}$$
$$\mathsf{error_or_list(E,T)} \;:=\; \mathsf{errormsg(E)} \;\sqcup\; \mathsf{list(T)}$$

you can see that the types investigated in this thesis can be specified with sort equations admitting the union of polymorphic sort terms.

The list concatenation relation append is defined with exactly the same clauses one would use in untyped logic programming. The declaration for append is used for type checking and type inference. The type inference algorithm completes the second clause of append to

$$\mathsf{append(cons(H,R),L,cons(H,RL))} \;\leftarrow$$
$$\mathsf{H:T} \;\&\; \mathsf{R:list(T)} \;\&\; \mathsf{L:list(T)} \;\&\; \mathsf{RL:list(T)} \;\&\; \mathsf{append(R,L,RL)}.$$

A few words on the difference between sort and types (as used in this thesis): a sort is just a set of values, while a type is an algebra specified by a collection of sort equations and consists of sorts, sort functions and value functions. A POS-type is a type that can be specified within the framework developed in this thesis. This use of the terms sort and type is common in the theory of algebraic specifications [GTW78, NR85, EM85]. In the context

of programming languages, however, one usually just talks of types and would thus refer to, for instance, bool as a type. Since the terms type discipline, well-typedness, type checking and type inference are so familiar from programming languages, I will use them in this thesis although it would be more appropriate to speak of sort discipline, well-sortedness, sort checking and sort inference.

Two well-known advantages of typed programming languages, which apply to typed logic programming in particular, are:

1. The data structures used by a program can be defined explicitly. This leads to clearer, much easier to understand programs. The explicit definition of data structures is particularly beneficial if they are complex, as it is typically the case in Artificial Intelligence.

2. Type checking detects many programming errors at compile time, a feature whose importance is proportional to the size of the program under development.

Well-known disadvantages of typed programming languages, whose weight has been significantly reduced by the invention of type inference and parametric polymorphism [Mil78], are that the programmer is burdened with specifying redundant type information and that typed programs tend to be unnecessarily complicated since the programmer is sometimes forced to program around the type discipline. For pure logic programming, however, the introduction of a type discipline actually amounts to a generalization rather than a restriction. By introducing only one sort and declaring every function as a constructor of this sort, every untyped logic program becomes a well-typed program. Of course, for logic programming to be practical, one needs extra-logical features. The programming language TEL [Smo88b], which embodies the logical language presented in this thesis, demonstrates that the necessary nonlogical features can be integrated such that they are type-safe and still practical.

As mentioned before, typed unification adds expressive power by exploiting the inclusion order on sorts. In untyped logic programming, one could express a sort as a unary predicate holding for the elements of the sort. To express, say, that the variable X ranges over the sort negint, one could write the atom negint(X). Now suppose that during the course of a computation the additional constraint posint(X) is imposed. While typed unification would immediately recognize that there is no value for X left, untyped logic programming cannot recognize this conflict. All it can do is bind X successively to elements of either posint or negint and find out each time anew that the other constraint is violated.

1.2 Related Work

The existing work on typed logic programming can be classified into a syntactic and a semantic approach.

The syntactic approach sees types as a syntactic discipline that must not change the semantics of a program. Since the semantics of the program is not changed, typed programs can be executed in exactly the same way as untyped programs, where, however, the structure imposed by the type discipline can be exploited for optimizations. Mycroft and O'Keefe

[MO84] show how Milner's [Mil78] polymorphic type discipline can be adapted to Pure Prolog. Their system relies on type declarations given by the programmer but does not require type declarations for the variables occurring in a clause. Dietrich and Hagl's [DH88] extends Mycroft and O'Keefe's work to include subsorts. Since his approach is purely syntactic, he has to restrict the class of well-typed programs severely by imposing a mode discipline. Moreover, it is an open problem whether his type checking discipline is decidable. Another direction of the syntactic approach investigates type inference for logic programs in the complete absence of programmer-provided type declarations [Mis84, Zob87].

The semantic approach, to which this thesis contributes, bases logic programming with types on logics that account for sorts. Consequently, the computational mechanisms may change, which typically shows up in the unification procedure. One direction, which appeared with Eqlog [GM86], takes order-sorted logic as base and employs order-sorted unification [Gog78, GM87b, HV87, MGS89, SNGM89, SS85, SS89, Wal83, Wal85, Wal87, Wal88]. This thesis generalizes this line of research by incorporating parametric polymorphism.

The distinctive difference between order-sorted types [Gog78, GM87a, SNGM89] and POS-types is that POS-types can come with sort functions such as list or pair while order-sorted types cannot. POS-types without sort functions are order-sorted types, but most order-sorted types cannot be obtained as POS-types since the specifications of POS-types admit neither multiple declarations of value functions nor equations between value terms.

Recent research of Hanus [Han88, Han89] develops a Horn logic with equality augmented with a polymorphic sort structure (not accommodating subsorts). Hanus gives an operational semantics for relational programs computing on the polymorphic types specifiable in his framework, where his programs need only satisfy a weak well-typedness condition admitting ad hoc polymorphism. The difference between parametric and ad hoc polymorphism (discussed already in [Str67]) is that a parametrically polymorphic operator must behave in exactly the same way for all parameter sorts while an ad hoc polymorphic operator is free to do different things for different sort parameters. Now, if there are no subsorts (as in [Han88, Han89] and [MO84]) and the clauses of the program are well-typed under a type discipline enforcing parametric polymorphism (as in this thesis and [MO84]), then the standard unsorted operational semantics turns out to be sound and complete. Hence, the sort-related operational methods of Hanus are orthogonal to the operational methods developed in this thesis: Hanus needs to account for ad hoc polymorphism while I stick to parametric polymorphism but need to account for subsorts.

There is a good reason why I insist on a type discipline enforcing parametric polymorphism: the operational methods developed in this thesis work only under this structural assumption, which provides for crucial optimizations of the constraint solver, and I even don't know if a complete computable constraint solver exists for a type discipline admitting ad hoc polymorphism in the presence of subsorts.

Another direction of the semantic approach was initiated by Aït-Kaci and Nasr's [AKN86] language LOGIN, which replaces ordinary first-order terms with record structures and employs a typed unification called ψ-unification. Mukai's [Muk87] language CIL is similar to LOGIN but has no subsorts. Smolka and Aït-Kaci [SA89] show how LOGIN can be captured in order-sorted logic and device a framework that combines order-sorted constructor types

with LOGIN's feature types. Feature Logic [Smo88a, Smo89] is a decidable logic that generalizes Aït-Kaci's formalism by adding negation and quantification. Feature Logic makes explicit that Aït-Kaci's ψ-terms, the feature descriptions developed by computational linguists [KB82, RK86, Joh88], and the knowledge representation language KL-ONE [BS85, LB87, Neb89, SSS91, NS90] are all closely related members of the same family of logics. These logics offer attributive concept descriptions that are interpreted as sets and are built from sorts and binary relations (called attributes, roles or features) using set operations such as intersection, union and complement.

1.3 Overview

Chapter 2 presents a general framework for relational logic programming extending and generalizing Jaffar and Lassez's [JL86, JL87] model of constraint logic programming (CLP, for short). While CLP relies on many-sorted Predicate Logic as base language, our framework is based on a general notion of constraint language that can be instantiated, for instance, to Predicate Logic, Order-Sorted Logic, attributive concept description logics or the logic underlying POS-types to be developed in Chapter 3. Like CLP our framework gives a generic operational semantics. Our framework comes with a type discipline and develops the notions of well-typedness, type checking and type inference in general without commitment to a particular constraint language. An interesting question for future research is whether polymorphic sort constructions such as lists and pairs can be provided already within such a general framework.

The next four chapters develop the theory of POS-types. Chapter 3 presents the logic underlying POS-types. Chapter 4 studies a class of rewriting systems used for the specification of the inclusion orders of POS-types. The emphasis is on operational methods needed for constraint solving and type inference. Chapter 5 defines the class of specifications from whose initial models POS-types are obtained by a canonical quotient construction turning sorts into the sets of their elements. Chapter 6, which is the heart of this thesis, develops the constraint solver to be employed in the interpreter for POS-programs. The constraint solver comes with several powerful optimizations avoiding redundant sort computations by exploiting the structure imposed by a type discipline enforcing parametric polymorphism. The optimizations rely on a compilation step replacing the sort terms in the program with approximations just retaining the absolutely necessary information.

In Chapter 7 the general framework of Chapter 2 and the theory of POS-types developed in Chapters 3–6 are put together. We define POS-programs and obtain a sound and complete interpreter by integrating the constraint solver of Chapter 6 with the generic operational semantics of Chapter 2. We show that type checking for POS-programs is decidable and present an incomplete type inference algorithm.

Chapter 2

Logic Programming over Constraint Languages

2.1 Constraint Languages
2.2 Canonical Extensions
2.3 Definite Clauses
2.4 Operational Semantics
2.5 A Type Discipline
2.6 Type Inference

In the last few years a new model of logic programming has emerged that views a logic programming language as consisting of a constraint language on top of which relations can be defined by means of definite clauses. Different logic programming languages can be obtained by employing different constraint languages. Conventional logic programming is obtained by employing equations that are interpreted in the algebra of first-order terms. Prolog II [CKC83, Col84] employs as constraint language equations and disequations that are interpreted in the algebra of rational trees. The constraint language of Prolog III [Col88] is interpreted in an algebra providing rational trees and rational numbers and allows for linear equations and inequations for numbers, boolean expressions for truth values, and equations and disequations for general terms. Other recent examples of constraint logic programming languages are CLP($\mathcal{R}$) [JM87], CIL [Muk87] and CHIP [DHS+88].

Jaffar and Lassez [JL86, JL87] were the first to identify the new model, coined the name Constraint Logic Programming, and developed a general framework that is parameterized

[0]This Chapter is an adaptation of [HS88].

with respect to the constraint language being employed and yields soundness and completeness results for a generic operational semantics relying on a constraint solver for the employed constraint language. A constraint solver is an algorithm deciding the satisfiability of constraint systems. In conventional logic programming, the constraint solver solves equations in the Herbrand universe, which is accomplished by term unification.

The original motivation for the research reported in this chapter was the development of a semantic foundation for the knowledge representation language LOGIN [AKN86], where relations are defined with definite clauses over a constraint language consisting of so-called ψ-terms [AK86].

The first step of this enterprise was to come up with a logical reformulation of Aït-Kaci's ψ-term calculus and led to the development of Feature Logic [Smo88a, Smo89], a decidable logic that generalizes Aït-Kaci's formalism by adding negation and quantification. Feature Logic makes explicit that Aït-Kaci's ψ-terms, the feature descriptions developed by computational linguists [KB82, RK86, Joh88], and the knowledge representation language KL-ONE [BS85, LB87, Neb89, SSS91, NS90] are all closely related members of the same family of logics. These logics offer attributive concept descriptions that are interpreted as sets and are built from sorts and binary relations (called attributes, roles or features) using set operations such as intersection, union and complement. Given an attributive concept description C, a constraint $x{:}C$ constrains the values of the variable x to elements of C.

Ideally, the second step of giving a semantic foundation to LOGIN should have consisted in simply applying Jaffar and Lassez's [JL86, JL87] constraint logic programming scheme (CLP, for short) to Feature Logic. However, this failed for three reasons:

1. CLP requires that the constraint language is interpreted in a single fixed domain. This is in accordance with the data structure paradigm underlying current programming languages, which views programs as computing with data structures that are, in most applications, merely representations of the objects one is actually interested in. For knowledge representation, however, data structures as representations of real objects are not adequate. Instead, one talks directly about the objects of interest, as this is accomplished, for instance, by the Tarski semantics of Predicate Logic. Since, in general, we have only partial information about the world we want to reason about, we need to take into account all worlds that are consistent with our partial knowledge. Thus we have to generalize CLP such that the constraint language can come with more than one interpretation and a constraint is considered satisfiable if there is at least one interpretation in which it has a solution.

2. CLP requires that the interpretations of constraint languages be "solution compact", which implies that every element of an interpretation must be obtainable as the unique solution of a possibly infinite set of constraints. While solution compactness is sensible for "data structure" interpretations, it is not acceptable for "real world" interpretations. CLP needs solution compactness since it provides soundness and completeness results for negation as failure. However, since the constraint language can provide for logical negation (for instance, disequations in Prolog II or set complements in Feature Logic) I feel that for many applications there is no further need for the problematic negation as failure.

3. CLP assumes that the constraint language is expressed in Predicate Logic: constraints must be formulas of Predicate Logic and interpretations must be interpretations of Predicate Logic. However, neither Feature Logic, KL-ONE, nor the logic we are going to use for the specification of POS-types satisfy these assumptions. Although these formalisms can be reduced to Predicate Logic in principle, providing customized model theories and notations for them is crucial in keeping them technically simple and in supporting the adequate intuitions. So what CLP is lacking is a sufficiently abstract formalization of the notion of a constraint language.

This chapter presents a framework that generalizes CLP so that the shortcomings discussed above are completely avoided. Our framework also extends CLP with a general type discipline providing for an abstract treatment of notions like well-typedness, type checking and type inference.

The framework presented in this chapter, which is designed as a foundation of Logic Programming in general, will be just the right starting point for our theory of logic programming over POS-types. Presenting our theory of logic programming over POS-types as an instance of the general framework provides for a clear distinction of notions and results in those that apply in general and in those that are specific to POS-types.

Section 2.1 gives a definition of constraint languages that is general enough to cover all mentioned formalisms. In our analysis, a constraint is a piece of syntax constraining the values the variables occurring in it can take. There is no need to know anything about the internal structure of a constraint. Since we are not concerned with negation as failure, we don't need to impose any requirements on the interpretations of constraint languages. A prominent example of a constraint language is Predicate Logic, where the formulas serve as constraints. Furthermore, the notion of a constraint language will be exploited for the development of POS-Logic in the next chapter and significantly reduce the notational overhead.

In Section 2.2 we show that every constraint language can be extended conservatively to a constraint language providing for relational atoms, the propositional connectives, and quantification. By taking equations with their Tarski interpretations as constraint language, this construction yields Predicate Logic.

In Section 2.3 we show that, for every set S of definite clauses in the extension of an arbitrary constraint language $\mathcal{L}$, every interpretation of $\mathcal{L}$ can be extended to a minimal model of S. This generalizes the key result of conventional logic programming to our framework, which is not restricted to Horn theories.

In Section 2.4 we present an operational semantics for our general definite clause specifications that generalizes the SLD-resolution method [Llo84] employed in conventional logic programming and prove its soundness and completeness.

In Section 2.5 we present a semantic type discipline for our generalized definite clause specifications. The discipline exploits the idea that declarations of relation symbols in a sorted language can be expressed as implications; for instance, to declare that the relation *plus* takes integers as arguments, we can write the implication

$$plus(x, y, z) \;\rightarrow\; x\!:\!int \,\&\, y\!:\!int \,\&\, z\!:\!int.$$

If Feature Logic is used as underlying constraint language, we can constrain the arguments of a relation with complex feature terms employing intersections, complements and feature constraints. The idea even applies to conventional logic programming, where we can write declarations like

$$p(x, y) \;\rightarrow\; \exists\, z.\; y = f(x, z).$$

We establish a weak notion of well-typedness by saying that a definite clause specification S is implicitly well-typed with respect to a set D of declarations if every minimal model of S is a model of D. Next we establish a strong notion of well-typedness by defining explicitly well-typed clauses and show that explicitly well-typed specifications are implicitly well-typed. Explicit well-typedness is decidable provided the underlying constraint language is decidable. Furthermore, we show that our operational semantics is type safe, that is, the reduction of an explicitly well-typed goal with an explicitly well-typed clause yields again an explicitly well-typed goal.

Section 2.6 gives a type inference rule that can be used to compute a most general explicitly well-typed weakening of a specification. We show that, if the explicitly well-typed specification S' is obtained from S by type inference, S' and S have the same minimal models, provided S is implicitly well-typed.

2.1 Constraint Languages

The basic idea is that a constraint is some piece of syntax constraining the values of the variables occurring in it. Our notion of constraint language does not make any assumptions about the syntax of constraints.

Technically, it is convenient to have more than one class of variables and values available. For instance, POS-Logic will have two disjoint classes of variables, one ranging over sorts and one ranging over the elements of sorts.

Let I be a set. An **I-indexed set** is a family $M = (M_i)_{i \in I}$ of pairwise disjoint nonempty sets. An I-indexed set M is **decidable** if there exists a computable function $\iota\colon \bigcup_{i \in I} M_i \rightarrow I$ such that $\iota(a) = i$ if and only if $a \in M_i$. An **I-indexed mapping** from an I-indexed set M to an I-indexed set M' is a mapping

$$f\colon \bigcup_{i \in I} M_i \rightarrow \bigcup_{i \in I} M'_i$$

such that $f(a) \in M'_i$ if $a \in M_i$.

Let M be an I-indexed set. To obtain a smooth notation, we will abuse M to also denote the union $\bigcup_{i \in I} M_i$. By an **element of** M we always mean an element of $\bigcup_{i \in I} M_i$ and $a \in M$ always stands for $a \in \bigcup_{i \in I} M_i$.

Warning. Our definition of I-indexed sets is different from the notion of "many-sortedness" used in the theory of algebraic specifications.

A **constraint language** is a tuple $(I, \mathrm{VAR}, \mathrm{CON}, \mathcal{V}, \mathrm{INT})$ such that

1. I is a decidable set

2. VAR is a decidable I-indexed set such that VAR_i is infinite for every $i \in I$; the elements of VAR are called **variables**

3. CON is a decidable set whose elements are called **constraints**

4. $\mathcal{V}$ is a computable function that assigns to every constraint ϕ a finite set $\mathcal{V}\phi$ of variables, called the **variables constrained by** ϕ

5. INT is a nonempty set of so-called **interpretations**, where every interpretation $\mathcal{I} \in \mathrm{INT}$ consists of an I-indexed set $\mathbf{D}^{\mathcal{I}}$, called the **domain of** $\mathcal{I}$, and a **solution mapping** $\mathcal{I}[\![\cdot]\!]$ such that:

   (a) an $\mathcal{I}$**-assignment** is an I-indexed mapping $\mathrm{VAR} \to \mathbf{D}^{\mathcal{I}}$, and $\mathrm{ASS}^{\mathcal{I}}$ is the set of all $\mathcal{I}$-assignments

   (b) $\mathcal{I}[\![\cdot]\!]$ is a function mapping every constraint ϕ to a set $\mathcal{I}[\![\phi]\!] \subseteq \mathrm{ASS}^{\mathcal{I}}$, where the $\mathcal{I}$-assignments in $\mathcal{I}[\![\phi]\!]$ are called the **solutions of** ϕ **in** $\mathcal{I}$

   (c) a constraint ϕ constrains only the variables in $\mathcal{V}\phi$, that is, if $\alpha \in \mathcal{I}[\![\phi]\!]$ and β is an $\mathcal{I}$-assignment that agrees with α on $\mathcal{V}\phi$, then $\beta \in \mathcal{I}[\![\phi]\!]$.

Note that our definition of I-indexed sets makes sure that for every interpretation $\mathcal{I}$ there exists at least one $\mathcal{I}$-assignment. Thus we won't be plagued by the infamous "empty sort problem" known in the theory of algebraic specifications.

Predicate logic is a prominent example of a constraint language: there is only one class of variables, the well-formed formulas are the constraints, $\mathcal{V}\phi$ can be taken as the set of all variables in ϕ that are free in ϕ, and for every Tarski interpretation $\mathcal{I}$ the solutions $\mathcal{I}[\![\phi]\!]$ are the $\mathcal{I}$-assignments satisfying ϕ. Viewing predicate logic as a constraint language abstracts away from the syntactic details of formulas.

The following definitions are all made with respect to some given constraint language $\mathcal{L} = (I, \mathrm{VAR}, \mathrm{CON}, \mathcal{V}, \mathrm{INT})$. Most of the definitions generalize terminology that is well-known for predicate logic.

A constraint is **satisfiable** if there exists at least one interpretation in which it has a solution.

A constraint ϕ is **valid** in an interpretation $\mathcal{I}$ if $\mathcal{I}[\![\phi]\!] = \mathrm{ASS}^{\mathcal{I}}$, that is, every $\mathcal{I}$-assignment is a solution of ϕ in $\mathcal{I}$. Conversely, we say that an interpretation $\mathcal{I}$ **satisfies** a constraint ϕ if ϕ is valid in $\mathcal{I}$. An interpretation is a **model** of a set Φ of constraints if it satisfies every constraint in Φ. Conversely, we say that an interpretation $\mathcal{I}$ **satisfies** a set Φ of constraints if it satisfies every constraint in Φ. A constraint ϕ is **valid** in a set Φ of constraints if every model of Φ satisfies ϕ.

A **renaming** is a bijective I-indexed mapping $\mathrm{VAR} \to \mathrm{VAR}$ that is the identity everywhere except for finitely many exceptions. If ρ is a renaming, we call a constraint ϕ' a ρ**-variant** of a constraint ϕ if

$$\mathcal{V}\phi' = \rho(\mathcal{V}\phi) \quad \text{and} \quad \mathcal{I}[\![\phi]\!] = \mathcal{I}[\![\phi']\!]\rho := \{\alpha\rho \mid \alpha \in \mathcal{I}[\![\phi']\!]\}$$

for every interpretation $\mathcal{I}$. A constraint ϕ' is called a **variant** of a constraint ϕ if there exists a renaming ρ such that ϕ' is a ρ-variant of ϕ.

Proposition 2.1.1 *A constraint is satisfiable if and only if each of its variants is satisfiable. Furthermore, a constraint is valid in an interpretation $\mathcal{I}$ if and only if each of its variants is valid in $\mathcal{I}$.*

A constraint language is **closed under renaming** if every constraint ϕ has a ρ-variant for every renaming ρ. A constraint language is **closed under intersection** if for every two constraints ϕ and ϕ' there exists a constraint ψ such that $\mathcal{I}[\![\phi]\!] \cap \mathcal{I}[\![\phi']\!] = \mathcal{I}[\![\psi]\!]$ for every interpretation $\mathcal{I}$. A constraint language is **decidable** if the satisfiability of its constraints is decidable.

Let Φ be a set of constraints and $\mathcal{I}$ be an interpretation. The **solutions of Φ in $\mathcal{I}$** are defined as

$$\mathcal{I}[\![\Phi]\!] := \bigcup_{\phi \in \Phi} \mathcal{I}[\![\phi]\!],$$

where $\mathcal{I}[\![\Phi]\!] := \emptyset$ if Φ is empty. Note that this definition interprets a set of constraints disjunctively, while the above definition of a model interprets a set of constraints conjunctively. To ease our notation, we often abbreviate a singleton $\{\phi\}$ to ϕ.

Given a set V of variables, the **V-solutions** of a set Φ of constraints in an interpretation $\mathcal{I}$ are defined as

$$\mathcal{I}[\![\Phi]\!]^V := \{\alpha|_V \mid \alpha \in \mathcal{I}[\![\Phi]\!]\} = \bigcup_{\phi \in \Phi} \{\alpha|_V \mid \alpha \in \mathcal{I}[\![\phi]\!]\}$$

where $\alpha|_V$ is the restriction of α to V. We say that a set of constraints Φ is **V-subsumed** by a set of constraints Φ' and write $\Phi \preceq_V \Phi'$ if $\mathcal{I}[\![\Phi]\!]^V \subseteq \mathcal{I}[\![\Phi']\!]^V$ for every interpretation $\mathcal{I}$. Obviously, V-subsumption defines a preorder on sets of constraints. The corresponding equivalence relation

$$\Phi \sim_V \Phi' \quad :\Longleftrightarrow \quad \Phi \preceq_V \Phi' \ \wedge \ \Phi' \preceq_V \Phi$$

is called **V-equivalence**.

Proposition 2.1.2 *Renaming is homomorphic with respect to V-subsumption, that is, if ρ and ρ' are renamings that agree on V, ϕ' is a ρ-variant of a constraint ϕ, ψ' is a ρ'-variant of a constraint ψ, and $\phi \preceq_V \psi$, then $\phi' \preceq_{\rho(V)} \psi'$.*

A constraint language is called **compact** if for every set V of variables, every constraint ϕ, and every set of constraints Φ, ϕ is V-subsumed by Φ if and only if ϕ is V-subsumed by some finite subset of Φ.

Predicate Logic is a compact and undecidable constraint language that is closed under renaming and intersection.

Let $\mathcal{A}$ and $\mathcal{B}$ be interpretations of $\mathcal{L}$. An **$\mathcal{L}$-morphism** $\mathcal{A} \to \mathcal{B}$ is an I-indexed mapping $\gamma\colon \mathbf{D}^{\mathcal{A}} \to \mathbf{D}^{\mathcal{B}}$ such that

1. $\gamma \mathrm{ASS}^{\mathcal{A}} = \mathrm{ASS}^{\mathcal{B}}$
2. $\gamma \mathcal{A}[\![\phi]\!] \subseteq \mathcal{B}[\![\phi]\!]$

3. $\{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![\phi]\!]\} \subseteq \mathcal{A}[\![\phi]\!]$.

Proposition 2.1.3 *Let γ be an $\mathcal{L}$-morphism $\mathcal{A} \to \mathcal{B}$. Then γ is surjective and*

$$\begin{aligned} \gamma\mathcal{A}[\![\phi]\!] &= \mathcal{B}[\![\phi]\!] \\ \{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![\phi]\!]\} &= \mathcal{A}[\![\phi]\!]. \end{aligned}$$

Proposition 2.1.4 *If γ is an $\mathcal{L}$-morphism $\mathcal{A} \to \mathcal{B}$, then a constraint ϕ is valid [satisfiable] in $\mathcal{A}$ if and only if it is valid [satisfiable] in $\mathcal{B}$.*

2.2 Canonical Extensions

Let $\mathcal{L}$ be a constraint language. Then we can extend the constraints of $\mathcal{L}$ according to the abstract syntax rule:

F, G	$\longrightarrow$	ϕ	*basic constraint*
	$\mid$	$\emptyset$	*empty conjunction*
	$\mid$	$F \,\&\, G$	*conjunction*
	$\mid$	$\neg F$	*negation*
	$\mid$	$\exists x.F$	*existential quantification.*

The variable mapping $\mathcal{V}$ of $\mathcal{L}$ can extended as follows:

$$\mathcal{V}\emptyset := \emptyset, \quad \mathcal{V}(F \,\&\, G) := \mathcal{V}F \cup \mathcal{V}G, \quad \mathcal{V}(\neg F) := \mathcal{V}F, \quad \mathcal{V}(\exists x.F) := \mathcal{V}F - \{x\}.$$

Furthermore, if $\mathcal{I}$ is an interpretation of $\mathcal{L}$, we extend the solution mapping $\mathcal{I}[\![\cdot]\!]$ of $\mathcal{I}$ as follows to the new constraints:

$$\begin{aligned} \mathcal{I}[\![\emptyset]\!] &:= \mathrm{ASS}^{\mathcal{I}} \\ \mathcal{I}[\![F \,\&\, G]\!] &:= \mathcal{I}[\![F]\!] \cap \mathcal{I}[\![G]\!] \\ \mathcal{I}[\![\neg F]\!] &:= \mathrm{ASS}^{\mathcal{I}} - \mathcal{I}[\![F]\!] \\ \mathcal{I}[\![\exists x.F]\!] &:= \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \exists\ \beta \in \mathcal{I}[\![F]\!].\ \ \beta|_{\mathcal{V}F-\{x\}} = \alpha|_{\mathcal{V}F-\{x\}}\}. \end{aligned}$$

Now, leaving the index set I and the variables VAR of $\mathcal{L}$ unchanged, we have arrived at a new constraint language $\mathcal{L}^*$ extending $\mathcal{L}$ conservatively. If $\mathcal{A}$ is an interpretation of $\mathcal{L}$, we will abuse $\mathcal{A}$ to also denote the extended interpretation, which is different only in that its solution mapping is extended to the new constraints. As usual, we will use the abbreviations

$F \mid G$	$\equiv \neg(\neg F \,\&\, \neg G)$	*disjunction*
$F \to G$	$\equiv (\neg F) \mid G$	*implication*
$\forall x.F$	$\equiv \neg(\exists x.\neg F)$	*universal quantification.*

Since $\mathcal{L}^*$ is a constraint language, all definitions we have made for constraint languages in general apply to $\mathcal{L}^*$ in particular. This shows that the notion of a constraint language can be applied iteratively.

Proposition 2.2.1 *Let $\mathcal{L}$ be a constraint language, ρ be an $\mathcal{L}$-renaming, and F be an $\mathcal{L}^*$-constraint. Then F' is a ρ-variant of F if F' can be obtained from F by replacing every variable x with $\rho(x)$ and every $\mathcal{L}$-constraint ϕ with a ρ-variant of ϕ. Thus $\mathcal{L}^*$ is closed under renaming if $\mathcal{L}$ is closed under renaming. Furthermore, $\mathcal{L}^*$ is always closed under intersection.*

Theorem 2.2.2 *Let $\mathcal{L}$ be a constraint language, $\mathcal{A}$ and $\mathcal{B}$ be interpretations of $\mathcal{L}$, and γ be an $\mathcal{L}$-morphism $\mathcal{A} \to \mathcal{B}$. Then γ is an $\mathcal{L}^*$-morphism $\mathcal{A} \to \mathcal{B}$.*

Proof. We show simultaneously by induction on F that $\gamma\mathcal{A}[\![F]\!] \subseteq \mathcal{B}[\![F]\!]$ and $\{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![F]\!]\} \subseteq \mathcal{A}[\![F]\!]$.

If F is an $\mathcal{L}$-constraint, then the claims hold since γ is an $\mathcal{L}$-morphism $\mathcal{A} \to \mathcal{B}$.

If $F = \emptyset$, then $\gamma\mathcal{A}[\![\emptyset]\!] = \gamma\mathrm{ASS}^{\mathcal{A}} = \mathrm{ASS}^{\mathcal{B}} = \mathcal{B}[\![\emptyset]\!]$ and $\{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![\emptyset]\!]\} \subseteq \mathrm{ASS}^{\mathcal{A}} = \mathcal{A}[\![\emptyset]\!]$.

Let $F = G \,\&\, G'$. Then we have

$$\begin{aligned}\gamma\mathcal{A}[\![G \,\&\, G']\!] = \gamma(\mathcal{A}[\![G]\!] \cap \mathcal{A}[\![G']\!]) \subseteq & \gamma\mathcal{A}[\![G]\!] \cap \gamma\mathcal{A}[\![G']\!] \\ & \subseteq \mathcal{B}[\![G]\!] \cap \mathcal{B}[\![G']\!] = \mathcal{B}[\![G \,\&\, G']\!]\end{aligned}$$

using the induction hypothesis for the first claim twice. Furthermore, we have

$$\begin{aligned}&\{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![G \,\&\, G']\!]\} \\ &\quad = \{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![G]\!]\} \cap \{\alpha \in \mathrm{ASS}^{\mathcal{A}} \mid \gamma\alpha \in \mathcal{B}[\![G']\!]\} \\ &\quad\quad \subseteq \mathcal{A}[\![G]\!] \cap \mathcal{A}[\![G']\!] = \mathcal{A}[\![G \,\&\, G']\!]\end{aligned}$$

using the induction hypothesis for the second claim twice.

Let $F = \neg G$ and $\alpha \in \mathcal{A}[\![\neg G]\!]$. Then $\alpha \notin \mathcal{A}[\![G]\!]$ and hence, by the induction hypothesis for the second claim, $\gamma\alpha \notin \mathcal{B}[\![G]\!]$. Thus $\gamma\alpha \in \mathcal{B}[\![\neg G]\!]$.

Let $F = \neg G$, $\alpha \in \mathrm{ASS}^{\mathcal{A}}$ and $\gamma\alpha \in \mathcal{B}[\![\neg G]\!]$. Then $\gamma\alpha \notin \mathcal{B}[\![G]\!]$ and hence, by the induction hypothesis for the first claim, $\alpha \notin \mathcal{A}[\![G]\!]$. Thus $\alpha \in \mathcal{A}[\![\neg G]\!]$.

Let $F = \exists x.G$ and $\alpha \in \mathcal{A}[\![\exists x.G]\!]$. Then there exists an assignment $\beta \in \mathcal{A}[\![G]\!]$ that agrees with α on $\mathcal{V}G - \{x\}$. Hence $\gamma\beta \in \mathcal{B}[\![G]\!]$ by the induction hypothesis for the first claim. Thus $\gamma\alpha \in \mathcal{B}[\![\exists x.G]\!]$ since $\gamma\alpha$ and $\gamma\beta$ agree on $\mathcal{V}G - \{x\}$.

Let $F = \exists x.G$, $\alpha \in \mathrm{ASS}^{\mathcal{A}}$, and $\gamma\alpha \in \mathcal{B}[\![\exists x.G]\!]$. Then there exists an assignment $\beta \in \mathcal{B}[\![G]\!]$ that agrees with $\gamma\alpha$ on $\mathcal{V}G - \{x\}$. Since $\gamma\mathrm{ASS}^{\mathcal{A}} = \mathrm{ASS}^{\mathcal{B}}$, there exists an assignment $\delta \in \mathrm{ASS}^{\mathcal{A}}$ such that $\gamma\delta = \beta \in \mathcal{B}[\![G]\!]$ and δ agrees with α on $\mathcal{V}G - \{x\}$. Hence $\delta \in \mathcal{A}[\![G]\!]$ by the induction hypothesis for the second claim. Thus $\alpha \in \mathcal{A}[\![\exists x.G]\!]$ since α and δ agree on $\mathcal{V}G - \{x\}$. □

From now on we assume that a set of **relation symbols** is given, where every relation symbol comes with a natural number specifying the number of arguments it takes.

Let $\mathcal{L}$ be a constraint language and R be a decidable set of relation symbols. An **$\mathcal{L}_R$-atom** has the form $r(\vec{x})$, where the tuple $\vec{x}$ consists of pairwise distinct $\mathcal{L}$-variables and has as

many elements as r has arguments. We extend the constraints of $\mathcal{L}$ by adding all $\mathcal{L}_R$-atoms and the variable mapping of $\mathcal{L}$ by defining

$$\mathcal{V}r(\vec{x}) := \mathcal{V}\vec{x}.$$

If $\mathcal{I}$ is an interpretation of $\mathcal{L}$, we extend $\mathcal{I}$ to a new interpretation $\mathcal{A}$ by leaving the domain of $\mathcal{I}$ unchanged, choosing for every relation symbol $r \in R$ a relation $r^{\mathcal{A}}$ on $\mathbf{D}^{\mathcal{I}}$ taking the right number of arguments, and defining the solution mapping $\mathcal{A}[\![\cdot]\!]$ as follows

$$\begin{aligned} \mathcal{A}[\![\phi]\!] &:= \mathcal{I}[\![\phi]\!] \quad \text{if } \phi \text{ is an } \mathcal{L}\text{-constraint} \\ \mathcal{A}[\![r(\vec{x})]\!] &:= \{\alpha \in \mathrm{ASS}^{\mathcal{I}} \mid \alpha(\vec{x}) \in r^{\mathcal{A}}\}. \end{aligned}$$

In this way we can obtain at least one extension for every interpretation of $\mathcal{L}$. Now, leaving the index set I and the variables VAR of $\mathcal{L}$ unchanged and taking all possible extensions of the interpretations of $\mathcal{L}$, we obtain a new constraint language $\mathcal{L}_R$ that extends $\mathcal{L}$ conservatively.

Proposition 2.2.3 *Let $\mathcal{L}$ be a constraint language, R be a decidable set of relation symbols, ρ be an $\mathcal{L}$-renaming, and F be an $\mathcal{L}_R$-constraint. Then F' is a ρ-variant of F if F' can be obtained from F by replacing every variable x with $\rho(x)$ and every $\mathcal{L}$-constraint ϕ with a ρ-variant of ϕ. Thus $\mathcal{L}_R$ is closed under renaming if $\mathcal{L}$ is closed under renaming.*

If $\mathcal{L}$ is a constraint language and R is a decidable set of relation symbols, then we denote $(\mathcal{L}_R)^*$ with $\mathcal{L}_R^*$. This construction yields Predicate Logic if the constraints of $\mathcal{L}$ are the equations between first-order terms and the interpretations of $\mathcal{L}$ are the usual Tarski interpretations. In $\mathcal{L}_R^*$ an atom $r(s_1, \ldots, s_n)$ takes the form

$$\exists x_1 \ldots \exists x_n.\ (x_1 = s_1 \,\&\, \ldots \,\&\, x_n = s_n \,\&\, r(x_1, \ldots, x_n)),$$

where $x_1, \ldots, x_n$ are pairwise distinct variables not occurring in the argument terms $s_1, \ldots, s_n$.

2.3 Definite Clauses

Here and in the rest of this chapter we assume that $\mathcal{L}$ is a constraint language and R is a decidable set of relation symbols. The letters ϕ and ψ will always denote $\mathcal{L}$-constraints, A and B will always denote $\mathcal{L}_R$-atoms, and F and G will always denote $\mathcal{L}_R^*$-constraints.

A **definite clause** is an $\mathcal{L}_R^*$-implication

$$A_1 \,\&\, \ldots \,\&\, A_n \,\&\, \phi \rightarrow B,$$

where $n \geq 0$, $A_1, \ldots, A_n$ and B are $\mathcal{L}_R$-atoms, and ϕ is an $\mathcal{L}$-constraint. If convenient, we write a clause as $B \leftarrow \phi \,\&\, G$ or $B \leftarrow G$.

A **definite clause specification** is a set of definite clauses.

Conventional logic programs are definite clause specifications over $\mathcal{E}$, where the constraints of $\mathcal{E}$ are conjunctions of equations between first-order terms and the corresponding ground

term algebra is the only interpretation of $\mathcal{E}$. To meet our definition of definite clauses, the clause

$$\mathsf{app(H.R,\ L,\ H.RL)} \leftarrow \mathsf{app(R,\ L,\ RL)},$$

for instance, is rewritten to the equivalent clause

$$\mathsf{app(X,\ L,\ Y)} \leftarrow \mathsf{(X{=}H.R\ \&\ Y{=}H.RL)\ \&\ app(R,\ L,\ RL)}$$

having the conjunction $\mathsf{X{=}H.R\ \&\ Y{=}H.RL}$ as $\mathcal{E}$-constraint.

We will show that the nice properties of conventional logic programs extend to definite clause specifications over arbitrary constraint languages.

The **base** of an $\mathcal{L}_R^*$-interpretation $\mathcal{A}$ is the $\mathcal{L}$-interpretation that $\mathcal{A}$ is extending. Two $\mathcal{L}_R^*$-interpretations are called **base equivalent** if they have the same base.

We define a partial ordering on the set of all $\mathcal{L}_R^*$-interpretations by:

$$\mathcal{A} \subseteq \mathcal{B} \quad :\Longleftrightarrow \quad \mathcal{A} \text{ and } \mathcal{B} \text{ are base equivalent and } \quad \forall\, r \in R.\ r^{\mathcal{A}} \subseteq r^{\mathcal{B}}.$$

Proposition 2.3.1 *Let $\mathcal{A}$ and $\mathcal{B}$ be two $\mathcal{L}_R^*$-interpretations and A be an $\mathcal{L}_R$-atom. Then $\mathcal{A}[\![A]\!] \subseteq \mathcal{B}[\![A]\!]$ if $\mathcal{A} \subseteq \mathcal{B}$.*

The **intersection** $\bigcap_{i\in I} \mathcal{A}_i$ of a family $(\mathcal{A}_i)_{i\in I}$ of base equivalent $\mathcal{L}_R^*$-interpretations is obtained by intersecting the denotations of the relation symbols and is again an $\mathcal{L}_R^*$-interpretation. Analogously, the **union** $\bigcup_{i\in I} \mathcal{A}_i$ of a family $(\mathcal{A}_i)_{i\in I}$ of base equivalent $\mathcal{L}_R^*$-interpretations is obtained by joining the denotations of the relation symbols and is again an $\mathcal{L}_R^*$-interpretation.

Proposition 2.3.2 *Let $\mathcal{I}$ be an $\mathcal{L}$-interpretation. Then the set of all $\mathcal{L}_R^*$-interpretations extending $\mathcal{I}$ is a complete lattice.*

Proposition 2.3.3 *The intersection of a family of base equivalent models of a definite clause specification S is a model of S.*

The following theorem generalizes the key result for conventional logic programs to general definite clause specifications.

Theorem 2.3.4 [Definiteness] *Let S be a definite clause specification in $\mathcal{L}_R^*$ and $\mathcal{I}$ be an $\mathcal{L}$-interpretation. Then the equations*

$$r^{\mathcal{A}_0} := \emptyset, \quad r^{\mathcal{A}_{i+1}} := \{\alpha(\vec{x}) \mid (r(\vec{x}) \leftarrow G) \in S \ \wedge\ \alpha \in \mathcal{A}_i[\![G]\!]\}$$

define a chain $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \cdots$ of $\mathcal{L}_R^*$-interpretations whose base is $\mathcal{I}$. Moreover, the union $\bigcup_{i\geq 0} \mathcal{A}_i$ is the least model of S extending $\mathcal{I}$.

Proof. By induction on i one easily verifies that $\mathcal{A}_i \subseteq \mathcal{A}_{i+1}$. Since every $\mathcal{A}_i$ is an $\mathcal{L}_R^*$-interpretation extending $\mathcal{I}$, the union $\mathcal{A} := \bigcup_{i\geq 0} \mathcal{A}_i$ is an $\mathcal{L}_R^*$-interpretation extending $\mathcal{I}$.

To show that $\mathcal{A}$ is a model of S, let $A \leftarrow G$ be a clause of S and $\alpha \in \mathcal{A}[\![G]\!]$. We have to show that $\alpha \in \mathcal{A}[\![A]\!]$. By the iterative definition of $\mathcal{A}$ we know that there is some i such that $\alpha \in \mathcal{A}_i[\![G]\!]$. Hence $\alpha \in \mathcal{A}_{i+1}[\![A]\!] \subseteq \mathcal{A}[\![A]\!]$.

To show that $\mathcal{A}$ is a minimal model of S, let $\mathcal{B}$ be a base equivalent model of S. By induction on i one verifies easily that $\mathcal{A}_i \subseteq \mathcal{B}$ for every i. Hence $\mathcal{A} = \bigcup_{i \geq 0} \mathcal{A}_i \subseteq \mathcal{B}$. □

A set M of $\mathcal{L}_R^*$-constraints is called a **definite specification** if every $\mathcal{L}$-interpretation can be extended to a minimal model of M. The Definiteness Theorem says that every definite clause specification is a definite specification. Many of the interesting properties of definite clause specifications depend solely on their definiteness. If M is a definite specification in $\mathcal{L}_R^*$, then M uniquely defines the relations of R, that is, for every $\mathcal{L}$-interpretation M defines unique minimal denotations for the relation symbols of R.

A **goal** is a possibly empty conjunction of $\mathcal{L}$-constraints and $\mathcal{L}_R$-atoms. To ease our notation, we identify a goal with the multiset consisting of its constraints.

An **observation** is an implication $\phi \rightarrow G$ consisting of an $\mathcal{L}$-constraint ϕ and a goal G.

Proposition 2.3.5 *Let M be a definite specification. Then an observation is valid in every model of M if and only if it is valid in every minimal model of M.*

Let M be a definite specification. An M**-answer** of a goal G is a satisfiable $\mathcal{L}$-constraint ϕ such that the observation $\phi \rightarrow G$ is valid in every model of M. The preceding proposition says that the M-answers of a goal are completely characterized by the minimal models of M. Thus we say that a set Φ of M-answers of a goal G is **complete** if $\mathcal{A}[\![\Phi]\!]^{\mathcal{V}G} = \mathcal{A}[\![G]\!]^{\mathcal{V}G}$ for every minimal model $\mathcal{A}$ of M.

Proposition 2.3.6 *Let M be a definite specification, G be a goal, ϕ be an M-answer of G, and Φ be a complete set of M-answers of G. Then:*

1. *$\mathcal{I}[\![\phi]\!]^{\mathcal{V}G} \subseteq \mathcal{I}[\![\Phi]\!]^{\mathcal{V}G}$ for every $\mathcal{L}$-interpretation $\mathcal{I}$*
2. *if $\mathcal{L}$ is compact, then there exists a finite subset $\Phi' \subseteq \Phi$ such that $\mathcal{I}[\![\phi]\!]^{\mathcal{V}G} \subseteq \mathcal{I}[\![\Phi']\!]^{\mathcal{V}G}$ for every $\mathcal{L}$-interpretation $\mathcal{I}$.*

Proof. The second claim follows immediately from the first claim. To show the first claim, suppose that $\mathcal{I}$ is an $\mathcal{L}$-interpretation. Since M is definite, there exists a minimal model $\mathcal{A}$ of M whose base is $\mathcal{I}$. Hence

$$\mathcal{I}[\![\phi]\!]^{\mathcal{V}G} = \mathcal{A}[\![\phi]\!]^{\mathcal{V}G} \subseteq \mathcal{A}[\![G]\!]^{\mathcal{V}G} = \mathcal{A}[\![\Phi]\!]^{\mathcal{V}G} = \mathcal{I}[\![\Phi]\!]^{\mathcal{V}G}$$

since ϕ is an M-answer of G and Φ is a complete set of M-answers of G. □

2.4 Operational Semantics

In this section we show that one can obtain a complete interpreter for general definite clause specifications by generalizing the SLD-resolution method [Llo84] employed in conventional

logic programming. Although our proofs are much more general than the proofs for conventional logic programming given in [Llo84], they are clearer and simpler. In particular, we give a new complexity measure based on a multiset ordering that provides for a strong completeness result making a careful distinction between don't care and don't know choices.

In the following we assume that $\mathcal{L}$ and R are given, S is a definite clause specification in $\mathcal{L}_R^*$, and V is a finite set of variables.

We define (S,V)**-goal reduction** as the binary relation $\xrightarrow{\mathrm{r}}_{S,V}$ on the set of goals given by the rule:

$$\begin{aligned} A \,\&\, G \;\xrightarrow{\mathrm{r}}_{S,V}\; & F \,\&\, G \\ & \text{if } A \leftarrow F \text{ is a variant of a clause of } S \\ & \quad \text{such that } (V \cup \mathcal{V}G) \cap \mathcal{V}F \subseteq \mathcal{V}A. \end{aligned}$$

We say that a goal G' is obtained from a goal G by (S,V)**-goal reduction on** A **with** γ if $G \xrightarrow{\mathrm{r}}_{S,V} G'$ by reducing the atom $A \in G$ with a variant of the clause $\gamma \in S$.

Proposition 2.4.1 [Soundness of Goal Reduction] *If $G \xrightarrow{\mathrm{r}}_{S,V} F$, then $\mathcal{A}[\![F]\!] \subseteq \mathcal{A}[\![G]\!]$ for every model $\mathcal{A}$ of S.*

We will now show that goal reduction is a complete rule for inferring S-answers, provided all necessary variants of the clauses of S exist, which is certainly the case if $\mathcal{L}$ and hence $\mathcal{L}_R^*$ are closed under renaming. The most important ingredient of the completeness proof is a well-founded complexity measure on goals that can be decreased by goal reduction. From the Definiteness Theorem we know that every minimal model $\mathcal{A}$ of S can be obtained as the union $\mathcal{A} = \bigcup_{i \geq 0} \mathcal{A}_i$ of a chain $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \cdots$ of $\mathcal{L}_R^*$ -interpretations being uniquely defined for $\mathcal{A}$. This provides for the following definitions:

1. if $\mathcal{A}$ is a minimal model of S, A is an atom and $\alpha \in \mathcal{A}[\![A]\!]$, then the **complexity** of α for A in $\mathcal{A}$ is
$$\textsc{comp}(\alpha, A, \mathcal{A}) := \min\{i \mid \alpha \in \mathcal{A}_i[\![A]\!]\}$$
2. if $\mathcal{A}$ is a minimal model of S, G is a goal, and $\alpha \in \mathcal{A}[\![G]\!]$, then the **complexity** $\textsc{comp}(\alpha, G, \mathcal{A})$ of α for G in $\mathcal{A}$ is the multiset consisting of the complexities $\textsc{comp}(\alpha, A, \mathcal{A})$ of the atoms A in G.

On the multiset complexities, which are finite multisets of natural numbers, we define a well-founded total ordering by

$$\begin{aligned} M \leq M' \;:\Longleftrightarrow\; & \exists \text{ multisets } X \subseteq M \text{ and } X' \subseteq M' \text{ such that} \\ & M = (M' - X') \cup X \;\text{ and} \\ & \forall\, x \in X\; \exists\, x' \in X'.\;\; x < x', \end{aligned}$$

where $\subseteq$, $-$, $\cup$, and $\in$ stand for the appropriate multiset operations (see [DM79] for details on multiset orderings).

Now we are ready for the definition of the complexity measure we are actually going to use. Let $\mathcal{A}$ be a minimal model of S, G be a goal and $\alpha \in \mathcal{A}[\![G]\!]^V$. Then the V**-complexity** of α for G in $\mathcal{A}$ is

$$\textsc{comp}_V(\alpha, G, \mathcal{A}) := \min\{\textsc{comp}(\beta, G, \mathcal{A}) \mid \beta \in \mathcal{A}[\![G]\!] \;\wedge\; \alpha = \beta|_V\},$$

where the minimum is taken with respect to the multiset ordering.

Theorem 2.4.2 [Completeness of Goal Reduction] *Let $\mathcal{L}$ be closed under renaming, $\mathcal{A}$ be a minimal model of S, G be a goal, A be an atom in G, and $\alpha \in \mathcal{A}[\![G]\!]^V$. Then there exists a clause $\gamma \in S$ such that*

1. *(S,V)-goal reduction of G on A using γ is possible*
2. *if G_1 is obtained from G by (S,V)-goal reduction on A using γ, then $\alpha \in \mathcal{A}[\![G_1]\!]^V$ and $\textsc{comp}_V(\alpha, G_1, \mathcal{A}) < \textsc{comp}_V(\alpha, G, \mathcal{A})$.*

Proof. Let $G = A \,\&\, G'$ and $\beta \in \mathcal{A}[\![A \,\&\, G']\!]$ such that $\alpha = \beta|_V$ and $\textsc{comp}_V(\alpha, G, \mathcal{A}) = \textsc{comp}(\beta, G, \mathcal{A})$. Furthermore, let $A = r(\vec{x})$ and $i := \textsc{comp}(\beta, A, \mathcal{A})$. Then $\beta\vec{x} \in r^{\mathcal{A}_i}$. Hence there exists a clause $r(\vec{y}) \leftarrow F$ in S and an assignment $\gamma \in \mathcal{A}_{i-1}[\![F]\!]$ such that $\gamma\vec{y} = \beta\vec{x}$.

Now let ρ be a renaming and $r(\vec{x}) \leftarrow H$ be a ρ-variant of $r(\vec{y}) \leftarrow F$ such that $\vec{x} = \rho(\vec{y})$ and $(V \cup \mathcal{V}G') \cap \mathcal{V}H \subseteq \mathcal{V}r(\vec{x})$. Such a variant always exists since $\mathcal{L}$ and hence $\mathcal{L}_R^*$ are closed under renaming, V is finite, and there are infinitely many variables for every index. Since $H \,\&\, G'$ can be obtained from G by an (S,V)-goal reduction on A, we have the first claim.

To show the second claim, we have to show that $\alpha \in \mathcal{A}[\![H \,\&\, G']\!]^V$ and that $\textsc{comp}_V(\alpha, H \,\&\, G', \mathcal{A}) < \textsc{comp}_V(\alpha, G, \mathcal{A})$.

We know that $\gamma\rho^{-1} \in \mathcal{A}_{i-1}[\![H]\!]$ and that $\gamma\rho^{-1}$ and β agree on $\vec{x}$. Hence there exists an assignment $\delta \in \mathrm{ASS}^{\mathcal{A}}$ that agrees with β on $V \cup \mathcal{V}G'$ and with $\gamma\rho^{-1}$ on $\mathcal{V}H$. One verifies easily that δ agrees with α on V, that $\delta \in \mathcal{A}[\![G']\!]$, and that $\delta \in \mathcal{A}_{i-1}[\![H]\!] \subseteq \mathcal{A}[\![H]\!]$. Hence $\alpha \in \mathcal{A}[\![H \,\&\, G']\!]^V$ and

$$\begin{aligned}
\textsc{comp}_V(\alpha, H \,\&\, G', \mathcal{A}) &\leq \textsc{comp}(\delta, H \,\&\, G', \mathcal{A}) \\
&= \{\textsc{comp}(\delta, H, \mathcal{A})\} \cup \textsc{comp}(\delta, G', \mathcal{A}) \\
&< \{i\} \cup \textsc{comp}(\beta, G', \mathcal{A}) \\
&= \textsc{comp}(\beta, G, \mathcal{A}) = \textsc{comp}_V(\alpha, G, \mathcal{A}).
\end{aligned}$$

□

Corollary 2.4.3 [Weak Completeness of Goal Reduction] *Let $\mathcal{L}$ be closed under renaming, $\mathcal{A}$ be a minimal model of S, G be a goal and $\alpha \in \mathcal{A}[\![G]\!]^V$. Then there exists an S-answer ϕ of G such that $G \xrightarrow{\mathrm{r}}{}^*_{S,V} \phi$ and $\alpha \in \mathcal{A}[\![\phi]\!]^V$.*

Proof. By induction on $\textsc{comp}_V(\alpha, G, \mathcal{A})$, using the Completeness and Soundness Theorems. □

The Completeness Theorem is stronger than the corollary since it makes a careful distinction between don't care and don't know choices: a complete interpreter can choose any atom in the goal to be reduced, has to try all clauses defining the relation symbol of the atom, and can reduce the goal with any suitable variant of the clause being tried.

In conventional logic programming the search space is significantly reduced by exploiting the fact that only clauses whose head unifies with the atom to be reduced need to be tried.

This crucial optimization generalizes nicely to our framework. To show this, we define an additional inference rule, called **V-constraint solving**:

$$\begin{aligned} \phi \,\&\, \phi' \,\&\, G \;\xrightarrow{c}_V\; & \phi'' \,\&\, G \\ & \text{if } \phi \,\&\, \phi' \sim_{V \cup \mathcal{V}G} \phi'' \text{ and} \\ & \quad \phi,\ \phi', \text{ and } \phi'' \text{ are } \mathcal{L}\text{-constraints.} \end{aligned}$$

Proposition 2.4.4 [Constraint Solving] *Let G be a goal and $G \xrightarrow{c}_V G'$. Then:*

1. $\mathcal{A}[\![G]\!]^V = \mathcal{A}[\![G']\!]^V$ *for every interpretation $\mathcal{A}$ of $\mathcal{L}_R^*$*
2. $\textsc{comp}_V(\alpha, G, \mathcal{A}) = \textsc{comp}_V(\alpha, G', \mathcal{A})$ *for every minimal model $\mathcal{A}$ of S and every $\alpha \in \mathcal{A}[\![G]\!]^V$.*

Next we can require that the underlying constraint language $\mathcal{L}$ comes with a set of **normal $\mathcal{L}$-constraints** such that every normal $\mathcal{L}$-constraint is satisfiable, and that for every satisfiable conjunction of $\mathcal{L}$-constraints and every finite set V of variables there exists a V-equivalent normal $\mathcal{L}$-constraint. For conventional logic programming, the normal constraints are the equational representations of idempotent substitutions.

Finally, we can require that the goal to be reduced contains only one $\mathcal{L}$-constraint that has to be normal and that the constraints in the clauses of S be normal. Obviously, a definite clause specification can be transformed to this format without changing its models.

The optimized interpreter works as follows: immediately after a goal reduction step, the constraint solving rule is applied to the conjunction $\phi \,\&\, \phi'$ consisting of the normal constraint from the reduced goal and the normal constraint from the applied clause, where a so-called **constraint solver** attempts to compute a normal constraint that is equivalent to $\phi \,\&\, \phi'$. If the constraint solver detects that $\phi \,\&\, \phi'$ is unsatisfiable, then the interpreter tries immediately another clause since this part of the search space cannot contain any answers. In conventional logic programming, the constraint solver is given by a term unification procedure, where unification succeeds if and only if the corresponding equations are satisfiable in the ground term algebra.

We now give our final Completeness Theorem, which hides how the complexity measure was obtained.

A **complexity measure** on a set M is a partial function from M to a set equiped with a total well-founded ordering.

Theorem 2.4.5 [Completeness] *Let $\mathcal{L}$ be closed under renaming, $\mathcal{A}$ be a minimal model of S, G be a goal, and $\alpha \in \mathcal{A}[\![G]\!]^V$. Then there exists a complexity measure "$\|G\|$" on the set of all goals such that*

1. $\|G\|$ *is defined*
2. *if $\|H\|$ is defined, then $\alpha \in \mathcal{A}[\![H]\!]^V$*
3. *if $\|H\|$ is defined and $H \xrightarrow{c}_V H'$, then $\|H'\| = \|H\|$*

4. *if $\|H\|$ is defined and A is an atom in H, then there exists a clause $\gamma \in S$ such that*
   (a) *(S,V)-goal reduction of H on A using γ is possible*
   (b) *if H' is obtained from H by (S,V)-goal reduction on A using γ, then $\|H'\| < \|H\|$.*

Proof. By the Completeness Theorem for goal reduction and the Constraint Solving Proposition we know that the complexity measure

$$\|H\| := \textsc{comp}_V(\alpha, H, \mathcal{A}) \quad \text{if } \alpha \in \mathcal{A}[\![H]\!]^V$$

satisfies the claims of the theorem. □

2.5 A Type Discipline

A **declaration** is an $\mathcal{L}_R^*$-implication of the form

$$A \;\rightarrow\; \exists y_1 \ldots \exists y_n .\; \phi,$$

where A is an atom, ϕ is a satisfiable $\mathcal{L}$-constraint, and $y_1, \ldots, y_n$ are the variables in $\mathcal{V}\phi - \mathcal{V}A$. For convenience we use the abbreviation $A \rightarrow_\exists \phi$.

Proposition 2.5.1 *A declaration $r(\vec{x}) \rightarrow_\exists \phi$ is* **valid** *in an $\mathcal{L}_R^*$-interpretation $\mathcal{A}$ if and only if $r^{\mathcal{A}} \subseteq \{\alpha(\vec{x}) \mid \alpha \in \mathcal{A}[\![\phi]\!]\}$.*

Declarations prescribe upper bounds for relations. If $\mathcal{L}$ is a constraint language with sorts, typical declarations might be:

```
plus(X,Y,Z) →∃ X:int & Y:int & Z:int
likes(X,Y) →∃ X:person & Y:person.
```

If Feature Logic [Smo88a, Smo89] is employed as the underlying constraint language, the arguments of a relation can be constrained with feature terms employing intersections, unions, complements and feature constraints. Similar declarations are possible using the concept and role descriptions of KL-ONE [BS85, LB87, Neb89, SSS91, NS90]. The idea even applies to conventional logic programming, where we can write declarations like

$$p(x, y) \;\rightarrow\; \exists\, z.\; y = f(x, z).$$

Giving declarations for the relation symbols of a definite clause specification makes it easier to understand the specification since looking at the declarations alone already gives one a rough understanding of the specified relations. Declarations are much easier to understand than clauses since a declaration specifies an upper bound for a relation without recourse to other relations.

We establish an undecidable notion of well-typedness by saying that a definite specification M **satisfies** a set D of declarations if every minimal model of M is a model of D.

Proposition 2.5.2 *Let M be a definite specification and D be a set of declarations. Then the following conditions are equivalent:*

1. *M satisfies D*
2. *$M \cup D$ is a definite specification*
3. *M and $M \cup D$ have the same minimal models.*

Furthermore, if the above conditions are satisfied, then an observation is valid in every model of M if and only if it is valid in every model of $M \cup D$.

Proof. "(1) $\Rightarrow$ (2)". Let $\mathcal{I}$ be an $\mathcal{L}$-interpretation. We have to show that $\mathcal{I}$ can be extended to a minimal model of $M \cup D$. Since M is a definite specification, $\mathcal{I}$ can be extended to a minimal model $\mathcal{A}$ of M. Hence we know by our assumption that $\mathcal{A}$ is a model of $M \cup D$. To show that $\mathcal{A}$ is a minimal model of $M \cup D$, let $\mathcal{B} \subseteq \mathcal{A}$ be a model of $M \cup D$. Then $\mathcal{B}$ is in particular a model of M and hence $\mathcal{B} = \mathcal{A}$ since $\mathcal{A}$ is a minimal model of M.

"(2) $\Rightarrow$ (3)". Let $\mathcal{A}$ be a minimal model of M. Since $M \cup D$ is a definite specification by assumption, we know that $M \cup D$ has a minimal model $\mathcal{B}$ such that $\mathcal{A}$ and $\mathcal{B}$ have the same base. In particular, we know that $\mathcal{B}$ is a model of M. Since $\mathcal{A}$ is a minimal model of M, we know that $\mathcal{A} \subseteq \mathcal{B}$. Since $\mathcal{B}$ is a model of D, we hence know that $\mathcal{A}$ is a model of $M \cup D$. Since $\mathcal{B}$ is a minimal model of $M \cup D$, we thus know that $\mathcal{A} = \mathcal{B}$. Hence $\mathcal{A}$ is a minimal model of $M \cup D$.

Let $\mathcal{A}$ be a minimal model of $M \cup D$. Then $\mathcal{A}$ is a model of M and, since M is definite, M has a minimal model $\mathcal{B} \subseteq \mathcal{A}$. Since $\mathcal{A}$ is a model of D, we know that $\mathcal{B}$ is a model of D. Hence $\mathcal{B}$ is a model of $M \cup D$ and, since $\mathcal{A}$ is a minimal model of $M \cup D$, we know that $\mathcal{A} = \mathcal{B}$. Hence $\mathcal{A}$ is a minimal model of M.

"(3) $\Rightarrow$ (1)". Trivial.

The observational equivalence of M and $M \cup D$ follows from (3) and Proposition 2.3.5. □

In practice, a major advantage of type disciplines is that one can detect specification errors automatically by checking whether a specification is well-typed. This, of course, requires that the well-typedness of a specification is decidable. Our current notion of well-typedness, however, is undecidable even if the underlying constraint language is decidable. We will now devise a stronger more syntactically oriented notion of well-typedness that is decidable if the underlying constraint language is decidable.

An atom A is **well-typed** under an $\mathcal{L}$-constraint ϕ with respect to a declaration δ if $\phi \preceq_{\mathcal{V}A} \psi$ for every variant $A \rightarrow_{\exists} \psi$ of δ. Note that, if A and δ have different relation symbols, then A is well-typed under every $\mathcal{L}$-constraint with respect to δ.

Proposition 2.5.3 *Let ϕ be an $\mathcal{L}$-constraint and $A \rightarrow_{\exists} \psi$ be a variant of a declaration δ. Then A is well-typed under ϕ with respect to δ if and only if $\phi \preceq_{\mathcal{V}A} \psi$.*

Proof. Follows from Proposition 2.1.2. □

Let D be a set of declarations. A definite clause γ is **well-typed** with respect to D if every atom of γ is well-typed under the $\mathcal{L}$-constraint of γ with respect to every declaration of D. (For technical convenience, we don't require that the $\mathcal{L}$-constraint of a well-typed clause be satisfiable.) A definite clause specification S is **well-typed** with respect to D if every clause of S is well-typed with respect to D.

Proposition 2.5.4 *Let $\mathcal{L}$ be a constraint language such that, for every renaming ρ and every finite set V of variables, ρ-variants are computable and V-subsumption is decidable. Then the well-typedness of finite definite clause specifications with respect to finite sets of declarations is decidable.*

Theorem 2.5.5 *Let $\mathcal{L}$ be closed under renaming, S be a definite clause specification and D be a set of declarations. Then S satisfies D if S is well-typed with respect to D.*

Proof. Let $\mathcal{A}$ be a minimal model of S, $r(\vec{x}) \rightarrow_{\exists} \phi$ be a declaration of D, and α be an $\mathcal{A}$-assignment such that $\alpha\vec{x} \in r^{\mathcal{A}}$. We have to show that there exists an assignment $\omega \in \mathcal{A}[\![\phi]\!]$ that agrees with α on $\mathcal{V}\vec{x}$.

Since $\mathcal{L}$ is closed under renaming, we can assume without loss of generality that $\vec{x} = \vec{y}$ for every clause $r(\vec{y}) \leftarrow G$ in S.

Using the construction of the Definiteness Theorem, we know that $\alpha\vec{x} \in r^{\mathcal{A}_{i+1}}$ for some i. Hence there exists a clause $r(\vec{x}) \leftarrow \psi \,\&\, F$ and an assignment $\beta \in \mathcal{A}[\![\psi]\!]$ such that $\beta\vec{x} = \alpha\vec{x}$. Since S is well-typed, we know $\psi \preceq_{\mathcal{V}\vec{x}} \phi$. Hence there exists an assignment $\omega \in \mathcal{A}[\![\phi]\!]$ such that ω agrees with β and hence with α on $\mathcal{V}\vec{x}$. □

A goal G is **well-typed** with respect to D if every atom in G is well-typed under some $\mathcal{L}$-constraint in G with respect to D.

Proposition 2.5.6 [Well-Typed Programs Don't Go Wrong] *Let S be a definite clause specification that is well-typed with respect to a set D of declarations, and let G be a goal that is well-typed with respect to D. Then G' is well-typed with respect to D if G' is obtained from G by (S, V)-goal reduction or V-constraint solving.*

2.6 Type Inference

In the following we assume that S is a definite clause specification and D is a set of declarations.

We will show that, if S satisfies D, one can compute, by superposing the declarations of D with the clauses of S, a definite clause specification S' that is well-typed with respect to D such that S and S' have the same minimal models. Thus S and its well-typed version S' are observationally equivalent. We will also show that, in general, $S' \cup D$ is semantically weaker than $S \cup D$, that is, has more nonminimal models than $S \cup D$.

This result together with the results of the preceding section clarifies the relationship between our two notions of well-typedness. Type inference is also useful for practical applications since one can write an abbreviated definite clause specification S together with a set

D of declarations and automatically infer the "intended" well-typed specification S' satisfying D. If type inference is used for this purpose, it isn't necessary that the abbreviated specification S satisfies D.

We start by defining a quasi-ordering on definite clauses:

$$(A \leftarrow \phi \,\&\, G) \;\preceq\; (A \leftarrow \phi' \,\&\, G) \quad :\Longleftrightarrow \quad \phi' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi.$$

If $\gamma \preceq \gamma'$, we say that γ' is a **weakening** of γ. Note that, if γ' is a weakening of γ, the clauses γ and γ' are equal up to their $\mathcal{L}$-constraints.

To render a clause well-typed with respect to D, we will replace it with a minimal weakening that is well-typed with respect to D. The next propositions says that it doesn't matter which minimal well-typed weakening we choose.

Proposition 2.6.1 *If γ' is a weakening of γ, then every model of γ is a model of γ'.*

To compute minimal well-typed weakenings, we define the following type inference rule for definite clauses:

$$(A \leftarrow \phi \,\&\, G) \quad \xrightarrow{\mathrm{t}}_D \quad (A \leftarrow \phi' \,\&\, G)$$

if B is an atom in $A \,\&\, G$ and
$B \rightarrow_\exists \psi$ is a variant of a declaration of D such that
$\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$,
$\phi \preceq_{\mathcal{V}B} \psi$ does not hold, and
$\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$.

Theorem 2.6.2 [Type Inference] *Let $\mathcal{L}$ be closed under renaming and intersection and let γ be a definite clause. Then:*

1. *there are no infinite chains $\gamma \xrightarrow{\mathrm{t}}_D \gamma_1 \xrightarrow{\mathrm{t}}_D \gamma_2 \xrightarrow{\mathrm{t}}_D \cdots$*
2. *if the type inference rule $\xrightarrow{\mathrm{t}}_D$ cannot be applied to γ, then γ is well-typed with respect to D*
3. *if $\gamma \xrightarrow{\mathrm{t}}_D \gamma'$, then γ' is a weakening of γ such that*
   (a) *if γ'' is a weakening of γ that is well-typed with respect to D, then γ'' is a weakening of γ'*
   (b) *if S satisfies D and S' is obtained from S by replacing γ with γ', then S and S' have the same minimal models.*

Proof. *1.* The clause γ has finitely many pairs (B, δ) such that B is an atom of γ that is not well-typed under the $\mathcal{L}$-constraint of γ with respect to the declaration $\delta \in D$. An application of the type inference rule reduces the number of these pairs.

2. The claim is easily verified using that $\mathcal{L}$ is closed under renaming and intersection.

3. Let $\gamma = (A \leftarrow \phi \,\&\, G)$, $\gamma' = (A \leftarrow \phi' \,\&\, G)$, B be an atom in γ, $B \rightarrow_\exists \psi$ be a variant of a declaration of D such that $\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$, and $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$. Then γ' is obviously a weakening of γ.

3.1. Let $\gamma'' = (A \leftarrow \phi'' \,\&\, G)$ be well-typed with respect to D and let $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi$. We have to show that $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi'$. Since we know that $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$, it suffices to show that $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$. Let $\mathcal{I}$ be an $\mathcal{L}$-interpretation and $\alpha \in \mathcal{I}[\![\phi'']\!]$. We have to show that there exists an assignment $\beta \in \mathcal{I}[\![\phi \,\&\, \psi]\!]$ that agrees with α on $\mathcal{V}A \cup \mathcal{V}G$.

Since $\phi'' \preceq_{\mathcal{V}A \cup \mathcal{V}G} \phi$, we know that there exists an assignment $\beta \in \mathcal{I}[\![\phi]\!]$ that agrees with α on $\mathcal{V}A \cup \mathcal{V}G$. Since B is well-typed under ϕ'' with respect to $B \rightarrow_{\exists} \psi$, we know that $\phi'' \preceq_{\mathcal{V}B} \psi$. Thus there exists an assignment $\omega \in \mathcal{I}[\![\psi]\!]$ that agrees with α and hence with β on $\mathcal{V}B$. Since $\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$, we can assume without loss of generality that ω agrees with β on $\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G$. Thus $\omega \in \mathcal{I}[\![\phi \,\&\, \psi]\!]$ and ω agrees with α on $\mathcal{V}A \cup \mathcal{V}G$.

3.2. Let S satisfy D and let S' be obtained from S by replacing γ with γ'. Furthermore, let $\mathcal{I}$ be an $\mathcal{L}$-interpretation and let $\mathcal{A}_0 \subseteq \mathcal{A}_1 \subseteq \cdots$ and $\mathcal{A}'_0 \subseteq \mathcal{A}'_1 \subseteq \cdots$ be the chains defining the extensions of $\mathcal{I}$ to minimal models of S and S' as in the proof of the Definiteness Theorem. We show by induction on i that $\mathcal{A}_i = \mathcal{A}'_i$ for every $i \geq 0$. For $i = 0$ the claim is trivial. To show $\mathcal{A}_{i+1} = \mathcal{A}'_{i+1}$, it suffices to show that $\mathcal{A}_i[\![\phi \,\&\, G]\!]^{\mathcal{V}A} = \mathcal{A}_i[\![\phi' \,\&\, G]\!]^{\mathcal{V}A}$.

3.2.1. Let $\alpha \in \mathcal{A}_i[\![\phi' \,\&\, G]\!]$. We show that there exists an assignment $\beta \in \mathcal{A}_i[\![\phi \,\&\, G]\!]$ that agrees with α on $\mathcal{V}A$. Since $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$, we know that there exists an assignment $\beta \in \mathcal{A}_i[\![\phi]\!]$ that agrees with α on $\mathcal{V}A \cup \mathcal{V}G$. Hence $\beta \in \mathcal{A}_i[\![\phi \,\&\, G]\!]$.

3.2.2. Let $\alpha \in \mathcal{A}_i[\![\phi \,\&\, G]\!]$ and B be an atom in G. We show that there exists an assignment $\omega \in \mathcal{A}_i[\![\phi' \,\&\, G]\!]$ that agrees with α on $\mathcal{V}A$. Since S satisfies D, we know that $\mathcal{A}_i$ satisfies $B \rightarrow_{\exists} \psi$. Hence there exists an assignment $\beta \in \mathcal{A}_i[\![\psi]\!]$ that agrees with α on $\mathcal{V}B$. Since $\mathcal{V}\psi \cap (\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}B$, we can assume without loss of generality that β agrees with α on $\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G$. Hence $\beta \in \mathcal{A}_i[\![\phi \,\&\, \psi \,\&\, G]\!]$. Since $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$, there exists an assignment $\omega \in \mathcal{A}_i[\![\phi']\!]$ that agrees with β on $\mathcal{V}A \cup \mathcal{V}G$. Hence $\omega \in \mathcal{A}_i[\![\phi' \,\&\, G]\!]$ and ω agrees with α on $\mathcal{V}A$.

3.2.3. Let $\alpha \in \mathcal{A}_i[\![\phi \,\&\, G]\!]$ and $B = A$. We show that there exists an assignment $\omega \in \mathcal{A}_i[\![\phi' \,\&\, G]\!]$ that agrees with α on $\mathcal{V}A$. Since S satisfies D, we know that $\mathcal{A}_{i+1}$ satisfies $A \rightarrow_{\exists} \psi$. Since $\alpha \in \mathcal{A}_{i+1}[\![A]\!]$, there exists an assignment $\beta \in \mathcal{A}_i[\![\psi]\!] = \mathcal{A}_{i+1}[\![\psi]\!]$ that agrees with α on $\mathcal{V}A$. Since $\mathcal{V}\psi \cap (\mathcal{V}\phi \cup \mathcal{V}G) \subseteq \mathcal{V}A$, we can assume without loss of generality that β agrees with α on $\mathcal{V}A \cup \mathcal{V}\phi \cup \mathcal{V}G$. Hence $\beta \in \mathcal{A}_i[\![\phi \,\&\, \psi \,\&\, G]\!]$. Since $\phi' \sim_{\mathcal{V}A \cup \mathcal{V}G} \phi \,\&\, \psi$, there exists an assignment $\omega \in \mathcal{A}_i[\![\phi']\!]$ that agrees with β on $\mathcal{V}A \cup \mathcal{V}G$. Hence $\omega \in \mathcal{A}_i[\![\phi' \,\&\, G]\!]$ and ω agrees with α on $\mathcal{V}A$. □

Corollary 2.6.3 *Let S' be obtained from S by replacing every clause of S by a minimal weakening that is well-typed with respect to D. Then S' is well-typed with respect to D and, if S satisfies D, then S and S' have the same minimal models.*

One could expect that S and S' not only have the same minimal models but have the same models in general. By Proposition 2.6.1 we know that every model of S is a model of S'. However, the following example shows that the other direction doesn't hold. This means that S' is semantically weaker than S in that it allows for more nonminimal models than S.

Example 2.6.4 Let $\mathcal{L}$ be the constraint language whose constraints are conjunctions of equations between first-order terms and let the ground term algebra be the only interpretation of $\mathcal{L}$. Furthermore, let a declaration δ and definite clauses γ and γ' be given as

follows:

$$\begin{aligned} \delta: &\quad p(x) \rightarrow_{\exists} x = a \\ \gamma: &\quad p(x) \leftarrow q(x) \\ \gamma': &\quad p(x) \leftarrow x = a \,\&\, q(x). \end{aligned}$$

The minimal model of γ has empty denotations for p and q and thus trivially satisfies δ. Note that γ' can be obtained from γ with type inference modulo δ. Now let $\mathcal{B}$ be an interpretation such that $p^{\mathcal{B}} = \{a\}$ and $q^{\mathcal{B}}$ is the set of all ground terms (assume that there is more than one). Obviously, $\mathcal{B}$ is a model of γ' and δ but is not a model of γ. $\square$

Chapter 3

POS-Logic

3.1 The Category of POS-Algebras
3.2 POS-Constraints
3.3 The Substitution Theorem
3.4 Sort Rewriting
3.5 Quasi-Extensional Algebras
3.6 Simple Specifications

This chapter presents the logic we will use for the specification of POS-types. The logic is interpreted over partial algebras whose carrier consists of *values* and *sorts* and whose operators are either total functions from sorts to sorts or partial functions from values to values. Sorts are partially ordered by an *inclusion relation* and values and sorts are related by a *membership relation*. To accomplish that, for instance, lists of natural numbers are a subsort of lists of integers, sort functions are required to be monotonic with respect to the inclusion order.

In so-called extensional POS-algebras, sorts are sets of values and inclusion and membership are set inclusion and set membership. The reason for considering nonextensional POS-algebras is that the type specifications we are aiming at do not have extensional initial models in general. Defining types as the initial models of their specifications is, however, the key paradigm of the theory of abstract data types. Nevertheless, we will show that POS-types can also be obtained in a natural way as extensional POS-algebras, where the initial interpretation of a POS-specification will be very closely related to its extensional interpretation. In fact, all operational methods that will be developed in this thesis will work unchanged for both initial and extensional POS-types.

The syntax of POS-Logic distinguishes two kinds of terms denoting sorts and values, re-

spectively. Primitive constraints are available for expressing inclusion between sort terms, membership between value and sort terms, and equality between value terms. This is a significant departure from Many-Sorted and Order-Sorted Logic, where only equality between value terms can be expressed within the logic. Furthermore, POS-Logic does not impose a notion of well-typedness a priori. Instead, different notions of well-typedness can be imposed yielding different specification disciplines. The consequence of not imposing a notion of well-typedness a priory is that the partiality of value functions shows up in that not every value term denotes in every algebra. Since for our purpose total homomorphisms suffice, the presence of partiality doesn't produce any complications.

Our development of POS-Logic goes only as far as is needed for our type specifications. Since our type specifications don't employ equational axioms, we don't study congruences and quotients. And since there exists a rather specialized deduction system for our type specifications, we don't give general deduction rules.

For so-called simple specifications we define a notion of well-typedness such that a ground value term is well-typed in a simple specification T if and only if it denotes in every model of T. As in Many- and Order-Sorted Logic, the well-typed terms of a simple specification T yield term models of T and the ground term model of T is an initial model of T. Well-typedness is established with deduction rules, which will turn out to be sound and complete for simple specifications.

3.1 The Category of POS-Algebras

From now on we assume that two disjoint, decidable sets of function symbols are given whose elements are called **sort function symbols** and **value function symbols**, respectively. The letters ξ, η, ζ will always denote sort function symbols and the letters f, g, h will always denote value function symbols.

A **POS-signature** is a set of sort and value function symbols.

A **POS-algebra** $\mathcal{A}$ consists of

1. a POS-signature $\Sigma^{\mathcal{A}}$,
2. a set $\mathbf{S}^{\mathcal{A}}$ whose elements are called the **sorts** of $\mathcal{A}$
3. a partial order $\leq^{\mathcal{A}}$ on $\mathbf{S}^{\mathcal{A}}$, called the **inclusion order** of $\mathcal{A}$
4. a nonempty set $\mathbf{V}^{\mathcal{A}}$ whose elements are called the **values** of $\mathcal{A}$, where $\mathbf{S}^{\mathcal{A}}$ and $\mathbf{V}^{\mathcal{A}}$ are disjoint
5. a relation $:^{\mathcal{A}} \subseteq \mathbf{V}^{\mathcal{A}} \times \mathbf{S}^{\mathcal{A}}$, called the **membership relation** of $\mathcal{A}$, such that
   (a) for every $a \in \mathbf{V}^{\mathcal{A}}$ there exists an $A \in \mathbf{S}^{\mathcal{A}}$ such that $a :^{\mathcal{A}} A$
   (b) if $a :^{\mathcal{A}} A$ and $A \leq^{\mathcal{A}} B$, then $a :^{\mathcal{A}} B$
6. a denotation $\xi^{\mathcal{A}}$ for every sort function symbol $\xi \in \Sigma^{\mathcal{A}}$, where $\xi^{\mathcal{A}}$ is a total function $\mathbf{S}^{\mathcal{A}} \times \cdots \times \mathbf{S}^{\mathcal{A}} \to \mathbf{S}^{\mathcal{A}}$ taking as many arguments as specified by the arity of ξ and

being monotonic with respect to the inclusion order of $\mathcal{A}$, that is,

$$\vec{A} \leq^{\mathcal{A}} \vec{B} \;\Rightarrow\; \xi^{\mathcal{A}}(\vec{A}) \leq^{\mathcal{A}} \xi^{\mathcal{A}}(\vec{B})$$

7. a denotation $f^{\mathcal{A}}$ for every value function symbol $f \in \Sigma^{\mathcal{A}}$, where $f^{\mathcal{A}}$ is a partial function $\mathbf{V}^{\mathcal{A}} \times \cdots \times \mathbf{V}^{\mathcal{A}} \to \mathbf{V}^{\mathcal{A}}$ taking as many arguments as specified by the arity of f; we use $\mathbf{D}[f^{\mathcal{A}}]$ to denote the domain of $f^{\mathcal{A}}$.

The set $\mathbf{D}^{\mathcal{A}} := \mathbf{S}^{\mathcal{A}} \cup \mathbf{V}^{\mathcal{A}}$ is called the **domain of** $\mathcal{A}$.

A POS-algebra $\mathcal{A}$ is called **extensional** if its sorts are sets, $\mathbf{V}^{\mathcal{A}} = \bigcup \mathbf{S}^{\mathcal{A}}$, its inclusion order is set inclusion, and its membership relation is set membership. General POS-algebras are more general than extensional POS-algebras in that they are not required to satisfy the direction "$\Rightarrow$" of the equivalence

$$(\forall\, a \in \mathbf{V}^{\mathcal{A}}.\;\; a :^{\mathcal{A}} A \Rightarrow a :^{\mathcal{A}} B) \quad \Longleftrightarrow \quad A \leq^{\mathcal{A}} B.$$

Let $\mathcal{A}$ and $\mathcal{B}$ be two POS-algebras such that $\Sigma^{\mathcal{A}} \subseteq \Sigma^{\mathcal{B}}$. A **homomorphism** $\mathcal{A} \to \mathcal{B}$ (read: from $\mathcal{A}$ to $\mathcal{B}$) is a mapping $\gamma\colon \mathbf{D}^{\mathcal{A}} \to \mathbf{D}^{\mathcal{B}}$ such that

1. $\gamma(\leq^{\mathcal{A}}) \subseteq \leq^{\mathcal{B}}$ and $\gamma(:^{\mathcal{A}}) \subseteq :^{\mathcal{B}}$
2. if $\xi \in \Sigma^{\mathcal{A}}$, then $\gamma(\xi^{\mathcal{A}}) \subseteq \xi^{\mathcal{B}}$
3. if $f \in \Sigma^{\mathcal{A}}$, then $\gamma(f^{\mathcal{A}}) \subseteq f^{\mathcal{B}}$.

Note that our definition of homomorphisms is completely natural. It relies on the fact that functions and relations on the domain of a POS-algebra are sets of tuples over the domain and that γ can be extended component-wise to tuples and element-wise to sets. The usual homomorphism equations for functions, for instance,

$$\gamma(f^{\mathcal{A}}(\vec{a})) = f^{\mathcal{B}}(\gamma(\vec{a})),$$

are obviously implied by our definition. Furthermore, our definition treats partial functions in the right way in that we have $\gamma(\mathbf{D}[f^{\mathcal{A}}]) \subseteq \mathbf{D}[f^{\mathcal{B}}]$ for every value function. Moreover, the first condition of our definition implies that $\gamma(\mathbf{S}^{\mathcal{A}}) \subseteq \mathbf{S}^{\mathcal{B}}$ and $\gamma(\mathbf{V}^{\mathcal{A}}) \subseteq \mathbf{V}^{\mathcal{B}}$.

Proposition 3.1.1 *The POS-algebras together with their homomorphisms comprise a category.*

Proof. We must show two things: first, if $\mathcal{A}$ is a POS-algebra, then the identity mapping of the domain of $\mathcal{A}$ is a homomorphism; and second, if $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ are POS-algebras, γ is a homomorphism $\mathcal{A} \to \mathcal{B}$, and δ is a homomorphism $\mathcal{B} \to \mathcal{C}$, then the composition $\delta\gamma$ is a homomorphism $\mathcal{A} \to \mathcal{C}$. It is straightforward to verify these requirements. □

A homomorphism $\gamma\colon \mathcal{A} \to \mathcal{B}$ is called an **isomorphism** if there exists a homomorphism $\gamma'\colon \mathcal{B} \to \mathcal{A}$ such that $\gamma\gamma' = \iota_{\mathcal{A}}$ and $\gamma'\gamma = \iota_{\mathcal{B}}$, where $\iota_{\mathcal{A}}$ and $\iota_{\mathcal{B}}$ are the identity homomorphisms of $\mathcal{A}$ and $\mathcal{B}$, respectively. Two POS-algebras are called **isomorphic** if there exists an isomorphism from one to the other.

Example 3.1.2 Not every bijective homomorphism is an isomorphism. To show this, we define two POS-algebras $\mathcal{A}$ and B as follows: $\Sigma^{\mathcal{A}} = \Sigma^{\mathcal{B}} = \emptyset$, $\mathbf{S}^{\mathcal{A}} = \mathbf{S}^{\mathcal{B}} = \{A, B\}$, $\mathbf{V}^{\mathcal{A}} = \mathbf{V}^{\mathcal{B}} = \{b\}$, $\leq^{\mathcal{A}} = \{(A, A),\ (B, B)\}$, $\leq^{\mathcal{B}} = \leq^{\mathcal{A}} \cup \{(A, B)\}$, and $:^{\mathcal{A}} = :^{\mathcal{B}} = \{(b, B)\}$. Then the identity function of $\mathbf{D}^{\mathcal{A}}$ is a bijective homomorphism $\mathcal{A} \to \mathcal{B}$ but not an isomorphism. □

Intuitively, it would be most appealing to restrict our semantic theory to extensional POS-algebras. However, insisting on this restriction would have the severe drawback that our type specifications would not have initial models in general although they employ only Horn-like axioms. This stems from the fact that the equivalence

$$(\forall\, a \in \mathbf{V}^{\mathcal{A}}.\ \ a :^{\mathcal{A}} A \Rightarrow a :^{\mathcal{A}} B) \quad \Longleftrightarrow \quad A \leq^{\mathcal{A}} B.$$

satisfied by extensional POS-algebras cannot be enforced by Horn-like axioms.

Order-Sorted Logic [GM87a, SNGM89] interprets sorts as sets and still has initial models for every Horn-like specification. This at first surprising difference to POS-Logic is caused by the fact that in an order-sorted algebra every sort is the denotation of a sort symbol in the signature, which allows for a weak noncanonical notion of homomorphism. Since POS-algebras come with sort functions, the notion of an order-sorted homomorphism doesn't generalize to POS-Logic. (Example 3.6.13 will shed more light on this point.)

Giving the semantics of a type specification by the isomorphism class of its initial models is the key paradigm of the theory of abstract data types. Admitting nonextensional POS-algebras will provide for an initial model semantics of our type specifications. Furthermore, the initial models of our type specifications will have associated with them unique extensional models yielding the same theories with respect to membership and equality.

3.2 POS-Constraints

In this section we define a family of constraint languages whose interpretations are POS-algebras. Having the general framework of Chapter 2 available, it suffices to give just the definitions that are specific to POS-Logic.

We employ two disjoint alphabets of variables, called **sort variables** and **value variables**. If $\mathcal{A}$ is a POS-algebra, an **$\mathcal{A}$-assignment** is a mapping from the set of all sort and value variables to the domain of $\mathcal{A}$ such that sort variables are mapped to sorts and value variables are mapped to values. We use $\mathrm{ASS}^{\mathcal{A}}$ to denote the set of all $\mathcal{A}$-assignments.

A **sort term** is a term that is built only from sort variables and sort function symbols. A **value term** is a term that is built only from value variables and value function symbols.

Whenever we are in the context of POS-Logic, we will tacitly assume that x, y, z are value variables, α, β, γ are sort variables, σ, τ, μ, and ν are sort terms, and that s, t, u, and v are value terms.

Let $\mathcal{A}$ be a POS-algebra and δ be an $\mathcal{A}$-assignment. The **δ-denotation** $\mathcal{A}[\![\cdot]\!]_\delta$ is the least partial function from $\Sigma^{\mathcal{A}}$-sort and value terms to the domain of $\mathcal{A}$ satisfying the following

equations:

$$\mathcal{A}[\![\alpha]\!]_\delta = \delta(\alpha), \qquad \mathcal{A}[\![\xi(\vec{\sigma})]\!]_\delta = \xi^{\mathcal{A}}(\mathcal{A}[\![\vec{\sigma}]\!]_\delta),$$
$$\mathcal{A}[\![x]\!]_\delta = \delta(x), \qquad \mathcal{A}[\![f(\vec{s})]\!]_\delta = f^{\mathcal{A}}(\mathcal{A}[\![\vec{s}]\!]_\delta).$$

If σ is a $\Sigma^{\mathcal{A}}$-sort term, then the δ-denotation of σ in $\mathcal{A}$ is always defined and $\mathcal{A}[\![\sigma]\!]_\delta \in \mathbf{S}^{\mathcal{A}}$. If s is a $\Sigma^{\mathcal{A}}$-value term and the δ-denotation of σ in $\mathcal{A}$ is defined, then $\mathcal{A}[\![s]\!]_\delta \in \mathbf{V}^{\mathcal{A}}$. Denotation is strict, that is, a term denotes if and only if each of its subterms denotes. Since the denotation of ground terms does not depend on the employed assignment, we may write $\mathcal{A}[\![\sigma]\!]$ and $\mathcal{A}[\![s]\!]$ rather than $\mathcal{A}[\![\sigma]\!]_\delta$ and $\mathcal{A}[\![s]\!]_\delta$ if σ and s are ground.

There are three kinds of **primitive POS-constraints**, which are given below together with their solutions in a POS-algebra $\mathcal{A}$ (σ and τ are $\Sigma^{\mathcal{A}}$-sort terms and s and t are $\Sigma^{\mathcal{A}}$-value terms):

1. **inclusions:** $\mathcal{A}[\![\sigma \sqsubseteq \tau]\!] = \{\delta \in \mathrm{ASS}^{\mathcal{A}} \mid \mathcal{A}[\![\sigma]\!]_\delta \leq^{\mathcal{A}} \mathcal{A}[\![\tau]\!]_\delta\}$
2. **memberships:** $\mathcal{A}[\![s{:}\,\sigma]\!] = \{\delta \in \mathrm{ASS}^{\mathcal{A}} \mid \mathcal{A}[\![s]\!]_\delta :^{\mathcal{A}} \mathcal{A}[\![\sigma]\!]_\delta\}$
3. **equations:** $\mathcal{A}[\![s \doteq t]\!] = \{\delta \in \mathrm{ASS}^{\mathcal{A}} \mid \mathcal{A}[\![s]\!]_\delta = \mathcal{A}[\![t]\!]_\delta\}$.

Note that $\mathcal{A}[\![\sigma]\!]_\delta$ and $\mathcal{A}[\![s]\!]_\delta$ must be defined if σ and s are terms occurring in a primitive POS-constraint A and $\delta \in \mathcal{A}[\![A]\!]$. Thus our notion of equality is what is known as "existential equality" in the theory of partial algebras: an equation does only hold if both sides are defined.

A **primitive $\Sigma^{\mathcal{A}}$-constraint** is a primitive POS-constraint containing only $\Sigma^{\mathcal{A}}$-sort and value terms. If convenient, we write an inclusion $\sigma \sqsubseteq \tau$ as $\tau \sqsupseteq \sigma$.

Proposition 3.2.1 *Let Σ be a POS-signature. Then the following defines a constraint language $\mathcal{L}(\Sigma)$ that is closed under renaming:*

1. *the variables of $\mathcal{L}(\Sigma)$ are the sort and value variables*
2. *the constraints of $\mathcal{L}(\Sigma)$ are the primitive Σ-constraints*
3. *if A is a constraint of $\mathcal{L}(\Sigma)$, then $\mathcal{V}A$ is the set of all variables occurring in A*
4. *the interpretations of $\mathcal{L}(\Sigma)$ are the POS-algebras whose signature contains Σ, where the solution mappings are defined as above.*

Given a POS-signature Σ, we call $\mathcal{L}(\Sigma)^*$-constraints for convenience just **Σ-constraints**. Furthermore, A and B will always denote primitive Σ-constraints and F and G will always denote Σ-constraints.

Theorem 3.2.2 [Homomorphism] *Let $\mathcal{A}$ and $\mathcal{B}$ be two POS-algebras, γ be a homomorphism $\mathcal{A} \to \mathcal{B}$ and δ be an $\mathcal{A}$-assignment. Then:*

1. *if σ is a $\Sigma^{\mathcal{A}}$-sort term, then $\gamma(\mathcal{A}[\![\sigma]\!]_\delta) = \mathcal{B}[\![\sigma]\!]_{\gamma\delta}$*
2. *if s is a $\Sigma^{\mathcal{A}}$-value term and $\mathcal{A}[\![s]\!]_\delta$ is defined, then $\gamma(\mathcal{A}[\![s]\!]_\delta) = \mathcal{B}[\![s]\!]_{\gamma\delta}$*

3. *if A is a primitive $\Sigma^{\mathcal{A}}$-constraint, then $\gamma\mathcal{A}[\![A]\!] \subseteq \mathcal{B}[\![A]\!]$*

4. *if γ is an isomorphism $\mathcal{A} \to \mathcal{B}$, then γ is a $\mathcal{L}(\Sigma)$- and $\mathcal{L}(\Sigma)^*$-morphism $\mathcal{A} \to \mathcal{B}$.*

Proof. The first two claims can be shown by straightforward inductions over the term structure of σ and s. The third claim follows immediately from the first two claims. The last claim follows from the third claim and Theorem 2.2.2. □

Corollary 3.2.3 *If $\mathcal{A}$ and $\mathcal{B}$ are isomorphic POS-algebras, then an $\Sigma^{\mathcal{A}}$-constraint is valid [satisfiable] in $\mathcal{A}$ if and only if it is valid [satisfiable] in $\mathcal{B}$.*

The next proposition states some obvious properties of POS-Logic:

Proposition 3.2.4 *Let $\mathcal{A}$ be a POS-algebra. Then:*

1. *(Reflexivity) if σ is a $\Sigma^{\mathcal{A}}$-sort term, then $\sigma \sqsubseteq \sigma$ is valid in $\mathcal{A}$*

2. *(Transitivity) if σ, τ and μ are $\Sigma^{\mathcal{A}}$-sort terms, then the implication*

$$\sigma \sqsubseteq \tau \,\&\, \tau \sqsubseteq \mu \;\rightarrow\; \sigma \sqsubseteq \mu$$

   is valid in $\mathcal{A}$

3. *(Monotonicity) if $\xi(\vec{\sigma})$ and $\xi(\vec{\tau})$ are $\Sigma^{\mathcal{A}}$-sort terms, then the implication*

$$\vec{\sigma} \sqsubseteq \vec{\tau} \;\rightarrow\; \xi(\vec{\sigma}) \sqsubseteq \xi(\vec{\tau})$$

   is valid in $\mathcal{A}$

4. *(Compatibility) if s, σ and τ are $\Sigma^{\mathcal{A}}$-terms, then the implication*

$$s{:}\,\sigma \,\&\, \sigma \sqsubseteq \tau \;\rightarrow\; s{:}\,\tau$$

   is valid in $\mathcal{A}$.

3.3 The Substitution Theorem

General Assumption. *In the context of POS-Logic we will always tacitly assume that substitutions map sort terms to sort terms and value terms to value terms.*

A **sort substitution** is a substitution that maps every value variable to itself. A **value substitution** is a substitution that maps every sort variable to itself.

Substitutions are extended to quantifier-free POS-constraints as one would expect (homomorphically with respect to the syntactic structure).

Let Σ be a POS-signature. A **Σ-substitution** is a substitution that maps Σ-sort terms to Σ-sort terms and Σ-value terms to Σ-value terms.

Lemma 3.3.1 *Let $\mathcal{A}$ be a POS-algebra, σ be a $\Sigma^{\mathcal{A}}$-sort term, s be a $\Sigma^{\mathcal{A}}$-value term, and θ be a $\Sigma^{\mathcal{A}}$-substitution. Furthermore, let δ and δ' be $\mathcal{A}$-assignments such that*

$$\begin{aligned}\delta'(x) =& \mathcal{A}[\![\theta x]\!]_\delta \quad \text{if } x \in \mathcal{V}s\\ \delta'(\alpha) =& \mathcal{A}[\![\theta \alpha]\!]_\delta \quad \text{if } \alpha \in \mathcal{V}\sigma.\end{aligned}$$

Then $\mathcal{A}[\![\theta\sigma]\!]_\delta = \mathcal{A}[\![\sigma]\!]_{\delta'}$ and $\mathcal{A}[\![\theta s]\!]_\delta = \mathcal{A}[\![s]\!]_{\delta'}$.

Proof. By straightforward inductions on σ and s. □

Lemma 3.3.2 *Let $\mathcal{A}$ be a POS-algebra, F be a quantifier-free $\Sigma^{\mathcal{A}}$-constraint and θ be a $\Sigma^{\mathcal{A}}$-substitution. Furthermore, let δ and δ' be $\mathcal{A}$-assignments such that*

$$\begin{aligned}\delta'(x) =& \mathcal{A}[\![\theta x]\!]_\delta \quad \text{if } x \in \mathcal{V}F\\ \delta'(\alpha) =& \mathcal{A}[\![\theta \alpha]\!]_\delta \quad \text{if } \alpha \in \mathcal{V}F.\end{aligned}$$

Then $\delta \in \mathcal{A}[\![\theta F]\!]$ if and only if $\delta' \in \mathcal{A}[\![F]\!]$.

Proof. We prove the claim by induction on F.

Let F be the membership $s{:}\sigma$. Then $\delta \in \mathcal{A}[\![\theta s{:}\theta\sigma]\!] \iff \mathcal{A}[\![\theta s]\!]_\delta :^{\mathcal{A}} \mathcal{A}[\![\theta\sigma]\!]_\delta \iff \mathcal{A}[\![s]\!]_{\delta'} :^{\mathcal{A}} \mathcal{A}[\![\sigma]\!]_{\delta'} \iff \delta' \in \mathcal{A}[\![s{:}\sigma]\!]$ using the preceding lemma.

If F is an inclusion or an equation, the claim is proved analogously.

If F is the empty conjunction, then the claim is trivial.

If F is the conjunction $G\&G'$, then $\delta \in \mathcal{A}[\![\theta F]\!]$ if and only if $\delta \in \mathcal{A}[\![\theta G]\!]$ and $\delta \in \mathcal{A}[\![\theta G']\!]$, and $\delta' \in \mathcal{A}[\![F]\!]$ if and only if $\delta' \in \mathcal{A}[\![G]\!]$ and $\delta \in \mathcal{A}[\![G']\!]$. Hence the claim follows by the induction hypothesis.

If F is the negation $\neg G$, then $\delta \in \mathcal{A}[\![\theta F]\!] \iff \delta \in \mathcal{A}[\![\neg\theta G]\!] \iff \delta \notin \mathcal{A}[\![\theta G]\!] \iff \delta' \notin \mathcal{A}[\![G]\!] \iff \delta' \in \mathcal{A}[\![\neg G]\!] \iff \delta' \in \mathcal{A}[\![F]\!]$ using the induction hypothesis. □

Theorem 3.3.3 [Substitution] *Let $\mathcal{A}$ be a POS-algebra, F be a conjunction of primitive $\Sigma^{\mathcal{A}}$-constraints, G be a quantifier-free $\Sigma^{\mathcal{A}}$-constraint and θ be a $\Sigma^{\mathcal{A}}$-substitution such that $\theta x = x$ for every value variable x not occurring in F. Then $\theta F \to \theta G$ is valid in $\mathcal{A}$ if $F \to G$ is valid in $\mathcal{A}$.*

Proof. Suppose $F \to G$ is valid in $\mathcal{A}$ and $\delta \in \mathcal{A}[\![\theta F]\!]$. We have to show that $\delta \in \mathcal{A}[\![\theta G]\!]$. Since F is a conjunction of primitive constraints and $\delta \in \mathcal{A}[\![\theta F]\!]$, $\mathcal{A}[\![\theta x]\!]_\delta$ is defined if x occurs in F. Hence

$$\delta'(\alpha) = \begin{cases} \mathcal{A}[\![\theta\alpha]\!]_\delta & \text{if } \alpha \text{ occurs in } F \to G\\ \delta(\alpha) & \text{otherwise}\end{cases}$$

and

$$\delta'(x) = \begin{cases} \mathcal{A}[\![\theta x]\!]_\delta & \text{if } x \text{ occurs in } F\\ \delta(x) & \text{otherwise}\end{cases}$$

define an $\mathcal{A}$-assignment δ'.

Since $\delta \in \mathcal{A}[\![\theta F]\!]$, we know by the preceding lemma that $\delta' \in \mathcal{A}[\![F]\!]$. Since $F \to G$ is valid in $\mathcal{A}$, we have $\delta' \in \mathcal{A}[\![G]\!]$. Hence we have $\delta \in \mathcal{A}[\![\theta G]\!]$ by the preceding lemma. □

Corollary 3.3.4 [Instantiation of Sort Variables] *Let $\mathcal{A}$ be a POS-algebra, F be a quantifier-free $\Sigma^{\mathcal{A}}$-constraint, and θ be a $\Sigma^{\mathcal{A}}$-sort substitution. Then θF is valid in $\mathcal{A}$ if F is valid in $\mathcal{A}$.*

Proof. Follows from the preceding theorem using the implication $\emptyset \to F$. □

Proposition 3.3.5 *Let $\mathcal{A}$ be a POS-algebra and F be a conjunction of primitive $\Sigma^{\mathcal{A}}$-constraints. Then F is satisfiable in $\mathcal{A}$ if some instance θF of F is satisfiable in $\mathcal{A}$.*

Proof. Suppose $\delta \in \mathcal{A}[\![\theta F]\!]$. Then $\mathcal{A}[\![\cdot]\!]_\delta$ is defined on every sort and value term in θF. Hence there exists an $\mathcal{A}$-assignment δ' such that $\delta'(x) = \mathcal{A}[\![\theta x]\!]_\delta$ if $x \in \mathcal{V}F$ and $\delta'(\alpha) = \mathcal{A}[\![\theta\alpha]\!]_\delta$ if $\alpha \in \mathcal{V}F$. Now we know by the preceding lemma that $\delta' \in \mathcal{A}[\![F]\!]$ and hence that F is satisfiable in $\mathcal{A}$. □

3.4 Sort Rewriting

An inclusion $\sigma \sqsupseteq \tau$ can be regarded as a rewrite rule $\sigma \to \tau$ if σ contains all variables occurring in τ. In this section we will show that sort rewriting is sound, that is, if $\sigma \to_R^* \tau$ and R is obtained from valid inclusions, then the inclusion $\sigma \sqsupseteq \tau$ is valid. Sort rewriting will be the cornerstone of the operational methods for POS-types to be developed in this thesis.

Theorem 3.4.1 [Sort Rewriting] *Let $\mathcal{A}$ be a POS-algebra and I be a set of $\Sigma^{\mathcal{A}}$-inclusions. If for every inclusion $\sigma \sqsupseteq \tau$ in I every variable occurring in τ occurs in σ, then*

$$R(I) = \{\sigma \to \tau \mid (\sigma \sqsupseteq \tau) \in I\}$$

is a rewrite system that rewrites $\Sigma^{\mathcal{A}}$-sort terms to $\Sigma^{\mathcal{A}}$-sort terms. Furthermore, if every inclusion in I is valid in $\mathcal{A}$, then $\sigma \sqsupseteq \tau$ is valid in $\mathcal{A}$ if $\sigma \to_{R(I)}^* \tau$ and σ is a $\Sigma^{\mathcal{A}}$-sort term.

Proof. Let σ be a $\Sigma^{\mathcal{A}}$-sort term and $\sigma \to_{R(I)} \tau$. We show by induction on σ that $\sigma \sqsupseteq \tau$ is valid in $\mathcal{A}$. This suffices since the transitivity property of inclusions yields the rest.

If $\sigma \sqsupseteq \tau$ is an instance of an inclusion in I, then $\sigma \sqsupseteq \tau$ is a $\Sigma^{\mathcal{A}}$-inclusion since σ is a $\Sigma^{\mathcal{A}}$-sort term and $R(I)$ is a $\Sigma^{\mathcal{A}}$-rewrite system. Hence we know by the corollary to the Substitution Theorem that $\sigma \sqsupseteq \tau$ is valid in $\mathcal{A}$.

If $\sigma \sqsupseteq \tau$ is not an instance of an inclusion of I, then $\sigma = \xi(\sigma_1, \ldots, \sigma_n)$, $\tau = \xi(\tau_1, \ldots, \tau_n)$, and, without loss of generality, $\sigma_1 \to_{R(I)} \tau_1$ and $\sigma_i = \tau_i$ for $i \in 2..n$. Now, using the induction hypothesis for $\sigma_1 \sqsupseteq \tau_1$, we know that $\sigma_i \sqsupseteq \tau_i$ is valid in $\mathcal{A}$ for $i \in 1..n$. Hence we know by the monotonicity property of inclusions that $\xi(\sigma_1, \ldots, \sigma_n) \sqsupseteq \xi(\tau_1, \ldots, \tau_n)$ is valid in $\mathcal{A}$. □

3.5 Quasi-Extensional Algebras

A quasi-extensional POS-algebra $\mathcal{A}$ comes with a canonically associated extensional POS-algebra $\mathcal{A}^\circ$ such that $\mathcal{A}[\![F]\!]^V = \mathcal{A}^\circ[\![F]\!]^V$ for every inclusion-free constraint F and every set

V of value variables. It will turn out that the initial algebras of the POS-type specifications we are interested in are quasi-extensional.

Let $\mathcal{A}$ be a POS-algebra. Then the values of a sort $A \in \mathbf{S}^{\mathcal{A}}$ in $\mathcal{A}$ are defined as

$$\mathrm{VAL}^{\mathcal{A}}[A] := \{a \in \mathbf{V}^{\mathcal{A}} \mid a :^{\mathcal{A}} A\}.$$

Obviously, $\mathrm{VAL}^{\mathcal{A}}[A] \subseteq \mathrm{VAL}^{\mathcal{A}}[B]$ if $A \leq^{\mathcal{A}} B$. Furthermore, $\mathrm{VAL}^{\mathcal{A}}[A] = A$ if $\mathcal{A}$ is a extensional POS-algebra.

A POS-algebra $\mathcal{A}$ is called **quasi-extensional** if every sort function $\xi^{\mathcal{A}}$ of $\mathcal{A}$ satisfies

$$\mathrm{VAL}^{\mathcal{A}}[\vec{A}] \subseteq \mathrm{VAL}^{\mathcal{A}}[\vec{B}] \;\Rightarrow\; \mathrm{VAL}^{\mathcal{A}}[\xi^{\mathcal{A}}(\vec{A})] \subseteq \mathrm{VAL}^{\mathcal{A}}[\xi^{\mathcal{A}}(\vec{B})]$$

for every two sort tuples $\vec{A}$ and $\vec{B}$ of the appropriate length.

Construction 3.5.1 [Extensional Algebra $\mathcal{A}^\circ$] *Let $\mathcal{A}$ be a quasi-extensional algebra. Then the following quotient construction defines a extensional POS-algebra $\mathcal{A}^\circ$:*

1. $\Sigma^{\mathcal{A}^\circ} := \Sigma^{A}$
2. $\mathbf{S}^{\mathcal{A}^\circ} := \{\mathrm{VAL}^{\mathcal{A}}[A] \mid A \in \mathbf{S}^{\mathcal{A}}\}$
3. $\mathbf{V}^{\mathcal{A}^\circ} := \mathbf{V}^{\mathcal{A}}$
4. $\xi^{\mathcal{A}^\circ}(\mathrm{VAL}^{\mathcal{A}}[\vec{A}]) := \mathrm{VAL}^{\mathcal{A}}[\xi^{\mathcal{A}}(\vec{A})]$
5. $f^{\mathcal{A}^\circ} := f^{\mathcal{A}}$.

Furthermore, $\kappa(a) := a$ if $a \in \mathbf{V}^{\mathcal{A}}$ and $\kappa(A) := \mathrm{VAL}^{\mathcal{A}}[A]$ if $A \in \mathbf{S}^{\mathcal{A}}$ defines a surjective homomorphism $\kappa\colon \mathcal{A} \to \mathcal{A}^\circ$.

If Σ is a POS-signature, the constraint language $\mathcal{L}_o(\Sigma)$ is obtained from $\mathcal{L}(\Sigma)$ by admitting only memberships and equations as constraints.

Theorem 3.5.2 *If $\mathcal{A}$ is a quasi-extensional POS-algebra, then $\mathcal{A}^\circ$ is a extensional POS-algebra and κ is a surjective $\mathcal{L}_o(\Sigma^{\mathcal{A}})^*$-morphism $\mathcal{A} \to \mathcal{A}^\circ$.*

Proof. It is straightforward to verify that $\mathcal{A}^\circ$ is in fact a extensional POS-algebra and that κ is a surjective homomorphism $\mathcal{A} \to \mathcal{A}^\circ$.

To show that κ is an $\mathcal{L}_o(\Sigma^{\mathcal{A}})^*$-morphism, it suffices by Theorem 2.2.2 to show that κ is an $\mathcal{L}_o(\Sigma^{\mathcal{A}})$-morphism. Since κ is surjective, we have $\kappa\mathrm{ASS}^{\mathcal{A}} = \mathrm{ASS}^{\mathcal{A}^\circ}$. Since κ is a homomorphism, we have $\kappa\mathcal{A}[\![A]\!] \subseteq \mathcal{A}^\circ[\![A]\!]$ for every primitive $\Sigma^{\mathcal{A}}$-constraint. It remains to show that $\{\delta \in \mathrm{ASS}^{\mathcal{A}} \mid \kappa\delta \in \mathcal{A}^\circ[\![A]\!]\} \subseteq \mathcal{A}[\![A]\!]$ for every primitive $\Sigma^{\mathcal{A}}$-constraint that isn't an inclusion.

Let $\delta \in \mathrm{ASS}^{\mathcal{A}}$ and $\mathcal{A}^\circ[\![s]\!]_{\kappa\delta} \in \mathcal{A}^\circ[\![\sigma]\!]_{\kappa\delta}$. Then $\mathcal{A}^\circ[\![s]\!]_{\kappa\delta} = \mathcal{A}[\![s]\!]_{\delta}$ since κ doesn't change values and $\mathcal{A}$ and $\mathcal{A}^\circ$ have the same values and value functions. Since κ is a homomorphism, we know by the Homomorphism Theorem that $\mathcal{A}^\circ[\![\sigma]\!]_{\kappa\delta} = \kappa\mathcal{A}[\![\sigma]\!]_{\delta} = \mathrm{VAL}^{\mathcal{A}}[\mathcal{A}[\![\sigma]\!]_{\delta}]$. Hence $\mathcal{A}[\![s]\!]_{\delta} :^{\mathcal{A}} \mathcal{A}[\![\sigma]\!]_{\delta}$ since $\mathcal{A}[\![s]\!]_{\delta} \in \mathrm{VAL}^{\mathcal{A}}[\mathcal{A}[\![\sigma]\!]_{\delta}]$.

For equations the claim is obvious since κ doesn't change values and $\mathcal{A}$ and $\mathcal{A}^\circ$ have the same values and the same value functions. $\square$

Corollary 3.5.3 *Let $\mathcal{A}$ be a quasi-extensional POS-algebra and F be an inclusion-free $\Sigma^{\mathcal{A}}$-constraint. Then:*

1. $\mathcal{A}^\circ[\![F]\!] = \kappa\mathcal{A}[\![F]\!]$ *and* $\mathcal{A}[\![F]\!] = \{\delta \in \mathrm{ASS}^{\mathcal{A}} \mid \kappa\delta \in \mathcal{A}^\circ[\![F]\!]\}$
2. *if V is a set of value variables, then* $\mathcal{A}[\![F]\!]^V = \mathcal{A}^\circ[\![F]\!]^V$
3. *F is valid [satisfiable] in $\mathcal{A}$ if and only if F is valid [satisfiable] in $\mathcal{A}^\circ$*
4. *if $\sigma \sqsubseteq \tau$ is a $\Sigma^{\mathcal{A}}$-inclusion and $F \to \sigma \sqsubseteq \tau$ is valid in $\mathcal{A}$, then $F \to \sigma \sqsubseteq \tau$ is valid in $\mathcal{A}^\circ$.*

Proof. The first, second and third claim are immediate consequences of the fact that κ is a surjective $\mathcal{L}_o(\Sigma^{\mathcal{A}})^*$-morphism.

To show the fourth claim, suppose $F \to \sigma \sqsubseteq \tau$ is valid in $\mathcal{A}$ and let $\delta \in \mathcal{A}^\circ[\![F]\!]$. We have to show that $\delta \in \mathcal{A}^\circ[\![\sigma \sqsubseteq \tau]\!]$. Since κ is an $\mathcal{L}_o(\Sigma^{\mathcal{A}})^*$-morphism and $F \to \sigma \sqsubseteq \tau$ is valid in $\mathcal{A}$, there exists a $\lambda \in \mathcal{A}[\![\sigma \sqsubseteq \tau]\!]$ such that $\kappa\lambda = \delta$. Since κ is a homomorphism, we have by the Homomorphism Theorem that $\delta = \kappa\lambda \in \mathcal{A}^\circ[\![\sigma \sqsubseteq \tau]\!]$. □

3.6 Simple Specifications

We are now ready to define a class of specifications, called simple specifications, that do have initial models. For simple specifications we define a notion of well-typedness such that the the set of all well-typed ground terms yields an initial model. Well-typedness is established with deduction rules that are sound and complete. Since simple specifications don't allow for equational axioms, there is only one natural notion of well-typedness: a value term is well-typed if and only if it denotes in every model of the specification.

Simple specifications are still to permissive to enjoy a well-behaved operational semantics. Furthermore, their initial models are in general not quasi-extensional. The subclass of simple specifications that yields operationally well-behaved POS-types will be defined in Chapter 5. In Chapter 4 we will study rewrite systems obtained from inclusional axioms, whose properties are crucial for the operational semantics of POS-types.

A **rank** for a value function symbol f is an implication

$$x_1{:}\sigma_1 \;\&\; \ldots \;\&\; x_n{:}\sigma_n \;\to\; f(x_1,\ldots,x_n){:}\sigma$$

such that $n \geq 0$ and $x_1,\ldots,x_n$ are pairwise distinct value variables. Since the validity of a rank in a POS-algebra does not depend on the particular value variables employed, the abbreviated notation

$$f{:}\sigma_1\cdots\sigma_n \to \sigma,$$

which can be abbreviated even further to $f{:}\vec{\sigma} \to \sigma$, can be used. The tuple $\vec{\sigma}$ is called the **domain** of the rank and σ is called the **codomain** of the rank. If the rank $f{:}\vec{\sigma} \to \sigma$ is used as an axiom, it requires that f is at least defined on $\vec{\sigma}$, and that f maps arguments in $\vec{\sigma}$ to elements of σ.

A **simple specification** is a set T of inclusions and ranks such that

1. T contains a least one value constant and at least one sort constant
2. the inclusions of T yield a rewrite system $R(T)$ if every inclusion $\sigma \sqsupseteq \tau$ is taken as a rewrite rule $\sigma \to \tau$
3. the rewrite relation "$\sigma \to^*_{R(T)} \tau$" is a partial order.

If T is a simple specification, we write $\sigma \geq_T \tau$ or $\tau \leq_T \sigma$ if $\sigma \to^*_{R(T)} \tau$, and $\theta \geq_T \psi$ or $\psi \leq_T \theta$ if $\theta \to^*_{R(T)} \psi$ and θ and ψ agree on all value variables. In anticipation of a completeness result to be shown shortly, we call "$\sigma \sqsubseteq \tau$" the **inclusion order of** T. The set of all sort and value functions occurring in T is called the **signature of** T and will be denoted by Σ^T.

General Assumption. *In this section we assume that T is a simple specification and that all terms and substitutions employ only function symbols in Σ^T. Furthermore, we tacitly assume that all constraints and interpretations are taken from $\mathcal{L}(\Sigma^T)^*$.*

A **prefix** is a conjunction

$$x_1{:}\,\sigma_1 \& \ldots \& x_n{:}\,\sigma_n$$

such that $n \geq 0$ and $x_1, \ldots, x_n$ are pairwise distinct value variables. The letters P and Q will always denote prefixes.

Prefixes will be used to qualify variables with sorts and are a central notion in our formalism. Many-Sorted and Order-Sorted Logic can do without prefixes since they stipulate that every variable has a fixed sort. This would be possible for POS-Logic as well, but it turns out that this notational trick has some rather unpleasant consequences. First, it is responsible for the annoying empty sort problem plaguing Many-Sorted and Order-Sorted Logic. Second, built-in sorts for variables cause great notational inconvenience with order-sorted unification [MGS89, SNGM89, Wal88], where they require the introduction of auxiliary variables in order to change the sort qualification of a variable.

Let $P = (x_1{:}\,\sigma_1 \& \ldots \& x_n{:}\,\sigma_n)$ be a prefix. Then $\mathcal{D}P := \{x_1, \ldots, x_n\}$ is called the **domain** of P. Moreover, P defines a mapping $Px_i := \sigma_i$ from $\mathcal{D}P$ to the set of sort terms. If V is a set of value variables, then the **restriction** of P to V is defined as

$$P|_V := \{x{:}\,Px \mid x \in \mathcal{D}P \cap V\}.$$

The inclusion order is extended to prefixes as follows:

$$P \leq_T Q \;:\Longleftrightarrow\; \mathcal{D}P = \mathcal{D}Q \;\wedge\; \forall\, x \in \mathcal{D}P.\;\; Px \leq_T Qx.$$

Note that "$P \leq_T Q$" is a partial order on the set of all prefixes.

Let P be a prefix. The **membership relation** "$P \vdash_T s{:}\,\sigma$" of T (read: "s is in σ under P") is defined as follows:

1. $P \vdash_T x{:}\,\sigma$ if and only if $Px \leq_T \sigma$
2. $P \vdash_T f(\vec{s}){:}\,\sigma$ if and only if there exists an instance $f{:}\,\vec{\mu} \to \tau$ of a rank in T such that $P \vdash_T \vec{s}{:}\,\vec{\mu}$ and $\tau \leq_T \sigma$.

We say that a value term s is **well-typed** under a prefix P if there exists a sort term σ such that $P \vdash_T s{:}\sigma$. We write $\vdash_T s{:}\sigma$ if $\emptyset \vdash_T s{:}\sigma$.

A **membership system** is a possibly empty conjunction of memberships. If M is a membership system, we write $P \vdash_T M$ if $P \vdash_T s{:}\sigma$ for every membership $s{:}\sigma \in M$. Note that every prefix is a membership system.

Proposition 3.6.1 *The membership relation "$P \vdash_T s{:}\sigma$" has the following properties:*

1. *(Freeness) if $P \subseteq Q$ and $\mathcal{V}s \subseteq \mathcal{D}P$, then $P \vdash_T s{:}\sigma$ if and only if $Q \vdash_T s{:}\sigma$*
2. *(Compatibility) if $P \vdash_T s{:}\sigma$ and $\sigma \leq_T \tau$, then $P \vdash_T s{:}\tau$*
3. *(Substitutability) if $P \vdash_T s{:}\sigma$ and $Q \vdash_T \theta P$, then $Q \vdash_T \theta s{:}\theta\sigma$.*

Proof. The first and second claim are obvious from the definition of "$P \vdash_T s{:}\sigma$". To show the third claim, suppose $P \vdash_T s{:}\sigma$ and $Q \vdash_T \theta P$. We show by induction on s that $Q \vdash_T \theta s{:}\theta\sigma$.

Let $s = x$. Then $Px \leq_T \sigma$ and hence $\theta Px \leq_T \theta\sigma$. Since $Q \vdash_T \theta P$, we know in particular that $Q \vdash_T \theta x{:}\theta Px$. Hence $Q \vdash_T \theta x{:}\theta\sigma$ by the compatibility of "$P \vdash_T s{:}\sigma$".

Let $s = f(\vec{s})$. Then there exists an instance $f{:}\vec{\mu} \to \tau$ of a rank in T such that $\tau \leq_T \sigma$ and $P \vdash_T \vec{s}{:}\vec{\mu}$. By the induction hypothesis we know that $Q \vdash_T \theta\vec{s}{:}\theta\vec{\mu}$. Hence $Q \vdash_T \theta f(\vec{s}){:}\theta\sigma$ since $f{:}\theta\vec{\mu} \to \theta\tau$ is an instance of a rank of T and $\theta\tau \leq_T \theta\sigma$. □

Proposition 3.6.2 [Soundness] *If $P \vdash_T s{:}\sigma$, then the implication $P \to s{:}\sigma$ is valid in T. Furthermore, if $\sigma \leq_T \tau$, then the inclusion $\sigma \sqsubseteq \tau$ is valid in T.*

Proof. The soundness of sort rewriting has already been proven in general. To show the soundness of "$P \vdash_T s{:}\sigma$", suppose $P \vdash_T s{:}\sigma$ and $\mathcal{A}$ is a model of T. We show by induction on s that the implication $P \to s{:}\sigma$ is valid in $\mathcal{A}$.

If $s = x$, then $Px \leq_T \sigma$ and hence $Px \sqsubseteq \sigma$ is valid in $\mathcal{A}$. Since $\mathcal{A}[\![P]\!] \subseteq \mathcal{A}[\![x{:}Px]\!]$, we know that $P \to x{:}\sigma$ is valid in $\mathcal{A}$.

If $s = f(\vec{s})$, then there exists an instance $f{:}\vec{\mu} \to \tau$ of a rank in T such that $\tau \leq_T \sigma$ and $P \vdash_T \vec{s}{:}\vec{\mu}$. By the Substitution Theorem we know that $\vec{s}{:}\vec{\mu} \to f(\vec{s}){:}\tau$ is valid in $\mathcal{A}$, and by the induction hypothesis we know that $P \to \vec{s}{:}\vec{\mu}$ is valid in $\mathcal{A}$. Hence we know by Modus Ponens that $P \to f(\vec{s}){:}\tau$ is valid in $\mathcal{A}$. Since $\tau \leq_T \sigma$, the inclusion $\sigma \sqsubseteq \tau$ and hence the implication $P \to f(\vec{s}){:}\sigma$ is valid in $\mathcal{A}$. □

Construction 3.6.3 [Term Algebra $\mathcal{I}(T,P)$] *If P is a prefix, then the following defines a POS-algebra $\mathcal{I}(T,P)$:*

1. $\Sigma^{\mathcal{I}(T,P)} := \Sigma^T$
2. $\mathbf{S}^{\mathcal{I}(T,P)}$ *is the set of all Σ^T-sort terms containing only sort variables that occur in P*
3. $\sigma \leq^{\mathcal{I}(T,P)} \tau$ *if and only if $\sigma \leq_T \tau$ and $\tau \in \mathbf{S}^{\mathcal{I}(T,P)}$*

4. $\mathbf{V}^{\mathcal{I}(T,P)}$ *is the set of all value terms that are well-typed in* T *under* P
5. $s :^{\mathcal{I}(T,P)} \sigma$ *if and only if* $P \vdash_T s{:}\,\sigma$ *and* $\sigma \in \mathbf{S}^{\mathcal{I}(T,P)}$
6. $\xi^{\mathcal{I}(T,P)}(\vec{\sigma}) := \xi(\vec{\sigma})$
7. $f^{\mathcal{I}(T,P)}(\vec{s}) := f(\vec{s})$ *if and only if* $f(\vec{s})$ *is well-typed in* T *under* P.

Proposition 3.6.4 *If* P *is a prefix, then* $\mathcal{I}(T,P)$ *is a POS-algebra.*

Proof. Since T contains a value constant c, T contains a rank $c{:}\,\sigma$. Since T contains a sort constant there exists a ground instance $c{:}\,\tau$ of $c{:}\,\sigma$. Hence $P \vdash_T c{:}\,\tau$ and thus c is a value of $\mathcal{I}(T,P)$. If $P \vdash_T s{:}\,\sigma$ and σ contains sort variables that don't occur in P, then we know by the Substitutivity of "$P \vdash_T s{:}\,\sigma$" that these "superfluous" sort variables can be replaced by a sort constant of T. Furthermore, we have required in the definition of simple specifications that "$\sigma \leq_T \tau$" is a partial order. With that it is straightforward to verify that $\mathcal{I}(T,P)$ is a POS-algebra. □

Proposition 3.6.5 *For every* $\mathcal{I}(T,P)$*-assignment* δ *there exists a unique substitution* θ_δ *that extends* $\mathcal{I}(T,P)[\![\cdot]\!]_\delta$. *Furthermore,* $\mathcal{I}(T,P)[\![s]\!]_\delta$ *is defined if and only if* $\theta_\delta s$ *is well-typed under* P.

Proof. The claims follow by straightforward inductions on s. □

Proposition 3.6.6 *Let* σ *and* s *be ground terms. Then* $\mathcal{I}(T,P)[\![\sigma]\!] = \sigma$ *and, if* $\mathcal{I}(T,P)[\![s]\!]$ *is defined,* $\mathcal{I}(T,P)[\![s]\!] = s$.

Proof. Let δ be an $\mathcal{I}(T,P)$-assignment. The the claim follows from the preceding proposition since $\mathcal{I}(T,P)[\![\sigma]\!] = \mathcal{I}(T,P)[\![\sigma]\!]_\delta$, $\mathcal{I}(T,P)[\![s]\!] = \mathcal{I}(T,P)[\![s]\!]_\delta$, $\sigma = \theta_\delta\sigma$ and $s = \theta_\delta s$. □

Proposition 3.6.7 *If* θQ *is ground and valid in* $\mathcal{I}(T,P)$, *then there exists an assignment* $\delta \in \mathcal{I}(T,P)[\![Q]\!]$ *that agrees with* θ *on every variable occurring in* Q.

Proof. Let θQ be ground and valid in $\mathcal{I}(T,P)$. Then $\theta x \in \mathbf{V}^{\mathcal{I}(T,P)}$ if x occurs in Q and $\theta\alpha \in \mathbf{S}^{\mathcal{I}(T,P)}$ if α occurs in Q. Hence there exists an assignment $\delta \in \mathcal{I}(T,P)[\![Q]\!]$ that agrees with θ on every variable occurring in Q. □

Theorem 3.6.8 [Term Model] *For every prefix* P *the term algebra* $\mathcal{I}(T,P)$ *is a model of* T.

Proof. We have to show to show that $\mathcal{I}(T,P)$ satisfies every inclusion and every rank of T.

Let $\sigma \sqsubseteq \tau$ be an inclusional axiom of T and let δ be an $\mathcal{I}(T,P)$-assignment. Then $\theta_\delta\sigma \leq_T \theta_\delta\tau$ and hence $\delta \in \mathcal{I}(T,P)[\![\sigma \sqsubseteq \tau]\!]$.

Let $\vec{x}{:}\,\vec{\sigma} \to f(\vec{x}){:}\,\tau$ be a rank of T and let $\delta \in \mathcal{I}(T,P)[\![\vec{x}{:}\,\vec{\sigma}]\!]$. By the construction of $\mathcal{I}(T,P)$ we know that $P \vdash_T \theta_\delta\vec{x}{:}\,\theta_\delta\vec{\sigma}$. Since $f{:}\,\theta_\delta\vec{\sigma} \to \theta_\delta\tau$ is an instance of a rank of T, we thus have $P \vdash_T f(\theta_\delta\vec{x}){:}\,\theta_\delta\tau$. Hence $\delta \in \mathcal{I}(T,P)[\![f(\vec{x}){:}\,\tau]\!]$. □

Lemma 3.6.9 *Let $\mathcal{A}$ be a POS-algebra, γ be a homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$, and δ be an $\mathcal{A}$-assignment that agrees with γ on every variable occurring in P. Then $\gamma(\sigma) = \mathcal{A}[\![\sigma]\!]_\delta$ if $\mathcal{V}\sigma \subseteq \mathcal{V}P$ and $\gamma(s) = \mathcal{A}[\![s]\!]_\delta$ if $P \vdash_T s{:}\,\sigma$.*

Proof. We prove the second claim and omit the proof of the first claim, which is analogous to the proof of the second claim.

Let $P \vdash_T s{:}\,\sigma$. Then s is a value of $\mathcal{I}(T,P)$ and hence γ is defined on s. Furthermore, every variable occurring in s occurs in P. We prove by induction on s that $\gamma(s) = \mathcal{A}[\![s]\!]_\delta$.

If $s = x$, then $\gamma(x) = \delta(x) = \mathcal{A}[\![x]\!]_\delta$.

If $s = f(\vec{s})$, then $\gamma(f(\vec{s})) = \gamma(f^{\mathcal{I}(T,P)}(\vec{s})) = f^{\mathcal{A}}(\gamma(\vec{s}))$. Since $P \vdash_T f(\vec{s}){:}\,\sigma$, we know by the definition of $\vdash_T$ that there exists a tuple $\vec{\mu}$ such that $P \vdash_T \vec{s}{:}\,\vec{\mu}$. Hence we know by the induction hypothesis that $\gamma(\vec{s}) = \mathcal{A}[\![\vec{s}]\!]_\delta$. Thus $\gamma(f(\vec{s})) = f^{\mathcal{A}}(\mathcal{A}[\![\vec{s}]\!]_\delta) = \mathcal{A}[\![f(\vec{s})]\!]_\delta$. □

Theorem 3.6.10 [Freeness] *Let $\mathcal{A}$ be a model of T and P be a prefix. Then:*

1. *if γ is a homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$, then there exists an $\mathcal{A}$-assignment $\delta \in \mathcal{A}[\![P]\!]$ that agrees with γ on every variable occurring in P*
2. *if $\delta \in \mathcal{A}[\![P]\!]$, then the restriction of $\mathcal{A}[\![\cdot]\!]_\delta$ to $\mathbf{D}^{\mathcal{I}(T,P)}$ is a homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$ and there exists no other homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$ that agrees with δ on every variable occurring in P.*

Proof. *1.* Let γ be a homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$, $a \in \mathbf{V}^{\mathcal{A}}$ and $A \in \mathbf{S}^{\mathcal{A}}$. Then

$$\delta(\alpha) = \begin{cases} \gamma(\alpha) & \text{if } \alpha \text{ occurs in } P \\ a & \text{otherwise} \end{cases}$$

and

$$\delta(x) = \begin{cases} \gamma(x) & \text{if } x \text{ occurs in } P \\ A & \text{otherwise} \end{cases}$$

define an $\mathcal{A}$-assignment δ.

Let $x{:}\,\sigma$ be a membership in P. Since $\mathcal{A}[\![x]\!]_\delta = \delta(x) = \gamma(x)$ and $\mathcal{A}[\![\sigma]\!]_\delta = \gamma(\sigma)$ by the preceding lemma, it suffices to show that $\gamma(x) :^{\mathcal{A}} \gamma(\sigma)$. Since $P \vdash_T x{:}\,\sigma$, we know $x :^{\mathcal{I}(T,P)} \sigma$. Thus we have $\gamma(x) :^{\mathcal{A}} \gamma(\sigma)$ since γ is a homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$.

2. Let $\delta \in \mathcal{A}[\![P]\!]$. We show that the restriction of $\mathcal{A}[\![\cdot]\!]_\delta$ to $\mathbf{D}^{\mathcal{I}(T,P)}$ is a homomorphism $\mathcal{I}(T,P) \to \mathcal{A}$.

Suppose $\sigma \leq^{\mathcal{I}(T,P)} \tau$. Then $\sigma \leq_T \tau$. Hence we know by the Soundness Proposition that $\sigma \sqsubseteq \tau$ is valid in $\mathcal{A}$. Thus $\mathcal{A}[\![\sigma]\!]_\delta \leq^{\mathcal{A}} \mathcal{A}[\![\tau]\!]_\delta$.

Suppose $s :^{\mathcal{I}(T,P)} \sigma$. Then $P \vdash_T s{:}\,\sigma$. Hence we know by the Soundness Proposition that $P \to s{:}\,\sigma$ is valid in $\mathcal{A}$. Since $\delta \in \mathcal{A}[\![P]\!]$, we have $\mathcal{A}[\![s]\!]_\delta :^{\mathcal{A}} \mathcal{A}[\![\sigma]\!]_\delta$.

It is straightforward to verify that $\mathcal{A}[\![\xi^{\mathcal{I}(T,P)}]\!]_\delta \subseteq \xi^{\mathcal{A}}$ and $\mathcal{A}[\![f^{\mathcal{I}(T,P)}]\!]_\delta \subseteq f^{\mathcal{A}}$ for the sort and value function symbols occurring in T, respectively.

The uniqueness follows from the preceding lemma. □

Theorem 3.6.11 [Initiality] *$\mathcal{I}(T) := \mathcal{I}(T,\emptyset)$ is an initial model of T.*

Proof. We have shown that for every prefix P the term algebra $\mathcal{I}(T,P)$ is a model of T. Hence $\mathcal{I}(T)$ is a model of T.

Let $\mathcal{A}$ be a model of T. We have to show that there exists a unique homomorphism $\mathcal{I}(T) \to \mathcal{A}$. Since there exists an assignment $\delta \in \mathcal{A}[\![\emptyset]\!]$, we know by the Freeness Theorem that the restriction of $\mathcal{A}[\![\cdot]\!]_\delta$ to $\mathbf{D}^{\mathcal{I}(T)}$ is a homomorphism $\mathcal{I}(T) \to \mathcal{A}$ and that there exists no other homomorphism $\mathcal{I}(T) \to \mathcal{A}$. □

The following example shows that, in general, the initial models of simple specifications are not quasi-extensional.

Example 3.6.12 Let T be the simple specification

$$\mathsf{b}\colon \mathsf{F}(\mathsf{B}), \quad \mathsf{A} \sqsubseteq \mathsf{B},$$

where b is a value constant, A and B are sort constants, and F is a unary sort function symbol. Then $\mathcal{I}(T)$ is not quasi-extensional since $\mathrm{VAL}^{\mathcal{I}(T)}[\mathsf{A}] = \mathrm{VAL}^{\mathcal{I}(T)}[\mathsf{B}] = \mathrm{VAL}^{\mathcal{I}(T)}[\mathsf{F}(\mathsf{A})] = \emptyset$ and $\mathrm{VAL}^{\mathcal{I}(T)}[\mathsf{F}(\mathsf{B})] = \{\mathsf{b}\}$. □

The next example shows that even in the absence of proper sort functions the category of extensional models of a simple specification can fail to have initial objects. This striking difference to Order-Sorted Logic [GM87a, SNGM89] stems from the fact that POS-homomorphisms are stronger than order-sorted homomorphisms, that is, every POS-homomorphism is an order-sorted homomorphism, but not vice versa.

Example 3.6.13 Let T be the simple specification

$$\mathsf{a}\colon \mathsf{A}, \quad \mathsf{A} \sqsubseteq \mathsf{B},$$

where a is a value constant and A and B are sort constants. Then

$$\mathsf{a}^{\mathcal{A}} = \mathsf{a}, \quad \mathsf{A}^{\mathcal{A}} = \{\mathsf{a}\}, \quad \mathsf{B}^{\mathcal{A}} = \{\mathsf{a}\}$$

and

$$\mathsf{a}^{\mathcal{B}} = \mathsf{a}, \quad \mathsf{A}^{\mathcal{B}} = \{\mathsf{a}\}, \quad \mathsf{B}^{\mathcal{B}} = \{\mathsf{a}, \mathsf{b}\}$$

define two extensional models of T. The initial model $\mathcal{I}(T)$ of T is quasi-extensional since T has just sort constants but no proper sort functions. It is easy to verify that $\mathcal{A}$ is the extensional algebra associated with $\mathcal{I}(T)$. Thus one could hope that $\mathcal{A}$ is an initial element in the category of the extensional models of T. However, this is not the case since there exists no POS-homomorphism $\mathcal{A} \to \mathcal{B}$. On the other hand, there exists an order-sorted homomorphism $\mathcal{A} \to \mathcal{B}$ and $\mathcal{A}$ is in fact an initial order-sorted model of T. □

Theorem 3.6.14 [Soundness and Completeness] *Let P be a prefix that contains all variables occurring in the value terms s and t. Then:*

1. $\sigma \sqsubseteq \tau$ *is valid in* $T \quad \Longleftrightarrow \quad \sigma \leq_T \tau$

2. *$P \to s{:}\sigma$ is valid in T $\iff$ $P \vdash_T s{:}\sigma$*

3. *$P \to s \doteq t$ is valid in T $\iff$ $s = t$ and s is well-typed in T under P.*

Proof. The soundness direction has been proven for the first and the second claim and is obvious for the third claim. Thus only the completeness direction remains to be shown. Let Q be a prefix such that $P \subseteq Q$ and Q contains all variables occurring in σ and τ. Furthermore, let ϵ be an $\mathcal{I}(T,Q)$-assignment such that ϵ maps every variable occurring in Q to itself. Then $\epsilon \in \mathcal{I}(T,Q)[\![Q]\!] \subseteq \mathcal{I}(T,Q)[\![P]\!]$.

1. Suppose $\sigma \sqsubseteq \tau$ is valid in T. Then $\sigma \sqsubseteq \tau$ is valid in $\mathcal{I}(T,Q)$. Hence $\sigma = \mathcal{I}(T,Q)[\![\sigma]\!]_\epsilon \leq^{\mathcal{I}(T,Q)} \mathcal{I}(T,Q)[\![\tau]\!]_\epsilon = \tau$ and thus $\sigma \leq_T \tau$.

2. Suppose $P \to s{:}\sigma$ is valid in T. Then $P \to s{:}\sigma$ is valid in $\mathcal{I}(T,Q)$. Hence $s = \mathcal{I}(T,Q)[\![s]\!]_\epsilon :^{\mathcal{I}(T,Q)} \mathcal{I}(T,Q)[\![\sigma]\!]_\epsilon = \sigma$ since $\epsilon \in \mathcal{I}(T,Q)[\![P]\!]$. Thus $Q \vdash_T s{:}\sigma$ and $P \vdash_T s{:}\sigma$ by Proposition 3.6.1.

3. Suppose $P \to s \doteq t$ is valid in T. Then $P \to s \doteq t$ is valid in $\mathcal{I}(T,Q)$. Hence $s = \mathcal{I}(T,Q)[\![s]\!]_\epsilon = \mathcal{I}(T,Q)[\![t]\!]_\epsilon = t$ since $\epsilon \in \mathcal{I}(T,Q)[\![P]\!]$. Furthermore, there exists a sort term σ such that $Q \vdash_T s{:}\sigma$. Thus $P \vdash_T s{:}\sigma$ by the freeness of the membership relation. □

Corollary 3.6.15 *Let P be a prefix containing all variables that occur in s. Then s is not well-typed in T under P if and only if there exist a model $\mathcal{A}$ of T and an $\mathcal{A}$-assignment $\delta \in \mathcal{A}[\![P]\!]$ such that $\mathcal{A}[\![s]\!]_\delta$ is not defined.*

Proof. From the Soundness and Completeness Theorem we know that $P \to s \doteq s$ is valid in T if and only if s is well-typed in T under P. The claim is obtained by negating both sides of this equivalence. □

Theorem 3.6.16 [Structural Induction] *Validity in the initial model $\mathcal{I}(T)$ of T can be characterized as follows:*

1. *$\sigma \sqsubseteq \tau$ is valid in $\mathcal{I}(T)$ if and only if $\theta\sigma \leq_T \theta\tau$ for every ground instance $\theta\sigma \sqsubseteq \theta\tau$ of $\sigma \sqsubseteq \tau$*

2. *if P contains every variable occurring in $s{:}\sigma$, then $P \to s{:}\sigma$ is valid in $\mathcal{I}(T)$ if and only if $\emptyset \vdash_T \theta s{:}\theta\sigma$ for every substitution θ such that $\emptyset \vdash_T \theta P$.*

Proof. *1.* Suppose $\sigma \sqsubseteq \tau$ is valid in $\mathcal{I}(T)$ and $\theta\sigma \sqsubseteq \theta\tau$ is a ground instance of $\sigma \sqsubseteq \tau$. Then we know by the Substitution Theorem that $\theta\sigma \sqsubseteq \theta\tau$ is valid in $\mathcal{I}(T)$. Hence $\theta\tau \leq^{\mathcal{I}(T)} \theta\sigma$ and thus $\theta\sigma \leq_T \theta\tau$.

Suppose $\theta\sigma \leq_T \theta\tau$ for every ground instance $\theta\sigma \sqsubseteq \theta\tau$ of $\sigma \sqsubseteq \tau$. Let δ be an $\mathcal{I}(T)$-assignment. Then $\theta_\delta\sigma \sqsubseteq \theta_\delta\tau$ is a ground instance of $\sigma \sqsubseteq \tau$ and hence $\theta_\delta\sigma \leq_T \theta_\delta\tau$. Thus $\delta \in \mathcal{I}(T)[\![\sigma \sqsubseteq \tau]\!]$ by Proposition 3.6.5.

2. Let P contain every variable occurring in $s{:}\sigma$, $P \to s{:}\sigma$ be valid in $\mathcal{I}(T)$, and let $\emptyset \vdash_T \theta P$. Then we know by the Soundness Proposition that θP is valid in $\mathcal{I}(T)$. Hence there exists an assignment $\delta \in \mathcal{I}(T)[\![P]\!]$ that agrees with θ on every variable in P. Since $P \to s{:}\sigma$

is valid in $\mathcal{I}(T)$, we have $\delta \in \mathcal{I}(T)[\![s\colon\sigma]\!]$. Hence $\emptyset \vdash_T \theta_\delta s\colon\theta_\delta\sigma$ and thus $\emptyset \vdash_T \theta s\colon\theta\sigma$ since θ and θ_δ agree on every variable occurring in $s\colon\sigma$.

Let $\emptyset \vdash_T \theta s\colon\theta\sigma$ for every substitution θ such that $\emptyset \vdash_T \theta P$, and let $\delta \in \mathcal{I}(T)[\![P]\!]$. Then $\emptyset \vdash_T \theta_\delta P$ and hence $\emptyset \vdash_T \theta_\delta s\colon\theta_\delta\sigma$. Thus $\delta \in \mathcal{I}(T)[\![s\colon\sigma]\!]$ by Proposition 3.6.5. $\square$

Chapter 4

Sort Rewriting Systems

4.1 Shallow Rewriting Systems
4.2 Upper Matchers and Suprema
4.3 Lower Matchers and Infima
4.4 Modes

In the last chapter we have seen that the sort inclusion order of a simple specification is just the rewriting relation of the rewrite system given by the inclusional axioms of the specification. This chapter is devoted to the study of rewrite systems generating well-behaved inclusion orders.

From the study of order-sorted unification [MGS89, SNGM89, Wal88] we know that there are two necessary conditions for the existence of principal unifiers, which is crucial for an efficient operational semantics:

1. *regularity*, that is, every value term must have a least sort
2. *lower completeness*, that is, every two sorts that have a common subsort must have a greatest common subsort.

If $\sigma \sqcap \tau$ is the greatest common subsort of σ and τ, then regularity is the necessary and sufficient condition to render a constraint

$$x : \sigma \,\&\, x : \tau$$

equivalent to

$$x : (\sigma \sqcap \tau)$$

in the initial algebra. The reduction of $x{:}\,\sigma \,\&\, x{:}\,\tau$ to $x{:}\,(\sigma \sqcap \tau)$ is the basic sort related operation of an order-sorted unification procedure.

Consider the following definition of lists:

$$\mathsf{nil}{:}\,\mathsf{list}(\alpha), \quad \mathsf{cons}{:}\,\alpha \times \mathsf{list}(\alpha) \to \mathsf{list}(\alpha).$$

Obviously, the empty list nil has no least sort and hence the specification is not regular. This problem can be solved as in Figure 1.1 by introducing a special sort for the empty list. However, a nicer way to get rid of this difficulty is to always have a unique empty sort $-$, which can be introduced by the inclusional axiom

$$- \sqsubseteq \alpha$$

and will in fact be empty in the initial model if $-$ occurs in no other axiom. With the axiom $- \sqsubseteq \alpha$ the least sort of nil is $\mathsf{list}(-)$.

Consider the polymorphic rank

$$f{:}\,\alpha \times \alpha \to \alpha.$$

Then a well-typed term $f(s,t)$ will only have a least sort if the least sorts of s and t have a least common supersort. Hence, in order to get regularity, we also need upper completeness, that is, every two sorts that have a common supersort must have a least common supersort. As in the order-sorted case, upper completeness will come for free, that is, the necessary conditions for lower completeness are already sufficient for upper completeness.

This shows that inclusion orders providing for a practical operational semantics must turn the set of sort terms into a quasi-lattice with a least element. We will see that the demand for greatest common subsorts necessarily requires that the inclusion order is well-founded, that is, that the generating rewrite system is terminating. Since we are aiming at a programming language, we can hence only admit relatively weak inclusional axioms for which it is decidable whether the rewrite system they define is terminating. Furthermore, to be practical, it is necessary that greatest common subsorts and least common supersorts are computable with reasonable resources.

By what I've said it is clear that the theory of sort rewriting systems is of central importance for the development of an operational semantics for relational programs computing over POS-types. Fortunately, there is a nice theory and there are good algorithms.

4.1 Shallow Rewriting Systems

For convenience we will use in this chapter our notation for value terms although in the rest of this thesis, of course, sort rewriting systems will rewrite sort terms to sort terms.

Let R be a rewrite system. Then we write $s \Rightarrow_R t$ if $s \to t$ is an instance of a rule of R. Furthermore, we write $f \Rightarrow_R g$ if there exist $\vec{s}$ and $\vec{t}$ such that $f(\vec{s}) \to g(\vec{t})$ is an instance of a rule of R.

In the following we assume that $-$ is a constant symbol.

A **shallow rewriting system** is a finite rewrite system R such that

1. R contains a rule $x \to -$, no other rule of R contains $-$, and every other rule of R has the form $f(\vec{x}) \to g(\vec{s})$, where $\vec{x}$ is a tuple of pairwise distinct variables
2. "$f \Rightarrow_R g$" is terminating
3. if u is the left hand side of a rule of R, then

$$u \Rightarrow_R^* g(\vec{s}) \ \wedge \ u \Rightarrow_R^* g(\vec{t}) \quad \Rightarrow \quad g(\vec{s}) = g(\vec{t}).$$

Proposition 4.1.1 *Every shallow rewriting system R allows for infinite ascending chains, for instance,*

$$\cdots \to_R f(f(f(-))) \to_R f(f(-)) \to_R f(-) \to_R -.$$

Example 4.1.2 Let R be the shallow rewriting system

$$\mathsf{x} \to -, \quad \mathsf{list}(\mathsf{x}) \to \mathsf{pair}(\mathsf{x}, \mathsf{list}(\mathsf{x})).$$

Then R allows for infinite descending chains, for instance,

$$\mathsf{list}(\mathsf{a}) \to_R \mathsf{pair}(\mathsf{a}, \mathsf{list}(\mathsf{a})) \to_R \mathsf{pair}(\mathsf{a}, \mathsf{pair}(\mathsf{a}, \mathsf{list}(\mathsf{a}))) \to_R \cdots.$$

□

Proposition 4.1.3 *It is decidable whether a finite rewrite system is shallow. Furthermore, if R is a shallow rewriting system, then "$s \Rightarrow_R t$" is terminating and it is decidable whether "$s \to_R t$" is terminating.*

Proof. Let R be a finite rewrite system. Then requirement (1) of the definition of shallow rewriting systems is certainly decidable. Now suppose R satisfies requirement (1). Then there are only finitely many pairs such that $f \Rightarrow_R g$ and $g \neq -$, and these pairs can be obtained directly from the rules of R. Now "$f \Rightarrow_R g$" is terminating if and only if the finite subrelation consisting of these pairs is terminating. Hence it is decidable whether "$f \Rightarrow_R g$" is terminating. Now let R satisfy requirements (1) and (2). Then the relation "$s \Rightarrow_R t$" is terminating and hence requirement (3) is decidable since R has only finitely many rules.

Now suppose R is a shallow rewriting system. Then a rule $f(\vec{x}) \to t \in R$ applies to every term $f(\vec{s})$ (this is the most important property of shallow rewriting systems). Hence R is terminating if and only if the finite relation

$$f \to g \ :\Longleftrightarrow \ \exists\, f(\vec{x}) \to t \in R.\ \ g \text{ occurs in } t$$

is terminating. Thus the termination of R is decidable. □

The following two propositions give you the main properties of shallow rewriting systems. There are many, and we will use them all in the rest of this thesis without explicitly referring their stating propositions.

Proposition 4.1.4 *Let R be a shallow rewriting system. Then:*

1. *"$f \Rightarrow_R^* g$" is a decidable well-founded order on the set of all function symbols whose least element is $-$*

2. *"$s \Rightarrow_R^* t$" is a decidable well-founded order on the set of all terms whose least element is $-$*

3. *if $s \Rightarrow_R^* t$, then $\theta s \Rightarrow_R^* \theta t$*

4. *if $s \Rightarrow_R^* t$ and $t \neq -$, then either s is a variable and $s = t$ or neither s nor t is a variable*

5. *if $\theta f(\vec{s}) \Rightarrow_R^* t$, then there exists a unique term u such that $f(\vec{s}) \Rightarrow_R^* u$ and $t = \theta u$*

6. *if $s \Rightarrow_R^* f(\vec{s})$ and $s \Rightarrow_R^* f(\vec{t})$, then $f(\vec{s}) = f(\vec{t})$*

7. *if $f(\vec{s})$ is a term and $f \Rightarrow_R^* g$, then there exists a unique term $g(\vec{t})$ such that $f(\vec{s}) \Rightarrow_R^* g(\vec{t})$*

8. *if $f(\vec{s}) \Rightarrow_R^* h(\vec{u})$ and $f \Rightarrow_R^* g \Rightarrow_R^* h$, then there exists a unique term $g(\vec{t})$ such that $f(\vec{s}) \Rightarrow_R^* g(\vec{t}) \Rightarrow_R^* h(\vec{u})$.*

Proof. *1.* Follows from the fact that "$f \Rightarrow_R^* g$" is reflexive and transitive and "$f \Rightarrow_R g$" is terminating.

2. Follows from the fact that "$s \Rightarrow_R^* t$" is reflexive and transitive and "$s \Rightarrow_R t$" is terminating.

3. and 4. Obvious.

5. Let $\theta f(\vec{s}) \Rightarrow_R^* t$. We show by induction on $f(\vec{s})$ with respect to the well-founded order "$s \Rightarrow_R^* s'$" that there exists a term u such that $f(\vec{s}) \Rightarrow_R^* u$ and $t = \theta u$. If $\theta f(\vec{s}) = t$, then the claim is trivial. Otherwise, we have $\theta f(\vec{s}) \Rightarrow_R g(\vec{u}) \Rightarrow_R^* t$. Hence we have $f(\vec{s}) \Rightarrow_R g(\vec{v})$ and $\theta g(\vec{v}) = g(\vec{u}) \Rightarrow_R^* t$. Thus we know by the induction hypothesis that there exists a term u such that $g(\vec{v}) \Rightarrow_R^* u$ and $t = \theta u$. Hence we have the claim since $f(\vec{s}) \Rightarrow_R g(\vec{v}) \Rightarrow_R^* u$.

To show that u is unique, suppose $f(\vec{s}) \Rightarrow_R^* u$, $f(\vec{s}) \Rightarrow_R^* v$ and $\theta u = \theta v = t$.

If $f(\vec{s}) = u$, then $v = f(\theta\vec{s})$. Since v is no variable, the top symbol of v must be f. Hence $v = f(\vec{s}) = u$ since otherwise $f \Rightarrow_R f$, which is impossible since "$g \Rightarrow_R h$" is terminating.

If $f(\vec{s}) \neq u$, then R has a rule whose left hand side is $f(\vec{x})$. Since $\{\vec{x}/\vec{s}\}f(\vec{s}) \Rightarrow_R^* u$ and $\{\vec{x}/\vec{s}\}f(\vec{s}) \Rightarrow_R^* v$ we know by the already proven existence claim that there exist u' and v' such that $f(\vec{x}) \Rightarrow_R^* u'$, $f(\vec{x}) \Rightarrow_R^* v'$, $u = \{\vec{x}/\vec{s}\}u'$ and $v = \{\vec{x}/\vec{s}\}v'$. Since neither t, u, v, u' nor v' is a variable, all five terms must have the same top symbol. Hence we know by requirement (3) of the definition of shallow rewriting systems that $u' = v'$. Thus $u = \{\vec{x}/\vec{s}\}u' = \{\vec{x}/\vec{s}\}v' = v$.

6. Let $s \Rightarrow_R^* f(\vec{s})$ and $s \Rightarrow_R^* f(\vec{t})$. If s is a variable, then $f = -$ and hence $f(\vec{s}) = f(\vec{t})$. If s is no variable, then the claim follows from statement (5) using $\theta = \emptyset$.

7. Let $f(\vec{s})$ be a term and $f \Rightarrow_R^* g$. Then one obtains by a straightforward induction on f with respect to the well-founded order "$h \Rightarrow_R^* h'$" that there exists a term $g(\vec{t})$ such that $f(\vec{s}) \Rightarrow_R^* g(\vec{t})$. The uniqueness follows by statement (6).

8. Follows immediately from statement (7). □

Lemma 4.1.5 *Let $s \to_R^* f(\vec{t})$. Then there exists a term $f(\vec{s})$ such that $s \Rightarrow_R^* f(\vec{s})$ and $\vec{s} \to_R^* \vec{t}$.*

Proof. We prove by induction on the length of a derivation $s \to_R^* f(\vec{t})$ that there exists a term $f(\vec{s})$ such that $s \Rightarrow_R^* f(\vec{s})$ and $\vec{s} \to_R^* \vec{t}$.

If $f = -$ or $s = f(\vec{t})$, then the claim is trivial.

Let $s \to_R^* g(\vec{u}) \to_R f(\vec{t})$. Then we know by the induction hypothesis that there exists a term $g(\vec{v})$ such that $s \Rightarrow_R^* g(\vec{v})$ and $\vec{v} \to_R^* \vec{u}$. If $g = f$, then we have the claim. Otherwise, we know that there exists a rule $g(\vec{x}) \to f(\vec{s}) \in R$ such that $\vec{t} = \{\vec{x}/\vec{u}\}\vec{s}$. This yields the claim since $s \Rightarrow_R^* g(\vec{v}) \Rightarrow_R^* \{\vec{x}/\vec{v}\}f(\vec{s})$ and $\{\vec{x}/\vec{v}\}\vec{s} \to_R^* \{\vec{x}/\vec{u}\}\vec{s} = \vec{t}$. □

Proposition 4.1.6 *Let R be a shallow rewriting system. Then:*

1. *(Uniqueness) if $s \to_R^* f(\vec{t})$, then there exists a unique term $f(\vec{s})$ such that $s \Rightarrow_R^* f(\vec{s})$*
2. *if $s \to_R^* f(\vec{t})$ and $s \Rightarrow_R^* f(\vec{s})$, then $\vec{s} \to_R^* \vec{t}$*
3. *(Orthogonality) if $s \to_R^* t$, $s \Rightarrow_R^* f(\vec{u})$ and $t \Rightarrow_R^* f(\vec{v})$, then $\vec{u} \to_R^* \vec{v}$*
4. $s \to_R^* f(\vec{t}) \iff \exists\, \vec{s}.\ s \Rightarrow_R^* f(\vec{s}) \wedge \vec{s} \to_R^* \vec{t}$
5. $s \to_R^* x \iff s = x$
6. *$s \to_R^* t$ is a decidable partial order on the set of all terms having $-$ as its least element*
7. *$\theta \to_R^* \psi$ is a partial order on the set of all substitutions having the substitution that maps every variable to $-$ as its least element.*

Proof. *1.* Let $s \to_R^* f(\vec{t})$. Then we know by the preceding lemma that there exists a term $f(\vec{s})$ such that $s \Rightarrow_R^* f(\vec{s})$. The uniqueness of $f(\vec{s})$ follows by statement (6) of the preceding proposition.

2. Follows by the preceding lemma and statement (1).

3. Let $s \to_R^* t$, $s \Rightarrow_R^* f(\vec{u})$ and $t \Rightarrow_R^* f(\vec{v})$. Then $s \to_R^* f(\vec{v})$ and $s \Rightarrow_R^* f(\vec{u})$. Hence we know by statement (2) that $\vec{u} \to_R^* \vec{v}$.

4. Follows from statements (1) and (2).

5. Obvious.

6. The decidability of $s \to_R^* t$ follows from statements (4) and (5). Furthermore, its obvious that $-$ is the least element.

Since "$s \to_R^* t$" is the reflexive and transitive closure of "$s \to_R t$", we know that "$s \to_R^* t$" is a quasi-order. To show that "$s \to_R^* t$" is antisymmetric, suppose $s \to_R^* t \to_R^* s$. We show by induction on s that $s = t$. If s is a variable, then we know that $s = t$. If $s = f(\vec{s})$, then there exists $\vec{t}$ such that $t = f(\vec{t})$ since "$f \Rightarrow_R^* g$" is a partial order. Hence $\vec{s} \to_R^* \vec{t} \to_R^* \vec{s}$ and thus $\vec{s} = \vec{t}$ by the induction hypothesis.

7. Follows immediately from statement (6). □

Let $(M, \leq)$ be a partially ordered set. Then c is called the **infimum** of a and b if $c \leq a, b$ and $d \leq c$ for every $d \leq a, b$. Furthermore, c is called the **supremum** of a and b if $a, b \leq c$ and $c \leq d$ for every $a, b \leq d$. It is easy to verify that infima and suprema are unique if they exist. Furthermore, the partial binary functions yielding infima and suprema are associative and commutative.

Proposition 4.1.7 *If R is a terminating shallow rewriting system such that the right hand side of every rule is linear, then there exists a natural number k such that $k|s| \geq |t|$ if $s \to_R^* t$.*

Proof. Let R be a terminating shallow rewriting system. Then there exists for every left hand side $f(\vec{x})$ of a rule of R a natural number k_f such that $|s| \leq k_f$ (recall, R has only finitely many rules). Then the greatest k_f is a constant as required. □

Recall that the purpose of a shallow rewriting system R is to define a partial order

$$s \leq_R t \;:\Longleftrightarrow\; t \to_R^* s$$

on the set of all terms. We denote the infimum [supremum] of s and t with respect to $\leq_R$ with $s \sqcap_R t$ [$s \sqcup_R t$] if it exists. The notations $f \sqcap_R g$, $f \sqcup_R g$, $\theta \sqcap_R \psi$, and $\theta \sqcup_R \psi$ are defined analogously.

A partially ordered set $(M, \leq)$ is a **lower quasi-lattice** if $a \sqcap b$ exists whenever a and b have a common lower bound. A partially ordered set $(M, \leq)$ is an **upper quasi-lattice** if $a \sqcup b$ exists whenever a and b have a common upper bound. A partially ordered set $(M, \leq)$ is a **quasi-lattice** if it is an upper and a lower quasi-lattice.

Proposition 4.1.8 *Let $(M, \leq)$ be a finite partially ordered set. Then $(M, \leq)$ is a lower quasi-lattice if and only if it is an upper quasi-lattice.*

Proof. Let $(M, \leq)$ be a lower quasi-lattice in which a and b have a common upper bound. We have to show that the supremum $a \sqcup b$ exists. Since $\{c \in M \mid a, b \leq c\}$ is nonempty and finite and $(M, \leq)$ is a lower quasi-lattice, we have $a \sqcup b = \sqcap\{c \in M \mid a, b \leq c\}$. The other direction is shown analogously. □

We call a shallow rewriting system R **complete** if the set of function symbols is a quasi-lattice under the partial order "$f \Rightarrow_R^* g$".

In Section 4.2 we will show that the partial order defined by a complete shallow rewriting system on the set of all terms is a upper quasi-lattice. The following example shows that completeness does not suffice to obtain a lower quasi-lattice.

Example 4.1.9 Let R be the complete and nonterminating shallow rewriting system consisting of the rules

$$x \to -, \quad a \to f(a), \quad b \to f(b).$$

Then $a \sqcap_R b$ does not exist since

$$a, b \to_R^* \cdots \to_R^* f(f(f(-))) \to_R^* f(f(-)) \to_R^* f(-).$$

□

In Section 4.3 we will show that a terminating and complete shallow rewriting system defines a quasi-lattice on the set of all terms.

We call a terminating and complete shallow rewriting system a **sort rewriting system**. The POS-type specifications to be defined in the next chapter will only admit inclusional axioms that form a sort rewriting system.

In Section 4.2 we will show that one can decide whether an inclusion system $\vec{s} \sqsupseteq \vec{t}$ has an *upper matcher*, that is, whether there exists a substitution θ such that $\theta\vec{s} \to_R^* \vec{t}$, provided R is a complete shallow rewriting system. In Section 4.3 we will show that one can decide whether an inclusion system $\vec{s} \sqsupseteq \vec{t}$ has a *lower matcher*, that is, whether there exists a substitution θ such that $\vec{s} \to_R^* \theta\vec{t}$, provided R is a sort rewriting system.

The natural generalization of these problems is the problem to decide whether an inclusion system $\vec{s} \sqsupseteq \vec{t}$ is **satisfiable**, that is, whether there exists a substitution θ such that $\theta\vec{s} \to_R^* \theta\vec{t}$. I don't know whether this problem is decidable for sort rewriting systems. Fortunately, there exist good type checking and good constraint solving algorithms that require only the computation of lower and upper matchers but not the computation of satisfying substitutions. However, a perfect type inference algorithm would require the computation of satisfying substitutions. Fortunately, the imperfect type inference algorithm given in Chapter 7, which relies only on the computation of matchers, does work quite well for practical applications.

4.2 Upper Matchers and Suprema

In this section we assume that R is a complete shallow rewriting system.

An **inclusion system** is a possibly empty conjunction of inclusions. As usual we identify an inclusion system with the multiset of its inclusions. If convenient, we will use the vector notation $\vec{s} \sqsubseteq \vec{t}$ for inclusion systems. Furthermore, the letter I will always denote an inclusion system.

An **upper matcher** of an inclusion system $\vec{s} \sqsupseteq \vec{t}$ (in R) is a substitution θ such that $\theta\vec{s} \to_R^* \vec{t}$. We use $\mathrm{UM}_R[I]$ to denote the set of all upper matchers of the inclusion system I in R. Note that the upper matchers of an inclusion system are partially ordered by "$\theta \to_R^* \psi$".

We call an inclusion system **upwards solved** if it has the form $\vec{x} \sqsupseteq \vec{t}$, where $\vec{x}$ is a possibly empty tuple of pairwise distinct variables and no component of the tuple $\vec{t}$ is $-$.

Proposition 4.2.1 *Let $I = (x_1 \sqsupseteq t_1 \,\&\, \ldots \,\&\, x_n \sqsupseteq t_n)$ be an upwards solved inclusion system. Then*

$$\theta x := \begin{cases} t_i & \text{if } x = x_i \text{ for some } i \in 1..n \\ - & \text{otherwise} \end{cases}$$

is the least upper matcher of I in every shallow rewriting system R.

We will show that the following reduction rules for inclusion systems constitute an algorithm that, given an inclusion system I, decides whether I has an upper matcher and, if I has an upper matcher, computes the least upper matcher of I.

1. $x \sqsupseteq y \,\&\, x \sqsupseteq y \,\&\, I \quad \xrightarrow{\mathrm{u}}_R \quad x \sqsupseteq y \,\&\, I$
2. $s \sqsupseteq - \,\&\, I \quad \xrightarrow{\mathrm{u}}_R \quad I$
3. $f(\vec{s}) \sqsupseteq g(\vec{t}) \,\&\, I \quad \xrightarrow{\mathrm{u}}_R \quad \vec{u} \sqsupseteq \vec{t} \,\&\, I \quad$ if $f(\vec{s}) \Rightarrow_R^* g(\vec{u})$
4. $x \sqsupseteq s \,\&\, x \sqsupseteq t \,\&\, I \quad \xrightarrow{\mathrm{u}}_R \quad x \sqsupseteq f(u_1, \ldots, u_n) \,\&\, I$
5. if neither s nor t is a variable or $-$,
   $f = \mathrm{TOPSYM}[s] \sqcup_R \mathrm{TOPSYM}[t]$,
   $x_1, \ldots, x_n$ are pairwise distinct variables,
   $f(x_1, \ldots, x_n) \sqsupseteq s \,\&\, f(x_1, \ldots, x_n) \sqsupseteq t \xrightarrow{\mathrm{u}}_R^* I'$ and I' is upwards solved,
   $u_i = \begin{cases} u & \text{if } (x_i \sqsupseteq u) \in I' \\ - & \text{otherwise} \end{cases}$ for $i \in 1..n$.

Example 4.2.2 Let R be the shallow rewriting system consisting of the rules

$$x \to -, \quad f(x) \to a, \quad f(x) \to b.$$

Then

$$x \sqsupseteq a \,\&\, x \sqsupseteq b \xrightarrow{\mathrm{u}}_R x \sqsupseteq f(-)$$

by rule (4) since

$$f(y) \sqsupseteq a \,\&\, f(y) \sqsupseteq b \xrightarrow{\mathrm{u}}_R f(y) \sqsupseteq b \xrightarrow{\mathrm{u}}_R \emptyset$$

by applying rule (3) twice. Note that $f(-)$ is the supremum of a and b.

Let $\vec{s} \sqsupseteq \vec{t}$ be an inclusion system. Then the **lower variables** of I are $\mathcal{LV}[\vec{s} \sqsupseteq \vec{t}] := \mathcal{V}\vec{t}$ and the **upper variables** of I are $\mathcal{UV}[\vec{s} \sqsupseteq \vec{t}] := \mathcal{V}\vec{s}$. □

Proposition 4.2.3 *If $I \xrightarrow{\mathrm{u}}_R^* I'$, then $\mathcal{LV}[I'] = \mathcal{LV}[I]$ and $\mathcal{UV}[I'] \subseteq \mathcal{UV}[I]$.*

Proposition 4.2.4 *Let V be a set of variables and Σ be a signature such that every rule of R consists of Σ-terms. Then I' is a (Σ, V)-inclusion system if I is a (Σ, V)-inclusion system and $I \xrightarrow{\mathrm{u}}_R^* I'$.*

The **U-complexity** of inclusion systems is defined as follows:

1. $|x| := 1$ and $|f(s_1, \ldots, s_n)| := \begin{cases} 0 & \text{if } f = - \\ 1 + \sum_{i=1}^{n} |s_i| & \text{otherwise} \end{cases}$
2. $|s \sqsupseteq t| := \begin{cases} 3|t| & \text{if } t = - \text{ or } s \text{ is a variable} \\ 3|t| - 2 & \text{otherwise} \end{cases}$
3. $|I| := (\sum_{(s \sqsupseteq t) \in I} |s \sqsupseteq t|,\ n)$, where n is the number of inclusions in I.

The set of all U-complexities is a well-founded partially ordered set under the lexicographic order induced by the canonical order on the natural numbers.

Lemma 4.2.5 [Termination] *If* $I \xrightarrow{\mathrm{u}}{}^*_R I'$, *then* $|I| \geq |I'|$, *and if* $I \xrightarrow{\mathrm{u}}_R I'$, *then* $|I| > |I'|$.

Proof. For the first three rules the second claim is easy to verify. The fourth rule is recursive and thus forces us to first show the first claim by induction on the U-complexity of the reduced inclusion system. Since $|x \sqsupseteq s| + |x \sqsupseteq t| = 3|s| + 3|t| > 3|s| + 3|t| - 4 = |f(x_1, \ldots, x_n) \sqsupseteq s| + |f(x_1, \ldots, x_n) \sqsupseteq t|$, we know by the induction hypothesis that the first component $|I'|_1$ of the U-complexity of I' satisfies $|I'|_1 \leq 3|s| + 3|t| - 4$. Since I' is upwards solved and $|-| = 0$, we have $3\sum_{i=1}^n |u_i| = |I'|_1 \leq 3|s| + 3|t| - 4$. Hence $|x \sqsupseteq f(u_1, \ldots, u_n)| = 3(1 + \sum_{i=1}^n |u_i|) \leq 3|s| + 3|t| - 1 < |x \sqsupseteq s| + |x \sqsupseteq t|$. □

Lemma 4.2.6 [Invariance] *If* $I \xrightarrow{\mathrm{u}}{}^*_R I'$, *then* $\mathrm{UM}_R[I] = \mathrm{UM}_R[I']$.

Proof. Since rule (4) is recursive, we prove the claim by induction on the U-complexity of I. If $|I| = (0, 0)$ or $I = I'$, then the claim is trivial. Otherwise, it suffices to show that the first reduction step leaves the upper matchers invariant. Rules (1) and (2) obviously leave the upper matchers invariant.

1. To show that rule (3) leaves the upper matchers invariant, suppose that

$$t \sqsupseteq f(s_1, \ldots, s_n) \,\&\, I \quad \xrightarrow{\mathrm{u}}_R \quad u_1 \sqsupseteq s_1 \,\&\, \ldots \,\&\, u_n \sqsupseteq s_n \,\&\, I$$

and $t \Rightarrow^*_R f(u_1, \ldots, u_n)$.

Let θ be an upper matcher of the left-hand side. Then $\theta t \rightarrow^*_R f(s_1, \ldots, s_n)$ and $\theta t \Rightarrow^*_R f(\theta u_1, \ldots, \theta u_n)$. Hence we know that $\theta u_i \rightarrow^*_R s_i$ for $i = 1, \ldots, n$. Thus θ is an upper matcher of the right-hand side.

Let θ be an upper matcher of the right-hand side. Then $\theta u_i \rightarrow^*_R s_i$ for $i = 1, \ldots, n$. Hence $\theta t \Rightarrow^*_R f(\theta u_1, \ldots, \theta u_n) \rightarrow^*_R f(s_1, \ldots, s_n)$. Thus θ is an upper matcher of the left-hand side.

2. To show that rule (4) leaves the upper matchers invariant, suppose that

$$x \sqsupseteq s \,\&\, x \sqsupseteq t \,\&\, I \quad \xrightarrow{\mathrm{u}}_R \quad x \sqsupseteq f(u_1, \ldots, u_n) \,\&\, I,$$

$f = \textsc{topsym}[s] \sqcup_R \textsc{topsym}[t]$, $x_1, \ldots, x_n$ are pairwise distinct variables,

$$f(x_1, \ldots, x_n) \sqsupseteq s \,\&\, f(x_1, \ldots, x_n) \sqsupseteq t \;\xrightarrow{\mathrm{u}}{}^*_R\; I',$$

I' is upwards solved, and $u_i = u$ if $(x_i \sqsupseteq u) \in I'$ and $u_i = -$ otherwise.

Let θ be an upper matcher of the left-hand side. Then $\theta x \rightarrow^*_R s$ and $\theta x \rightarrow^*_R t$. Hence we know that there exist terms $s_1, \ldots, s_n$ such that $\theta x \Rightarrow^*_R f(s_1, \ldots, s_n) \rightarrow^*_R s, t$. Thus $\psi := \{x_1/s_1, \ldots, x_n/s_n\}$ is an upper matcher of $f(x_1, \ldots, x_n) \sqsupseteq s \,\&\, f(x_1, \ldots, x_n) \sqsupseteq t$. Now we know by the induction hypothesis that ψ is an upper matcher of I'. Hence we have $\psi x_i \rightarrow^*_R u_i$ for $i = 1, \ldots, n$. Thus $\theta x \Rightarrow^*_R f(s_1, \ldots, s_n) = \psi f(x_1, \ldots, x_n) \rightarrow^*_R f(u_1, \ldots, u_n)$. Hence θ is an upper matcher of the right-hand side.

Let θ be an upper matcher of the right-hand side. Then $\theta x \rightarrow^*_R f(u_1, \ldots, u_n)$. Hence there exist terms $s_1, \ldots, s_n$ such that $\theta x \Rightarrow^*_R f(s_1, \ldots, s_n)$ and $s_i \rightarrow^*_R u_i$ for $i = 1, \ldots, n$.

Since I' is upwards solved and its right-hand sides can only be the variables $x_1, \ldots, x_n$, we know that $\psi := \{x_1/s_1, \ldots, x_n/s_n\}$ is an upper matcher of I'. Hence we know by the induction hypothesis that ψ is an upper matcher of $f(x_1, \ldots, x_n) \sqsupseteq s \,\&\, f(x_1, \ldots, x_n) \sqsupseteq t$. Thus $\theta x \Rightarrow_R^* f(s_1, \ldots, s_n) = \psi f(x_1, \ldots, x_n) \rightarrow_R^* s, t$. Hence ψ is an upper matcher of the left-hand side. □

Lemma 4.2.7 [Completeness] *Let I be an inclusion system that has an upper matcher. Then there exists an upwards solved inclusion system I' such that $I \xrightarrow{\mathrm{u}}_R^* I'$.*

Proof. We prove the claim by induction on the U-complexity of I. If $|I| = (0,0)$ or I is upwards solved, the claim is trivial. Otherwise, since we know by the preceding lemma that reduction with $\xrightarrow{\mathrm{u}}_R$ maintains the existence of upper matchers, it suffices to show that at least one rule applies to I. Since I has an upper matcher, at least one of the following cases applies.

1. I contains an inclusion $x \sqsupseteq y$ twice. Then rule (1) applies.

2. I contains an inclusion $s \sqsupseteq -$. Then rule (2) applies.

3. I contains an inclusion $f(s_1, \ldots, s_m) \sqsupseteq g(t_1, \ldots, t_n)$. Since I has an upper matcher, we have $\theta f(s_1, \ldots, s_m) \rightarrow_R^* g(t_1, \ldots, t_n)$. Hence there exist terms $u_1, \ldots, u_n$ such that $f(s_1, \ldots, s_m) \Rightarrow_R^* g(u_1, \ldots, u_n)$. Thus rule (3) applies.

4. I contains two inclusions $x \sqsupseteq s$ and $x \sqsupseteq t$ such that neither s nor t is a variable or $-$. Since I has an upper matcher, we have $\theta x \rightarrow_R^* s, t$. Hence $\textsc{topsym}[s]$ and $\textsc{topsym}[t]$ have a common upper bound. Since R is complete, $f := \textsc{topsym}[s] \sqcup_R \textsc{topsym}[t]$ exists. Hence there exist terms $u_1, \ldots, u_n$ such that $\theta x \rightarrow_R^* f(u_1, \ldots, u_n) \rightarrow_R^* s, t$. Thus $f(x_1, \ldots, x_n) \sqsupseteq s \,\&\, f(x_1, \ldots, x_n) \sqsupseteq t$ has an upper matcher if $x_1, \ldots, x_n$ are pairwise distinct variables. Now we know by the induction hypothesis that there exists an upwards solved inclusion system I' such that

$$f(x_1, \ldots, x_n) \sqsupseteq s \,\&\, f(x_1, \ldots, x_n) \sqsupseteq t \quad \xrightarrow{\mathrm{u}}_R^* \quad I'.$$

Hence rule (4) is applicable. □

Theorem 4.2.8 [Upper Matching] *Let R be an complete shallow rewriting system. Then it is decidable whether an inclusion system has an upper matcher. Furthermore, if I has an upper matcher, one can compute an upwards solved inclusion system I' such that $\mathrm{UM}_R[I] = \mathrm{UM}_R[I']$.*

Proof. The claims follow immediately from the preceding lemmas. □

Corollary 4.2.9 [Least Upper Matchers] *Let R be a complete shallow rewriting system and I be an inclusion system that has an upper matcher. Then there exists one and only one upwards solved inclusion system*

$$x_1 \sqsupseteq s_1 \,\&\, \ldots \,\&\, x_n \sqsupseteq s_n$$

having the same upper matchers as I. Furthermore,

$$\theta x := \begin{cases} s_i & \text{if } x = x_i \text{ for some } i \in 1..n \\ - & \text{otherwise} \end{cases}$$

is the least upper matcher of I.

Proof. The existence of the upwards solved inclusion system follows from the preceding theorem. Thus θ is the least upper matcher of I. Since the least upper matcher of I is unique and the upwards solved system does not contain inclusions $x_i \sqsupseteq -$ (by definition), the upwards solved system must be unique. □

Corollary 4.2.10 [Suprema] *Let R be an complete shallow rewriting system. Then every two terms that have a common upper bound have a supremum and, if $s \neq -$,*

$$s \sqcup_R t = u \quad \Longleftrightarrow \quad x \sqsupseteq s \,\&\, x \sqsupseteq t \ \xrightarrow{\mathrm{u}}{}^*_R \ x \sqsupseteq u.$$

4.3 Lower Matchers and Infima

In this section we assume that R is a sort rewriting system (that is, a terminating and complete shallow rewriting system).

Lemma 4.3.1 *If $\theta \sqcap_R \psi$ exists, then $\theta s \sqcap_R \psi s = (\theta \sqcap_R \psi)s$ for every term s.*

Proof. Let s be a term and let $\theta \sqcap_R \psi$ exist. Since $\theta, \psi \to^*_R \theta \sqcap_R \psi$, we have $\theta s, \psi s \to^*_R (\theta \sqcap_R \psi)s$. Now let t be a term such that $\theta s, \psi s \to^*_R t$. We show by induction on t that $(\theta \sqcap_R \psi)s \to^*_R t$. If t is a variable, then $t = \theta s = \psi s = (\theta \sqcap_R \psi)s$. If $t = g(\vec{t})$, we distinguish two cases.

1. s is a variable. Then $\theta s \sqcap_R \psi s$ exists and $\theta s \sqcap_R \psi s = (\theta \sqcap_R \psi)s$. Since we assumed $\theta s, \psi s \to^*_R t$, we have the claim.

2. $s = f(\vec{s})$. Since we assumed $\theta s \to^*_R g(\vec{t}) = t$, we know that there exists $\vec{u}$ such that $s \Rightarrow^*_R g(\vec{u})$. Hence $\theta s \Rightarrow^*_R \theta g(\vec{u})$ and $\psi s \Rightarrow^*_R \psi g(\vec{u})$. Since we assumed $\theta s, \psi s \to^*_R g(\vec{t})$, we have $\theta \vec{u}, \psi \vec{u} \to^*_R \vec{t}$. Hence we know by the induction hypothesis that $(\theta \sqcap_R \psi)\vec{u} \to^*_R \vec{t}$. Thus $(\theta \sqcap_R \psi)s \Rightarrow^*_R (\theta \sqcap_R \psi)g(\vec{u}) \to^*_R g(\vec{t}) = t$. □

Lemma 4.3.2 *Let $f(\vec{s}) \Rightarrow^*_R (f \sqcap_R g)(\vec{u})$, $g(\vec{t}) \Rightarrow^*_R (f \sqcap_R g)(\vec{v})$, and let $\vec{u} \sqcap_R \vec{v}$ exist. Then $f(\vec{s}) \sqcap_R g(\vec{t}) = (f \sqcap_R g)(\vec{u} \sqcap_R \vec{v})$.*

Proof. Let $f(\vec{s}) \to^*_R u$ and $g(\vec{t}) \to^*_R u$. We have to show that $(f \sqcap_R g)(\vec{u} \sqcap_R \vec{v}) \to^*_R u$. Since u cannot be a variable, we know that $(f \sqcap_R g) \Rightarrow^*_R \textsc{topsym}[u]$. Hence $f(\vec{s}) \Rightarrow^*_R (f \sqcap_R g)(\vec{u}) \to^*_R u$ and $g(\vec{t}) \Rightarrow^*_R (f \sqcap_R g)(\vec{v}) \to^*_R u$. By the preceding lemma we know that

$$\begin{aligned}(f \sqcap_R g)(\vec{u}) \sqcap_R (f \sqcap_R g)(\vec{v}) &= (\{\vec{x}/\vec{u}\}(f \sqcap_R g)(\vec{x})) \sqcap_R (\{\vec{x}/\vec{v}\}(f \sqcap_R g)(\vec{x})) \\ &= \{\vec{x}/(\vec{u} \sqcap_R \vec{v})\}(f \sqcap_R g)(\vec{x}) = (f \sqcap_R g)(\vec{u} \sqcap_R \vec{v}),\end{aligned}$$

for some tuple $\vec{x}$ of pairwise distinct variables. Hence $(f \sqcap_R g)(\vec{u} \sqcap_R \vec{v}) \to^*_R u$. □

The R-**complexity** $\|s\|_R$ of a term s is defined as the pair

$$\|s\|_R := (k, l),$$

where k is the maximal length of a chain $s \to_R s_1 \to_R s_2 \to_R \cdots$ and l is the number of function symbol occurrences in s. The set of all R-complexities is a well-founded partially ordered set under the lexicographic order induced by the canonical order on the natural numbers.

Theorem 4.3.3 (Infima) *Let R be a sort rewriting system. Then $s \sqcap_R t$ exists for every two terms s and t and can be computed as follows:*

1. $x \sqcap_R x = x$
2. $x \sqcap_R s = -$ *if* $x \neq s$
3. $s \sqcap_R x = -$ *if* $x \neq s$
4. $f(\vec{s}) \sqcap_R g(\vec{t}) = (f \sqcap_R g)(\vec{u} \sqcap_R \vec{v})$
5. *if* $f(\vec{s}) \Rightarrow_R^* (f \sqcap_R g)(\vec{u})$ *and* $g(\vec{t}) \Rightarrow_R^* (f \sqcap_R g)(\vec{v})$.

If the right-hand side of every rule of R is linear, then the infimum two terms can be computed in linear time.

Proof. The first three claims are obvious. The last claim holds by the preceding lemma, provided we can show that $s \sqcap_R t$ exists for every two terms s and t. We show this by induction on $\|s\|_R$. If s or t is a variable, then $s \sqcap_R t$ exists by one of the first three equations. If $s = f(\vec{s})$ and $t = g(\vec{t})$, then there exist terms $\vec{u}$ and $\vec{v}$ such that $s \Rightarrow_R^* (f \sqcap_R g)(\vec{u})$ and $t \Rightarrow_R^* (f \sqcap_R g)(\vec{v})$. Since $\|s\|_R > \|u_i\|_R$ for every component u_i of $\vec{u}$, we know by the induction hypothesis that $\vec{u} \sqcap_R \vec{v}$ exists. Hence we know by the preceding lemma that $s \sqcap_R t$ exists.

The linear time complexity follows by Proposition 4.1.7. □

Corollary 4.3.4 *Let R be a sort rewriting system. Then the set of all terms ordered by "$s \to_R^* t$" is a well-founded quasi-lattice having $-$ as its least element.*

Proof. Follows immediately from the Infima Theorem and the Suprema Corollary of the last section. □

Example 4.3.5 Let R be the sort rewriting system

$$x \to -, \quad f(x) \to a, \quad f(x) \to b.$$

Then $f(c) \sqcap_R f(d) = f(-)$ and a and b are common lower bounds of $f(c)$ and $f(d)$. Note that without the empty sort rule $x \to -$ the terms $f(c)$ and $f(d)$ would not have an infimum although they still would have a and b as common lower bounds. □

Example 4.3.6 In general, infima in a sort rewriting system R are not stable under instantiation, that is, $\theta s \sqcap_R \theta t = \theta(s \sqcap_R t)$ does not hold. To see this, let R be the trivial sort rewriting system just consisting of the rule $x \to -$, x and y be two distinct variables, and $\theta = \{x/a, y/a\}$, where a is a constant symbol different from $-$. Then $\theta(x \sqcap_R y) = -$ and $\theta x \sqcap_R \theta y = a$. □

We call the infimum of s and t **stable** if $\theta(s \sqcap_R t) = \theta s \sqcap_R \theta t$ for every substitution θ. Furthermore, the **stable infimum function** is the partial function defined by

$$s \sqcap_R^\circ t := s \sqcap_R t \qquad \text{if the infimum of } s \text{ and } t \text{ is stable.}$$

Theorem 4.3.7 [Stable Infima] *Let R be a sort rewriting system. Then $t_1 \sqcap_R t_2$ is stable if t_1 and t_2 have a common upper bound, that is, there exists a term s such that $s \to_R^* t_1$ and $s \to_R^* t_2$.*

Proof. Suppose $s \to_R^* t_1$ and $s \to_R^* t_2$. We prove by induction on $\|s\|_R$ that $t_1 \sqcap_R t_2$ is stable. If $s = x$, then either $t_1 = t_2 = x$, or $t_1 = t_2 = -$, or $t_1 = x$ and $t_2 = -$, or $t_1 = -$ and $t_2 = x$. For all these cases $t_1 \sqcap_R t_2$ is stable. Otherwise, let $s = f(\vec{s})$. Furthermore, let $t_1 = g_1(\vec{t_1})$, $t_2 = g_2(\vec{t_2})$, $h = g_1 \sqcap_R g_2$, $f(\vec{s}) \Rightarrow_R^* h(\vec{u})$, $g_1(\vec{t_1}) \Rightarrow_R^* h(\vec{u}_1)$, and $g_2(\vec{t_2}) \Rightarrow_R^* h(\vec{u}_2)$. Then we know that $\vec{u} \to_R^* \vec{u}_1$ and $\vec{u} \to_R^* \vec{u}_2$. Hence we know by the induction hypothesis that $u_1 \sqcap_R u_2$ is stable. Hence we have

$$\theta(t_1 \sqcap_R t_2) = \theta h(\vec{u}_1 \sqcap_R \vec{u}_2) = h(\theta(\vec{u}_1 \sqcap_R \vec{u}_2)) = h(\theta\vec{u}_1 \sqcap_R \theta\vec{u}_2) = \theta t_1 \sqcap_R \theta t_2$$

by the Infima Theorem since $\theta t_1 = \theta g_1(\vec{t_1}) \Rightarrow_R^* h(\theta\vec{u}_1)$ and $\theta t_2 = \theta g_2(\vec{t_2}) \Rightarrow_R^* h(\theta\vec{u}_2)$. Thus $t_1 \sqcap_R t_2$ is stable. □

A **lower matcher** of an inclusion system $\vec{s} \sqsupseteq \vec{t}$ (in R) is a substitution θ such that $\vec{s} \to_R^* \theta\vec{t}$ and $\mathcal{D}\theta \subseteq \mathcal{V}\vec{t}$. We use $\mathrm{LM}_R[I]$ to denote the set of all lower matchers of an inclusion system I in R. Note that the lower matchers of an inclusion system are partially ordered by "$\theta \to_R^* \psi$".

We call an inclusion system **downwards solved** if it has the form $\vec{s} \sqsupseteq \vec{x}$, where $\vec{x}$ is a possibly empty tuple pairwise distinct variables.

Proposition 4.3.8 *Let $I = (s_1 \sqsupseteq x_1 \,\&\, \ldots \,\&\, s_n \sqsupseteq x_n)$ be a downwards solved inclusion system. Then*

$$\theta x := \begin{cases} s_i & \text{if } x = x_i \text{ for some } i \in 1..n \\ x & \text{otherwise} \end{cases}$$

is the greatest lower matcher of I in every shallow rewriting system R.

We will show that the following reduction rules for inclusion systems constitute an algorithm that, given an inclusion system I, decides whether I has a lower matcher and, if I has a lower matcher, computes the greatest lower matcher of I.

1. $s \sqsupseteq - \,\&\, I \quad \xrightarrow{1}_R \quad I$
2. $f(\vec{s}) \sqsupseteq g(\vec{t}) \,\&\, I \quad \xrightarrow{1}_R \quad \vec{u} \sqsupseteq \vec{t} \,\&\, I \qquad \text{if } f(\vec{s}) \Rightarrow_R^* g(\vec{u})$
3. $s \sqsupseteq x \,\&\, t \sqsupseteq x \,\&\, I \quad \xrightarrow{1}_R \quad s \sqcap_R t \sqsupseteq x \,\&\, I.$

Proposition 4.3.9 *If $I \xrightarrow{1}_R^* I'$, then $\mathcal{LV}[I'] = \mathcal{LV}[I]$ and $\mathcal{UV}[I'] \subseteq \mathcal{UV}[I]$.*

Proposition 4.3.10 *Let V be a set of variables and Σ be a signature such that every rule of R consists of Σ-terms. Then I' is a (Σ, V)-inclusion system if I is a (Σ, V)-inclusion system and $I \xrightarrow{1}_R^* I'$.*

Proposition 4.3.11 [Termination] *There are no infinite chains $I \xrightarrow{1}_R I_1 \xrightarrow{1}_R I_2 \xrightarrow{1}_R \cdots$ issuing from an inclusion system I.*

Proof. Every $\xrightarrow{\downarrow}_R$-step reduces the sum of the sizes of the right hand sides. □

Lemma 4.3.12 [Invariance] *If $I \xrightarrow{\downarrow}_R I'$, then $\mathrm{LM}_R[I] = \mathrm{LM}_R[I']$.*

Proof. The claim is obvious for the first rule. The second rule leaves the lower matchers invariant since

$$\theta f(\vec{s}) \to_R^* g(\vec{t}) \iff \theta g(\vec{u}) \to_R^* g(\vec{t}) \iff \theta \vec{u} \to_R^* \vec{t}$$

if $f(\vec{s}) \Rightarrow_R^* g(\vec{u})$. The third rule leaves the lower matchers invariant since

$$s, t \to_R^* \theta x \iff s \sqcap_R t \to_R^* \theta x.$$

□

Lemma 4.3.13 [Completeness] *Let I be an inclusion system that has a lower matcher and is not downwards solved. Then there exists an inclusion system I' such that $I \xrightarrow{\downarrow}_R I'$.*

Proof. Since I has a lower matcher, at least one of the following cases applies.

1. I contains an inclusion $s \sqsupseteq -$. Then rule (1) applies.

2. I contains an inclusion $f(\vec{s}) \sqsupseteq g(\vec{t})$. Since I has a lower matcher, we have $f(\vec{s}) \to_R^* \theta g(\vec{t})$. Hence there exists $\vec{u}$ such that $f(\vec{s}) \Rightarrow_R^* g(\vec{u})$. Thus rule (2) applies.

3. I contains two inclusions $s \sqsupseteq x$ and $t \sqsupseteq x$. Since R is sort rewriting system, we know that $s \sqcap_R t$ exists. Hence rule (3) is applicable. □

Theorem 4.3.14 [Lower Matching] *Let R be a sort rewriting system. Then it is decidable whether an inclusion system has a lower matcher. Furthermore, if an inclusion system I has a lower matcher, one can compute a downwards solved inclusion system I' such that $\mathrm{LM}_R[I] = \mathrm{LM}_R[I']$.*

Proof. The claims follow immediately from the preceding lemmas. □

Corollary 4.3.15 [Greatest Lower Matcher] *Let R be a sort rewriting system and I be an inclusion system that has a lower matcher. Then there exists one and only one downwards solved inclusion system*

$$s_1 \sqsupseteq x_1 \,\&\, \ldots \,\&\, s_n \sqsupseteq x_n$$

having the same lower matchers as I. Furthermore,

$$\theta x := \begin{cases} s_i & \text{if } x = x_i \text{ for some } i \in 1..n \\ x & \text{otherwise} \end{cases}$$

is the greatest lower matcher of I.

Proof. The existence of the downwards solved inclusion system follows from the preceding theorem. Thus θ is the greatest lower matcher of I. Since the greatest lower matcher of I is unique and $\mathcal{D}\theta \subseteq \mathcal{LV}[I]$ (by definition), the downwards solved system must be unique. □

4.4 Modes

This section prepares material for Section 5.2 on inhabitation and I recommend to read it together with Section 5.2.

We now generalize shallow term rewriting to shallow rewriting of term tuples. A **tuple rewriting rule** is a pair $s \to \vec{s}$, where s is a term, $\vec{s}$ is a possibly empty tuple of terms, and every variable occurring in the right-hand side $\vec{s}$ occurs in the left-hand side s. We use $\emptyset$ to denote the empty tuple. A **tuple rewrite system** is a set of tuple rewriting rules. A tuple rewrite system R defines a binary relation "$\vec{s} \Rightarrow_R \vec{t}$" on the set of all term tuples as follows:

1. $(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n) \Rightarrow_R (s_1, \ldots, s_{i-1}, t_1, \ldots, t_m, s_{i+1}, \ldots, s_n)$
2. if and only if $s_i \to (t_1, \ldots, t_m)$ is an instance of a rule of R.

A **shallow tuple rewrite system** is a tuple rewrite system R such that every rule of R has the form $f(\vec{x}) \to \vec{s}$, where $\vec{x}$ is a tuple of pairwise distinct variables. (These tuple rewrite systems are called shallow because of the form of the left hand sides of their rules. Unfortunately, I abused the name shallow in the definition of shallow rewriting systems in Section 4.1 for systems that have to satisfy additional properties and also contain the nonshallow rule $x \to -$ since I couldn't come up with another nice name. I do apologize.)

In this section we assume that R is a shallow tuple rewrite system.

A term s is called **weak in** R if there exists a tuple $\vec{x}$ of variables such that $s \Rightarrow_R^* \vec{x}$. We will show in this section that one can compile a shallow tuple rewrite system R into an algorithm that decides in linear time whether a term is weak in R. In Section 5.2 we will show that a sort term denotes a nonempty set if and only if it is weak in a certain shallow tuple rewrite system that can be obtained from the type specification.

Lemma 4.4.1 *Let $\theta s \Rightarrow_R^* \vec{x}$. Then there exists a tuple $\vec{y}$ of variables such that $s \Rightarrow_R^* \vec{y}$ and $\theta\vec{y} \Rightarrow_R^* \vec{x}$.*

Proof. We prove the claim by induction on the length of the derivation $\theta s \Rightarrow_R^* \vec{x}$. If s is a variable, then the claim is trivial. Otherwise, we have $s = f(\vec{s})$ and there exists a rule $f(\vec{z}) \to \vec{t}$ in R such that $\theta s = \theta f(\vec{s}) \Rightarrow_R \theta\{\vec{z}/\vec{s}\}\vec{t} \Rightarrow_R^* \vec{x}$. Hence we know by the induction hypothesis that there exists a tuple $\vec{y}$ of variables such that $\{\vec{z}/\vec{s}\}\vec{t} \Rightarrow_R^* \vec{y}$ and $\theta\vec{y} \Rightarrow_R^* \vec{x}$. Furthermore, $s \Rightarrow_R^* \{\vec{z}/\vec{s}\}t \Rightarrow_R^* \vec{y}$ since $s \to \{\vec{z}/\vec{s}\}t$ is an instance of $f(\vec{z}) \to \vec{t} \in R$. $\square$

A **mode** of R is a pair $f(\vec{x}) \to V$ such that

1. $\vec{x}$ is a tuple of pairwise distinct variables
2. there exists a tuple $\vec{y}$ of variables such that $f(\vec{x}) \Rightarrow_R^* \vec{y}$ and $\mathcal{V}\vec{y} = V$.

Note that the right-hand side of a mode contains only variables that occur in the left-hand side of the mode. Hence a finite shallow tuple rewrite system has only finitely many modes up to consistent variable renaming.

A set M of modes of R is **complete for** R if, for every pair $f(\vec{x})$ and $\vec{y}$ such that $f(\vec{x}) \Rightarrow_R^* \vec{y}$, there exists a variant $f(\vec{x}) \to V$ of a mode in M such that $V \subseteq \mathcal{V}\vec{y}$.

Proposition 4.4.2 *For every finite shallow tuple rewrite system there exists a finite complete set of modes.*

By assuming a total order on variables, we can assume that every mode $f(\vec{x}) \to V$ defines a unique tuple rewriting rule $f(\vec{x}) \to \vec{y}$ such that $\mathcal{V}\vec{y} = V$ and the variables in $\vec{y}$ are pairwise distinct. Thus every set of modes defines a unique shallow tuple rewrite system. We use $\Rightarrow_M$ to denote the rewriting relation defined by the shallow tuple rewrite system defined by a set of modes M.

Lemma 4.4.3 *Let M be a set of modes of R. Then:*

1. *there are no infinite chains $\vec{s}_1 \Rightarrow_M \vec{s}_2 \Rightarrow_M \vec{s}_3 \Rightarrow_M \cdots$*
2. *if $\vec{s} \Rightarrow_M^* \vec{x}$, there exists a tuple $\vec{y}$ of variables such that $\vec{s} \Rightarrow_R^* \vec{y}$ and $\mathcal{V}\vec{y} = \mathcal{V}\vec{x}$*
3. *M is complete for R if and only if, for every pair $\vec{s}$ and $\vec{x}$ such that $\vec{s} \Rightarrow_R^* \vec{x}$, there exists a tuple $\vec{y}$ of variables such that $\vec{s} \Rightarrow_M^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$.*

Proof. *1.* If $\vec{t}$ is obtained from $\vec{s}$ by applying a tuple rewrite rule obtained from a mode, then the size of $\vec{t}$ is strictly smaller than the size of $\vec{s}$ (although the length of $\vec{t}$ can be greater than the length of $\vec{s}$).

2. Let a derivation $\vec{s} \Rightarrow_M^* \vec{x}$ be given. We prove the claim by induction on on the length of the derivation. If $\vec{s} = \vec{x}$, then the claim is trivial. Otherwise, we have $\vec{s} \Rightarrow_M \vec{t} \Rightarrow_M^* \vec{x}$. Thus we know by the induction hypothesis that there exists a tuple $\vec{y}$ of variables such that $\vec{t} \Rightarrow_R^* \vec{y}$ and $\mathcal{V}\vec{y} = \mathcal{V}\vec{x}$. By the definition of modes we know that there exists a tuple $\vec{u}$ containing exactly the same sort terms as $\vec{t}$ such that $\vec{s} \Rightarrow_R^* \vec{u}$. Hence there exists a tuple $\vec{z}$ of variables such that $\vec{s} \Rightarrow_R^* \vec{u} \Rightarrow_R^* \vec{z}$ and $\mathcal{V}\vec{x} = \mathcal{V}\vec{y} = \mathcal{V}\vec{z}$.

3. Suppose M is complete for R and $s \Rightarrow_R^* \vec{x}$. We show by induction on s that there exists a tuple $\vec{y}$ of variables such that $s \Rightarrow_M^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$. If $s = x$, then $s = \vec{x}$ and the claim is trivial.

If $s = f(\vec{s})$, then we have $s = \{\vec{z}/\vec{s}\}f(\vec{z}) \Rightarrow_R^* \vec{x}$ for some tuple $\vec{z}$ of variables. Hence we know by Lemma 4.4.1 that there exists a tuple $\vec{z}_1$ of variables such that $f(\vec{z}) \Rightarrow_R^* \vec{z}_1$ and $\{\vec{z}/\vec{s}\}\vec{z}_1 \Rightarrow_R^* \vec{x}$. Since M is complete, we know that there exists a tuple $\vec{z}_2$ of variables such that $f(\vec{z}) \Rightarrow_M \vec{z}_2$ and $\mathcal{V}\vec{z}_2 \subseteq \mathcal{V}\vec{z}_1$. Furthermore, we know by the induction hypothesis that there exists a tuple $\vec{y}$ of variables such that $\{\vec{z}/\vec{s}\}\vec{z}_1 \Rightarrow_M^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$. Hence we have $s = \{\vec{z}/\vec{s}\}f(\vec{z}) \Rightarrow_M \{\vec{z}/\vec{s}\}\vec{z}_2 \Rightarrow_M^* \vec{y}$.

To show the other direction of the equivalence, suppose that, for every pair $\vec{s}$ and $\vec{x}$ such that $\vec{s} \Rightarrow_R^* \vec{x}$, there exists a tuple $\vec{y}$ of variables such that $\vec{s} \Rightarrow_M^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$. Furthermore, let $f(\vec{x}) \Rightarrow_R^* \vec{y}$. Then we know that there exists a tuple $\vec{z}$ of variables such that $f(\vec{x}) \Rightarrow_M^* \vec{s}$ and $\mathcal{V}\vec{z} \subseteq \mathcal{V}\vec{y}$. Hence M contains a variant of the mode $f(\vec{x}) \to \mathcal{V}\vec{z}$. $\square$

Theorem 4.4.4 *Let M be a complete set of modes for R. Then*

$$\begin{aligned} f(\vec{s}) \text{ is weak in } R &\iff f(\vec{s}) \text{ is weak in } M \\ &\iff \exists\, f(\vec{x}) \to V \in M \\ &\qquad\quad \forall y \in V.\ \{\vec{x}/\vec{s}\}y \text{ is weak in } M. \end{aligned}$$

Proof. The first equivalence follows immediately from the preceding lemma. Furthermore, the "$\Leftarrow$" direction of the second equivalence is obvious. To show the "$\Rightarrow$" direction of the second equivalence, let $f(\vec{s}) \Rightarrow_M^* \vec{x}_1$. Then we have by Lemma 4.4.1 that $f(\vec{x}_2) \Rightarrow_M \vec{y}$ and $\{\vec{x}_2/\vec{s}\}\vec{y} \Rightarrow_M^* \vec{x}_1$. Hence there exists a mode $f(\vec{x}) \to V \in M$ such that for every variable $y \in V$ there exists a tuple $\vec{z}$ such that $\{\vec{x}/\vec{s}\}y \Rightarrow_M^* \vec{z}$. □

Corollary 4.4.5 *Given a finite complete set of modes for R, one can decide in linear time with respect to the size of a term s whether s is weak in R.*

Proof. To decide whether $f(s_1, \ldots, s_n)$ is weak in R, one first decides this property for every subterm s_i and checks afterwards whether there is an applicable mode in M. □

We will now give an algorithm for computing finite complete sets of modes for finite shallow tuple rewrite systems. Since, in general, $\vec{s} \Rightarrow_R^* \vec{t}$ is not terminating, such an algorithm is not obvious.

In the following we assume without loss of generality that $\vec{x} = \vec{y}$ if $f(\vec{x}) \to \vec{s}$ and $f(\vec{y}) \to \vec{t}$ are rules of R.

Given a finite shallow tuple rewrite system R, we define a descending chain

$$R = R_0 \supseteq R_1 \supseteq R_2 \supseteq \cdots$$

of finite tuple rewrite systems and an ascending chain

$$\emptyset = M_0 \subseteq M_1 \subseteq M_2 \subseteq \cdots$$

of finite sets of modes of R as follows:

$$\begin{aligned} R_{n+1} &:= \{f(\vec{x}) \to \vec{s} \in R_n \mid f(\vec{x}) \to \emptyset \notin M_n\} \\ M_{n+1} &:= M_n \cup \{f(\vec{x}) \to \mathcal{V}\vec{x} \mid \exists\, f(\vec{x}) \to \vec{s} \in R_{n+1}.\ \vec{s} \Rightarrow_{M_n}^* \vec{x}\}. \end{aligned}$$

Lemma 4.4.6 *Let the chain $M_0 \subseteq M_1 \subseteq \cdots$ be defined as above. Then, if $s \Rightarrow_R^* \vec{x}$, there exist a number n and a tuple $\vec{y}$ of variables such that $s \Rightarrow_{M_n}^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$.*

Proof. Let a derivation $s \Rightarrow_R^* \vec{x}$ be given. We prove by induction on the length of the derivation that there exist a number n and a tuple $\vec{y}$ of variables such that $s \Rightarrow_{M_n}^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$. If $s = \vec{x}$, then the claim is trivial. Otherwise, we have $s \Rightarrow_R \vec{t} \Rightarrow_R^* \vec{x}$. Hence we know by the induction hypothesis that there exist a number n and a tuple $\vec{y}$ of variables such that $\vec{t} \Rightarrow_{M_n}^* \vec{y}$ and $\mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$. Since s can't be a variable, we have $s = f(\vec{s})$ and there exists a rule $f(\vec{z}) \to \vec{u}$ in R such that

$$s = f(\vec{s}) \Rightarrow_R \{\vec{z}/\vec{s}\}\vec{u} \Rightarrow_{M_n}^* \vec{y}.$$

Hence we know by Lemma 4.4.1 that there exists a tuple $\vec{z}_1$ of variables such that $\vec{u} \Rightarrow^*_{M_n} \vec{z}_1$ and $\{\vec{z}/\vec{s}\}\vec{z}_1 \Rightarrow^*_{M_n} \vec{y}$.

If $f(\vec{z}) \to \vec{u}$ is not in R_{n+1}, then M_n contains the mode $f(\vec{z}) \to \emptyset$. Hence $s = f(\vec{s}) \Rightarrow_{M_n} \emptyset$, which yields the claim.

If $f(\vec{z}) \to \vec{u}$ is in R_{n+1}, then the mode $f(\vec{z}) \to \mathcal{V}\vec{z}_1$ is in M_{n+1}. Hence there exists a tuple $\vec{v}$ such that $\vec{v}$ contains exactly the same sort terms as $\{\vec{z}/\vec{s}\}\vec{z}_1$ and $s = f(\vec{s}) \Rightarrow_{M_{n+1}} \vec{v}$. Since $\{\vec{z}/\vec{s}\}\vec{z}_1 \Rightarrow^*_{M_n} \vec{y}$, there exists a tuple $\vec{y}_1$ of variables such that $\vec{v} \Rightarrow^*_{M_n} \vec{y}_1$ and $\mathcal{V}\vec{y}_1 = \mathcal{V}\vec{y} \subseteq \mathcal{V}\vec{x}$. Hence $s \Rightarrow^*_{M_{n+1}} \vec{y}_1$ and $\mathcal{V}\vec{y}_1 \subseteq \mathcal{V}\vec{x}$. □

Theorem 4.4.7 [Computation of Complete Sets of Modes] *Let R be a finite shallow tuple rewrite system and let the chain $M_0 \subseteq M_1 \subseteq \cdots$ be defined as above. Then:*

1. *(Termination) there exists a number n such that $M_n = M_{n+1}$*
2. *(Soundness) every M_n is a set of modes of R*
3. *(Completeness) if $M_n = M_{n+1}$, then M_n is a complete set of modes for R.*

Proof. *1.* For every M_n, the left-hand side of every mode in M_n is the left-hand side of a rule in R. Since R is finite, there are only finitely many of such modes.

2. We show by induction on n that M_n is a set of modes of R. Since $M_0 = \emptyset$, the claim is trivial for $n = 0$. To show that M_{n+1} is a set of modes of R, suppose M_n is a set of modes of R, $f(\vec{x}) \to \vec{s}$ is a rule of R, and $\vec{s} \Rightarrow^*_{M_n} \vec{x}$. We have to show that $f(\vec{x}) \to \mathcal{V}\vec{x}$ is a mode of R. Since M_n is a set of modes of R, we know by the preceding lemma that there exists a tuple $\vec{y}$ of variables such that $\vec{s} \Rightarrow^*_R \vec{y}$ and $\mathcal{V}\vec{y} = \mathcal{V}\vec{x}$. Hence $f(\vec{x}) \Rightarrow^*_R \vec{y}$. Thus $f(\vec{x}) \to \mathcal{V}\vec{x}$ is a mode of R.

3. Let $M_n = M_{n+1}$. Then $M_i \subseteq M_n$ for every i. Hence the claim follows by the preceding lemma and Lemma 4.4.3. □

Corollary 4.4.8 *If R is a finite shallow tuple rewrite system and every function symbol of R is a constant, then a complete set of modes for R can be computed in quadratic time.*

Proof. If R contains only constants, then every mode of R has the form $c \to \emptyset$. Hence the iteration $M_0, M_1, \ldots$ can take at most as many steps as there are constants occurring in R. Furthermore, every iteration step can be performed in time linear with respect to the size of R. □

Chapter 5

POS-Types

5.1 Type Specifications
5.2 Inhabitation
5.3 Unifiers and Solution Schemata

We are now ready to define POS-types. They are specified with simple specifications whose inclusions yield a sort rewriting system, where every value function is specified with exactly one rank of the form $f\colon \vec{\mu} \to \xi(\vec{\alpha})$. These specifications, which are just called type specifications, have quasi-extensional initial models and hence we choose the extensional model associated with the initial model as the POS-type specified.

Type specifications do have nice properties. If a value term s is well-typed under a prefix P, then there is a computable least sort σ such that $P \vdash s\colon\sigma$. Together with the existence of sort infima this implies that the sorts of a POS-type are closed under intersection. Moreover, if s is in σ for some prefix, then one can compute a greatest prefix P such that $P \vdash s\colon\sigma$.

Most of this chapter is already devoted to constraint solving. The constraint solver for our relational programs computing over POS-types will have to solve constraints of the form $E \,\&\, M$, where E is a conjunction of equations and M is a conjunction of memberships. In Section 5.2 we will attack the subproblem of deciding whether a constraint $x\colon\sigma$ is satisfiable in a POS-type and develop a linear-time decision algorithm for this problem. In Section 5.3 we will generalize the notion of a unifier to constraints of the form $E \,\&\, M$ and work out how the unifiers of a constraint relate to its solutions. Finally, we will define a notion of solved form for the explicit representation of solutions and investigate its most important properties.

5.1 Type Specifications

From now on we assume that the symbol $-$, called the **empty sort**, is a nullary sort function.

A **type specification** T is a finite set of inclusions and ranks such that

1. the inclusions of T yield a sort rewriting system $R(T)$ if every inclusion $\sigma \sqsupseteq \tau$ is taken as a rewrite rule $\sigma \to \tau$

2. every rank in T has the form $f{:}\vec{\sigma} \to \xi(\vec{\alpha})$, where $\vec{\alpha}$ is a tuple of pairwise distinct sort variables and $\mathcal{V}(\vec{\sigma}) \subseteq \mathcal{V}(\vec{\alpha})$

3. no rank in T contains the empty sort $-$

4. for no function symbol T contains more than one rank

5. T contains at least one value constant.

Proposition 5.1.1 *Every type specification is a simple specification.*

The sort equations in Figure 1.1 define a typical type specification. The left hand side of a sort equation has the form $\xi(\vec{\alpha})$, where $\vec{\alpha}$ is a tuple of pairwise distinct sort variables, and the right hand side of the sort equation gives all ranks that go to $\xi(\vec{\alpha})$ and all sort rewriting rules whose left hand side if $\xi(\vec{\alpha})$. For instance,

$$\mathsf{error_or_list(E,T)} \;:=\; \mathsf{errormsg(E)} \;\sqcup\; \mathsf{list(T)}$$

translates into the inclusions

$$\mathsf{error_or_list(E,T)} \sqsupseteq \mathsf{errormsg(E)}, \quad \mathsf{error_or_list(E,T)} \sqsupseteq \mathsf{list(T)}$$

and

$$\mathsf{list(T)} \;:=\; \mathsf{nil}{:}\,[] \;\sqcup\; \mathsf{cons}{:}\,\mathsf{T} \times \mathsf{list(T)}$$

translates into the ranks

$$\mathsf{nil}{:}\,\mathsf{list(T)}, \quad \mathsf{cons}{:}\,\mathsf{T} \times \mathsf{list(T)} \to \mathsf{list(T)}.$$

Obviously, every type specification can be given by sort equations, and that's the syntax type specifications are presented in in TEL [Smo88b]. Sort equations are also commonly used in functional programming languages such as ML [HMM86], there, of course, without the provision for subsorts. Sort equations already presume that type specifications specify their initial models, which satisfy the equations in the obvious sense. If one has all models of a type specification in mind one could still write sort inclusions $\xi(\vec{\alpha}) \sqsupseteq \cdots$.

General Assumption. *We assume in the rest of this thesis that T is a type specification and that all terms and substitutions employ only function symbols occurring in T. Furthermore, we assume that all constraints and all interpretations are taken from $\mathcal{L}(\Sigma^T)^*$ (Σ^T is the set of all function symbols occurring in T). To obtain a smooth notation, we will always drop the subscripts T and $R(T)$. By this convention the notations*

$\sigma \leq \tau,\ \sigma \geq \tau,$	
$\theta \leq \psi,\ \psi \geq \theta,$	
$P \leq Q,\ P \geq Q$	*inclusion order, see Section 3.6*
$P \vdash s{:}\,\sigma,\ P \vdash M$	*membership relation, see Section 3.6*
$\sigma \Rightarrow^* \tau$	*top level sort rewriting, see Section 4.1*
$\sigma \sqcap \tau,\ \sigma \sqcap^\circ \tau$	*[stable] infimum, see Section 4.3*

are all well-defined.

We will use $\mathrm{LUM}[\vec{\sigma} \sqsubseteq \vec{\tau}]$ to denote the least upper matcher of the inclusion system $\vec{\sigma} \sqsubseteq \vec{\tau}$ (see Section 4.2), where we assume without loss of generality that upper matchers are sort substitutions (that is, map every value variable to itself, see Section 3.3).

Let P be a prefix. Then the following two equations define a computable partial function $\sigma^P[\cdot]$ from value terms to sort terms:

1. $\sigma^P[x] = Px$
2. $\sigma^P[f(\vec{s})] = \mathrm{LUM}[\sigma^P[\vec{s}] \sqsubseteq \vec{\mu}]\xi(\vec{\alpha})$ if and only if $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$.

Theorem 5.1.2 [Least Sort] *$P \vdash s{:}\,\sigma$ if and only if $\sigma^P[s] \leq \sigma$.*

Proof. *1.* Suppose $P \vdash s{:}\,\sigma$. We prove by induction on s that $\sigma^P[s] \leq \sigma$. If $s = x$, then $\sigma^P[x] = Px \leq \sigma$.

Let $s = f(\vec{s})$. Then there exists a rank $f{:}\,\vec{\mu} \to \tau$ in T and a substitution θ such that $\theta\tau \leq \sigma$ and $P \vdash \vec{s}{:}\,\theta\vec{\mu}$. By the induction hypothesis we know that $\sigma^P[\vec{s}] \leq \theta\vec{\mu}$. Hence $\psi := \mathrm{LUM}[\sigma^P[\vec{s}] \sqsubseteq \vec{\mu}] \leq \theta$ and thus $\sigma^P[f(\vec{s})] = \psi\tau \leq \theta\tau \leq \sigma$.

2. Suppose $\sigma^P[s] \leq \sigma$. We prove by induction on s that $P \vdash s{:}\,\sigma$. If $s = x$, then $Px = \sigma^P[s] \leq \sigma$ and hence $P \vdash x{:}\,\sigma$.

Let $s = f(\vec{s})$. Then $\vec{\sigma} := \sigma^P[\vec{s}]$ exists, $f{:}\,\vec{\mu} \to \tau$ is a rank in T, $\theta := \mathrm{LUM}[\vec{\sigma} \sqsubseteq \vec{\mu}]$ exists, and $\theta\tau \leq \sigma$. Furthermore, we know by the induction hypothesis that $P \vdash \vec{s}{:}\,\vec{\sigma}$. Hence $P \vdash \vec{s}{:}\,\theta\vec{\mu}$ since $\vec{\sigma} \leq \theta\mu$. Thus $P \vdash f(\vec{s}){:}\,\sigma$ since $f{:}\,\theta\vec{\mu} \to \theta\tau$ is an instance of a rank in T and $\theta\tau \leq \sigma$. □

If $\sigma^P[s]$ is defined, we say that $\sigma^P[s]$ is the **least sort term of** s under P. The existence of least sort terms is crucial for the existence of good unification and type checking algorithms.

Corollary 5.1.3 *The relation $P \vdash s{:}\,\sigma$ is decidable.*

Corollary 5.1.4 $P \vdash s{:}\,\sigma \ \wedge\ P \vdash s{:}\,\tau \quad \Longleftrightarrow \quad P \vdash s{:}\,\sigma \sqcap \tau.$

Proposition 5.1.5 *If $\sigma^P[s]$ is defined, then $\mathcal{V}(\sigma^P[s]) = \bigcup_{x \in \mathcal{V}s} \mathcal{V}(Px)$.*

Proof. Let $\sigma^P[s]$ be defined. We prove by induction on s that $\mathcal{V}(\sigma^P[s]) = \bigcup_{x \in \mathcal{V}s} \mathcal{V}(Px)$. If $s = x$, then $\sigma^P[x] = Px$ and the claim is trivial.

Let $s = f(\vec{s})$. Then T contains a rank $f\colon \vec{\mu} \to \tau$, $\theta := \mathrm{LUM}[\sigma^P[\vec{s}] \sqsubseteq \vec{\mu}]$ exists, and $\sigma^P[s] = \theta\tau$. We know by the induction hypothesis that $\mathcal{V}(\sigma^P[\vec{s}]) = \bigcup_{x \in \mathcal{V}(f(\vec{s}))} \mathcal{V}(Px)$. Since $\mathcal{V}(\vec{\mu}) \subseteq \mathcal{V}(\tau)$, we know by Proposition 4.2.3 that $\mathcal{V}(\theta\tau) = \mathcal{V}(\sigma^P[\vec{s}]) = \bigcup_{x \in \mathcal{V}(f(\vec{s}))} \mathcal{V}(Px)$. □

Proposition 5.1.6 *If $P \to s\colon\sigma$ is valid in T, then $\mathcal{V}s \subseteq \mathcal{V}P$.*

Proof. Let $P \to s\colon\sigma$ be valid in T, $x_1, \ldots, x_m$ be the variables occurring in s but not in P, and $\alpha_1, \ldots, \alpha_n$ be the variables occurring in σ but not in P. Furthermore, let $\beta_1, \ldots, \beta_m$ be pairwise distinct sort variables not occurring in $P \to s\colon\sigma$ and $y_1, \ldots, y_n$ be pairwise distinct value variables not occurring in $P \to s\colon\sigma$. Then

$$Q \;:=\; P \;\&\; x_1\colon\beta_1 \,\&\, \cdots \,\&\, x_m\colon\beta_m \;\&\; y_1\colon\alpha_1 \,\&\, \cdots \,\&\, y_n\colon\alpha_n$$

is a prefix and $P \to s\colon\sigma$ is valid in $\mathcal{I}(T,Q)$. Let ϵ be an $\mathcal{I}(T,Q)$-assignment that maps every variable occurring in Q to itself. Then $\epsilon \in \mathcal{I}(T,Q)[\![Q]\!] \subseteq \mathcal{I}(T,Q)[\![P]\!]$ and hence $\epsilon \in \mathcal{I}(T,Q)[\![s\colon\sigma]\!]$. Thus $Q \vdash s\colon\sigma$ and $\sigma \geq \sigma^Q[s]$. Hence $\bigcup_{x \in \mathcal{V}s} Qx = \mathcal{V}(\sigma^Q[s]) \subseteq \mathcal{V}\sigma$.

Now suppose $m > 0$. Then $\beta_1 \in \mathcal{V}\sigma$, which contradicts our assumptions. Thus $\mathcal{V}s \subseteq \mathcal{V}P$. □

The preceding proposition allows us to get rid of the variable condition in the Soundness and Completeness Theorem for simple specifications:

Theorem 5.1.7 [Soundness and Completeness] *For every type specification T the following equivalences hold:*

1. *$\sigma \sqsubseteq \tau$ is valid in T $\iff$ $\sigma \leq \tau$*
2. *$P \to s\colon\sigma$ is valid in T $\iff$ $P \vdash s\colon\sigma$*
3. *$P \to s \doteq t$ is valid in T $\iff$ $s = t$ and s is well-typed in T under P.*

Proof. Follows by the preceding proposition from the Soundness Proposition and the Soundness and Completeness Theorem for simple specifications. □

Using $\sigma^P[\cdot]$ we can decide $P \vdash s\colon\sigma$ by going bottom up through s. The next proposition gives us a top down decision method for $P \vdash s\colon\sigma$.

Proposition 5.1.8 *$P \vdash f(\vec{s})\colon\sigma$ holds if and only if T contains a rank $f\colon\vec{\mu} \to \xi(\vec{\alpha})$ such that $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash \vec{s}\colon\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$.*

Proof. "$\Rightarrow$". Let $P \vdash f(\vec{s}):\sigma$. Then T contains a rank $f:\vec{\mu} \to \xi(\vec{\alpha})$ and there exists a substitution θ such that $\theta\xi(\vec{\alpha}) \leq \sigma$ and $P \vdash \vec{s}:\theta\vec{\mu}$. Hence there exists $\vec{\sigma}$ such that $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\vec{\sigma} \geq \theta\vec{\alpha}$. Thus $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \geq \{\vec{\alpha}/\theta\vec{\alpha}\}\vec{\mu} = \theta\vec{\mu}$ and hence $P \vdash \vec{s}:\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ since $P \vdash \vec{s}:\theta\vec{\mu}$.

"$\Leftarrow$". Let $f:\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash \vec{s}:\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Then $f:\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \to \xi(\vec{\sigma})$ is an instance of a rank in T. Hence $P \vdash f(\vec{s}):\sigma$. □

The following reduction rules for membership systems define a binary relation "$M \xrightarrow{\mathrm{d}} M'$", called **membership decomposition**:

1. $M \,\&\, f(\vec{s}):\sigma \xrightarrow{\mathrm{d}} M \,\&\, \vec{s}:\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$
   if $f:\vec{\mu} \to \xi(\vec{\alpha}) \in T$ and $\sigma \Rightarrow^* \xi(\vec{\sigma})$
2. $M \,\&\, x:\sigma \,\&\, x:\tau \xrightarrow{\mathrm{d}} M \,\&\, x:(\sigma \sqcap \tau)$.

Proposition 5.1.9 *The membership decomposition relation "$M \xrightarrow{\mathrm{d}} M'$" has the following properties:*

1. *"$M \xrightarrow{\mathrm{d}} M'$" is terminating and confluent*
2. *if $M \xrightarrow{\mathrm{d}} M'$, then $P \vdash M \iff P \vdash M'$ for every prefix P*
3. *if $P \vdash M$ and M is no prefix, then $\xrightarrow{\mathrm{d}}$ applies to M.*

Proof. The termination of $\xrightarrow{\mathrm{d}}$ is obvious. Since the infima function "$\sigma \sqcap \tau$" is associative, we know that $\xrightarrow{\mathrm{d}}$ is locally confluent and hence confluent. The second and the third claim follow with the preceding proposition. □

If there exists a prefix P such that $M \xrightarrow{\mathrm{d}}{}^* P$, then we call $\mathrm{GP}[M] := P$ the **greatest prefix supporting** M.

Theorem 5.1.10 [Greatest Prefix] *If $P \vdash M$, then $\mathrm{GP}[M]$ exists, $P|_{\mathcal{D}(\mathrm{GP}[M])} \leq \mathrm{GP}[M]$, and $\mathrm{GP}[M] \vdash M$.*

Proof. Follows immediately from the preceding proposition. □

We now show that the initial model $\mathcal{I}(T)$ of a type specification T is quasi-extensional. The **extensional inclusion preorder** $\preceq^{\mathcal{I}(T)}$ is the preorder on ground sort terms satisfying

$$\sigma \preceq^{\mathcal{I}(T)} \tau \;:\Longleftrightarrow\; \forall\, s.\ \vdash s:\sigma \;\Rightarrow \vdash s:\tau.$$

We extend "$\sigma \preceq^{\mathcal{I}(T)} \tau$" to substitutions as follows:

$$\theta \preceq^{\mathcal{I}(T)} \psi \quad :\Longleftrightarrow \quad \mathcal{D}\theta = \mathcal{D}\psi \;\wedge\; \forall\, \alpha \in \mathcal{D}\theta.\ \theta\alpha \preceq^{\mathcal{I}(T)} \psi\alpha.$$

Lemma 5.1.11 *If $\theta \preceq^{\mathcal{I}(T)} \psi$ and $\mathcal{V}\sigma \subseteq \mathcal{D}\theta$, then $\theta\sigma \preceq^{\mathcal{I}(T)} \psi\tau$.*

Proof. Let $\theta \preceq^{\mathcal{I}(T)} \psi$, $\mathcal{V}\sigma \subseteq \mathcal{D}\theta$ and $\vdash s\colon\theta\sigma$. Since s is ground, we know that $s = f(\vec{s})$ and T contains a rank $f\colon\vec{\mu} \to \xi(\vec{\alpha})$. We prove by induction on s that $\vdash s\colon\psi\sigma$.

Let $\sigma = \alpha$. Then $\theta\sigma = \theta\alpha \preceq^{\mathcal{I}(T)} \psi\alpha = \psi\sigma$. Hence $\vdash s\colon\psi\sigma$ since $\vdash s\colon\theta\sigma$.

Let $\sigma = \eta(\vec{\sigma})$. By the preceding proposition we know that $\theta\sigma \Rightarrow^* \xi(\cdots)$. Hence there exists $\vec{\nu}$ such that $\sigma \Rightarrow^* \xi(\vec{\nu})$ and $\theta\sigma \Rightarrow^* \xi(\theta\vec{\nu})$. Thus we know by the preceding proposition that $\vdash \vec{s}\colon\theta(\{\vec{\alpha}/\vec{\nu}\}\vec{\mu})$. Now we have $\vdash \vec{s}\colon\psi(\{\vec{\alpha}/\vec{\nu}\}\vec{\mu})$ by the induction hypothesis, which yields $\vdash f(\vec{s})\colon\psi\sigma$ since $\psi\sigma \Rightarrow^* \xi(\psi\vec{\nu})$. □

Theorem 5.1.12 [Extensional Interpretation] *The initial model $\mathcal{I}(T)$ of a type specification T is quasi-extensional and the extensional algebra $\mathcal{I}(T)^\circ$ associated with $\mathcal{I}(T)$ is a model of T.*

Proof. Follows from the preceding lemma and statement (4) of Corollary 3.5.3. □

We are now ready to make the central definitions of this thesis (this took quite some preparation, didn't it?): the extensional algebra $\mathcal{I}(T)^\circ$ associated with the initial model $\mathcal{I}(T)$ of T is called the **type specified by** T, and a **POS-type** is an extensional POS-algebra that is isomorphic to the type specified by some type specification. For convenience, we use simply $\mathcal{T}$ to denote $\mathcal{I}(T)^\circ$.

As you just read, I prefer to take the extensional interpretation of a type specification as the type it specifies. This is more or less a matter of taste. If you would prefer the initial interpretation, you are free to do so since everything that follows holds unchanged both for the extensional and the initial interpretation. This is due to the fact that inclusions are only used for the specification of POS-types but won't appear in the constraints we will compute with in the following. Furthermore, we will only be interested in solutions for value variables. Formally, the interchangeability of $\mathcal{T}$ and $\mathcal{I}(T)$ is stated best as follows:

Proposition 5.1.13 *If F is an inclusion-free constraint and V is a set of value variables, then $\mathcal{T}[\![F]\!]^V = \mathcal{I}(T)[\![F]\!]^V$.*

Proof. The claim just specializes Corollary 3.5.3. □

Proposition 5.1.14 *If σ is a ground sort term, then*

$$\mathcal{T}[\![\sigma]\!] = \mathrm{VAL}^{\mathcal{I}(T)}[\sigma] = \{s \mid \vdash s\colon\sigma\}.$$

Furthermore,

$$\begin{aligned}
&\mathbf{S}^{\mathcal{T}} = \{\{s \mid \vdash s\colon\sigma\} \mid \ \sigma \text{ is a ground sort term}\},\\
&\mathcal{T}[\![-]\!] = \emptyset,\\
&\mathcal{T}[\![\sigma \sqcap \tau]\!] = \mathcal{T}[\![\sigma]\!] \cap \mathcal{T}[\![\tau]\!] \quad \text{if } \sigma \text{ and } \tau \text{ are ground sort terms,}
\end{aligned}$$

and the sorts of $\mathcal{T}$ are closed under intersection.

Proof. By the construction of $\mathcal{T} = \mathcal{I}(T)^\circ$ (Construction 3.5.1) we know that κ is a homomorphism $\mathcal{I}(T) \to \mathcal{T}$ such that $\kappa(\sigma) = \mathrm{VAL}^{\mathcal{I}(T)}[\sigma]$ for every ground sort term σ. Hence we have $\mathcal{T}[\![\sigma]\!] = \mathrm{VAL}^{\mathcal{I}(T)}[\sigma]$ by Lemma 3.6.9. Furthermore, $\mathrm{VAL}^{\mathcal{I}(T)}[\sigma] = \{s \mid \vdash s{:}\sigma\}$ holds by the construction of $\mathcal{I}(T)$. Since κ is surjective, we know that every sort of $\mathcal{T}$ can be obtained as $\kappa(\sigma) = \mathrm{VAL}^{\mathcal{I}(T)}[\sigma] = \{s \mid \vdash s{:}\sigma\}$.

Now suppose $\mathcal{T}[\![-]\!] \neq \emptyset$. Then we know by what we have just proved that there exists a ground value term s such that $\vdash s{:}-$. Since every rank of T has the form $f{:}\vec{\mu} \to \xi(\vec{\alpha})$, where $\xi \neq -$, this is impossible.

The equation $\mathcal{T}[\![\sigma \sqcap \tau]\!] = \mathcal{T}[\![\sigma]\!] \cap \mathcal{T}[\![\tau]\!]$ follows from Corollary 5.1.4 using $\mathcal{T}[\![\sigma]\!] = \{s \mid \vdash s{:}\sigma\}$, $\mathcal{T}[\![\tau]\!] = \{s \mid \vdash s{:}\tau\}$ and $\mathcal{T}[\![\sigma \sqcap \tau]\!] = \{s \mid \vdash s{:}\sigma \sqcap \tau\}$. □

5.2 Inhabitation

In this section we devise an algorithm that decides in linear time whether a membership $x{:}\sigma$ is valid in $\mathcal{T}$. This algorithm will be one of the cornerstones of the constraint solving algorithm to be presented in Chapter 6. The algorithm is given as a confluent and terminating rewriting system that rewrites a sort term leaving its denotation in $\mathcal{T}$ unchanged such that $x{:}\sigma$ is satisfiable in $\mathcal{T}$ if and only if the normal form of σ isn't $-$.

A sort term σ is called **inhabited** if there exists a ground value term s and a substitution θ such that $\vdash s{:}\theta\sigma$. A sort term is called **void** if it is not inhabited.

A prefix P is **inhabited** if Px is inhabited for every $x \in \mathcal{D}P$. A substitution θ is **inhabited** if $\theta\alpha$ is inhabited for every sort variable $\alpha \in \mathcal{D}\theta$.

A type specification T is **fully inhabited** if every sort term of T that doesn't contain $-$ is inhabited.

Proposition 5.2.1 *In every type specification T the following holds:*

1. *$-$ is void and every sort variable is inhabited*
2. *there exist a value constant c and a ground sort term σ such that $\vdash c{:}\sigma$*
3. *a sort term σ is inhabited if and only if $x{:}\sigma$ is satisfiable in $\mathcal{T}$.*

Proposition 5.2.2 *Let $\vdash s{:}\theta\sigma$. Then there exist a prefix $\vec{x}{:}\vec{\alpha}$ and a linear term t such that s is an instance of t and $\vec{x}{:}\vec{\alpha} \vdash t{:}\sigma$.*

Proof. We prove the claim by induction on s.

If $\sigma = \alpha$, then choose some value variable x and let $t := x$ and $\vec{x}{:}\vec{\alpha} := x{:}\alpha$.

If σ is not a sort variable, then $s = f(\vec{s})$ and T contains a rank $f{:}\vec{\mu} \to \xi(\vec{\beta})$ such that $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\vdash \vec{s}{:}\theta\{\vec{\beta}/\vec{\sigma}\}\vec{\mu}$. Hence we know by the induction hypothesis that there exists a prefix $\vec{x}{:}\vec{\alpha}$ and a linear tuple $\vec{t}$ such that $\vec{s}$ is an instance of $\vec{t}$ and $\vec{x}{:}\vec{\alpha} \vdash \vec{t}{:}\{\vec{\beta}/\vec{\sigma}\}\vec{\mu}$. Thus $\vec{x}{:}\vec{\alpha} \vdash f(\vec{t}){:}\sigma$, $f(\vec{t})$ is linear and s is an instance of $f(\vec{t})$. □

A type specification T defines a finite shallow tuple rewrite system $\mathrm{TR}(T)$ as follows:

$$\mathrm{TR}(T) := \{\xi(\vec{\alpha}) \to \sigma \mid \xi(\vec{\alpha}) \sqsupseteq \sigma \in T\} \ \cup \ \{\xi(\vec{\alpha}) \to \vec{\mu} \mid f\colon \vec{\mu} \to \xi(\vec{\alpha}) \in T\}.$$

We will prove that σ is inhabited if and only if σ is weak in $\mathrm{TR}(T)$.

Proposition 5.2.3 *If $\sigma \Rightarrow^* \tau$ and $\tau \neq -$, then $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \tau$.*

Lemma 5.2.4 *Let $\vec{x}\colon\vec{\alpha} \vdash s\colon\sigma$ and $\mathcal{V}\vec{x} = \mathcal{V}s$. Then there exists a tuple $\vec{\beta}$ of sort variables such that $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \vec{\beta}$ and $\mathcal{V}\vec{\alpha} = \mathcal{V}\vec{\beta}$.*

Proof. We prove the claim by induction on s.

If $s = x$, then $\vec{x}\colon\vec{\alpha} = x\colon\alpha$ and $\sigma = \alpha$ and thus the claim is trivial.

If $s = f(\vec{s})$, then T contains a rank $f\colon\vec{\mu} \to \xi(\vec{\beta})$ such that $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\vec{x}\colon\vec{\alpha} \vdash \vec{s}\colon\{\vec{\beta}/\vec{\sigma}\}\vec{\mu}$. Hence we have by the induction hypothesis that $\{\vec{\beta}/\vec{\sigma}\}\vec{\mu} \Rightarrow^*_{\mathrm{TR}(T)} \vec{\beta}$ and $\mathcal{V}\vec{\alpha} = \mathcal{V}\vec{\beta}$. Thus $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \xi(\vec{\sigma}) \Rightarrow_{\mathrm{TR}(T)} \{\vec{\beta}/\vec{\sigma}\}\vec{\mu} \Rightarrow^*_{\mathrm{TR}(T)} \vec{\beta}$. □

Lemma 5.2.5 *Let $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \vec{\alpha}$. Then there exist a prefix $\vec{x}\colon\vec{\alpha}$ and a linear term s such that $\mathcal{V}\vec{x} = \mathcal{V}s$ and $\vec{x}\colon\vec{\alpha} \vdash s\colon\sigma$.*

Proof. We prove the claim by induction on the length of the derivation $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \vec{\alpha}$. If $\sigma = \vec{\alpha}$, then the claim is trivial. Otherwise, there exists a rank $f\colon\vec{\mu} \to \xi(\vec{\beta})$ such that $\sigma \Rightarrow^* \xi(\vec{\sigma}) \Rightarrow_{\mathrm{TR}(T)} \{\vec{\beta}/\vec{\sigma}\}\vec{\mu} \Rightarrow^*_{\mathrm{TR}(T)} \vec{\alpha}$. Hence we know by the induction hypothesis that there exist a prefix $\vec{x}\colon\vec{\alpha}$ and a linear tuple $\vec{s}$ such that $\mathcal{V}\vec{x} = \mathcal{V}\vec{s}$ and $\vec{x}\colon\vec{\alpha} \vdash \vec{s}\colon\{\vec{\beta}/\vec{\sigma}\}\vec{\mu}$. Hence $\vec{x}\colon\vec{\alpha} \vdash f(\vec{s})\colon\sigma$, which yields the claim. □

Theorem 5.2.6 *A sort term is inhabited if and only if it is weak in $\mathrm{TR}(T)$.*

Proof. Suppose σ is inhabited. Then there exist a value term s and a substitution θ such that $\vdash s\colon\theta\sigma$. Hence we know by Proposition 5.2.2 that there exist a prefix $\vec{x}\colon\vec{\alpha}$ and a term t such that $\vec{x}\colon\vec{\alpha} \vdash t\colon\sigma$ and $\mathcal{V}\vec{x} = \mathcal{V}t$. Thus we know by one of the preceding lemmas that $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \vec{\alpha}$.

Suppose $\sigma \Rightarrow^*_{\mathrm{TR}(T)} \vec{\alpha}$. Then we know by the preceding lemma that there exist a prefix $\vec{x}\colon\vec{\alpha}$ and a term s such that $\vec{x}\colon\vec{\alpha} \vdash s\colon\sigma$. Since there exists a ground membership that is valid in T, there exists a substitution θ such that $\vdash \theta s\colon\theta\sigma$. Hence σ is inhabited. □

Corollary 5.2.7 *Let M be a complete set of modes for $\mathrm{TR}(T)$. Then a sort term $\xi(\vec{\sigma})$ is inhabited if and only if there exists a mode $\xi(\vec{\alpha}) \to V \in M$ such that every sort term in $\{\vec{\alpha}/\vec{\sigma}\}V$ is inhabited.*

Proof. Follows from the preceding theorem and Theorem 4.4.4. □

Corollary 5.2.8 *For a given type specification T, one can decide in linear time whether a sort term is inhabited in T.*

Proof. Follows from the preceding theorem, Corollary 4.4.5 and Proposition 4.4.2. □

Corollary 5.2.9 *Inhabited prefixes and substitutions have the following properties:*

1. *if θ is inhabited, then a sort term σ is inhabited if and only if $\theta\sigma$ is inhabited*
2. *two substitutions θ and ψ are inhabited if and only if their composition $\theta\psi$ is inhabited*
3. *if θ is inhabited, then a prefix P is inhabited if and only if θP is inhabited*
4. *a prefix P is inhabited if and only if P is satisfiable in $\mathcal{T}$*
5. *if P is inhabited and $P \vdash s{:}\sigma$, then σ is inhabited.*

Next we give simplification rules for sort terms such that a sort term is void if and only if its normal form is $-$. The application of a simplification rule to a sort term σ won't change the denotation of σ in $\mathcal{T}$.

A sort term $\xi(\vec{\sigma})$ is called **top-level void** if $- \in \{\vec{\alpha}/\vec{\sigma}\}V$ for every mode $\xi(\vec{\alpha}) \to V$ of $\mathrm{TR}(T)$.

Proposition 5.2.10 *Top-level voidness has the following properties:*

1. *every sort term that is top-level void is void*
2. *every instance of a top-level void sort term is top-level void*
3. *if ξ has no mode in $\mathrm{TR}(T)$, the $\xi(\vec{\sigma})$ is top-level void; in particular, $-$ is top-level void*
4. *if M is a complete set of modes for $\mathrm{TR}(T)$, then $\xi(\vec{\sigma})$ is top-level void if and only if $- \in \{\vec{\alpha}/\vec{\sigma}\}V$ for every mode $\xi(\vec{\alpha}) \to V \in M$.*

We write $\sigma \xrightarrow{\mathrm{n}} \tau$ if τ can be obtained from σ by replacing a top-level void subterm $\mu \neq -$ with $-$.

Proposition 5.2.11 *The relation $\sigma \xrightarrow{\mathrm{n}} \tau$ is terminating and confluent.*

Proof. The termination follows from the fact that every reduction step reduces the number of occurrences of sort function symbols different from $-$. Since $\sigma \xrightarrow{\mathrm{n}} \tau$ is locally confluent, we hence know that $\sigma \xrightarrow{\mathrm{n}} \tau$ is confluent. □

A sort term is called **normal** if it is normal with respect to to $\xrightarrow{\mathrm{n}}$. We use $\mathrm{NF}[\sigma]$ to denote the **normal form of** σ with respect to $\xrightarrow{\mathrm{n}}$.

We extend the normalization relation to prefixes as follows:

$$P \xrightarrow{\mathrm{n}}{}^{*} Q \;:\Longleftrightarrow\; \mathcal{D}P = \mathcal{D}Q \;\wedge\; \forall\, x \in \mathcal{D}P.\;\; Px \xrightarrow{\mathrm{n}}{}^{*} Qx.$$

Note that "$P \xrightarrow{\mathrm{n}}{}^* Q$" is again terminating and confluent. We use $\mathrm{NF}[P]$ to denote the normal form of P. We call a prefix **normal** if every sort term occurring in it is normal.

If θ and ψ are substitutions, let $\theta \xrightarrow{\mathrm{n}}{}^* \psi$ if and only if $\theta\alpha \xrightarrow{\mathrm{n}}{}^* \theta\alpha$ for every sort variable α and θ and ψ agree on all value variables. If θ is a substitution, then $\mathrm{NF}[\theta]$ is the substitution satisfying $(\mathrm{NF}[\theta])x = \theta x$ and $(\mathrm{NF}[\theta])\alpha = \mathrm{NF}[\theta\alpha]$ for every value variable x and every sort variable α. Note that $\theta \xrightarrow{\mathrm{n}}{}^* \mathrm{NF}[\theta]$ and "$\theta \xrightarrow{\mathrm{n}}{}^* \psi$" is confluent but, in general, not terminating.

Theorem 5.2.12 [Normalization] *The normalization relation "$\sigma \xrightarrow{\mathrm{n}} \tau$" has the following properties:*

1. *if $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$ and $\theta \xrightarrow{\mathrm{n}}{}^* \psi$, then $\theta\sigma \xrightarrow{\mathrm{n}}{}^* \psi\tau$ and $\psi\tau \leq \theta\sigma$*
2. *$\theta\sigma \xrightarrow{\mathrm{n}}{}^* \mathrm{NF}[\theta]\sigma$, $\theta\mathrm{NF}[\sigma]$ and $\mathrm{NF}[\theta]\sigma$, $\theta\mathrm{NF}[\sigma] \xrightarrow{\mathrm{n}}{}^* \mathrm{NF}[\theta\sigma]$*
3. *(Orthogonality) if $\sigma \Rightarrow^* \xi(\vec{\sigma})$, $\tau \Rightarrow^* \xi(\vec{\tau})$ and $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$, then $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau}$*
4. *(Invariance) if $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$, then σ is inhabited if and only if τ is inhabited*
5. *(Completeness) if σ is normal, then σ is void if and only if $\sigma = -$*
6. *σ is void $\iff$ $\sigma \xrightarrow{\mathrm{n}}{}^* - \iff \mathrm{NF}[\sigma] = -$*
7. *$\mathrm{NF}[\alpha] = \alpha$ and*

$$\mathrm{NF}[\xi(\vec{\sigma})] \;=\; \begin{cases} - & \textit{if } \xi(\mathrm{NF}[\vec{\sigma}]) \textit{ is top-level void} \\ \xi(\mathrm{NF}[\vec{\sigma}]) & \textit{otherwise} \end{cases}$$

8. *$\mathrm{NF}[\sigma]$ can be computed in linear time*
9. *(Monotonicity) if $\sigma \leq \tau$, then $\mathrm{NF}[\sigma] \leq \mathrm{NF}[\tau]$*
10. *if σ is normal, $\sigma \leq \tau$ and $\tau \xrightarrow{\mathrm{n}}{}^* \mu$, then $\sigma \leq \mu$*
11. *if P is a normal and inhabited prefix and $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$, then $P \vdash s{:}\sigma$ if and only if $P \vdash s{:}\tau$*
12. *(Invariance) if $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$, then $\mathcal{T}[\![\sigma]\!]_\delta = \mathcal{T}[\![\tau]\!]_\delta$ for every $\mathcal{T}$-assignment δ*
13. *if P is normal and inhabited and s is well-typed under P, then $\sigma^P[s]$ is normal and inhabited.*

Proof. *1.* Let $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$ and $\theta \xrightarrow{\mathrm{n}}{}^* \psi$. We prove by induction on σ that $\theta\sigma \xrightarrow{\mathrm{n}}{}^* \psi\tau$ and $\psi\tau \leq \theta\sigma$.

Let $\sigma = \alpha$. Then $\tau = \alpha$ and hence $\theta\sigma = \theta\alpha \xrightarrow{\mathrm{n}}{}^* \psi\alpha = \psi\theta$. It remains to show that $\psi\alpha \leq \theta\alpha$, which follows by a straightforward induction on the length of a derivation $\theta\alpha \xrightarrow{\mathrm{n}}{}^* \psi\alpha$.

Let $\sigma = \xi(\vec{\sigma})$ and $\tau = -$. Then $- = \psi\tau \leq \theta\sigma$ and it remains to show that $\theta\sigma \xrightarrow{\mathrm{n}}{}^* -$, which follows by induction on the length of a derivation $\sigma \xrightarrow{\mathrm{n}}{}^* -$ exploiting the fact that every instance of a top-level void sort term is top-level void.

Let $\sigma = \xi(\vec{\sigma})$ and $\tau \neq -$. Then $\tau = \xi(\vec{\tau})$ and $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau}$. Hence we know by the induction hypothesis that $\theta\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \psi\vec{\tau} \leq \theta\vec{\sigma}$, which yields $\theta\xi(\vec{\sigma}) \xrightarrow{\mathrm{n}}{}^* \psi\xi(\vec{\tau}) \leq \theta\xi(\vec{\sigma})$.

2. Follows from the first statement and the confluence of $\xrightarrow{\mathrm{n}}{}^*$.

3. Let $\sigma \Rightarrow^* \xi(\vec{\sigma})$, $\tau \Rightarrow^* \xi(\vec{\tau})$ and $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$. We show that $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau}$.

If $\tau = -$, then $- = \xi(\vec{\tau}) = \xi(\vec{\sigma})$ and hence the claim is trivial. Otherwise, $\sigma = \eta(\vec{\sigma}_1)$, $\tau = \eta(\vec{\tau}_1)$ and $\vec{\sigma}_1 \xrightarrow{\mathrm{n}}{}^* \vec{\tau}_1$. Furthermore, we have $\eta(\vec{\alpha}) \Rightarrow^* \xi(\vec{\mu})$, $\vec{\sigma} = \{\vec{\alpha}/\vec{\sigma}_1\}\vec{\mu}$ and $\vec{\tau} = \{\vec{\alpha}/\vec{\tau}_1\}\vec{\mu}$. Hence $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau}$ by statement (1) since $\{\vec{\alpha}/\vec{\sigma}_1\} \xrightarrow{\mathrm{n}}{}^* \{\vec{\alpha}/\vec{\tau}_1\}$.

4. Let $\sigma \xrightarrow{\mathrm{n}} \tau$. Then $\tau \leq \sigma$ by statement (1) and hence σ is inhabited if τ is inhabited. Now suppose σ is inhabited. We show by induction on σ that τ is inhabited.

Since $\sigma \xrightarrow{\mathrm{n}} \tau$, we know that σ is no variable. Let $\sigma = \xi(\vec{\sigma})$. Since σ is inhabited, σ is not top-level void and hence $\tau = \xi(\vec{\tau})$ and $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau}$. Since σ is inhabited, there exists a mode $\xi(\vec{\alpha}) \to V$ of $\mathrm{TR}(T)$ such that every element of $\{\vec{\alpha}/\vec{\sigma}\}V$ is inhabited. By the induction hypothesis we know that every element of $\{\vec{\alpha}/\vec{\tau}\}V$ is inhabited. Hence $\tau = \xi(\vec{\tau})$ is inhabited.

5. Let σ be a normal sort term different from $-$. We prove by induction on σ that σ is inhabited. If σ is a variable, then the claim is trivial. Otherwise, let $\sigma = \xi(\vec{\sigma})$. Since $\xi(\vec{\sigma})$ is not top-level void, there exists a mode $\xi(\vec{\alpha}) \to V$ of $\mathrm{TR}(T)$ such that $- \notin \{\vec{\alpha}/\vec{\sigma}\}V$. Since every sort term in $\{\vec{\alpha}/\vec{\sigma}\}V$ is normal, we know by the induction hypothesis that ever sort term in $\{\vec{\alpha}/\vec{\sigma}\}V$ is inhabited. Hence we know by Corollary 5.2.9 that $\sigma = \xi(\vec{\sigma})$ is inhabited.

6. Follows immediately from statement (5).

7. Obvious.

8. Follows immediately from statement (7).

9. Let $\sigma \leq \tau$. We prove by induction on σ that $\mathrm{NF}[\sigma] \leq \mathrm{NF}[\tau]$. If $\sigma = \alpha$, then $\sigma = \tau$ and hence $\mathrm{NF}[\sigma] = \mathrm{NF}[\tau]$. Otherwise, let $\sigma = \xi(\vec{\sigma})$. Then we have $\tau = \eta(\vec{\tau})$, $\eta(\vec{\alpha}) \Rightarrow^* \xi(\vec{\mu})$, where $\vec{\alpha}$ is linear, and $\vec{\sigma} \leq \{\vec{\alpha}/\vec{\tau}\}\vec{\mu}$. Hence we know by the induction hypothesis that $\mathrm{NF}[\vec{\sigma}] \leq \mathrm{NF}[\{\vec{\alpha}/\vec{\tau}\}\vec{\mu}]$. Thus $\mathrm{NF}[\vec{\sigma}] \leq \{\vec{\alpha}/\mathrm{NF}[\vec{\tau}]\}\vec{\mu}$ by statement (2). If $\mathrm{NF}[\sigma] = -$, then the claim is trivial. Otherwise, $\mathrm{NF}[\sigma] = \xi(\mathrm{NF}[\vec{\sigma}])$ and σ is inhabited. Hence τ is inhabited and thus $\mathrm{NF}[\tau] = \eta(\mathrm{NF}[\vec{\tau}]) \Rightarrow^* \xi(\{\vec{\alpha}/\mathrm{NF}[\vec{\tau}]\}\vec{\mu}) \geq \xi(\mathrm{NF}[\vec{\sigma}]) = \mathrm{NF}[\sigma]$.

10. Let σ be normal, $\sigma \leq \tau$ and $\tau \xrightarrow{\mathrm{n}}{}^* \mu$. Then $\sigma = \mathrm{NF}[\sigma] \leq \mathrm{NF}[\tau]$ by the preceding statement. Hence $\mu \xrightarrow{\mathrm{n}}{}^* \mathrm{NF}[\tau]$ by the confluence of $\xrightarrow{\mathrm{n}}$ and thus $\sigma = \mathrm{NF}[\sigma] \leq \mathrm{NF}[\tau] \leq \mu$.

11. Let P be a normal and inhabited prefix and $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$. Since $\tau \leq \sigma$ by the first statement, $P \vdash s{:}\tau$ implies $P \vdash s{:}\sigma$. To show the other direction, let $P \vdash s{:}\sigma$. We show by induction on s that $P \vdash s{:}\tau$.

If $s = x$, then $Px \leq \tau$. Since Px is normal, we know by the preceding statement that $Px \leq \tau$ and thus $P \vdash x{:}\tau$.

If $s = f(\vec{s})$, then we have $f{:}\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash \vec{s}{:}\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Since P is inhabited, σ is inhabited and hence, using statement (5), σ and τ have the same top symbol. Hence $\tau \Rightarrow^* \xi(\vec{\tau})$ and thus $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau}$ by statement (3). Hence $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \xrightarrow{\mathrm{n}}{}^* \{\vec{\alpha}/\vec{\tau}\}\vec{\mu}$ by statement (1) and thus $P \vdash \vec{s}{:}\{\vec{\alpha}/\vec{\tau}\}\vec{\mu}$ by the induction hypothesis. Hence $P \vdash s{:}\tau$.

12. Let $\sigma \xrightarrow{\mathrm{n}}{}^{*} \tau$ and let δ be an $\mathcal{I}(T)$-assignment. Since the canonical homomorphism $\kappa : \mathcal{I}(T) \to \mathcal{T} = \mathcal{I}(T)^{\circ}$ is surjective, it suffices to show that $\mathcal{T}[\![\sigma]\!]_{\kappa\delta} = \mathcal{T}[\![\tau]\!]_{\kappa\delta}$. By the Homomorphism Theorem and the definition of κ we know that

$$\mathcal{T}[\![\mu]\!]_{\kappa\delta} = \kappa(\mathcal{I}(T)[\![\mu]\!]_{\delta}) = \mathrm{VAL}^{\mathcal{I}(T)}[\theta_{\delta}\mu]$$

for every sort term μ. Hence the claim follows by statement (11) since $\theta_{\delta}\sigma \xrightarrow{\mathrm{n}}{}^{*} \theta_{\delta}\tau$ by statement (1).

13. Let P be normal and inhabited and s be well-typed under P. Then $P \vdash s{:}\sigma^{P}[s]$ and hence $P \vdash s{:}\mathrm{NF}[\sigma^{P}[s]]$ by statement (11). Thus $\sigma^{P}[s] \leq \mathrm{NF}[\sigma^{P}[s]] \leq \sigma^{P}[s]$ by the Least Sort Theorem and statement (1). Furthermore, we know by Corollary 5.2.9 that $\sigma^{P}[s]$ is inhabited. □

Corollary 5.2.13 [Full Inhabitation] *A type specification T is fully inhabited if and only if for every sort function symbol $\xi \neq -$ of T there exists a mode $\xi(\vec{\alpha}) \to V$ of $\mathrm{TR}(T)$. Hence it is decidable whether a type specification is fully inhabited. Furthermore, if T is fully inhabited, then every sort term not containing $-$ is inhabited and normal.*

Example 5.2.14 Let T be the type specification

$$\begin{aligned} \zeta &= c{:}[\,] \\ \xi(\alpha,\beta) &= f{:}\alpha \times \beta \ \sqcup\ g{:}\xi(\alpha,\alpha). \end{aligned}$$

Then $\mathrm{TR}(T)$ consists of the rules

$$\zeta \to \emptyset, \quad \xi(\alpha,\beta) \to \alpha\beta, \quad \xi(\alpha,\beta) \to \xi(\alpha,\alpha).$$

A complete set of modes for $\mathrm{TR}(T)$ is obtained with three iteration steps, where

$$M_2 = M_3 = \{\zeta \to \emptyset,\ \ \xi(\alpha,\beta) \to \{\alpha,\beta\},\ \ \xi(\alpha,\beta) \to \{\alpha\}\}.$$

The subset $\{\zeta \to \emptyset,\ \xi(\alpha,\beta) \to \{\alpha\}\}$ is a minimal complete set of modes for T. Using the modes one verifies easily that $\xi(\zeta,-)$ is normal and hence inhabited. □

Example 5.2.15 Let T be the type specification

$$\begin{aligned} \zeta &= c{:}[\,] \\ \xi(\alpha,\beta) &= f{:}\alpha \ \sqcup\ g{:}\beta \\ \eta(\alpha) &= h{:}\xi(\alpha,\eta(\alpha)). \end{aligned}$$

Then $\mathrm{TR}(T)$ consists of the rules

$$\zeta \to \emptyset, \quad \xi(\alpha,\beta) \to \alpha, \quad \xi(\alpha,\beta) \to \beta, \quad \eta(\alpha) \to \xi(\alpha,\eta(\alpha)).$$

A minimal complete set of modes for $\mathrm{TR}(T)$ is obtained with three iteration steps, where

$$M_2 = M_3 = \{\zeta \to \emptyset,\ \ \xi(\alpha,\beta) \to \{\alpha\},\ \ \xi(\alpha,\beta) \to \{\beta\},\ \ \eta(\alpha) \to \{\alpha\}\}.$$

Using the modes one verifies easily that $\mathrm{NF}[\xi(\eta(-),-)] = -$. Hence $\xi(\eta(-),-)$ is void. □

Example 5.2.16 Let T be the type specification

$$\begin{aligned} \zeta &= c\colon[] \\ \xi(\alpha,\beta) &= f\colon \alpha \times \beta \\ \eta(\alpha) &= g\colon \xi(\alpha,\eta(\alpha)). \end{aligned}$$

Then $\mathrm{TR}(T)$ consists of the rules

$$\zeta \to \emptyset, \quad \xi(\alpha,\beta) \to \alpha\beta, \quad \eta(\alpha) \to \xi(\alpha,\eta(\alpha)).$$

A minimal complete set of modes for $\mathrm{TR}(T)$ is obtained with two iteration steps, where

$$M_1 = M_2 = \{\zeta \to \emptyset, \ \ \xi(\alpha,\beta) \to \{\alpha,\beta\}\}.$$

Using the modes one verifies easily that $\mathrm{NF}[\xi(\eta(\zeta),\zeta)] = -$. Hence $\xi(\eta(\zeta),\zeta)$ is void. □

5.3 Unifiers and Solution Schemata

In this section we will investigate solved forms for representing the V-solutions of a constraint as explicitly as possible. The idea is that we represent the solutions of a constraint for the variables $x_1,\ldots,x_n$ by a constraint

$$\begin{aligned} &x_1 \doteq s_1 \,\&\, \ldots \,\&\, x_k \doteq s_k \ \ \& \\ &x_{k+1}\colon\sigma_{k_1} \,\&\, \ldots \,\&\, x_n\colon\sigma_n \ \ \& \\ &y_1\colon\tau_1 \,\&\, \ldots \,\&\, y_m\colon\tau_m \end{aligned}$$

such that no s_i contains one of the variables x_j, every y_j occurs in some s_i, and every sort term is inhabited and normal.

Our constraint solving methods will apply to constraints of the form $E \,\&\, M$, where E is a conjunction of equations and M is a conjunction of memberships. For such constraints we will define suitable unifiers generalizing ordinary unifiers for unsorted terms. Unifiers are a slightly more general and more syntactically oriented alternative to solutions and will play a central role in Chapter 6, where we will develop unification and constraint solving methods. It will turn out that systems of the form $E \,\&\, M$ have principal unifiers, provided they satisfy certain weak well-typedness conditions.

An **equation system** is a possibly empty conjunction of equations. Recall that a membership system is a possibly empty conjunction of memberships. From now on, the letter E will always denote an equation system and the letter M will always denote a membership system. We use $\mathrm{NF}[M]$ to denote the membership system that can be obtained from M by replacing every sort term with its normal form.

An equation $s \doteq t$ is **trivial** if $s = t$. An equation system is **trivial** if every equation occurring in it is trivial. An equation $s \doteq t$ is **well-typed** under a prefix P if there exists a sort term σ such that $P \vdash s\colon\sigma$ and $P \vdash t\colon\sigma$. An equation system is **well-typed** under a prefix P if every equation occurring in it is well-typed under P. An equation is **well-typed** if it is well-typed under the empty prefix. An equation system is **well-typed** if every equation occurring in it is well-typed.

A **unifier** is a pair θ_P consisting of a substitution θ and a normal and inhabited prefix P. A **unifier** of a conjunction $E \,\&\, M$ is a unifier θ_P such that $P \vdash \theta M$ and the equation system θE is trivial and well-typed under P. We use $\mathrm{U}[E \,\&\, M]$ to denote the set of all unifiers of $E \,\&\, M$. We say that a conjunction $E \,\&\, M$ is **unifiable** if $E \,\&\, M$ has a unifier.

Proposition 5.3.1 [Unifier] *Unifiers have the following properties:*

1. $\mathrm{U}[E \,\&\, M] = \mathrm{U}[E] \cap \mathrm{U}[M] = \mathrm{U}[E \,\&\, \mathrm{NF}[M]]$
2. *if P is a normal and inhabited prefix, then*
$$\theta_P \in \mathrm{U}[E \,\&\, M] \iff P \to \theta E \,\&\, \theta M \text{ is valid in } T$$
3. *if δ is an $\mathcal{I}(T)$-assignment, then*
$$\delta \in \mathcal{I}(T)[\![E \,\&\, M]\!] \iff (\theta_\delta)_\emptyset \in \mathrm{U}[E \,\&\, M]$$
4. ?????????? *4.* $E \,\&\, M$ *satisfiable in* $\mathcal{T}$ $\iff$ $E \,\&\, M$ *satisfiable in* $\mathcal{I}(T)$
$\iff$ $E \,\&\, M$ *is unifiable*
5. *if* $\mathrm{U}[E \,\&\, M] \subseteq \mathrm{U}[E' \,\&\, M']$, *then* $\mathrm{U}[\theta E \,\&\, \theta M] \subseteq \mathrm{U}[\theta E' \,\&\, \theta M']$
6. *if* $\mathrm{U}[E \,\&\, M] \subseteq \mathrm{U}[E' \,\&\, M']$, *then* $\mathcal{I}(T)[\![E \,\&\, M]\!] \subseteq \mathcal{I}(T)[\![E' \,\&\, M']\!]$.

Proof. *1.* The first equation is obvious. The second equation follows from the Normalization Theorem.

2. Follows from the Soundness and Completeness Theorem for type specifications.

3. Follows from Proposition 3.6.5 and the Soundness and Completeness Theorem for type specifications.

4. The first equivalence follows from Corollary 3.5.3. The direction "$\Rightarrow$" of the second equivalence follows from statement (3). To show the other direction, suppose $\theta_P \in \mathrm{U}[E \,\&\, M]$. Then we know by statement (2) that $P \to \theta E \,\&\, \theta M$ is valid in $\mathcal{T}$. Since P is inhabited, P is satisfiable in $\mathcal{I}(T)$ (Corollary 5.2.9). Hence $\theta E \,\&\, \theta M$ is satisfiable in $\mathcal{I}(T)$. Thus we know by Lemma 3.3.5 that $E \,\&\, M$ is satisfiable in $\mathcal{I}(T)$.

5. Let $\mathrm{U}[E \,\&\, M] \subseteq \mathrm{U}[E' \,\&\, M']$ and suppose $\psi_Q \in \mathrm{U}[\theta E \,\&\, \theta M]$. Then $(\psi\theta)_Q \in \mathrm{U}[E \,\&\, M] \subseteq \mathrm{U}[E' \,\&\, M']$. Hence $\psi_Q \in \mathrm{U}[\theta E' \,\&\, \theta M']$.

6. Let $\mathrm{U}[E \,\&\, M] \subseteq \mathrm{U}[E' \,\&\, M']$ and suppose $\delta \in \mathcal{I}(T)[\![E \,\&\, M]\!]$. Then we know by statement (3) that $(\theta_\delta)_\emptyset \in \mathrm{U}[E \,\&\, M] \subseteq \mathrm{U}[E' \,\&\, M']$ and hence, again by statement (3), that $\delta \in \mathcal{I}(T)[\![E' \,\&\, M']\!]$. □

Recall that a value substitution is a substitution that maps every sort variable to itself. A finite value substitution $\theta = \{x_1/s_1, \ldots, x_n/s_n\}$ defines an equation system $\mathrm{E}[\theta]$ as follows:

$$\mathrm{E}[\theta] := (x_1 \doteq s_1 \,\&\, \ldots \,\&\, x_n \doteq s_n).$$

Proposition 5.3.2 *Let θ be a finite and idempotent value substitution and P be an inhabited and normal prefix such that $P \vdash \theta P$. Then $\theta_P \in \mathrm{U}[\mathrm{E}[\theta] \,\&\, P]$.*

A **principal unifier** is a unifier θ_P such that θ is a finite and idempotent value substitution, $P \vdash \theta P$ and $\mathcal{D}\theta \subseteq \mathcal{D}P$. A **principal unifier** of $E \,\&\, M$ is a principal unifier θ_P such that $\mathrm{U}[\mathrm{E}[\theta] \,\&\, P] = \mathrm{U}[E \,\&\, M]$.

Proposition 5.3.3 *If θ_P is a principal unifier, then $\theta_P \in \mathrm{U}[\mathrm{E}[\theta] \,\&\, P]$.*

General Assumption. *In the following V is a set of value variables.*

Proposition 5.3.4 *For every conjunction $E \,\&\, M$ the following holds:*

$$\begin{aligned} \mathcal{T}[\![E \,\&\, M]\!]^V \;=\; \mathcal{I}(T)[\![E \,\&\, M]\!]^V \;=\; \{\delta|_V \mid\; & \delta \in \mathrm{ASS}^{\mathcal{I}(T)}, \\ & \theta_\delta \text{ is normal and inhabited, and} \\ & (\theta_\delta)_\emptyset \in \mathrm{U}[E \,\&\, M]\ \}. \end{aligned}$$

Proof. The first equation follows from Corollary 3.5.3 since $\mathcal{T} = \mathcal{I}(T)^\circ$. The direction "$\supseteq$" of the second equation follows from statement (3) of the Unifier Proposition.

To show the direction "$\subseteq$" of the second equation, let $\delta \in \mathcal{I}(T)[\![E \,\&\, M]\!]$. Then we know by the Unifier Proposition that $(\theta_\delta)_\emptyset \in \mathrm{U}[E \,\&\, M]$ and hence by the Normalization Theorem that $\mathrm{NF}[\theta_\delta]_\emptyset \in \mathrm{U}[E \,\&\, M]$. Now let σ be some normal and inhabited ground sort term and let δ' be the $\mathcal{I}(T)$-assignment defined as follows: $\delta'(x) = \delta(x)$ for every value variable x, $\delta'(\alpha) = \sigma$ if $\mathrm{NF}[\delta(\alpha)] = -$, and $\delta'(\alpha) = \mathrm{NF}[\delta(\alpha)]$ otherwise. Since $\mathrm{NF}[\theta_\delta] \leq \theta_{\delta'}$, we know that $(\theta_{\delta'})_\emptyset \in \mathrm{U}[E \,\&\, M]$. This yields the claim since δ and δ' agree on V. □

We are now ready to define solved forms for explicitly representing V-solutions.

A **V-solution schema** is a unifier θ_P such that

1. θ is a finite and idempotent value substitution
2. θx is well-typed under P for every $x \in \mathcal{D}\theta$
3. $\mathcal{D}\theta \subseteq V$, $\mathcal{D}P \subseteq V \cup \mathcal{I}\theta$, and $\mathcal{D}P$ and $\mathcal{D}\theta$ are disjoint.

A **V-solution schema for** $E \,\&\, M$ is a V-solution schema θ_P such that $\mathcal{T}[\![E \,\&\, M]\!]^V = \mathcal{T}[\![\mathrm{E}[\theta] \,\&\, P]\!]^V$.

The next theorem makes precise in what sense a V-solution schema represents the solutions of a constraint "explicitly".

Theorem 5.3.5 [Solution Schema] *Let θ_P be a V-solution schema for $E \,\&\, M$. Then $E \,\&\, M$ is satisfiable in $\mathcal{T}$. Furthermore, if $V \subseteq \mathcal{D}P \cup \mathcal{D}\theta$, then*

$$\mathcal{T}[\![E \,\&\, M]\!]^V = \mathcal{I}(T)[\![E \,\&\, M]\!]^V = \{(\psi\theta)|_V \mid \vdash \psi P\}.$$

Proof. Since θ_P is a V-solution schema for $E \,\&\, M$, we know that $\mathcal{T}[\![E \,\&\, M]\!]^V = \mathcal{T}[\![\mathrm{E}[\theta] \,\&\, P]\!]^V$. Hence $E \,\&\, M$ is satisfiable in $\mathcal{T}$ if $\mathrm{E}[\theta] \,\&\, P$ is unifiable. Since $\mathcal{D}P$ and $\mathcal{D}\theta$ are disjoint and θx is well-typed under P for every $x \in \mathcal{D}\theta$, $Q := P \,\&\, \{x{:}\,\sigma^P[\theta x] \mid x \in \mathcal{D}\theta\}$ defines a prefix. Since P is normal and inhabited, we know by the Normalization Theorem that Q is normal and inhabited. Now it's easy to verify that $\theta_Q \in \mathrm{U}[\mathrm{E}[\theta] \,\&\, P]$.

To show the second claim, let $V \subseteq \mathcal{D}P \cup \mathcal{D}\theta$. Since the first equation has been already established in the preceding proposition, it suffices to show that $\mathcal{I}(T)[\![\mathrm{E}[\theta] \,\&\, P]\!]^V = \{(\psi\theta)|_V \mid \vdash \psi P\}$.

"$\subseteq$". Let $\delta \in \mathcal{I}(T)[\![\mathrm{E}[\theta] \,\&\, P]\!]$. Then we know by the Unifier Proposition that $(\theta_\delta)_\emptyset \in \mathrm{U}[\mathrm{E}[\theta] \,\&\, P]$ and hence that $\theta_\delta = \theta_\delta\theta$ and $\vdash \theta_\delta P$. Thus $\delta|_V = (\theta_\delta)|_V = (\theta_\delta\theta)|_V \in \{(\psi\theta)|_V \mid \vdash \psi P\}$.

"$\supseteq$". Let $\vdash \psi P$. Furthermore, let π be a sort substitution such that $\pi\alpha$ is ground for every sort variable α. Then $\vdash \pi\psi P$.

First we show that $\pi\psi\theta x = \psi\theta x$ is a well-typed ground value term for every $x \in \mathcal{D}P \cup \mathcal{D}\theta$. If $x \in \mathcal{D}P$, then $x \notin \mathcal{D}\theta$ and hence $\psi\theta x = \psi x$. Thus $\vdash \psi\theta x{:}\,\psi P x$ since $\vdash \psi P$. If $x \in \mathcal{D}\theta$, then $P \vdash \theta x{:}\,\sigma$ for some σ and hence $\vdash \psi\theta x{:}\,\psi\sigma$ since $\vdash \psi P$.

Hence there exists an $\mathcal{I}(T)$-assignment δ such that $\delta x = \pi\psi\theta x$ for every $x \in \mathcal{D}P \cup \mathcal{D}\theta$ and $\delta\alpha = \pi\psi\theta\alpha$ for every sort variable. Since $V \subseteq \mathcal{D}P \cup \mathcal{D}\theta$, we know that δ agrees with $\pi\psi\theta$ on V. Furthermore, δ and $\pi\psi\theta$ agree on every variable occurring in $\mathrm{E}[\theta] \,\&\, P$. Hence we know by the Unifier Proposition that it suffices to show that $(\pi\psi\theta)_\emptyset \in \mathrm{U}[\mathrm{E}[\theta] \,\&\, P]$.

Since $\vdash \pi\psi P$ and $\mathcal{D}\theta$ and $\mathcal{V}P$ are disjoint, we know $\vdash \pi\psi\theta P$. Since θ is idempotent, we know that $\pi\psi\theta = \pi\psi\theta\theta$ and hence that $\pi\psi\theta\mathrm{E}[\theta]$ is trivial. Furthermore, since we know that $\pi\psi\theta x$ is well-typed for every $x \in \mathcal{D}\theta$, we know that $\pi\psi\theta\mathrm{E}[\theta]$ is well-typed. □

The final theorem of this section tells us that we can obtain a V-solution schema for F from a principal unifier of F by throwing away redundant information.

Theorem 5.3.6 [Garbage Collection] *Let θ_P be a principal unifier of $E \,\&\, M$ and $W = (V - \mathcal{D}\theta) \cup \mathcal{I}(\theta|_V)$. Then $(\theta|_V)_{P|_W}$ is a V-solution schema for $E \,\&\, M$.*

Proof. It is easy to verify that $(\theta|_V)_{P|_W}$ is a V-solution schema. Since θ_P be a principal unifier of $E \,\&\, M$, we know $\mathrm{U}[\mathrm{E}[\theta] \,\&\, P] = \mathrm{U}[E \,\&\, M]$ and hence $\mathcal{I}(T)[\![\mathrm{E}[\theta] \,\&\, P]\!]^V = \mathcal{I}(T)[\![E \,\&\, M]\!]^V$ by the Unifier Proposition. Thus it suffices to show that $\mathcal{I}(T)[\![\mathrm{E}[\theta] \,\&\, P]\!]^V = \mathcal{I}(T)[\![\mathrm{E}[\theta|_V] \,\&\, P|_W]\!]^V$.

"$\subseteq$". Since $\mathrm{E}[\theta|_V] \,\&\, P|_W \subseteq \mathrm{E}[\theta] \,\&\, P$, we know that $\mathcal{I}(T)[\![\mathrm{E}[\theta] \,\&\, P]\!]^V \subseteq \mathcal{I}(T)[\![\mathrm{E}[\theta|_V] \,\&\, P|_W]\!]^V$.

"$\supseteq$". Let $\delta \in \mathcal{I}(T)[\![\mathrm{E}[\theta|_V] \,\&\, P|_W]\!]$. We have to show that there exists a $\delta' \in \mathcal{I}(T)[\![\mathrm{E}[\theta] \,\&\, P]\!]$ that agree with δ on V. By the preceding proposition we know that we can assume without loss of generality that θ_δ is inhabited. Let $\psi := \theta_\delta$. Then we know that $\vdash \psi(P|_W)$ and that $\psi x = \psi\theta x$ for every $x \in V$. Since θ_P is a principal unifier we also know that $P \vdash \theta{:}\,P$.

Let $X := \mathcal{D}P - (\mathcal{D}\theta \cup W)$ and $Y := \mathcal{D}\theta - V$. Then we know that $\mathcal{D}P \subseteq \mathcal{D}\theta \cup W \cup X$ and that $\mathcal{D}\theta$, W and X are pairwise disjoint. Since P and ψ are inhabited, we can choose a

value substitution ω such that $\mathcal{D}\omega = X$ and $\vdash \omega x\colon \psi P x$. Since ωx is ground for $x \in X$, we know that $\vdash \psi\omega(P|_X)$.

Now let $\phi := \psi\omega\theta$. Then $\phi\alpha = \psi\alpha$ for every sort variable α. Furthermore, if $x \notin X \cup Y$, then $\phi x = \psi x$. To see this, suppose $x \notin X \cup Y$. If $x \notin \mathcal{D}P$, then $\phi x = \psi\omega\theta x = \psi\omega x = \psi x$ since $x \notin X = \mathcal{D}\omega$. If $x \in \mathcal{D}\theta$, then $x \in \mathcal{D}\theta \cap V$ since $x \notin Y$. Hence $\mathcal{V}\theta x \subseteq \mathcal{I}(\theta|_V)$ is disjoint with $X = \mathcal{D}\omega$. Hence $\phi x = \psi\omega\theta x = \psi\theta x = \psi x$ since $x \in \mathcal{D}\theta \cap V$.

Since V, X and Y are pairwise distinct, we know that ϕ agrees with δ on V. Furthermore, ϕ maps every sort variable to a ground sort term and every value variable to a well-typed ground value term. Thus ϕ extends a $\mathcal{I}(T)$-assignment. Hence it suffices to show that $\phi_\emptyset \in \mathrm{U}[\mathrm{E}[\theta] \,\&\, P]$.

First we show that $\vdash \phi P$. Since we know $\vdash \psi(P|_W)$ and $\mathcal{D}\omega = X$ is disjoint with W, we have $\vdash \psi\omega(P|_W)$. Since we know $\vdash \psi\omega(P|_X)$, we have $\vdash \psi\omega(P|_{X\cup W})$. Since we know $P \vdash \theta P$ and $\mathcal{I}\theta \subseteq X \cup W$, we have $P|_{X\cup W} \vdash \theta(P|_{\mathcal{D}\theta})$. Hence we know $\vdash \psi\omega\theta(P|_{\mathcal{D}\theta})$. Thus $\vdash \phi(P|_{\mathcal{D}\theta})$. Since $\vdash \psi(P|_W)$ and W and $\mathcal{D}\theta \cup \mathcal{D}\omega$ are disjoint, we know that $\vdash \psi\omega\theta(P|_W)$ and hence $\vdash \phi(P|_W)$. Since we know that $\vdash \psi\omega(P|_X)$ and ωx is ground for every $x \in X = \mathcal{D}\omega$, we know that $\vdash \psi\omega\theta(P|_X)$ and hence $\vdash \phi(P|_X)$. Thus $\vdash \phi P$ since $\mathcal{D}P \subseteq \mathcal{D}\theta \cup W \cup X$.

To show that $\phi\mathrm{E}[\theta]$ is trivial, it suffices to show that $\phi = \phi\theta$. This is the case since θ is idempotent and hence $\phi = \psi\omega\theta = \psi\omega\theta\theta = \phi\theta$.

To show that $\phi\mathrm{E}[\theta]$ is well-typed, it suffices to show that ϕx is well-typed for every $x \in \mathcal{D}\theta$. This is the case since $\mathcal{D}\theta \subseteq \mathcal{D}P$ and we have already shown that $\vdash \phi P$. □

Proposition 5.3.7 *If π is a sort substitution, then $\mathcal{T}[\![E \,\&\, \pi M]\!]^V \subseteq \mathcal{T}[\![E \,\&\, M]\!]^V$.*

Proof. Let $\delta \in \mathcal{T}[\![E \,\&\, \pi M]\!]$ and define δ' as follows: $\delta'(x) = \delta(x)$ for every value variable x and $\delta'(\alpha) = \mathcal{T}[\![\pi\alpha]\!]_\delta$ for every sort variable α. Then we know by Lemma 3.3.2 that $\delta' \in \mathcal{T}[\![E \,\&\, M]\!]$. Since δ and δ' agree on all value variables, we have the claim. □

Chapter 6

POS-Constraint Solving

6.1 The Constraint Solver
6.2 Well-Typedness in Pyramids
6.3 Approximations
6.4 P-Infima and Mergings
6.5 Solving PM-Systems
6.6 Solving PE-Systems
6.7 Proof of the Hauptsatz

We have now arrived at the heart of this thesis. This chapter presents the constraint solver to be employed in the interpreter for relational programs over POS-types. The constraint solver must solve constraints of the form $P \,\&\, E$, where E is typically not well-typed under P. This is in sharp contrast to logic programming over order-sorted types, where the constraints produced by the interpreter are always well-typed.

Suppose the program in Figure 1.1 is executed with the well-typed query

$$\mathsf{L}\colon \mathsf{list}(\mathsf{int}) \,\&\, \mathsf{append}(\mathsf{nil}, \mathsf{cons}(\mathsf{p}(\mathsf{o}), \mathsf{nil}), \mathsf{L}).$$

Then reduction with the first clause of append yields the constraint

$$\mathsf{L}\colon \mathsf{list}(\mathsf{int}) \,\&\, \mathsf{L}'\colon \mathsf{list}(\alpha) \,\&\, \mathsf{nil} \doteq \mathsf{nil} \,\&\, \mathsf{cons}(\mathsf{p}(\mathsf{o}), \mathsf{nil}) \doteq \mathsf{L}' \,\&\, \mathsf{L} \doteq \mathsf{L}'$$

whose second and third equation are not well-typed under the prefix of the constraint. Clearly, the ill-typedness is caused by the fact that the "right" instantiation of the sort variable α is missing.

Thus a constraint solver for logic programming over POS-types must deal with a further problem that doesn't show up at all with order-sorted types. Understanding and solving

this problem was the major difficulty in successfully finishing the research reported in this thesis.

It is clear that the constraints produced by the interpreter are not completely ill-typed but do have a strong structural property. What made things difficult is that the initial structure of the constraint is not preserved by the constraint solving rules. Thus what I had to find was a sufficiently strong invariant that is preserved by the constraint solving rules, and this invariant turned out to be complex. Of course, now that we have the invariant things are simple again.

The algorithm I will present in this chapter comes with several powerful optimizations exploiting the structure asserted by the invariant. In particular, it solves a problem that has been bugging me for quite some time: if an order-sorted unification algorithm is applied to a many-sorted equation system, all the sort-related computations it performs are redundant and are thus wasted time. A unification algorithm for POS-equation systems is a lot more general than an order-sorted algorithm and thus the overhead for unnecessary sort computations increases dramatically. Now the rule of the game in logic programming is speed, and if one looks at practical applications there are only a few (but important) places where nonredundant sort computations are necessary. So what we need to arrive at a practical theory of logic programming over POS-types is a constraint solver that does only the nonredundant sort computations and doesn't spend any time with unnecessary sort computations.

Fortunately, the problem does have a beautiful solution. The key idea is to not compute with the actual sort terms but with approximations that are obtained from the sort terms in the program and the query at compile-time. The approximations are obtained by replacing sort terms that are maximal with respect to the inclusion order with a wildcard symbol that can be thought of as a "no operation" code. For instance, if

$$\mathsf{pair}(\mathsf{int}, \mathsf{list}(\mathsf{string}))$$

is maximal, we can just replace it with the wildcard symbol '$_$'. Moreover, if we have the sort term

$$\mathsf{pair}(\mathsf{nat}, \mathsf{list}(\mathsf{string})),$$

where nat is the only nonmaximal sort symbol involved, we can still replace it with the approximation

$$\mathsf{pair}(\mathsf{nat}, _).$$

Once the computation is finished, we are nevertheless able to print the exact answers since there exists a retract function computing the actual sort terms from their approximations and the initial sorts of the variables we were solving for.

The first section of this chapter presents the complete algorithm and states its properties in the Hauptsatz—the main result of this thesis. The following six sections explain why the constraint solver works by stepwise developing the proof of the Hauptsatz.

6.1 The Constraint Solver

This section presents the constraint solving algorithm. If you just want to know *how* the algorithm works (for instance, if you want to implement it), this section will suffice and you can skip the rest of this chapter. If, however, you want to know *why* the algorithm works, you will have to work yourself through the remaining six sections of this chapter.

In this section we devise enough notation to state our main result, the so-called Hauptsatz, which gives the precise precondition under which the constraint solver is correct. The Hauptsatz and a few results of Section 6.2 is all that is needed for the proofs of the next chapter, where we will finally give an operational semantics for relational programs over POS-types.

6.1.1 Approximations

For many-sorted unification one can just take an unsorted unification algorithm while for order-sorted unification one needs a more complex algorithm computing with sorts. One, of course, can take the order-sorted unification algorithm as well for doing many-sorted unification, but only at the price of paying the overhead for the then unnecessary sort computations. This situation remains unchanged if we have sort functions: as long as there are no inclusional axioms, any unsorted unification algorithm will do the job (see [MO84]). But in the presence of sort functions and subsorts computations with sorts are more complex and hence the overhead of a general algorithm with respect to an unsorted algorithm increases. So wouldn't it be nice if we had an algorithm that automatically adapts to the unification problem to be solved and always just does the work that is absolutely necessary?

Now, the constraint solver presented here is such an automatically adapting algorithm. The adaptation is accomplished by a compilation replacing every sort term in the program and the query with an approximation containing just the nonredundant information. Every sort term that is maximal with respect to the inclusion order is replaced with the wildcard symbol saying that no sort related computation is necessary. For instance, if

$$\mathsf{pair}(\mathsf{int}, \mathsf{list}(\mathsf{string}))$$

is maximal, we can just replace it with the wildcard symbol '_'. Moreover, if we have the sort term

$$\mathsf{pair}(\mathsf{nat}, \mathsf{list}(\mathsf{string})),$$

where nat is the only nonmaximal sort symbol involved, we can still replace it with the approximation

$$\mathsf{pair}(\mathsf{nat}, _).$$

Once the computation is finished, we are nevertheless able to print the exact answers since there exists a retract function computing the actual sort terms from their approximations and the initial sorts of the variables we were solving for.

Now that you have got the idea, let's start with the technical definitions.

A sort function symbol is called **maximal** if it is maximal with respect to the partial order "$\xi \leq \eta$". A sort term is called **maximal** if it is maximal with respect to the partial order "$\sigma \leq \tau$".

From now on we assume that the symbol '$_$', called **wildcard**, is a sort variable. Defining wildcard as a sort variable is technically convenient, but, be warned, rather than behaving like a sort variable, wildcard behaves like a maximal sort having all values as elements.

A sort term is called an **approximation** if it is ground or if it contains no variable but possibly the wildcard symbol. An approximation is called a **weakest approximation** if it is a normal sort term and contains no maximal subterm but the wildcard symbol. The constraint solver will only compute with weakest approximations. Keep in mind that approximations are syntax and that their denotations aren't meaningful.

The **weakest approximation** $\downarrow\sigma$ of a sort term σ can be computed as follows:

$$\begin{aligned} \downarrow\alpha &= _ \\ \downarrow\xi(\vec{\sigma}) &= \begin{cases} _ & \text{if } \xi \text{ is maximal and } \downarrow\vec{\sigma} = (_, \ldots, _) \\ _ & \text{if } \xi(\downarrow\vec{\sigma}) \text{ is top-level void} \\ \xi(\downarrow\vec{\sigma}) & \text{otherwise.} \end{cases} \end{aligned}$$

It is easy to verify that the weakest approximation of a sort term is in fact a weakest approximation. The **weakest approximation** $\downarrow P$ of a prefix P is defined as

$$\downarrow P \;:=\; \{x{:}\downarrow Px \mid x \in \mathcal{D}P\}.$$

Replacing every sort term with its weakest approximation is the compilation step required by our constraint solver. Since this eliminates all occurring sort variables (except wildcard, but wildcard is just technically a sort variable), our constraint solver doesn't have to compute with sort variables.

Next we define the retract operation, which is the inverse of the weakest approximation operation and decompiles an approximation back into the actual sort term it stands for. The following equations define a computable partial function "$\sigma \uparrow \tau$" from sort terms to sort terms:

$$\begin{aligned} \alpha \uparrow \tau &= \tau \\ \xi(\vec{\sigma}) \uparrow \tau &= \xi(\vec{\sigma} \uparrow \vec{\tau}) \quad \text{if } \tau \Rightarrow^* \xi(\vec{\tau}). \end{aligned}$$

If $\sigma \uparrow \tau$ is defined, then $\sigma \uparrow \tau$ is called the **retract** of σ with respect to τ. The retract function is extended to prefixes as follows:

$$P \uparrow Q \;=\; \{x{:}(Px \uparrow Qx) \mid x \in \mathcal{D}P \cap \mathcal{D}Q\}.$$

In Section 6.3 we will show that

$$\sigma = (\downarrow\sigma) \uparrow \tau$$

if σ is normal and $\sigma \leq \tau$.

If the constraint solver computed with the actual sort terms, the main sort-related operation would be computing $\mathrm{NF}[\sigma \sqcap \tau]$ from σ and τ. The following equations define a computable

total function "$\sigma \downarrow \tau$" from approximations to approximations simulating this operation:

$$
\begin{aligned}
_ \downarrow \tau &= \tau \\
\sigma \downarrow _ &= \sigma \\
\xi(\vec{\sigma}) \downarrow \eta(\vec{\tau}) &= \begin{cases} - & \text{if } \zeta(\vec{\mu}) \text{ is top-level void} \\ \zeta(\vec{\mu}) & \text{otherwise} \end{cases} \\
&\quad \text{if } \zeta = \xi \sqcap \eta,\ \xi(\vec{\sigma}) \Rightarrow^* \zeta(\vec{\sigma}_1),\ \eta(\vec{\tau}) \Rightarrow^* \zeta(\vec{\tau}_1), \text{ and} \\
&\quad\ \vec{\mu} = (\downarrow\vec{\sigma}_1) \downarrow (\downarrow\vec{\tau}_1).
\end{aligned}
$$

If $\sigma \downarrow \tau$ is defined, then $\sigma \downarrow \tau$ is called the **merging** of σ and τ.

For reasons of efficiency the merging operation integrates the infimum computation with the reduction to normal form.

In Section 6.4 we will show that the merging of two weakest approximations is again a weakest approximation.

6.1.2 The Constraint Solving Rules

The constraint solver is organized in two levels. The equation level consists of the usual unification rules for equations and calls the sort level only if it attempts to bind a variable x to a term s. The sort level deals with memberships and reduces the membership $s\colon Px$ consisting of s and the current sort qualification of x under the current sort qualifications of the variables in s. For instance, if we take the type specification of Figure 1.1 and have $Px = \mathsf{list}(\mathsf{posint})$ and $s = \mathsf{cons}(y, z)$, we may have to solve the constraint

$$y\colon \mathsf{nat} \,\&\, z\colon \mathsf{list}(\mathsf{negint}) \,\&\, \mathsf{cons}(y, z)\colon \mathsf{list}(\mathsf{posint}),$$

which reduces to the satisfiable constraint

$$y\colon \mathsf{posint} \,\&\, z\colon \mathsf{list}(-).$$

Let's start with the reduction rules for memberships. The following rules define a decidable binary relation $\xrightarrow{\mathrm{m}}$ on pairs $P.M$ consisting of a prefix and a membership system:

1. $P.s\colon _ \,\&\, M \xrightarrow{\mathrm{m}} P.M$
2. $P \,\&\, x\colon\sigma.x\colon\tau \,\&\, M \xrightarrow{\mathrm{m}} P \,\&\, x\colon(\sigma \downarrow \tau).M$
   if $\sigma \downarrow \tau \neq -$ and $\tau \neq _$
3. $P.f(\vec{s})\colon\sigma \,\&\, M \xrightarrow{\mathrm{m}} P.M$
   if $f\colon\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$, and $\vec{\mu}$ is ground
4. $P.f(\vec{s})\colon\sigma \,\&\, M \xrightarrow{\mathrm{m}} P.\vec{s}\colon\downarrow(\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}) \,\&\, M$
   if $f\colon\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$, and $\vec{\mu}$ is not ground.

Initially, the pair to be solved has the form $P.s\colon Px$, where the prefix P represents the current sort qualifications of the variables and $s\colon Px$ is the membership that must be satisfied to validate the binding of x to s. The first rule says that a membership $s\colon _$ is redundant

and can be thrown away. Hence, if the sort qualification of x is wildcard, no sort-related action needs to be taken to validate the binding x/s. The second rule reduces a constraint $x{:}\sigma \,\&\, x{:}\tau$ to $x{:}(\sigma \downarrow \tau)$, provided the actual sort terms σ and τ stand for have a nonempty intersection. The third and the fourth rule solve nontrivial memberships. The third rule is an important optimization for monomorphic value functions, for which it suffices to check that the declared codomain satisfies the required sort. Only for polymorphic functions it is necessary to actually decompose the membership and solve the memberships consisting of the argument terms and the instantiated sort terms declared for the arguments. The constraint in the example above can be reduced by first applying the polymorphic decomposition rule and then applying the merging rule twice.

Proposition 6.1.1 *There are no infinite chains $P.M \xrightarrow{\mathrm{m}} P'.M' \xrightarrow{\mathrm{m}} \cdots$. Furthermore, if $P.M \xrightarrow{\mathrm{m}} P'.M'$ and every sort term in $P.M$ is a weakest approximation, then every sort term in $P'.M'$ is a weakest approximation.*

A pair $P.M$ is called **don't care** if $P.M \xrightarrow{\mathrm{m}}{}^{*} Q.\emptyset$ and $P.M \xrightarrow{\mathrm{m}}{}^{*} P'.M'$ always implies $P'.M' \xrightarrow{\mathrm{m}}{}^{*} Q.\emptyset$. If $P.M$ is don't care, then the choice which $\xrightarrow{\mathrm{m}}$-rule is applied where is don't care nondeterministic. The equation level of the constraint solver will call the sort level only with don't care pairs.

Next let's give the equation level of the constraint solver, which consists of a suitable variant of the ordinary unification rules [MM82], where the variable binding rule is augmented with the already discussed hook to the sort level. The following rules define a computable binary relation $\xrightarrow{\mathrm{e}}$ on triples $P.\theta.E$:

1. $P.\theta.E \,\&\, x \doteq x \;\xrightarrow{\mathrm{e}}\; P.\theta.E$
2. $P.\theta.E \,\&\, f(\vec{s}) \doteq f(\vec{t}) \;\xrightarrow{\mathrm{e}}\; P.\theta.E \,\&\, \vec{s} \doteq \vec{t}$
3. $P.\theta.E \,\&\, s \doteq x \;\xrightarrow{\mathrm{e}}\; P.\theta.E \,\&\, x \doteq s$
   if s is no variable
4. $P.\theta.E \,\&\, x \doteq s \;\xrightarrow{\mathrm{e}}\; Q.\{x/s\}\theta.\{x/s\}E$
   if $x \notin \mathcal{V}s$ and $P.s{:}Px \xrightarrow{\mathrm{m}}{}^{*} Q.\emptyset$.

A triple $P.\theta.E$ consists of a prefix P representing the current sort qualifications of the variables, a substitution θ representing the variable bindings already made, and an equation system E still to be solved. To solve a constraint $P \,\&\, E$, the constraint solver attempts to reduce the triple $\downarrow P.\emptyset.E$. The reductions succeeds if a "solved" triple $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$ can be obtained, and this will be the case if and only if $P \,\&\, E$ is satisfiable. The Hauptsatz to be stated soon will give the precise retract function (recall, the solver starts with the weakest approximation of P and hence the obtained sort qualifications represented by Q will only be approximations) and the precondition on $P \,\&\, E$ under which the constraint solver is correct. As you will expect, the application of the equation reduction rules is don't care nondeterministic.

Proposition 6.1.2 *There are no infinite chains $P.\theta.E \xrightarrow{\mathrm{e}} P'.\theta'.E' \xrightarrow{\mathrm{e}} \cdots$. Furthermore, if $P.\emptyset.E \xrightarrow{\mathrm{e}} Q.\theta.E'$, then*

1. $\mathcal{D}Q = \mathcal{D}P$ *and* $\mathcal{D}\theta \cup \mathcal{I}\theta \cup \mathcal{V}E' \subseteq \mathcal{V}E$
2. θ *is idempotent and* $\mathcal{D}\theta$ *and* $\mathcal{V}E'$ *are disjoint*
3. *if every sort term in* P *is a weakest approximation, then every sort term in* Q *is a weakest approximation.*

Proof. Suppose there is an infinite chain $P.\theta.E \xrightarrow{e} P'.\theta'.E' \xrightarrow{e} \cdots$. Since no rule increases the number of variables in E, this number is decreased by the fourth rule, the first and the second rules reduce the size of E, and the third rule does not change the size of E, there must be an infinite chain employing only the third rule, which is impossible.

The other claims are easy to verify. □

6.1.3 Pyramids

By now you know the algorithm. What you don't know is the precondition under which it works. From the reduction rules for memberships it's clear that there must be some strong invariant since otherwise the wildcard rule and the monomorphic decomposition rule couldn't possibly work.

The precondition is delicate and it took me many absent-minded walks around a certain block in Stuttgart until I came up with it. A sufficient precondition is to require of the constraint $P \,\&\, E$ to be solved that E is well-typed under P, but this is too strong a condition to be satisfied by the constraints produced by the interpreter for POS-programs. To see this, recall the program in Figure 1.1 and suppose we want to reduce the query

$$\mathsf{L}\colon \mathsf{list}(\mathsf{int}) \,\&\, \mathsf{append}(\mathsf{cons}(\mathsf{o}, \mathsf{nil}), \mathsf{cons}(\mathsf{p}(\mathsf{o}), \mathsf{nil}), \mathsf{L}).$$

A first reduction step with the second clause of **append** yields the goal

$$\begin{array}{l}\mathsf{L}\colon \mathsf{list}(\mathsf{int}) \,\&\, \mathsf{H}\colon \alpha \,\&\, \mathsf{R}\colon \mathsf{list}(\alpha) \,\&\, \mathsf{L'}\colon \mathsf{list}(\alpha) \,\&\, \mathsf{RL}\colon \mathsf{list}(\alpha) \,\& \\ \mathsf{cons}(\mathsf{o}, \mathsf{nil}) \doteq \mathsf{cons}(\mathsf{H}, \mathsf{R}) \,\&\, \mathsf{cons}(\mathsf{p}(\mathsf{o}), \mathsf{nil}) \doteq \mathsf{L'} \,\&\, \mathsf{L} \doteq \mathsf{cons}(\mathsf{H}, \mathsf{RL}) \,\& \\ \mathsf{append}(\mathsf{R}, \mathsf{L'}, \mathsf{RL})\end{array}$$

whose equations are certainly not well-typed under the sort qualifications of the goal. Things get more complicated if we reduce again, this time with the first clause of **append**:

$$\begin{array}{l}\mathsf{L}\colon \mathsf{list}(\mathsf{int}) \,\&\, \mathsf{H}\colon \alpha \,\&\, \mathsf{R}\colon \mathsf{list}(\alpha) \,\&\, \mathsf{L'}\colon \mathsf{list}(\alpha) \,\&\, \mathsf{RL}\colon \mathsf{list}(\alpha) \,\&\, \mathsf{L''}\colon \mathsf{list}(\beta) \,\& \\ \mathsf{cons}(\mathsf{o}, \mathsf{nil}) \doteq \mathsf{cons}(\mathsf{H}, \mathsf{R}) \,\&\, \mathsf{cons}(\mathsf{p}(\mathsf{o}), \mathsf{nil}) \doteq \mathsf{L'} \,\&\, \mathsf{L} \doteq \mathsf{cons}(\mathsf{H}, \mathsf{RL}) \,\& \\ \mathsf{R} \doteq \mathsf{nil} \,\&\, \mathsf{L'} \doteq \mathsf{L''} \,\&\, \mathsf{RL} \doteq \mathsf{L''}.\end{array}$$

From this example we can see that the constraints produced by the interpreter are well-typed modulo application of certain sort substitutions existing for every reduction step. In the example above, $\{\alpha/\mathsf{int}\}$ is a suitable substitution for the first reduction step, and $\{\beta/\alpha\}$ is a suitable substitution for the second reduction step. For the correctness proof to go through it will be necessary to have the precise structure of the supporting sequence of sort substitutions available, which will be called a *pyramid*. The intuition is that a pyramid is created according to a stack discipline: each reduction step pushes a new layer on top

and extends the existing layers by right hand composition with the new layer. (This is a little bit simplified, but Chapter 7 will tell you all the glory details.)

My argumentation indicates that the correctness proof will take the viewpoint that the goal is first reduced using only goal reduction. Only after all atoms have been eliminated, the solution of the piled up constraint system is attempted, where the constraint system is well-typed modulo the simultaneously obtained pyramid. Of course, in practice such a strategy would be disastrous since unsatisfiable parts of the search space couldn't be pruned early. Fortunately, the structure of the constraint solving rules is incremental enough to allow incremental constraint solving just in the same way it is done in ordinary Prolog.

Let's now start with the formal definitions. It will be tough to get all this stuff into your head, but the proof of the Hauptsatz doesn't come for less. A good strategy might be to only skim the following definitions (which I will talk you through in Section 6.2), look carefully at the Hauptsatz, and then go immediately to Chapter 7. Once you have read Chapter 7, you will know how the constraint solver is integrated with the rest and that everything works out well. Then you should be well-prepared for the guts of this Chapter, which look more intimidating than they actually are. Once you can properly visualize pyramids and the corresponding notion of well-typedness in your mind, everything will turn out to be rather simple.

A **pyramid** is a finite sequence $\Pi = \pi_0\pi_1\cdots\pi_n$ of inhabited sort substitutions such that $n \geq 0$ and

1. $\pi_0 = \emptyset$
2. $\pi_i = \pi_i\pi_j$ if $n \geq i \geq j \geq 0$
3. $\mathcal{D}\pi_i \cup \mathcal{I}\pi_i \subseteq \mathcal{D}\pi_{i+1}$ if $0 \leq i \leq n-1$.

If $\Pi = \pi_0\pi_1\cdots\pi_n$ is a pyramid, we will often use π to denote π_n. We call n the **height** of Π. The pyramid of height 0 consists only of the empty substitution and is called the **trivial pyramid**.

Let SV be the set of all sort variables. A pyramid $\Pi = \pi_0\pi_1\cdots\pi_n$ defines a partition $\mathrm{SV} = \mathrm{SV}_0 \uplus \cdots \uplus \mathrm{SV}_n$ as follows:

$$\begin{aligned} \mathrm{SV}_i &:= \mathcal{D}\pi_{i+1} - \mathcal{D}\pi_i \quad \text{if } i \in 0..n-1 \\ \mathrm{SV}_n &:= \mathrm{SV} - \mathcal{D}\pi_n. \end{aligned}$$

Proposition 6.1.3 *Let $\Pi = \pi_0\pi_1\cdots\pi_n$ be a pyramid. Then:*

1. $\mathrm{SV} = \mathrm{SV}_0 \uplus \cdots \uplus \mathrm{SV}_n$
2. $\mathcal{D}\pi_i = \mathrm{SV}_0 \uplus \cdots \uplus \mathrm{SV}_{i-1}$ *if* $i \in 1..n$
3. $\mathcal{I}\pi_i \subseteq \mathrm{SV}_i$ *if* $i \in 0..n$
4. $\pi_i\pi_j = \pi_j\pi_i$ *if* $i, j \in 0..n$.

In the following we assume that $\Pi = \pi_0\pi_1 \cdots \pi_n$ is a pyramid.

If α is a sort variable, we say that α has **level** i and write $\lambda\alpha = i$ if $\alpha \in \mathrm{SV}_i$.

A sort term is called Π**-admissible** if all sort variables occurring in it have the same level with respect to Π. A prefix is called Π**-admissible** if every sort term occurring in it is Π-admissible.

The **level** $\lambda\sigma$ of a Π-admissible sort term is defined as follows:

$$\lambda\sigma = \begin{cases} n & \text{if } \sigma \text{ is ground} \\ i & \text{if } \sigma \text{ is not ground and } \forall\alpha \in \mathcal{V}\sigma.\ \lambda\alpha = i. \end{cases}$$

Proposition 6.1.4 *Π-admissible sort terms have the following properties:*

1. *if σ is Π-admissible, then $\pi_i\sigma$ is Π-admissible for every $i \in 0..n$*
2. *if $\sigma \leq \tau$ and τ is Π-admissible, then σ is Π-admissible, $\lambda\sigma \geq \lambda\tau$, and $\lambda\sigma = \lambda\tau$ if σ contains a variable*
3. *if σ is Π-admissible and $i \leq \lambda\sigma$, then $\pi_i\sigma = \sigma$*
4. *if σ is Π-admissible, then $\lambda\pi_i\sigma = \max\{i, \lambda\sigma\}$.*

If P is a Π-admissible prefix and s is a value term such that $\mathcal{V}s \subseteq \mathcal{D}P$, then the **level** $\lambda^P s$ of s with respect to P is defined as

$$\lambda^P s := \begin{cases} n & \text{if } s \text{ is ground} \\ \min\{\lambda Px \mid x \in \mathcal{V}s\} & \text{otherwise.} \end{cases}$$

We now generalize ordinary well-typedness to *well-typedness modulo pyramids*. The following rules define inductively a relation "$P \vdash^\Pi s{:}\sigma$" taking a Π-admissible prefix P, a value term s and a Π-admissible sort term σ as arguments:

1. $P \vdash^\Pi x{:}\sigma$ if $Px \leq \pi_{\lambda Px}\sigma$
2. $P \vdash^\Pi s{:}\alpha$ if s is no variable, $\lambda^P s > \lambda\alpha$, and $P \vdash^\Pi s{:}\pi_{\lambda^P s}\alpha$
3. $P \vdash^\Pi f(\vec{s}){:}\sigma$ if $f{:}\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^\Pi \vec{s}{:}\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$.

This definition will be discussed in Section 6.2.

If Π is the trivial pyramid, then $P \vdash s{:}\sigma$ if and only if $P \vdash^\Pi s{:}\sigma$. Thus well-typedness modulo pyramids generalizes ordinary POS-well-typedness.

6.1.4 The Hauptsatz

First we state the precise precondition under which the constraint solver is correct.

A constraint $P \,\&\, E$ is called V**-admissible** if there exists a pyramid Π such that

1. P is a normal, inhabited and Π-admissible prefix

2. E is an equation system such that for every equation $s \doteq t \in E$ there exists a Π-admissible sort term σ such that $P \vdash^{\Pi} s{:}\,\sigma$ and $P \vdash^{\Pi} t{:}\,\sigma$

3. $V \subseteq \mathcal{D}P$ and $Px = \pi Px$ for every variable $x \in V$.

Proposition 6.1.5 *Let P be a normal and inhabited prefix and let the equation system E be well-typed under P. Then $P \,\&\, E$ is $\mathcal{D}P$-admissible.*

Proof. Take the trivial pyramid. □

The following defines the final retract operation, where Q is the computed approximating prefix, ψ represents the computed variable bindings, and P is the prefix representing the initial sort qualifications:

$$\begin{aligned} \uparrow[Q, \psi, P] \;&:= \mathrm{E}[\psi|_{\mathcal{D}P}] \,\&\, (Q \uparrow \mathrm{GP}[\psi P]) \\ &\qquad \text{if } (\psi|_{\mathcal{D}P})_{(Q\uparrow \mathrm{GP}[\psi P])} \text{ is a } \mathcal{D}P\text{-solution schema.} \end{aligned}$$

Theorem 6.1.6 [Hauptsatz] *Let T be a fully inhabited type specification and let $P \,\&\, E$ be V-admissible. Then:*

1. *$P \,\&\, E$ is satisfiable in $\mathcal{T}$ if and only if there exist Q and ψ such that $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$*

2. *if $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$, then $\mathcal{T}[\![P \,\&\, E]\!]^V = \mathcal{T}[\![\uparrow[Q, \psi, P|_V]]\!]^V$*

3. *if $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\theta.E' \,\&\, x \doteq s$, then $Q.s{:}\,Qx$ is don't care*

4. *if $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\theta.\emptyset$ and $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q'.\theta'.E'$, then there exist Q'' and θ'' such that $Q'.\theta'.E' \xrightarrow{\mathrm{e}}{}^{*} Q''.\theta''.\emptyset$.*

The first two statements say that the constraint solver does what it is supposed to, and the next two statements say that all choices involved in the application of the constraint solving rules are don't care nondeterministic.

6.2 Well-Typedness in Pyramids

This section investigates the properties of the membership relation "$P \vdash^{\Pi} s{:}\,\sigma$". I will try to talk you through the definitions made in Subsection 6.1.3 and give you some of my visualizations.

General Assumption. *Here and in the rest of this chapter we assume that T is a type specification, $\Pi = \pi_0\pi_1 \cdots \pi_n$ is a pyramid, and that every occurring sort term is Π-admissible.*

Recall the a pyramid of height n partitions the set of sort variables into n classes. It is useful to think of this classes as classes of informedness. Sort variables of level n (the maximal level) are fully informed and sort variables of a lower level are partially informed.

Recall that a sort term is admissible if all sort variables occurring in it have the same level of information. All sort terms we need to consider in this chapter will be admissible. Hence we can say that a ground sort term is fully informed, and that a sort term containing sort variables is informed at the same level the variables occurring in it are informed.

Every layer π_i of the pyramid is an information providing function that maps all variables of an information level less than i to sort terms that are either ground or are informed at level i. Variables of level i or higher are not affected by π_i.

If σ is admissible and we apply the information layer π_i to σ, then there are two possibilities: if σ is already informed at level i or higher, then $\sigma = \pi_i\sigma$; otherwise, $\pi_i\sigma \neq \sigma$ and $\pi_i\sigma$ is either ground or informed at level i.

We now generalize the inclusion order to pyramids:

$$\sigma \leq^{\Pi} \tau \;\; :\Longleftrightarrow \;\; \sigma \leq \pi_{\lambda\sigma}\tau.$$

The idea is to tolerate that τ is less informed than σ. This is accomplished by informing τ up to the level of σ before testing the ordinary inclusion relation.

Proposition 6.2.1 *The relation "$\sigma \leq^{\Pi} \tau$" has the following properties:*

1. *if $\sigma \leq \pi_i\tau$, then $\sigma \leq^{\Pi} \tau$; in particular, if $\sigma \leq \tau$, then $\sigma \leq^{\Pi} \tau$*
2. *if $\sigma \leq^{\Pi} \tau$, then $\lambda\sigma \geq \lambda\tau$*
3. *"$\sigma \leq^{\Pi} \tau$" is a partial order on the set of all Π-admissible sort terms*
4. *(Orthogonality) if $\sigma \leq^{\Pi} \tau$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\tau \Rightarrow^* \xi(\vec{\tau})$, then $\vec{\sigma} \leq^{\Pi} \vec{\tau}$.*

Proof. *1.* Let $\sigma \leq \pi_i\tau$. Then $\lambda\sigma \geq i$ and hence $\sigma = \pi_{\lambda\sigma}\sigma \leq \pi_{\lambda\sigma}\pi_i\tau = \pi_{\lambda\sigma}\tau$.

2. Let $\sigma \leq \pi_{\lambda\sigma}\tau$. If σ is ground, then $\lambda\sigma = n \geq \lambda\tau$. If σ contains a variable, then $\lambda\sigma = \lambda\pi_{\lambda\sigma}\tau = \max\{\lambda\sigma, \lambda\tau\}$ and hence $\lambda\sigma \leq \lambda\tau$.

3. Since $\pi_{\lambda\sigma}\sigma = \sigma$, we know that "$\sigma \leq^{\Pi} \tau$" is reflexive. To show the antisymmetry, suppose $\sigma \leq^{\Pi} \tau$ and $\tau \leq^{\Pi} \sigma$. Then we know by statement (2) that $\lambda\sigma = \lambda\tau$. Hence $\sigma \leq \pi_{\lambda\sigma}\tau = \tau$ and $\tau \leq \pi_{\lambda\tau}\sigma = \sigma$ and thus $\sigma = \tau$ since "$\sigma \leq \tau$" is a partial order. To show the transitivity, suppose $\sigma \leq^{\Pi} \tau$ and $\tau \leq^{\Pi} \mu$. Hence $\lambda\sigma \geq \lambda\tau$ by statement (2) and hence $\sigma \leq \pi_{\lambda\sigma}\tau \leq \pi_{\lambda\sigma}\pi_{\lambda\tau}\mu = \pi_{\lambda\sigma}\mu$.

4. Let $\sigma \leq \pi_{\lambda\sigma}\tau$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\tau \Rightarrow^* \xi(\vec{\tau})$. Then $\vec{\sigma} \leq \pi_{\lambda\sigma}\vec{\tau}$. If a component σ_i of $\vec{\sigma}$ is ground, then $\lambda\sigma_i \geq \lambda\sigma$ and hence $\sigma_i = \pi_{\lambda\sigma_i}\sigma_i \leq \pi_{\lambda\sigma_i}\pi_{\lambda\sigma}\tau_i = \pi_{\lambda\sigma_i}\tau_i$. If a component σ_i of $\vec{\sigma}$ contains a variable, then $\lambda\sigma_i = \lambda\sigma$ and hence $\sigma_i \leq \pi_{\lambda\sigma_i}\tau_i$. □

Let P be a prefix. In Subsection 6.1.3 we have assigned to every value variable in $\mathcal{D}P$ the information level of its sort qualification Px. For a value term s that is pyramid well-typed with respect to P it's not possible to require that all value variables occurring in s are informed at the same level. Hence we have defined that s is fully informed if s is ground and that s is informed at the level of the least informed value variable occurring in it if s is not ground (this is the definition of $\lambda^P s$, remember?).

Now look again at the definition of the pyramid membership relation "$P \vdash^{\Pi} s{:}\,\sigma$" in Subsection 6.1.3. Analogously to the pyramid inclusion order "$\sigma \leq^{\Pi} \tau$", the relation "$P \vdash^{\Pi} s{:}\,\sigma$" tolerates that σ is less informed than P. Things will become clearer as we state and prove more properties.

Proposition 6.2.2 *If $P \vdash s{:}\,\sigma$, then $P \vdash^{\Pi} s{:}\,\sigma$.*

Proof. Let $P \vdash s{:}\,\sigma$. We prove $P \vdash^{\Pi} s{:}\,\sigma$ by induction on s.

If $s = x$, then $Px \leq \sigma$ and hence $Px \leq^{\Pi} \sigma$. Hence $P \vdash^{\Pi} s{:}\,\sigma$.

If $s = f(\vec{s})$, then we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Hence we know $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ by the induction hypothesis and thus $P \vdash^{\Pi} s{:}\,\sigma$. □

Proposition 6.2.3 *If $P \vdash^{\Pi} s{:}\,\sigma$, then $\pi P \vdash s{:}\,\pi\sigma$.*

Proof. Let $P \vdash^{\Pi} s{:}\,\sigma$. We prove $\pi P \vdash s{:}\,\pi\sigma$ by induction on the pair $(|s|, n - \lambda\sigma)$ with respect to the canonical lexicographic order, where $|s|$ is the size of s and n is the height of the pyramid Π.

Let $s = x$. Then $Px \leq \pi_{\lambda Px}\sigma$ and hence $\pi Px \leq \pi\sigma$. Hence $\pi P \vdash x{:}\,\pi\sigma$.

Let $s = f(\vec{s})$ and σ be a variable. Then $\lambda^P s > \lambda\sigma$ and $P \vdash^{\Pi} s{:}\,\pi_{\lambda^P s}\sigma$. Hence we have $\pi P \vdash s{:}\,\pi\pi_{\lambda^P s}\sigma$ by the induction hypothesis, which yields $\pi P \vdash s{:}\,\pi\sigma$ since $\pi\pi_{\lambda^P s}\sigma = \pi\sigma$.

Let $s = f(\vec{s})$ and σ not be a variable. Then we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Hence we know $\pi P \vdash^{\Pi} \vec{s}{:}\,\pi\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ by the induction hypothesis and thus $\pi P \vdash s{:}\,\pi\sigma$. □

Proposition 6.2.4 *If $P \vdash^{\Pi} s{:}\,\sigma$ and P is inhabited, then σ is inhabited.*

Proof. Let $P \vdash^{\Pi} s{:}\,\sigma$ and P be inhabited. By the preceding proposition we know that $\pi P \vdash s{:}\,\pi\sigma$. Since P and π are inhabited, πP is inhabited. Hence we know by Corollary 5.2.9 that $\pi\sigma$ and hence σ is inhabited. □

The Monotonicity Lemma says that $P \vdash^{\Pi} s{:}\,\sigma$ can only hold if σ is at most as informed as s with respect to P:

Lemma 6.2.5 [Monotonicity] *If $P \vdash^{\Pi} s{:}\,\sigma$, then $\lambda^P s \geq \lambda\sigma$.*

Proof. Let $P \vdash^{\Pi} s{:}\,\sigma$. We prove by induction on s that $\lambda^P s \geq \lambda\sigma$.

If $s = x$, then $Px \leq^{\Pi} \sigma$ and hence $\lambda^P s = \lambda Px \geq \lambda\sigma$.

Let $s = f(\vec{s})$. If $\sigma = \alpha$, then $\lambda^P s > \lambda\sigma$ by the definition of "$P \vdash^{\Pi} s{:}\,\sigma$". Otherwise, we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. By the induction hypothesis we have $\lambda^P s_i \geq \lambda\{\vec{\alpha}/\vec{\sigma}\}\mu_i$ for every component. Hence $\lambda^P s = \min_i \lambda^P s_i \geq \min_i \lambda\{\vec{\alpha}/\vec{\sigma}\}\mu_i = \lambda\sigma$ since $\mathcal{V}(\{\vec{\alpha}/\vec{\sigma}\}\mu_i) \subseteq \mathcal{V}\sigma$. □

The Flexibility Lemma, which will be used in many proofs to come, says that $P \vdash^{\Pi} s{:}\,\sigma$ remains valid if one withdraws information from σ or adds information to σ (only up to level $\lambda^P s$, of course).

Lemma 6.2.6 [Flexibility] *If $P \vdash^{\Pi} s{:}\,\pi_i\sigma$ and $j \leq \lambda^P s$, then $P \vdash^{\Pi} s{:}\,\pi_j\sigma$. Furthermore, if $P \vdash^{\Pi} s{:}\,\pi_i\sigma$, then $P \vdash^{\Pi} s{:}\,\sigma$.*

Proof. We first show the second claim using the first claim. Suppose $P \vdash^{\Pi} s{:}\,\pi_i\sigma$. Then $\lambda^P s \geq \lambda\pi_i\sigma \geq \lambda\sigma$ by the Monotonicity Lemma. Hence we know by the first claim that $P \vdash^{\Pi} s{:}\,\pi_{\lambda\sigma}\sigma$, which yields $P \vdash^{\Pi} s{:}\,\sigma$ since $\sigma = \pi_{\lambda\sigma}\sigma$.

To show the first claim, let $P \vdash^{\Pi} s{:}\,\pi_i\sigma$ and $j \leq \lambda^P s$. We prove $P \vdash^{\Pi} s{:}\,\pi_j\sigma$ by induction on the triple $(|s|, n-i, n-j)$ with respect to the canonical lexicographic order, where $|s|$ is the size of s and n is the height of the pyramid Π.

Let $s = x$. Then $Px \leq \pi_{\lambda Px}\pi_i\sigma$ and $\lambda Px \geq j$. Hence $\lambda Px \geq i$ and thus $Px \leq \pi_{\lambda Px}\pi_i\sigma = \pi_{\lambda Px}(\pi_j\sigma)$, which yields $P \vdash^{\Pi} x{:}\,\pi_j\sigma$.

Let $s = f(\vec{s})$ and $\pi_i\sigma$ be a variable. Then $\lambda^P s > \lambda\pi_i\sigma \geq i, \lambda\sigma$ and $P \vdash^{\Pi} s{:}\,\pi_{\lambda^P s}\sigma$. Hence $P \vdash^{\Pi} s{:}\,\pi_j\sigma$ by the induction hypothesis.

Let $s = f(\vec{s})$ and $\pi_i\sigma$ not be a variable. We distinguish two cases.

1. Let $\pi_j\sigma$ be a variable. Then we know $\lambda^P s \geq i \geq j$ by the Monotonicity Lemma and hence $P \vdash^{\Pi} s{:}\,\pi_{\lambda^P s}\sigma$ by the induction hypothesis. Hence $P \vdash^{\Pi} s{:}\,\pi_j\sigma$ by the definition of "$P \vdash^{\Pi} s{:}\,\sigma$".

2. Let $\pi_j\sigma$ be no variable. Then we distinguish once more two cases.

2.1. Let $i \leq j$. Then we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\pi_i\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Since $j \leq \lambda^P s = \min \lambda^P \vec{s}$, we have by the induction hypothesis that $P \vdash^{\Pi} \vec{s}{:}\,\pi_j\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Since $\pi_j\sigma = \pi_j(\pi_i\sigma) \Rightarrow^* \xi(\pi_j\vec{\sigma})$, we have $P \vdash^{\Pi} s{:}\,\pi_j\sigma$.

2.2. Let $j < i$. Then we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\pi_i\sigma \Rightarrow^* \xi(\pi_i\vec{\sigma})$, $\pi_j\sigma \Rightarrow^* \xi(\pi_j\vec{\sigma})$, and $P \vdash^{\Pi} \vec{s}{:}\,\pi_i\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Since $j < i \leq \lambda^P s = \min \lambda^P \vec{s}$, we have by the induction hypothesis that $P \vdash^{\Pi} \vec{s}{:}\,\pi_j\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ and hence $P \vdash^{\Pi} s{:}\,\pi_j\sigma$. □

The remaining three lemmas of this section show that the major properties of the ordinary membership relation are kind enough to generalize to the pyramid membership relation.

Lemma 6.2.7 [Upper Weakening] *If $P \vdash^{\Pi} s{:}\,\sigma$ and $\sigma \leq^{\Pi} \tau$, then $P \vdash^{\Pi} s{:}\,\tau$.*

Proof. Let $P \vdash^{\Pi} s{:}\,\sigma$ and $\sigma \leq \tau$. We prove by induction on s that $P \vdash^{\Pi} s{:}\,\tau$. Together with the Flexibility Lemma this yields the claim of the lemma.

If $s = x$, then $Px \leq^{\Pi} \sigma \leq^{\Pi} \tau$ and hence $P \vdash^{\Pi} s{:}\,\tau$.

Let $s = f(\vec{s})$. If σ is a variable, then $\sigma = \tau$, which yields the claim trivial. Otherwise, we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$, $\tau \Rightarrow^* \xi(\vec{\tau})$, and $\vec{\sigma} \leq \vec{\tau}$. Hence $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\tau}\}\vec{\mu}$ by the induction hypothesis and thus $P \vdash^{\Pi} s{:}\,\tau$. □

Lemma 6.2.8 [Normalization] *If $P \vdash^{\Pi} s{:}\,\sigma$ and P is normal, then $P \vdash^{\Pi} s{:}\,\mathrm{NF}[\sigma]$.*

Proof. Let $P \vdash^{\Pi} s{:}\,\sigma$ and let P be normal. We prove by induction on s that $P \vdash^{\Pi} s{:}\,\mathrm{NF}[\sigma]$.

If $s = x$, then $Px \leq \pi_{\lambda Px}\sigma$. Hence $Px = \mathrm{NF}[Px] \leq \mathrm{NF}[\pi_{\lambda Px}\sigma] \leq \pi_{\lambda Px}\mathrm{NF}[\sigma]$ and thus $P \vdash^{\Pi} x{:}\mathrm{NF}[\sigma]$.

If $s = f(\vec{s})$ and σ is a variable, then $\sigma = \mathrm{NF}[\sigma]$ and hence the claim is trivial.

If $s = f(\vec{s})$ and σ is not a variable, then we have $f{:}\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Hence we have $P \vdash^{\Pi} \vec{s}{:}\mathrm{NF}[\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}]$ by the induction hypothesis and thus $P \vdash^{\Pi} \vec{s}{:}\{\vec{\alpha}/\mathrm{NF}[\vec{\sigma}]\}\vec{\mu}$ by the Upper Weakening Lemma. Hence $P \vdash^{\Pi} s{:}\xi(\mathrm{NF}[\vec{\sigma}])$. Since P is inhabited, $\xi(\vec{\sigma})$ is inhabited and hence $\mathrm{NF}[\xi(\vec{\sigma})] = \xi(\mathrm{NF}[\vec{\sigma}])$. Since $\sigma \Rightarrow^* \xi(\vec{\sigma})$, we have $\xi(\mathrm{NF}[\vec{\sigma}]) = \mathrm{NF}[\xi(\vec{\sigma})] \leq \mathrm{NF}[\sigma]$. Hence $P \vdash^{\Pi} s{:}\mathrm{NF}[\sigma]$ by the Upper Weakening Lemma. □

We extend "$P \vdash^{\Pi} s{:}\sigma$" to membership systems as one would expect:

$$P \vdash^{\Pi} M \;:\Longleftrightarrow\; \forall\, s{:}\sigma \in M.\;\; P \vdash^{\Pi} s{:}\sigma.$$

Lemma 6.2.9 [Substitution] *Let θ be a value substitution, $Q \vdash^{\Pi} \theta P$ and $P \vdash^{\Pi} s{:}\sigma$. Then $Q \vdash^{\Pi} \theta s{:}\sigma$.*

Proof. We prove $Q \vdash^{\Pi} \theta s{:}\sigma$ by induction on the pair $(|s|, n - \lambda\sigma)$ with respect to the canonical lexicographic order, where $|s|$ is the size of s and n is the height of the pyramid Π.

Let $s = x$. Then $Px \leq^{\Pi} \sigma$ and $Q \vdash^{\Pi} \theta x{:}Px$. Hence $Q \vdash^{\Pi} \theta x{:}\sigma$ by the Upper Weakening Lemma.

Let $s = f(\vec{s})$ and σ be a variable. Then $\lambda^P s > \lambda\sigma$ and $P \vdash^{\Pi} s{:}\pi_{\lambda^P s}\sigma$. Hence we have $Q \vdash^{\Pi} \theta s{:}\pi_{\lambda^P s}\sigma$ by the induction hypothesis, which yields $Q \vdash^{\Pi} \theta s{:}\sigma$ with the Flexibility Lemma.

Let $s = f(\vec{s})$ and σ not be a variable. Then we have $f{:}\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Hence we know $Q \vdash^{\Pi} \theta\vec{s}{:}\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ by the induction hypothesis and thus $Q \vdash^{\Pi} \theta s{:}\sigma$. □

6.3 Approximations

General Assumption. *In this section we assume that T is fully inhabited.*

We now show that one can compute the weakest approximation $\downarrow\sigma$ of a sort term σ with a confluent and terminating rewrite system that contains the normalization rule of Section 5.2. The characterization of weakest approximations as normal forms of a convergent system of simplification rules will provide for pleasant proofs of their properties.

We write $\sigma \xrightarrow{\mathrm{a}} \tau$ if one of the following conditions holds:

1. τ can be obtained from σ by replacing a variable subterm $\alpha \neq _$ with $_$
2. τ can be obtained from σ by replacing a subterm $\xi(_, \ldots, _)$ such that ξ is maximal with $_$

3. τ can be obtained from σ by replacing a top-level void subterm $\mu \neq -$ with $-$.

Proposition 6.3.1 *The relation $\sigma \xrightarrow{\mathrm{a}} \tau$ is terminating and confluent.*

Proof. The termination follows from the fact that every reduction step reduces $k + l$, where k is the number of occurrences of sort function symbols different from $-$ and l is the number of occurrences of sort variables different from $_$. Furthermore, $\sigma \xrightarrow{\mathrm{a}} \tau$ is locally confluent since a replacement according to condition (2) cannot overlap with a replacement according to condition (3) because T is fully inhabited. Hence we know that $\sigma \xrightarrow{\mathrm{a}} \tau$ is confluent. □

Proposition 6.3.2 *A sort term τ is the normal form of a sort term σ with respect to $\xrightarrow{\mathrm{a}}$ if and only if $\tau = \downarrow\sigma$. Furthermore, the weakest approximations are exactly the normal forms of "$\sigma \xrightarrow{\mathrm{a}} \tau$".*

If θ and ψ are substitutions, we write $\theta \xrightarrow{\mathrm{a}}{}^* \psi$ if θ and ψ agree on every value variable and $\theta\alpha \xrightarrow{\mathrm{a}}{}^* \psi\alpha$ for every sort variable α.

The relation "$\sigma \xrightarrow{\mathrm{a}} \tau$" is extended to prefixes as follows:

$$P \xrightarrow{\mathrm{a}}{}^* Q \;:\Longleftrightarrow\; \mathcal{D}P = \mathcal{D}Q \;\wedge\; \forall\, x \in \mathcal{D}P.\;\; Px \xrightarrow{\mathrm{a}}{}^* Qx.$$

Note that "$P \xrightarrow{\mathrm{a}}{}^* Q$" is again terminating and confluent, and that the weakest approximation $\downarrow P$ defined in Section 6.1 is in fact the $\xrightarrow{\mathrm{a}}$-normal form of P.

Theorem 6.3.3 [Approximation] *The approximation relation "$\sigma \xrightarrow{\mathrm{a}} \tau$" has the following properties:*

1. *if $\sigma \xrightarrow{\mathrm{n}}{}^* \tau$, then $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$*
2. *if $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$, then σ is inhabited if and only if τ is inhabited*
3. *σ is void $\iff \downarrow\sigma = - \iff \mathrm{NF}[\sigma] = -$*
4. *σ is maximal $\iff \downarrow\sigma = _$*
5. *if $\theta \xrightarrow{\mathrm{a}}{}^* \psi$, then $\theta\sigma \xrightarrow{\mathrm{a}}{}^* \psi\sigma$*
6. *(Orthogonality) if $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\tau \Rightarrow^* \xi(\vec{\tau})$, then $\vec{\sigma} \xrightarrow{\mathrm{a}}{}^* \vec{\tau}$*
7. *if $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$, τ is an approximation, and $\theta\alpha$ is maximal for every $\alpha \in \mathcal{V}\sigma$, then $\theta\sigma \xrightarrow{\mathrm{a}}{}^* \tau$.*

Proof. *1.* Obvious.

2. Let $\sigma \xrightarrow{\mathrm{a}} \tau$. We prove by induction on σ that σ is inhabited if and only if τ is inhabited. If $\sigma = \alpha$, then $\tau = _$ and hence the claim is trivial.

Let $\sigma = \xi(\vec{\sigma})$. If $\tau = _$, then $\sigma = \xi(_, \ldots, _) \neq -$ and hence σ is inhabited, which yields the claim trivial. If $\tau = -$, then σ is top-level void and hence void, which yields the claim

trivial. Otherwise, we have $\tau = \xi(\vec{\tau})$ and $\vec{\sigma} \xrightarrow{\mathrm{a}}{}^* \vec{\tau}$. By the induction hypothesis we know that a component of $\vec{\sigma}$ is inhabited if and only if the corresponding component of $\vec{\tau}$ is inhabited. Hence we know by Corollary 5.2.7 that $\xi(\vec{\sigma})$ is inhabited if and only if $\xi(\vec{\tau})$ is inhabited.

3. Follows from statement (1) and the Normalization Theorem.

4. Let σ be maximal. We prove by induction on σ that $\downarrow\sigma = _$. If σ is a variable, then the claim is obvious. Otherwise, $\sigma = \xi(\vec{\sigma})$, ξ is maximal and every component of $\vec{\sigma}$ is maximal. Hence we know by the induction hypothesis that $\downarrow\vec{\sigma} = (_, \ldots, _)$ and thus, since ξ is maximal, $\downarrow\xi(\vec{\sigma}) = _$.

Let $\downarrow\sigma = _$. We prove by induction on σ that σ is maximal. If σ is a variable, then the claim is obvious. Otherwise, $\sigma = \xi(\vec{\sigma})$, ξ is maximal and $\downarrow\vec{\sigma} = (_, \ldots, _)$. Hence we know by the induction hypothesis that every component of $\vec{\sigma}$ is maximal and thus, since ξ is maximal, that $\xi(\vec{\sigma})$ is maximal.

5. Let $\theta \xrightarrow{\mathrm{a}}{}^* \psi$. We prove by induction on σ that $\theta\sigma \xrightarrow{\mathrm{a}}{}^* \psi\sigma$. If $\sigma = \alpha$, then the claim is trivial. Otherwise, we have $\sigma = \xi(\vec{\sigma})$ and $\theta\vec{\sigma} \xrightarrow{\mathrm{a}}{}^* \psi\vec{\sigma}$ by the induction hypothesis, which yields $\theta\sigma = \xi(\theta\vec{\sigma}) \xrightarrow{\mathrm{a}}{}^* \xi(\psi\vec{\sigma}) = \psi\sigma$.

6. Let $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $\tau \Rightarrow^* \xi(\vec{\tau})$. Then neither σ nor τ is a variable. If $\tau = -$, then $- = \xi(\vec{\tau}) = \xi(\vec{\sigma})$ and hence the claim is trivial. Otherwise, $\sigma = \eta(\vec{\sigma}_1)$, $\tau = \eta(\vec{\tau}_1)$ and $\vec{\sigma}_1 \xrightarrow{\mathrm{a}}{}^* \vec{\tau}_1$. Furthermore, $\eta(\vec{\alpha}) \Rightarrow^* \xi(\vec{\mu})$ and $\vec{\sigma} = \{\vec{\alpha}/\vec{\sigma}_1\}\vec{\mu} \xrightarrow{\mathrm{a}}{}^* \{\vec{\alpha}/\vec{\tau}_1\}\vec{\mu} = \vec{\tau}$ by the preceding statement.

7. Follows by induction on σ using statement (4). □

Theorem 6.3.4 [Retract] *If $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$ and $\sigma \leq \mu$, then $\sigma \xrightarrow{\mathrm{n}}{}^* \tau \uparrow \mu$.*

Proof. Let $\sigma \xrightarrow{\mathrm{a}}{}^* \tau$ and $\sigma \leq \mu$. We prove by induction on σ that $\sigma \xrightarrow{\mathrm{n}}{}^* \tau \uparrow \mu$.

Let σ be a variable. Then $\sigma = \mu$ and τ is a variable. Hence $\sigma = \tau \uparrow \sigma = \tau \uparrow \mu$.

Let $\sigma = \xi(\vec{\sigma})$ and τ be a variable. Then $\downarrow\sigma = _$ and hence σ is maximal. Thus $\sigma = \mu$ and $\sigma = \tau \uparrow \sigma = \tau \uparrow \mu$.

Let $\sigma = \xi(\vec{\sigma})$ and $\tau = -$. Then σ is void and hence $\sigma \xrightarrow{\mathrm{n}}{}^* - = \tau \uparrow \mu$.

Let $\sigma = \xi(\vec{\sigma})$ and $\tau = \xi(\vec{\tau})$. Then $\vec{\sigma} \xrightarrow{\mathrm{a}}{}^* \vec{\tau}$, $\mu \Rightarrow^* \xi(\vec{\mu})$ and $\vec{\sigma} \leq \vec{\mu}$. Hence $\vec{\sigma} \xrightarrow{\mathrm{n}}{}^* \vec{\tau} \uparrow \vec{\mu}$ by the induction hypothesis. Thus $\sigma = \xi(\vec{\sigma}) \xrightarrow{\mathrm{n}}{}^* \xi(\vec{\tau} \uparrow \vec{\mu}) = \xi(\vec{\tau}) \uparrow \mu$. □

Corollary 6.3.5 *If σ is normal and $\sigma \leq \tau$, then $\sigma = (\downarrow\sigma) \uparrow \tau$.*

The next two lemmas will be needed in the proofs to come.

Lemma 6.3.6 *Let Π be a pyramid, σ be Π-admissible, and $\pi_i\alpha$ be maximal for every $\alpha \in \mathcal{V}\sigma$. Then, for every $\pi_j \in \Pi$, $\pi_i\alpha$ is maximal for every $\alpha \in \mathcal{V}\pi_j\sigma$.*

Proof. If $\lambda\sigma \geq j$, then $\sigma = \pi_j\sigma$ and hence the claim is trivial. Otherwise, $\lambda\sigma < j$ and let $\alpha \in \mathcal{V}\pi_j\sigma$. Then there exists $\beta \in \mathcal{V}\sigma$ such that $\alpha \in \mathcal{V}\pi_j\beta$. If $i \geq j$, then $\pi_i\alpha$ is a subterm of $\pi_i\pi_j\beta = \pi_i\beta$ and hence $\pi_i\alpha$ is maximal since $\pi_i\beta$ is maximal. If $i < j$, then $\pi_i\alpha = \alpha$ since $\lambda\alpha = j > i$. □

Lemma 6.3.7 *Let $P \vdash^{\Pi} s\colon\sigma$ and $P \xrightarrow{\mathrm{a}}{}^{*} Q$. Furthermore, let every sort term in Q be an approximation and let $\pi_i\alpha$ be maximal for every $\alpha \in \mathcal{V}\sigma$. Then $\pi_i Px \xrightarrow{\mathrm{a}}{}^{*} Qx$ for every $x \in \mathcal{V}s$.*

Proof. We prove $\pi_i Px \xrightarrow{\mathrm{a}}{}^{*} Qx$ by induction on the length of a derivation $P \vdash^{\Pi} s\colon\sigma$.

Let $s = x$. Then $Px \leq \pi_{\lambda Px}\sigma$ and hence $\mathcal{V}Px \subseteq \mathcal{V}(\pi_{\lambda Px}\sigma)$. Thus we know by the preceding lemma that $\pi\alpha$ is maximal for every $\alpha \in \mathcal{V}Px$. Hence $\pi_i Px \xrightarrow{\mathrm{a}}{}^{*} Qx$ by statement (7) of the Approximation Theorem.

Let $s = f(\vec{s})$ and σ be a variable. Then $\lambda^P s > \lambda\sigma$ and $P \vdash^{\Pi} s\colon\pi_{\lambda^P s}\sigma$. Thus we know by the preceding lemma that $\pi\alpha$ is maximal for every $\alpha \in \mathcal{V}\pi_{\lambda^P s}\sigma$. Hence we have by the induction hypothesis that $\pi_i Px \xrightarrow{\mathrm{a}}{}^{*} Qx$ for every $x \in \mathcal{V}s$.

Let $s = f(\vec{s})$ and σ be no variable. Then we have $f\colon\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}\colon\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Since $\mathcal{V}(\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}) \subseteq \mathcal{V}\sigma$, the claim follows by the induction hypothesis. □

6.4 P-Infima and Mergings

Now it's time to tell you more about the proof of the Hauptsatz. The basic idea, which I discussed before, is that the constraint solver simulates a hypothetical computation using the actual sort terms rather than their weakest approximations. The hypothetical computation, which is called *P-computation* (P stands for pyramid), attempts to solve the system $\pi P \,\&\, E$, where $\pi = \pi_n$ is the lowest layer of the pyramid providing full information. However, the P-computation has still very limited access to the pyramid Π and only knows the information levels of the occurring sort variables. In particular, the P-computation has to start with $P \,\&\, E$ rather than $\pi P \,\&\, E$ and cannot access the information layers π_i of the pyramid.

Not using the pyramid in the P-computation provides for a close coupling (realized by "$\downarrow\sigma$") of the approximations used by the constraint solver and the sort terms used by the P-computation.

The merging operation "$\sigma \downarrow \tau$" simulates the operation "NF$[\pi\sigma' \sqcap \pi\tau']$, where σ' and τ' are the actual sort terms. For the P-computation we need an operation "$\sigma' * \tau'$" simulating "$\pi\sigma' \sqcap \pi\tau'$", which is defined on Π-admissible sort terms as follows:

$$
\begin{aligned}
\alpha * \alpha &= \alpha \\
\alpha * \beta &= \alpha \qquad \text{if } \lambda\alpha > \lambda\beta \\
\alpha * \beta &= \beta \qquad \text{if } \lambda\alpha < \lambda\beta \\
\alpha * \tau &= \tau \qquad \text{if } \tau \text{ is no variable} \\
\sigma * \beta &= \sigma \qquad \text{if } \sigma \text{ is no variable} \\
\xi(\vec{\sigma}) * \eta(\vec{\tau}) &= \zeta(\vec{\mu} * \vec{\nu}) \qquad \text{if } \zeta = \xi \sqcap \eta,\ \xi(\vec{\sigma}) \Rightarrow^{*} \zeta(\vec{\mu}),\ \eta(\vec{\tau}) \Rightarrow^{*} \zeta(\vec{\nu}).
\end{aligned}
$$

If $\sigma * \tau$ exists, then we call $\sigma * \tau$ the **P-infima** of σ and τ.

Theorem 6.4.1 [P-Infima] *P-Infima have the following properties:*

1. *if* $\sigma * \tau$ *exists, then* $\sigma * \tau = \tau * \sigma$

2. *if* $\sigma \leq \mu$ *and* $\tau \leq^{\Pi} \mu$*, then* $\sigma * \tau$ *exists and* $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap^{\circ} \tau$

3. *if* $\sigma, \tau \leq^{\Pi} \mu$*, then* $\sigma * \tau$ *exists and* $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap^{\circ} \pi_{\lambda\sigma}\tau \leq^{\Pi} \sigma, \tau$

4. *if* $\sigma * \tau$ *exists and* $\mu \leq \theta\sigma, \theta\tau$*, then* $\mu \leq \theta(\sigma * \tau)$

5. *if* $\sigma * \tau$ *exists and* $\mu \leq^{\Pi} \sigma, \tau$*, then* $\mu \leq^{\Pi} \sigma * \tau$.

Proof. *1.* Can be shown with a straightforward induction on σ.

2. Let $\sigma \leq \mu$ and $\tau \leq^{\Pi} \mu$. We show by induction on on $\|\sigma\|$ (see Section 4.3, called R-complexity there) that $\sigma * \tau$ exists and $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap \tau$. This suffices since the infimum of $\pi_{\lambda\tau}\sigma$ and τ is stable because $\pi_{\lambda\tau}\sigma, \tau \leq \pi_{\lambda\tau}\mu$ (Theorem 4.3.7).

Let $\sigma = \alpha$. Then $\mu = \alpha = \sigma$ and hence $\tau \leq^{\Pi} \sigma$ and $\pi_{\lambda\tau}\sigma \sqcap \tau = \tau$. If τ is no variable, then $\sigma * \tau = \tau$. Otherwise, let $\tau = \beta$. Then $\beta = \pi_{\lambda\beta}\alpha$ and hence $\lambda\beta \geq \lambda\alpha$. If $\lambda\beta = \lambda\alpha$, then $\beta = \alpha$ and $\sigma * \tau = \beta * \beta = \beta = \tau$. If $\lambda\beta > \lambda\alpha$, then $\sigma * \tau = \alpha * \beta = \beta = \tau$.

Let $\sigma = \xi(\vec{\sigma})$ and $\tau = \beta$. Then $\beta = \pi_{\lambda\beta}\mu$ and hence μ is a variable. Thus $\sigma = -$. Hence $\sigma * \tau = - = - \sqcap \tau = \pi_{\lambda\tau}\sigma \sqcap \tau$.

Let $\sigma = \xi(\cdots)$ and $\tau = \eta(\cdots)$. Furthermore, let $\zeta = \xi \sqcap \eta$, $\sigma \Rightarrow^{*} \zeta(\vec{\sigma})$, $\tau \Rightarrow^{*} \zeta(\vec{\tau})$ and $\mu \Rightarrow^{*} \zeta(\vec{\mu})$. Then $\vec{\sigma} \leq \vec{\mu}$ and $\vec{\tau} \leq^{\Pi} \vec{\mu}$. Hence we know by the induction hypothesis that $\vec{\sigma} * \vec{\tau} = \pi_{\lambda\vec{\tau}}\vec{\sigma} \sqcap \vec{\tau}$. Thus $\sigma * \tau = \zeta(\pi_{\lambda\vec{\tau}}\vec{\sigma} \sqcap \vec{\tau}) = \pi_{\lambda\tau}\sigma \sqcap \tau$, provided we can show that $\pi_{\lambda\vec{\tau}}\vec{\sigma} \sqcap \vec{\tau} = \pi_{\lambda\tau}\vec{\sigma} \sqcap \vec{\tau}$.

Let σ_i and τ_i be corresponding components of $\vec{\sigma}$ and $\vec{\tau}$. If τ_i contains a variable, then $\lambda\tau_i = \lambda\tau$. Otherwise, τ_i is ground. Since we already know that $\pi_{\lambda\tau}\sigma \sqcap \tau$ is stable, we know in particular that $\pi_{\lambda\tau}\sigma_i \sqcap \tau_i$ is stable. Hence we have $\pi_{\lambda\tau}\sigma_i \sqcap \tau_i = \pi(\pi_{\lambda\tau}\sigma_i \sqcap \tau_i) = \pi\pi_{\lambda\tau}\sigma_i \sqcap \pi\tau_i = \pi_{\lambda\tau_i}\sigma_i \sqcap \tau_i$ since τ_i and hence $\pi_{\lambda\tau}\sigma_i \sqcap \tau_i$ is ground.

3. Let $\sigma, \tau \leq \mu$. Since "$\sigma * \tau$" and "$\sigma \sqcap^{\circ} \tau$" are commutative, we can assume without loss of generality that $\lambda\sigma \leq \lambda\tau$. Then $\sigma \leq \pi_{\lambda\sigma}\mu$ and $\tau \leq^{\Pi} \pi_{\lambda\sigma}\mu$. Thus we have $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap^{\circ} \tau = \pi_{\lambda\tau}\sigma \sqcap^{\circ} \pi_{\lambda\sigma}\tau$ using the preceding statement. Since $\sigma * \tau \leq \pi_{\lambda\tau}\sigma, \tau$, we know that $\lambda(\sigma * \tau) \geq \lambda\tau \geq \lambda\sigma$ and thus $\sigma * \tau = \pi_{\lambda(\sigma*\tau)}(\sigma * \tau) \leq \pi_{\lambda(\sigma*\tau)}\sigma, \pi_{\lambda(\sigma*\tau)}\tau$. Hence $\sigma * \tau \leq^{\Pi} \sigma, \tau$.

4. Let $\sigma * \tau$ exist and $\mu \leq \theta\sigma, \theta\tau$. We prove by induction on $\|\sigma\|$ that $\mu \leq \theta(\sigma * \tau)$.

If σ or τ is a variable, then $\sigma * \tau \in \{\sigma, \tau\}$ and hence the claim is trivial.

Let $\sigma = \xi_{\sigma}(\cdots)$ and $\tau = \xi_{\tau}(\cdots)$. Then we have $\xi = \xi_{\sigma} \sqcap \xi_{\tau}$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$, $\tau \Rightarrow^{*} \xi(\vec{\tau})$, $\sigma * \tau = \xi(\vec{\sigma} * \vec{\tau})$, $\vec{\mu} = \xi_{\mu}(\vec{\mu})$, $\xi(\vec{\alpha}) \Rightarrow^{*} \xi_{\mu}(\vec{\nu})$, $\vec{\mu} \leq \theta\{\vec{\alpha}/\vec{\sigma}\}\vec{\nu}$, and $\vec{\mu} \leq \theta\{\vec{\alpha}/\vec{\tau}\}\vec{\nu}$. Let μ_i be a component of $\vec{\mu}$ and let ν_i be the corresponding component of $\vec{\nu}$.

We now prove by induction on $\|\nu_i\|$ that $\mu_i \leq \theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\nu_i$.

Let $\nu_i = \beta$. Furthermore, let $\sigma_i := \{\vec{\alpha}/\vec{\sigma}\}\beta$ and $\tau_i := \{\vec{\alpha}/\vec{\tau}\}\beta$. Then $\mu_i \leq \theta\sigma_i, \theta\tau_i$. Hence we know by the outer induction hypothesis that $\mu_i \leq \theta(\sigma_i * \tau_i) = \theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\nu_i$.

Otherwise, let $\mu_i = \eta(\vec{\mu}_i)$ and $\nu_i \Rightarrow^{*} \nu(\vec{\nu}_i)$. Then we have $\vec{\mu}_i \leq \theta\{\vec{\alpha}/\vec{\sigma}\}\vec{\nu}_i$ and $\vec{\mu}_i \leq \theta\{\vec{\alpha}/\vec{\tau}\}\vec{\nu}_i$. Hence we know by the inner induction hypothesis that $\vec{\mu}_i \leq \theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\vec{\nu}_i$. Thus $\theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\nu_i \Rightarrow^{*} \eta(\theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\vec{\nu}_i) \geq \eta(\vec{\mu}_i) = \mu_i$.

We now return to the outer induction. We now know that $\mu \leq \theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\nu$. Hence $\theta(\sigma * \tau) = \xi(\theta(\vec{\sigma} * \vec{\tau})) \Rightarrow^* \xi_\mu(\theta\{\vec{\alpha}/(\vec{\sigma} * \vec{\tau})\}\vec{\nu}) \geq \xi_\mu(\vec{\mu}) = \mu$.

5. Follows immediately from statement (4). □

Theorem 6.4.2 [Merging] *Let T be fully inhabited. Then the merging operation "$\sigma \downarrow \tau$" has the following properties:*

1. *if σ and τ are approximations, then their merging exists and $\sigma \downarrow \tau = \tau \downarrow \sigma$*
2. *if σ and τ are weakest approximations, then their merging $\sigma \downarrow \tau$ is a weakest approximation*
3. *if $\sigma, \tau \leq^\Pi \mu$, then $\downarrow(\sigma * \tau) = (\downarrow\sigma) \downarrow (\downarrow\tau)$.*

Proof. 1. Can be shown by a straightforward induction on σ.

2. Let σ and τ be weakest approximations. We show by induction on $\|\sigma\|$ that $\sigma \downarrow \tau$ is a weakest approximation.

If $\sigma = _$ or $\tau = _$, then $\sigma \downarrow \tau \in \{\sigma, \tau\}$ and hence the claim is obvious.

Otherwise, let $\sigma = \xi(\vec{\sigma})$, $\tau = \eta(\vec{\tau})$, $\zeta = \xi \sqcap \eta$, $\sigma \Rightarrow^* \zeta(\vec{\sigma}_1)$, $\tau \Rightarrow^* \zeta(\vec{\tau}_1)$, and $\vec{\mu} = (\downarrow\vec{\sigma}_1) \downarrow (\downarrow\vec{\tau}_1)$. Then we know by the induction hypothesis that $\vec{\mu}$ is a tuple of weakest approximations. If $\zeta(\vec{\mu})$ is top-level void then $\sigma \downarrow \tau = -$ and hence the claim is trivial. If $\zeta(\vec{\mu})$ is not top-level void, it suffices to show that either ζ is not maximal or $\vec{\mu} \neq (_, \ldots, _)$.

Suppose $\vec{\mu} = (_, \ldots, _)$ and let ζ be maximal. Then $\xi = \eta = \zeta$ and, by the definition of "$\sigma \downarrow \tau$", $\vec{\sigma} = \downarrow\vec{\sigma} = \downarrow\vec{\sigma}_1 = (_, \ldots, _)$ and $\vec{\tau} = \downarrow\vec{\tau} = \downarrow\vec{\tau}_1 = (_, \ldots, _)$. Hence $\sigma = \xi(\vec{\sigma})$ is not a weakest approximation, which contradicts our assumptions.

3. Let $\sigma, \tau \leq^\Pi \mu$. Since "$\sigma * \tau$" and "$\sigma \downarrow \tau$" are commutative, we can assume without loss of generality that $\lambda\sigma \leq \lambda\tau$. Since then $\sigma \leq \pi_{\lambda\sigma}\mu$ and $\tau \leq^\Pi \pi_{\lambda\sigma}\mu$ we can assume without loss of generality that $\sigma \leq \mu$ and $\tau \leq^\Pi \mu$. Now it suffices to show that $\sigma * \tau \xrightarrow{a}{}^* (\downarrow\sigma) \downarrow (\downarrow\tau)$ since we know by the preceding statement that $(\downarrow\sigma) \downarrow (\downarrow\tau)$ is a weakest approximation.

Assuming $\sigma \leq \mu$ and $\tau \leq^\Pi \mu$, we show by induction on $\|\sigma\|$ that $\sigma * \tau \xrightarrow{a}{}^* (\downarrow\sigma) \downarrow (\downarrow\tau)$.

If $\downarrow\sigma = -$ or $\downarrow\tau = -$, then $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap \tau$ is void and hence $\sigma * \tau = \xrightarrow{a}{}^* - = (\downarrow\sigma) \downarrow (\downarrow\tau)$.

If $\downarrow\sigma = _$, then σ is maximal and hence $\sigma = \mu$. Thus $\tau \leq \pi_{\lambda\tau}\sigma$ and hence $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap \tau = \tau \xrightarrow{a}{}^* \downarrow\tau = (\downarrow\sigma) \downarrow (\downarrow\tau)$.

If $\downarrow\tau = _$, then τ is maximal and hence $\tau = \pi_{\lambda\tau}\mu$. Since $\sigma \leq \mu$ we have $\pi_{\lambda\tau}\sigma \leq \tau$. Hence $\sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap \tau = \pi_{\lambda\tau}\sigma$ and $(\downarrow\sigma) \downarrow (\downarrow\tau) = \downarrow\sigma$. It remains to show that $\pi_{\lambda\tau}\sigma \xrightarrow{a}{}^* \downarrow\sigma$. Since $\sigma \xrightarrow{a}{}^* \downarrow\sigma$, we know by statement (7) of the Approximation Theorem that it suffices to show that $\pi_{\lambda\tau}\alpha$ is maximal for every $\alpha \in \mathcal{V}\sigma$. This is the case since $\mathcal{V}\sigma \subseteq \mathcal{V}\mu$ and $\pi_{\lambda\tau}\mu = \tau$ is maximal by assumption.

Otherwise, let $\sigma = \xi(\vec{\sigma})$, $\tau = \eta(\vec{\tau})$, $\downarrow\sigma = \xi(\downarrow\vec{\sigma})$ and $\downarrow\tau = \eta(\downarrow\vec{\tau})$. Furthermore, let $\zeta = \xi \sqcap \eta$, $\sigma \Rightarrow^* \zeta(\vec{\sigma}_1)$, $\tau \Rightarrow^* \zeta(\vec{\tau}_1)$, $\mu \Rightarrow^* \zeta(\vec{\mu}_1)$, $\xi(\downarrow\vec{\sigma}) \Rightarrow^* \zeta(\vec{\sigma}_2)$, and $\eta(\downarrow\vec{\tau}) \Rightarrow^* \zeta(\vec{\tau}_2)$. Then $\sigma * \tau = \zeta(\vec{\sigma}_1 * \vec{\tau}_1)$, $\vec{\sigma}_1 \leq \vec{\mu}_1$, $\vec{\tau}_1 \leq^\Pi \vec{\mu}_1$, and, using statement (6) of the Approximation Theorem, $\vec{\sigma}_1 \xrightarrow{a}{}^* \vec{\sigma}_2$ and $\vec{\tau}_1 \xrightarrow{a}{}^* \vec{\tau}_2$. Hence $\downarrow\vec{\sigma}_1 = \downarrow\vec{\sigma}_2$ and $\downarrow\vec{\tau}_1 = \downarrow\vec{\tau}_2$ by the confluence of $\xrightarrow{a}{}^*$.

By the induction hypothesis we know $\vec{\sigma}_1 * \vec{\tau}_1 \xrightarrow{\mathrm{a}}{}^* (\downarrow\vec{\sigma}_1) \downarrow (\downarrow\vec{\tau}_1)$. Hence $\sigma * \tau = \zeta(\vec{\sigma}_1 * \vec{\tau}_1) \xrightarrow{\mathrm{a}}{}^* \zeta((\downarrow\vec{\sigma}_1) \downarrow (\downarrow\vec{\tau}_1)) = \zeta((\downarrow\vec{\sigma}_2) \downarrow (\downarrow\vec{\tau}_2)) \xrightarrow{\mathrm{a}}{}^* \xi(\downarrow\vec{\sigma}) \downarrow \eta(\downarrow\tau) = (\downarrow\vec{\sigma}) \downarrow (\downarrow\vec{\tau})$. □

6.5 Solving PM-Systems

General Assumption. *In this section we assume that T is fully inhabited.*

Now we are ready to present the part of the P-computation that is simulated by the sort level of the constraint solver. Recall that the sort level of the constraint solver solves membership systems of the form $P \,\&\, M$.

We start with the invariant preserved by the membership reduction rules employed in the P-computation.

A pair $P.M$ is **Π-well-typed** if

1. P is a normal, inhabited and Π-admissible prefix
2. M is a Π-admissible membership system such that for every membership $s{:}\,\sigma \in M$ there exists a Π-admissible sort term μ such that $P \vdash^{\Pi} s{:}\,\mu$ and $\sigma \leq^{\Pi} \mu$.

The following rules define a binary relation $\xrightarrow{\mathrm{p}}$ on pairs $P.M$:

1. $P.s{:}\,\alpha \,\&\, M \xrightarrow{\mathrm{p}} P.M$
   if s is no variable
2. $P \,\&\, x{:}\,\sigma.x{:}\,\tau \,\&\, M \xrightarrow{\mathrm{p}} P \,\&\, x{:}\,\mathrm{NF}[\sigma * \tau].M$
   if $\mathrm{NF}[\sigma * \tau] \neq -$
3. $P.f(\vec{s}){:}\,\sigma \,\&\, M \xrightarrow{\mathrm{p}} P.M$
   if $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$, and $\vec{\mu}$ is ground
4. $P.f(\vec{s}){:}\,\sigma \,\&\, M \xrightarrow{\mathrm{p}} P.\vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \,\&\, M$
   if $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$, and $\vec{\mu}$ is not ground.

The relation $\xrightarrow{\mathrm{p}}$ mimics the relation $\xrightarrow{\mathrm{m}}$, where $\xrightarrow{\mathrm{p}}$ employs Π-admissible sort terms while $\xrightarrow{\mathrm{m}}$ employs weakest approximations.

Proposition 6.5.1 *There are no infinite chains $P.M \xrightarrow{\mathrm{p}} P'.M' \xrightarrow{\mathrm{p}} \cdots$ or $P.M \xrightarrow{\mathrm{m}} P'.M' \xrightarrow{\mathrm{m}} \cdots$. Furthermore, if $P.M \xrightarrow{\mathrm{p}}{}^* P'.M'$ or $P.M \xrightarrow{\mathrm{m}}{}^* P'.M'$, then $\mathcal{D}P = \mathcal{D}P'$ and $Px = P'x$ for every $x \in \mathcal{D}P - \mathcal{V}M$.*

Our goal is to prove Theorems 6.5.6 and 6.5.7 appearing at the end of this section. I suggest that you first look at these theorems and convenience yourself that they just state what you expect anyway at this point. Then you can go back and look at the preparing lemmas following now.

Lemma 6.5.2 [Invariance] *Let $P.M$ be Π-well-typed and $P.M \xrightarrow{\mathrm{p}} P'.M'$. Then:*

1. *$P'.M'$ is Π-well-typed*
2. *if Q is a normal prefix, then $Q \vdash^{\Pi} P \,\&\, M \iff Q \vdash^{\Pi} P' \,\&\, M'$*
3. $\mathrm{U}[P \,\&\, M] \subseteq \mathrm{U}[P' \,\&\, M']$.

Proof. We have to prove the claims for each of the four $\xrightarrow{\mathrm{p}}$-rules.

I. Let $P.s{:}\,\alpha \,\&\, M \xrightarrow{\mathrm{p}} P.M$, s be no variable, and let $P.s{:}\,\alpha \,\&\, M$ be Π-well-typed.

I.1. Obvious.

I.2. The direction "$\Rightarrow$" is obvious. To show the other direction, suppose Q is a normal prefix such that $Q \vdash^{\Pi} P \,\&\, M$. Since $P.s{:}\,\alpha \,\&\, M$ is Π-well-typed, we know that there exists a variable β such that $\alpha = \pi_{\lambda\alpha}\beta$ and $P \vdash^{\Pi} s{:}\,\beta$. Hence we know by the Substitution Lemma that $Q \vdash^{\Pi} s{:}\,\beta$. Thus $\lambda\alpha < \lambda^{Q}s$ and hence $Q \vdash^{\Pi} s{:}\,\alpha$ by the Flexibility Lemma. Thus $Q \vdash^{\Pi} P \,\&\, s{:}\,\alpha \,\&\, M$.

I.3. Obvious.

II. Let $P \,\&\, x{:}\,\sigma.x{:}\,\tau \,\&\, M \xrightarrow{\mathrm{p}} P \,\&\, x{:}\,\mathrm{NF}[\sigma * \tau].M$, $\mathrm{NF}[\sigma * \tau] \neq -$, and let $P \,\&\, x{:}\,\sigma.x{:}\,\tau \,\&\, M$ be Π-well-typed. Then we have $\sigma, \tau \leq^{\Pi} \mu$ and hence

$$\mathrm{NF}[\sigma * \tau] \leq \sigma * \tau = \pi_{\lambda\tau}\sigma \sqcap^{\circ} \pi_{\lambda\sigma}\tau \leq^{\Pi} \sigma, \tau$$

by the P-Infima Theorem. Thus $P' := P \,\&\, x{:}\,\mathrm{NF}[\sigma * \tau]$ is a normal, inhabited and Π-admissible prefix.

II.1. Let $s{:}\,\nu \in M$. It suffices to show that there exists a Π-admissible μ such that $P' \vdash^{\Pi} s{:}\,\mu$ and $\nu \leq^{\Pi} \mu$. Since $P \,\&\, x{:}\,\sigma.x{:}\,\tau \,\&\, M$ is Π-well-typed, there exists a Π-admissible μ such that $P \,\&\, x{:}\,\sigma \vdash^{\Pi} s{:}\,\mu$ and $\nu \leq^{\Pi} \mu$. Since $P' \vdash^{\Pi} P \,\&\, x{:}\,\sigma$, we know $P' \vdash^{\Pi} s{:}\,\mu$ by the Substitution Lemma.

II.2. Let Q be a normal prefix such that $Q \vdash^{\Pi} P \,\&\, x{:}\,\sigma \,\&\, x{:}\,\tau \,\&\, M$. It suffices to show that $Q \vdash^{\Pi} x{:}\,\mathrm{NF}[\sigma * \tau]$. Since $Qx \leq^{\Pi} \sigma, \tau$, we know by the P-Infima Theorem that $Qx \leq^{\Pi} \sigma * \tau$. Hence $Qx \leq \pi_{\lambda Qx}(\sigma * \tau)$ and thus

$$Qx = \mathrm{NF}[Qx] \leq \mathrm{NF}[\pi_{\lambda Qx}(\sigma * \tau)] \leq \pi_{\lambda Qx}\mathrm{NF}[\sigma * \tau]$$

by the normality of Q and the Normalization Theorem. Thus $Q \vdash^{\Pi} x{:}\,\mathrm{NF}[\sigma * \tau]$.

To show the other direction, suppose Q is a normal prefix such that $Q \vdash^{\Pi} P \,\&\, x{:}\,\mathrm{NF}[\sigma * \tau] \,\&\, M$. Then $Qx \leq^{\Pi} \mathrm{NF}[\sigma * \tau] \leq^{\Pi} \sigma, \tau$. Hence $Q \vdash^{\Pi} P \,\&\, x{:}\,\sigma \,\&\, x{:}\,\tau \,\&\, M$.

II.3. Let $\psi_Q \in \mathrm{U}[P \,\&\, x{:}\,\sigma \,\&\, x{:}\,\tau \,\&\, M]$. It suffices to show that $Q \vdash \psi x{:}\,\psi\mathrm{NF}[\sigma * \tau]$. Since $Q \vdash \psi x{:}\,\psi\sigma \,\&\, \psi x{:}\,\psi\tau$, we know that $\sigma^{Q}[\psi x] \leq \psi\sigma, \psi\tau$. Hence we know by the P-Infima Theorem that $\sigma^{Q}[\psi x] \leq \psi(\sigma * \tau)$ and thus

$$\sigma^{Q}[\psi x] = \mathrm{NF}[\sigma^{Q}[\psi x]] \leq \mathrm{NF}[\psi(\sigma * \tau)] \leq \psi\mathrm{NF}[\sigma * \tau]$$

by the normality and inhabitation of Q and the Normalization Theorem. Hence $Q \vdash \psi x{:}\,\psi\mathrm{NF}[\sigma * \tau]$.

III. Let $P.f(\vec{s}){:}\,\sigma \,\&\, M \xrightarrow{\mathrm{p}} P.M$, $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$, $\vec{\mu}$ be ground, and let $P.f(\vec{s}){:}\,\sigma \,\&\, M$ be Π-well-typed.

III.1. Obvious.

III.2. The direction "$\Rightarrow$" is obvious. To show the other direction, suppose Q is a normal prefix such that $Q \vdash^{\Pi} P \,\&\, M$. It suffices to show that $Q \vdash^{\Pi} f(\vec{s}){:}\,\sigma$. Since $P.\,f(\vec{s}){:}\,\sigma \,\&\, M$ is Π-well-typed, we know that there exists a Π-admissible ν such that $\sigma \leq^{\Pi} \nu$ and $P \vdash^{\Pi} f(\vec{s}){:}\,\nu$. Since σ is no variable, we can assume without loss of generality (by the Flexibility Lemma) that ν is no variable. Hence we know by the definition of $P \vdash^{\Pi} f(\vec{s}){:}\,\nu$ that $P \vdash^{\Pi} f(\vec{s}){:}\,\vec{\mu}$ since $\vec{\mu}$ is ground. Since $Q \vdash^{\Pi} P$, we know by the Substitution Lemma that $Q \vdash^{\Pi} \vec{s}{:}\,\vec{\mu}$. Hence $Q \vdash^{\Pi} f(\vec{s}){:}\,\sigma$ since $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$ and $\vec{\mu}$ is ground.

III.3. Obvious.

IV. Let $P.\,f(\vec{s}){:}\,\sigma \,\&\, M \xrightarrow{\mathrm{p}} P.\,\vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \,\&\, M$, $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$, $\vec{\mu}$ not be ground, and let $P.\,f(\vec{s}){:}\,\sigma \,\&\, M$ be Π-well-typed.

IV.1. It suffices to show that there exists a Π-admissible tuple $\vec{\nu}$ such that $P \vdash^{\Pi} \vec{s}{:}\,\vec{\nu}$ and $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \leq^{\Pi} \vec{\nu}$. Since $P.\,f(\vec{s}){:}\,\sigma \,\&\, M$ is Π-well-typed, there exists a Π-admissible ν such that $P \vdash^{\Pi} f(\vec{s}){:}\,\nu$ and $\sigma \leq^{\Pi} \nu$. Since σ is no variable, we can assume without loss of generality (by the Flexibility Lemma) that ν is no variable. Hence $\nu \Rightarrow^{*} \xi(\vec{\nu})$ and $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\nu}\}\vec{\mu}$. It remains to show that $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \leq^{\Pi} \{\vec{\alpha}/\vec{\nu}\}\vec{\mu}$.

Since $\sigma \leq \pi_{\lambda\sigma}\nu$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$ and $\pi_{\lambda\sigma}\nu \Rightarrow^{*} \xi(\pi_{\lambda\sigma}\vec{\nu})$, we have $\vec{\sigma} \leq \pi_{\lambda\sigma}\vec{\nu}$ and hence $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \leq \pi_{\lambda\sigma}\{\vec{\alpha}/\vec{\nu}\}\vec{\mu}$.

Now let μ_i be a component of μ. If $\{\vec{\alpha}/\vec{\sigma}\}\mu_i$ is ground, then $\{\vec{\alpha}/\vec{\sigma}\}\mu_i = \pi\{\vec{\alpha}/\vec{\sigma}\}\mu_i \leq \pi\pi_{\lambda\sigma}\{\vec{\alpha}/\vec{\nu}\}\mu_i = \pi\{\vec{\alpha}/\vec{\nu}\}\mu_i$ and hence $\{\vec{\alpha}/\vec{\sigma}\}\mu_i \leq^{\Pi} \{\vec{\alpha}/\vec{\nu}\}\mu_i$. If $\{\vec{\alpha}/\vec{\sigma}\}\mu_i$ is not ground, then $\lambda\{\vec{\alpha}/\vec{\sigma}\}\mu_i = \lambda\sigma$ since $\mathcal{V}\mu_i \subseteq \mathcal{V}\vec{\alpha}$ and $\mathcal{V}\vec{\sigma} \subseteq \mathcal{V}\sigma$. Hence $\{\vec{\alpha}/\vec{\sigma}\}\mu_i \leq^{\Pi} \{\vec{\alpha}/\vec{\nu}\}\mu_i$ since $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \leq \pi_{\lambda\sigma}\{\vec{\alpha}/\vec{\nu}\}\vec{\mu}$.

IV.2. Let Q be a normal prefix such that $Q \vdash^{\Pi} P \,\&\, f(\vec{s}){:}\,\sigma \,\&\, M$. Since $Q \vdash^{\Pi} f(\vec{s}){:}\,\sigma$, $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$ and $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$, we have $Q \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ and hence $Q \vdash^{\Pi} P \,\&\, \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \,\&\, M$.

To show the other direction, let Q be a normal prefix such that $Q \vdash^{\Pi} P \,\&\, \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu} \,\&\, M$. Since $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^{*} \xi(\vec{\sigma})$ and $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$, we have $Q \vdash^{\Pi} f(\vec{s}){:}\,\sigma$ and hence $Q \vdash^{\Pi} P \,\&\, f(\vec{s}){:}\,\sigma \,\&\, M$.

IV.3. Let $\psi_Q \in \mathrm{U}[P \,\&\, f(\vec{s}){:}\,\sigma \,\&\, M]$. It suffices to show that $Q \vdash \psi\vec{s}{:}\,\psi\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$. Since $Q \vdash f(\psi\vec{s}){:}\,\psi\sigma$, $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$ and $\psi\sigma \Rightarrow^{*} \xi(\psi\vec{\sigma})$, we have $Q \vdash \psi\vec{s}{:}\,\{\vec{\alpha}/\psi\vec{\sigma}\}\vec{\mu}$, which yields the claim since $\mathcal{V}\vec{\mu} \subseteq \mathcal{V}\vec{\alpha}$. □

Lemma 6.5.3 *Let P be a normal and inhabited prefix and $P \vdash^{\Pi} s{:}\,\sigma$. Then $P.\,s{:}\,\pi_i\sigma \xrightarrow{\mathrm{p}}{}^{*} P|_{\mathcal{D}P-\mathcal{V}s} \;\&\; \mathrm{NF}[\pi_i P|_{\mathcal{V}s}]\,.\,\emptyset$.*

Proof. We prove the claim by induction on the pair $(|s|, n - \lambda\sigma)$ with respect to the canonical lexicographic order, where $|s|$ is the size of s and n is the height of the pyramid Π. According to the definition of $P \vdash^{\Pi} s{:}\,\sigma$, we distinguish three cases.

1. Let $s = x$. Then $Px \leq^{\Pi} \sigma$. It suffices to show that $Px * \pi_i\sigma = \pi_i Px$ and that $\pi_i Px$ is inhabited, since then the second $\xrightarrow{\mathrm{p}}$-rule applies and yields the claim. Since P and π_i are inhabited, we know that $\pi_i Px$ is inhabited. Since $\pi_i\sigma, Px \leq^{\Pi} \sigma$, we know by the P-Infima Lemma that $Px * \pi_i\sigma = \pi_{\lambda\pi_i\sigma}Px \sqcap \pi_{\lambda Px}\pi_i\sigma$. Hence $Px * \pi_i\sigma = \pi_i Px \sqcap \pi_{\lambda Px}\pi_i\sigma = \pi_i Px$ since $\lambda Px \geq \lambda\sigma$ and $\pi_i Px \leq \pi_{\lambda Px}\pi_i\sigma = \pi_i\pi_{\lambda Px}\sigma$.

2. Let $s = f(\vec{s})$ and $\sigma = \alpha$. Then $\lambda\sigma < \lambda^P_s$, $P \vdash^\Pi s{:}\,\pi_{\lambda^P_s}\sigma$ and $\pi_{\lambda^P_s}\sigma$ is no variable.

If $\lambda^P_s \leq i$, then $\pi_i\sigma = \pi_i(\pi_{\lambda^P_s}\sigma)$ and the claim follows by the induction hypothesis.

Otherwise, we have $i < \lambda^P_s$ and hence $\mathrm{NF}[\pi_i P|_{\mathcal{V}s}] = \mathrm{NF}[P|_{\mathcal{V}s}] = P|_{\mathcal{V}s}$ since P is normal. If $\pi_i\sigma$ is a variable, then $P.\,s{:}\,\pi_i\sigma \xrightarrow{\mathrm{p}} P.\,\emptyset$ with the first $\xrightarrow{\mathrm{p}}$-rule, which yields the claim. If $\pi_i\sigma$ is no variable, then $P \vdash^\Pi s{:}\,\pi_i\sigma$ by the Flexibility Lemma and $\lambda\pi_i\sigma > \lambda\sigma$. Hence the claim follows by the induction hypothesis since $\pi_i(\pi_i\sigma) = \pi_i\sigma$.

3. Let $s = f(\vec{s})$ and σ be no variable. Then $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\sigma \Rightarrow^* \xi(\vec{\sigma})$ and $P \vdash^\Pi \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$.

If $\vec{\mu}$ is ground, we have $P.\,\pi_i\sigma \xrightarrow{\mathrm{p}} P.\,\emptyset$ by the third $\xrightarrow{\mathrm{p}}$-rule. Since $P \vdash^\Pi \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$, we know $\lambda^P_s = n$ by the Monotonicity Lemma. Hence $\mathrm{NF}[\pi_i P|_{\mathcal{V}s}] = \mathrm{NF}[P|_{\mathcal{V}s}] = P|_{\mathcal{V}s}$ since P is normal, which yields the claim.

If $\vec{\mu}$ is not ground, we have $P.\,\pi_i\sigma \xrightarrow{\mathrm{p}} P.\,\vec{s}{:}\,\pi_i\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$ by the fourth $\xrightarrow{\mathrm{p}}$-rule. Since we have $\mathrm{NF}[\pi_i\mathrm{NF}[\pi_i Px]] = \mathrm{NF}[\mathrm{NF}[\pi_i Px]] = \mathrm{NF}[\pi_i Px] \leq \pi_i Px \leq^\Pi Px$ for every $x \in \mathcal{D}P$, the claim follows by repeated application (one for every component of the tuple $\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$) of the induction hypothesis and the Substitution Lemma. □

Lemma 6.5.4 [Simulation] *Let $P.\,M$ be Π-well-typed and let $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}} Q.\,N$. Then there exist P' and M' such that $P.\,M \xrightarrow{\mathrm{p}}{}^+ P'.\,M'$ and $Q.\,N = \downarrow P'.\downarrow M'$.*

Proof. We assume without loss of generality that M consists of a single membership. Let $P.\,s{:}\,\sigma$ be Π-well-typed and let $\xrightarrow{\mathrm{m}}$ apply to $\downarrow P.\,s{:}\downarrow\sigma$. Since in this case $\xrightarrow{\mathrm{m}}$ is deterministic, it suffices to show that there exist P' and M' such that $P.\,s{:}\,\sigma \xrightarrow{\mathrm{p}}{}^+ P'.\,M'$ and $\downarrow P.\,s{:}\downarrow\sigma \xrightarrow{\mathrm{m}} \downarrow P'.\downarrow M'$. We distinguish two cases.

1. Let $\downarrow\sigma = _$. Then $\downarrow P.\,s{:}\downarrow\sigma \xrightarrow{\mathrm{m}} \downarrow P'.\,\emptyset$ and σ is maximal. Since $P.\,s{:}\,\sigma$ is Π-well-typed, we have $P \vdash^\Pi s{:}\,\mu$ and $\sigma \leq^\Pi \mu$. Hence $\sigma = \pi_{\lambda\sigma}\mu$ and we have

$$P.\,s{:}\,\sigma \xrightarrow{\mathrm{p}}{}^+ P|_{\mathcal{D}P-\mathcal{V}s} \;\&\; \mathrm{NF}[\pi_{\lambda\sigma}(P|_{\mathcal{V}s})].\,\emptyset$$

by the preceding lemma. It remains to show that $\downarrow\mathrm{NF}[\pi_{\lambda\sigma}(P|_{\mathcal{V}s})] = \downarrow(P|_{\mathcal{V}s})$.

Since σ is maximal and $\sigma = \pi_{\lambda\sigma}\mu$, we know that $\pi_{\lambda\sigma}x$ is maximal for every $\alpha \in \mathcal{V}\mu$. Since $P.\,s{:}\,\sigma$ is Π-well-typed, we know $P \vdash^\Pi s{:}\,\mu$. Hence we know by Lemma 6.3.7 that $\pi_{\lambda\sigma}(P|_{\mathcal{V}s}) \xrightarrow{\mathrm{a}}{}^* \downarrow(P|_{\mathcal{V}s})$. Hence $\mathrm{NF}[\pi_{\lambda\sigma}(P|_{\mathcal{V}s})] \xrightarrow{\mathrm{a}}{}^* \downarrow(P|_{\mathcal{V}s})$ since $\xrightarrow{\mathrm{n}} \subseteq \xrightarrow{\mathrm{a}}$ and $\xrightarrow{\mathrm{a}}$ is confluent. Thus $\downarrow\mathrm{NF}[\pi_{\lambda\sigma}(P|_{\mathcal{V}s})] = \downarrow(P|_{\mathcal{V}s})$.

2. Let $\downarrow\sigma \neq _$. Since $\tau \downarrow - = -$ for every τ and $\xrightarrow{\mathrm{m}}$ applies to $\downarrow P.\,s{:}\downarrow\sigma$, we know that $\downarrow\sigma \neq -$. Hence $\sigma = \eta(\vec{\sigma})$ and $\downarrow\sigma = \eta(\downarrow\vec{\sigma})$. Again, we distinguish two cases.

2.1. Let $s = x$. Then

$$\downarrow P.\,s{:}\downarrow\sigma \xrightarrow{\mathrm{m}} (\downarrow P - \{x{:}\downarrow Px\}) \;\&\; x{:}\,(\downarrow Px) \downarrow (\downarrow\sigma)\;.\,\emptyset$$

and $(\downarrow Px) \downarrow (\downarrow\sigma) \neq -$. Since $P.\,s{:}\,\sigma$ is Π-well-typed, we have $Px, \sigma \leq^\Pi \mu$. Hence $\downarrow(Px * \sigma) = (\downarrow Px) \downarrow (\downarrow\sigma) \neq -$ by the Merging Theorem. Thus $\downarrow\mathrm{NF}[Px * \sigma] = (\downarrow Px) \downarrow (\downarrow\sigma) \neq -$ since $\xrightarrow{\mathrm{n}} \subseteq \xrightarrow{\mathrm{a}}$ and $\xrightarrow{\mathrm{a}}$ is confluent. Hence

$$P.\,s{:}\,\sigma \xrightarrow{\mathrm{p}} (P - \{x{:}\,Px\}) \;\&\; x{:}\mathrm{NF}[Px * \sigma]\;.\,\emptyset,$$

which yields the claim.

2.2. Let $s = f(\vec{s})$. Then we have $f\colon\vec{\mu} \to \xi(\vec{\alpha}) \in T$ and $\downarrow\sigma = \eta(\downarrow\vec{\sigma}) \Rightarrow^* \xi(\vec{\sigma}_1)$ since $\xrightarrow{\mathrm{m}}$ applies to $\downarrow P.s\colon\downarrow\sigma$. Furthermore, we have $\sigma = \eta(\vec{\sigma}) \Rightarrow^* \xi(\vec{\sigma}_2)$ and $\vec{\sigma}_2 \xrightarrow{\mathrm{a}}{}^* \vec{\sigma}_1$ using the Approximation Theorem. If $\vec{\mu}$ is ground, then $\downarrow P.s\colon\downarrow\sigma \xrightarrow{\mathrm{m}} \downarrow P.\emptyset$ and $P.s\colon\sigma \xrightarrow{\mathrm{p}} P.\emptyset$, which yields the claim. Otherwise, we have

$$\downarrow P.s\colon\downarrow\sigma \xrightarrow{\mathrm{m}} \downarrow P.\vec{s}\colon\downarrow(\{\vec{\alpha}/\vec{\sigma}_1\}\vec{\mu})$$

and

$$P.s\colon\sigma \xrightarrow{\mathrm{p}} P.\vec{s}\colon\{\vec{\alpha}/\vec{\sigma}_2\}\vec{\mu}.$$

It remains to show that $\downarrow(\{\vec{\alpha}/\vec{\sigma}_2\}\vec{\mu}) = \downarrow(\{\vec{\alpha}/\vec{\sigma}_1\}\vec{\mu})$. Since $\vec{\sigma}_2 \xrightarrow{\mathrm{a}}{}^* \vec{\sigma}_1$, we have $\{\vec{\alpha}/\vec{\sigma}_2\}\vec{\mu} \xrightarrow{\mathrm{a}}{}^* \{\vec{\alpha}/\vec{\sigma}_1\}\vec{\mu}$ by the Approximation Theorem. Hence $\downarrow(\{\vec{\alpha}/\vec{\sigma}_2\}\vec{\mu}) = \downarrow(\{\vec{\alpha}/\vec{\sigma}_1\}\vec{\mu})$ by the confluence of $\xrightarrow{\mathrm{a}}$. □

Lemma 6.5.5 [Completeness] *Let $P.M$ be Π-well-typed and let $P \,\&\, M$ be unifiable. Then $\xrightarrow{\mathrm{m}}$ applies to $\downarrow P.\downarrow M$ if M is nonempty.*

Proof. We assume without loss of generality that M consists of a single membership. Let $P.s\colon\sigma$ be Π-well-typed and let $P \,\&\, s\colon\sigma$ be unifiable. Then there exists a substitution theta such that $\vdash \theta P$ and $\vdash \theta s\colon\theta\sigma$. Hence σ is inhabited and $\downarrow\sigma \neq -$. If $\downarrow\sigma = _$, then the second $\xrightarrow{\mathrm{m}}$-rule applies to $\downarrow P.s\colon\downarrow\sigma$. Otherwise we know that $\sigma = \eta(\vec{\sigma})$ and $\downarrow\sigma = \eta(\downarrow\vec{\sigma})$. Now we distinguish two cases.

Let $s = x$. It suffices to show that $(\downarrow Px) \downarrow (\downarrow\sigma) \neq -$ since then the second $\xrightarrow{\mathrm{m}}$-rule applies to $\downarrow P.s\colon\downarrow\sigma$. Since $P.s\colon\sigma$ is Π-well-typed we have $Px, \sigma \leq^{\Pi} \mu$. Hence we know by the Merging Theorem that $(\downarrow Px) \downarrow (\downarrow\sigma) = \downarrow(Px * \sigma)$. Furthermore, we know by the P-Infima Theorem that $\mathcal{L}^*\theta x \leq \theta(Px * \sigma)$ since $\mathcal{L}^*\theta x \leq \theta Px, \theta\sigma$ because $\vdash \theta x\colon\theta Px$ and $\vdash \theta x\colon\theta\sigma$. Since $\vdash \theta x\colon\mathcal{L}^*\theta x$ we know that $\mathcal{L}^*\theta x$ is inhabited. Hence $Px * \sigma$ is inhabited and thus $(\downarrow Px) \downarrow (\downarrow\sigma) = \downarrow(Px * \sigma) \neq -$.

Let $s = f(\vec{s})$. Since $\vdash \theta s\colon\theta\sigma$, we know $f\colon\vec{\mu} \to \xi(\vec{\alpha}) \in T$ and $\theta\sigma = \eta(\theta\vec{\sigma}) \Rightarrow^* \xi(\cdots)$. Hence $\downarrow\sigma = \eta(\downarrow\vec{\sigma}) \Rightarrow^* \xi(\cdots)$ and thus either the third or fourth $\xrightarrow{\mathrm{m}}$-rule is applicable. □

Theorem 6.5.6 [P-Reduction] *Let $P.M$ be Π-well-typed. Then:*

1. *$P \,\&\, M$ unifiable $\iff \exists\, Q.\ P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$*
2. *if $P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$, then:*
   (a) *if $P.M \xrightarrow{\mathrm{p}}{}^* P'.M'$, then $P'.M' \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$*
   (b) *Q is a normal, inhabited and Π-admissible prefix such that $Q \vdash^{\Pi} P \,\&\, M$, $\mathrm{U}[P \,\&\, M] \subseteq \mathrm{U}[Q]$, and $\emptyset_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, \pi M$.*

Proof. *1.* Suppose $P.M$ is Π-well-typed and $P \,\&\, M$ is unifiable. We show by induction on $P.M$ with respect to $\xrightarrow{\mathrm{p}}$ that there exists a prefix Q such that $P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$. If $M = \emptyset$, then the claim is trivial. Otherwise, we know by the Completeness and the Simulation Lemma that there exist P' and M' such that $P.M \xrightarrow{\mathrm{p}}{}^+ P'.M'$ and $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}} \downarrow P'.\downarrow M'$.

By the Invariance Lemma we know that $P'.M'$ is Π-well-typed and unifiable. Hence we know by the induction hypothesis that there exists a prefix Q such that $P'.M' \xrightarrow{\mathrm{m}}{}^* Q.\emptyset$.

Suppose $P.M$ is Π-well-typed and $P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$. By the Invariance Lemma we know that Q is a normal, inhabited and Π-admissible prefix. Hence $Q \vdash^{\Pi} Q$ and thus $Q \vdash^{\Pi} P \,\&\, M$ by the Invariance Lemma. Thus $\pi Q \vdash \pi P \,\&\, \pi M$ and hence $\pi Q \rightarrow \pi P \,\&\, \pi M$ is valid in $\mathcal{T}$. Since Q and π are inhabited, the prefix πQ is inhabited and hence unifiable. Thus $\pi P \,\&\, \pi M$ and hence $P \,\&\, M$ is unifiable.

2. Let $P.M$ be Π-well-typed and $P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$. Then we know by (1) that $P \,\&\, M$ is unifiable.

2.1. Suppose $P.M \xrightarrow{\mathrm{p}}{}^* P'.M'$. Then we know by the Invariance Lemma that $P'.M'$ is Π-well-typed and unifiable. Hence we know by (1) that there exists a prefix Q' such that $P'.M' \xrightarrow{\mathrm{p}}{}^* Q'.\emptyset$. By the Invariance Lemma we know that Q and Q' are normal and Π-admissible prefixes. Hence $Q \vdash^{\Pi} Q$ and $Q' \vdash^{\Pi} Q'$. Thus we know by the Invariance Lemma that $Q \vdash^{\Pi} Q'$ and $Q' \vdash^{\Pi} Q$. Hence $Q = Q'$.

2.2. By the Invariance Lemma we know that Q is a normal, inhabited and Π-admissible prefix such that $\mathrm{U}[P \,\&\, M] \subseteq \mathrm{U}[Q]$. Hence $\mathrm{U}[\pi P \,\&\, \pi M] \subseteq \mathrm{U}[\pi Q]$. Since $Q \vdash^{\Pi} Q$, we also have $Q \vdash^{\Pi} P \,\&\, M$ by the Invariance Lemma.

To show that $\emptyset_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, \pi M$, it remains to show that πQ is inhabited and that $\mathrm{U}[\pi Q] \subseteq \mathrm{U}[\pi P \,\&\, \pi M]$. Since π and Q are inhabited, we know that πQ is inhabited. Furthermore, suppose $\psi_R \in \mathrm{U}[\pi Q]$. Then $R \vdash \psi \pi Q$. Since $Q \vdash^{\Pi} P \,\&\, M$, we know $\pi Q \vdash \pi P \,\&\, \pi M$. Hence $R \vdash \psi \pi P \,\&\, \psi \pi M$ and thus $\psi_R \in \mathrm{U}[\pi P \,\&\, \pi M]$. □

Theorem 6.5.7 [M-Reduction] *Let $P.M$ be Π-well-typed. Then:*

1. *if $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q.N$, then there exist P' and M' such that $P.M \xrightarrow{\mathrm{p}}{}^* P'.M'$ and $Q.N = \downarrow P'.\downarrow M'$*
2. *$P \,\&\, M$ unifiable $\iff \exists\, Q.\ \downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q.\emptyset$*
3. *$\downarrow P.\downarrow M$ is don't care*
4. *if $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q_a.\emptyset$, then there exists a normal, inhabited and Π-admissible prefix Q such that $Q_a = \downarrow Q$, $Q \vdash^{\Pi} P \,\&\, M$, $\mathrm{U}[P \,\&\, M] \subseteq \mathrm{U}[Q]$, and $\emptyset_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, \pi M$.*

Proof. *1.* Follows by a straightforward induction using the Simulation and the Invariance Lemma.

2. The direction "$\Leftarrow$" follows by statement (1) and the first statement of the P-Reduction Theorem. To show the other direction, suppose $P.M$ is Π-well-typed and unifiable. We show by induction on $P.M$ with respect to $\xrightarrow{\mathrm{p}}$ that there exists a prefix Q such that $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q.\emptyset$.

If $M = \emptyset$, then the claim is trivial. Otherwise, we know by the Completeness and the Simulation Lemma that there exist P' and M' such that $P.M \xrightarrow{\mathrm{p}}{}^+ P'.M'$ and $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}} \downarrow P'.\downarrow M'$. By the Invariance Lemma we know that $P'.M'$ is Π-well-typed and unifiable. Hence we know by the induction hypothesis that there exists a prefix Q such that $\downarrow P'.\downarrow M' \xrightarrow{\mathrm{m}}{}^* Q.\emptyset$.

3. Let $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q_a.\emptyset$ and $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* P'.M'$. We have to show that $P'.M' \xrightarrow{\mathrm{m}}{}^* Q_a.\emptyset$.

By statement (1) we have $P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$, $Q_a = \downarrow Q$, $P.M \xrightarrow{\mathrm{p}}{}^* P''.M''$, $P' = \downarrow P''$, and $M' = \downarrow M''$. Since $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q_a.\emptyset$, we know by statement (2) that $P \,\&\, M$ is unifiable. Hence we know by the Invariance Lemma that $P'' \,\&\, M''$ is unifiable. Thus we have $P'.M' \xrightarrow{\mathrm{m}}{}^* Q'_a.\emptyset$ by statement (2) and hence $P''.M'' \xrightarrow{\mathrm{p}}{}^* Q'.\emptyset$ and $Q'_a = \downarrow Q'$ by statement (1). By statement (2.1) of the P-Reduction Theorem we have $Q = Q'$. Hence $Q_a = \downarrow Q = \downarrow Q' = Q'_a$.

4. Let $\downarrow P.\downarrow M \xrightarrow{\mathrm{m}}{}^* Q_a.\emptyset$. Then we know by statement (1) that there exists a prefix Q such that $Q_a = \downarrow Q$ and $P.M \xrightarrow{\mathrm{p}}{}^* Q.\emptyset$. Now the claims follow by the P-Reduction Theorem. □

6.6 Solving PE-Systems

General Assumption. *In this section we assume that T is fully inhabited.*

This section presents the part of the P-computation that is simulated by the equation level of the constraint solver. We start with the invariant that is preserved by the equation reduction rules employed in the P-computation.

A triple $P.\theta.E$ is **Π-well-typed** if

1. P is a normal, inhabited and Π-admissible prefix
2. θ is an idempotent value substitution such that $\mathcal{D}\theta \cap \mathcal{V}E = \emptyset$, $\mathcal{D}\theta \subseteq \mathcal{D}P$, and $P \vdash^{\Pi} \theta P$
3. for every equation $s \doteq t \in E$ there exists a Π-admissible sort term μ such that $P \vdash^{\Pi} s\colon \mu$ and $P \vdash^{\Pi} t\colon \mu$.

Proposition 6.6.1 *Π-well-typed triples have the following properties:*

1. *if $P.\theta.E$ is Π-well-typed, then $\mathcal{D}\theta \cup \mathcal{I}\theta \cup \mathcal{V}E \subseteq \mathcal{D}P$*
2. *if $P.\theta.\emptyset$ is Π-well-typed, then $\theta_{\mathrm{NF}[\pi P]}$ is a principal unifier of $\pi P \,\&\, \mathrm{E}[\theta]$*
3. *if $P.\theta.E \,\&\, x \doteq s$ is Π-well-typed, then $P.s\colon Px$ is Π-well-typed.*

The relation $\xrightarrow{\mathrm{u}}$ is defined by the same rules as $\xrightarrow{\mathrm{e}}$, except that the fourth rule of $\xrightarrow{\mathrm{u}}$ employs the relation $\xrightarrow{\mathrm{p}}{}^*$ rather than the relation $\xrightarrow{\mathrm{m}}{}^*$:

1. $P.\theta.E \,\&\, x \doteq x \;\xrightarrow{\mathrm{u}}\; P.\theta.E$
2. $P.\theta.E \,\&\, f(\vec{s}) \doteq f(\vec{t}) \;\xrightarrow{\mathrm{u}}\; P.\theta.E \,\&\, \vec{s} \doteq \vec{t}$
3. $P.\theta.E \,\&\, s \doteq x \;\xrightarrow{\mathrm{u}}\; P.\theta.E \,\&\, x \doteq s$
   if s is not a variable

4. $P.\theta.E \,\&\, x \doteq s \xrightarrow{\mathrm{u}} Q.\{x/s\}\theta.\{x/s\}E$
   if $x \notin \mathcal{V}s$ and $P.s{:}Px \xrightarrow{\mathrm{p}}{}^{*} Q.\emptyset$.

Proposition 6.6.2 *There are no infinite chains $P.\theta.E \xrightarrow{\mathrm{u}} P'.\theta'.E' \xrightarrow{\mathrm{u}} \cdots$. Furthermore, if $P.\theta.E \xrightarrow{\mathrm{u}} P'.\theta'.E'$, then $\mathcal{D}P = \mathcal{D}P'$.*

Proof. The termination of $\xrightarrow{\mathrm{u}}$ follows by the same argument as the termination of $\xrightarrow{\mathrm{e}}$ (Proposition 6.1.3). The second claim is obvious from the defining rules. □

Our goal is to prove Theorems 6.6.6 and 6.6.9 and Corollary 6.6.7 appearing at the end of this section. As in the last section I propose to first look at these theorems and understand what they say. Then you can go through the preparing lemmas following now.

Lemma 6.6.3 [Invariance] *Let $P.\theta.E$ be Π-well-typed and let $P.\theta.E \xrightarrow{\mathrm{u}} P'.\theta'.E'$. Then:*

1. *$P'.\theta'.E'$ is Π-well-typed*
2. *$P' \vdash^{\Pi} P$*
3. $\mathrm{U}[P \,\&\, \mathrm{E}[\theta] \,\&\, E] \subseteq \mathrm{U}[P' \,\&\, \mathrm{E}[\theta'] \,\&\, E']$
4. $\mathrm{U}[\pi P' \,\&\, \mathrm{E}[\theta'] \,\&\, E'] \subseteq \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E]$.

Proof. We have to prove the claims for each of the four $\xrightarrow{\mathrm{u}}$-rules.

I. Let $P.\theta.E \,\&\, x \doteq x \xrightarrow{\mathrm{u}} P.\theta.E$ and let $P.\theta.E \,\&\, x \doteq x$ be Π-well-typed. Then all claims are trivial.

II. Let $P.\theta.E \,\&\, f(\vec{s}) \doteq f(\vec{t}) \xrightarrow{\mathrm{u}} P.\theta.E \,\&\, \vec{s} \doteq \vec{t}$ and let $P.\theta.E \,\&\, f(\vec{s}) \doteq f(\vec{t})$ be Π-well-typed. Then there exists a Π-admissible sort term μ such that $P \vdash^{\Pi} f(\vec{s}){:}\,\mu$ and $P \vdash^{\Pi} f(\vec{t}){:}\,\mu$.

II.1. It suffices to show that there exists a tuple $\vec{\nu}$ of Π-admissible sort terms such that $P \vdash^{\Pi} \vec{s}{:}\,\vec{\nu}$ and $P \vdash^{\Pi} \vec{t}{:}\,\vec{\nu}$. Let $i := \min\{\lambda^{P}f(\vec{s}), \lambda^{Q}f(\vec{t})\}$. Then we know by the Flexibility Lemma that $P \vdash^{\Pi} f(\vec{s}){:}\,\pi_i\mu$ and $P \vdash^{\Pi} f(\vec{t}){:}\,\pi_i\mu$. Since $\pi_i\mu$ cannot be a variable, we have $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$, $\pi_i\mu \Rightarrow^{*} \xi(\vec{\sigma})$, $P \vdash^{\Pi} \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\mu$, and $P \vdash^{\Pi} \vec{t}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\mu$.

II.2 and II.3. Trivial.

II.4. Let $\psi_Q \in \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E \,\&\, \vec{s} \doteq \vec{t}]$. It suffices to show that $\psi f(\vec{s})$ is well-typed with respect to Q. Since $P \vdash^{\Pi} f(\vec{s}){:}\,\mu$, we know $\pi P \vdash f(\vec{s}){:}\,\pi\mu$. Since $Q \vdash \psi\pi P$, we have $Q \vdash \psi f(\vec{s}){:}\,\psi\pi\mu$.

III. Let $P.\theta.E \,\&\, s \doteq x \xrightarrow{\mathrm{u}} P.\theta.E \,\&\, x \doteq s$ and let $P.\theta.E \,\&\, s \doteq x$ be Π-well-typed. Then all claims are trivial.

IV. Let $P.\theta.E \,\&\, x \doteq s \xrightarrow{\mathrm{u}} Q.\{x/s\}\theta.\{x/s\}E$, $x \notin \mathcal{V}s$, $P.s{:}\,Px \xrightarrow{\mathrm{p}}{}^{*} Q.\emptyset$, and let $P.\theta.E \,\&\, x \doteq s$ be Π-well-typed. Then $\mathcal{D}\theta \cup \mathcal{I}\theta \cup \mathcal{V}E \cup \{x\} \cup \mathcal{V}s \subseteq \mathcal{D}P = \mathcal{D}Q$ and $Py = Qy$ for every $y \in \mathcal{D}P - \mathcal{V}s$. Furthermore, we know by Proposition 6.6.1 that $P.s{:}\,Px$ is Π-well-typed. Hence we know by the P-Reduction Theorem that Q is a normal, inhabited and

Π-admissible prefix such that $Q \vdash^{\Pi} P \,\&\, s{:}\,Px$, $\mathrm{U}[P \,\&\, s{:}\,Px] \subseteq \mathrm{U}[Q]$, and $\emptyset_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, s{:}\,\pi Px$. Thus $\mathrm{U}[\pi Q] = \mathrm{U}[\pi P \,\&\, s{:}\,\pi Px]$. Since $Q \vdash^{\Pi} P \,\&\, s{:}\,Px$, we have $Q \vdash^{\Pi} \{x/s\}P$. Since $P.\theta.E \,\&\, x \doteq s$ is Π-well-typed, we have $P \vdash^{\Pi} \theta P$ and thus $Q \vdash^{\Pi} \{x/s\}\theta P$ by the Substitution Lemma.

IV.1. Since $\{x\} \cup \mathcal{V}s$ and $\mathcal{D}\theta$ are disjoint and $x \notin \mathcal{V}s$, we know that $\{x/s\}\theta$ is an idempotent value substitution. Furthermore, $\mathcal{D}(\{x/s\}\theta)$ and $\mathcal{V}(\{x/s\}E)$ are disjoint since $x \notin \mathcal{V}s$ and $\mathcal{D}\theta$ and $\mathcal{V}E \cup \mathcal{V}s$ are disjoint. Moreover, $\mathcal{D}(\{x/s\}\theta) = \{x\} \cup \mathcal{D}\theta \subseteq \mathcal{D}P = \mathcal{D}Q$.

Next we show that $Q \vdash^{\Pi} \{x/s\}\theta Q$. Suppose $y \in \mathcal{D}Q$. We have to show that $Q \vdash^{\Pi} \{x/s\}\theta y{:}\,Qy$. If $y \in \mathcal{V}s$, then $y \neq x$ and $y \notin \mathcal{D}\theta$. Hence $\{x/s\}\theta y = y$ and the claim is trivial. If $y \notin \mathcal{V}s$, then $Qy = Py$ and hence $Q \vdash^{\Pi} \{x/s\}\theta y{:}\,Qy$ since $Q \vdash^{\Pi} \{x/s\}\theta P$.

Let $u \doteq v \in E$. We have to show that there exists a Π-admissible μ such that $Q \vdash^{\Pi} \{x/s\}u{:}\,\mu$ and $Q \vdash^{\Pi} \{x/s\}v{:}\,\mu$. Since $P.\theta.E \,\&\, x \doteq s$ is Π-well-typed, there exists a Π-admissible μ such that $P \vdash^{\Pi} u{:}\,\mu$ and $P \vdash^{\Pi} v{:}\,\mu$. Since $Q \vdash^{\Pi} \{x/s\}P$ we have $Q \vdash^{\Pi} \{x/s\}u{:}\,\mu$ and $Q \vdash^{\Pi} \{x/s\}v{:}\,\mu$ by the Substitution Lemma.

IV.2. We know $Q \vdash^{\Pi} P$ since $Q \vdash^{\Pi} P \,\&\, s{:}\,Px$.

IV.3. Let $\psi_R \in \mathrm{U}[P \,\&\, \mathrm{E}[\theta] \,\&\, E \,\&\, x \doteq s]$. Then $\psi x = \psi s$ and hence $\psi = \psi\{x/s\}$. Since $x \notin \mathcal{D}\theta$, we have $\mathrm{E}[\{x/s\}\theta] = \{x/s\}\mathrm{E}[\theta] \,\&\, x \doteq s$. Thus $\psi(\mathrm{E}[\{x/s\}\theta] \,\&\, \{x/s\}E) = \psi(\{x/s\}\mathrm{E}[\theta] \,\&\, x \doteq s \,\&\, \{x/s\}E) = \psi(\mathrm{E}[\theta] \,\&\, E \,\&\, x \doteq s)$. Furthermore, since $R \vdash \psi P$, we have $R \vdash \psi s{:}\,\psi Px$ and thus $\psi_R \in \mathrm{U}[P \,\&\, s{:}\,PX] \subseteq \mathrm{U}[Q]$. Hence $\psi_R \in \mathrm{U}[Q \,\&\, \mathrm{E}[\{x/s\}\theta] \,\&\, \{x/s\}E]$.

IV.4. Let $\psi_R \in \mathrm{U}[\pi Q \,\&\, \mathrm{E}[\{x/s\}\theta] \,\&\, \{x/s\}E]$. Since $x \notin \mathcal{D}\theta$, we have $\mathrm{E}[\{x/s\}\theta] = \{x/s\}\mathrm{E}[\theta] \,\&\, x \doteq s$. Thus $\psi(\mathrm{E}[\{x/s\}\theta] \,\&\, \{x/s\}E) = \psi(\{x/s\}\mathrm{E}[\theta] \,\&\, x \doteq s \,\&\, \{x/s\}E) = \psi(\mathrm{E}[\theta] \,\&\, E \,\&\, x \doteq s)$ since $\psi(\mathrm{E}[\{x/s\}\theta] \,\&\, \{x/s\}E)$ is trivial, hence $\psi x = \psi s$, and thus $\psi\{x/s\} = \psi$. Furthermore, we know $\mathrm{U}[\pi Q] = \mathrm{U}[\pi P \,\&\, s{:}\,\pi Px] \subseteq \mathrm{U}[\pi P]$. Hence $\psi_R \in \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E \,\&\, x \doteq s]$. □

Lemma 6.6.4 [Simulation] *Let $P.\theta.E$ be Π-well-typed and let $\downarrow P.\theta.E \xrightarrow{\mathrm{e}} Q.\theta'.E'$. Then there exists a prefix P' such that $Q = \downarrow P'$ and $P.\theta.E \xrightarrow{\mathrm{u}} P'.\theta'.E'$.*

Proof. If $\downarrow P.\theta.E \xrightarrow{\mathrm{e}} Q.\theta'.E'$ by one of the first three $\xrightarrow{\mathrm{e}}$-rules, then the claim is obvious since the first three $\xrightarrow{\mathrm{e}}$-rules are identical with the first three $\xrightarrow{\mathrm{u}}$-rules.

Otherwise, suppose $\downarrow P.\theta.E \,\&\, x \doteq s \xrightarrow{\mathrm{e}} Q.\{x/s\}\theta.\{x/s\}E$, $x \notin \mathcal{V}s$, $\downarrow P.s{:}\,\downarrow Px \xrightarrow{\mathrm{m}}{}^{*} Q.\emptyset$, and let $\downarrow P.\theta.E \,\&\, x \doteq s$ be Π-well-typed. Then we know by Proposition 6.6.1 that $P.s{:}\,Px$ is Π-well-typed. Hence we know by the M-Reduction Theorem that there exists a prefix P' such that $P.s{:}\,Px \xrightarrow{\mathrm{p}}{}^{*} P'.\emptyset$ and $Q = \downarrow P'$. Thus $P.\theta.E \,\&\, x \doteq s \xrightarrow{\mathrm{u}} P'.\{x/s\}\theta.\{x/s\}E$, which yields the claim. □

Lemma 6.6.5 [Completeness] *Let $P.\theta.E$ be Π-well-typed and let $P \,\&\, \mathrm{E}[\theta] \,\&\, E$ be unifiable. Then $\xrightarrow{\mathrm{e}}$ applies to $\downarrow P.\theta.E$.*

Proof. Without loss of generality we can assume that E consists of a single equation $s \doteq t$. Let $P.\theta.s \doteq t$ be Π-well-typed and let $\psi_Q \in \mathrm{U}[P.\theta.s \doteq t]$.

If $s = f(\vec{s})$ and $t = g(\vec{t})$, then $f = g$ since $\psi s = \psi t$, and hence the second $\xrightarrow{\mathrm{e}}$-rule applies.

If $s = f(\vec{s})$ and $t = x$, then the third $\xrightarrow{\mathrm{e}}$-rule applies.

If $s = t = x$, then the first $\xrightarrow{\mathrm{e}}$-rule applies.

Otherwise, we have $s = x \neq t$. Hence $x \notin \mathcal{V}t$ since $\psi x = \psi t$. Thus the fourth $\xrightarrow{\mathrm{e}}$-rule applies if we can show that there exists a prefix Q such that $\downarrow P.t{:}\downarrow Px \xrightarrow{\mathrm{m}}{}^* Q.\emptyset$.

Since $\psi_q \in \mathrm{U}[P.\theta.s \doteq t]$, we know that $Q \vdash \psi P$ and $\psi x = \psi t$. By Proposition 6.6.1 we know that $P.t{:}Px$ is Π-well-typed. Thus $x \in \mathcal{D}P$ and hence $Q \vdash \psi x{:}\psi Px$. Since $\psi x = \psi t$, we have $Q \vdash \psi t{:}\psi Px$ and thus $\psi_Q \in \mathrm{U}[P \,\&\, t{:}Px]$. Hence we know by the M-Reduction Theorem that there exists a prefix Q such that $\downarrow P.t{:}\downarrow Px \xrightarrow{\mathrm{m}}{}^* Q.\emptyset$. □

Theorem 6.6.6 [U-Reduction] *Let $P.\theta.E$ be Π-well-typed. Then:*

1. *$P \,\&\, \mathrm{E}[\theta] \,\&\, E$ is unifiable $\iff \exists\, Q, \psi.\ \ P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$*
2. *if $P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$, then*
   (a) *$Q \vdash^{\Pi} \psi Q$, $Q \vdash^{\Pi} P$, $\mathrm{U}[P \,\&\, \mathrm{E}[\theta] \,\&\, E] \subseteq \mathrm{U}[Q \,\&\, \mathrm{E}[\psi]]$, and $\psi_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E$*
   (b) *if $P.\theta.E \xrightarrow{\mathrm{u}}{}^* P'.\theta'.E'$, then $P'.\theta'.E'$ is Π-well-typed and there exist Q' and ψ' such that $P'.\theta'.E' \xrightarrow{\mathrm{u}}{}^* Q'.\psi'.\emptyset$.*

Proof. *1.* Suppose $P.\theta.E$ is Π-well-typed and $P \,\&\, \mathrm{E}[\theta] \,\&\, E$ is unifiable. We show by induction on $P.\theta.E$ with respect to $\xrightarrow{\mathrm{u}}$ that there exists Q and ψ such that $P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$. If $E = \emptyset$, then the claim is trivial. Otherwise, we know by the Completeness and the Simulation Lemma that there exist P', θ' and E' such that $P.\theta.E \xrightarrow{\mathrm{u}} P'.\theta'.E'$ and $\downarrow P.\theta.E \xrightarrow{\mathrm{e}} \downarrow P'.\theta'.E'$. By the Invariance Lemma we know that $P'.\theta'.E'$ is Π-well-typed and that $P' \,\&\, \mathrm{E}[\theta'] \,\&\, E'$ is unifiable. Hence we know by the induction hypothesis that there exist Q and ψ such that $P'.\theta'.E' \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$.

Suppose $P.\theta.E$ is Π-well-typed and $P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$. By the Invariance Lemma we know that $Q.\psi.\emptyset$ is Π-well-typed and that $\mathrm{U}[\pi Q \,\&\, \mathrm{E}[\psi]] \subseteq \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E]$. Hence $\psi_{\mathrm{NF}[\pi Q]}$ is a principal unifier and $\psi_{\mathrm{NF}[\pi Q]} \in \mathrm{U}[\pi Q \,\&\, \mathrm{E}[\psi]] \subseteq \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E]$. Thus $(\psi\pi)_{\mathrm{NF}[\pi Q]}$ is a unifier of $P \,\&\, \mathrm{E}[\theta] \,\&\, E$.

2. Let $P.\theta.E$ is Π-well-typed and $P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$.

2.1. By the Invariance and the Substitution Lemma we know that $Q \vdash^{\Pi} \psi Q$, $Q \vdash^{\Pi} P$, $\mathrm{U}[P \,\&\, \mathrm{E}[\theta] \,\&\, E] \subseteq \mathrm{U}[Q \,\&\, \mathrm{E}[\psi]]$, and $\mathrm{U}[\pi Q \,\&\, \mathrm{E}[\psi]] \subseteq \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E]$. Thus we know $\mathrm{U}[\pi Q \,\&\, \mathrm{E}[\psi]] = \mathrm{U}[\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E]$. Since $Q.\psi.\emptyset$ is Π-well-typed, we know that $\psi_{\mathrm{NF}[\pi q]}$ is a principal unifier of $\pi Q \,\&\, \mathrm{E}[\psi]$. Hence $\psi_{\mathrm{NF}[\pi q]}$ is a principal unifier of $\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E$.

2.2. Suppose $P.\theta.E \xrightarrow{\mathrm{u}}{}^* P'.\theta'.E'$. Then we know by the Invariance Lemma that $P'.\theta'.E'$ is Π-well-typed. Since we assumed $P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\psi.\emptyset$, we know by statement (1) that $P \,\&\, \mathrm{E}[\theta] \,\&\, E$ is unifiable. Thus we know by the Invariance Lemma that $P' \,\&\, \mathrm{E}[\theta'] \,\&\, E'$ is unifiable. Hence we know by statement (1) that there exist Q' and ψ' such that $P'.\theta'.E' \xrightarrow{\mathrm{u}}{}^* Q'.\psi'.\emptyset$. □

The following corollary is a weak version of the U-Reduction Theorem obtained by employing the trivial pyramid. What we get is POS-unification, which is the natural generalization of order-sorted unification.

Corollary 6.6.7 [POS-Unification] *Let P be a normal and inhabited prefix and let E be an equation system that is well-typed under P. Then:*

1. *$P \,\&\, E$ is unifiable $\iff \exists\, Q, \psi.\ P.\theta.E \xrightarrow{\mathrm{u}}{}^{*} Q.\psi.\emptyset$*
2. *$P \,\&\, E$ is unifiable $\iff$ $P \,\&\, E$ has a principal unifier*
3. *if $P.\emptyset.E \xrightarrow{\mathrm{u}}{}^{*} Q.\psi.\emptyset$, then ψ_Q is a principal unifier of $P \,\&\, E$*
4. *if $P.\emptyset.E \xrightarrow{\mathrm{u}}{}^{*} Q.\psi.\emptyset$ and $P.\emptyset.E \xrightarrow{\mathrm{u}}{}^{*} P'.\theta'.E'$, then $P' \vdash \theta' P'$, E' is well-typed under P', and there exist Q' and ψ' such that $P'.\theta'.E' \xrightarrow{\mathrm{u}}{}^{*} Q'.\psi'.\emptyset$.*

Theorem 6.6.8 [Simulation] *Let $P.\theta.E$ be Π-well-typed and let $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} Q.\theta'.E'$. Then there exists a prefix P' such that $P.\theta.E \xrightarrow{\mathrm{u}}{}^{*} P'.\theta'.E'$ and $Q = \downarrow P'$.*

Proof. Follows by a straightforward induction using the Simulation and the Invariance Lemma. □

Theorem 6.6.9 [E-Reduction] *Let $P.\theta.E$ be Π-well-typed. Then:*

1. *$P \,\&\, \mathrm{E}[\theta] \,\&\, E$ is unifiable $\iff \exists\, Q, \psi.\ \downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$*
2. *if $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} P'.\theta'.E' \,\&\, x \doteq s$, then $P'.s{:}P'x$ is don't care*
3. *if $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$ and $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} P'.\theta'.E'$, then there exist Q' and ψ' such that $P'.\theta'.E' \xrightarrow{\mathrm{e}}{}^{*} Q'.\psi'.\emptyset$*
4. *if $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} Q_a.\psi.\emptyset$, then there exists a normal, inhabited and Π-admissible prefix Q such that $Q_a = \downarrow Q$, $Q \vdash^{\Pi} \psi Q$, $Q \vdash^{\Pi} P$, $\mathrm{U}[P \,\&\, \mathrm{E}[\theta] \,\&\, E] \subseteq \mathrm{U}[Q \,\&\, \mathrm{E}[\psi]]$, and $\psi_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, \mathrm{E}[\theta] \,\&\, E$.*

Proof. *1.* The direction "$\Leftarrow$" follows by the Simulation Theorem and the first statement of the U-Reduction Theorem. To show the other direction, suppose $P.\theta.E$ is Π-well-typed and $P \,\&\, \mathrm{E}[\theta] \,\&\, E$ is unifiable. We show by induction on $P.\theta.E$ with respect to $\xrightarrow{\mathrm{u}}$ that there exist Q and ψ such that $\downarrow P.\theta.E \xrightarrow{\mathrm{e}} Q.\psi.\emptyset$.

If $E = \emptyset$, then the claim is trivial. Otherwise, we know by the Completeness and the Simulation Lemma that there exist P', θ' and E' such that $P.\theta.E \xrightarrow{\mathrm{u}} P'.\theta'.E'$ and $\downarrow P.\theta.E \xrightarrow{\mathrm{e}} \downarrow P'.\theta'.E'$. By the Invariance Lemma we know that $P'.\theta'.E'$ is Π-well-typed and that $P' \,\&\, \mathrm{E}[\theta'] \,\&\, E'$ is unifiable. Hence we know by the induction hypothesis that there exist Q and ψ such that $\downarrow P.\theta.E \xrightarrow{\mathrm{e}} Q.\psi.\emptyset$.

2. Let $P.\theta.E$ be Π-well-typed and let $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} P'.\theta'.E' \,\&\, x \doteq s$. Then we know by the Simulation Theorem that there exist a prefix Q such that $P.\theta.E \xrightarrow{\mathrm{u}}{}^{*} Q.\theta'.E' \,\&\, x \doteq s$ and $P' = \downarrow Q$. By the Invariance Lemma we know that $Q.\theta'.E' \,\&\, x \doteq s$ is Π-well-typed. Hence we know by Proposition 6.6.1 that $Q.s{:}Qx$ is Π-well-typed. Thus we know by the M-Reduction Theorem that $\downarrow Q.s{:}\downarrow Qx = P'.s{:}P'x$ is don't care.

3. Let $P.\theta.E$ be Π-well-typed, $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$, and $\downarrow P.\theta.E \xrightarrow{\mathrm{e}}{}^{*} P'.\theta'.E'$. Then we know by the first statement that $P.\theta.E$ is unifiable. By the Simulation Theorem we know

that there exist a prefix Q such that $P.\theta.E \xrightarrow{\mathrm{u}}{}^* Q.\theta'.E'$ and $P' = \downarrow Q$. By the Invariance Lemma we know that $Q.\theta'.E'$ is Π-well-typed and that $Q \,\&\, \mathrm{E}[\theta'] \,\&\, E'$ is unifiable since $P.\theta.E$ is unifiable. Hence we know by the first statement there exist Q' and ψ' such that $P'.\theta'.E' \xrightarrow{\mathrm{e}}{}^* Q'.\psi'.\emptyset$.

4. Follows by the Simulation Theorem and the U-Reduction Theorem. □

6.7 Proof of the Hauptsatz

Now we are ready to enjoy the rewards of our hard work and prove the Hauptsatz.

Proof. Let $P \,\&\, E$ be V-admissible and let Π be a validating pyramid. Then $P.\emptyset.E$ is Π-well-typed and all except the second claim of the Hauptsatz follow immediately from corresponding statements of the E-Reduction Theorem.

Now let $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^* Q_a.\psi.\emptyset$. Then we know by the E-Reduction Theorem that there exists a normal, inhabited and Π-admissible prefix Q such that $Q_a = \downarrow Q$, $Q \vdash^{\Pi} \psi Q$, $Q \vdash^{\Pi} P$, $\mathrm{U}[E \,\&\, P] \subseteq \mathrm{U}[Q \,\&\, \mathrm{E}[\psi]]$, and $\psi_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, E$.

Let $W := (V - \mathcal{D}\psi) \cup \mathcal{I}(\psi|_V)$.

Since $Q \vdash^{\Pi} P$ and $P|_V = \pi P|_V$, we know that $\pi Q|_V = Q|_V \leq P|_V$. Since $Q \vdash^{\Pi} \psi Q$, we hence know by the Monotonicity Lemma that $\pi Q|_W = Q|_W$. Since $Q|_V \leq P|_V$, we know by the Upper Weakening Lemma that $Q|_W \vdash^{\Pi} \psi P|_V$. Hence $\pi Q|_W \vdash \pi\psi P|_V$ and thus $Q|_W \vdash \psi P|_V$ since $Q|_W$ and $\psi P|_V$ are fully informed. Hence we have $Q|_W \leq \mathrm{GP}[\psi P|_V]$.

Since $Q_a = \downarrow Q$ and Q is normal, we know by the Corollary to the Retract Theorem that

$$Q|_W = Q_a \uparrow \mathrm{GP}[\psi P|_V].$$

Since $\psi_{\mathrm{NF}[\pi Q]}$ is a principal unifier of $\pi P \,\&\, E$, we know by the Garbage Collection Theorem that $(\psi|_V)_{(\mathrm{NF}[\pi Q]|_W)}$ is a V-solution schema for $\pi P \,\&\, E$. Since Q is normal and $Q|_W = \pi Q|_W$, we know that $(\psi|_V)_{(Q|_W)}$ is a V-solution schema and

$$\mathcal{T}[\![\uparrow[Q, \psi, P|_V]]\!]^V = \mathcal{T}[\![\mathrm{E}[\psi|_V] \,\&\, Q|_W]\!]^V = \mathcal{T}[\![\pi P \,\&\, E]\!]^V.$$

It remains to show that $\mathcal{T}[\![\mathrm{E}[\psi|_V] \,\&\, Q|_W]\!]^V = \mathcal{T}[\![E \,\&\, P]\!]^V$.

Since $\mathrm{U}[E \,\&\, P] \subseteq \mathrm{U}[\mathrm{E}[\psi] \,\&\, Q]$ and $\mathrm{E}[\psi|_V] \,\&\, Q|_W \subseteq \mathrm{E}[\psi] \,\&\, Q$, we have $\mathcal{T}[\![E \,\&\, P]\!]^V \subseteq \mathcal{T}[\![\mathrm{E}[\psi] \,\&\, Q]\!]^V \subseteq \mathcal{T}[\![\mathrm{E}[\psi|_V] \,\&\, Q|_W]\!]^V$. Furthermore, since π is a sort substitution and V is a set of value variables, we have $\mathcal{T}[\![E \,\&\, \pi P]\!]^V \subseteq \mathcal{T}[\![E \,\&\, P]\!]^V$ by Proposition 5.3.7 and hence $\mathcal{T}[\![\mathrm{E}[\psi|_V] \,\&\, Q|_W]\!]^V \subseteq \mathcal{T}[\![E \,\&\, P]\!]^V$. □

Chapter 7

Logic Programming over POS-Types

7.1 POS-Programs
7.2 The Interpreter
7.3 Type Inference

Now that we have gone trough four chapters on POS-types developing their declarative and operational semantics, we are well-prepared for the final step in our enterprise. The POS-programs defined in this chapter rely on a type discipline that is considerably more restrictive than the canonical notion of well-typedness developed in Chapter 2. While the canonical type discipline would result in ad hoc polymorphism, the type discipline imposed on POS-programs yields parametric polymorphism, which is the form of polymorphism employed in functional programming languages such as ML. There is a good reason for insisting on parametric polymorphism: the constraint solver developed in Chapter 6 works only for this less general form of polymorphism, and I even don't know if there is a computable complete constraint solver for programs relying on ad hoc polymorphism.

The last section of this chapter addresses type checking and inference. I will show that it is decidable whether some finite collection of syntactic objects is a POS-program and thus solve the type checking problem. Type inference generalizes type checking in that the programmer is allowed to omit some or all sort qualifications for the variables employed in a clause and the system is expected to infer most general sort qualifications under which the clause becomes well-typed. Here I cannot offer a perfect solution, but only give an incomplete algorithm, whose implementation in TEL works quite well for practical applications. The programmer can always prevent the type inference algorithm from failing by explicitly

giving the sort qualifications for "critical" variables.

7.1 POS-Programs

This section defines the class of relational programs over POS-types that can be executed using the constraint solver of Chapter 6. The well-typedness condition imposed prevents ad hoc polymorphism and enforces parametric polymorphism [Str67], the form of polymorphism realized in functional programming languages such as ML [HMM86] and in Mycroft and O'Keefe's [MO84] polymorphic type discipline for Prolog. Using the results of Chapter 2 we show that POS-programs do have least models satisfying the sort declarations of the relations. Finally, we give a tailored goal reduction rule for POS-programs and show its soundness and completeness exploiting the general results of Chapter 2.

General Assumption. *In this chapter we assume that T is a fully inhabited type specification and that R is a set of relation symbols.*

Let $\mathcal{L}(T)$ be the constraint language obtained from $\mathcal{L}(\Sigma^T)^*$ by forgetting all interpretations but $\mathcal{T}$. In the following we assume tacitly that all constraints and all interpretations are taken from $\mathcal{L}(T)^*_R$.

Of course, we could also take the initial interpretation $\mathcal{I}(T)$ rather than the extensional interpretation $\mathcal{T}$ of T. Since our constraint solver works for both interpretations in exactly the same way, all results of this chapter stay true if $\mathcal{I}(T)$ instead of $\mathcal{T}$ is taken as the base interpretation.

A **POS-declaration** is a declaration $r(\vec{x}) \rightarrow_{\exists} \vec{x} : \vec{\sigma}$ such that $\vec{\sigma}$ is a tuple of normal and inhabited sort terms. Since the validity of a POS-declaration in an interpretation does not depend on the particular variables employed, we will use the abbreviation $r : \vec{\sigma}$. A set of POS-declarations is called **singular** if it contains for no relation symbol more than one declaration.

Proposition 7.1.1 *If $\mathcal{A}$ is an interpretation and $r : \vec{\sigma}$ is a POS-declaration, then $\mathcal{A}$ satisfies $r : \vec{\sigma}$ if and only if $r^{\mathcal{A}} \subseteq \bigcup\{\mathcal{T}[\![\vec{\sigma}]\!]_\delta \mid \delta \in \mathrm{ASS}^{\mathcal{T}}\}$.*

A **POS-atom** is a constraint $\exists \vec{x}.(\vec{x} \doteq \vec{s} \,\&\, r(\vec{x}))$ such that $\vec{s}$ has no variable in common with $\vec{x}$. Since the denotation of a POS-atom does not depend on the particular variables in $\vec{x}$, we use the abbreviation $r(\vec{s})$. In the following, the letters A and B will always denote a POS-atom. Furthermore, the letters G and H will always denote a possibly empty conjunction of POS-atoms.

Proposition 7.1.2 *If $\mathcal{A}$ is an interpretation and $r(\vec{s})$ is a POS-atom, then $\mathcal{A}[\![r(\vec{s})]\!] = \{\delta \in \mathrm{ASS}^{\mathcal{T}} \mid \mathcal{T}[\![\vec{s}]\!]_\delta \in r^{\mathcal{A}}\}$.*

A **POS-clause** is an implication $A \leftarrow P \,\&\, A_1 \,\&\, \ldots \,\&\, A_n$ such that $n \geq 0$, P is a normal and inhabited prefix, and A and $A_1, \ldots, A_n$ are POS-atoms.

Proposition 7.1.3 *An interpretation satisfies the POS-clause*

$$r_0(\vec{s}_0) \leftarrow P \,\&\, r_1(\vec{s}_1) \,\&\, \ldots \,\&\, r_n(\vec{s}_n)$$

if and only if it satisfies the definite clause

$$r_0(\vec{x}_0) \leftarrow P \,\&\, \{\vec{x}_i \doteq \vec{s}_i\}_{i=0}^{n} \,\&\, r_1(\vec{x}_1) \,\&\, \ldots \,\&\, r_n(\vec{x}_n),$$

provided the sets $\mathcal{V}\vec{x}_0, \ldots, \mathcal{V}\vec{x}_n$ are pairwise disjoint and have no variable in common with P and $\vec{s}_0, \ldots, \vec{s}_n$.

Let D be a singular set of POS-declarations. A POS-atom $r(\vec{s})$ is **POS-well-typed** under a prefix P with respect to D if there exists a declaration $r{:}\vec{\sigma} \in D$ and a substitution θ such that $P \vdash \vec{s}{:}\theta\vec{\sigma}$. A POS-clause $r(\vec{s}) \leftarrow P \,\&\, A_1 \,\&\, \ldots \,\&\, A_n$ is **POS-well-typed** with respect to D if the POS-atoms $A_1, \ldots, A_n$ are POS-well-typed under P with respect to D and there exists a variant $r{:}\vec{\sigma}$ of a declaration in D such that $P \vdash \vec{s}{:}\vec{\sigma}$.

Our definition of well-typed clauses is considerably more restrictive than the general definition in Chapter 2 since the head of the clause is required to satisfy a variant of the declaration rather than an instance. For instance, the program

$$a{:}A, \quad r{:}\alpha, \quad r(a) \leftarrow \emptyset$$

is well-typed under the general definition of Chapter 2 but is not POS-well-typed. Our notion of POS-well-typedness agrees with the notion of well-typedness in polymorphically typed functional programming languages such as ML [HMM86] and in Mycroft and O'Keefe's [MO84] polymorphic type discipline for Prolog. This strong notion of well-typedness is necessary to obtain "parametric polymorphism" rather than "ad hoc polymorphism" [Str67]. It is also crucial for our constraint solver to work with our programs. (In fact, I don't know whether the satisfiability of the constraints produced by "weakly well-typed" programs is decidable.) Interestingly, Hanus [Han88, Han89] investigates a polymorphic type discipline for Horn Logic that employs the general notion of well-typedness of Chapter 2 and thus admits "ad hoc polymorphism". Hanus' system doesn't accommodate subsorting.

A **POS-program** is a pair (D, C) consisting of a singular set D of POS-declarations and a set C of POS-clauses that are POS-well-typed with respect to D.

Theorem 7.1.4 *Let $S = (D, C)$ be a POS-program. Then C has a least model and the least model of C satisfies D.*

Proof. By Proposition 7.1.3 we know that we can obtain from C a set $\bar{C}$ of definite clauses such that an interpretation satisfies C if and only if it satisfies $\bar{C}$. Since all interpretations have $\mathcal{T}$ as base, we know by the Definiteness Theorem of Chapter 2 that $\bar{C}$ and hence C have a least model.

One verifies easily that $\bar{C}$ is well-typed with respect to D (as defined in Section 2.5). Hence we know by Theorem 2.5.5 that the least model of C, which is also the least model of $\bar{C}$, satisfies D. □

To obtain a convenient notation, we fix some POS-program (D, C) and use $\mathcal{S}$ to denote its least model. Moreover, the letter V will always denote a finite set of value variables.

The following rule defines a binary relation "$F \xrightarrow{\mathrm{h}} F'$", called **H-reduction**, on constraints:

$$\begin{array}{rcl} F \,\&\, r(\vec{s}) & \xrightarrow{\mathrm{h}} & F \,\&\, \vec{s} \doteq \vec{t} \,\&\, F' \\ & \text{if} & r(\vec{t}) \leftarrow F' \text{ is a variant of a clause in } C \\ & & \text{having no variables in common with } F \,\&\, r(\vec{s}). \end{array}$$

Proposition 7.1.5 [Soundness] *If $F \xrightarrow{\mathrm{h}}{}^{*} F'$, then $\mathcal{S}[\![F']\!] \subseteq \mathcal{S}[\![F]\!]$.*

A **POS-goal** is a conjunction $P \,\&\, G$ such that P is an inhabited and normal prefix and G is a conjunction of POS-atoms that are POS-well-typed under P with respect to D.

Theorem 7.1.6 [Completeness] *Let $P \,\&\, G$ be a POS-goal and $\delta \in \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{D}P}$. Then there exists a complexity measure "$\|F\|$" such that*

1. *$\|P \,\&\, G\|$ is defined*
2. *if $\|F\|$ is defined, then $\delta \in \mathcal{S}[\![F]\!]^{\mathcal{D}P}$*
3. *if $P \,\&\, G \xrightarrow{\mathrm{h}}{}^{*} F$, $\|F\|$ is defined, and A is a POS-atom in F, then there exists a clause $\gamma \in C$ such that*
   (a) *H-reduction of F on A with γ is possible*
   (b) *if F' is obtained from F by H-reduction on A with γ, then $\|F'\| < \|F\|$.*

Proof. Let $\bar{C}$ be obtained from C be replacing every clause with an equivalent definite clause according to Proposition 7.1.3. Then $\bar{C}$ is a definite clause specification in the sense of Chapter 2. The idea is to simulate $\xrightarrow{\mathrm{h}}$ with $\xrightarrow{\mathrm{r}}_{\bar{C},\mathcal{D}P} \circ \xrightarrow{\mathrm{c}}_{\mathcal{D}P}$ and to obtain the claims from the general Completeness Theorem 2.4.5.

Recall that a POS-atom $r(\vec{s})$ is actually a constraint $\exists \vec{x}.(\vec{x} \doteq \vec{s} \,\&\, r(\vec{x}))$. Now let $\bar{G}$ be obtained from G by omitting all quantifiers, that is, by replacing every POS-atom $\exists \vec{x}.(\vec{x} \doteq \vec{s} \,\&\, r(\vec{x}))$ with $\vec{x} \doteq \vec{s} \,\&\, r(\vec{x})$, where the existentially quantified variables $\vec{x}$ are renamed such that they occur nowhere else. Then $\mathcal{S}[\![G]\!]^{\mathcal{D}P} = \mathcal{S}[\![\bar{G}]\!]^{\mathcal{D}P}$ and $\bar{G}$ is a goal in the sense of Chapter 2.

Now let $G = G_1 \,\&\, r(\vec{s})$ and $\bar{G} = \bar{G}_1 \,\&\, \vec{x} \doteq \vec{s} \,\&\, r(\vec{x})$. Furthermore, suppose

$$P \,\&\, G_1 \,\&\, r(\vec{s}) \xrightarrow{\mathrm{h}} P \,\&\, P' \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G_2,$$

where $r(\vec{t}) \leftarrow P' \,\&\, G_2$ is a variant of a clause in C having no variables in common with $P \,\&\, G_1 \,\&\, r(\vec{s})$. Then there exists a variant $r(\vec{x}) \leftarrow \vec{x} \doteq \vec{t} \,\&\, P' \,\&\, \bar{G}_2$ of a clause in $\bar{C}$ having no variables in common with $\mathcal{V}(P \,\&\, \bar{G}_1) \cup \mathcal{V}\vec{s}$ such that $\bar{G}_2$ can be obtained from G_2 in the same way $\bar{G}$ was obtained from G. Hence

$$\begin{aligned} P \,\&\, \bar{G}_1 \,\&\, \vec{x} \doteq \vec{s} \,\&\, r(\vec{x}) &\xrightarrow{\mathrm{r}}_{\bar{C},\mathcal{D}P} P \,\&\, P' \,\&\, \vec{x} \doteq \vec{s} \,\&\, \vec{x} \doteq \vec{t} \,\&\, \bar{G}_1 \,\&\, \bar{G}_2 \\ &\xrightarrow{\mathrm{c}}_{\mathcal{D}P} P \,\&\, P' \,\&\, \vec{s} \doteq \vec{t} \,\&\, \bar{G}_1 \,\&\, \bar{G}_2. \end{aligned}$$

□

That's as much as we can get out of the general framework of Chapter 2. We cannot integrate the constraint solver of Chapter 6 directly since it is not incremental in the strong sense required by the general constraint solving rule. In particular, this failure is caused by the fact that the constraint solver doesn't compute with the "real" constraints but only with their approximations. We will see in the next section that our constraint solver, nevertheless, can do the job perfectly.

7.2 The Interpreter

As in the last section, we assume that a fully inhabited type specification T, a set R of relation symbols, and a POS-program (C, D) in $\mathcal{L}(T)^*_R$ are given. All constraints and interpretations are taken from $\mathcal{L}(T)^*_R$. Furthermore, let $\mathcal{S}$ be the least model of C and V be a finite set of value variables.

Our final interpreter for POS-programs consists of a single rule, called **G-reduction**:

$$\begin{array}{lll} P.\theta.G \,\&\, r(\vec{s}) & \xrightarrow{\mathrm{g}} & Q.\psi.G \,\&\, G' \\ & \text{if} & r(\vec{t}) \leftarrow P' \,\&\, G' \text{ is a variant of a clause in } C \text{ such that} \\ & & \mathcal{D}P \cap DP' = \emptyset \text{ and } (P \,\&\, {\downarrow}P').\theta.\theta\vec{s} \doteq \vec{t} \xrightarrow{\mathrm{e}}{}^* \; Q.\psi.\emptyset. \end{array}$$

Let $P \,\&\, G$ be a POS-goal. We will show that G-reduction is sound, that is,

$$\downarrow P.\emptyset.G \xrightarrow{\mathrm{g}}{}^* Q.\psi.\emptyset \quad \Rightarrow \quad \mathcal{T}[\![\uparrow[Q, \psi, P]]\!]^{\mathcal{D}P} \subseteq \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{D}P},$$

and that G-reduction is complete, that is,

$$\delta \in \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{D}P} \;\Rightarrow\; \exists\, Q, \psi.\; \downarrow P.\emptyset.G \xrightarrow{\mathrm{g}}{}^* Q.\psi.\emptyset \;\wedge\; \delta \in \mathcal{T}[\![\uparrow[Q, \psi, P]]\!]^{\mathcal{D}P}.$$

Of course, we will show a stronger completeness result saying that only the choice of the clause is don't know and that all other involved choices are don't care (in particular, during constraint solving).

The proof idea is simple: we will show that every $\xrightarrow{\mathrm{g}}$-derivation issuing from a POS-goal can be simulated by a $\xrightarrow{\mathrm{h}}$-derivation, and that enough $\xrightarrow{\mathrm{h}}$-derivations can be anticipated to be complete. The conjunctions of the $\xrightarrow{\mathrm{h}}$-derivation will satisfy a sufficiently strong invariant ensuring that E-reduction is only applied to constraint systems that are admissible in the sense of the Hauptsatz.

A conjunction $P \,\&\, E \,\&\, G$ is **V-admissible** if there exists a pyramid Π such that

1. P is a normal, inhabited and Π-admissible prefix such that $V \subseteq \mathcal{D}P$ and $P|_V = \pi P|_V$
2. E is an equation system such that for every equation $s \doteq t \in E$ there exists a Π-admissible sort term μ such that $\mathcal{V}\mu \subseteq \mathcal{D}\pi$, $P \vdash s\colon \pi_{\lambda P_s}\mu$, and $P \vdash t\colon \mu$
3. G is a conjunction of POS-atoms such that every POS-atom $r(\vec{s}) \in G$ is POS-well-typed under P with respect to D and $\lambda Px = \lambda Py$ for any two variables $x, y \in \mathcal{V}\vec{s}$.

Proposition 7.2.1 *If $P \,\&\, G$ is a POS-goal and $V \subseteq \mathcal{D}P$, then $P \,\&\, G$ is V-admissible. Furthermore, if $P \,\&\, E \,\&\, G$ is V-admissible, then $P \,\&\, E$ is V-admissible (as defined in Subsection 6.1.4).*

We now define so-called V-pairs relating the triples of the $\xrightarrow{\mathrm{g}}$-derivation with the "hypothetical" conjunctions of the simulating $\xrightarrow{\mathrm{h}}$-derivation. A pair $<Q.\psi.G,\, P \,\&\, E \,\&\, G>$ is called a **V-pair** if $P \,\&\, E \,\&\, G$ is V-admissible and $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$. Note that the V-admissibility of $P \,\&\, E \,\&\, G$ implies that $P \,\&\, E$ is V-admissible as required by the Hauptsatz.

The next lemma states the two crucial properties of V-pairs, where the second property is definitely nontrivial and follows from the Hauptsatz. This is one of the two places where the Hauptsatz is needed in the proofs to come.

Lemma 7.2.2 [V-Pair] *If $P \,\&\, G$ is a POS-goal, then $<\downarrow P.\emptyset.G,\, P \,\&\, G>$ is a $\mathcal{D}P$-pair. Furthermore, if $<Q.\psi.G,\, P \,\&\, E \,\&\, G>$ is a V-pair, then $\mathcal{T}[\![P \,\&\, E]\!]^V = \mathcal{T}[\![\uparrow[Q, \psi, P|_V]]\!]^V$.*

Proof. The first claim is obvious from the definitions and the preceding proposition. To show the second claim, let $<Q.\psi.G,\, P \,\&\, E \,\&\, G>$ be a V-pair. Then $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$ and $P \,\&\, E$ is V-admissible since $P \,\&\, E \,\&\, G$ is V-admissible. Hence we know by the Hauptsatz that $\mathcal{T}[\![P \,\&\, E]\!]^V = \mathcal{T}[\![\uparrow[Q, \psi, P|_V]]\!]^V$. □

Next we define so-called $\longrightarrow_V$-derivations on V-pairs relating $\xrightarrow{\mathrm{g}}$-derivations with their simulating $\xrightarrow{\mathrm{h}}$-derivations:

$$\begin{array}{lll} <?\,,\, F> & \longrightarrow_V & <?\,',\, F'> \\ & \text{if} & <?\,,\, F> \text{ and } <?\,',\, F'> \text{ are } V\text{-pairs,} \\ & & ? \xrightarrow{\mathrm{g}} ?\,', \text{ and } F \xrightarrow{\mathrm{h}} F'. \end{array}$$

Proposition 7.2.3 *Let $P \,\&\, G$ be a POS-goal and $<\downarrow P.\emptyset.G,\, P \,\&\, G> \longrightarrow_{\mathcal{D}P}^{*} <Q_a.\psi.\emptyset,\, Q \,\&\, E>$. Then*

$$\mathcal{T}[\![\uparrow[Q_a, \psi, P]]\!]^{\mathcal{D}P} = \mathcal{T}[\![Q \,\&\, E]\!]^{\mathcal{D}P} \subseteq \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{D}P}.$$

Proof. Follows from the soundness of H-reduction and the V-Pair Lemma since $Q|_{\mathcal{D}P} = P$. □

Next we will show that every $\xrightarrow{\mathrm{g}}$-reduction issuing from a POS-goal $P \,\&\, G$ can be extended to a $\longrightarrow_{\mathcal{D}P}$-derivation. This yields the soundness of G-reduction. The proof will force us for the very last time to get involved with the tedious details of pyramid well-typedness.

Following a $\xrightarrow{\mathrm{g}}$-derivation issuing from a POS-goal $P \,\&\, G$, one can build up incrementally the simulating $\xrightarrow{\mathrm{h}}$-derivation together with the pyramids validating its $\mathcal{D}P$-admissibility using the following lemma.

Lemma 7.2.4 [Pyramid Construction] *Let $\Pi = \pi_n \cdots \pi_0$ be a pyramid and ψ be an idempotent sort substitution such that $\mathcal{D}\psi$ and $\mathcal{D}\pi_n \cup \mathcal{I}\pi_n$ are disjoint. Then:*

1. *if* $\mathcal{I}\psi \subseteq \mathrm{SV}_0$, *then*

$$\Pi' = (\pi_n\psi)\cdots(\pi_0\psi)\emptyset$$

is a pyramid such that

$$\lambda'\alpha = \begin{cases} 0 & \text{if } \alpha \in \mathcal{D}\psi \\ \lambda\alpha + 1 & \text{otherwise} \end{cases} \quad \text{and} \quad \pi'_{i+1}\alpha = \begin{cases} \pi_i\psi\alpha & \text{if } \alpha \in \mathcal{D}\psi \\ \pi_i\alpha & \text{otherwise} \end{cases}$$

2. *if* $\mathcal{I}\psi \subseteq \mathrm{SV}_k$ *and* $k \in 1..n$, *then*

$$\Pi' = (\pi_n\psi)\cdots(\pi_k\psi)\pi_{k-1}\cdots\pi_0$$

is a pyramid such that

$$\lambda'\alpha = \begin{cases} k-1 & \text{if } \alpha \in \mathcal{D}\psi \\ \lambda\alpha & \text{otherwise} \end{cases} \quad \text{and} \quad \pi'_i\alpha = \begin{cases} \pi_i\psi\alpha & \text{if } \alpha \in \mathcal{D}\psi \text{ and } i \geq k \\ \pi_i\alpha & \text{otherwise.} \end{cases}$$

Proof. Tedious but straightforward. □

Lemma 7.2.5 *Let* $P \,\&\, E \,\&\, G \,\&\, r(\vec{s})$ *be* V*-admissible,* $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^* Q.\psi.\emptyset$, *and* $r(\vec{t}) \leftarrow P' \,\&\, G'$ *be a variant of a clause* $\gamma \in C$ *such that* $\mathcal{D}Q$ *and* $\mathcal{D}P'$ *are disjoint. Then there exists a variant* $r(\vec{t}) \leftarrow P'' \,\&\, G'$ *of* γ *having no variables in common with* P *such that*

1. $P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G \,\&\, G'$ *is* V*-admissible*
2. $\downarrow P \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \xrightarrow{\mathrm{e}}{}^* Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t}$.

Proof. *1.* Let $\Pi = \pi_n \cdots \pi_0$ be a pyramid such that: P is Π-admissible, $P|_V = \pi P|_V$ and $V \subseteq \mathcal{D}P$; for every $s \doteq t \in E$ there exist a Π-admissible μ such that $\mathcal{V}\mu \subseteq \mathcal{D}\pi$, $P \vdash s{:}\,\pi_{\lambda^P_s}\mu$ and $P \vdash t{:}\,\mu$; and $\lambda Px = \lambda Py$ for every $r(\vec{u}) \in G \,\&\, r(\vec{s})$ and for every two variables $x, y \in \mathcal{V}\vec{u}$.

Furthermore, let $r(\vec{t}) \leftarrow P'' \,\&\, G'$ be a variant of γ such that $\mathcal{D}P'' = \mathcal{D}P'$ and $r(\vec{t}) \leftarrow P'' \,\&\, G'$ has no variable in common with $\mathcal{V}P \cup \mathcal{D}\pi \cup \mathcal{I}\pi$. Such a variant can be obtained by renaming the sort variables of P' since $r(\vec{t}) \leftarrow P' \,\&\, G'$ has no value variable in common with $\mathcal{D}P = \mathcal{D}(\downarrow P) = \mathcal{D}Q$.

Finally, let $r{:}\,\vec{\sigma}$ be a variant of a declaration in D such that $P'' \vdash \vec{t}{:}\,\vec{\sigma}$ and $\vec{\sigma}$ has no variable in common with $\mathcal{V}P \cup \mathcal{D}\pi \cup \mathcal{I}\pi$.

Since $r(\vec{s})$ is POS-well-typed under P with respect to D, there exists a sort substitution ψ such that $P \vdash \vec{s}{:}\,\psi\vec{\sigma}$. Since T is fully inhabited, we can assume without loss of generality that ψ is inhabited, $\mathcal{D}\psi = \mathcal{V}\vec{\sigma} \cup \bigcup_{x \in \mathcal{D}P''} \mathcal{V}P''x$, and, using Proposition 5.1.5, that $\mathcal{I}\psi = \bigcup_{x \in \mathcal{V}\vec{s}} \mathcal{V}Px$. Hence $\mathcal{D}\psi$ has no variable in common with $\mathcal{V}P \cup \mathcal{D}\pi \cup \mathcal{I}\pi$ and ψ is idempotent. Since we know by our assumptions that $\lambda Px = \lambda Py$ for every two variables $x, y \in \mathcal{V}\vec{s}$, there exists a unique number k such that $\mathcal{I}\psi \subseteq \mathrm{SV}_k$ and $k = 0$ if $\mathcal{I}\psi = \emptyset$.

Let Π' be obtained by either the first construction of the Pyramid Construction Lemma if $k = 0$ or the second construction if $k > 0$. We now verify that $P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G \,\&\, G'$ is V-admissible with respect to Π'.

1.1. Since $\mathcal{D}P$ and $\mathcal{D}P'$ are disjoint and all sort variables in P'' are in $\mathcal{D}\psi$ and hence have the same level with respect to Π', $P \,\&\, P''$ is a Π'-admissible prefix. By our assumptions it is also clear that $P \,\&\, P''$ is normal and inhabited. Furthermore, $V \subseteq \mathcal{D}P \subseteq \mathcal{D}(P \,\&\, P'')$ and hence

$$(P \,\&\, P'')|_V = P|_V = \pi' P|_V = \pi'(P \,\&\, P'')|_V$$

since $\mathcal{D}\psi$ has no variables in common with P.

1.2. Let $s \doteq t \in E$. Then there exists a Π-admissible μ such that $\mathcal{V}\mu \subseteq \mathcal{D}\pi$, $P \vdash s{:}\,\pi_{\lambda^P s}\mu$ and $P \vdash t{:}\,\mu$. Hence μ is Π'-admissible, $\mathcal{V}\mu \subseteq \mathcal{D}\pi \subseteq \mathcal{D}\pi'$ and $P \,\&\, P'' \vdash s{:}\,(\pi'_{\lambda' P \& P''(s)})\mu$ since $(\pi'_{\lambda' P \& P''(s)})\mu = \pi_{\lambda^P s}\mu$ by the construction of Π'.

Now let s_i be a component of $\vec{s}$, t_i be the corresponding component of $\vec{t}$, and σ_i be the corresponding component of $\vec{\sigma}$. Since $\mathcal{V}\sigma_i \subseteq \mathcal{D}\psi$, we know that σ_i is Π'-admissible and $\mathcal{V}\sigma_i \subseteq \mathcal{D}\pi'$. Since $P'' \vdash t_i{:}\,\sigma_i$, we know that $P \,\&\, P'' \vdash t_i{:}\,\sigma_i$. It remains to show that

$$P \,\&\, P'' \vdash s_i{:}\,(\pi'_{\lambda' P \& P''(s_i)})\sigma_i.$$

We know that $P \vdash s_i{:}\,\psi\sigma_i$. Thus $P|_{\mathcal{V}s_i} \vdash s_i{:}\,\psi\sigma_i$ and hence $P|_{\mathcal{V}s_i} \vdash s_i{:}\,\pi_{\lambda^P(s_i)}\psi\sigma_i$ since $\pi_{\lambda^P(s_i)}P|_{\mathcal{V}s_i} = P|_{\mathcal{V}s_i}$. Thus we have $P \vdash s_i{:}\,\pi_{\lambda^P(s_i)}\psi\sigma_i$. Since $\bigcup_{x \in \mathcal{V}\vec{s}} \mathcal{V}Px = \mathcal{I}\psi \subseteq \mathrm{SV}_k$ we know that $\lambda^P(s_i) \geq k$ and hence

$$\pi_{\lambda^P(s_i)}\psi = \pi'_{\lambda' P(s_i)} = \pi'_{\lambda' P \& P''(s_i)}$$

for either construction of Π'. Thus $P \,\&\, P'' \vdash s_i{:}\,(\pi'_{\lambda' P \& P''(s_i)})\sigma_i$.

1.3. Let $r(\vec{u}) \in G$. Then $r(\vec{u})$ is POS-well-typed under $P \,\&\, P''$ with respect to D since $r(\vec{u})$ is POS-well-typed under P with respect to D. Furthermore, $\lambda' Px = \lambda' Py$ for every two variables $x, y \in \mathcal{V}\vec{u}$ since $\lambda Px = \lambda Py$ and P has no variables in common with $\mathcal{D}\psi$.

Let $r(\vec{u}) \in G'$. Then $r(\vec{u})$ is POS-well-typed under $P \,\&\, P''$ with respect to D since $r(\vec{u})$ is POS-well-typed under P'' with respect to D because every clause in C is POS-well-typed with respect to D. Furthermore, $\lambda' P''x = \lambda' P''y$ for every two variables $x, y \in \mathcal{V}\vec{u}$ since all sort variables in P'' are in $\mathcal{D}\psi$ and hence have the same level.

2. We know that $\downarrow P.\emptyset.E \xrightarrow{\mathrm{e}}{}^* Q.\psi.\emptyset$. Since $\mathcal{D}(\downarrow P'') = \mathcal{D}P''$ is disjoint with $\mathcal{D}(\downarrow P)$, we know that $\downarrow P \,\&\, \downarrow P''.\emptyset.E \xrightarrow{\mathrm{e}}{}^* Q \,\&\, \downarrow P''.\psi.\emptyset$ and hence

$$\downarrow P \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \xrightarrow{\mathrm{e}}{}^* Q \,\&\, \downarrow P''.\psi.\psi\vec{s} \doteq \psi\vec{t}.$$

Since $\mathcal{D}\psi \subseteq \mathcal{D}P$ and $\vec{t}$ has no variable in common with P, we know that $\psi\vec{t} = \vec{t}$. Since P'' is a variant of P' such that P'' and P' differ only in sort variables, we know that $\downarrow P' = \downarrow P''$. Hence $\downarrow P \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \xrightarrow{\mathrm{e}}{}^* Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t}$. □

Lemma 7.2.6 [Simulation] *If* $<?\,, F>$ *is a* V*-pair and* $? \xrightarrow{\mathrm{g}} ?'$*, then there exists a constraint* F' *such that* $<?\,, F> \longrightarrow_V <?'\,, F'>$.

Proof. Let $<Q.\psi.G \,\&\, r(\vec{s}), P \,\&\, E \,\&\, G \,\&\, r(\vec{s})>$ be a V-pair, $r(\vec{t}) \leftarrow P' \,\&\, G'$ be a variant of $\gamma \in C$ such that $\mathcal{D}Q$ and $\mathcal{D}P$ are disjoint, $Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t} \xrightarrow{\mathrm{e}}{}^* Q'.\psi'.\emptyset$, and $Q.\psi.G \xrightarrow{\mathrm{g}} Q'.\psi'.G \,\&\, G'$. By the preceding lemma we know that there exists a variant $r(\vec{t}) \leftarrow P'' \,\&\, G'$ of γ having no variables in common with P such that

(a) $P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G \,\&\, G'$ is V-admissible

(b) $\downarrow P \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \;\xrightarrow{\mathrm{e}}{}^{*}\; Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t}$.

Since P contains all variables in $P \,\&\, E \,\&\, G \,\&\, r(\vec{s})$, we know that

$$P \,\&\, E \,\&\, G \,\&\, r(\vec{s}) \;\xrightarrow{\mathrm{h}}\; P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G \,\&\, G'.$$

Furthermore, we have

$$\downarrow P \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \;\xrightarrow{\mathrm{e}}{}^{*}\; Q'.\psi'.\emptyset$$

by plugging the two know $\xrightarrow{\mathrm{e}}$-derivations together. Hence

$$<Q'.\psi'.G \,\&\, G',\, P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G \,\&\, G'>$$

is a V-pair that can be obtained from $<Q.\psi.G \,\&\, r(\vec{s}),\, P \,\&\, E \,\&\, G \,\&\, r(\vec{s})>$ by a $\longrightarrow_V$-step. □

Theorem 7.2.7 [Soundness] *If $P \,\&\, G$ is a POS-goal and $\downarrow P.\emptyset.G \xrightarrow{\mathrm{g}}{}^{*} Q.\psi.\emptyset$, then $\mathcal{T}[\![\uparrow[Q,\psi,P]]\!]^{\mathcal{DP}} \subseteq \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{DP}}$.*

Proof. Follows from the Simulation Lemma and Proposition 7.2.3. □

We call a triple $Q.\psi.E$ **don't care** if there exist P, E' and V such that $P \,\&\, E'$ is V-admissible and $P.\emptyset.E' \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.E$. By the Hauptsatz we know that if we apply E-reduction to don't care triples, all possible choices of what to do next how are don't care nondeterministic.

Theorem 7.2.8 [Don't careness of Constraint Solving] *Let $P \,\&\, G$ be a POS-goal, $\downarrow P.\emptyset.G \xrightarrow{\mathrm{g}}{}^{*} Q.\psi.H \,\&\, r(\vec{s})$, and $r(\vec{t}) \leftarrow P' \,\&\, G'$ be a variant of a clause in C such that $\mathcal{D}Q$ and $\mathcal{D}P'$ are disjoint. Then $(Q \,\&\, \downarrow P').\psi.\psi\vec{s} \doteq \vec{t}$ is don't care.*

Proof. We know that $<\downarrow P.\emptyset.G,\, P \,\&\, G>$ is a $\mathcal{D}P$-pair. Hence we know by the Simulation Lemma that there exists a $\mathcal{D}P$-admissible conjunction

$$P_1 \,\&\, E \,\&\, H \,\&\, r(\vec{s})$$

such that $\downarrow P_1.\emptyset.E \xrightarrow{\mathrm{e}}{}^{*} Q.\psi.\emptyset$. Now we know by Lemma 7.2.5 that there exists a $\mathcal{D}P$-admissible conjunction $(P \,\&\, P'') \,\&\, (E \,\&\, \vec{s} \doteq \vec{t})$ such that

$$\downarrow P \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \;\xrightarrow{\mathrm{e}}{}^{*}\; Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t}.$$

Hence $Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t}$ is don't care. □

Theorem 7.2.9 [Completeness] *Let $P \,\&\, G$ be a POS-goal and $\delta \in \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{DP}}$. Then there exists a complexity measure "$\|?\,\|$" such that*

1. *$\|P.\emptyset.G\|$ is defined*
2. *if $\|Q.\psi.\emptyset\|$ is defined, then $\delta \in \mathcal{T}[\![\uparrow[Q,\psi,P]]\!]^{\mathcal{D}P}$*
3. *if $P.\emptyset.G \xrightarrow{g}{}^{*} ?$, $\|?\|$ is defined, and A is a POS-atom in $?$, then there exists a clause $\gamma \in C$ such that*
   (a) *G-reduction of $?$ on A with γ is possible*
   (b) *if $?'$ is obtained from $?$ by G-reduction on A with γ, then $\|?'\| < \|?\|$.*

Proof. By the Completeness Theorem for H-reduction we know that there exists a suitable complexity measure "$\|F\|$" for H-reduction, the POS-goal $P \,\&\, G$ and $\delta \in \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{D}P}$. With that we define the complexity function "$\|?\|$" as follows:

$$\begin{array}{rcl} \|?\| & = & \min\{\|F\| \mid <P.\emptyset.G,\, P \,\&\, G> \longrightarrow^{*}_{\mathcal{D}P} <?\,,\, F> \;\wedge\; \|F\| \text{ is defined}\} \\ & \text{if} & \text{there exists a conjunction } F \text{ such that } \|F\| \text{ is defined and} \\ & & <P.\emptyset.G,\, P \,\&\, G> \longrightarrow^{*}_{\mathcal{D}P} <?\,,\, F>. \end{array}$$

Now we verify the claims of the Theorem.

1. Obvious.

2. Suppose $\|Q.\psi.\emptyset\|$ is defined. Then there exists Q' and E such that $<P.\emptyset.G,\, P \,\&\, G> \longrightarrow^{*}_{\mathcal{D}P} <Q.\psi.\emptyset,\, Q' \,\&\, E>$ and $\|Q' \,\&\, E\|$ is defined. Hence we know by the Completeness Theorem for H-reduction that $\delta \in \mathcal{S}[\![Q' \,\&\, E]\!]^{\mathcal{D}P} = \mathcal{T}[\![Q' \,\&\, E]\!]^{\mathcal{D}P}$. By Proposition 7.2.3 we know that $\mathcal{T}[\![\uparrow[Q,\psi,P]]\!]^{\mathcal{D}P} = \mathcal{T}[\![Q' \,\&\, E]\!]^{\mathcal{D}P}$. Hence $\delta \in \mathcal{T}[\![\uparrow[Q,\psi,P]]\!]^{\mathcal{D}P}$.

3. Let $P.\emptyset.G \xrightarrow{g}{}^{*} Q.\psi.G_1 \,\&\, r(\vec{s})$ and $\|Q.\psi.G_1 \,\&\, r(\vec{s})\|$ be defined. Then we know that there exist a $\mathcal{D}P$-admissible constraint $P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s})$ such that

$$<P.\emptyset.G,\, P \,\&\, G> \longrightarrow_{\mathcal{D}P} <Q.\psi.G_1 \,\&\, r(\vec{s}),\, P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s})>$$

and $\|Q.\psi.G_1 \,\&\, r(\vec{s})\| = \|P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s})\|$. Now let γ be the clause that exists for $P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s})$ according to statement (3) of the Completeness Theorem for H-reduction. Furthermore, let $r(\vec{t}) \leftarrow P' \,\&\, G'$ be a variant of γ such that $\mathcal{D}Q$ and $\mathcal{D}P'$ are disjoint. Now it suffices to show that:

1. (a) $\exists\, Q', \psi'.\;\; Q.\psi.\psi\vec{s} \doteq \vec{t} \xrightarrow{e}{}^{*} Q'.\psi'.\emptyset$
2. (b) $\|Q'.\psi'.G_1 \,\&\, G'\| < \|Q.\psi.G_1 \,\&\, r(\vec{s})\|$.

By Lemma 7.2.5 we know that there exists a variant $r(\vec{t}) \leftarrow P'' \,\&\, G'$ of γ having no variables in common with P such that:

1. (c) $P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G'$ is $\mathcal{D}P$-admissible
2. (d) $\downarrow P_1 \,\&\, \downarrow P''.\emptyset.E \,\&\, \vec{s} \doteq \vec{t} \;\xrightarrow{e}{}^{*}\; Q \,\&\, \downarrow P'.\psi.\psi\vec{s} \doteq \vec{t}$.

Hence

$$P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s}) \;\xrightarrow{h}\; P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G'$$

by a step on $r(\vec{s})$ with γ. By the Completeness Theorem for H-reduction we know that

$$\|P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G'\| < \|P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s})\|.$$

Hence $\delta \in \mathcal{S}[\![P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G']\!]^{\mathcal{D}P}$ and thus $P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t}$ is satisfiable in $\mathcal{T}$. Furthermore, we know by (c) that $(P \,\&\, P'') \,\&\, (E \,\&\, \vec{s} \doteq \vec{t})$ is $\mathcal{D}P$-admissible. Hence we have claim (a) by (d) and the Hauptsatz.

Since we now have shown claim (a), we know that

$$\begin{aligned} &<Q.\,\psi.\,G_1 \,\&\, r(\vec{s}),\, P_1 \,\&\, E \,\&\, G_1 \,\&\, r(\vec{s})> \\ &\qquad \longrightarrow_{\mathcal{D}P} \; <Q'.\,\psi'.\,G_1 \,\&\, G',\, P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G'>. \end{aligned}$$

Hence

$$\|Q'.\,\psi'.\,G_1 \,\&\, G'\| \leq \|P \,\&\, P'' \,\&\, E \,\&\, \vec{s} \doteq \vec{t} \,\&\, G_1 \,\&\, G'\| < \|Q.\,\psi.\,G_1 \,\&\, r(\vec{s})\|.$$

□

Corollary 7.2.10 [Weak Completeness] *If $P \,\&\, G$ is a POS-goal and $\delta \in \mathcal{S}[\![P \,\&\, G]\!]^{\mathcal{D}P}$, then there exist Q and ψ such that $P.\,\emptyset.\,G \xrightarrow{g}{}^{*} Q.\,\psi.\,\emptyset$ and $\delta \in \mathcal{T}[\![\uparrow[Q, \psi, P]]\!]^{\mathcal{D}P}$.*

7.3 Type Inference

In this section, let T be a fully inhabited type specification, R be a decidable set of relation symbols, and D be a finite and singular set of POS-declarations in $\mathcal{L}(T)^*_R$. As before, we tacitly assume that all constraints are taken from $\mathcal{L}(T)^*_R$.

We will show that it is decidable whether a POS-clause or a POS-goal are POS-well-typed with respect to D. Together with our decidability results for type specifications this shows that one can decide for a finite collection of syntactic objects whether they constitute a POS-program.

In particular, we will investigate type inference, which generalizes the problem of type checking. We will devise an algorithm that, given a POS-clause whose prefix consists of approximations, computes a prefix under which the clause is POS-well-typed. This algorithm doesn't perform perfect type checking in the sense of Chapter 2, in that it may fail although the clause could be well-typed, and in that it may compute a prefix that is not most general. This flaws only appear if the clause employs relations with polymorphic declarations. Nevertheless, an implementation of this algorithm in the TEL programming system works quite well for practical programs. Furthermore, the programmer can always prevent the type inference algorithm from failing by giving more informative sort qualifications for the critical variables.

I don't know whether there is a perfect type inference algorithm for POS-programs in general. However, even if there is one, it is not clear whether it would be useful in practice. Our experiments with a more powerful, backtracking type inference algorithm were rather frustrating since for clauses that couldn't be well-typed (the ones you want to find in practice) it was very slow and we didn't succeed in making it produce good (that is, specific) error messages.

Deciding whether a POS-atom is well-typed is easy:

Proposition 7.3.1 *A POS-atom $r(\vec{s})$ is POS-well-typed under a prefix P with respect to D if and only if there exists a declaration $r{:}\vec{\sigma} \in D$ such that $\sigma^P[\vec{s}] \sqsubseteq \vec{\sigma}$ has an upper matcher.*

To show that a POS-clause $r(\vec{s}) \leftarrow P \,\&\, G$ is POS-well-typed with respect to D, we need to show that every atom in G is POS-well-typed under P with respect to D, and that there exists a variant $r{:}\vec{\sigma}$ of a declaration in D such that $P \vdash \vec{s}{:}\vec{\sigma}$. The well-typedness of the head $r(\vec{s})$ can be decided with the method given in the following proposition.

Proposition 7.3.2 *There exists a variant $\psi\vec{\sigma}$ of $\vec{\sigma}$ such that $P \vdash \vec{s}{:}\psi\vec{\sigma}$ if and only if $\theta := \mathrm{LUM}[\sigma^P[\vec{s}] \sqsubseteq \vec{\sigma}]$ exists and*

1. *if $\alpha \in \mathcal{V}\vec{\sigma}$, then $\theta\alpha$ is a variable or $-$*
2. *if $\alpha, \beta \in \mathcal{V}\vec{\sigma}$ are distinct sort variables, then $\theta\alpha \neq \theta\beta$ or $\theta\alpha = \theta\beta = -$.*

Proposition 7.3.3 *It is decidable whether a POS-clause is POS-well-typed with respect to D. Furthermore, it is decidable whether a conjunction $P \,\&\, G$ is a POS-goal.*

We now attack the problem of type inference. First, we extend the inclusion order by defining the wildcard symbol as the greatest element. The following defines inductively a partial order $\bar{\leq}$ on the set of all sort terms:

$$\begin{array}{l} \sigma \bar{\leq} _, \quad - \bar{\leq} \alpha, \quad \alpha \bar{\leq} \alpha, \\ \xi(\vec{\sigma}) \bar{\leq} \tau \quad \text{if } \tau \Rightarrow^* \xi(\vec{\tau}) \text{ and } \vec{\sigma} \bar{\leq} \vec{\tau}. \end{array}$$

We call a sort term **proper** if it doesn't contain the wildcard symbol.

Proposition 7.3.4 *The relation $\bar{\leq}$ is a partial order on the set of all sort terms such that:*

1. *if $\sigma \leq \tau$, then $\sigma \bar{\leq} \tau$*
2. *if τ is proper and $\sigma \bar{\leq} \tau$, then σ is proper*
3. *if τ is proper, then $\sigma \leq \tau$ if and only if $\sigma \bar{\leq} \tau$.*

The following equations define a computable total function "$\sigma \bar{\sqcap} \tau$" from sort terms to sort terms:

$$\begin{array}{rcl} \sigma \bar{\sqcap} _ & = & \sigma \\ _ \bar{\sqcap} \tau & = & \tau \\ \alpha \bar{\sqcap} \alpha & = & \alpha \\ \xi(\vec{\sigma}) \bar{\sqcap} \eta(\vec{\tau}) & = & \zeta(\vec{\mu} \bar{\sqcap} \vec{\tau}) \quad \text{if } \zeta = \xi \sqcap \eta,\ \xi(\vec{\sigma}) \Rightarrow^* \zeta(\vec{\mu}), \text{ and } \eta(\vec{\tau}) \Rightarrow^* \zeta(\vec{\nu}) \\ \sigma \bar{\sqcap} \tau & = & - \quad \text{if none of the equations above applies.} \end{array}$$

Proposition 7.3.5 *If σ and τ are sort terms, then $\sigma \bar{\sqcap} \tau$ is the infimum of σ and τ with respect to $\bar{\leq}$ and $\sigma \sqcap \tau \leq \sigma \bar{\sqcap} \tau$. Furthermore, if σ and τ are both proper, then $\sigma \bar{\sqcap} \tau = \sigma \sqcap \tau$.*

Next we define a new decomposition relation "$M \xrightarrow{\mathrm{i}} M'$" on membership systems exploiting the extended inclusion order $\bar{\leq}$:

1. $M \,\&\, f(\vec{s}){:}\,\sigma \;\xrightarrow{\mathrm{i}}\; M \,\&\, \vec{s}{:}\,\{\vec{\alpha}/\vec{\sigma}\}\vec{\mu}$
   if $f{:}\,\vec{\mu} \to \xi(\vec{\alpha}) \in T$ and $\sigma \;\Rightarrow^{*}\; \xi(\vec{\sigma})$
2. $M \,\&\, x{:}\,\sigma \,\&\, x{:}\,\tau \;\xrightarrow{\mathrm{i}}\; M \,\&\, x{:}\,(\sigma \bar{\sqcap} \tau)$.

Proposition 7.3.6 *The decomposition relation $M \xrightarrow{\mathrm{i}} M'$ is terminating and confluent.*

Proof. The termination of $\xrightarrow{\mathrm{i}}$ is obvious. Since the infimum function "$\sigma \bar{\sqcap} \tau$" is associative, we know that $\xrightarrow{\mathrm{i}}$ is locally confluent and hence confluent. □

The following conditional equation defines a computable partial function "$\mathrm{IP}[M]$" from membership systems to normal and inhabited prefixes:

$$\mathrm{IP}[M] := \mathrm{NF}[P] \quad \text{if } M \xrightarrow{\mathrm{i}}{}^{*} P \text{ and } P \text{ is an inhabited prefix.}$$

Note that $\mathrm{IP}[M]$ is unique if it exists since a prefix is normal with respect to $\xrightarrow{\mathrm{i}}$ and $\xrightarrow{\mathrm{i}}$ is confluent. If $\mathrm{IP}[M]$ exists, we call it the **inferred prefix of** M.

Proposition 7.3.7 *If every sort term in M is proper and $\mathrm{IP}[M \,\&\, M']$ exists, then $\mathrm{IP}[M \,\&\, M'] \vdash M$ and $\mathrm{IP}[M \,\&\, M'](x)$ is proper if $x \in \mathcal{V}M$.*

If P is a prefix, then $P_{_}$ denotes the prefix obtainable from P by replacing all occurrences of the wildcard symbol with $_$.

Now we are ready to give the type inference algorithm by extending $\mathrm{IP}[\cdot]$ to clauses and their bodies:

1. $\mathrm{IP}[r(\vec{s}) \leftarrow P \,\&\, G] = \mathrm{IP}[\mathrm{IP}[Q \,\&\, \vec{s}{:}\,\vec{\sigma}] \,\&\, G]$ if $r{:}\,\vec{\sigma} \in D$ and
   $Q := P \,\&\, \{x{:}\,_ \mid x \in (\mathcal{V}\vec{s} \cup \mathcal{V}G) - \mathcal{D}P\}$
2. $\mathrm{IP}[P] = P$ if every sort term in P is proper
3. $\mathrm{IP}[P \,\&\, r(\vec{s}) \,\&\, G] = \mathrm{IP}[\mathrm{IP}[P \,\&\, \vec{s}{:}\,\vec{\sigma}] \,\&\, G]$ if $r{:}\,\vec{\sigma} \in D$ and $\vec{\sigma}$ is ground
4. $\mathrm{IP}[P \,\&\, r(\vec{s}) \,\&\, G] = \mathrm{IP}[\mathrm{IP}[P \,\&\, \vec{s}{:}\,\theta\vec{\sigma}] \,\&\, G]$ if $r{:}\,\vec{\sigma} \in D$, $\vec{\sigma}$ is nonground, and $\theta = \mathrm{LUM}[\sigma^{P_\perp}[\vec{s}] \sqsubseteq \vec{\sigma}]$.

To make this definition work, we assume here that the atoms in the body of the clause are ordered and that no declaration in D contains the wildcard symbol. In practical programs that are executed with the usual left-to-right strategy an atom order that is good with respect to control is often also good with respect to type inference.

Proposition 7.3.8 [Type Inference] *Let $A \leftarrow P \,\&\, G$ be a POS-clause such that $Q := \mathrm{IP}[A \leftarrow P \,\&\, G]$ exists. Then:*

1. *$A \leftarrow Q \,\&\, G$ is POS-well-typed with respect to D*
2. *if $x \in \mathcal{D}P$, then $Qx \bar{\leq} Px$.*

The next proposition states that our type inference algorithm is at least perfect for clauses not containing polymorphic relations.

Proposition 7.3.9 [Monomorphic Type Inference] *Let $A \leftarrow P \,\&\, G$ be a POS-clause such that every relation symbol occurring in it has a ground declaration in D. Furthermore, let Q' be a prefix such that $A \leftarrow Q' \,\&\, G$ is POS-well-typed with respect to D, $Q'x \bar{\leq} Px$ for every $x \in \mathcal{D}P$, and $\mathcal{D}Q' = \mathcal{V}A \cup \mathcal{D}P \cup \mathcal{V}G$. Then $Q := \mathrm{IP}[A \leftarrow P \,\&\, G]$ exists, $Q' \leq Q$, and $A \leftarrow Q \,\&\, G$ is POS-well-typed with respect to D.*

Example 7.3.10 Consider the declarations

$$\mathsf{app}\colon \mathsf{l}(\alpha) \times \mathsf{l}(\alpha) \times \mathsf{l}(\alpha), \quad \mathsf{sub}\colon \mathsf{l}(\alpha) \times \mathsf{l}(\alpha)$$

and the clause

$$\mathsf{sub}(\mathsf{S}, \mathsf{L}) \;\leftarrow\; \mathsf{app}(\mathsf{X}, \mathsf{S}, \mathsf{XS}) \,\&\, \mathsf{app}(\mathsf{XS}, \mathsf{Y}, \mathsf{L})$$

and think of app as a list concatenation relation and of sub as a sublist relation. Then the type inference algorithm computes the prefix

$$\mathsf{S}\colon \mathsf{l}(\alpha) \,\&\, \mathsf{L}\colon \mathsf{l}(\alpha) \,\&\, \mathsf{X}\colon \mathsf{l}(\alpha) \,\&\, \mathsf{XS}\colon \mathsf{l}(\alpha) \,\&\, \mathsf{Y}\colon \mathsf{l}(\alpha).$$

Furthermore, for the clause

$$\mathsf{sub}(\mathsf{S}, \mathsf{L}) \;\leftarrow\; \mathsf{S}\colon \mathsf{l}(\mathsf{nat}) \,\&\, \mathsf{app}(\mathsf{X}, \mathsf{S}, \mathsf{XS}) \,\&\, \mathsf{app}(\mathsf{XS}, \mathsf{Y}, \mathsf{L})$$

the type inference algorithm computes the prefix

$$\mathsf{S}\colon \mathsf{l}(-) \,\&\, \mathsf{L}\colon \mathsf{l}(\alpha) \,\&\, \mathsf{X}\colon \mathsf{l}(-) \,\&\, \mathsf{XS}\colon \mathsf{l}(-) \,\&\, \mathsf{Y}\colon \mathsf{l}(\alpha).$$

Incidentally, TEL [Smo88b] would not accept this clause since TEL requires that for every explicitly qualified variable in a clause the given sort term is an approximation of the inferred sort term. □

Example 7.3.11 Consider the type specification

$$\mathsf{nat} \sqsubseteq \mathsf{int}, \quad 1\colon \mathsf{nat}, \quad -1\colon \mathsf{int},$$

the declaration

$$\mathsf{m}\colon \alpha \times \mathsf{l}(\alpha),$$

and the goal

$$\mathsf{m}(1, \mathsf{L}) \,\&\, \mathsf{m}(-1, \mathsf{L})$$

and think of m as a list membership relation. Then the type inference algorithm computes the prefix $\mathsf{L}\colon \mathsf{l}(\mathsf{nat})$ although the weaker prefix $\mathsf{L}\colon \mathsf{l}(\mathsf{int})$ would suffice. Note that the inferred

prefix $\mathsf{L}\colon \mathsf{l}(\mathsf{nat})$ renders the goal unsatisfiable while $\mathsf{L}\colon \mathsf{l}(\mathsf{int}) \,\&\, \mathsf{m}(1, \mathsf{L}) \,\&\, \mathsf{m}(-1, \mathsf{L})$ is satisfiable. However, if the goal is rearranged to

$$\mathsf{m}(-1, \mathsf{L}) \,\&\, \mathsf{m}(1, \mathsf{L})$$

the type inference algorithm computes the most general prefix $\mathsf{L}\colon \mathsf{int}$. Now suppose we extend the program with

$$\mathsf{mtwo} \sqsubseteq \mathsf{int}, \quad -2\colon \mathsf{mtwo}, \quad \mathsf{cons}\colon \alpha \times \mathsf{l}(\alpha) \to \mathsf{l}(\alpha), \quad \mathsf{r}\colon \mathsf{mtwo}.$$

Then the type inference algorithm will fail on the goal

$$\mathsf{m}(1, \mathsf{cons}(\mathsf{X}, \mathsf{L})) \,\&\, \mathsf{r}(\mathsf{X})$$

although the goal is well-typed under its most general prefix

$$\mathsf{X}\colon \mathsf{mtwo} \,\&\, \mathsf{L}\colon \mathsf{l}(\mathsf{int}).$$

However, if the goal is rearranged to

$$\mathsf{r}(\mathsf{X}) \,\&\, \mathsf{m}(1, \mathsf{cons}(\mathsf{X}, \mathsf{L}))$$

the type inference algorithm succeeds with the most general prefix. The difficulty is obviously caused by the fact that, for polymorphic atoms, the type inference algorithm relies on the least upper matcher. Unfortunately, I don't know of any weakening method for the least upper matcher that, under agreeable restrictions, works in general. One difficulty in coming up with such a weakening is that there are always infinite ascending chains like, for instance, $- \sqsubseteq \mathsf{l}(-) \sqsubseteq \mathsf{l}(\mathsf{l}(-)) \sqsubseteq \cdots$. □

Appendix A

Mathematical Preliminaries

In this thesis we use some of the notations and results in the paper [Hue80]. The following just collects a few slight deviations from and additions to this standard notation.

Let $\rightarrow$ be a binary relation on a set M. Then we use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$, and $\rightarrow^+$ to denote the transitive closure of $\rightarrow$. We call $\rightarrow$ **terminating** if there are no infinite chains $a_1 \rightarrow a_1 \rightarrow a_3 \rightarrow \cdots$.

We assume that a decidable set of **function symbols** and a decidable set of **variables** are given such that no variable is a function symbol. Every function symbol comes with a nonnegative integer specifying the number of arguments it takes. Function symbols that take zero arguments are called **constant symbols**. We assume that there are infinitely many variables, and that, for every nonnegative integer n, there are infinitely many function symbols taking n arguments.

A **signature** is a set of function symbols.

Terms are built from variables and function symbols as usual. We often write $f(\vec{s})$ for a term of the form $f(s_1, \ldots, s_n)$, where $n \geq 0$. In this case $\vec{s}$ denotes the tuple $(s_1, \ldots, s_n)$. We assume that there is an **empty tuple** having zero components. A term is called a **Σ-term** if every function symbol occurring in it is in the signature Σ. We use $\mathcal{V}s$ [$\mathcal{V}\vec{s}$] to denote the set of all variables occurring in a term s [tuple $\vec{s}$]. A term is called **linear** if no variable occurs more than once in it.

A **substitution** is a total function θ from terms to terms such that $\theta f(\vec{s}) = f(\theta\vec{s})$ for every term $f(\vec{s})$. If θ is a substitution, the **domain** and the **introduced variables** of θ are defined as follows:

$$\begin{aligned} \mathcal{D}\theta &:= \{x \mid \theta x \neq x \text{ and } x \text{ is a variable}\} \\ \mathcal{I}\theta &:= \bigcup_{x \in \mathcal{D}\theta} \mathcal{V}(\theta x). \end{aligned}$$

A substitution θ is called **finite** if $\mathcal{D}\theta$ is finite. If V is a set of variables, the **restriction**

of θ to V is defined as the substitution $\theta|_V$ satisfying

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

for every variable x. A substitution θ is **idempotent** if $\theta = \theta\theta$. Note that a substitution θ is idempotent if and only if $\mathcal{D}\theta$ and $\mathcal{I}\theta$ are disjoint. We use

$$\{x_1/s_1, \ldots, x_n/s_n\}$$

to denote the substitution θ satisfying $\mathcal{D}\theta \subseteq \{x_1, \ldots, x_n\}$ and $\theta x_i = s_i$ for $i \in 1..n$. Furthermore, we use $\emptyset$ to denote the identity function on the set of all terms (called the **empty substitution**).

A term t is called an **instance** of a term s if there exists a substitution θ such that $t = \theta s$.

A **rewrite rule** is an ordered pair $s \to t$ consisting of two terms s and t such that $\mathcal{V}(t) \subseteq \mathcal{V}(s)$. A **rewrite system** is a set of rewrite rules. If R is a rewrite system, we write $s \Rightarrow_R t$ and call $s \to t$ an **instance** of a rule of R if there exists a substitution θ and a rule $u \to v \in R$ such that $s = \theta u$ and $t = \theta v$. We write $s \to_R t$ if s contains a subterm u such that $u \Rightarrow_R v$ and t can be obtained from s by replacing some (not every) subterm u with v.

If R is a rewrite system and θ and ψ are substitutions, we write $\theta \to_R^* \psi$ if $\theta x \to_R^* \psi x$ for every variable x. Note that $\theta s \to_R^* \psi t$ if $s \to_R^* t$ and $\theta \to_R^* \psi$.

A rewrite system R is called **terminating** if there are no infinite chains $s_1 \to_R s_2 \to_R s_3 \to_R \cdots$.

Bibliography

[AK86] Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351, 1986.

[AKN86] Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *The Journal of Logic Programming*, 3:185–215, 1986.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April 1985.

[Bur69] R. Burstall. Proving properties of programs by structural induction. *Computer Journal*, 12, 1969.

[CKC83] A. Colmerauer, H. Kanoui, and M. Van Caneghem. Prolog, theoretical principles and current trends. *Technology and Science of Informatics*, 2(4):255–292, 1983.

[Col84] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

[Col88] A. Colmerauer. Final specifications for Prolog-III. Manuscript, Esprit Reference Number P1210(1106), February 1988. See also: Opening the Prolog-III Universe, Byte Magazine, August 1987.

[DH88] R. Dietrich and F. Hagl. A polymorphic type system with subtypes for Prolog. In *Proceedings of the 2nd European Symposium on Programming*, volume 300 of *Lectures Notes in Computer Science*, pages 79–93, Berlin, Heidelberg, New York, 1988. Springer-Verlag.

[DHS+88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. Applications of chip to industrial and engineering problems. In *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Tullahoma, Tennessee, June 1988.

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22:465–476, 1979.

[DM82] L. Damas and R. Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM Symposium on Principles of Programming Languages*, pages 207–212, 1982.

[EM85] Harmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1 - Equations and Initial Semantics*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Heidelberg, New York, 1985.

[FGJM85] K. Futatsugi, J. Goguen, J.-P. Jouannaud, and J. Meseguer. Principles of OBJ2. In B. Reid, editor, *Proceedings of 12th ACM Conference on Principles of Programming Languages*, pages 52–66. ACM, 1985.

[GM86] Joseph A. Goguen and Jose Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In Douglas DeGroot and Gary Lindstrom, editors, *Functional and Logic Programming*, pages 295–363. Prentice Hall, 1986.

[GM87a] J. A. Goguen and J. Meseguer. Order-Sorted Algebra I: Partial and Overloaded Operators, Errors and Inheritance. Technical report, Computer Science Lab., SRI International, Menlo Park, 1987. Draft.

[GM87b] Joseph A. Goguen and José Meseguer. Models and equality for logic programming. In *TAPSOFT'87, Pisa, Italy*, pages 1–22, Berlin, West Germany, 1987. LNCS 250, Springer-Verlag.

[Gog78] J. Goguen. Order sorted algebra. Semantics and theory of computation report no. 14, University of California, Los Angeles, 1978.

[GTW78] Joseph A. Goguen, James W. Thatcher, and Eric G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology IV: Data and Structuring*, pages 80–144. Prentice Hall, 1978.

[Han88] M. Hanus. *Horn Clause Specifications with Polymorphic Types*. PhD thesis, FB Informatik, Universität Dortmund, 1988.

[Han89] M. Hanus. Horn clause programs with polymorphic types. In *Proceedings TAPSOFT'89*. Springer-Verlag, 1989.

[HMM86] R. Harper, D. MacQueen, and R. Milner. Standard ml. Report ecs-lfcs-86-2, Dep. of Computer Science, Univ. of Edinburgh, 1986.

[Hoa75] C. A. R. Hoare. Recursive data structures. *International Journal of Computer and Information Sciences*, 1975.

[HS88] Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, October 1988. To appear in the Journal of Logic Programming.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[HV87] M. Huber and I. Varsek. Extended Prolog for order-sorted resolution. In *Proceedings of the 4th IEEE Symposium on Logic Programming*, pages 34–45, San Francisco, 1987.

[JL86] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. Technical report, Department of Computer Science, Monash University, Australia, June 1986.

[JL87] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, West Germany, January 1987. ACM.

[JM87] J. Jaffar and S. Michaylov. Methodology and implementation of a clp system. In J.-L. Lassez, editor, *Proceedings of the 4th International Conference on Logic Programming*, Cambridge, Mass., 1987. MIT Press.

[Joh88] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes 16. Center for the Study of Language and Information, Stanford University, 1988.

[KB82] Ronald M. Kaplan and Joan Bresnan. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. MIT Press, Cambridge, Mass., 1982.

[Lan64] Peter J. Landin. The mechanical evaluation of expressions. *Computer Journal*, pages 308–320, 1964.

[LB87] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[Llo84] J. W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation. Springer-Verlag, Berlin, Heidelberg, New York, 1984.

[MGS89] José Meseguer, Joseph A. Goguen, and Gert Smolka. Order-sorted unification. *Journal of Symbolic Computation*, 8:383–413, 1989.

[Mil78] R. Milner. A Teory of Type Polymorphism in Programming. *Journal of Computer and System Science*, 17:348–375, 1978.

[Mis84] P. Mishra. Towards a theory of types in prolog. In *Proceedings of the 1st IEEE Symposium on Logic Programming*, pages 289–298, Atlantic City, New Jersey, 1984.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[MO84] A. Mycroft and R. A. O'Keefe. A polymorphic type system for Prolog. *Artificial Intelligence*, 23:295–307, 1984.

[Muk87] Kuniaki Mukai. Anadic tuples in prolog. Technical Report TR-239, ICOT, Tokyo, Japan, 1987.

[Neb89] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems.* PhD thesis, Universität des Saarlandes, Saarbrücken, West Germany, June 1989. To appear in Lecture Notes in Artificial Intelligence, Springer-Verlag.

[NR85] Maurice Nivat and John C. Reynolds, editors. *Algebraic Methods in Semantics.* Cambridge University Press, Cambridge, England, 1985.

[NS90] Bernhard Nebel and Gert Smolka. Representation and reasoning with attributive descriptions. In K.H. Bläsius, U.Hedtstück, and C.-R. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, volume 418 of *Lectures Notes in Computer Science*, pages 112–139. Springer-Verlag, Berlin Heidelberg, 1990.

[RK86] William C. Rounds and Robert T. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, pages 38–43, Boston, Mass., 1986.

[SA89] Gert Smolka and Hassan Aït-Kaci. Inheritance hierarchies: Semantics and unification. *Journal of Symbolic Computation*, 7:343–370, 1989.

[Smo88a] Gert Smolka. A feature logic with subsorts. LILOG Report 33, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, May 1988.

[Smo88b] Gert Smolka. Tel (version 0.9), report and user manual. Seki-report sr 87-17, FB Informatik, Universität Kaiserslautern, 1988.

[Smo89] Gert Smolka. Feature constraint logics for unification grammars. IWBS Report 93, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, November 1989. To appear in the Journal of Logic Programming.

[SNGM89] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-Sorted Equational Computation. In Hassan Aït-Kaci and Maurice Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2, Rewriting Techniques*, chapter 10, pages 297–367. Academic Press, New York, N.Y., 1989.

[SS85] Manfred Schmidt-Schauß. A many-sorted calculus with polymorphic functions based on resolution and paramodulation. In *Proceedings of the 9th International Conference on Artificial Intelligence*, pages 1162–1168. Kaufmann, 1985.

[SS89] Manfred Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin, West Germany, 1989.

[SSS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, (47), 1991.

[Str67] C. Strachey. Fundamental concepts in programming languages. Notes for the International Summer School in Computer Programming, Copenhagen, Denmark, 1967.

[Wal83] C. Walther. A many-sorted calculus based on resolution and paramodulation. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 882–891, 1983.

[Wal85] C. Walther. A mechanical solution of Schubert's steamroller by many-sorted resolution. *Artificial Intelligence*, 26:217–224, 1985.

[Wal87] C. Walther. *A Many-Sorted Calculus Based on Resolution and Paramodulation*. Research Notes in Artificial Intelligence. Pitman, London, and Morgan Kaufmann, Los Altos, Calif., 1987.

[Wal88] C. Walther. Many-sorted unification. *Journal of the ACM*, 35(1):1–17, January 1988.

[Zob87] J. Zobel. Derivation of polymorphic types for prolog programs. In *Proceedings of the 4th International Conference on Logic Programming*, pages 817–838, Cambridge, Mass., 1987. MIT Press.