J. Symbolic Computation (1989) **7**, 343–370

Inheritance Hierarchies: Semantics and Unification†

GERT SMOLKA AND HASSAN AIT-KACI‡

FB Informatik, Universität Kaiserslautern,
6750 Kaiserslautern, West Germany
and

‡ *Microelectronics and Computer Technology Corporation,*
3500 West Balcones Center Drive, Austin, Texas 78759, U.S.A.

Inheritance hierarchies are introduced as a means of representing taxonomically organized data. The hierarchies are built up from so-called feature types that are ordered by subtyping and whose elements are records. Every feature type comes with a set of features prescribing fields of its record elements. So-called feature terms are available to denote subsets of feature types. Feature unification is introduced as an operation that decides whether two feature terms have a nonempty intersection and computes a feature term denoting the intersection.

We model our inheritance hierarchies as algebraic specifications in order-sorted equational logic using initial algebra semantics. Our framework integrates feature types whose elements are obtained as records with constructor types whose elements are obtained by constructor application. Unification in these hierarchies combines record unification with order-sorted term unification and is presented as constraint solving. We specify a unitary unification algorithm by a set of simplification rules and prove its soundness and completeness with respect to the model-theoretic semantics.

1. Introduction

This paper presents inheritance hierarchies as a means of representing taxonomically organized data. In our approach, an inheritance hierarchy is built up from so-called feature types, which are ordered by subtyping and whose elements are records. Every feature type comes with a set of features prescribing fields of its record elements.

Figure 1.1 shows an inheritance hierarchy consisting of the feature types **object**, **house**, **vehicle**, **machine**, **car**, **other_car**, **gas_car**, **electric_car**, **fuel_type**, **gasoline**, and **electricity**. Subtypes are given below their supertypes. The type **object** has the feature *age* of type **nat**. The type **vehicle** has two features—the feature *age*: **nat** inherited from the supertype **object** and the feature *speed*: **nat** explicitly specified. The type **car** has three features—*age*: **nat** inherited from both **vehicle** and **machine**, *speed*: **nat** inherited from **vehicle**, and *fuel*: **fuel_type** inherited from **machine**. (The fact that a subtype inherits features from more than one supertype could be referred to as multiple inheritance.) The type **other_car** has the same features as **car**. The

† This paper is a revised version of the MCC Technical Report AI-057-87, Microelectronics and Computer Technology Corporation, Austin, Texas 78759, May 1987.

0747–7171/89/030343+28 $03.00/0

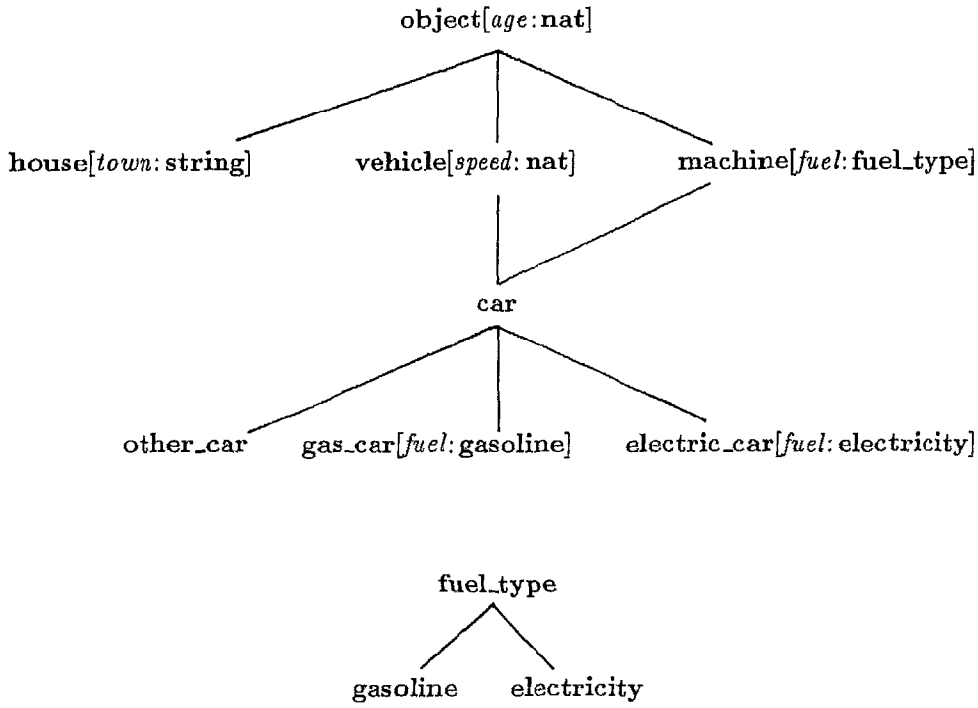

FIGURE 1.1. An inheritance hierarchy.

type **gas_car** illustrates a further concept: the feature *fuel* is inherited from **car**, but is restricted from **fuel_type** to **gasoline**. The feature types **fuel_type**, **gasoline**, and **electricity** don't have features.

Semantically, feature types are interpreted as sets and subtyping is interpreted as set inclusion. Minimal feature types are assumed to be pairwise disjoint and nonminimal feature types are assumed to be the union of their subtypes. Consequently, the types **vehicle**, **machine** and **car** of the hierarchy in Figure 1.1 all denote the same set, and every element of the type **object** is either an element of **house** or **car**. To allow for machines that aren't cars, one could equip **machine** with a subtype **other_machine**.

A minimal feature type ξ is interpreted as the set of all records labeled with ξ and possessing exactly the fields declared for or inherited by ξ. Consequently, the type **gasoline** is interpreted as a singleton consisting of the empty record labeled with **gasoline**, and every element of the type **house** is a record labeled with **house** and having exactly the fields *age* and *town*.

With so-called feature terms one can describe subsets of feature types. The feature term

house[*age* $\Rightarrow$ 150; *town* $\Rightarrow$ 'Austin']

denotes all elements of the feature type **house** whose feature *age* is 150 and whose feature *town* is 'Austin'. Given the hierarchy in Figure 1.1, this feature term denotes the

singleton consisting of the record labeled with **house** and having the fields *age* $\Rightarrow$ 150 and *town* $\Rightarrow$ 'Austin'. The feature term

$$\mathbf{vehicle}[\mathit{speed} \Rightarrow N;\ \mathit{age} \Rightarrow N]$$

denotes all elements of the feature type **vehicle** whose features *speed* and *age* agree.

Feature unification is an operation that, given two feature terms R and S, computes a feature term T denoting the intersection of the denotations of R and S; if the intersection is empty, feature unification fails. Given the hierarchy in Figure 1.1, unification of the two feature terms

$$\mathbf{vehicle}[\mathit{speed} \Rightarrow N;\ \mathit{age} \Rightarrow N]$$
$$\mathbf{machine}[\mathit{age} \Rightarrow 30]$$

yields the feature term

$$\mathbf{car}[\mathit{age} \Rightarrow 30;\ \mathit{speed} \Rightarrow 30],$$

while unification of the terms

$$\mathbf{vehicle}[\mathit{speed} \Rightarrow N;\ \mathit{age} \Rightarrow N]$$
$$\mathbf{vehicle}[\mathit{speed} \Rightarrow 120;\ \mathit{age} \Rightarrow 2]$$

fails.

Unification of feature terms is investigated in (Aït-Kaci, 1984, 1986; Aït-Kaci & Nasr, 1986). In this approach, a class of feature terms, called ψ-terms, is formally defined together with a preorder, called subsumption. Intuitively, a ψ-term S is subsumed by a ψ-term T if S contains more information than T, or, equivalently, S denotes a subset of T. Under this subsumption order, the set of all ψ-terms is a prelattice, provided the sort symbols are ordered as a lattice. Unification is then defined as computing greatest lower bounds in the prelattice of ψ-terms.

The main purpose of this paper is to reconcile feature terms and their unification with the algebraic approach to abstract data types (Goguen et al., 1978; Ehrig & Mahr, 1985), where data types are specified in many-sorted equational logic and are defined as the initial algebras of their specifications. Using order-sorted equational logic (Goguen, 1978; Goguen & Meseguer, 1987b; Smolka et al., 1987), it is straightforward to specify inheritance hierarchies such that the right initial algebras are obtained. It is less obvious, however, how to accommodate feature terms and their unification. Feature terms are different from ordinary terms in that they denote subsets of sorts rather than elements of sorts. We will solve the problem by formalizing a containment $x{:}S$, which constrains a variable x to the elements of a feature term S, as a solved equation system constraining x. Unification of two feature terms S and T will then mean to solve the combined equation system $x{:}S \,\&\, x{:}T$, where solving an equation system means to compute an equivalent system that is in solved form and to decide thereby whether the given system is satisfiable.

Our formulation of inheritance hierarchies as algebraic specifications yields a framework which naturally accommodates feature types as well as ordinary algebraic types whose elements are obtained by constructor application. We will restrict our interest to constructor types that are defined without equations. We will present a constraint

solving method for solving equation systems over specifications consisting of feature and constructor types and prove its soundness and completeness. This constraint solving method generalizes both ψ-term unification (Aït-Kaci, 1984, 1986), which applies to feature types, and order-sorted unification (Walther, 1984, 1988), which applies to constructor types.

Our account of feature unification as equation solving, or, more general, as constraint solving, shows its compatibility and relationship with ordinary term unification (Herbrand, 1930; Robinson, 1965). The view of ordinary term unification as equation solving goes back to Herbrand (1930), was lost by Robinson (1965), and was later reestablished by Colmerauer (1982, 1984) and Martelli and Montanari (1982). In the equation solving paradigm, the notion of a most general unifier is replaced by the more general notion of a solved form, and unification is the stepwise transformation of an equation system to solved form by the application of simplification rules.

Our framework prepares the ground for the integration of inheritance hierarchies and feature terms into typed logic-based programming languages such as OBJ (Futatsugi, 1985) and Eqlog (Goguen & Meseguer, 1986). LOGIN (Aït-Kaci & Nasr, 1986) is an already implemented elaboration of Prolog, where ordinary terms are replaced with ψ-terms and ordinary unification is replaced with ψ-unification. The research presented in this paper contributes to the theoretical foundation of LOGIN and shows its compatibility with the ideas underlying the language proposal Eqlog. Mukai's (1987) language CIL bears many similarities with LOGIN. Cardelli (1984) investigates the semantics of inheritance hierarchies similar to ours in the framework of higher-order functional programming where there is no need for unification.

Several variants of feature descriptions are being used in computational linguistics (see (Shieber, 1986; Pereira, 1987; Pollard & Sag, 1987) for introductory expositions). In so-called unification grammars, unification of feature descriptions is the basic operation for parsing and generating natural language. Rounds and Kasper (1986) were the first to come up with a logical formalization of feature descriptions. In their approach, a feature description denotes a set of so-called feature structures, which correspond roughly to our feature terms. For building complex feature descriptions, set intersection and union are available. A unification method for feature descriptions consists of a normal form that exhibits inconsistency and an algorithm that, given two normal feature descriptions, computes a normal feature description denoting their intersection. Recent work (Moshier & Rounds, 1987; Johnson, 1987; Smolka, 1988; Kasper, 1988) studies feature descriptions with set complements and negations.

The paper is organized as follows. In Section 2 we discuss informally how inheritance hierarchies and feature unification can be expressed in order-sorted equational logic. In Section 3 we review the necessary notations and results for this logic, which we use in Section 4 to formalize feature and constructor types. In Section 5 we devise a framework for unification that is general enough to accommodate feature unification, which is finally presented in Section 6.

2. Feature Types as Algebraic Types

2.1. Constructor Types and Subtyping

A basic method of introducing a data type is to define its elements by a set of free constructors. For instance, the type of natural numbers can be defined by the signature

equation

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

saying that every natural number can be obtained with the two constructors

$$o\colon \rightarrow \mathbf{nat}$$
$$s\colon \mathbf{nat} \rightarrow \mathbf{nat}.$$

The symbol o is a constructor with no arguments representing the number zero. Positive numbers are obtained by applying the constructor s repeatedly to zero; for instance, one is represented as $s(o)$ and two as $s(s(o))$. We speak of free constructors since there are no laws associated with the constructors. Two ground terms denote the same number if and only if they are constructed in exactly the same way.

We call types that are defined by free constructors constructor types. Constructor types have been popular for a long time (Landin, 1964; Burstall, 1969; Hoare, 1975) and are employed, for instance, in the functional programming languages ML (Milner, 1984).

Constructor types are a special case of algebraic types, which enjoy a well-established mathematical foundation (Goguen et al., 1978; Nivat & Reynolds, 1985; Ehrig & Mahr, 1985). Algebraic types are specified in many-sorted equational logic by a signature and equational axioms, and their semantics is given by an initial model, which always uniquely exists. Constructor types are free algebraic types, that is, they can be specified without equational axioms.

Many-sorted logic can be generalized to order-sorted logic (Goguen, 1978) by introducing the notion of subtyping. Semantically, ξ being a subtype of η means that the denotation of ξ is a subset of the denotation of η. Subtypes arise naturally when we define a type as the union of other types. The specification and programming language OBJ2 (Futatsugi et al., 1985) is based on order-sorted logic.

The notion of subtyping applies to constructor types. For instance, an alternative way to declare the natural numbers is

$$\mathbf{nat} := \mathbf{zero} \cup \mathbf{posint}$$
$$\mathbf{zero} := \{o\}$$
$$\mathbf{posint} := \{s\colon \mathbf{nat}\}.$$

The first signature equation says that **zero** and **posint** are subtypes of **nat** and that the elements of **nat** are obtained as the union of the elements of **zero** and **posint**. The second equation says that **zero** has exactly one element, which is obtained by the constructor $o\colon \rightarrow$ **zero**. The third equation says that all elements of **posint** are obtained by applying the constructor $s\colon$ **nat** $\rightarrow$ **posint** to natural numbers. To define the negative integers, we can add the equation

$$\mathbf{negint} := \{-\colon \mathbf{posint}\},$$

which declares the constructor $-\colon$ **posint** $\rightarrow$ **negint**. With that we finally can define the integers

$$\mathbf{int} := \mathbf{negint} \cup \mathbf{nat}$$

as the union of the negative integers and the natural numbers.

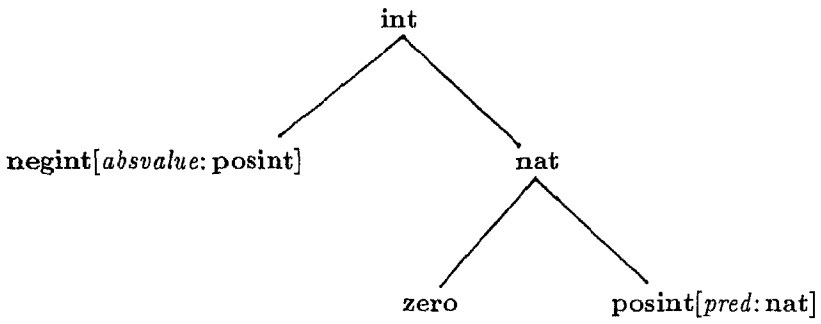

FIGURE 2.1. The integers as inheritance hierarchy.

2.2. FEATURES AS SELECTORS

Free constructors are complemented by so-called selectors that recover their arguments. In our last example, the selectors for the constructors s: **nat** $\rightarrow$ **posint** and $-$: **posint** $\rightarrow$ **negint** are their inverses in the initial model and can be specified as follows:

$$\begin{aligned} &\mathit{pred}: \mathbf{posint} \rightarrow \mathbf{nat} \\ &\quad \mathit{pred}(s(N)) = N \end{aligned}$$

$$\begin{aligned} &\mathit{absvalue}: \mathbf{negint} \rightarrow \mathbf{posint} \\ &\quad \mathit{absvalue}(-P) = P. \end{aligned}$$

Now observe that it makes sense to say that positive integers have the feature *pred*: **nat**, and that negative integers have the feature *absvalue*: **posint**. Taking this idea further, it turns out that the integers can be represented by the inheritance hierarchy shown in Figure 2.1. (See Goguen and Meseguer (1987a) for a thorough discussion of constructors and selectors in order-sorted algebraic specifications.)

Now observe that every constructor term can be represented as a feature term, for instance:

o	**zero**
$s(o)$	**posint**[*pred* $\Rightarrow$ **zero**]
$s(s(X))$	**posint**[*pred* $\Rightarrow$ **posint**[*pred* $\Rightarrow X$]].
$-s(o)$	**negint**[*absvalue* $\Rightarrow$ **posint**[*pred* $\Rightarrow$ **zero**]]

This suggests that feature types and constructor types are dual concepts. Constructor types are defined by giving their constructors, while feature types are defined by giving their selectors. From the definition of a constructor type we can uniquely determine the corresponding selectors (not their names, of course); vice versa, from the definition of a feature type we can uniquely determine the corresponding constructors.

In other words, feature types can be expressed as algebraic types if we supply the missing constructors, which we will call *implicit constructors*. Figure 2.2 gives an algebraic

object := **house** ∪ **vehicle** ∪ **machine**
house := {*con_house*: **nat** × **string**}
vehicle := **car**
machine := **car**
car := **other_car** ∪ **gas_car** ∪ **electric_car**
other_car := {*con_other_car*: **nat** × **nat** × **fuel_type**}
gas_car := {*con_gas_car*: **nat** × **nat** × **gasoline**}
electric_car := {*con_electric_car*: **nat** × **nat** × **electricity**}
fuel_type := **gasoline** ∪ **electricity**
gasoline := {*con_gasoline*}
electricity := {*con_electricity*}

variables A: **nat**, T: **string**, S: **nat**,
F: **fuel_type**, G: **gasoline**, E: **electricity**

age: **object** → **nat**
$age(con\_house(A, T)) = A$
$age(con\_other\_car(A, S, F)) = A$
$age(con\_gas\_car(A, S, G)) = A$
$age(con\_electric\_car(A, S, E)) = A$

town: **house** → **string**
$town(con\_house(A, T)) = T$

speed: **vehicle** → **nat**
$speed(con\_other\_car(A, S, F)) = S$
$speed(con\_gas\_car(A, S, G)) = S$
$speed(con\_electric\_car(A, S, E)) = S$

fuel: **machine** → **fuel_type**
fuel: **gas_car** → **gasoline**
fuel: **electric_car** → **electricity**
$fuel(con\_other\_car(A, S, F)) = F$
$fuel(con\_gas\_car(A, S, G)) = G$
$fuel(con\_electric\_car(A, S, E)) = E$

FIGURE 2.2. An algebraic specification of the inheritance hierarchy in Figure 1.1.

specification of the inheritance hierarchy in Figure 1.1. In accordance with our discussion in Section 1, only minimal feature types are equipped with implicit constructors. This is in fact necessary to make feature types and constructor types dual concepts, since otherwise not every constructor term could be expressed as a feature term. Furthermore, if one would like, for instance, to have other machines than cars, one could accomplish this by adding **other_machine** as an additional subtype of **machine**.

There are three declarations for the selector *fuel* to express the restrictions for the subtypes **gas_car** and **electric_car**. Semantically, the selector *fuel* is a single function **machine** → **fuel_type** that, when applied to an element of **gas_car**, yields an element of **gasoline**, and, when applied to an element of **electric_car**, yields an element of **electricity**.

The following signature equations are a possible textual representation of the inheri-

tance hierarchy in Figure 1.1:

$$
\begin{aligned}
&\mathbf{object} := [\mathit{age}\colon \mathbf{nat}]\\
&\mathbf{house} := \mathbf{object}[\mathit{town}\colon \mathbf{string}]\\
&\mathbf{vehicle} := \mathbf{object}[\mathit{speed}\colon \mathbf{nat}]\\
&\mathbf{machine} := \mathbf{object}[\mathit{fuel}\colon \mathbf{fuel\_type}]\\
&\mathbf{car} := (\mathbf{vehicle} * \mathbf{machine})[]\\
&\mathbf{other\_car} := \mathbf{car}[]\\
&\mathbf{gas\_car} := \mathbf{car}[\mathit{fuel}\colon \mathbf{gasoline}]\\
&\mathbf{electric\_car} := \mathbf{car}[\mathit{fuel}\colon \mathbf{electricity}]\\
&\\
&\mathbf{fuel\_type} := [\,]\\
&\mathbf{gasoline} := \mathbf{fuel\_type}[\,]\\
&\mathbf{electricity} := \mathbf{fuel\_type}[\,].
\end{aligned}
$$

Note that these type equations uniquely determine the specification in Figure 2.2.

Figure 2.3 further clarifies the duality of constructor and feature types by defining the natural numbers and their addition in both the constructor and the feature-oriented approach.

2.3. Equation Solving in Inheritance Hierarchies

A main benefit obtained from expressing inheritance hierarchies as algebraic specifications is the provision of a well understood notion of equality. For instance, consider the inheritance hierarchy of Figures 1.1 and 2.2 and the equation (capital letters are variables)

$$
\begin{aligned}
&V = M\\
&\mathit{where}\quad V\colon \mathbf{vehicle}[\mathit{speed} \Rightarrow N\colon \mathbf{nat};\ \mathit{age} \Rightarrow N],\\
&\qquad\quad M\colon \mathbf{machine}[\mathit{age} \Rightarrow 30;\ \mathit{fuel} \Rightarrow \mathbf{gasoline}].
\end{aligned}
\tag{1}
$$

We will consider this equation whose variables are constrained by feature terms to be syntactic sugar for the "ordinary" equation system

$$
\begin{aligned}
&\mathit{speed}(V) = N\ \&\ \mathit{age}(V) = N\ \&\\
&\mathit{age}(M) = 30\ \&\ \mathit{fuel}(M) = G\ \&\\
&V = M\\
&\mathit{where}\ V\colon \mathbf{vehicle},\ M\colon \mathbf{machine},\ N\colon \mathbf{nat},\ G\colon \mathbf{gasoline}.
\end{aligned}
\tag{2}
$$

One of the two solutions of this system is

$$
\begin{aligned}
&V = \mathit{con\_other\_car}(30, 30, \mathit{con\_gasoline})\\
&M = \mathit{con\_other\_car}(30, 30, \mathit{con\_gasoline})\\
&N = 30\\
&G = \mathit{con\_gasoline}.
\end{aligned}
$$

Solving an equation system means deciding whether the system has a solution and, if so, to compute a compact representation of the solution space. Usually, it is not a good idea to enumerate all solutions since there can be infinitely many. For instance,

$\mathbf{nat} := \mathbf{zero} \cup \mathbf{posint}$
$\mathbf{zero} := \{o\}$
$\mathbf{posint} := \{s\colon \mathbf{nat}\}$

$+\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}$ (*variables* $N\colon \mathbf{nat}$, $M\colon \mathbf{nat}$)
$o + N = N$
$s(N) + M = s(N + M)$

FIGURE 2.3.a. Constructor-oriented definition of the natural numbers and their addition.

$\mathbf{nat} := [\,]$
$\mathbf{zero} := \mathbf{nat}[\,]$
$\mathbf{posint} := \mathbf{nat}[pred\colon \mathbf{nat}]$

$+\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}$ (*variables* $N\colon \mathbf{nat}$, $M\colon \mathbf{nat}$)
$\mathbf{zero}[\,] + N = N$
$\mathbf{posint}[pred \Rightarrow N] + M = \mathbf{posint}[pred \Rightarrow N + M]$

FIGURE 2.3.b. Feature-oriented definition of the natural numbers and their addition.

$+\colon \mathbf{nat} \times \mathbf{nat} \to \mathbf{nat}$ (*variables* $Z\colon \mathbf{zero}$, $N\colon \mathbf{nat}$, $P, P'\colon \mathbf{posint}$)
$Z + N = N$
$P + N = P'$ **if** $pred(P') = pred(P) + N$

FIGURE 2.3.c. Feature-oriented definition of addition not using feature term syntax. The second equation of $+$ must now be formulated conditionally.

the equation $N = pred(P)$ has infinitely many solutions in the inheritance hierarchy in Figure 2.1.

We will introduce feature unification as a solution method for equations over constructor and feature types. Let us illustrate how feature unification works by applying it to the equation system (2). The equation $V = M$ in (2) requires that the value of V and M be an element of a common subtype of **vehicle** and **machine**. A glance at Figure 1.1 tells us that the greatest common subtype of **vehicle** and **machine** is **car**. Hence, by introducing an auxiliary variable $C\colon \mathbf{car}$, we can simplify system (2) to

$$V = C \;\&\; M = C \;\&\; speed(C) = N \;\&\; age(C) = N \;\&\; age(C) = 30 \;\&\; fuel(C) = G \quad (3)$$
where $V\colon \mathbf{vehicle}$, $M\colon \mathbf{machine}$, $C\colon \mathbf{car}$, $N\colon \mathbf{nat}$, $G\colon \mathbf{gasoline}$.

Note that system (2) and (3) have the same solutions for the "primary variables" V, M, and N.

System (3) has two constraints $age(C) = N$ and $age(C) = 30$ for the feature *age* of C. This information can be represented equivalently by $age(C) = 30$ and $N = 30$, which yields the simplified system

$$\begin{aligned} &V = C \;\&\; M = C \;\&\; N = 30 \;\& \\ &speed(C) = 30 \;\&\; age(C) = 30 \;\&\; fuel(C) = G \\ &\textit{where } V\text{: }\mathbf{vehicle},\ M\text{: }\mathbf{machine},\ C\text{: }\mathbf{car},\ N\text{: }\mathbf{nat},\ G\text{: }\mathbf{gasoline}, \end{aligned} \tag{4}$$

which is in solved form. Since all simplification steps kept the solutions for the primary variables V, M, and N invariant, system (4) represents the solution space of system (1). Using again feature term syntax, system (4) becomes

$$\begin{aligned} &V = C \;\&\; M = C \;\&\; N = 30 \\ &\textit{where } \ C\text{: }\mathbf{car}[\textit{speed} \Rightarrow 30;\ \textit{age} \Rightarrow 30;\ \textit{fuel} \Rightarrow \mathbf{gasoline}]. \end{aligned}$$

Note that the feature term $\mathbf{car}[\textit{speed} \Rightarrow 30;\ \textit{age} \Rightarrow 30;\ \textit{fuel} \Rightarrow \mathbf{gasoline}]$ represents the set consisting of the two constructor terms $con\_other\_car(30, 30, con\_gasoline)$ and $con\_gas\_car(30, 30, con\_gasoline)$.

3. Order-Sorted Equational Logic

In this section we review the necessary notations and results of order-sorted equational logic following (Smolka et al., 1987).

Every specification in equational logic has an initial model, which is unique up to isomorphism. Goguen et al. (1978) discovered that data types can be understood as the initial models of their specifications in many-sorted equational logic and thus started a field of research now known as Algebraic Specification (Ehrig & Mahr, 1985). Although already (Goguen, 1978) advocated the use of subsorts for the algebraic specification of data types, technically mature accounts of order-sorted equational logic appeared only recently (Goguen & Meseguer, 1987; Smolka et al., 1987).

3.1. Syntax

We assume three pairwise disjoint, countably infinite sets of symbols:

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

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

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

Remark. In this paper "type" is used as a synonym of "sort".

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

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

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

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

Let Σ be a signature.

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

A *Σ-equation* has the form $s \doteq t$, where s and t are Σ-terms. The letter P will always range over equations.

A *Σ-equation system* is either the *empty equation system* $\emptyset$ or has the form $P_1 \;\&\; \cdots \;\&\; P_n$, where $P_1, \ldots, P_n$ are Σ-equations. To ease our notation, we assume that the conjunction operator & is associative and commutative and satisfies $E \;\&\; \emptyset = E$. The letter E will always range over equation systems.

A *Σ-implication* has the form $E \rightarrow E'$ where E and E' are Σ-equation systems. Sometimes we will write an implication $E \rightarrow E'$ in backward notation as $E' \leftarrow E$.

A *syntactic Σ-object* is either a Σ-term, Σ-equation, Σ-equation system, or a Σ-implication. A syntactic object is called *ground* if it doesn't contain variables. We use $\mathcal{V}(O)$ to denote the set of all variables occurring in a syntactic object O.

A signature Σ is *regular* if every Σ-term s has a least sort σs, that is, there is a unique function σ from the set of all Σ-terms to the set of Σ-sort symbols such that (1) s is a term of sort σs and (2) $\sigma s \leq \xi$ if s is a term of sort ξ. In regular signatures the sort function "σs" extends the sort function "σx" from Σ-variables to Σ-terms.

Regularity is a property needed to make order-sorted term unification well-behaved (see (Smolka et al., 1987) for a discussion).

THEOREM 3.1. *A signature Σ is regular if and only if for every function symbol f and every string $\vec{\xi}$ of sort symbols the set $\{\eta \mid (f: \vec{\eta} \rightarrow \eta) \in \Sigma \text{ and } \vec{\xi} \leq \vec{\eta}\}$ is either empty or has a minimum with respect to the subsort order of Σ.*

COROLLARY 3.2. *Regularity of finite signatures is decidable.*

COROLLARY 3.3. *Every signature without multiple function declarations is regular.*

The signature $\{a: \rightarrow \mathbf{A},\ a: \rightarrow \mathbf{B}\}$ isn't regular since the constant a doesn't have a least sort. Multiple constant declarations don't make sense in regular signatures.

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

3.2. SUBSTITUTIONS AND INSTANCES

An important technicality of order-sorted logic is that only substitutions are admitted that map well-sorted terms to well-sorted terms.

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

1. if s is a Σ-term of sort ξ, then θs is a Σ-term of sort ξ
2. $\theta f(s_1, \ldots, s_n) = f(\theta s_1, \ldots, \theta s_n)$

3. $\mathcal{D}\theta := \{x \mid \theta x \neq x\}$ is finite.

Following the usual abuse of notation, we call $\mathcal{D}\theta$ the *domain of* θ, $\mathcal{C}\theta := \{\theta x \mid x \in \mathcal{D}\theta\}$ the *codomain of* θ, and $\mathcal{I}\theta := \mathcal{V}(\mathcal{C}\theta)$ the set of *variables introduced by* θ. The letters θ, ψ, and ϕ will always range over substitutions. The identity function on Σ-terms is called the *empty substitution* and is denoted by ϵ. The composition of Σ-substitutions is again a Σ-substitution. Σ-substitutions are extended to syntactic Σ-objects as usual. A substitution θ is *idempotent* if $\theta\theta = \theta$. Note that this is the case if and only if $\mathcal{D}\theta$ and $\mathcal{I}\theta$ are disjoint.

The *equational representation* $[\theta]$ of a substitution θ is the equation system

$$x_1 \doteq \theta x_1 \;\&\; \cdots \;\&\; x_n \doteq \theta x_n,$$

where $\{x_1, \ldots, x_n\} = \mathcal{D}\theta$. Two substitutions are equal if and only if their equational representations are equal (recall that the conjunction operator & is associative and commutative). Every equation system $x_1 \doteq s_1 \;\&\; \cdots \;\&\; x_n \doteq s_n$ such that $x_1, \ldots, x_n$ are distinct variables and s_i is a Σ-term of sort σx_i for all i is the equational representation of some Σ-substitution, which we denote by $\langle x_1 \doteq s_1 \;\&\; \cdots \;\&\; x_n \doteq s_n\rangle$. Note that $\theta = \langle[\theta]\rangle$ for every substitution θ.

Let θ be a Σ-substitution and V be a set of Σ-variables. The *restriction* $\theta|_V$ *of* θ *to* V is defined by: $\theta|_V(x) := \theta x$ if $x \in V$, otherwise $\theta|_V(x) := x$. Furthermore, if s is a term of sort σy, the *update* $\theta[y \leftarrow s]$ *of* θ *at* y *with* s is defined by: $\theta[y \leftarrow s](x) := s$ if $x = y$, otherwise $\theta[y \leftarrow s](x) := \theta x$.

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

3.3. Semantics

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

1. $\xi^{\mathcal{A}}$ is a set
2. if $(\xi < \eta) \in \Sigma$ then $\xi^{\mathcal{A}} \subseteq \eta^{\mathcal{A}}$
3. $\mathrm{C}_{\mathcal{A}} := \bigcup \{\xi^{\mathcal{A}} \mid \xi \text{ is a sort symbol of } \Sigma\}$ is called the *carrier of* $\mathcal{A}$
4. $f^{\mathcal{A}}$ is a mapping $\mathrm{D}_f^{\mathcal{A}} \to \mathrm{C}_{\mathcal{A}}$ whose domain $\mathrm{D}_f^{\mathcal{A}}$ is a subset of $\mathrm{C}_{\mathcal{A}}^{|f|}$
5. if $(f : \xi_1 \ldots \xi_n \to \xi) \in \Sigma$ and $a_i \in \xi_i^{\mathcal{A}}$ for $i = 1, \ldots, n$, then $(a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}}$ and $f^{\mathcal{A}}(a_1, \ldots, a_n) \in \xi^{\mathcal{A}}$.

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

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

1. $\gamma(\xi^{\mathcal{A}}) \subseteq \xi^{\mathcal{B}}$ for every Σ-sort symbol ξ
2. $\gamma(\mathrm{D}_f^{\mathcal{A}}) \subseteq \mathrm{D}_f^{\mathcal{B}}$ for every Σ-function symbol f
3. $\gamma(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(\gamma(a_1), \ldots, \gamma(a_n))$ for every Σ-function symbol f and every tuple $(a_1, \ldots, a_n) \in \mathrm{D}_f^{\mathcal{A}}$.

PROPOSITION 3.4. *Let Σ be a signature. Then the Σ-algebras together with their homomorphisms comprise a category.*

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

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

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

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

Validity with respect to a Σ-algebra $\mathcal{A}$ and a $(V, \mathcal{A})$-assignment α is defined as follows (s, t, P, E, and E' are Σ-objects containing only variables in V):

- $\mathcal{A}, \alpha \models s \doteq t \quad :\Longleftrightarrow \quad [\![s]\!]_\alpha = [\![t]\!]_\alpha$
- $\mathcal{A}, \alpha \models P \& E \quad :\Longleftrightarrow \quad \mathcal{A}, \alpha \models P$ and $\mathcal{A}, \alpha \models E$
- $\mathcal{A}, \alpha \models \emptyset$ always holds ($\emptyset$ is the empty equation system)
- $\mathcal{A}, \alpha \models E \to E' \quad :\Longleftrightarrow \quad$ if $\mathcal{A}, \alpha \models E$, then $\mathcal{A}, \alpha \models E'$.

Validity with respect to a Σ-algebra $\mathcal{A}$ is defined as follows:

- $\mathcal{A} \models E \quad :\Longleftrightarrow \quad \forall\ (\mathcal{V}(E), \mathcal{A})$-assignment α. $\mathcal{A}, \alpha \models E$
- $\mathcal{A} \models E \to E' \quad :\Longleftrightarrow \quad \forall\ (\mathcal{V}(E \to E'), \mathcal{A})$-assignment α. $\mathcal{A}, \alpha \models E \to E'$.

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and $\mathcal{A}$ be a Σ-algebra. We say that $\mathcal{A}$ is an $\mathcal{S}$-*algebra* or $\mathcal{A}$ is a *model of* $\mathcal{S}$ if every equation of $\mathcal{E}$ is valid in $\mathcal{A}$. We say that a Σ-equation system E *is valid in* $\mathcal{S}$ or $\mathcal{S}$ *satisfies* E if E is valid in every $\mathcal{S}$-algebra; in this case we write $\mathcal{S} \models E$. The relation $\mathcal{S} \models E \to E'$ is defined accordingly. We write $\Sigma \models E$ if the free specification $(\Sigma, \emptyset)$ satisfies E.

THEOREM 3.5. *Let s and t be two Σ-terms. Then $\Sigma \models s \doteq t$ if and only if $s = t$.*

THEOREM 3.6. *A specification satisfies an equation $s \doteq t$ if and only if it satisfies every instance of $s \doteq t$.*

A model $\mathcal{I}$ of a specification $\mathcal{S}$ is called an *initial model of* $\mathcal{S}$ if, for every model $\mathcal{A}$ of $\mathcal{S}$, there exists a unique homomorphism $\mathcal{I} \to \mathcal{A}$.

THEOREM 3.7. *Every specification has an initial model, which is unique up to isomorphism.*

THEOREM 3.8. *Let $\mathcal{S}$ be a specification. A model $\mathcal{I}$ of $\mathcal{S}$ is an initial model of $\mathcal{S}$ if and only if*

- *$\mathcal{I}$ has no junk, that is,*
  - *for every sort symbol ξ of $\mathcal{S}$ every element of $\xi^\mathcal{I}$ can be obtained as the denotation of a ground $\mathcal{S}$-term of sort ξ*
  - *for every function symbol f of $\mathcal{S}$ every element of $\mathcal{D}_f^\mathcal{I}$ can be obtained as the denotation of a tuple of ground $\mathcal{S}$-terms*

- *$\mathcal{I}$ has no confusion, that is, a ground Σ-equation is valid in $\mathcal{I}$ if and only if it is valid in every model of $\mathcal{S}$.*

3.4. Congruence Characterization of Equality

We use TER_Σ to denote the set of all Σ-terms. A *congruence on* TER_Σ is an equivalence relation $\sim$ on TER_Σ that is closed under replacement, that is,

$$s_1 \sim t_1 \ \wedge \ \cdots \ \wedge \ s_n \sim t_n \quad \Rightarrow \quad f(s_1, \ldots, s_n) \sim f(t_1, \ldots, t_n),$$

provided, that $f(s_1, \ldots, s_n)$ and $f(t_1, \ldots, t_n)$ are Σ-terms. A congruence $\sim$ on TER_Σ is *stable* if it is closed under instantiation, that is, if $s \sim t$ and θ is a Σ-substitution, then $\theta s \sim \theta t$.

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification. *$\mathcal{S}$-equality* on Σ-terms and Σ-substitutions is defined as follows:

$$s =_\mathcal{S} t : \Longleftrightarrow \ \mathcal{S} \models s \doteq t$$

$$\theta =_\mathcal{S} \psi : \Longleftrightarrow \ \forall x.\, \theta x =_\mathcal{S} \psi x.$$

One would expect that "$s =_\mathcal{S} t$" is a stable congruence on TER_Σ, as it is the case for untyped equational logic. In general, however, this is not the case since "$s =_\mathcal{S} t$" might not be transitive. To see this, consider the specification $\mathcal{S}$

$$\mathit{true}: \rightarrow \mathbf{bool}, \quad \mathit{false}: \rightarrow \mathbf{bool}, \quad \mathit{foo}: \mathbf{void} \rightarrow \mathbf{bool}$$
$$\mathit{foo}(x_{\mathbf{void}}) \doteq \mathit{true}, \quad \mathit{foo}(x_{\mathbf{void}}) \doteq \mathit{false}$$

where $x_{\mathbf{void}}$ is a variable having the sort **void**. Then $\mathit{true} =_\mathcal{S} \mathit{foo}(x_{\mathbf{void}})$ and $\mathit{foo}(x_{\mathbf{void}}) =_\mathcal{S} \mathit{false}$, but not $\mathit{true} =_\mathcal{S} \mathit{false}$ since $\mathit{true} \doteq \mathit{false}$ isn't valid in the initial model of $\mathcal{S}$. The problem is obviously caused by the sort **void** whose denotation in the initial model is empty.

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

We say that a congruence $\sim$ on TER_Σ *satisfies* a Σ-equation $s \doteq t$ if $s \sim t$. Congruences on TER_Σ are partially ordered by set inclusion.

Theorem 3.9. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification whose signature is fully inhabited. Then "$s =_\mathcal{S} t$" is the least stable congruence on TER_Σ satisfying every equation in $\mathcal{E}$.*

Corollary 3.10. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification whose signature is fully inhabited. Then:*

- *"$\theta =_\mathcal{S} \psi$" is an equivalence relation on the set of all Σ-substitutions*
- *if $s =_\mathcal{S} t$ and $\theta =_\mathcal{S} \psi$, then $\theta s =_\mathcal{S} \psi t$*
- *if $\theta =_\mathcal{S} \psi$ and $\theta' =_\mathcal{S} \psi'$, then $\theta\theta' =_\mathcal{S} \psi\psi'$.*

Proposition 3.11. *Let $\mathcal{S}$ be a specification whose signature is fully inhabited. If $\mathcal{S} \models E \rightarrow E'$ and $\mathcal{S} \models E' \rightarrow E''$, then $\mathcal{S} \models E \rightarrow E''$. Furthermore, $\mathcal{S} \models E$ if and only if $\mathcal{S} \models \emptyset \rightarrow E$.*

4. Algebraic Specification of Inheritance Hierarchies

In this section we show how inheritance hierarchies consisting of constructor and feature types can be specified in order-sorted equational logic. The initial model of such a specification is taken as the formalization of the specified hierarchy.

We start by assuming the following subclasses of sort and function symbols:

- There is a subclass of sort symbols whose elements are called *feature sort symbols.*
- There are two disjoint subclasses of function symbols whose elements are called *constructor symbols* and *feature symbols.* Every feature symbol is a unary function symbol. The letters l and k will always range over feature symbols.
- There are two disjoint subclasses of constructor symbols whose elements are called *explicit constructor symbols* and *implicit constructor symbols.* Every constructor symbol is either an explicit or an implicit constructor symbol.

To be able to relate features to implicit constructors, we assume a total order "$l \leq k$" on the set of all feature symbols. We will write $l < k$ if $l \leq k$ and $l \neq k$. To make our notation less tedious, we will often write feature sort instead of feature sort symbol, constructor instead of constructor symbol, and so on.

Given a signature Σ, we say that f *is a constructor of* ξ if f is a constructor and Σ contains a declaration $f\colon \vec{\eta} \to \eta$ such that $\eta \leq \xi$. We say that l *is a feature of* ξ if l is a feature and Σ contains a declaration $l\colon \eta \to \zeta$ such that $\xi \leq \eta$. We say that l is *the i-th feature of* ξ if l is a feature of ξ and the set $\{k \mid k \text{ is a feature of } \xi \text{ and } k \leq l\}$ has i elements. A term is called *canonical* if it is built only from variables and constructors (explicit or implicit). A term $l(x)$ consisting of a feature l and a variable x is called a *quasi-variable.*

A *basic signature* is a signature Σ satisfying the following conditions:

1. every function symbol of Σ is either a constructor or a feature
2. every constructor of Σ has exactly one declaration and is a constructor of a minimal sort of Σ
3. Σ is fully inhabited
4. only feature sorts have features
5. every subsort of a feature sort is a feature sort
6. every minimal feature sort ξ has exactly one (implicit) constructor $\overline{\xi}$
7. a constructor is implicit if and only if it is the constructor of a minimal feature sort
8. if l is a feature of ξ, then the set $\{\zeta \mid (l\colon \eta \to \zeta) \in \Sigma \ \wedge \ \xi \leq \eta\}$ has a minimum (denoted by $\sigma(l, \xi)$) with respect to the subsort order of Σ; we call $\sigma(l, \xi)$ the *sort of l for ξ.*
9. if ξ is a minimal feature sort and $l_1 < \cdots < l_n$ are the features of ξ, then Σ contains the declaration $\overline{\xi}\colon \sigma(l_1, \xi) \cdots \sigma(l_n, \xi) \to \xi$.

We will only admit inheritance hierarchies that can be specified with a basic signature. You might find it helpful to write down the signatures for the examples discussed in the previous sections and check that all conditions are satisfied.

PROPOSITION 4.1. *Every basic signature is regular. Furthermore, every canonical ground term of a basic signature has a minimal sort.*

PROPOSITION 4.2. *Let Σ be a basic signature and $l(f(s_1, \ldots, s_n))$ be a Σ-term such that l is a feature and f is a constructor. Then $\xi := \sigma f(s_1, \ldots, s_n)$ is a minimal feature sort and f is the implicit constructor of ξ, that is, $f = \overline{\xi}$.*

To relate features and implicit constructors, we need equations. Let Σ be a basic signature. A Σ-*projection* is a Σ-equation of the form

$$l(\overline{\xi}(x_1, \ldots, x_n)) = x_i$$

where $x_1, \ldots, x_n$ are distinct variables, ξ is a minimal feature sort, l is the i-th feature of ξ, and $\sigma x_j = \sigma(k, \xi)$ if k is the j-th feature of ξ. Note that, up to variable renaming, there is exactly one projection for every pair (ξ, l) such that ξ is a minimal feature sort and l is a feature of ξ. With $\mathcal{P}(\Sigma)$ we denote some set of Σ-projections that contains exactly one projection for every such pair (ξ, l). Furthermore, we call $\mathcal{S}(\Sigma) := (\Sigma, \mathcal{P}(\Sigma))$ the *specification associated with* Σ.

We will regard $\mathcal{S}(\Sigma)$ as a rewriting system by orienting its equations from left to right. The corresponding rewriting relation $\to_\Sigma$ is defined on Σ-terms and can be characterized as follows: $s \to_\Sigma t$ if and only if s has a subterm $l(\overline{\xi}(s_1, \ldots, s_n))$ and t can be obtained from s by replacing this subterm with s_i, where l is the i-th feature of ξ.

For the proof of the next theorem, we need some basic results on order-sorted rewriting, which we borrow from (Smolka et al., 1987).

THEOREM 4.3. *Let Σ be a basic signature. Then we have:*

1. *$\to_\Sigma$ is sort decreasing, that is, if $s \to_\Sigma t$, then $\sigma t \leq \sigma s$*
2. *$\to_\Sigma$ is terminating and confluent*
3. *$\mathcal{S}(\Sigma) \models s \doteq t$ if and only if s and t have the same normal form with respect to $\to_\Sigma$*
4. *a ground Σ-term is normal with respect to $\to_\Sigma$ if and only if it is canonical.*

PROOF. 1. It suffices to show that every instance $l(\overline{\xi}(s_1, \ldots, s_n)) \doteq s_i$ of a projection is sort decreasing, that is, $\sigma s_i \leq \sigma l(\overline{\xi}(s_1, \ldots, s_n))$. This follows from the conditions 9, 10, and 2 for basic signatures.

2. The rewriting relation $\to_\Sigma$ is terminating since rewriting with a projection decreases the size of a term. The confluence follows from a critical pair theorem in (Smolka et al., 1987) using that $\to_\Sigma$ is sort decreasing and terminating and that the left hand sides of the generating projections don't overlap.

3. For untyped rewriting (that is, there is just one sort), it is well-known that the claim holds if the rewriting relation $\to_\Sigma$ is confluent and terminating. Smolka et al. (1987) give an analogous result for sort decreasing order-sorted rewriting.

4. A canonical term is normal since it doesn't contain feature symbols. To show the other direction, suppose there is a normal ground term s that isn't canonical. Then s must contain a feature symbol and without loss of generality we can assume that s has the form $l(f(s_1, \ldots, s_n))$ where $f(s_1, \ldots, s_n)$ is canonical. Hence, $\xi := \sigma f(s_1, \ldots, s_n)$ must be a minimal feature sort such that $\overline{\xi} = f$. This implies that $l(f(s_1, \ldots, s_n))$ can be rewritten with respect to $\to_\Sigma$, which contradicts our assumption. □

COROLLARY 4.4. *Let Σ be a basic signature and $f(s_1, \ldots, s_m)$ and $g(t_1, \ldots, t_n)$ be two Σ-terms such that f and g are constructors. Then $\mathcal{S}(\Sigma) \models f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n)$ if and only if $f = g$ and $\mathcal{S}(\Sigma) \models s_i \doteq t_i$ for every i.*

COROLLARY 4.5. *Let Σ be a basic signature and s and t be two canonical Σ-terms. Then $\mathcal{S}(\Sigma) \models s \doteq t$ if and only if $s = t$.*

This completes our formalization of inheritance hierarchies. A system of constructor and feature types can now be specified by a basic signature Σ, which provides both syntax (syntactic Σ-objects) and semantics (the isomorphism class of initial models of $\mathcal{S}(\Sigma)$).

To make feature unification work, we need three additional requirements. We call a basic signature *admissible* if it satisfies the following conditions:

1. for every feature sort ξ such that $l_1, \ldots, l_n$ are the features of ξ and for every minimal $\vec{\xi} \leq \sigma(l_1, \xi) \cdots \sigma(l_n, \xi)$ there exists a minimal feature sort $\eta \leq \xi$ such that $\vec{\xi} \leq \sigma(l_1, \eta) \cdots \sigma(l_n, \eta)$

2. there are no infinite descending sort chains $\cdots \xi_3 \leq \xi_2 \leq \xi_1$
3. if two sorts have a common subsort, then they have a greatest common subsort.

Condition 1 is needed to ensure that every solved system (will be defined in Section 6) has a unifier. Condition 2 and 3 aren't really restrictions for practical applications: condition 2 is satisfied by every finite signature and condition 3 can always be satisfied by introducing the missing greatest common subsorts, which, semantically, don't introduce new elements since they are just the unions of already existing sorts.

5. Unification with Solved Forms

To formalize feature unification, we need a framework making a clear distinction between unifiers and solved systems representing unifiers. In the standard framework (Fages & Huet, 1986; Siekmann, 1984, 1988), the assumption that solved systems are (most general) unifiers is hard-wired in. Keeping apart conceptually unifiers and solved forms simplifies the framework, since the rather technical notion of a subsumption order on substitutions turns out to be superfluous.

GENERAL ASSUMPTION. *In this section we consider only fully inhabited regular signatures. Thus, "$s =_{\mathcal{S}} t$" and "$\theta =_{\mathcal{S}} \psi$" are equivalence relations.*

Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and E be a Σ-equation system. The set of *$\mathcal{S}$-unifiers of E* is defined as

$$\mathcal{U}_{\mathcal{S}}(E) := \{\theta \in \mathrm{SUB}_\Sigma \mid \mathcal{S} \models \theta E\},$$

where SUB_Σ is the set of all Σ-substitutions. A Σ-equation system E is called *$\mathcal{S}$-unifiable* or *consistent in $\mathcal{S}$* if it has at least one $\mathcal{S}$-unifier.

Technically, things work out nicely if we have the possibility of restricting unifiers to some set V of "primary variables", for instance, the variables actually occurring in the equation system E. Thus, we define the set of *$\mathcal{S}$-unifiers of E wrt V* as

$$\mathcal{U}^V_{\mathcal{S}}(E) := \{\theta|_V \mid \theta \in \mathcal{U}_{\mathcal{S}}(E)\} = \{\theta|_V \mid \mathcal{S} \models \theta E \ \wedge \ \theta \in \mathrm{SUB}_\Sigma\}.$$

We say that E *$\mathcal{S}$-subsumes E' wrt V* if $\mathcal{U}^V_{\mathcal{S}}(E') \subseteq \mathcal{U}^V_{\mathcal{S}}(E)$; we say that E and E' are *$\mathcal{S}$-equivalent wrt V* if $\mathcal{U}^V_{\mathcal{S}}(E) = \mathcal{U}^V_{\mathcal{S}}(E')$.

In our view, unification is the process of deciding whether an equation system E is unifiable; if E is unifiable, then unification tries to represent the unifiers of E by so-called solved systems. What a solved system precisely is cannot be said in general since the solved form depends on the specification and the unification technique to be applied. However, the literature on unification offers a variety of possibilities: the standard framework (Fages & Huet, 1986; Siekmann, 1984, 1988) employs idempotent substitutions as solved form; Martelli and Montanari (1982) and later Kirchner (1986) use acyclic systems of multiequations as solved form; Colmerauer's (1982, 1984) work on unification with "infinite terms" employs the equational representation of possibly nonidempotent substitutions as solved form; and work on disunification (Colmerauer, 1984; Kirchner & Lescanne, 1987) uses a disjunctive normal form.

The minimal requirements for a solved form are (1) that every solved system S determine a nonempty set $\mathcal{U}^V_{\mathcal{S}}(S)$ of substitutions, and (2) that it be decidable (to be practical, in polynomial time) whether a system is solved.

A set V of variables is called *admissible* if for every sort ξ there are infinitely many variables of sort ξ that are not in V. We assume that there are sufficiently many variables so that every finite set of variables is admissible. Note that the set of all variables is not admissible. The letter V will always denote an admissible set of variables. The admissibility restriction on V is necessary since many solved forms, for instance the one for feature unification, require the introduction of new auxiliary variables.

Next we generalize the notions of unitary and finitary unification (Siekmann, 1984, 1988) to unification with solved forms. Suppose a specification $\mathcal{S}$ and a solved form are given. Then we call $\mathcal{S}$ *unitary unifying* (wrt the given solved form) if for every $\mathcal{S}$-unifiable equation system E and every admissible V there exists a solved system S such that $\mathcal{U}_{\mathcal{S}}^{V}(E) = \mathcal{U}_{\mathcal{S}}^{V}(S)$. We call $\mathcal{S}$ *finitary unifying* (wrt the given solved form) if for every equation system E and every admissible V there exist solved systems $S_1, \ldots, S_n$ such that

$$\mathcal{U}_{\mathcal{S}}^{V}(E) = \mathcal{U}_{\mathcal{S}}^{V}(S_1) \cup \cdots \cup \mathcal{U}_{\mathcal{S}}^{V}(S_n).$$

An *$\mathcal{S}$-approximation* of an equation system E wrt V is a set A of solved systems such that

$$\mathcal{U}_{\mathcal{S}}^{V}(A) := \bigcup_{S \in A} \mathcal{U}_{\mathcal{S}}^{V}(S) \quad \subseteq \quad \mathcal{U}_{\mathcal{S}}^{V}(E).$$

An $\mathcal{S}$-approximation A of E wrt V is *complete* if $\mathcal{U}_{\mathcal{S}}^{V}(A) = \mathcal{U}_{\mathcal{S}}^{V}(E)$. An $\mathcal{S}$-approximation A is *minimal wrt $\mathcal{S}$ and V* if $\mathcal{U}_{\mathcal{S}}^{V}(A) \neq \mathcal{U}_{\mathcal{S}}^{V}(A - \{S\})$ for every $S \in A$.

A *unification calculus for a specification* $\mathcal{S}$ is a formal system that allows us, for every E and V, to enumerate complete $\mathcal{S}$-approximations of E wrt V. A *unification algorithm for a specification* $\mathcal{S}$ computes, for every E and V, a complete (possibly minimal) $\mathcal{S}$-approximation of E wrt V.

A major motivation for employing a nonstandard solved form is the fact that for some applications—for instance, feature unification or disunification—the "right" solved form results in unitary unification while the standard approach results in nonunitary unification. For practical applications, unitary unification is very desirable since nonunitary unification adds a source of nondeterminism blowing up the involved search spaces.

The next two propositions will be used in proofs.

PROPOSITION 5.1. *Let $\mathcal{S}$ be a specification. Then:*

- *if $\theta \in \mathcal{U}_{\mathcal{S}}^{V}(E)$, then $(\psi\theta)|_V \in \mathcal{U}_{\mathcal{S}}^{V}(E)$ for every ψ*
- $$\begin{aligned} \mathcal{U}_{\mathcal{S}}^{V}(E) \subseteq \mathcal{U}_{\mathcal{S}}^{V}(E') \quad &\Longleftrightarrow \quad \forall\theta \in \mathcal{U}_{\mathcal{S}}(E)\ \exists\theta' \in \mathcal{U}_{\mathcal{S}}(E').\ \theta|_V = \theta'|_V \\ &\Longleftrightarrow \quad \forall\theta \in \mathcal{U}_{\mathcal{S}}(E)\ \exists\theta' \in \mathcal{U}_{\mathcal{S}}(E').\ \theta|_V =_{\mathcal{S}} \theta'|_V \end{aligned}$$
- *if $\mathcal{S} \models E \rightarrow \psi E'$ and $\mathcal{D}\psi \cap V = \emptyset$, then $\mathcal{U}_{\mathcal{S}}^{V}(E) \subseteq \mathcal{U}_{\mathcal{S}}^{V}(E')$.*

Let $\langle x \doteq s \rangle$ be an idempotent substitution and E be an equation system containing the variable x. Then we say that $E \;\&\; x \doteq s$ can be obtained from $\langle x \doteq s \rangle E$ by *unfolding*.

PROPOSITION 5.2. (Unfolding) *Let $E \;\&\; x \doteq s$ be obtainable from $\langle x \doteq s \rangle E$ by unfolding. Then $\mathcal{U}_{\mathcal{S}}^{V}(E \;\&\; x \doteq s) = \mathcal{U}_{\mathcal{S}}^{V}(\langle x \doteq s \rangle E)$ if $x \notin V$.*

PROOF. The claim follows from $\mathcal{S} \models \langle x \doteq s \rangle E \rightarrow \langle x \doteq s \rangle (E \;\&\; x \doteq s)$ and $\mathcal{S} \models (E \;\&\; x \doteq s) \rightarrow \langle x \doteq s \rangle E$ by the third part of the previous proposition. □

We end this section with a few definitions and propositions showing that our unification framework is equivalent to the standard one if we employ idempotent substitutions as solved form.

An equation $s \doteq t$ is *trivial* if $s = t$; an equation system is *trivial* if all of its equations are trivial.

PROPOSITION 5.3. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and ϕ be an idempotent Σ-substitution. Then:*

- *$\phi[\phi]$ is a trivial equation system; in particular, ϕ is an $\mathcal{S}$-unifier of $[\phi]$*
- *$\theta \in \mathcal{U}_{\mathcal{S}}([\phi]) \iff \mathcal{S} \models \theta[\phi] \iff \theta =_{\mathcal{S}} \theta\phi \iff \exists\psi.\, \theta =_{\mathcal{S}} \psi\phi$.*

PROOF. The only not obviously valid claim is $\theta =_{\mathcal{S}} \psi\phi \Rightarrow \theta =_{\mathcal{S}} \theta\phi$. Suppose, we have $\theta =_{\mathcal{S}} \psi\phi$. Then $\theta =_{\mathcal{S}} \psi\phi =_{\mathcal{S}} \psi\phi\phi =_{\mathcal{S}} \theta\phi$ by the assumption, the idempotence of ϕ, and again the assumption. □

PROPOSITION 5.4. *Let $\mathcal{S} = (\Sigma, \mathcal{E})$ be a specification and ϕ be an idempotent Σ-substitution. Then $\theta \in \mathcal{U}_{\mathcal{S}}^{V}([\phi])$ if and only if $\exists\psi\ \forall x \in V.\, \theta x =_{\mathcal{S}} \psi\phi x$.*

PROOF. "⇒" Suppose $\theta = \theta'|_V$ and $\theta' \in \mathcal{U}_{\mathcal{S}}([\phi])$. Then $\theta' =_{\mathcal{S}} \theta'\phi$ by the previous proposition, which yields the claim.

"⇐" Suppose $\theta = \theta'|_V$ and $\theta' =_{\mathcal{S}} \psi\phi$. Then $\theta' \in \mathcal{U}_{\mathcal{S}}([\phi])$ by the previous proposition, which yields the claim. □

Another point, where we feel the standard notion of unification is not general enough, is the syntactic notion of a unifier, which should be replaced by the semantic notion of a solution. Given a signature Σ, a Σ-algebra $\mathcal{A}$, and a Σ-equation system E, an *$\mathcal{A}$-solution of E* can be defined as a $(\mathcal{V}(E), \mathcal{A})$-assignment α such that $\mathcal{A}, \alpha \models E$. As long as we want to solve with respect to some "no junk" model—for instance, the initial model—substitutions and thus unifiers suffice to represent all solutions. However, if we want to solve with respect to a model containing elements not denotable by ground terms (for instance, "infinite terms"), which in fact would be necessary to exactly capture Aït-Kaci's (1984, 1986) ψ-unification, substitutions and thus unifiers don't suffice anymore to express all solutions.

We would like to remark that Colmerauer (1984) uses a nonstandard solved form and also solves with respect to a noninitial model providing for "infinite terms".

6. Feature Unification

Feature unification is a method for deciding whether an equation system over an admissible basic signature Σ is unifiable in $\mathcal{S}(\Sigma)$. This is done by simplifying the given equation system with a collection of unification rules. If the simplified system is in a certain solved form, then the original system is unifiable and its unifiers are exactly the unifiers of the simplified system; if the simplified system isn't in solved form, then the original system isn't unifiable.

6.1. The Solved Form

We call an equation $s \doteq t$ *trimmed* if it contains no implicit constructors, the right-hand side t is a canonical term, and the left-hand side s is either a canonical term or a

quasi-variable. A *trimmed equation system* is an equation system whose equations are all trimmed. The letter T will always range over trimmed equation systems.

A trimmed equation system T is called *quasi-solved* if it has the form

$$x_1 \doteq s_1 \,\&\, \cdots \,\&\, x_m \doteq s_m \,\&\, l_1(y_1) \doteq t_1 \,\&\, \cdots \,\&\, l_n(y_n) \doteq t_n$$

where $m, n \geq 0$ and

1. $x_1, \ldots, x_m$ are pairwise distinct variables different from $y_1, \ldots, y_n$ (where $y_1, \ldots, y_n$ are not necessarily distinct)
2. $l_1(y_1), \ldots, l_n(y_n)$ are pairwise distinct quasi-variables
3. if s_i is a variable, then x_i occurs only once in T
4. $\sigma s_i \leq \sigma x_i$ and $\sigma t_j \leq \sigma l_j(y_j)$ for all i and j.

On the variables of a quasi-solved equation system T we define a *dependency relation* $\rightarrow_T$ as follows: $x \rightarrow_T y$ if and only if T contains an equation $s \doteq t$ such that x occurs in s and y occurs in t. A quasi-solved equation system T is called *solved* if its dependency relation is acyclic. The letter S will always range over solved equation systems. System (4) in Section 2.3 is an example of a solved equation system.

A *canonical substitution* is a substitution θ such that θx is a canonical term for every $x \in \mathcal{D}\theta$. A *ground substitution* is a substitution θ such that θx is a ground term for every $x \in \mathcal{D}\theta$.

THEOREM 6.1. *Let Σ be a basic signature. Then a Σ-equation system has an $\mathcal{S}(\Sigma)$-unifier if and only if it has a canonical ground $\mathcal{S}(\Sigma)$-unifier.*

PROOF. Let $\mathcal{S}(\Sigma) \models \theta E$. We have to show that E has a canonical ground $\mathcal{S}(\Sigma)$-unifier. Since Σ is fully inhabited, there exists a substitution ψ such that $\psi\theta$ is a ground substitution. Let ϕ be obtained from $\psi\theta$ by normalization with respect to $\rightarrow_\Sigma$, that is, ϕx is the normal form of $\psi\theta x$ for all x. Then ϕ is a canonical ground substitution such that $\psi\theta =_{\mathcal{S}(\Sigma)} \phi$. Hence, ϕ is an $\mathcal{S}(\Sigma)$-unifier of E, which yields the claim. □

THEOREM 6.2. *Let Σ be an admissible basic signature. Then, a quasi-solved Σ-equation system is $\mathcal{S}(\Sigma)$-unifiable if and only if it is solved.*

PROOF. 1. Let T be an $\mathcal{S}(\Sigma)$-unifiable quasi-solved equation system. We have to show that the dependency relation of T is acyclic. Suppose the dependency relation of T is cyclic. Since T is $\mathcal{S}(\Sigma)$-unifiable, there exists a canonical substitution θ such that $\mathcal{S}(\Sigma) \models \theta T$. Furthermore, there must be an edge $x \rightarrow_T y$ of a cycle and an equation $s \doteq t$ in T such that x occurs in s, y occurs in t, and either s or t is not a variable. To obtain a contradiction, it suffices to show that $|\theta x| > |\theta y|$. There are two cases:

1.1. $s \doteq t$ has the form $x \doteq f(s_1, \ldots, s_n)$ and $y \in \mathcal{V}(s_i)$. Since $\mathcal{S}(\Sigma) \models \theta T$ and both sides of $\theta x \doteq \theta f(s_1, \ldots, s_n)$ are canonical, we have $\theta x = f(\theta s_1, \ldots, \theta s_n)$. Hence, $|\theta x| = |f(\theta s_1, \ldots, \theta s_n)| > |\theta s_i| \geq |\theta y|$.

1.2. $s \doteq t$ has the form $l(x) \doteq u$ and $y \in \mathcal{V}(u)$. Since $\mathcal{S}(\Sigma) \models l(\theta x) \doteq \theta u$ and θu is canonical, θu must be the normal form of $l(\theta x)$. Since θx is canonical, we have $l(\theta x) = l(\overline{\xi}(s_1, \ldots, s_n))$, where ξ is a minimal subsort of σx. Thus, $s_i = \theta u$ for some i, which yields $|\theta x| > |s_i| = |\theta u| \geq |\theta y|$.

2. Let S be a solved equation system. We prove by induction on the number n of quasi-variables in S that S is $\mathcal{S}(\Sigma)$-unifiable.

$n = 0$. Then $S = [\theta]$ for some substitution θ. Since the dependency relation of S is acyclic, we have $\theta^m\theta = \theta^m$ for some $m > 0$. Thus, θ^m is an $\mathcal{S}(\Sigma)$-unifier of S.

$n > 0$. Then $S = [\theta]\ \&\ S_1$, where S_1 is a solved system whose left-hand sides are all quasi-variables. From the base case we know that there exists a natural number m such that $\theta^m = \theta^m\theta$. Let $S_2 := \theta^m S_1$. It is easy to verify that S_2 is solved and that S is $\mathcal{S}(\Sigma)$-unifiable if S_2 is $\mathcal{S}(\Sigma)$-unifiable. Thus it suffices to show that S_2 is $\mathcal{S}(\Sigma)$-unifiable. Since S_2 is solved and every left-hand side of S_2 is a quasi-variable, there exists a variable x occurring in a left-hand side of S_2 such that for every equation $l(x) \doteq t$ in S_2 no variable occurring in t occurs in a left-hand side of S_2. Recall that every canonical term that is not a variable has a minimal sort. Hence there exists a Σ-substitution ϕ mapping variables to variables such that $S_3 := \phi S_2$ is solved, $x \notin \mathcal{D}\phi$, and for every equation $l(x) \doteq s$ in S_3 the right-hand side s has a minimal sort. Since S_2 is $\mathcal{S}(\Sigma)$-unifiable if S_3 is $\mathcal{S}(\Sigma)$-unifiable, it suffices to show that S_3 is $\mathcal{S}(\Sigma)$-unifiable. Since Σ is admissible, we know by the first admissibility condition that there exists a minimal feature sort $\xi \leq \sigma x$ such that $\sigma t \leq \sigma(l, \xi)$ for every $(l(x) \doteq t) \in S_3$. Let $l_1 < \cdots < l_m$ be the features of ξ and choose pairwise distinct variables $x_1, \ldots, x_n$ not occurring in S_3 such that $\sigma x_i = \sigma(l_i, \xi)$. Define $E_4 := \langle x \doteq \overline{\xi}(x_1, \ldots, x_m)\rangle S_3$ and observe that S_3 is $\mathcal{S}(\Sigma)$-unifiable if E_4 is $\mathcal{S}(\Sigma)$-unifiable. Now obtain S_5 from E_4 by rewriting all left-hand sides that aren't quasi-variables and observe that E_4 is $\mathcal{S}(\Sigma)$-unifiable if S_5 is $\mathcal{S}(\Sigma)$-unifiable. One can verify that S_5 is solved. With that the induction hypothesis applies and yields that S_5 is $\mathcal{S}(\Sigma)$-unifiable, which proves the claim. □

If a basic signature doesn't satisfy the first admissibility condition, there can be solved systems that are not unifiable. To see this, consider the signature defined by the equations

$$\mathbf{A} := [l : \mathbf{C}], \quad \mathbf{B} := \mathbf{A}[l : \mathbf{D}], \quad \mathbf{C} := \mathbf{D} \cup \mathbf{E}, \quad \mathbf{D} := \{d\}, \quad \mathbf{E} := \{e\}.$$

In this signature, the solved system $l(x_\mathbf{A}) \doteq e$ ($x_\mathbf{A}$ is a variable of sort $\mathbf{A}$) has no unifier since the implicit constructor of the feature sort $\mathbf{B}$ cannot take e as an argument.

We could get rid of the admissibility condition (1) by adding a "type consistency" requirement to the definition of a solved system. However, such an extra-condition would blow up the worst-case complexity of deciding whether an equation system is solved.

6.2. The Unification Rules

Let Σ be an admissible basic signature. In this subsection we will show that an equation system E not containing implicit constructors is $\mathcal{S}(\Sigma)$-unifiable if and only if there exists a solved system S such that $\mathcal{U}_\mathcal{S}^V(E) = \mathcal{U}_\mathcal{S}^V(S)$. Furthermore, we will give a method, called *feature unification*, with which we can decide whether E is $\mathcal{S}(\Sigma)$-unifiable, and, in case it is, compute a solved system representing its $\mathcal{S}(\Sigma)$-unifiers.

Let us first discuss the significance of this result. Feature unification is a form of E-unification since the unifiers are defined with respect to a specification with equational axioms. Since $\mathcal{S}(\Sigma)$ is a sort decreasing, confluent and terminating rewriting system, (order-sorted) narrowing would be a complete unification method for $\mathcal{S}(\Sigma)$. In contrast to feature unification, however, narrowing employs idempotent substitutions as solved form, thus resulting in nonunitary unification. For instance, consider the hierarchy in Figures 1.1 and 2.2 and the equation $age(O) = 5$, which is in solved form with respect to feature unification. Narrowing, however, will produce four independent most general unifiers:

$O = con\_house(5, T)$

$O = con\_other\_car(5, S, F)$

$O = con\_gas\_car(5, S, G)$

$O = con\_electric\_car(5, S, E)$.

Since narrowing with respect to $\mathcal{S}(\Sigma)$ strictly reduces the number of feature symbol occurrences, all narrowing trees must be finite. Thus, order-sorted narrowing yields a finitary unification method for $\mathcal{S}(\Sigma)$. Narrowing could be used to postprocess a solved system produced by feature unification, but $age(O) = 5$ certainly is a far better representation of the obtained result than the four most general unifiers.

The following proposition, which follows from the unfolding proposition, allows us to compile equation systems into trimmed ones.

PROPOSITION 6.3. *Let Σ be a basic signature and E be a Σ-equation system not containing implicit constructors. Then we can obtain by unfolding a trimmed equation system T from E such that $\mathcal{U}^V_{\mathcal{S}(\Sigma)}(E) = \mathcal{U}^V_{\mathcal{S}(\Sigma)}(T)$.*

The unification rules shown in Figure 6.1 apply to trimmed equation systems over admissible basic signatures. To solve a trimmed equation system, the rules are applied as long as a rule is applicable. If the so obtained final equation system is solved, then the initial system is unifiable and its unifiers are represented by the final system; if the final system is not solved, then the initial system is not unifiable.

Note that the rules W1, W2, and I depend on the underlying signature. If T is a trimmed equation system and E is obtained from T by one of the unification rules, then E is again a trimmed equation system.

THEOREM 6.4. (Feature Unification) *Let Σ be a admissible basic signature. Then the following holds:*

1. *(Invariance) If $T \xrightarrow{u}_V T'$, then $\mathcal{U}^V_{\mathcal{S}(\Sigma)}(T) = \mathcal{U}^V_{\mathcal{S}(\Sigma)}(T')$.*
2. *(Termination) There are no infinite derivations $T_1 \xrightarrow{u}_V T_2 \xrightarrow{u}_V \cdots$.*
3. *(Completeness) T is $\mathcal{S}(\Sigma)$-unifiable if and only if there exists a solved system S such that $T \xrightarrow{u}{}^*_V S$.*

PROOF. *1. (Invariance)* Let $T \xrightarrow{u}_V T'$. Then its easy to verify that $\mathcal{S}(\Sigma) \models T' \to T$, which yields the direction $\mathcal{U}^V_{\mathcal{S}(\Sigma)}(T') \subseteq \mathcal{U}^V_{\mathcal{S}(\Sigma)}(T)$. To show $\mathcal{U}^V_{\mathcal{S}(\Sigma)}(T) \subseteq \mathcal{U}^V_{\mathcal{S}(\Sigma)}(T')$, let the unification rule employed in $T \xrightarrow{u}_V T'$ be:

D. Then the claim follows from Corollary 14 in Section 4.

M1, M2, I, O, or E. Then it's easy to verify that $\mathcal{S}(\Sigma) \models T \to T'$, which yields the claim.

W1. Then $T = (x \doteq y \;\&\; T'')$, $T' = (x \doteq z \;\&\; y \doteq z \;\&\; \langle x \doteq z \;\&\; y \doteq z \rangle T'')$, $z \notin V \cup \mathcal{V}(x \doteq y \;\&\; T'')$, and σz is the greatest common subsort of σx and σy. Let $\theta \in \mathcal{U}_{\mathcal{S}(\Sigma)}(T)$ be normal. Then $\theta x = \theta y$ since $\mathcal{S}(\Sigma) \models \theta x \doteq \theta y$ and θ is normal. Thus, $\sigma\theta x \leq \sigma z$ since $\sigma\theta x \leq \sigma x$ and $\sigma\theta x = \sigma\theta y \leq \sigma y$. Define $\psi := \theta[z \leftarrow \theta x]$. Now one verifies easily that $\psi \in \mathcal{U}_{\mathcal{S}(\Sigma)}(T')$, which yields the claim.

W2. Then $T = (l(x) \doteq y \;\&\; T'')$, $T' = (y \doteq z \;\&\; \langle y \doteq z \rangle (l(x) \doteq y \;\&\; T''))$, $z \notin V \cup \mathcal{V}(l(x) \doteq y \;\&\; T'')$, and σz is the greatest common subsort of $\sigma l(x)$ and σy. Let $\theta \in \mathcal{U}_{\mathcal{S}(\Sigma)}(T)$ be normal. Then θy is the normal form of $\theta l(x)$ since $\mathcal{S}(\Sigma) \models \theta l(x) \doteq \theta y$. Thus, $\sigma\theta y \leq \sigma z$ since $\sigma\theta y \leq \sigma y$ and $\sigma\theta y \leq \sigma\theta l(x) \leq \sigma l(x)$. Define $\psi := \theta[z \leftarrow \theta y]$. Now one verifies easily that $\psi \in \mathcal{U}_{\mathcal{S}(\Sigma)}(T')$, which yields the claim.

2. (Termination) Suppose there is an infinite derivation $T_1 \xrightarrow{u}_V T_2 \xrightarrow{u}_V \cdots$.

2.1. Consider the number of quasi-variable occurrences in T_1. This number is increased by no rule and decreased by M2. Thus, we can assume that the infinite derivation doesn't employ M2.

2.2. Let us call a variable x isolated in an equation system E if x occurs only once in E and E contains an equation $x \doteq s$ such that $\sigma s \leq \sigma x$. Now consider the number of

Decomposition

$(D)\quad f(s_1,\ldots,s_n) \doteq f(t_1,\ldots,t_n)\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad s_1 \doteq t_1\ \&\cdots\&\ s_n \doteq t_n\ \&\ T$

if f is a constructor

Merging

$(M1)\quad x \doteq s\ \&\ x \doteq t\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad x \doteq s\ \&\ s \doteq t\ \&\ T$

if s and t aren't variables and $|s| \leq |t|$

$(M2)\quad l(x) \doteq s\ \&\ l(x) \doteq t\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad l(x) \doteq s\ \&\ s \doteq t\ \&\ T$

Weakening

$(W1)\quad x \doteq y\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad x \doteq z\ \&\ y \doteq z\ \&\ \langle x \doteq z\ \&\ y \doteq z\rangle T$

if not $\sigma y \leq \sigma x$ and z is a variable not occurring in V or $x \doteq y\ \&\ T$ such that σz is the greatest common subsort of σx and σy

$(W2)\quad l(x) \doteq y\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad y \doteq z\ \&\ \langle y \doteq z\rangle(l(x) \doteq y\ \&\ T)$

if not $\sigma y \leq \sigma l(x)$ and z is a variable not occurring in V or $l(x) \doteq y\ \&\ T$ such that σz is the greatest common subsort of $\sigma l(x)$ and σy

Isolation

$(I)\quad x \doteq y\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad x \doteq y\ \&\ \langle x \doteq y\rangle T$

if x is different from y, x occurs in T, and $\sigma y \leq \sigma x$

Orientation

$(O)\quad s \doteq x\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad x \doteq s\ \&\ T$

if s is neither a variable nor a quasi-variable

Elimination

$(E)\quad x \doteq x\ \&\ T \quad \xrightarrow{\mathrm{u}}_V \quad T$

FIGURE 6.1. The feature unification rules. The notation assumes that the conjunction symbol $T\ \&\ T'$ is associative, commutative, and satisfies $T\ \&\ \emptyset = T$.

nonisolated variables in T_1. This number is increased by no rule and decreased by W1 and I (W1 introduces a new nonisolated variable and isolates two variables that were nonisolated before). Thus, we can assume that the infinite derivation employs only the rules D, M1, W2, O, and E.

2.3. Let k be a fixed natural number greater than the arity of every function symbol occurring in T_1. Let us call an equation nonflat if one of its sides is a canonical, nonvariable term. Furthermore, define the complexity of an equation system E as $\sum_{i=1}^{n} k^{\max\{|s_i|,|t_i|\}}$, where $s_1 \doteq t_1, \ldots, s_n \doteq t_n$ are the nonflat equation occurrences of E. It is easy to verify that the complexity of an equation system is not changed by the rules M1, W2, O, or E. The decomposition rule D, however, decreases this complexity since

$$k^{\max\{(1+\sum_{i=1}^{n}|s_i|),\,(1+\sum_{i=1}^{n}|t_i|)\}} > \sum_{i=1}^{n} k^{\max\{|s_i|,|t_i|\}}$$

for $n < k$. Thus, we can assume that the infinite derivation employs only the rules M1, W2, O, and E.

2.4. Consider the number of equation occurrences in T_1 whose one side is a variable and whose other side is neither a variable nor a quasi-variable. This number is decreased by M1 and not changed by W2, O, and E. Thus, we can assume that the infinite derivation employs only the rules W2, O, and E.

2.5. Consider the number of equation occurrences in T_1 whose left-hand side is neither a variable nor a quasi-variable. This number is not changed by W2 and E and decreased by O. Thus, we can assume that the infinite derivation employs only the rules W2 and E.

2.6. Let $\| x \|$ be the maximal length of a descending sort chain issuing from σx. Since there are no infinite descending sort chains by the admissibility condition 2, we know that $\| x \|$ is always finite. Now consider the sum $\sum \| y \|$ over all equation occurrences in T_1 that have the form $l(x) \doteq y$. This sum is decreased by W2 and not changed by E. Thus, we can assume that the infinite derivation employs only the rule E, which is impossible.

3. (Completeness) Suppose no unification rule applies to T and θ is a canonical $\mathcal{S}(\Sigma)$-unifier of T. To prove the claim, it suffices to show that T is quasi-solved. We show this in eight steps.

3.1. T contains no equation $x \doteq y$ such that not $\sigma y \leq \sigma x$. To see this, suppose T contains such an equation. Since $\mathcal{S}(\Sigma) \models \theta x \doteq \theta y$ and both sides are canonical, we have $\theta x = \theta y$, which implies that $\sigma\theta x$ is a common subsort of σx and σy. Thus, σx and σy have a greatest common subsort by the admissibility condition 3. This, however, is impossible since the weakening rule W1 isn't applicable.

3.2. T contains no equation $x \doteq f(s_1, \ldots, s_n)$ such that not $\sigma f(s_1, \ldots, s_n) \leq \sigma x$. To see this, suppose T contains such an equation. Since $\mathcal{S}(\Sigma) \models \theta x \doteq \theta f(s_1, \ldots, s_n)$ and both sides are canonical, we have $\theta x = \theta f(s_1, \ldots, s_n)$. This yields $\sigma\theta x = \sigma f(\theta s_1, \ldots, \theta s_n) = \sigma f(s_1, \ldots, s_n)$ since in basic signatures no constructor has more than one declaration. This, however, implies $\sigma f(s_1, \ldots, s_n) = \sigma\theta x \leq \sigma x$, which contradicts our assumptions.

3.3. T contains no equation $l(x) \doteq s$ such that not $\sigma s \leq \sigma l(x)$. To see this, suppose T contains such an equation. Since $\mathcal{S}(\Sigma) \models \theta l(x) \doteq \theta s$ and θs is canonical, we know that θs is the normal form of $\theta l(x)$. Thus, $\sigma\theta s \leq \sigma\theta l(x) \leq \sigma l(x)$ since rewriting and substitution application are sort decreasing. Now we distinguish two cases. (a) If s isn't a variable,

then $\sigma s = \sigma\theta s$ since in basic signatures no constructor has more than one declaration. Thus $\sigma s \leq \sigma l(x)$, which contradicts our assumptions. (b) If s is a variable, then $\sigma l(x)$ and σs have a greatest common subsort by the admissibility condition 3 since they have the common subsort $\sigma\theta s$. This, however, is impossible since the weakening rule W2 isn't applicable.

3.4. Every equation of T has either the form $x \doteq s$ or $l(x) \doteq s$ where s is canonical. To see this, note first that T can't contain an equation $f(s_1, \ldots, s_m) \doteq x$ where f is a constructor, since the orientation rule isn't applicable. Now suppose T contains an equation $f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n)$ such that f and g are constructors. Since $\mathcal{S}(\Sigma) \models f(\theta s_1, \ldots, \theta s_m) \doteq g(\theta t_1, \ldots, \theta t_n)$, we know that $f = g$. This, however, is impossible since the decomposition rule isn't applicable.

3.5. No left hand side $l(x)$ occurs more than once in T, since otherwise the merging rule M2 were applicable.

3.6. T contains no equation $x \doteq y$ such that x occurs more than once in T. To see this, suppose T contains such an equation. Since the elimination rule isn't applicable, we know that x is different from y. Because of 3.1 we know that $\sigma y \leq \sigma x$. This, however, is impossible since the isolation rule isn't applicable.

3.7. No two left-hand sides of T are equal. To see this, suppose T contains two equations whose left-hand sides are equal. Because of 3.5 and 3.6 we know that T must contain two equations $x \doteq s$ and $x \doteq t$ such that neither s nor t is a variable. This, however, is impossible since the merging rule M1 isn't applicable.

3.8 T doesn't contain two equations $x \doteq s$ and $l(x) \doteq t$. To see this, suppose T contains two equations $x \doteq s$ and $l(x) \doteq t$. Because of 3.6, we know that s has the form $f(s_1, \ldots, s_n)$. Since T is trimmed, we know that f is an explicit constructor. Because of 3.2, we know that f is an explicit constructor of σx. Because $l(x)$ is well-formed, we also know that σx is a feature sort. This, however, is impossible since feature sorts have no explicit constructors. □

Our feature unification rules extend Colmerauer's (1982) rules for unification with infinite terms. To deal with subsorts, we had to add the weakening rule W1, and to deal with features, we had to add the merging rule M2 and the weakening rule W2. The completeness of Colmerauer's rules for infinite terms suggests that feature unification is complete for infinite terms as well, provided, we employ quasi-solved systems as solved form. Feature unification with infinite terms is more efficient than feature unification without infinite terms since it doesn't require testing the produced quasi-solved system for cycles.

The check $|s| \leq |t|$ in the merging rule M1 is necessary to ensure termination, as the following example of Colmerauer (1982) shows:

$$\begin{aligned}
x \doteq y \;\&\; y \doteq h(h(y)) \;\&\; y \doteq h(y) \quad &\xrightarrow{\mathrm{u}}_V \quad \textit{by M1 without test} \\
x \doteq y \;\&\; y \doteq h(h(y)) \;\&\; h(h(y)) \doteq h(y) \quad &\xrightarrow{\mathrm{u}}_V \quad \textit{by D} \\
x \doteq y \;\&\; y \doteq h(h(y)) \;\&\; h(y) \doteq y \quad &\xrightarrow{\mathrm{u}}_V \quad \textit{by O} \\
x \doteq y \;\&\; y \doteq h(h(y)) \;\&\; y \doteq h(y). &
\end{aligned}$$

However, the check $|s| \leq |t|$ can be avoided if we unfold the equation system to be solved such that every nonvariable right-hand side has the form $f(x_1, \ldots, x_n)$.

Feature unification in signatures without feature sorts is order-sorted unification (Walther, 1984, 1988; Meseguer et al., 1988; Smolka et al., 1987), which generalizes unsorted term unification. Feature unification in signatures without explicit constructors

$(E1)$ $x \doteq s \;\&\; S \xrightarrow{e}_V S$

if x doesn't occur in V or S

$(E2)$ $x \doteq y \;\&\; S \xrightarrow{e}_V S$

if $\sigma x = \sigma y$ and y doesn't occur in V or S

$(E3)$ $l(x) \doteq s \;\&\; S \xrightarrow{e}_V S$

if x doesn't occur in V or a right-hand side of S

$(E4)$ $l(x) \doteq y \;\&\; S \xrightarrow{e}_V S$

if $\sigma l(x) = \sigma y$ and y doesn't occur in V or S

FIGURE 6.2. The elimination rules. The rules are applicable to solved equation systems over admissible basic signatures.

is ψ-unification (Aït-Kaci 1984, 1986; Aït-Kaci & Nasr 1986), where, however, "infinite ψ-terms" aren't admitted. Thus, feature unification clarifies the relationship between order-sorted and ψ-unification, an open question originally motivating our research.

Figure 6.2 shows some rules for the elimination of redundant equations in solved equation systems. For instance, the new equations introduced by the weakening rules may become redundant after some further simplification steps. Note that the deletion of an equation in a solved system yields again a solved equation system.

THEOREM 6.5. *Let Σ be an admissible basic signature and S' be a solved equation system. If $S' \xrightarrow{e}_V S$, then $\mathcal{U}^V_{\mathcal{S}(\Sigma)}(S') = \mathcal{U}^V_{\mathcal{S}(\Sigma)}(S)$.*

PROOF. Since $\mathcal{S}(\Sigma) \models S' \to S$, we have $\mathcal{U}^V_{\mathcal{S}(\Sigma)}(S') \subseteq \mathcal{U}^V_{\mathcal{S}(\Sigma)}(S)$. To show the other direction, suppose θ is an $\mathcal{S}(\Sigma)$-unifier of S. Let the rule employed in $S' \xrightarrow{e}_V S$ be:

E1. Then $S' = (x \doteq s \;\&\; S)$ and x doesn't occur in V or S. Since the dependency relation of S' is acyclic, we know that x doesn't occur in s. Thus, $\theta[x \leftarrow \theta s]$ is an $\mathcal{S}(\Sigma)$-unifier of S', which yields the claim.

E2. Then $S' = (x \doteq y \;\&\; S)$, $\sigma x = \sigma y$, and y doesn't occur in V or S. Then $\theta[y \leftarrow \theta x]$ is an $\mathcal{S}(\Sigma)$-unifier of S', which yields the claim.

E3. Then $S' = (l(x) \doteq s \;\&\; S)$ and x doesn't occur in V or a right-hand side of S. Since S' is solved, we also know that x occurs in no right-hand side of S', and that every left-hand side of S' in which x occurs is a quasi-variable $k(x)$. Let $l_1 < \cdots < l_n$ be the features of σx. By the admissibility condition 1 we know that there is a minimal feature sort $\xi \leq \sigma x$ such that $\sigma(l_i, \xi) = \sigma(l_i, \sigma x) = \sigma l_i(x)$ for all i. Now define $s_1, \ldots, s_n$ as follows: $s_i := l_i(x)$ if $l_i(x)$ doesn't occur in S'; otherwise, let s_i be the term such that $l_i(x) \doteq s_i$ occurs in S'. Then $\theta[x \leftarrow \overline{\xi}(s_1, \ldots, s_n)]$ is an $\mathcal{S}(\Sigma)$-unifier of S', which yields the claim.

E4. Then $S' = (l(x) \doteq y \;\&\; S)$, $\sigma l(x) = \sigma y$, and y doesn't occur in V or S. Since S'

is solved, we know that x is different from y. Thus, $\theta[y \leftarrow l(x)]$ is an $\mathcal{S}(\Sigma)$-unifier of S', which yields the claim. □

The research reported in this paper started while Gert Smolka was visiting the AI Program of MCC in the summer of 1986. Gert Smolka's research was also funded by the Bundesminister für Forschung und Technologie under grant ITR8501A. We are grateful to the members of the Unification Group at Kaiserslautern for many discussions on the topic of this paper.

References

Aït-Kaci, H. (1984). *A Lattice Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures.* Ph.D. Thesis, University of Pennsylvania.

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

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

Burstall, R. (1969). Proving Properties of Programs by Structural Induction. *Computer Journal* **12**.

Cardelli, L. (1984). A Semantics of Multiple Inheritance. *Proc. of the Symposium on Semantics of Data Types,* Springer LNCS **173**, 51–67.

Colmerauer, A. (1982). Prolog and Infinite Trees. In Clark, K.L., Tärnlund, S.-A. (Eds.), *Logic Programming,* Academic Press, 231–251.

Colmerauer, A. (1984). Equations and Inequations on Finite and Infinite Trees. *Proc. of the 2nd International Conference on Fifth Generation Computer Systems,* 85–99.

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

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

Futatsugi, K., Goguen, J.A., Jouannaud, J.-P., Meseguer J. (1985). Principles of OBJ2. *Proc. of the 13th ACM Symposium on Principles of Programming Languages,* 52–66.

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

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

Goguen, J.A., Meseguer, J. (1986). Eqlog: Equality, Types, and Generic Modules for Logic Programming. In DeGroot, D., Lindstrom, G. (Eds.), *Logic Programming, Functions, Relations, and Equations,* Prentice Hall, 295–363.

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

Goguen, J.A., Meseguer, J. (1987b). *Order-Sorted Algebra I: Partial and Overloaded Operators, Errors and Inheritance.* Draft, Computer Science Lab, SRI International, Menlo Park, California.

Herbrand, J. (1930). *Recherches sur la Théorie de la Démonstration.* Thesis, University of Paris. Reprinted in Goldfarb, W. (Ed.), *Logical Writings,* Cambridge 1971.

Hoare, C.A.R. (1975). Recursive Data Structures. *International Journal of Computer and Information Sciences.*

Johnson, M.E. (1987). *Attribute-Value Logic and the Theory of Grammar.* PhD Dissertation, Stanford University. (To appear as CSLI Lecture Notes.)

Kasper, R.T. (1988). Conditional Descriptions in Functional Unification Grammar. *Proc. of the 26th Annual Meeting of the Association for Computational Linguistics,* Buffalo, New York, 233–240.

Kirchner, C. (1986). Computing Unification Algorithms. *Proc. of the 1st IEEE Symposium on Logic in Computer Science,* 206–216.

Kirchner, C., Lescanne, P. (1987). Solving Disequations. *Proc. of the 2nd IEEE Symposium on Logic in Computer Science,* 347–352.

Landin, P.J. (1964). The Mechanical Evaluation of Expressions. *Computer Journal,* 308–320.

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

Meseguer, J., Goguen, J.A., Smolka G. (1988). Order-Sorted Unification. (This issue.)

Milner, R., (1984). A Proposal for Standard ML. *Proc. of the ACM Symposium on Lisp and Functional Programming,* Austin, Texas, 184–197.

Moshier, M.D., Rounds, W.C. (1987). A Logic for Partially Specified Data Structures. *Proc. of the 14th ACM Symposium on Principles of Programming Languages,* Munich, West Germany, 156–167.

Mukai, K. (1987). *Anadic Tuples in Prolog.* Technical Report TR-239, ICOT, Tokyo, Japan.

Nivat. M., Reynolds, J.C. (Eds.) (1985). *Algebraic Methods in Semantics.* Cambridge University Press.

Pereira, F.C.N. (1987). Grammars and Logics of Partial Information. *Proc. of the 4th International Conference on Logic Programming,* The MIT Press, 989–1013.

Pollard, C., Sag, I.A. (1987). An Information-Based Syntax and Semantics, Volume 1, Fundamentals. *CSLI Lecture Notes* **13**, Stanford University.

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

Rounds W.C., Kasper, R.T. (1986). A Complete Logical Calculus for Record Structures Representing Linguistic Information. *Proc. of the 1st IEEE Symposium on Logic in Computer Science,* 38–43.

Shieber, S.M. (1986). An Introduction to Unification-Based Approaches to Grammar. Stanford University, *CSLI Lecture Notes* **4**.

Siekmann, J.H. (1984) Universal Unification. *Proc. 7th International Conference on Automated Deduction,* Springer LNCS **170**, 1–42.

Siekmann, J.H. (1988) Unification Theory. (This issue.)

Smolka, G., Nutt, W., Goguen, J.A., Meseguer, J. (1987). *Order-Sorted Equational Computation.* SEKI Report SR-87-14, FB Informatik, Universität Kaiserslautern, West Germany. To appear in Aït-Kaci, H., Nivat, M., *Resolution of Equations in Algebraic Structures,* Academic Press.

Smolka, G. (1988). *A Feature Logic with Subsorts.* LILOG Report 33, IBM Deutschland, Stuttgart, West Germany. To appear in Proc. of the Workshop on Unification Formalisms—Syntax, Semantics and Implementation, Titisee, West Germany.

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

Walther, C. (1988). Many-Sorted Unification. *Journal of the ACM* **35**(1), 1–17.