ORDER-SORTED FEATURE THEORY UNIFICATION

HASSAN AÏT-KACI, ANDREAS PODELSKI AND SETH COPEN GOLDSTEIN

▷ Order-sorted feature (OSF) terms provide an adequate representation for objects as flexible records. They are sorted, attributed, possibly nested, structures, ordered thanks to a subsort ordering. Sorts definitions offer the functionality of classes imposing structural constraints on objects. These constraints involve variable sorting and equations among feature paths, including self-reference. Formally, sort definitions may be seen as axioms forming an OSF theory. OSF theory unification is the process of normalizing an OSF term taking into account sort definitions, enforcing structural constraints imposed by an OSF theory. It allows objects to inherit, and thus abide by, constraints from their classes. We propose a formal system that logically models record objects with (possibly recursive) class definitions accommodating multiple inheritance. We show that OSF theory unification is undecidable in general. However, we give a set of confluent normalization rules which is complete for detecting inconsistency of an object with respect to an OSF theory. Furthermore, a subset consisting of all rules but one is confluent and terminating. This yields a practical complete normalization strategy, as well as an effective compilation scheme. ◁

This article is a revision of [8]. A short form was also published as [9]. This research was done while the two first authors were employed at Digital Equipment Corporation's Paris Research Laboratory (PRL), in Rueil-Malmaison, France, and the third author was a summer intern in PRL's Paradise project. The work was partly supported by ESPRIT Basic Research Action ACCLAIM Project No. 7195. We thank Gert Smolka and Martin Emele for their comments. We are grateful to Martin Vorbeck for questioning the need for the complex notion of "path-compatibility" used in an earlier version: his insistence prompted us to find a simpler reformulation eliminating this notion altogether. We also thank the anonymous referees for their constructive comments. Finally, we express our special gratitude to Leon Sterling, in his capacity of editor for the JLP, for his prompt editorial work and his commendable patience waiting for our more than overdue revised manuscript.

Address correspondence to Hassan Aït-Kaci: Intelligent Software Group, Simon Fraser University, School of Computing Science, Burnaby, British Columbia, V5A 1S6, Canada (`hak@cs.sfu.ca`); Andreas Podelski: Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany (`podelski@mpi-sb.mpg.de`); Seth Goldstein: University of California at Berkeley, Computer Science Division, EECS, Evans Hall, Berkeley, CA 94720, USA (`sethg@cs.berkeley.edu`).

1. INTRODUCTION

This article presents a method for using defined symbols that appear inside complex data structures. The definitions associate to symbols data structures in which such symbols appear, and thus recursive definitions are possible. A symbol definition unfolding scheme for the effective use of this facility is proposed and studied in detail. Before we develop the technical details of our approach and method, it is important that we give the reader an informal motivation, assuming little background. In this introduction, we will also relate our work to others, and outline the organization of the remainder of the paper.

1.1. Motivation of Problem

The notion of *ψ-term* was introduced in [2] as an extension of the conventional algebraic terms. In [5], ψ-terms were proposed as flexible record structures for logic programming. However, ψ-terms are of wider interest. Since they are a generalization of first-order terms, and since the latter are the pervasive data structures used by symbolic programming languages, whether based on predicate or equational logic, or pattern-directed λ-calculus, the more flexible ψ-terms offer an interesting alternative.[1]

The easiest way to describe a ψ-term is with an example. Here is a ψ-term that may be used to denote a generic person object:

$$\begin{aligned}
P : person(&name \Rightarrow id(first \Rightarrow string,\\
&\qquad\qquad\quad last \Rightarrow S : string),\\
&age \Rightarrow 30,\\
&spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S),\\
&\qquad\qquad\qquad\quad spouse \Rightarrow P)).
\end{aligned}$$

In words: a 30 year-old person who has a name in which the first and last parts are strings, and whose spouse is a person sharing his or her last name, that latter person's spouse being the first person in question.

This expression looks like a record structure. Like a typical record, it has field names; *i.e.*, the symbols on the left of $\Rightarrow$. We call these *feature* symbols. In contrast with conventional records, however, ψ-terms can carry more information. Namely, the fields are attached to *sort* symbols (*e.g.*, *person*, *id*, *string*, 30, *etc.*). These sorts may indifferently denote individual values (*e.g.*, 30) or sets of values (*e.g.*, *person*, *string*). In fact, values are assimilated to singleton-denoting sorts. Sorts are partially ordered so as to reflect set inclusion; *e.g.*, $employee < person$ means that all employees are persons. Finally, sharing of structure can be expressed with *variables* (*e.g.*, P and S). This sharing may be circular (*e.g.*, P).

Clearly, a first-order term can be viewed as a particular ψ-term. Namely, considering only singleton sorts, a sort ordering reduced to syntactic equality, and numbers as features, a term $f(t_1, \ldots, t_n)$ is the ψ-term $f(1 \Rightarrow t_1, \ldots, n \Rightarrow t_n)$. In fact, ψ-terms enjoy the same powerful operations as first-order terms: *matching* (as, say, in term-rewriting systems, or ML function definitions) and *unification* (as, say, in Prolog, or equational narrowing). This makes them a more flexible data

[1]Accommodating functional programming with the convenience offered by ψ-terms is investigated in [7, 17].

structure for symbolic programming since both operations take into account the partial-order on sorts and extensibility with features. Therefore, they can supplement first-order terms in a functional programming language or logic programming language [5, 6]. In this manner, a form of single inheritance (matching) and multiple inheritance (unification) is obtained cleanly and efficiently. Pattern-directed definition of functions or predicates will indeed be inherited along the partial order of sorts (the *sort hierarchy*) thanks to matching or unification.

In object-oriented programming, typically, objects do not enjoy the expressivity offered by ψ-terms. On the other hand, they are made according to blueprints specified as *class* definitions. A class acts as a template, restricting the aspect of the objects that are its instances. Our intention is to conceive such a convenience for ψ-terms and, in so doing, expand the capability of the constraining effect of classes on objects. We propose to achieve this using *sort definitions.* A sort definition associates a ψ-term structure to a sort. Intuitively, one may then see a sort as an *abbreviation* of a more complex structure. Hence, a sort definition specifies a template that an object of this sort must abide by, whenever it uses any part of the structure appearing in the ψ-term defining the sort.

For example, consider the ψ-term:[2]

$$\begin{array}{l} person(name \Rightarrow \top(last \Rightarrow string), \\ \qquad spouse \Rightarrow \top(spouse \Rightarrow \top, \\ \qquad\qquad\qquad name \Rightarrow \top(last \Rightarrow \text{“}smith\text{”}))). \end{array}$$

Without sort definitions, there is no reason to expect that this structure should be incomplete, or inconsistent, as intended. Let us now define the sort *person* as an abbreviation of the structure:

$$\begin{array}{l} P : person(name \Rightarrow id(first \Rightarrow string, \\ \qquad\qquad\qquad\qquad last \Rightarrow S : string), \\ \qquad spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S), \\ \qquad\qquad\qquad\qquad spouse \Rightarrow P)). \end{array}$$

This definition of the sort *person* expresses the expectation whereby, whenever a *person* object has features *name* and *spouse*, these should lead to objects of sort *id* and *person*, respectively. Moreover, if the features *first* and *last* are present in the object indicated by *name*, then they should be of sort *string*. Also, if a *person* object had sufficient structure as to involve feature paths *name.last* and *spouse.name.last*, then these two paths should lead to the same object. And so on.

For example, with this sort definition, the *person* object with last name *“smith”* above should be made to comply with the definition template by being *normalized* into the term:[3]

$$\begin{array}{l} X : person(name \Rightarrow id(last \Rightarrow N : \text{“}smith\text{”}), \\ \qquad spouse \Rightarrow person(spouse \Rightarrow X, \\ \qquad\qquad\qquad name \Rightarrow id(last \Rightarrow N))). \end{array}$$

Note that in our approach, we do not wish to enforce the explicit presence of the complete generic structure of a sort's definition in every object of that sort.

[2] The sort symbol $\top$ is the top of the partial order, the sort of all objects.

[3] In this example, it is assumed, of course, that *“smith”*$<$*string*.

Rather, we want to enforce the minimal restrictions that will guarantee that every object of a given sort denotes the largest possible set consistent with the sort's definition. Therefore the presence of a given feature in a sort definition does not necessarily imply its presence in every instance of the sort, and conversely, the absence of a feature in a sort definition does not rule out its presence in an instance of the sort. For example, with the above definition for the sort *person*, a *person* instance need not necessarily have a *name*, and conversely, we could use $person(hobby \Rightarrow movie\_going)$ without worrying about violating the template for *person* since the feature *hobby* is not constrained by the definition of *person*.

This lazy scheme of inheritance of structural constraints from the class template into an object's structure is invaluable for efficiency reasons. Indeed, if all the (possibly voluminous) template structure of a sort were to be systematically expanded into an object of this sort that uses only a tiny portion of it, space and time would be wasted. More importantly, lazy inheritance is a way to ensure termination of consistency checking. For example, consider the definition for a sort s to be $s(f \Rightarrow s)$. It is recursive, as it involves the sort s in its body. A non-lazy scheme would expand the OSF term s into $s(f \Rightarrow s)$, $s(f \Rightarrow s(f \Rightarrow s))$, $s(f \Rightarrow s(f \Rightarrow s(f \Rightarrow s)))$, $\ldots$, and so on *ad infinitum*. Clearly, expanding all occurrences of sorts into their definition templates whether or not they are needed would go on for ever.

An incidental benefit of sort-unfolding in the context of a sort semilattice is what we call *proof memoing*. Namely, once the definition of a sort for a variable X has been unfolded, and the attached constraints proven for X, this proof is automatically and efficiently recorded by the expanded sort. The accumulation of proofs corresponds exactly to the greatest lower bound operation. Besides the evident advantage of not having to repeat computations, this memoing phenomenon accommodates expressions which otherwise would loop. Let us take a small example to illustrate this point. Lists can be specified by declaring *nil* and *cons* to be subsorts of the sort *list* and by defining for the sort *cons* the template ψ-term $cons(head \Rightarrow \top, tail \Rightarrow list)$. Now, consider the expression $X : [1|X]$, the circular list containing the one element 1—*i.e.*, desugared as $X : cons(head \Rightarrow 1, tail \Rightarrow X)$. Verifying that X is a list, since it is the *tail* of a *cons*, terminates immediately on the grounds that X has already been memoized to be a *cons*, and $cons < list$. In contrast, the semantically equivalent Prolog program with two clauses: $list([])$ and $list([H|T])$:- $list(T)$ would make the goal $list(X : [1|X])$ loop.

1.2. Overview of Our Approach

In this paper we present a formal and practical solution for the problem of checking the consistency of a ψ-term object modulo a sort hierarchy of structural class templates. We call this problem OSF theory unification. We formalize the problem in first-order logic: objects as OSF constraint formulae, classes as axioms defining an OSF theory, class inheritance as testing the satisfiability of an OSF constraint in a model of the OSF theory.

We give conditions for the existence of non-trivial models for OSF theories, and prove the undecidability of the OSF theory unification problem. We also show that *failure* of OSF theory unification (*i.e.*, non-satisfiability of an OSF term modulo an OSF theory) is semi-decidable. We propose a system of ten normalization rules that is complete for detecting incompatibility of an object with respect to an OSF theory; *i.e.*, checking non-satisfiability of a constraint in a model of the axioms.

This system specifies the third Turing-complete calculus used in LIFE [4], besides the logical and the functional one.

As a calculus, the ten-rule system enjoys an interesting property of consisting of two complementary rule subsets: a system of nine confluent and terminating *weak* rules, and one additional *strong* rule, whose addition to the other rules preserves confluence, but loses termination. There are two great consequences of this property: (1) it yields a complete normalization strategy consisting of repeatedly normalizing a term first with the terminating rules, and then apply, if at all necessary, the tenth rule; and (2) it provides a compilation scheme for an OSF theory since all sort definitions of the theory can be normalized with respect to the theory itself using the weak rules.

1.3. Relation to Other Work

Our system is unique in that it comes with a semantic foundation and constitutes the first proven correct and complete, practical algorithm for the problem of unfolding sort definitions in order-sorted feature structures.

The problem was first already addressed in [3]. A significant difference is that the method was restricted to single inheritance and was non-lazy. Operationally, it amounted to a breadth-first expansion of all sorts and was not very practical.

Concerning undecidability of OSF theory unification, a related, but different result was proven by Gert Smolka in [20]. The undecidability of our problem uses explicitly the existence of a model satisfying the sort definitions while this is overlooked in [20] (*cf.*, also, Footnote 6).

As for unfolding sort definitions, we know of two other works, both relevant to computational linguistics: that of Bob Carpenter and that of Martin Emele and Rémi Zajac. Bob Carpenter [12] proposed a simple type-checking of a system of sort definitions for feature terms that are essentially a variation of ψ-terms. However, besides being purely operational, this system is limited to the simple case where sort definitions specify sort constraints on features alone, without feature compositions and, more importantly, without shared variables imposing coreference constraints on feature paths. On the other hand, his formalism handles partial features, while what we present works with total features. As it turns out, our system can be made to handle partial features with the addition of one simple decidable rule whose effect is to narrow the sort of a variable to intersect a feature's domain when that feature is applied to it. Therefore, the system described in [12] is a special case of what we present here. In the recent book [13], Chapter 15 deals with "recursive type constraint systems" extending that of [3] to be of the kind we study here. He gives a complete resolution method similar to Horn clause resolution. That method differs from ours in that it is not lazy.

The work of Emele and Zajac on typed unification grammars [16] is actually quite close to what we report here. Their work is an elaboration of [3], with the assumption that features are partial. Their main contribution has been the study of clever algorithms to carry out type unfolding efficiently. In [15], Martin Emele describes an implementation that shares many insights with the method that we describe here. In particular, he uses structure-sharing to avoid much copying overhead, and whenever copying must be done, it is done such that no redundant copying is performed. However, his technique differs from ours, in that when copying is done, all the defined features of a sort are brought into the formula where it appears. Most importantly, Emele's algorithm is not explained in formal terms, let alone proven correct. No semantics is provided, and no clear delineation is made, as our rules do, between a maximal decidable subset of cases and the complete normalization.

The functional programming community has been using variations on, and generalizations of, an extensible record formalism pioneered by Luca Cardelli [11] and used to endow polymorphically typed languages of the ML family with a form of multiple inheritance [21, 19]. Records are viewed as partial functions from field label symbols to values. Record types are defined similarly as partial functions from labels to types. What corresponds to unification in our formalism is rendered there as so-called *record concatenation.* In contrast to our (possibly circular) use of logical variables and unification, coreference constraints are not supported, and self-reference is handled using a special fix-point functional abstraction. Subtyping in the Cardelli style of records is checked using static inference rules that are essentially performing the kind of verification done by Carpenter's system [12], but made more complicated by the presence of polymorphic function types. It is hence very hard to compare that trend of work and ours because of these differences in the nature, restriction, and use of records.

1.4. Organization of Paper

Section 2 recalls the necessary OSF formalism and terminology, already introduced in [6], that are needed to make this article self-contained. Section 3 presents our formalization of OSF theories and exposes essential facts about them. Section 4, the crux of the paper, presents the OSF normalization system and its formal properties. We have adjoined an appendix giving a detailed example of OSF theory normalization.

2. OSF FORMALISM

2.1. OSF Algebras

An *OSF Signature* is given by $\langle \mathcal{S}, \leq, \wedge, \mathcal{F} \rangle$ such that:

- $\mathcal{S}$ is a set of *sorts* containing the sorts $\top$ and $\perp$;
- $\leq$ is a decidable partial order on $\mathcal{S}$ such that $\perp$ is the least and $\top$ is the greatest element;
- $\langle \mathcal{S}, \leq, \wedge \rangle$ is a lower semi-lattice ($s \wedge s'$ is called the greatest common subsort of s and s');
- $\mathcal{F}$ is a set of *feature symbols.*

Given an OSF signature $\langle \mathcal{S}, \leq, \wedge, \mathcal{F} \rangle$, an *OSF algebra* is a structure

$\mathcal{A} = \langle D^{\mathcal{A}}, (s^{\mathcal{A}})_{s \in \mathcal{S}}, (f^{\mathcal{A}})_{f \in \mathcal{F}} \rangle$

such that:

- $D^{\mathcal{A}}$ is a non-empty set, called the *domain* of $\mathcal{A}$;
- for each sort symbol s in $\mathcal{S}$, $s^{\mathcal{A}}$ is a subset of the domain; in particular, $\top^{\mathcal{A}} = D^{\mathcal{A}}$ and $\perp^{\mathcal{A}} = \emptyset$;
- $(s \wedge s')^{\mathcal{A}} = s^{\mathcal{A}} \cap s'^{\mathcal{A}}$ for two sorts s and s' in $\mathcal{S}$;
- for each feature f in $\mathcal{F}$, $f^{\mathcal{A}}$ is a total unary function from the domain into the domain; *i.e.*, $f^{\mathcal{A}} : D^{\mathcal{A}} \mapsto D^{\mathcal{A}}$.

An *OSF homomorphism* $\gamma : \mathcal{A} \mapsto \mathcal{B}$ between two OSF algebras $\mathcal{A}$ and $\mathcal{B}$ is a function $\gamma : D^{\mathcal{A}} \mapsto D^{\mathcal{B}}$ such that:

- $\gamma(f^{\mathcal{A}}(d)) = f^{\mathcal{B}}(\gamma(d))$ for all $d \in D^{\mathcal{A}}$;
- $\gamma(s^{\mathcal{A}}) \subseteq s^{\mathcal{B}}$.

2.2. OSF Terms

Let $\mathcal{V}$ be a countably infinite set of variables. An *OSF term* t is an expression of the form:

$$X : s(f_1 \Rightarrow t_1, \ldots, f_n \Rightarrow t_n)$$

where X is a variable in $\mathcal{V}$, s is a sort in $\mathcal{S}$, $f_1, \ldots, f_n$ are features in $\mathcal{F}$, $n \geq 0$, $t_1, \ldots, t_n$ are OSF terms.

Here is an example of an OSF term (call it t_{person}):

$$\begin{aligned} X : person(&name \Rightarrow N : \top(first \Rightarrow F : string),\\ &name \Rightarrow M : id(last \Rightarrow S : string),\\ &spouse \Rightarrow P : person(name \Rightarrow I : id(last \Rightarrow S : \top),\\ &\qquad\qquad\qquad\qquad spouse \Rightarrow X : \top)).\end{aligned}$$

We shall use a lighter notation, omitting variables that are not shared, and the sort of a variable when it is $\top$:

$$\begin{aligned} X : person(&name \Rightarrow \top(first \Rightarrow string),\\ &name \Rightarrow id(last \Rightarrow S : string),\\ &spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S),\\ &\qquad\qquad\qquad spouse \Rightarrow X)).\end{aligned}$$

Given a term $t = X : s(f_1 \Rightarrow t_1, \ldots, f_n \Rightarrow t_n)$, the variable X is called its *root* variable and sometimes referred to as $Root(t)$. The set of all variables occurring in t is defined as $Var(t) = \{Root(t)\} \cup \bigcup_{i=1}^{n} Var(t_i)$.

Given a term t as above, an OSF interpretation $\mathcal{A}$, and an $\mathcal{A}$-valuation $\alpha : \mathcal{V} \mapsto \mathcal{D}^{\mathcal{A}}$, the *denotation* of t is given by:

$$[\![t]\!]^{\mathcal{A},\alpha} = \{\alpha(X)\} \cap s^{\mathcal{A}} \cap \bigcap_{1 \leq i \leq n} (f_i^{\mathcal{A}})^{-1}([\![t_i]\!]^{\mathcal{A},\alpha}).$$

Thus, for all possible valuations of the variables, $[\![t]\!]^{\mathcal{A}} = \bigcup_{\alpha:\mathcal{V}\mapsto\mathcal{D}^{\mathcal{A}}} [\![t]\!]^{\mathcal{A},\alpha}$.

An OSF term $\psi = X : s(f_1 \Rightarrow \psi_1, \ldots, f_n \Rightarrow \psi_n)$ is in *normal form* (and then called a *ψ-term*) if:

- there is at most one occurrence of a variable Y in ψ such that Y is the root variable of a non-trivial OSF term (*i.e.*, different than $Y : \top$);
- s is a non-bottom sort in $\mathcal{S}$;
- $f_1, \ldots, f_n$ are pairwise distinct features in $\mathcal{F}$, $n \geq 0$,
- $\psi_1, \ldots, \psi_n$ are ψ-terms.

We call Ψ the set of all ψ-terms.

For example, the OSF term,

$$\begin{aligned} X : person(&name \Rightarrow id(first \Rightarrow string,\\ &\qquad\qquad\qquad last \Rightarrow S : string),\\ &spouse \Rightarrow person(name \Rightarrow id(last \Rightarrow S),\\ &\qquad\qquad\qquad spouse \Rightarrow X))\end{aligned}$$

is a normal OSF term and denotes the same set as t_{person}.

Definition 2.1. Let ψ and ψ' be two OSF terms. Then, $\psi \leq \psi'$ ("ψ is *subsumed* by ψ'") if and only if, for all OSF algebras $\mathcal{A}$, $[\![\psi]\!]^{\mathcal{A}} \subseteq [\![\psi']\!]^{\mathcal{A}}$.

Given a ψ-term ψ, the sort of a variable $V \in Var(\psi)$ will sometimes be referred to as $Sort_{\psi}(V)$. The subscript ψ will often be omitted when the context is clear.

Here are a few facts about OSF terms.

- *OSF terms generalize first-order terms.* First-order terms form a special OSF algebra where the sorts form a flat lattice and the features are (natural number) positions. Thus, the first-order term $f(t_1, \ldots, t_n)$, is just the ψ-term: $f(1 \Rightarrow t_1, \ldots, n \Rightarrow t_n)$.
- *All variables occurring in an OSF term are implicitly existentially quantified at the term's outset* (assuming no further outer context). As a corollary, sorts are particular (basic) OSF terms: indeed, $[\![X : s]\!]^{\mathcal{A}} = s^{\mathcal{A}}$ since $\bigcup_{\alpha:\mathcal{V}\mapsto\mathcal{D}^{\mathcal{A}}}(\{\alpha(X)\} \cap s^{\mathcal{A}}) = s^{\mathcal{A}}$.
- *An OSF term ψ is the empty set in all interpretations if ψ has an occurrence of a variable sorted by the empty sort $\bot$.*
- *Dually, $[\![\psi]\!]^{\mathcal{A}} = D^{\mathcal{A}}$ in all interpretations $\mathcal{A}$ if all its variables occur only once in ψ and are sorted by $\top$.*
- *Features are total functions.* If $\psi = X : s(f_1 \Rightarrow \psi_1, \ldots, f_n \Rightarrow \psi_n)$, and $Z \notin Var(\psi)$, then $[\![\psi]\!]^{\mathcal{A}} = [\![X : s(f_1 \Rightarrow \psi_1, \ldots, f_n \Rightarrow \psi_n, f \Rightarrow Z : \top)]\!]^{\mathcal{A}}$ for any feature symbol $f \in \mathcal{F}$ and any OSF interpretation $\mathcal{A}$.
- *Variables denote essentially an equality among attribute compositions.* For example, $[\![X : \top(f_1 \Rightarrow Y : \top, f_2 \Rightarrow Y : \top)]\!]^{\mathcal{A}} = \{d \in D^{\mathcal{A}} \mid f_1^{\mathcal{A}}(d) = f_2^{\mathcal{A}}(d)\}$. This justifies our referring to variables as *coreference tags.*

2.3. OSF Clauses

A logical reading of an OSF term is immediate as its information content can be characterized by a simple formula. For this purpose, we need a simple clausal language as follows.

An *OSF constraint* is one of (1) $X : s$, (2) $X \doteq X'$, or (3) $X.f \doteq X'$, where X and X' are variables in $\mathcal{V}$, s is a sort in $\mathcal{S}$, and f is a feature in $\mathcal{F}$. An *OSF clause* is a (possibly infinite) set of OSF constraints (to be interpreted as their conjunction).

Given $\mathcal{A}$ an OSF algebra and $\alpha : \mathcal{V} \mapsto \mathcal{D}^{\mathcal{A}}$ an $\mathcal{A}$-valuation, we write $\mathcal{A}, \alpha \models \phi$ to say that an OSF clause ϕ is *satisfied* in $\mathcal{A}$ by α. Formally, $\mathcal{A}, \alpha \models \phi$ iff $\mathcal{A}, \alpha \models \phi'$ for every OSF constraint ϕ' in ϕ, where:

- $\mathcal{A}, \alpha \models X : s$ if and only if $\alpha(X) \in s^{\mathcal{A}}$;
- $\mathcal{A}, \alpha \models X \doteq Y$ if and only if $\alpha(X) = \alpha(Y)$;
- $\mathcal{A}, \alpha \models X.f \doteq Y$ if and only if $f^{\mathcal{A}}(\alpha(X)) = \alpha(Y)$.

We say that ϕ is *satisfiable* in $\mathcal{A}$ if there exists an $\mathcal{A}$-valuation α that satisfies it.

2.4. From OSF Terms to OSF Clauses

We can always associate with an OSF term $\psi = X : s(f_1 \Rightarrow \psi_1, \ldots, f_n \Rightarrow \psi_n)$ a corresponding OSF clause $\phi(\psi)$ as follows:

$$\begin{array}{llllll} X : \mathit{person} & \& \; X.\mathit{name} \doteq N & \& \; N : \top & \& \; N.\mathit{first} \doteq F & \& \; F : \mathit{string} \\ & \& \; X.\mathit{name} \doteq M & \& \; M : \mathit{id} & \& \; M.\mathit{last} \doteq S & \& \; S : \mathit{string} \\ & \& \; X.\mathit{spouse} \doteq P & \& \; P : \mathit{person} & \& \; P.\mathit{name} \doteq I & \& \; I : \mathit{id} \\ & & & \& \; I.\mathit{last} \doteq S & \& \; S : \top \\ & & & \& \; P.\mathit{spouse} \doteq X & \& \; X : \top. \end{array}$$

FIGURE 2.1. OSF clause form of OSF term t_{person}

$$\begin{aligned} \phi(\psi) = X : s \;\&\; X.f_1 \doteq X_1' \;\&\; \ldots \;\&\; X.f_n \doteq X_n' \\ \&\; \phi(\psi_1) \qquad \&\; \ldots \;\&\; \phi(\psi_n) \end{aligned}$$

where $X_1', \ldots, X_n'$ are the roots of $\psi_1, \ldots, \psi_n$, respectively. We say that $\phi(\psi)$ is obtained from *dissolving* the OSF term ψ. For example, the non-normal OSF term t_{person} of Section 2.2 is dissolved into the OSF clause shown in Figure 2.1.

We will use a shorthand notation to express that a variable X is *constrained* by an OSF term t. Namely, we denote by $C_t[X]$ the formula $X \doteq \mathit{Root}(t) \;\&\; \phi(t)$ and by $C_t^{\exists}[X]$ the formula $\exists\, \mathit{Var}(t)\; C_t[X]$.[4]Note that $C_t^{\exists}[X]$ is not quite the existential closure of $C_t[X]$ because it can (and will) be that $X \notin \mathit{Var}(t)$.

It can be shown that the set-theoretic denotation of an OSF term and the logical semantics of its dissolved form coincide exactly [6]:

$$[\![\psi]\!]^{\mathcal{A}} = \{\alpha(X) \mid \alpha \in \mathit{Val}(\mathcal{A}),\; \mathcal{A}, \alpha \models C_\psi^{\exists}[X]\}.$$

For this reason, and to lighten notation, we shall confuse an OSF term for its dissolved form, writing ψ when we actually mean $\phi(\psi)$.

2.5. OSF Unification

Definition 2.2. An OSF clause ϕ is called <u>*solved*</u> if for every variable X, ϕ contains:

- at most one sort constraint of the form $X : s$, with $\bot < s$; and,
- at most one feature constraint of the form $X.f \doteq X'$ for each f;
- if $X \doteq X' \in \phi$, then X does not appear anywhere else in ϕ.

Given an OSF clause ϕ, non-deterministically applying any applicable rule among the four shown in Figure 2.2 until none apply will always terminate in a solved OSF clause. A rule transforms the numerator into the denominator. The expression $\phi[X/X']$ stands for the formula obtained from ϕ after replacing all occurrences of X' by X. We also refer to any clause of the form $X : \bot$ as the *inconsistent clause.* The following is immediate [6].

Theorem 2.1. The rules of Figure 2.2 are solution-preserving, finite terminating, and confluent (modulo variable renaming). Furthermore, they always result in a normal form that is either the inconsistent clause or an OSF clause in solved form.

Proof. Solution preservation is immediate as each rule transforms an OSF-clause

[4]We use a lax notation here for convenience; given a set of variables $\mathcal{X} = \{X_1, \ldots, X_n\}$, the formula $\exists \mathcal{X} \phi$ stands for: $\exists X_1 \ldots \exists X_n \phi$.

Sort Intersection:

$$(1)\quad \frac{\phi \;\&\; X : s \;\&\; X : s'}{\phi \;\&\; X : s \wedge s'}$$

Inconsistent Sort:

$$(2)\quad \frac{\phi \;\&\; X : \bot}{X : \bot}$$

Variable Elimination:

$$(3)\quad \frac{\phi \;\&\; X \doteq X'}{\phi[X'/X] \;\&\; X \doteq X'} \qquad \begin{array}{l}\text{if } X \neq X' \\ \quad \text{and } X \in \mathit{Var}(\phi)\end{array}$$

Feature Decomposition:

$$(4)\quad \frac{\phi \;\&\; X.f \doteq X' \;\&\; X.f \doteq X''}{\phi \;\&\; X.f \doteq X' \;\&\; X' \doteq X''}$$

FIGURE 2.2. OSF Clause Normalization Rules

$$\begin{array}{llllllll}
X : \mathit{person} & \&\; X.\mathit{name} \doteq N & \&\; N : \mathit{id} & \&\; N.\mathit{first} & \doteq F & \&\; F : \mathit{string} \\
 & & & \&\; N.\mathit{last} & \doteq S & \&\; S : \mathit{string} \\
 & \&\; X.\mathit{spouse} \doteq P & \&\; P : \mathit{person} & \&\; P.\mathit{name} & \doteq I & \&\; I : \mathit{id} \\
 & & & \&\; I.\mathit{last} & \doteq S & \\
 & & & \&\; P.\mathit{spouse} & \doteq X. &
\end{array}$$

FIGURE 2.3. Normal form of OSF clause of Figure 2.1

into a semantically equivalent one.

Termination follows from the fact that each of the three first rules strictly decreases the number of non-equality atoms. The last rule eliminates a variable possibly making new redexes appear. But, the number of variables in a formula being finite, new redexes cannot be formed indefinitely.

Confluence is clear as consistent normal forms are syntactically identical modulo the least equivalence on V generated by the set of variable equalities. □

For example, the normalization of the OSF clause in the last example leads to the solved OSF clause which is the conjunction of the equality constraint $M \doteq N$ and the OSF clause shown in Figure 2.3. The rules of Figure 2.2 are all we need to perform the unification of two OSF terms. Namely, two terms t_1 and t_2 are OSF unifiable if and only if the normal form of $\mathit{Root}(t_1) \doteq \mathit{Root}(t_2) \;\&\; t_1 \;\&\; t_2$ is not $\bot$.

An OSF clause ϕ in solved form is always satisfiable in the term algebra Ψ, so the rules on Figure 2.2 are a decision procedure for the satisfiability of OSF clauses.

3. OSF THEORIES

3.1. Sort Definitions

As explained in Section 1.1, we may view a class template as a ψ-term. Hence, to define a sort s as a class is to associate to this sort a ψ-term whose root sort is s. Informally, an OSF theory is a set of sort definitions, each of which is a ψ-term whose root sort is the name of the class defined by that sort.

Formally, an *OSF theory* is a function $\Theta : \mathcal{S} \mapsto \Psi$ such that $Sort(Root(\Theta(s))) = s$ for all $s \in \mathcal{S}$ and $\Theta(\top) = \top$, $\Theta(\bot) = \bot$. The OSF theory $\Theta = 1\!\!1_{\mathcal{S}}$ which is the identity on $\mathcal{S}$ is called the *empty OSF theory*.

An OSF theory Θ is *order-consistent* if it is monotonic; *i.e.*, if:

$$\forall s, s' \in \mathcal{S},\ s \leq s' \Rightarrow \Theta(s) \leq \Theta(s').$$

Recall that $\leq$ is defined on ψ-terms (see Definition 1 on Page 106) extending the ordering on sorts.

We shall always assume the OSF theory Θ to be order-consistent. By setting $\Theta(s) = \bigwedge_{s \leq s'} \Theta(s')$ if different from $\bot$, it is easily possible to normalize a non order-consistent theory into an equivalent order-consistent one, if it exists.

Clearly, an OSF algebra is a logical first-order structure $\mathcal{A}$ interpreting sort symbols as unary predicates, *i.e.*, sets, and feature symbols as unary functions, and satisfying the axioms specified by the sort hierarchy. Namely, for all sorts s, s', s'' such that $s \wedge s' = s''$, the following axiom is valid in $\mathcal{A}$:

$$Axiom_{[s \wedge s' = s'']} :\ \forall X\ \ (X : s\ \&\ X : s' \ \rightarrow\ X : s'').$$

The name OSF theory is justified by the fact that the function Θ specifies a system of axioms; *i.e.*, for each $s \in \mathcal{S}$, the axiom:

$$Axiom_{[\Theta(s)]} :\ \forall X\ \ \big(X : s\ \leftrightarrow\ C^{\exists}_{\Theta(s)}(X)\big)$$

expressing that an element in the sort s necessarily satisfies the constraints attached to s (the constraints coming from the dissolved ψ-term assigned to s by Θ). Note that $\Theta(s)$ contains the constraint $Root(\Theta(s)) : s$. Thus, the equivalence ($\leftrightarrow$) in $Axiom_{[\Theta(s)]}$ is, in fact, an implication ($\rightarrow$).

The class of all Θ-OSF algebras is the class of all OSF algebras such that $s^{\mathcal{A}} = [\![\Theta(s)]\!]^{\mathcal{A}}$. Thus, Θ specifies a first-order theory, namely through the system of all the axioms $Axiom_{[s \wedge s' = s'']}$ and $Axiom_{[\Theta(s)]}$. The notion of Θ-satisfiability refers to satisfiability in a Θ-OSF algebra; *i.e.*, in a logical first-order structure where the axioms above hold.

We will see next that such a structure actually exists (under the overall assumption that Θ is order-consistent). We first define the OSF algebra Ψ_0 of possibly infinite OSF graphs.

An OSF graph $g = (V, E)$ consists of nodes denoted by mutually distinct variables in $\mathcal{V}$, *i.e.*, $V \subseteq \mathcal{V}$, and arcs between them, *i.e.*, $E \subseteq \mathcal{V} \times \mathcal{V}$. It has a distinguished node, its root, from which all its other nodes are reachable. All nodes and arcs of an OSF graph are labeled. Nodes are labeled with non-bottom sorts and arcs are labeled with feature symbols such that the same feature may not be attributed to two distinct arcs coming from the same node.

The set of all OSF graphs forms an OSF algebra:

- the OSF graph denotation of a sort s is the set of all graphs whose root sort is equal to or less than s;

- applying the feature f to a graph g rooted in X is the maximal subgraph of g rooted in X' if g has an arc labeled f between nodes X and X'; otherwise, it is a one-node arcless graph whose node is a new distinct variable $X_{f,g}$ labeled with $\top$.

We next define the (possibly infinite) OSF clauses ϕ^* obtained from an OSF clause ϕ by unfolding all sort definitions. Formally, $\phi^* = \bigcup_{n\geq 0} \phi^n$, where $\phi^0 = \phi$ and:

$$\phi^{n+1} \;=\; \phi^n \cup \{C_{\Theta(s)}[X] \mid X : s \in \phi^n\}.$$

We assume that the variables in the OSF constraints added to ϕ^n, $Var(\Theta(s))$ are new for each unfolded sort constraint $X : s$.

We define two formulae to be Θ-*equivalent* if they are equivalent modulo the axioms specified by Θ and the sort hierarchy and modulo existential quantification of variables in either of the formulae. Thus, ϕ and $\phi^n, (n \geq 0)$, and even ϕ^*, are all Θ-equivalent. The next lemma compares satisfiability of ϕ and ϕ^* in different structures.

Lemma 3.1. An OSF clause ϕ is Θ-satisfiable if and only if ϕ^* is satisfiable.

Proof. Every Θ-OSF algebra where ϕ is satisfiable is in particular an OSF algebra where ϕ^* is satisfiable. Vice versa, the domain of an OSF algebra where ϕ^* is satisfiable can be "trimmed down" to the domain of a Θ-OSF algebra (by including only elements which are values of the valuations which make ϕ^* hold true) such that $Axiom_{[\Theta(s)]}$ holds for every sort s which occurs in ϕ^*, and ϕ is satisfiable. Since Θ is order-consistent, the interpretation of the sorts can be chosen as the restriction of the old interpretation to the new domain. $\square$

Definition 3.1. A (possibly infinite) OSF clause ϕ is called *solved* if, for every variable X, ϕ contains:

- at most one sort constraint of the form $X : s$, with $\bot < s$; and,
- at most one feature constraint of the form $X.f \doteq X'$ for each f;
- if $X \doteq X' \in \phi$, then X does not appear in any other OSF constraint in ϕ.

Lemma 3.2. A (possibly infinite) OSF clause ϕ in solved form is satisfiable in Ψ_0, the OSF algebra of possibly infinite OSF graphs.

Proof. Let X be a variable in ϕ where X is not on the left side of the symbol $\doteq$ anywhere in ϕ. We define the valuation α on X as the graph (V, E) with the root node X, where $V = \bigcup_{n\geq 0} V_n$, $E = \bigcup_{n\geq 0} E_n$, $V_0 = \{X\}$, $E_0 = \emptyset$, $V_{n+1} = V_n \cup \{Z \mid Y.f \doteq Z \in \phi \text{ for some } Y \in V_n\}$, $E_{n+1} = E_n \cup \{(Y, Z) \mid Y.f \doteq Z \in \phi \text{ for some } Y \in V_n\}$. A node Y is labeled by s if $Y : s \in \phi$ for some $s \in \mathcal{S}$, and by $\top$ otherwise. An arc (Y, Z) is labeled by f if $Y.f \doteq Z \in \phi$.

If $X \doteq X' \in \phi$, then we set $\alpha(X) = \alpha(X')$. Clearly, every OSF constraint of ϕ holds in Ψ_0 under the valuation α. $\square$

Definition 3.2. An OSF clause ϕ is called Θ-*solved* if the OSF clause ϕ^1, obtained by unfolding all sort definitions *once*, can be normalized into a solved form which contains ϕ, and no other constraints whose variables are those from ϕ.

That is, if the solved form contains $X : s$, then either $X : s \in \phi$ or $X \notin \mathit{Var}(\phi)$. Similarly, if it contains $Y \doteq X$, then either $Y \doteq X \in \phi$ or $Y \notin \mathit{Var}(\phi)$; and if it contains $X.f \doteq Y$, then either $X.f \doteq Y \in \phi$ or $Y \notin \mathit{Var}(\phi)$.

Thus, the OSF clause ϕ is Θ-solved if the OSF clause:

$$\phi^1 = \phi \cup \bigcup_{X:s\in\phi} \{C_{\Theta(s)}[X]\}$$

can be transformed, by repeated applications of the rules in Figure 2.2, into an OSF constraint ϕ' of the form $\phi' = \phi \cup \phi_1 \cup \phi_2$ where ϕ_1 contains only equalities of the form $Y \doteq X$ where $X \in \mathit{Var}(\phi)$ and $Y \notin \mathit{Var}(\phi)$ and ϕ_2 is an OSF constraint in solved form whose variables are new for ϕ; *i.e.*, $\mathit{Var}(\phi) \cap \mathit{Var}(\phi_2) = \emptyset$.

The OSF theory Θ is *well-formed* if, for every $s \in \mathcal{S}$, the dissolved ψ-term $\Theta(s)$ is in Θ-solved form. From now on we are interested only in well-formed (and order-consistent) OSF theories.

We introduce next the OSF algebra Ψ_Θ. The domain of Ψ_Θ, and the interpretation of the features, are the ones of Ψ_0. If $s \in \mathcal{S}$ is a sort, then:

$$s^{\Psi_\Theta} = \{g \in D^{\Psi_0} \mid \Psi_0, \alpha \models (X : s)^*,\ \alpha(X) = g\}.$$

In the special case of the empty theory, Ψ_Θ is the OSF graph algebra Ψ_0.

As in the case of OSF unification, *i.e.*, of satisfiability of OSF clauses in OSF algebras, it is sufficient to consider Θ-satisfiability in one particular Θ-OSF algebra, here Ψ_Θ. This characterizes Ψ_Θ as a canonical Θ-OSF algebra (meaning: any Θ-satisfiable OSF clause is satisfiable in Ψ_Θ). It follows from the fact that one can easily construct a homomorphism from any Θ-algebra into Ψ_Θ (and, thus, Ψ_Θ is *weakly final* (*cf.*, [6]) in the category of all Θ-OSF algebras).

Proposition 3.1. Given a well-formed order-consistent OSF theory Θ, a Θ-solved OSF clause is satisfiable in Ψ_Θ. In particular, Ψ_Θ is a Θ-OSF algebra, i.e., *a model of the axioms specified by the sort hierarchy $\langle \mathcal{S}, \leq, \wedge \rangle$ and the OSF theory Θ.*

PROOF. Since, for each sort $s \in \mathcal{S}$, $\Theta(s)$ is Θ-solved, ϕ^n is Θ-solved, for all n. In particular, for all n ϕ^n, and hence also ϕ^*, is Θ-equivalent to an OSF clause in solved form. Thus, according to Lemma 3.2, ϕ^* is satisfiable in Ψ_0, the OSF algebra of possibly infinite OSF graphs. Say, ϕ^* holds under the valuation α. Since all sort definitions in ϕ^* are unfolded, each graph g rooted in a node labeled by a sort s lies in the Ψ_Θ-denotation of s; *i.e.*, $g \in s^{\Psi_\Theta}$ $(\ldots \subseteq s^{\Psi_0})$. Thus, α is in particular a Ψ_Θ-valuation. That is, ϕ^*, and hence $\phi \subseteq \phi'$, are satisfiable in Ψ_Θ. $\square$

4. OSF THEORY UNIFICATION

We next investigate the operational and denotational semantics of a scheme for using a class template structure in an object instance. We call this scheme *OSF Theory Unification* since it amounts to solving OSF clauses in the presence of an OSF theory. This is a generalization of OSF unification, the solving of OSF clauses in the empty theory (*cf.*, Figure 2.2).

Formally, OSF Theory Unification is the procedure which Θ-solves an OSF clause ϕ; *i.e.*, it transforms ϕ into a Θ-equivalent OSF clause ϕ' which is either $\perp$ or in Θ-solved form (and, in this case, exhibits it).

Frame Allocation:

$$(0)\quad \frac{\Gamma \qquad\qquad \vdash X : s \ \&\ \phi}{\Gamma \bigcup \Big\{ \{X \backslash Y_s\} \Big\} \vdash X : s \ \&\ \phi} \qquad \text{if } X\backslash Y_{s'} \notin F, \text{ for any } s' \in \mathcal{S}, \text{ for all } F \in \Gamma$$

Sort Intersection:

$$(1)\quad \frac{\Gamma \bigcup \Big\{ \{X \backslash Y_{s'}\} \cup F \Big\} \quad \vdash X : s \ \&\ X : s' \ \&\ \phi}{\Gamma \bigcup \Big\{ \{X \backslash Y_{s \wedge s'}\} \cup F \Big\} \vdash X : s \wedge s' \ \&\ \phi}$$

Inconsistent Sort:

$$(2)\quad \frac{\Gamma \bigcup \Big\{ \{X \backslash Y_{\bot}\} \cup F \Big\} \vdash \phi}{\emptyset \qquad\qquad \vdash \bot}$$

Variable Elimination:

$$(3)\quad \frac{\Gamma \qquad \vdash X \doteq X' \ \&\ \phi}{\Gamma[X'/X] \vdash X \doteq X' \ \&\ \phi[X'/X]} \qquad \text{if } X \neq X' \text{ and } X \in \mathit{Var}(\Gamma) \cup \mathit{Var}(\phi)$$

Feature Decomposition:

$$(4)\quad \frac{\Gamma \vdash X.f \doteq X' \ \&\ X.f \doteq X'' \ \&\ \phi}{\Gamma \vdash X.f \doteq X' \ \&\ X' \doteq X'' \ \&\ \phi}$$

FIGURE 4.1. Weak OSF Theory Normalization Rules—Empty Theory

We will show that such a procedure exists that transforms ϕ successively until either $\bot$ or a Θ-solved form is obtained. If ϕ is Θ-equivalent to $\bot$, then $\bot$ is reachable in a finite number of steps. Generally, however, there exists no such procedure that is always terminating. Indeed, if such a procedure existed, then according to Proposition 3.1, there would be an algorithm deciding whether an OSF constraint ϕ is satisfiable in the Θ-OSF algebra Ψ_Θ. This, however, is impossible as Theorem 4.1 will show.

The normalization rules that perform OSF theory unification are given in Figures 4.1, 4.2, and 4.3 and are called OSF theory normalization rules.[5]The rules in Figures 4.1 and 4.2 alone are called the weak (OSF theory) normalization rules.

Next, we will informally describe and motivate the effect of each rule. Before doing so, we need to introduce additional notation. We will follow a strict variable-naming convention in order to differentiate variables. We shall use X's for variables appearing in a formula being normalized, and call these *global* or *formula* variables. We shall use Y's for variables in the theory, and call these *local* or *theory* variables. Local and global variables are always assumed disjoint.

The theory variables appearing in a sort definition $\Theta(s)$ are all local to this

[5]A full example of sort-unfolding using these rules is detailed in appendix Section A.

Feature Constraint:

$$(5)\quad \frac{\Gamma \bigcup \left\{ \{X \backslash Y\} \cup F \right\} \quad \vdash X.f \doteq X' \ \&\ \phi}{\Gamma \bigcup \left\{ \{X \backslash Y, X' \backslash Y'\} \cup F \right\} \vdash X.f \doteq X' \ \&\ X' : Sort(Y') \ \&\ \phi} \quad \begin{array}{l}\text{if } f_\Theta(Y) = Y' \\ \text{and } X' \backslash Y' \notin F\end{array}$$

Frame Merging:

$$(6)\quad \frac{\Gamma \bigcup \left\{ \{X \backslash Y_s\} \cup F, \{X \backslash Y_{s'}\} \cup F' \right\} \vdash \phi}{\Gamma \bigcup \left\{ \{X \backslash Y_{s \wedge s'}\} \cup F \cup F' \right\} \quad \vdash \phi}$$

Frame Reduction:

$$(7)\quad \frac{\Gamma \bigcup \left\{ \{X \backslash Y, X \backslash Y'\} \cup F \right\} \vdash \phi}{\Gamma \bigcup \left\{ \{X \backslash Y\} \cup F \right\} \quad \vdash \phi} \quad \text{if } Y \leq Y'$$

Theory Coreference:

$$(8)\quad \frac{\Gamma \bigcup \left\{ \{X \backslash Y, X' \backslash Y\} \cup F \right\} \vdash \phi}{\Gamma \bigcup \left\{ \{X \backslash Y\} \cup F \right\} \quad \vdash X \doteq X' \ \&\ \phi}$$

FIGURE 4.2. Weak OSF Theory Normalization Rules—Non-Empty Theory

Theory Feature Closure:

$$(9)\quad \frac{\Gamma \vdash \phi}{\Gamma \vdash X.f \doteq Z \ \&\ \phi} \quad \begin{array}{l}\text{if } X \backslash Y \in F \text{ and } X \backslash Y' \in F' \text{ for some } F, F' \in \Gamma, \\ \text{and both } f_\Theta(Y),\ f_\Theta(Y') \text{ exist} \\ (Z \text{ is a new variable})\end{array}$$

FIGURE 4.3. Strong OSF Theory Normalization Rule

definition alone. Thus, without loss of generality, we shall assume distinct names for all variables across sort definitions. More precisely, $s \neq s' \Rightarrow Var(\Theta(s)) \cap Var(\Theta(s')) = \emptyset$. Let $Var(\Theta) = \bigcup_{s \in \mathcal{S}} Var(\Theta(s))$ denote the set of all theory variables.

We shall use Z's for new global variables introduced into a formula being normalized. Finally, the theory variable at the root of $\Theta(s)$, the definition of a sort s, will be identified as Y_s. We will denote by $Roots(\Theta)$ the set of all root theory variables.

We will denote by $f_\Theta(Y)$ the theory variable Y', if it exists, such that $f(Y) = Y'$ in some sort definition $\Theta(s)$.

Note that $Roots(\Theta)$ is in bijection with $\mathcal{S}$. In particular, the operation $\wedge$ on $\mathcal{S}$ can be defined on $Roots(\Theta)$ as $Y_s \wedge Y_{s'} = Y_{s \wedge s'}$. As a result, the partial ordering on sorts may be carried over to theory root variables; namely, $Y_s \leq Y_{s'}$ if and only if $s \leq s'$. This ordering may then be extended homomorphically from $Roots(\Theta)$ to all theory variables in $Var(\Theta)$ as follows:

$$Y_1 \leq Y_2 \text{ iff } \begin{cases} Y_1 = Y_{s_1} \text{ and } Y_2 = Y_{s_2} \text{ and } s_1 \leq s_2; \\ \text{or,} \\ \exists f,\ Y_1 = f_\Theta(Y_1') \text{ and } Y_2 = f_\Theta(Y_2'),\ \text{and } Y_1' \leq Y_2'. \end{cases}$$

This relation is well-defined because Θ is order-consistent.

As in the case of plain OSF normalization, each rule specifies a transformation of the pattern in the numerator into that of the denominator. While the rules of Figure 2.2 transform OSF clauses, the new rules transform *contexted OSF clauses.* A contexted clause is a formula of the form $\Gamma \vdash \phi$ where ϕ is an OSF clause and Γ, called the *context*, is a set of *frames.* A frame is a set of pairs of variables $X\backslash Y$ (read "X stands for Y") where $X \in Var(\phi)$ and $Y \in Var(\Theta)$. We write simply ϕ for $\emptyset \vdash \phi$.

The rules proceed to normalize a formula from an originally empty context, creating at most one frame per formula variable. These rules maintain frames so that there is exactly one root theory variable per frame at any moment. The global variable in a frame that stands for the root local variable is called the frame's *principal variable.* Intuitively, one may think of a context as a set of activation frames, each being a local environment for a sort occurring in the formula ϕ, the pairs indicating which global variables stand for which local variables. Alternatively, one can think of a frame as the materialization of an object instance. Thus, the rules must ensure that a global variable is eventually principal in at most one frame. In addition, note that the rules will materialize only what is necessary to ensure that the instance is consistent with the class definition.

Rule (0) simply spawns a new frame for a global variable if none exists for it yet in the current context. This is akin to creating an instance in object-oriented programming. Rules (1)–(4) do exactly the same work as Rules (1)–(4) in Figure 2.2. The only difference is that they keep track of the sort information in the context Γ using root theory variables. Rule (5) ensures that whenever a feature is used in the formula it fits the sort constraint, if any, imposed on it by the theory. Rule (6) recognizes that a global variable is principal in two frames and merges them. This case arises from variable elimination; *i.e.*, from two originally distinct global variables that are later made to corefer. Rule (7) determines that the same global variable stands for two ordered local variables within the same frame. Therefore, the global variable must stand for the lesser of these two local variables. Rule (8) enforces an equation of paths as prescribed by the theory when it finds that two distinct global variables stand for the same local variable in the same frame.

Rule (9) looks more complex than Rules (0)–(8). In fact, it simply completes the enforcing of functionality of features. Functionality of a feature f means that if $X = X'$ then $f(X) = f(X')$. Rule (4) enforces feature functionality in the formula alone as f is applied at two occurrences of the same variable in the formula. Rule (5) does the same for the case when one occurrence is in the formula and the other is in the theory on the corresponding local variable.

The only remaining case is the following. At some point, a formula variable X may stand for two distinct theory variables that both constrain the feature f in the theory. If X does not have the feature f in the formula, it may yet be that the two theory variables that X stands for do not agree on this implicit feature. This is best illustrated by an example. Let us assume that

$$\begin{aligned} \Theta(s) = Y_s : s(&a \Rightarrow Y_1 : s_1, \\ &b \Rightarrow Y : s(a \Rightarrow Y_2 : s_2)) \end{aligned}$$

such that $s_1 \wedge s_2 = \bot$, and consider the formula $X : s(b \Rightarrow X)$. Because of its coreference in the formula, X will eventually stand for both Y_s and Y. Both these theory variables constrain the feature a. But, since X does not possess the feature a in the formula, Rule (5) cannot be applied. However, the formula is clearly inconsistent because Θ constrains X to be both in $a^{-1}([\![s_1]\!])$ and $a^{-1}([\![s_2]\!])$ whose intersection is empty.

This consistency check is what Rule (9) is designed to perform. It "fakes" the presence in the formula of the missing theory feature f. It does this by introducing a new global variable into the formula to be the result of applying f to the global variable in question. After that, Rule (5) will do the right thing, bridging the gap between the two local variables using this new global variable. In fact, it guarantees the transitivity of congruence of feature path equations as per the theory. It is this rule that may make the normalization algorithm diverge on consistent formulae as there is, in general, no way to predict how deep along a feature path an inconsistency might arise. This is indeed confirmed by the following fact.[6]

Theorem 4.1. Given a well-formed order-consistent OSF theory Θ, the problem of the satisfiability of an OSF constraint in the Θ-OSF algebra Ψ_Θ is generally undecidable.

Proof. We show that a *complete* OSF Theory Unification algorithm is also a decision procedure for the word problem for Thue systems of equations on strings (see, *e.g.*, [18]). Consider a finite alphabet Σ and a finite set $E \subset \Sigma^\star \times \Sigma^\star$ of equations of words on Σ. The word problem that consists in deciding whether two words w_1 and w_2 in $\Sigma^\star$ are equal modulo the equations in E can be encoded as the following OSF theory unification problem. Let us take for sorts $\mathcal{S} = \{\top, s, 0, 1, \bot\}$ with $0 < s$, $1 < s$, and $0 \wedge 1 = \bot$, and for the features $\mathcal{F} = \Sigma$. Let us define Θ such that $\Theta(s)$ is the ψ-term whose variables are all sorted with s and such that to each equation $u = v$ in E corresponds a pair of feature paths from the root that meet in a common variable at their end.

Let us take an example to explicate this encoding. Consider the system of equations $E = \{bc = ed, ae = b, bd = de\}$. It is encoded as an OSF theory over the sorts of $\mathcal{S}$ above and the set of features $\mathcal{F} = \{a, b, c, d, e\}$. The sort definitions are:

$$\begin{aligned}\Theta(s) = s(&b \Rightarrow Y1 : s(c \Rightarrow Y2 : s, d \Rightarrow Y3 : s),\\ &e \Rightarrow s(d \Rightarrow Y2),\\ &a \Rightarrow s(e \Rightarrow Y1),\\ &d \Rightarrow s(e \Rightarrow Y3)).\end{aligned}$$

As for $\Theta(0)$ and $\Theta(1)$, they both inherit the exact same structure as $\Theta(s)$ except for the root sort since $\mathit{Sort}(\mathit{Root}(\Theta(0))) = 0$, and $\mathit{Sort}(\mathit{Root}(\Theta(1))) = 1$. Clearly, Θ is a well-formed and order-consistent OSF theory.

Now, to decide whether an equality $w_1 = w_2$ holds modulo the equations, it suffices to normalize the OSF term consisting of just two non-coreferring feature

[6] A related, but different result can be found in [20] where well-formedness, order-consistency and the *existence* of one generic model of an OSF theory (there called a system a recursive sort equations) are not considered. In fact, without Proposition 3.1, we do not know whether there is *any* OSF constraint which is satisfiable modulo a system of sort definitions. Thus, the result in [20] is on a test of satisfiability in *all* Θ-OSF algebras, and its proof has to provide the construction of a particular one.

paths w_1 and w_2, and whose root sort is s and all other sorts are $\top$ except for the tips of the two paths which are 0 and 1. For a decision procedure to be complete, it must make the two paths corefer (and thus end with a sort clash, *i.e.*, normalize the dissolved ψ-term to the equivalent OSF clause $\perp$) if and only if the equality $w_1 = w_2$ holds. Otherwise, *i.e.*, if and only if the equality does not hold, it will normalize the dissolved ψ-term to an equivalent Θ-solved OSF clause and, thus, exhibit its Θ-satisfiability.

For example, to decide whether $abc = de$ modulo the above equations, we need to check whether the ψ-term:

$$\begin{array}{l} s(a \Rightarrow \top(b \Rightarrow \top(c \Rightarrow 0)), \\ \quad d \Rightarrow \top(e \Rightarrow 1)) \end{array}$$

(*i.e.*, the OSF clause obtained by dissolving it) is not satisfiable modulo the OSF theory Θ given above. □

Lemma 4.1. If ϕ is transformed into $\Gamma \vdash \phi'$ by the (strong) OSF theory normalization rules, then ϕ is Θ-equivalent to ϕ'.

Proof. For a contexted formula $\Gamma \vdash \phi$, let us define the OSF clause:

$$[\Gamma \vdash \phi] = \phi \cup \bigcup\{C_{\Theta(s)}[X] \ \& \ Y_1 \doteq X_1 \ \& \ \ldots \ \& \ Y_n \doteq X_n\}$$

where the big union is taken over the frames $\{X\backslash Y_s, X_1\backslash Y_1, \ldots, X_n\backslash Y_n\} \in \Gamma$.

The variables in $C_{\Theta(s)}[X] \ \& \ Y_1 \doteq X_1 \ \& \ \ldots \ \& \ Y_n \doteq X_n$ are taken new for each of these frames.

Clearly, ϕ is Θ-equivalent to $[\Gamma \vdash \phi]$.

If $\Gamma \vdash \phi$ is transformed to $\Gamma' \vdash \phi'$ then $[\Gamma \vdash \phi]$ is Θ-equivalent to $[\Gamma' \vdash \phi']$. This can be verified by inspection of each of the OSF theory normalization rules. For each application by one of these, we will give corresponding Θ-equivalence transformations on $[\Gamma \vdash \phi]$. These will either consist of adding $C_{\Theta(s)}[X]$ (again, obtained by naming its variables apart), or of applications of one of the rules of Figure 2.2. Since these are all equivalence transformations, $[\Gamma \vdash \phi]$ is equivalent, and thus also Θ-equivalent, to $[\Gamma' \vdash \phi']$.

Each application of Rule (0) of Figure 4.1 adds a frame $\{X\backslash Y_s\}$ to the context of $\Gamma \vdash \phi$. The corresponding transformation on the OSF clause $[\Gamma \vdash \phi]$ consists of adding the OSF clause $C_{\Theta(s)}[X]$. One hereby obtains a Θ-equivalent OSF clause.

Clearly, each step by application of Rule (i) of Figure 4.1 to $\Gamma \vdash \phi$ corresponds to one step of application of Rule (i) of Figure 2.2 to $[\Gamma \vdash \phi]$, for $i = 1, \ldots, 4$. In the case of Rule (1), if $s \wedge s'$ is a strict subsort of s', then, in addition, $C_{\Theta(s \wedge s')}[X]$ has to be added.

An application of Rule (5) of Figure 4.2 to $\Gamma \vdash \phi$ corresponds to one variable elimination step, followed by one step of application of Rule (4) of Figure 2.2 (the feature constraint $Y.f \doteq Y'$ is part of $[\Gamma \vdash \phi]$), followed by another variable elimination step to $[\Gamma \vdash \phi]$.

An application of Rule (6) of Figure 4.2 to $\Gamma \vdash \phi$ yielding $\Gamma' \vdash \phi'$ corresponds to two variable elimination steps, followed by one step of application of Rule (1) of Figure 2.2 to $[\Gamma \vdash \phi]$. We add the OSF clause $C_{\Theta(s \wedge s')}[X]$, hereby obtaining the Θ-equivalent OSF clause $[\Gamma' \vdash \phi']$.

An application of Rule (7) of Figure 4.2 corresponds to one variable elimination step, followed by one step of application of Rule (4) of Figure 2.2 (the feature constraints $X'.f \doteq X$ and $X'.f \doteq Y$ are part of the derived OSF clause).

An application of Rule (8) of Figure 4.2 corresponds to several variable elimination steps.

Finally, an application of Rule (9) in Figure 4.3 adds a feature constraint $X.f \doteq Z$ with a new variable Z. Clearly, $[\Gamma \vdash \phi]$ is Θ-equivalent to $[\Gamma \vdash \phi \ \& \ X.f \doteq Z]$. □

Theorem 4.2. If ϕ is transformed into the non-bottom normal form $\Gamma_N \vdash \phi_N$ by the (strong) OSF theory normalization rules, then ϕ_N is an OSF clause in Θ-solved form which is Θ-equivalent to ϕ.

In particular, because we assume Θ to be well-formed and order-consistent, ϕ is, then, Θ-satisfiable (*e.g.*, in Ψ_Θ). Of course, if ϕ is transformed into $\phi_N = \bot$, then ϕ is not Θ-satisfiable.

PROOF. It is easy to see that, if $\Gamma_N \vdash \phi_N$ is in non-bottom normal form, then $[\Gamma_N \vdash \phi_N]$ is in solved form. Namely, otherwise one could apply an OSF clause normalization rule from Figure 2.2 to $[\Gamma_N \vdash \phi_N]$; this application could, in turn, be simulated by an application of an OSF theory normalization rule from Figure 4.1–4.3. But this means exactly that ϕ_N is in Θ-solved form. □

Theorem 4.3. The weak OSF theory normalization rules are terminating and confluent (modulo a renaming of formula variables).

PROOF. The number of times a sort definition is unfolded (*via* Rule (0)) is limited by the number of sort and of feature constraints in the OSF clause to be normalized. Let ϕ' is the OSF clause obtained from ϕ by doing all these unfoldings, *i.e.*, by adding the OSF clauses $C_{\Theta(s)}[X]$, obtained by dissolving the corresponding ψ-terms $\Theta(s)$ and naming its variables apart. Then, using the correspondence from the proof of Theorem 4.2, each OSF theory weak normalization step on ϕ can be simulated by an OSF clause normalization step on ϕ'. Then, Theorem 2.1 yields the statement. □

Theorem 4.4. The weak OSF theory normalization rules normalize a formula in almost linear time (in the size of the formula).

PROOF. We use the simulation of OSF theory normalization by plain OSF clause normalization from the preceding proof and the fact that OSF clause normalization is almost linear (the size of each unfolded sort definition is assumed constant).[7] □

Theorem 4.5. If terminating, the (strong) OSF theory normalization rules are confluent (modulo a renaming of formula variables).

PROOF. If the (strong) OSF theory normalization is terminating, Rule (9) is applied only a finite number of times. Each time, it adds a feature constraint $X.f \doteq Z$ with a new variable Z. Let σ be the OSF clause of all these feature constraints. Then, $\phi \ \& \ \sigma$ is transformed into the non-bottom normal form $\Gamma_N \vdash \phi_N$ by the weak OSF theory normalization rules only, and we can apply Theorem 4.3. □

Theorem 4.6. If ϕ is not Θ-satisfiable then ϕ is reduced to $\bot$ by the OSF theory normalization rules.

[7]By "almost linear" we refer to the well-known complexity bound of the Union/Find problem [1] that underlies the implementation of OSF term normalization.

Proof. Using Lemma 3.1, if ϕ is not Θ-satisfiable, then ϕ^* is not satisfiable.

We use the fact (which is a consequence of the compactness theorem [14]) that, given a first-order theory T and a set W of open first-order formulae, $T \cup (\exists)W$ has a model if and only if, for every finite subset F of W, $T \cup (\exists)F$ has a model. Here, T is given by the axioms $Axiom_{[s \wedge s' = s'']}$ and $Axiom_{[\Theta(s)]}$ specifying the sort hierarchy and the OSF theory.

Thus, if a possibly infinite OSF clause is not satisfiable, then there exists a finite subset of it that is not satisfiable. Now, if ϕ is not Θ-satisfiable, then there exists an index n such that ϕ^n is not satisfiable. Let ϕ' be the minimal non-satisfiable extension of ϕ with sort-unfoldings, *i.e.*, with additions of OSF clauses of the form $C_{\Theta(s \wedge s')}[X]$.

According to Theorem 2.1, the finite OSF clause ϕ' is reduced to $\perp$ using the OSF clause normalization rules (1)–(4) of Figure 2.2. Now, every OSF clause normalization step can be simulated by an OSF theory normalization step, under the correspondence described in the proof of Theorem 4.2. The only difficulty is the application of the feature decomposition rule on two feature constraints which both come from sort unfoldings, *i.e.*, from added OSF clauses of the form $\phi(\Theta(s))$. In this case, the applicability of Rule (9) has to be shown. But if follows from the fact (Theorem 4.3) that the weak OSF theory normalization are terminating. That is, after finitely many applications of Rules (0) to (8), none of them is applicable, and, thus, Rule (9) is. $\square$

We have divided the normalization processes into two phases. The first phase, consisting of the weak normalization rules, is guaranteed to terminate in almost linear time. If the first phase ends with the clause still not in normal form then the second phase, one application of the strong normalization rule, is performed. From these two phases we derive a complete normalization strategy. Namely, the repeated application of phase one followed by phase two. Note that if the process terminates, it terminates in phase one.

The fact that it is only Rule (9) that leads to undecidability gives us the ability to explore what makes certain theories and queries non-terminating. For instance, a loose criterion for a theory that guarantees that the normalization of all queries will terminate is that no two variables have the same feature symbols. This is clear by looking at Rule (9)'s side conditions. It is also clear that more complex, yet decidable, analysis can provide programmers using this system with this guarantee.

Another benefit of the separation is that the terminating rules can be used to "compile" a theory by using a partial evaluation technique. Namely, each sort definition can be normalized with respect to the theory using the terminating rules only. In fact, we have now completed and implemented a compiling scheme for the decidable subset of the ten rules [10]. This scheme also accommodates arbitrary user-defined constraints associated to sort as actually allowed in LIFE [4].

5. CONCLUSION

We have presented a formal system of record objects with recursive class definitions accommodating multiple inheritance and equational constraints among feature paths, including self-reference. Although the problem of normalizing an object to fit class templates is undecidable in general, we have proposed a complete and efficient set of rules to perform this normalization whenever it may be done.

An interesting property of this OSF theory unification process is that it consists of a terminating set of rules and an additional one which makes it complete. This property can be used to explore the exact situations when the full set of rules will be guaranteed to terminate.

A. A DETAILED EXAMPLE

Let us take $\mathcal{S} = \{\top, s, s_1, s_2, s_3, \bot\}$ ordered minimally such that $s_1 \wedge s_2 = s_3$ and define Θ as:

$\Theta(s_1) = Y_{s_1} : s_1(f_1 \Rightarrow Y_1 : s)$

$\Theta(s_2) = Y_{s_2} : s_2(f_2 \Rightarrow Y_2 : s)$

$\Theta(s_3) = Y_{s_3} : s_3(f_1 \Rightarrow Y_3 : s(f \Rightarrow Y_4 : s), f_2 \Rightarrow Y_3)$

$\Theta(s) \;\; = Y_s : s(f \Rightarrow Y_5 : s)$.

The set of all theory variables is

$Var(\Theta) = \{Y_\top, Y_\bot, Y_{s_1}, Y_{s_2}, Y_{s_3}, Y_s, Y_1, Y_2, Y_3, Y_4, Y_5\}$.

The ordering relation on $Var(\Theta)$ is such that $Y_\bot \leq Y_x \leq Y_\top$, for all $x \in \mathcal{S}$, and $Y_{s_3} \leq Y_{s_1}$, $Y_{s_3} \leq Y_{s_2}$, $Y_3 \leq Y_1$, $Y_3 \leq Y_2$, as well as all reflexive pairs.

Unifying the two ψ-terms $t_1 = s_1(f_1 \Rightarrow s)$ and $t_2 = s_2(f_2 \Rightarrow s)$ modulo the *empty* theory yields the ψ-term (up to variable renaming):

$t_1 \wedge_\emptyset t_2 = s_3(f_1 \Rightarrow s, f_2 \Rightarrow s)$.

However, with respect to the theory Θ above, it yields the ψ-term (up to variable renaming):

$t_3 = t_1 \wedge_\Theta t_2 = s_3(f_1 \Rightarrow X : s(f \Rightarrow s), f_2 \Rightarrow X)$

as illustrated by the following reduction trace.[8]

Step 0. Given empty context and formula:

$\emptyset$

$\vdash \;\; \underline{X_1 : s_1} \;\&\; X_1.f_1 \doteq X_1' \;\&\; X_2 : s_2 \;\&\; X_2.f_2 \doteq X_2' \;\&\; X_1' : s \;\&\; X_2' : s \;\&\; X_1 \doteq X_2$

Step 1. Use Rule (0)—Frame Allocation:

$\{\underline{X_1 \backslash Y_{s_1}}\}$

$\vdash \;\; X_1 : s_1 \;\&\; \underline{X_1.f_1 \doteq X_1'} \;\&\; X_2 : s_2 \;\&\; X_2.f_2 \doteq X_2' \;\&\; X_1' : s \;\&\; X_2' : s \;\&\; X_1 \doteq X_2$

Step 2. Use Rule (5)—Feature Constraint:

$\{X_1 \backslash Y_{s_1}, X_1' \backslash Y_1\}$

$\vdash \;\; X_1 : s_1 \;\&\; X_1.f_1 \doteq X_1' \;\&\; \underline{X_1' : s} \;\&\; X_2 : s_2 \;\&\; X_2.f_2 \doteq X_2' \;\&\; X_1' : s \;\&\; X_2' : s$
$\;\&\; X_1 \doteq X_2$

[8] In the derivation sequence that follows, the parts of a contexted formula that make up the redex of the rule to apply *next* are highlighted by underlining.

Step 3. Use Rule (0)—Frame Allocation:

$\{X_1\backslash Y_{s_1}, X_1'\backslash Y_1\},\ \{X_1'\backslash Y_s\}$

$\vdash\quad X_1 : s_1 \ \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ \underline{X_2 : s_2} \ \& \ X_2.f_2 \doteq X_2' \ \& \ X_1' : s \ \& \ X_2' : s$
$\ \ \& \ X_1 \doteq X_2$

Step 4. Use Rule (0)—Frame Allocation:

$\{X_1\backslash Y_{s_1}, X_1'\backslash Y_1\},\ \{X_1'\backslash Y_s\},\ \{\underline{X_2\backslash Y_{s_2}}\}$

$\vdash\quad X_1 : s_1 \ \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_2 : s_2 \ \& \ \underline{X_2.f_2 \doteq X_2'} \ \& \ X_1' : s \ \& \ X_2' : s$
$\ \ \& \ X_1 \doteq X_2$

Step 5. Use Rule (5)—Feature Constraint:

$\{X_1\backslash Y_{s_1}, X_1'\backslash Y_1\},\ \{X_1'\backslash Y_s\},\ \{X_2\backslash Y_{s_2}, X_2'\backslash Y_2\}$

$\vdash\quad X_1 : s_1 \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_2 : s_2 \ \& \ X_2.f_2 \doteq X_2' \ \& \ \underline{X_2' : s} \ \& \ X_1' : s$
$\ \ \& \ X_2' : s \ \ \& \ X_1 \doteq X_2$

Step 6. Use Rule (0)—Frame Allocation:

$\{X_1\backslash Y_{s_1}, X_1'\backslash Y_1\},\ \{\underline{X_1'\backslash Y_s}\},\ \{X_2\backslash Y_{s_2}, X_2'\backslash Y_2\},\ \{X_2'\backslash Y_s\}$

$\vdash\quad X_1 : s_1 \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_2 : s_2 \ \& \ X_2.f_2 \doteq X_2' \ \& \ X_2' : s \ \& \ \underline{X_1' : s}$
$\ \ \& \ X_2' : s \ \ \& \ X_1 \doteq X_2$

Step 7. Use Rule (1)—Sort Intersection:

$\{X_1\backslash Y_{s_1}, X_1'\backslash Y_1\},\ \{X_1'\backslash Y_s\},\ \{X_2\backslash Y_{s_2}, X_2'\backslash Y_2\},\ \{\underline{X_2'\backslash Y_s}\}$

$\vdash\quad X_1 : s_1 \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_2 : s_2 \ \& \ X_2.f_2 \doteq X_2' \ \& \ \ \& \ X_2' : s$
$\ \ \& \ \underline{X_2' : s} \ \ \& \ X_1 \doteq X_2$

Step 8. Use Rule (1)—Sort Intersection:

$\{X_1\backslash Y_{s_1}, X_1'\backslash Y_1\},\ \{X_1'\backslash Y_s\},\ \{X_2\backslash Y_{s_2}, X_2'\backslash Y_2\},\ \{\underline{X_2'\backslash Y_s}\}$

$\vdash\ X_1 : s_1 \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_2 : s_2 \ \& \ X_2.f_2 \doteq X_2' \ \& \ X_2' : s \ \& \ \underline{X_1 \doteq X_2}$

Step 9. Use Rule (3)—Variable Elimination:

$\{\underline{X_1\backslash Y_{s_1}}, X_1'\backslash Y_1\},\ \{X_1'\backslash Y_s\},\ \{X_1\backslash Y_{s_2}, X_2'\backslash Y_2\},\ \{X_2'\backslash Y_s\}$

$\vdash\ \underline{X_1 : s_1} \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ \underline{X_1 : s_2} \ \& \ X_1.f_2 \doteq X_2' \ \& \ X_2' : s \ \& \ X_1 \doteq X_2$

Step 10. Use Rule (1)—Sort Intersection:

$\underline{\{X_1\backslash Y_{s_3}, X_1'\backslash Y_1\}}, \{X_1'\backslash Y_s\},\ \underline{\{X_1\backslash Y_{s_2}, X_2'\backslash Y_2\}}, \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3 \ \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_1.f_2 \doteq X_2' \ \& \ X_2' : s \ \& \ X_1 \doteq X_2$

Step 11. Use Rule (6)—Frame Merging:

$\{\underline{X_1\backslash Y_{s_3}}, X_1'\backslash Y_1, X_2'\backslash Y_2\},\ \{X_1'\backslash Y_s\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3 \ \& \ \underline{X_1.f_1 \doteq X_1'} \ \& \ X_1' : s \ \& \ X_1.f_2 \doteq X_2' \ \& \ X_2' : s \ \& \ X_1 \doteq X_2$

Step 12. Use Rule (5)—Feature Constraint:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, X_1'\backslash Y_1, X_2'\backslash Y_2\},\ \{\underline{X_1'\backslash Y_s}\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3 \ \& \ X_1.f_1 \doteq X_1' \ \& \ \underline{X_1' : s} \ \& \ \underline{X_1' : s} \ \& \ X_1.f_2 \doteq X_2' \ \& \ X_2' : s \ \& \ X_1 \doteq X_2$

Step 13. Use Rule (1)—Sort Intersection:

$\{X_1\backslash Y_{s_3}, \underline{X_1'\backslash Y_3}, \underline{X_1'\backslash Y_1}, X_2'\backslash Y_2\},\ \{X_1'\backslash Y_s\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_2'\ \&\ X_2' : s\ \&\ X_1 \doteq X_2$

Step 14. Use Rule (7)—Frame Reduction:

$\{\underline{X_1\backslash Y_{s_3}}, X_1'\backslash Y_3, X_2'\backslash Y_2\},\ \{X_1'\backslash Y_s\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ \underline{X_1.f_2 \doteq X_2'}\ \&\ X_2' : s\ \&\ X_1 \doteq X_2$

Step 15. Use Rule (5)—Feature Constraint:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, X_2'\backslash Y_3, X_2'\backslash Y_2\},\ \{X_1'\backslash Y_s\},\ \{\underline{X_2'\backslash Y_s}\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_2'\ \&\ \underline{X_2' : s}\ \&\ \underline{X_2' : s}\ \&\ X_1 \doteq X_2$

Step 16. Use Rule (1)—Sort Intersection:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, \underline{X_2'\backslash Y_3}, \underline{X_2'\backslash Y_2}\},\ \{X_1'\backslash Y_s\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_2'\ \&\ X_2' : s\ \&\ X_1 \doteq X_2$

Step 17. Use Rule (7)—Frame Reduction:

$\{X_1\backslash Y_{s_3}, \underline{X_1'\backslash Y_3}, \underline{X_2'\backslash Y_3}\},\ \{X_1'\backslash Y_s\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_2'\ \&\ X_2' : s\ \&\ X_1 \doteq X_2$

Step 18. Use Rule (8)—Theory Coreference:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3\},\ \{X_1'\backslash Y_s\},\ \{X_2'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_2'\ \&\ X_2' : s\ \&\ X_1 \doteq X_2\ \&\ \underline{X_1' \doteq X_2'}$

Step 19. Use Rule (3)—Variable Elimination:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3\},\ \{\underline{X_1'\backslash Y_s}\},\ \{X_1'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ \underline{X_1' : s}\ \&\ X_1.f_2 \doteq X_1'\ \&\ \underline{X_1' : s}\ \&\ X_1 \doteq X_2\ \&\ X_1' \doteq X_2'$

Step 20. Use Rule (1)—Sort Intersection:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3\},\ \underline{\{X_1'\backslash Y_s\}}, \underline{\{X_1'\backslash Y_s\}}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_1'\ \&\ X_1 \doteq X_2\ \&\ X_1' \doteq X_2'$

Step 21. Use Rule (6)—Frame Merging:

$\{X_1\backslash Y_{s_3}, \underline{X_1'\backslash Y_3}\},\ \{\underline{X_1'\backslash Y_s}\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_1'\ \&\ X_1 \doteq X_2\ \&\ X_1' \doteq X_2'$

Step 22. Use Rule (9)—Theory Feature Closure:

$\{X_1\backslash Y_{s_3}, \underline{X_1'\backslash Y_3}\},\ \{X_1'\backslash Y_s\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_1'\ \&\ \underline{X_1'.f \doteq Z}\ \&\ X_1 \doteq X_2\ \&\ X_1' \doteq X_2'$

Step 23. Use Rule (5)—Feature Constraint:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, Z\backslash Y_4\},\ \{\underline{X_1'\backslash Y_s}\}$

$\vdash\ X_1 : s_3\ \&\ X_1.f_1 \doteq X_1'\ \&\ X_1' : s\ \&\ X_1.f_2 \doteq X_1'\ \&\ \underline{X_1'.f \doteq Z}\ \&\ Z : s\ \&\ X_1 \doteq X_2$
$\&\ X_1' \doteq X_2'$

Step 24. Use Rule (5)—Feature Constraint:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, Z\backslash Y_4\},\ \{X_1'\backslash Y_s, Z\backslash Y_5\}$

$\vdash\quad X_1 : s_3 \quad \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_1.f_2 \doteq X_1' \ \& \ X_1'.f \doteq Z \ \& \ \underline{Z : s} \ \& \ Z : s$
$\quad \& \ X_1 \doteq X_2 \ \& \ X_1' \doteq X_2'$

Step 25. Use Rule (0)—Frame Allocation:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, Z\backslash Y_4\},\ \{X_1'\backslash Y_s, Z\backslash Y_5\},\ \{\underline{Z\backslash Y_s}\}$

$\vdash\quad X_1 : s_3 \quad \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_1.f_2 \doteq X_1' \ \& \ X_1'.f \doteq Z \ \& \ \underline{Z : s} \ \& \ \underline{Z : s}$
$\quad \& \ X_1 \doteq X_2 \ \& \ X_1' \doteq X_2'$

Step 26. Use Rule (1)—Sort Intersection:

$\{X_1\backslash Y_{s_3}, X_1'\backslash Y_3, Z\backslash Y_4\},\ \{X_1'\backslash Y_s, Z\backslash Y_5\},\ \{Z\backslash Y_s\}$

$\vdash\quad X_1 : s_3 \quad \& \ X_1.f_1 \doteq X_1' \ \& \ X_1' : s \ \& \ X_1.f_2 \doteq X_1' \ \& \ X_1'.f \doteq Z \ \& \ Z : s \ \& \ X_1 \doteq X_2$
$\quad \& \ X_1' \doteq X_2'$

This is in (strong) Θ-normal form, yielding the ψ-term (up to variable renaming):

$t_3 = t_1 \wedge_\Theta t_2 = s_3(f_1 \Rightarrow X : s(f \Rightarrow s), f_2 \Rightarrow X).$

REFERENCES

1. Alfred Aho, John Hopcroft, and Jeffrey Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.
2. Hassan Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures.* PhD thesis, Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1984.
3. Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351, 1986.
4. Hassan Aït-Kaci. An introduction to LIFE—programming with logic, inheritance, functions, and equations. In Dale Miller, editor, *Proceedings of the International Symposium on Logic Programming*, pages 52–68, Cambridge, MA, October 1993. MIT Press.
5. Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.
6. Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. *Journal of Logic Programming*, 16(3-4):195–234, July-August 1993.
7. Hassan Aït-Kaci and Andreas Podelski. Functions as passive constraints in LIFE. *ACM Transactions on Programming Languages and Systems*, 16(4):1279–1318, July 1994.
8. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. Order-sorted feature theory unification. PRL Research Report 32, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France, May 1993.
9. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. Order-sorted feature theory unification. In Dale Miller, editor, *Proceedings of the International Symposium on Logic Programming*, pages 506–524, Cambridge, MA, October 1993. MIT Press.
10. Hassan Aït-Kaci and Martin Vorbeck. Compiling order-sorted feature theory unification. Research report draft, Intelligent Software Group, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, February 1996.
11. Luca Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988.

12. Bob Carpenter. Typed feature structures: A generalization of first-order terms. In Vijay Saraswat and Kazunori Ueda, editors, *Proceedings of the 1991 International Symposium on Logic Programming*, pages 187–201, Cambridge, MA, 1991. MIT Press.
13. Bob Carpenter. *The Logic of Typed Feature Structures*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1992.
14. C. C. Chang and H. J. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, The Netherlands, third edition, 1990.
15. Martin C. Emele. Unification with lazy non-redundant copying. In *Proceedings of the 29th annual meeting of the ACL*, Berkeley, California, June 1991. Association for Computational Linguistics.
16. Martin C. Emele and Rémi Zajac. Typed unification grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (CoLing90)*, Helsinki, August 1990.
17. Jacques Garrigue and Hassan Aït-Kaci. The typed polymorphic label-selective λ-calculus. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages*, pages 35–47, January 1994.
18. Zohar Manna. *Mathematical Theory of Computation*. McGraw-Hill Computer Science Series. McGraw-Hill, New York, NY, 1974.
19. Didier Rémy. Type inference for records in a natural extension of ML. Technical Report 1431, INRIA, Rocquencourt, France, May 1991.
20. Gert Smolka. Feature constraint logic for unification grammar. *Journal of Logic Programming*, 12:51–87, 1992.
21. Mitchell Wand. Type inference for record concatenation and multiple inheritance. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, pages 92–97, 1989.