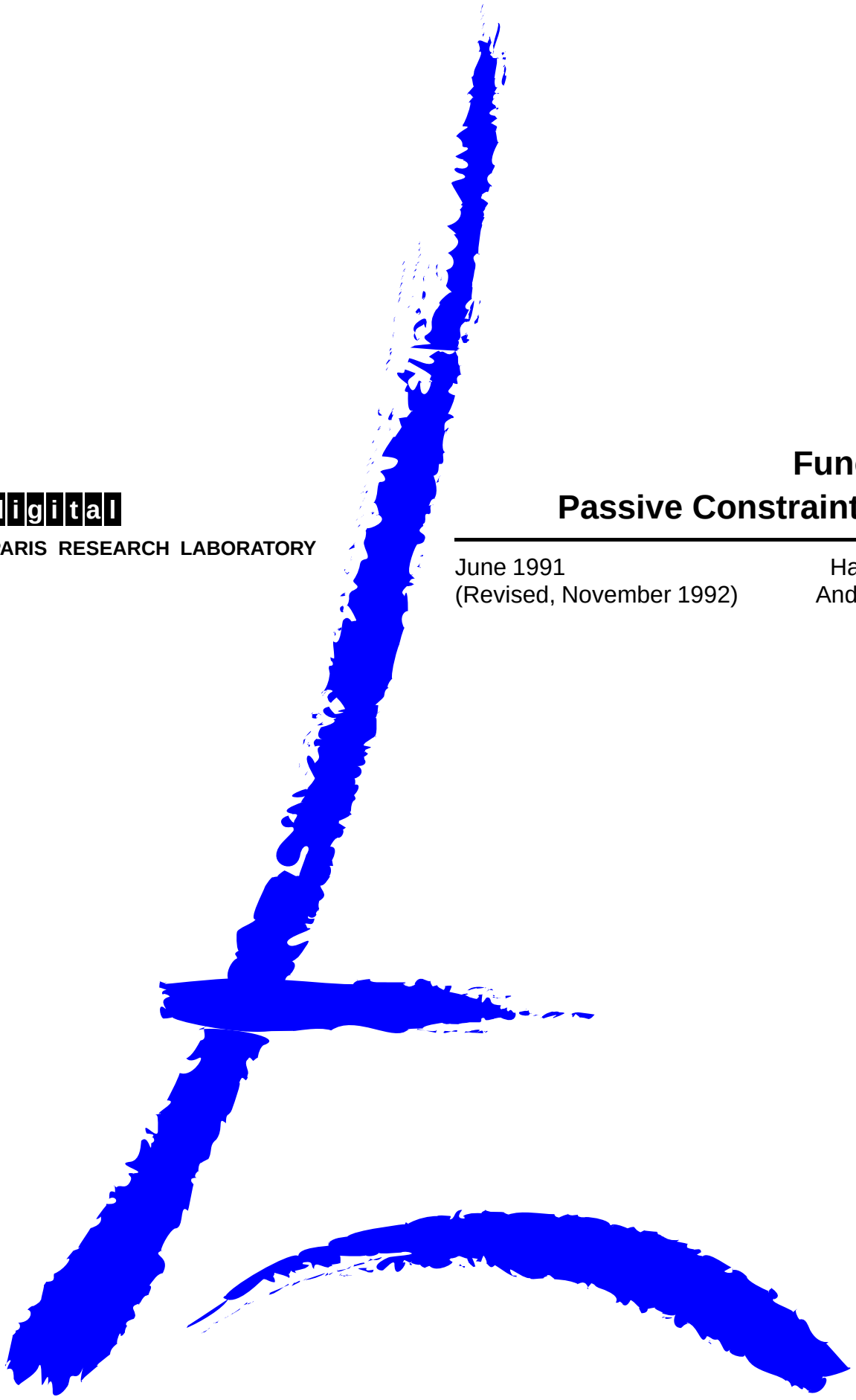

13

Functions as Passive Constraints in LIFE

digital
PARIS RESEARCH LABORATORY

June 1991
(Revised, November 1992)

Hassan Aït-Kaci
Andreas Podelski

13

Functions as Passive Constraints in LIFE

Hassan Aït-Kaci
Andreas Podelski

June 1991 (Revised, November 1992)

Publication Notes

The part of this report corresponding to Section 2 is to appear in the *Proceedings of the Third International Workshop on Extensions of Logic Programming*, edited by Evelina Lamma and Paola Mello, as Spinger-Verlag Lecture Notes in Computer Science, 1992.

Abstract

LIFE is an experimental programming language proposing to integrate logic programming, functional programming, and object-oriented programming. It replaces first-order terms with ψ-terms, data structures which allow computing with partial information. These are approximation structures denoting sets of values. LIFE further enriches the expressiveness of ψ-terms with functional dependency constraints. We must explain the meaning and use of functions in LIFE declaratively as solving partial information constraints. These constraints do not attempt to generate their solutions but behave as demons filtering out anything else. In this manner, LIFE functions act as declarative coroutines. We need to show that the ψ-term's approximation semantics is congruent with an operational semantics viewing functional reduction as an effective enforcing of passive constraints.

In this article, we develop a general formal framework for entailment and disentailment of constraints based on a technique called relative simplification, we study its operational and semantical properties, and we use it to account for functional application over ψ-terms in LIFE.

Résumé

LIFE est un langage de programmation expérimental proposant d'intégrer la programmation logique, la programmation fonctionnelle et la programmation orientée-objet. Il remplace les termes du premier ordre par des ψ-termes, des structures de données qui permettent le calcul avec information partielle. Ceux-ci sont des structures d'approximation qui dénotent des ensembles de valeurs. LIFE enrichit encore l'expressivité des ψ-termes avec des contraintes de dépendance fonctionnelle. Nous devons expliquer déclarativement la signification et l'utilisation des fonctions en LIFE comme une résolution de contraintes avec information partielle. Une telle contrainte ne tente pas d'énumérer ses solutions mais se comporte comme un démon excluant toute valuation qui n'en soit pas. De cette manière, les fonctions de LIFE jouent le rôle de coroutines déclaratives. Il nous faut montrer que la sémantique d'approximation du ψ-terme est congrue avec une sémantique opérationnelle concevant la réduction des fonctions comme une application effective de contraintes passives.

Dans cet article, nous présentons un schéma général pour tester la conséquence et la réfutation de contraintes basé sur une technique appelée simplification relative, en étudions les propriétés opérationnelles et sémantiques, et l'utilisons pour expliquer l'application fonctionnelle sur les ψ-termes dans LIFE.

Keywords

Constraint systems, first-order terms, ψ-terms, unification, matching, constraint entailment, constraint disentailment, relative simplification, logic programming, functional programming, committed-choice languages, residuation, coroutining.

Acknowledgements

We are grateful to Michael Maher for detailed comments on the form and contents of the material of Section 5, and to Gert Smolka for stimulating discussions and his continuing collaboration on issues related to the topic of this report.

We also wish to thank the members of the Paradise Project at PRL for many discussions that led to a better formulation than what we originally had. In particular, we are indebted to Peter Van Roy for his careful proofreading and commenting on the relative simplification rules, Roberto Di Cosmo for his perspicacious questions on our general residuation framework, and Seth Copen Goldstein for using our operational semantics as the basis of his compiler design for LIFE.

Not least, and as usual, we gratefully acknowledge the help of Jean-Christophe Patat, PRL's librarian, for his thorough proofreading and critical remarks. We thank him most of all, in this particular instance, for his patience waiting for the final delivery of this report's revision.

For the same reason, we also apologetically thank the many who have been waiting, more or less patiently, for this revision to come out finally, after ordering the advertized report only to receive a "Sorry, not available" note. The truth is that we had to fight a curse that was cast upon this report as its number should have ominously forewarned us. We only hope that the contents of this will have been worth their waiting so long.

Contents

> The paradox of culture is that language [...] is too linear, not comprehensive enough, too slow, too limited, too constrained, too unnatural, too much a product of its own evolution, and too artificial. This means that [man] must constantly keep in mind the limitations language places upon him.
>
> EDWARD T. HALL, *Beyond Culture*.

1 Introduction

1.1 The task

LIFE extends the computational paradigm of Logic Programming in two essential ways:

- using a data structure richer than that provided by first-order constructor terms; and,
- allowing interpretable functional expressions as *bona fide* terms.

The first extension is based on ψ-terms which are attributed partially-ordered sorts denoting sets of objects [1, 3]. In particular, ψ-terms generalize first-order constructor terms in their rôle as data structures in that they are endowed with a unification operation denoting type intersection. This gives an elegant means to incorporate a calculus of multiple inheritance into symbolic programming. Importantly, the denotation-as-value of constructor terms is replaced by the denotation-as-approximation of ψ-terms. As a result, the notion of fully defined element, or ground term, is no longer available. Hence, such familiar tools as variable substitutions, instantiation, unification, *etc.*, must be reformulated in the new setting [5].

The second extension deals with building into the unification operation a means to reduce functional expressions using definitions of interpretable symbols over data patterns.[1] Our basic idea is that unification is no longer seen as an atomic operation by the resolution rule. Indeed, since unification amounts to normalizing a conjunction of equations, and since this normalization process commutes with resolution, these equations may be left in a normal form that is not a fully solved form. In particular, if an equation involves a functional expression whose arguments are not sufficiently instantiated to match a *definiens* of the function in question, it is simply left untouched. Resolution may proceed until the arguments are *proven* to match a definition from the accumulated constraints in the context [4]. This simple idea turns out invaluable in practice. Here are a few benefits.

- Such non-declarative heresies as the *is* predicate in Prolog and the *freeze* meta-predicate in some of its extensions [21, 12] are not needed.
- Functional computations are determinate and do not incur the overhead of the search strategy needed by logic programming.

[1] Several patterns specifying a same function may possibly have overlapping denotations. Therefore, the order of the specified patterns defines an implicit priority, as is usual in functional programming using first-order patterns (*e.g.*, [16]).

- Higher-order functions are easy to return or pass as arguments since functional variables can be bound to partially applied functions.
- Functions can be called before the arguments are known, freeing the programmer from having to know what the data dependencies are.
- It provides a powerful search-space pruning facility by changing "generate-and-test" search into demon-controlled "test-and-generate" search.
- Communication with the external world is made simple and clean [9].
- More generally, it allows concurrent computation. Synchronization is obtained by checking entailment [20, 23].

There are two orthogonal dimensions to elucidate regarding the use of functions in LIFE:

- characterizing functions as approximation-driven coroutines; and,
- constructing a higher-order model of LIFE approximation structures.

This present article is concerned only with the first item, and therefore considers the case of first-order rules defining partial functions over ψ-terms.

1.2 The method

The most direct way to explain the issue is with an example. In LIFE, one can define functions as usual; say:

$$\begin{aligned} &\mathit{fact}(0) &&\rightarrow 1.\\ &\mathit{fact}(N : \mathit{int}) &&\rightarrow N * \mathit{fact}(N - 1). \end{aligned}$$

More interesting is the possibility to compute with partial information. For example:

$$\begin{aligned} &\mathit{minus}(\mathit{negint}) &&\rightarrow \mathit{posint}.\\ &\mathit{minus}(\mathit{posint}) &&\rightarrow \mathit{negint}.\\ &\mathit{minus}(\mathit{zero}) &&\rightarrow \mathit{zero}. \end{aligned}$$

Let us assume that the symbols *int*, *posint*, *negint*, and *zero* have been defined as sorts with the approximation ordering such that $\mathit{posint}, \mathit{zero}, \mathit{negint}$ are pairwise incompatible subsorts of the sort *int* (*i.e.*, $\mathit{posint} \wedge \mathit{zero} = \bot, \mathit{negint} \wedge \mathit{zero} = \bot, \mathit{posint} \wedge \mathit{negint} = \bot$). This is declared in LIFE as $\mathit{int} := \{\mathit{posint}; \mathit{zero}; \mathit{negint}\}$. Furthermore, we assume the sort definition $\mathit{posint} := \{\mathit{posodd}; \mathit{poseven}\}$; *i.e.*, *posodd* and *poseven* are subsorts of *posint* and mutually incompatible.

The LIFE query $Y = \mathit{minus}(X : \mathit{poseven})$? will return $Y = \mathit{negint}$. The sort *poseven* of the actual parameter is incompatible with the sort *negint* of the formal parameter of the first rule defining the function *minus*. Therefore, that rule is skipped. The sort *poseven* is more specific than the sort *posint* of the formal parameter of the second rule. Hence, that rule is applicable and yields the result $Y = \mathit{negint}$.

The LIFE query $Y = \mathit{minus}(X : \mathit{string})$ will fail. Indeed, the sort *string* is incompatible with the sort of the formal parameter of every rule defining *minus*.

Thus, in order to determine which of the rules, if any, defining the function in a given functional expression will be applied, two tests are necessary:

- verify whether the actual parameter is more specific than or equal to the formal parameter;
- verify whether the actual parameter is at all compatible with the formal parameter.

What happens if both of these tests fail? For example, consider the query consisting of the conjunction:

$$Y = \mathit{minus}(X : \mathit{int}), X = \mathit{minus}(\mathit{zero})?$$

Like Prolog, LIFE follows a left-to-right resolution strategy and examines the equation $Y = \mathit{minus}(X : \mathit{int})$ first. However, both foregoing tests fail and deciding which rule to use among those defining *minus* is inconclusive. Indeed, the sort *int* of the actual parameter in that call is neither more specific than, nor incompatible with, the sort *negint* of the first rule's formal parameter. Therefore, the function call will *residuate* on the variable X. This means that the functional evaluation is suspended pending more information on X. The second goal in the query is treated next. There, it is found that the actual parameter is incompatible with the first two rules and is the same as the last rule's. This allows reduction and binds X to *zero*. At this point, X has been instantiated and therefore the residual equation pending on X can be reexamined. Again, as before, a redex is found for the last rule and yields $Y = \mathit{zero}$.

The two tests above can in fact be worded in a more general setting. Viewing data structures as constraints, "more specific" is simply a particular case of constraint entailment. We will say that a constraint *disentails* another whenever their conjunction is unsatisfiable; or, equivalently, whenever it entails its negation. In particular, first-order matching is deciding entailment between constraints consisting of equations over first-order terms. Similarly, deciding unifiability of first-order terms amounts to deciding "compatibility" in the sense used informally above.

The suspension/resumption mechanism illustrated in our example is repeated each time a residuated actual parameter becomes more instantiated from the context; *i.e.*, through solving other parts of the query. Therefore, it is most beneficial for a practical algorithm testing entailment and disentailment to be incremental. This means that, upon resumption, the test for the instantiated actual parameter builds upon partial results obtained by the previous test. One outcome of the results presented in this paper is that it is possible to build such a test; namely, an algorithm deciding simultaneously two problems in an incremental manner—entailment and disentailment. The technique that we have devised to do that is called *relative simplification* of constraints.

This technique is relevant in the general framework of concurrent constraint logic programming, represented by, *e.g.*, the guarded Horn-clause scheme of Maher [20], Concurrent Constraint Programming (CCP) [23], and Kernel Andorra Prolog (KAP) [15]. These schemes are

parameterized with respect to an abstract class of constraint systems. An incremental test for entailment and disentailment between constraints is needed for advanced control mechanisms such as delaying, coroutining, synchronization, committed choice, and deep constraint propagation. LIFE is formally an instance of this scheme, namely a CLP language using a constraint system based on order-sorted feature (OSF) structures [6]. It employs a related, but limited, suspension strategy to enforce deterministic functional application. Roughly, these systems are concurrent thanks to a new effective discipline for procedure parameter-passing that we could describe as "call-by-constraint-entailment" (as opposed to Prolog's call-by-unification).

1.3 Organization of paper

We have organized the rest of this paper as follows. In Section 2, we cover informally the essence of LIFE that is relevant to functions and explain the gist of our approach. Reading only that section will provide a detailed intuition of the formal contents of the paper. It may be skipped altogether by the formally-minded reader who can travel through the technical details to follow without a road map. On the other hand, time spent there might reward the patient reader with a better sense of direction and hence a faster pace through later technicalities.

The remainder of the paper is technical. In Section 3, we recall the necessary formalism introduced in [6, 5] accounting for LIFE's structures and operations. It is meant to make this document self-contained. The reader already familiar with those notions could skip that section, although reading it will provide a timely summary.

The last four sections contain the formal details and rigorous justifications of the material presented informally in Section 2 and its relation to the semantics of ψ-terms and LIFE's operational semantics. First, in Section 4, we introduce the concept of relative simplification as a general proof-theoretic method for proving guards in concurrent constraint logic languages using guarded rules. Then, in Section 5, we explain residuation using relative simplification. Section 6 ties the operational semantics of function reduction with the semantics of ψ-terms as approximation structures. Finally, we conclude with Section 7, giving a brief recapitulation of the contribution of this paper and a few perspectives.

2 Synopsis

This section is an informal, albeit precise and detailed, overview of the main ideas. Using schematic examples, we explain the operational mechanism underlying functional reduction over order-sorted feature terms in the context of a logic programming framework. We recall the basic terminology and notation of LIFE, unification and matching, and we sketch the essence of relative simplification. Formal material rewording everything in rigorous terms will be exposed in the following sections.

2.1 LIFE data structures

The data objects of LIFE are ψ-terms. They are structures built out of sorts and features. ψ-Terms are partially ordered as data descriptions to reflect more specific information content. A ψ-term is said to *match* another one if it is a more specific description. For first-order terms, a matching substitution is a variable binding which makes the more general term equal to the more specific one. This notion is not appropriate here. Unification is introduced as taking the greatest lower bound (GLB) with respect to this ordering.

Sorts and features

Sorts are symbols. They are meant to denote sets of values. Here are a few examples: *person*, *int*, *true*, 3.5, $\bot$, $\top$. Note that a value is assimilated to a singleton sort. We call $\mathcal{S}$ the set of all sorts. They come with a partial ordering $\leq$, meant to reflect set inclusion.[2] For example,

- $\bot \leq$ *john* $\leq$ *man* $\leq$ *person* $\leq \top$;
- $\bot \leq$ *true* $\leq$ *bool* $\leq \top$;
- $\bot \leq 2 \leq$ *poseven* $\leq$ *int* $\leq \top$.

The sorts $\top$ (top) and $\bot$ (bottom) are respectively the greatest and the least sort in $\mathcal{S}$ and denote respectively the whole domain of interpretation and the empty set.

Sorts also come with a GLB operation $\wedge$. For example,

- *person* $\wedge$ *male* $=$ *man*;
- *male* $\wedge$ *female* $=$ *hermaphrodite*;
- *man* $\wedge$ *woman* $= \bot$;

etc., which can be visualized as shown in Figure 1. We will refer back to this figure in several examples to come.

Features (or attribute labels) are also symbols and used to build ψ-terms by attaching attributes to sorts. The set of feature symbols is called $\mathcal{F}$. We will use words and natural numbers as features. The latter are handy to specify attributes by positions as subterms in first-order terms. Examples of feature symbols are *age*, *spouse*, 1, 2.

ψ-Terms

Basic ψ-terms are the simplest form of ψ-terms. They are:

- variables; *e.g.*, $X, Y, Z, \ldots$
- sorts; *e.g.*, *person*, *int*, *true*, $3.5, \top, \ldots$
- tagged sorts; *e.g.*, $X : \top,\ Y :$ *person*, $\ldots$

Stand-alone variables are always implicitly sorted by $\top$, and stand-alone sorts are always implicitly tagged by some variable occurring nowhere else. Thus, one might say that a basic

[2] Sorts and their relative ordering are specified by the user.

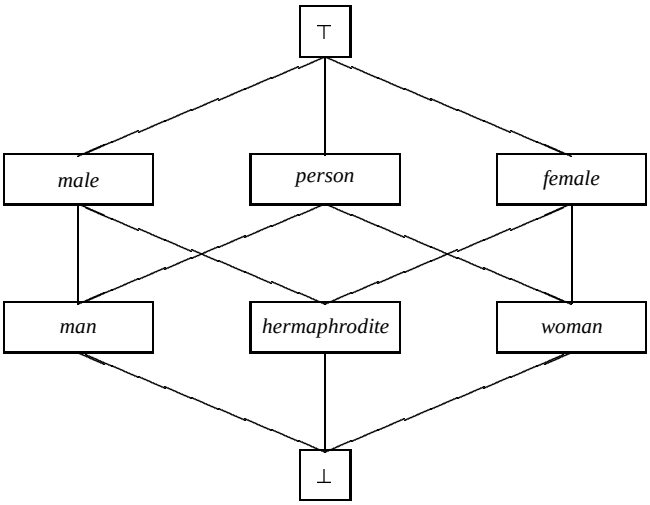

Figure 1. A partial order of sorts

ψ-term is always of the form *variable* : *sort*.

Features are used to build up more complex ψ-terms. Thus, the following ψ-term is obtained from the ψ-term *person* by attaching the feature *age* typed by the ψ-term *int*:[3]

$$X : person(age \Rightarrow I : int).$$

The sort at the root of a ψ-term, here *person*, is called its *principal sort*. A ψ-term can be seen as a record structure. Features correspond to field identifiers, and fields are, in turn, associated to ψ-terms. These are flexible records in the sense that variably many fields may be attached to the principal sort. For example, we can augment the ψ-term above with another feature:

$$X : person(age \Rightarrow I : int,\\ spouse \Rightarrow Y : person(age \Rightarrow J : int)).$$

This ψ-term denotes the set of all objects X of sort *person* (in the intended domain), whose value I under the function *age* is of sort *int*, whose value Y under the function *spouse* is of sort *person*, and the value J of Y under the function *age* is of sort *int*.

The following ψ-term is more specific, in the sense that the above set becomes smaller if one further requires that the values I and J coincide; namely, $age(X) = age(spouse(X))$:

$$X : person(age \Rightarrow I : int,\\ spouse \Rightarrow Y : person(age \Rightarrow I)).$$

[3]To illustrate the ψ-term ordering, we will give a decreasing matching sequence of ψ-terms going from more general to more specific ones.

It denotes the subset of individuals in the previous set of *person*'s whose age is the same as their spouse's. This ψ-term uses a coreference thanks to sharing the variable I. The next ψ-term is even more specific, since it contains an additional (circular) coreference; namely, $X = spouse(spouse(X))$:

$$X : person(age \Rightarrow I : int, \\ spouse \Rightarrow Y : person(age \Rightarrow I, \\ spouse \Rightarrow X)).$$

It denotes the set of all individuals in the previous set whose spouse's spouse is the individual in question. Note that only variables that are used as coreference tags need to be put explicitly; *i.e.,* those that occur at least twice.

To be well-formed, the syntax of a ψ-term requires three conditions to be satisfied: (1) the sort $\bot$ may not occur; (2) at most one occurrence of each variable has a sort; (3) all the features attached to a sort are pairwise different. These conditions are necessary to ensure that a ψ-term expresses coherent information. For example, $X : man(friend \Rightarrow X : woman)$, violating Condition (2), is not a ψ-term, but $X : man(friend \Rightarrow X)$ is.

As for ordering, a ψ-term is made more specific through:

- sort refinement; *e.g.*, $X : int \leq U : \top$;
- adding features typed by ψ-terms; *e.g.*, $X : \top(age \Rightarrow int) \leq U : \top$;
- adding coreference; *e.g.*, $X : \top(likes \Rightarrow X) \leq U : \top(likes \Rightarrow V)$.

Note that, as record structures, ψ-terms are both record types and record instances. In addition, they allow *mixing* type and value information. Finally, they also permit constraining records with equations on their parts.

ψ-Terms as graphs

There is a straightforward representation of a ψ-term as a rooted directed graph. Let us assume that every variable is explicitly sorted (if necessary, by the sort $\top$) and every sort is explicitly tagged (if necessary, by a single-occurrence variable). The nodes of the graph are the variables, their labels are the corresponding sorts; for every feature mapping one variable X to another one Y there is an arc (X, Y) labeled by that feature. One node is marked as the root (whose label is called the root sort or the principal sort of the ψ-term).

For example, the ψ-term:

$$X_1 : person(name \Rightarrow X_2 : id(first \Rightarrow X_3 : string, \\ last \Rightarrow X_4 : string), \\ spouse \Rightarrow X_5 : person(name \Rightarrow X_6 : id(last \Rightarrow X_4), \\ spouse \Rightarrow X_1)).$$

corresponds to the OSF-graph shown in Figure 2.

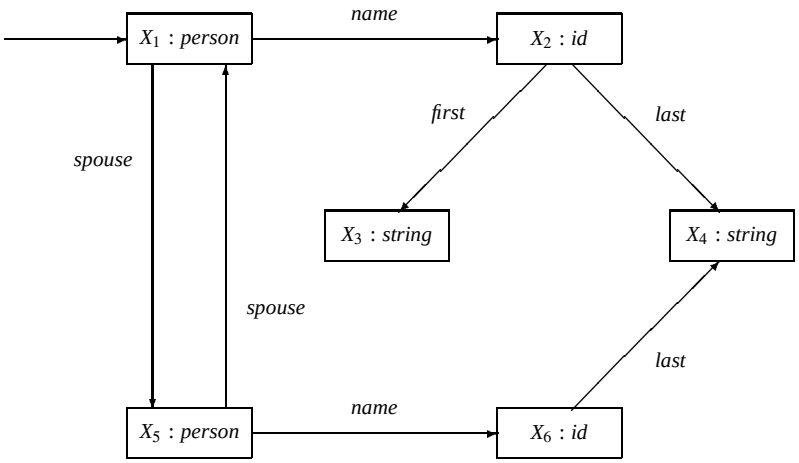

Figure 2. An OSF-Graph

ψ-Terms as values

One particular interpretation is readily available for ψ-terms. Namely, the syntactic interpretation whose domain is the set of all ψ-terms. Note that ψ-terms have a dual personality. They are syntactic objects (graphs) representing the values of the domain of , *and* they also are types which denote sets. In the particular case of the interpretation , they denote subsets of the domain of ; *i.e.*, sets of ψ-terms. We shall see this dual view does not lead to paradox, *au contraire.*

In the interpretation , a sort $s \in \mathcal{S}$ denotes the set of all ψ-terms whose root sort is a subsort of s. A feature $\ell \in \mathcal{F}$ denotes the function mapping a ψ-term to its sub-ψ-term under that feature, or to $\top$, if there is none.

Thus, a sort denotes the set of all ψ-term values which, as ψ-term types, are more specific than the basic ψ-term s. In fact, it is possible to show that in general a ψ-term denotes the set of all ψ-terms which are more specific than the ψ-term itself. This is the "ψ-terms as filters" principle established in [5]. It yields directly the fact that the partial ordering $\leq$ on ψ-terms is exactly set-inclusion of the sets denoted by the ψ-terms in the ψ-term domain.

Feature trees as values

We obtain two other examples of OSF-algebras when we "compress" the ψ-term domain by identifying values. In a first step, we say that two ψ-terms which are equal up to variable renaming represent the same value of the domain, or: two isomorphic graphs are identified. We call the OSF-algebra hereby obtained $_0$.

It is well known that a rooted directed graph represents a unique rational tree obtained by

unfolding. Hence, unfolding an OSF-graph yields what we call a feature tree. Such a tree is one whose nodes are labeled with sorts and whose edges are labeled with features. Therefore, we can also identify ψ-terms which represent the same rational tree. The domain hereby obtained is essentially the feature tree structure $\mathcal{T}$ introduced first in [7] and [8].

Unification of ψ-terms

We say that ψ_1 is unifiable with ψ_2 if $\psi_1 \wedge \psi_2 \neq \bot$; *i.e.*, if there exist ψ-terms with non-empty denotations which are more specific than both ψ_1 and ψ_2. Then, one can show that there exists a unique (up to variable renaming) ψ-term ψ which is the most general of all these, the 'greatest lower bound' (GLB) of ψ_1 and ψ_2, written $\psi = \psi_1 \wedge \psi_2$.

For the set denotation of ψ-terms, $\wedge$ is exactly set intersection. An important result illustrating the significance of the ψ-term interpretation is that ψ_1 is unifiable with ψ_2 if and only if the intersection of the two sets denoted by ψ_1 and ψ_2 in the ψ-term domain is non-empty.

Constraints and ψ-terms

We also view a ψ-term logically as a constraint formula by flattening it into what we call its *dissolved form*. For ease of notation, we shall write $(X : \psi)$ to indicate that the root variable of the ψ-term ψ is X.

More precisely, the ψ-term $X : s(\ell_1 \Rightarrow (X_1 : \psi_1), \ldots, \ell_n \Rightarrow (X_n : \psi_n))$ corresponds to the conjunction of the constraint $X : s \;\&\; X.\ell_1 \doteq X_1 \;\&\; X.\ell_n \doteq X_n$ and of the constraints corresponding to $\psi_1, \ldots, \psi_n$. A basic ψ-term $X : s$ corresponds to the sort constraint $X : s$. For example, the ψ-term:

$$\begin{aligned}\psi \;\equiv\; & X : \mathit{person}(\mathit{likes} \Rightarrow X,\\ & \qquad \mathit{age} \Rightarrow Y : \mathit{int})\end{aligned}$$

is identified with the constraint:

$$\psi \;\equiv\; X : \mathit{person} \;\&\; X.\mathit{age} \doteq Y \;\&\; Y : \mathit{int} \;\&\; X.\mathit{likes} \doteq X.$$

Thus, the constraint ψ is a conjunction of atomic sort constraints of the form $X : s$ and atomic feature constraints of the form $X.\ell \doteq Y$. The interpretation of the sort and feature constraints over the intended domain is straightforward, given that sorts are interpreted as subsets of the domain and features as unary functions over the domain.

A value lies in the set denoted by the ψ-term ψ in an interpretation $\mathcal{I}$ if and only if the constraint $X \doteq Z \;\&\; \psi$ is satisfiable in the interpretation $\mathcal{I}$, with that value assigned to the variable X, and Z being the root variable of ψ. All variables of ψ are implicitly existentially quantified. This reflects our view of ψ-terms as set-denoting types.

Rules for unification

Unifying $(X_1 : \psi_1)$ and $(X_2 : \psi_2)$ amounts to deciding satisfiability of the conjunction $\psi_1 \;\&\; \psi_2 \;\&\; X_1 \doteq X_2$. Thus, the unification algorithm can be specified in terms of constraint normalization rules. A constraint containing the conjunction over the line is rewritten into an *equivalent* constraint by replacing this conjunction by the constraint under the line. We only need four rules that are illustrated schematically on an example below. (Refer to the sorts of Figure 1.)

Equality:

$$\frac{\ldots\ X : \mathit{person} \quad \& \quad U : \mathit{male} \quad \& \quad U \doteq X\ \ldots}{\ldots\ X : \mathit{person} \quad \& \quad X : \mathit{male} \quad \& \quad U \doteq X\ \ldots}$$

Sorts:

$$\frac{\ldots\ X : \mathit{person} \quad \& \quad X : \mathit{male}\ \ldots}{\ldots\ X : \mathit{man}\ \ldots}$$

Features:

$$\frac{\ldots\ X.\mathit{likes} \doteq Y \quad \& \quad X.\mathit{likes} \doteq V\ \ldots}{\ldots\ X.\mathit{likes} \doteq Y \quad \& \quad V \doteq Y\ \ldots}$$

Clash:

$$\frac{\ldots\ X : \bot\ \ldots}{\bot}$$

One can show that a constraint is satisfiable if and only if it is normalized to a constraint different from the *false* constraint $\bot$. If we identify every constraint containing a sort constraint of the form $X : \bot$ with the *false* constraint, we omit the clash rule.

In particular, the ψ-terms $(X_1 : \psi_1)$ and $(X_2 : \psi_2)$ are unifiable if and only if $\psi_1 \;\&\; \psi_2 \;\&\; X_1 \doteq X_2$ is normalized into a constraint ψ different from $\bot$. This constraint ψ corresponds, apart from its equalities (between variables), to the ψ-term (unique up to variable renaming) $\psi_1 \wedge \psi_2$.

2.2 Relative simplification

We use the framework of first-order logic to transform the combined entailment/disentailment problem into one that can be solved by the relative simplification algorithm.

Matching and entailment

In the remainder of this paper, when considering the matching problem $\psi_1 \leq \psi_2$, we will refer to ψ_1 as the actual parameter and its variables (named $X, Y, Z, \ldots$) as global, and to ψ_2 as the formal parameter and its variables (named $U, V, W, \ldots$) as local.

In the Concurrent Constraint Logic Programming framework, the matching problem generalizes to the entailment problem; namely, whether the actual constraint, also called context, entails the formal constraint, also called guard [20, 23].

First observe that, for example, the first-order term $t_1 = f(Z, f(Y, Y))$ matches the term $t_2 = f(W, V)$, and that the implication:

$$\forall X \forall Y \forall Z \ \big(X \doteq f(Z, f(Y, Y)) \rightarrow \exists U \exists V \exists W \ (X \doteq U \ \& \ U \doteq f(W, V)) \big)$$

is valid. Generally, the term t_1 matches t_2 (noted $t_1 \leq t_2$) if and only the implication $X \doteq t_1 \rightarrow \exists U \exists V \ (X \doteq U \ \& \ U \doteq t_2)$ is valid, where V stands for all variables of t_2. More shortly, $X \doteq t_1$ entails $X \doteq U \ \& \ U \doteq t_2$.

Note, however, that there is an essential difference between ψ-term matching and first-order term matching. For example, the term $f(a, a)$ matches the term $f(V, V)$. This is true because first order terms denote individuals. This is no longer true in LIFE. For example, the ψ-term $X : f(1 \Rightarrow Y : \mathit{int}, 2 \Rightarrow Z : \mathit{int})$ does not match the ψ-term $U : f(1 \Rightarrow V, 2 \Rightarrow V)$. Indeed, the presence of two occurrences of the same sort does not entail that the individuals in that sort be equal. Therefore, $X : s(1 \Rightarrow \psi_1, 2 \Rightarrow \psi_2)$ is less specific than the ψ-term $U : s(1 \Rightarrow V, 2 \Rightarrow V)$ only if the root variables of ψ_1 and ψ_2 are identical (or bound together).

This does *not* mean that values and operations on them are not available in LIFE.[4] What the above point illustrates is that to recognize that a sort is a fully determined value, and hence to enforce identity of all its distinct occurrences, one needs this information declared explicitly, in effect adding an axiom to the formalization of such sorts. So-declared *extensional* sorts can then be treated accordingly thanks to an additional inference rule (being a minimal non-bottom sort is not sufficient). Without this rule, however, equality of distinct occurrences cannot be entailed and the behavior illustrated is the only correct one. The point of this paper being independent if this issue, we shall omit this additional rule.

The fact that $(X : \psi_1) \leq (U : \psi_2)$, *i.e.*, the ψ-term $(X : \psi_1)$ matches the ψ-term $(U : \psi_2)$, translates into the fact that the corresponding constraint ψ_1 *entails* the constraint $\psi_2 \ \& \ U \doteq X$. This means that the implication $\psi_1 \rightarrow \exists U, V, W \ldots \ \psi_2 \ \& \ U \doteq X$ is valid. Here, $\exists U, V, W \ldots$ indicates that all local variables are existentially quantified. The global variables are universally quantified.

Entailment of general constraints

We will now give a precise explanation of a fact which is well-known for constructor terms. An actual parameter t_1 matches a formal parameter t_2 if and only if the unification of the two terms binds only variables of t_2, but no variable of t_1. In other words, only local, but no global, variables are instantiated.

[4] Of course, one can use actual values of sort *int*, *real*, or *string* in expressions with their usual operations as in most programming languages. In fact, LIFE provides the additional freedom to write such expressions mixing actual values or their sort approximations *int*, *real*, or *string*. Such expressions are either solved by local propagation or *residuate* pending further refinements of the non-value sorts into values.

The unification of the term $t_1 = f(Z, f(Y, Y))$ and the term $t_2 = f(W, V)$ yields the variable bindings $W \doteq Z$ and $V \doteq f(Y, Y)$. On the other hand, the conjunction:

$$X \doteq f(Z, f(Y, Y)) \;\&\; U \doteq X \;\&\; U \doteq f(W, V)$$

is equivalent to:

$$X \doteq f(Z, f(Y, Y)) \;\&\; (\; U \doteq X \;\&\; V \doteq f(Y, Y) \;\&\; W \doteq Z \;),$$

and the last part of this conjunction is valid if the local variables U, V, W are existentially quantified.

This is the general principle which underlies the relative simplification algorithm. Namely, the actual constraint ψ_1 entails $\psi_2 \;\&\; U \doteq X$ if and only if the following holds. Their conjunction $\psi_1 \;\&\; \psi_2 \;\&\; U \doteq X$ is equivalent to the conjunction $\psi_1 \;\&\; \psi_2'$ of the actual constraint ψ_1 and a constraint ψ_2' which is valid if existentially quantified over the local variables. In our case, ψ_2' will be a conjunction of equalities binding local to global variables. Formally,

$$\models \psi_1 \rightarrow \exists U, V, W, \ldots \; \psi_2 \;\&\; U \doteq X$$

if and only if there exists a formula ψ_2' such that:

$$\models (\psi_1 \;\&\; \psi_2 \;\&\; U \doteq X) \leftrightarrow (\psi_1 \;\&\; \psi_2') \;\text{ and }\; \models \exists U, V, W \ldots \; \psi_2'.$$

This statement is correct since validity of the implication $\psi_1 \rightarrow \exists U \; \psi_2 \;\&\; U \doteq X$ is the same as the validity of the equivalence $(\psi_1 \;\&\; (\exists U \; \psi_2 \;\&\; U \doteq X)) \leftrightarrow \psi_1$. This fact is analogous to the fact that a set is the subset of another one if and only if it is equal to the intersection of the two. The condition $\models \exists U, V, W \ldots \; \psi_2'$ in the statement expresses that $\psi_1 \;\&\; (\exists U, V, W, \ldots \; \psi_2')$ is equivalent to ψ_1.

Towards relative simplification

Operationally, in order to show that $(X : \psi_1) \leq (U : \psi_2)$ holds, it is sufficient to show that the conjunction $\psi_1 \;\&\; \psi_2 \;\&\; U \doteq X$ is equivalent to $\psi_1 \;\&\; \psi_2'$, where ψ_2' is some constraint which, existentially quantified over the variables of ψ_2, is valid. In our case, again, ψ_2' will be a conjunction of equalities binding variables of ψ_2 to variables of ψ_1.

Therefore, in order to test $(X : \psi_1) \leq (U : \psi_2)$, we will apply successively the unification rules on the constraint $\psi_1 \;\&\; \psi_2 \;\&\; U \doteq X$ if they do not modify ψ_1. We obtain three kinds of transformations which are illustrated schematically below. (Refer to the sorts of Figure 1.)

Equality:

$$\frac{\ldots \; X \doteq Y \quad \& \quad U \doteq X \; \ldots}{\ldots \; X \doteq Y \quad \& \quad U \doteq Y \; \ldots}$$

Sorts:

$$\frac{\ldots\ X : man \ \ \&\ \ U \doteq X \ \ \&\ \ U : person\ \ldots}{\ldots\ X : man \ \ \&\ \ U \doteq X\ \ldots}$$

Features:

$$\frac{\ldots\ X.likes \doteq Y \ \ \&\ \ U \doteq X \ \ \&\ \ U.likes \doteq V\ \ldots}{\ldots\ X.likes \doteq Y \ \ \&\ \ U \doteq X \ \ \&\ \ V \doteq Y\ \ldots}$$

The equality rule is derived from the corresponding unification rule, which has to be restricted to modify only the formal constraint. If the actual constraint contains an equality between two global variables, then one of them may be eliminated for the other. A global variable is never eliminated for a local one.

The sort rule corresponds to two applications of unification rules, first the elimination of the local by the global variable, and then the reduction of two sort constraints on the same variable (here $X : man$ & $X : person$) to one sort constraint (namely $X : man \wedge person$). Clearly, if the "global sort" is a subsort of the "local sort" then this application does not modify the global constraint. The feature rule works quite similarly.

For example, the rules above can be used to show that the ψ-term:

$$\psi_1 \equiv X : man\big(likes \Rightarrow Y : person, age \Rightarrow I : int\big)$$

matches the ψ-term:

$$\psi_2 \equiv U : person\big(likes \Rightarrow V\big).$$

Namely, the constraint ψ_1 & ψ_2 & $U \doteq X$:

$$\begin{array}{lll} & X : man & \& \ X.likes \doteq Y \ \& \ Y : person \ \& \ X.age \doteq I \ \& \ I : int \\ \& & U : person & \& \ U.likes \doteq V \\ \& & U \doteq X & \end{array}$$

is normalized into:

$$\begin{array}{lll} & X : man & \& \ X.likes \doteq Y \ \& \ Y : person \ \& \ X.age \doteq I \ \& \ I : int \\ \& & V \doteq Y & \& \ U \doteq X; \end{array}$$

that is,

$$\psi_1 \ \& \ V \doteq Y \ \& \ U \doteq X.$$

Clearly, $\exists U \exists V \ \big(V \doteq Y \ \& \ U \doteq X\big)$ is valid. Therefore, the constraint ψ_1 entails the constraint ψ_2 & $U \doteq X$.

Relative simplification for entailment

The rules above are such that $\psi_1 \;\&\; \psi$ rewrites to $\psi_1 \;\&\; \psi'$; *i.e.*, the global constraint ψ_1 is not modified by the simplification. In this case, we say that the constraint ψ simplifies to ψ' relatively to the actual constraint ψ_1. In other words, ψ_1 acts as a context relatively to which simplification of ψ is carried out. In general, this context formula may be any formula. Hence, we can reformulate the rules above as relative-simplification rules. We use the notation $\frac{\psi}{\psi'}\ [\phi]$ to mean that ψ is simplified into ψ' relatively to the context formula ϕ. Schematically,

Equality:

$$\frac{\ldots\ U \doteq X\ \ldots}{\ldots\ U \doteq Y\ \ldots}\ [\ \ldots\ X \doteq Y\ \ldots\ ]$$

Sorts:

$$\frac{\ldots\ U \doteq X \;\&\; U : \mathit{person}\ \ldots}{\ldots\ U \doteq X \;\&\;\ \ldots}\ [\ \ldots\ X : \mathit{man}\ \ldots\ ]$$

Features:

$$\frac{\ldots\ U \doteq X \;\&\; U.\mathit{likes} \doteq V\ \ldots}{\ldots\ U \doteq X \;\&\; V \doteq Y\ \ldots}\ [\ \ldots\ X.\mathit{likes} \doteq Y\ \ldots\ ]$$

Using these rules, the constraint $\psi_2 \equiv U \doteq X \;\&\; U : \mathit{person} \;\&\; U.\mathit{likes} \doteq V$ in the previous example simplifies to $\psi_2' \equiv U \doteq X \;\&\; V \doteq Y$ relatively to:

$\psi_1 \equiv X : \mathit{man} \;\&\; X.\mathit{likes} \doteq Y \;\&\; Y : \mathit{person} \;\&\; X.\mathit{age} \doteq I \;\&\; I : \mathit{int}.$

Invariance of relative simplification is the following property. If ψ simplifies to ψ' relatively to ϕ, then the conjunction of ψ with ϕ is equivalent to the conjunction of ψ' with ϕ.

This invariance justifies the correctness of the relative simplification algorithm with respect to entailment. Namely, if ψ simplifies to ψ' relatively to ϕ, and if ψ' consists only of equations binding local variables, then ϕ entails ψ.

Proof of completeness of the algorithm needs the assumption that the set $\mathcal{F}$ of features is infinite. Note that exactly thanks to the infiniteness of $\mathcal{F}$ our framework accounts for flexible records; *i.e.*, the indefinite capacity of adding fields to records.

Relative simplification for disentailment

If the result of the matching test $\psi_1 \leq \psi_2$ is negative, *i.e.*, the actual constraint does not entail the formal constraint, then we must know more; namely, whether the two terms are non-unifiable. Non-unifiability is equivalent to the fact that the actual parameter will not

match the formal one even when further instantiated; *e.g.*, when further constraints are attached as conjuncts. Logically, this amounts to saying that a context formula ϕ *disentails* a guard constraint ψ if and only if the conjunction $\phi \;\&\; \psi$ is unsatisfiable. In terms of relative simplification, ϕ disentails ψ if and only if ψ simplifies to the *false* constraint $\perp$ relatively to ϕ.

For example, $X : \mathit{male}$ is non-unifiable with $U : \mathit{woman}$.[5] The constraint $U : \mathit{woman} \;\&\; U \doteq X$ simplifies to $\perp$ relatively to the constraint $X : \mathit{male}$, since $\mathit{woman} \wedge \mathit{male} = \perp$, using a rule of the form indicated below, and then the Clash rule.

Sorts:

$$\frac{\ldots\ U \doteq X\ \&\ U : \mathit{woman}\ \ldots}{\ldots\ U \doteq X\ \&\ U : \mathit{woman} \wedge \mathit{male}\ \ldots}\ [\ \ldots X : \mathit{male}\ \ldots\ ]$$

The following example shows that a sort clash cannot always be detected by comparing sorts in the formal constraint one by one with sorts in the actual constraint; *i.e.*, one needs several steps with intermediate sort intersections.

The ψ-term $Z : \top(\mathit{likes} \Rightarrow X : \mathit{male}, \mathit{friend} \Rightarrow Y : \mathit{female})$ is non-unifiable with the ψ-term $W : \top(\mathit{likes} \Rightarrow U : \mathit{person}, \mathit{friend} \Rightarrow U)$. The constraint $\phi \equiv X : \mathit{male}\ \&\ Y : \mathit{female}$ disentails the constraint $\psi \equiv U \doteq X\ \&\ U \doteq Y\ \&\ U : \mathit{person}$. Operationally, the constraint ψ simplifies to $\perp$ relatively to the context ϕ. Here are the steps needed to determine this:

$$\frac{\begin{array}{c}\ldots\ U \doteq X\ \&\ U \doteq Y\ \&\ U : \mathit{person}\ \ldots \\ \hline \ldots\ U \doteq X\ \&\ U \doteq Y\ \&\ U : \mathit{person} \wedge \mathit{male}\ \ldots \\ \hline \ldots\ U \doteq X\ \&\ U \doteq Y\ \&\ U : \mathit{man} \wedge \mathit{female}\ \ldots\end{array}}{\perp}$$

There is an issue regarding the enforcing of functionality of features in the simplification of a constraint ψ relatively to a context ϕ. This may be explained as follows. Let us suppose that two global variables X and Y become bound to the same local variable U. Then,

- the context ϕ entails the constraint ψ only if ϕ contains $X \doteq Y$; and,
- the context ϕ disentails the constraint ψ if the same path of features starting from X and Y, respectively, leads to variables X' and Y', respectively, whose sorts are incompatible.

There are essentially two cases, depending on whether a new local variable has to be introduced or not. Each case is illustrated in the next two examples.

The ψ-term:[6]

$$\begin{array}{rl}\phi\ \equiv\ & Z : \top(\mathit{likes} \Rightarrow X : \top(\mathit{age} \Rightarrow I_1 : \mathit{poseven}), \\ & \quad \mathit{friend} \Rightarrow Y : \top(\mathit{age} \Rightarrow I_2 : \mathit{posodd}))\end{array}$$

[5] Refer to the sorts of Figure 1.

[6] We assume that $\mathit{poseven} \wedge \mathit{posodd} = \perp$.

is non-unifiable with the ψ-term:

$$\psi \equiv W : \top(\mathit{likes} \Rightarrow U, \mathit{friend} \Rightarrow U)$$

That is, the constraint ϕ disentails the constraint ψ. Operationally, with the context ϕ, the constraint ψ simplifies, in a first step, to:

$$W \doteq Z \;\&\; U \doteq X \;\&\; U \doteq Y.$$

Then, using the rule:

$$\frac{\ldots\ U \doteq X \;\&\; U \doteq Y\ \ldots}{\ldots\ U \doteq X \;\&\; U \doteq Y \;\&\; J \doteq I_1 \;\&\; J \doteq I_2\ \ldots}\ [\ \ldots\ X.\mathit{age} \doteq I_1 \;\&\; Y.\mathit{age} \doteq I_2\ \ldots\ ]$$

where J is a new variable, to:

$$W \doteq Z \;\&\; U \doteq X \;\&\; U \doteq Y \;\&\; J \doteq I_1 \;\&\; J \doteq I_2$$

and finally to $\bot$, since the sorts of I_1 and I_2 (*poseven* and *posodd*) are incompatible.

The rules enforce the following property: a global variable is never bound to more than one local variable. Therefore, if the variable X or the variable Y is already bound to a local variable, *no* new local variable must be introduced. This is illustrated by the second example.

The ψ-term:

$$\phi \equiv Z : \top(\mathit{likes} \Rightarrow X : \top(\mathit{age} \Rightarrow I_1 : \mathit{poseven}), \mathit{friend} \Rightarrow Y : \top(\mathit{age} \Rightarrow I_2 : \mathit{posodd}), \mathit{age} \Rightarrow I_1)$$

is non-unifiable with the ψ-term:

$$\psi \equiv W : \top(\mathit{likes} \Rightarrow U, \mathit{friend} \Rightarrow U(\mathit{age} \Rightarrow J), \mathit{age} \Rightarrow J).$$

Operationally, with the context ϕ, the constraint ψ simplifies, in a first step, to:

$$W \doteq Z \;\&\; U \doteq X \;\&\; U \doteq Y \;\&\; J \doteq I_1.$$

Then, using the rule:

$$\frac{\ldots\ U \doteq X\ \&\ U \doteq Y\ \&\ J \doteq I_1\ \ldots}{\ldots\ U \doteq X\ \&\ U \doteq Y\ \&\ J \doteq I_1\ \&\ J \doteq I_2\ \ldots}\ [\ \ldots\ X.age \doteq I_1\ \&\ Y.age \doteq I_2\ \ldots\ ]$$

where J is a new variable, to:

$$W \doteq Z\ \&\ U \doteq X\ \&\ U \doteq Y\ \&\ J \doteq I_1\ \&\ J \doteq I_2$$

and finally to $\perp$, for the same reason as above.

In order to be complete with respect to disentailment, the algorithm must keep track of all pairs of variables $(X, Y), \ldots, (X', Y')$ whose equality is induced by the binding of X and Y to the same local variable. That is, it must propagate equalities along features. In our presentation, it will be conceptually sufficient to refer explicitly to the actual equalities binding the global variables to a common local variable. Practically, this can of course be done more efficiently.

Specifying the relative simplification algorithm

If $\psi\ \&\ U \doteq X$ simplifies to ψ' relatively to ϕ and no relative-simplification rule can be applied further, then:

- ϕ entails $\psi\ \&\ U \doteq X$; formally,

  $\models \phi \rightarrow \exists U, V, W \ldots (\psi\ \&\ U \doteq X)$,

  if and only if ψ', with the variables of ψ existentially quantified, is valid; formally:

  $\models \exists U, V, W \ldots\ \psi'$.

- ϕ disentails $\psi\ \&\ U \doteq X$; formally:

  $\models \phi \rightarrow \neg\exists U, V, W \ldots (\psi\ \&\ U \doteq X)$,

  if and only if $\psi' = \perp$.

This test is *incremental*. Namely, every relative simplification of the constraint ψ to some constraint ψ' relatively to the context ϕ is also a relative simplification relatively to an incremented context $\phi\ \&\ \phi'$, for any constraint ϕ'.

Recapitulating, our original goal was a simultaneous test of matching and non-unifiability for two given ψ-terms ψ_1 and ψ_2. This test was recast as a test of entailment and disentailment for the constraints to which the ψ-terms dissolve. Namely, if X and U are the root variables of ψ_1 and ψ_2, respectively, the test whether ψ_1 entails or disentails $\psi_2\ \&\ U \doteq X$.

In our setting, the entailment test succeeds if and only if ψ_2' is a conjunction of matching equations; *i.e.*, of the form $\psi_2' \equiv U \doteq X\ \&\ V \doteq Y\ \&\ W \doteq Z \ldots$, where the local variables U, V, W, $\ldots$ are all different.

3 Background

We introduce briefly the notions that we have used informally in Section 2. For a thorough investigation of these notions, the reader is referred to [6, 5].

We start with the notion of OSF-algebras. They are the semantic structures interpreting complex data objects built out of features and partially-ordered sorts. Mathematically, an OSF-algebra formalizes access into the parts making up a piece of datum as well as their categorization. We then introduce OSF-constraints. They are important since, although they are formal objects which are part of a logical formalism, they are also quite primitive to constitute a low-level implementation logic.[7] We then formalize ψ-terms as they not only constitute a syntactically pleasant and convenient surface language for data objects in LIFE, but also comprise a syntactic OSF-algebra. Namely, they are representations of values of the domain of the standard interpretation. Finally, we summarize a few facts about this formalism that are relevant as related to the global contents of the paper.

3.1 OSF-algebras and OSF-constraints

The building blocks of OSF-algebras are sorts and features.

An *order-sorted feature signature* (or simply OSF-signature) is a tuple $\langle \mathcal{S}, \leq, \wedge, \mathcal{F} \rangle$ such that:

- $\mathcal{S}$ is a set of *sorts* containing the sorts $\top$ and $\bot$;
- $\leq$ is a decidable partial order on $\mathcal{S}$ such that $\bot$ 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 sorts s and s');
- $\mathcal{F}$ is a set of *feature symbols*.

An OSF-signature has the following interpretation. An *OSF-algebra* over the signature $\langle \mathcal{S}, \leq, \wedge, \mathcal{F} \rangle$ is a structure:

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

such that:

- $D^{\mathcal{A}}$ is a non-empty set, called the *domain* of $\mathcal{A}$ (or, universe);
- 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 $\bot^{\mathcal{A}} = \emptyset$;
- the greatest lower bound (*GLB*) operation on the sorts is interpreted as the intersection; *i.e.*, $(s \wedge s')^{\mathcal{A}} = s^{\mathcal{A}} \cap s'^{\mathcal{A}}$ for two sorts s and s' in $\mathcal{S}$.
- for each feature ℓ in $\mathcal{F}$, $\ell^{\mathcal{A}}$ is a total unary function from the domain into the domain; *i.e.*, $\ell^{\mathcal{A}} : D^{\mathcal{A}} \mapsto D^{\mathcal{A}}$;

[7] In fact, the reader familiar with implementation techniques of Prolog [2] should recognize that they are of the exact same granularity as WAM term representation and instructions.

The notion of OSF-algebra calls naturally for a corresponding notion of homomorphism preserving structure appropriately. Namely,

Definition 1 (OSF-Homomorphism) *An OSF-algebra 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:*

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

It is straightforward to verify that OSF-algebras together with OSF-homomorphisms form a category. We call this category OSF.

Let $\mathcal{V}$ be a countably infinite set of variables.

Definition 2 (OSF-Constraint) *An atomic OSF-constraint is one of:*

- $X : s$,
- $X \doteq X'$,
- $X.\ell \doteq X'$,

where X and X' are variables in $\mathcal{V}$, s is a sort in $\mathcal{S}$, and ℓ is a feature in $\mathcal{F}$. An OSF-constraint is a conjunction of atomic OSF-constraints.

One reads the three forms of atomic OSF-constraints as, respectively, "X lies in sort s," "X is equal to X'," and "X' is the feature ℓ of X." The set $Var(\phi)$ of variables occurring in an OSF-constraint ϕ is defined in the standard way. OSF-constraints will always be considered equal if they are equal modulo the commutativity, associativity and idempotence of conjunction "&." Therefore, a constraint can also be formalized as the set consisting of its conjuncts. As usual, the empty conjunction corresponds to the propositional constant interpreted as *true*.

Let $\mathcal{A}$ be an OSF-algebra. We call $Val(\mathcal{A}) = \{\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}\}$ the set of all possible valuations in the interpretation $\mathcal{A}$. The semantics of OSF-constraints is straightforward.

Given $\mathcal{A}$ is OSF-algebra, an OSF-constraint ϕ is *satisfiable* in $\mathcal{A}$, if there exists a valuation $\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}$ such that $\mathcal{A}, \alpha \models \phi$, 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 $\alpha(X) = \alpha(Y)$;
- $\mathcal{A}, \alpha \models X.\ell \doteq Y$ if and only if $\ell^{\mathcal{A}}(\alpha(X)) = \alpha(Y)$;
- $\mathcal{A}, \alpha \models \phi \,\&\, \phi'$ if and only if $\mathcal{A}, \alpha \models \phi$ and $\mathcal{A}, \alpha \models \phi'$.

3.2 ψ-Terms

We now introduce the syntactic objects that we intend to use as expressions of approximate descriptions to be interpreted as subsets of the domain of an OSF-algebra. Later, we will use them as well as representations of values constituting the domain of a specific interpretation.

Definition 3 (ψ-Term) *A ψ-term ψ is an expression of the form:*

$$X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n)$$

where

- *X is a variable in $\mathcal{V}$ called the root of ψ;*
- *s is a sort different from $\bot$ in $\mathcal{S}$;*
- *$\ell_1, \ldots, \ell_n$ are pairwise different features in $\mathcal{F}$, $n \geq 0$;*
- *$\psi_1, \ldots, \psi_n$ are again ψ-terms; and,*
- *no variable Y occurring in ψ is the root variable of more than one non-trivial ψ-term (i.e., different than $Y : \top$).*

Note that the equation above includes $n = 0$ as a base case. That is, the simplest ψ-terms are of the form $X : s$.

We can associate to a ψ-term $\psi = X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n)$ the OSF-constraint:

$$\phi(\psi) = X : s \;\&\; X.\ell_1 \doteq Y_1 \;\&\; \ldots \;\&\; X.\ell_n \doteq Y_n \;\&\; \phi(\psi_1) \;\&\; \ldots \;\&\; \phi(\psi_n)$$

where $Y_1, \ldots, Y_n$ are the roots of $\psi_1, \ldots, \psi_n$, respectively. We say that the OSF-constraint $\phi(\psi)$ is obtained from *dissolving* the ψ-term ψ, and refer to the OSF-constraint as the *dissolved ψ-term*. We will often deliberately confuse a ψ-term ψ with its dissolved form $\phi(\psi)$ and simply refer to $\phi(\psi)$ simply as ψ.

Given the interpretation $\mathcal{A}$, the *denotation $[\![\psi]\!]^{\mathcal{A},\alpha}$ under a valuation $\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}$* of a ψ-term ψ with root X is given as:

$$[\![\psi]\!]^{\mathcal{A},\alpha} = \{d \in D^{\mathcal{A}} \mid \alpha(X) = d,\ \mathcal{A}, \alpha \models \psi\}.$$

Note that this is either the singleton $\{\alpha(X)\}$ or the empty set.

The *type-as-set denotation* of a ψ-term ψ is defined as the set of domain elements:

$$[\![\psi]\!]^{\mathcal{A}} = \bigcup_{\alpha \in Val(\mathcal{A})} [\![\psi]\!]^{\mathcal{A},\alpha}.$$

This amounts to saying that:

$$[\![\psi]\!]^{\mathcal{A}} = \{d \in D^{\mathcal{A}} \mid \text{there exists } \alpha \in Val(\mathcal{A}) \text{ such that } \alpha(Z) = d, \text{ and } \mathcal{A}, \alpha \models \exists \mathcal{X}\ Z : \psi\}$$

where Z is a new variable not occurring in ψ, $\mathcal{X} = Var(\psi)$, $Z : \psi$ stands for $Z \doteq X \;\&\; \psi$, and $X \in \mathcal{X}$ is ψ's root variable.

A ψ-term ψ with root X corresponds to a unique rooted graph g which is the direct translation of the constraint ψ together with an indication of the root. The nodes of g are exactly the variables of ψ. A node Z is labeled by the sort s if the conjunction ψ contains a non-trivial sort constraint $Z : s$, and by the sort $\top$, otherwise. For every feature constraint $Y.\ell \doteq Z$ the graph g has a directed edge (Y, Z) which is labeled by the feature ℓ. The root of g is the node X. Clearly, g is the natural graphical representation of ψ.[8]

3.3 Syntactic interpretations

Among all OSF-algebras, there are those whose domain elements are concrete data structures. We call these *syntactic interpretations*. We will now present three important examples obtained directly from the syntactic expressions of ψ-terms. They turn out to be *canonical interpretations* for OSF-constraints.[9]

The most immediate syntactic OSF-interpretation is the OSF-algebra of ψ-terms. The domain of is the set of all ψ-terms, up to graph representation. That is, we identify ψ-terms as values of if they are represented by the same graph. For example, the two ψ-terms $Y : s(\ell_1 \Rightarrow X : s', \ell_2 \Rightarrow X)$ and $Y : s(\ell_1 \Rightarrow X, \ell_2 \Rightarrow X : s')$ clearly correspond to the same object. Indeed, they have the same OSF-graph representation.

Sorts $s \in \mathcal{S}$ are interpreted as:

$$s = \{\psi \in D \mid s' \leq s, \text{ where } s' \text{ is the root sort of the graph of } \psi\},$$

and features $\ell \in \mathcal{F}$ are interpreted as functions $\ell : D \mapsto D$ as follows. Let ψ be a ψ-term and g its graph. If (X, Y) is the edge of g labeled by ℓ, then $\ell(g)$ is the ψ-term represented by the maximally connected subgraph g' of g rooted at the node Y. That is, g' is obtained by removing all nodes and edges which are not reachable by a directed path from the node Y.

If X does not have the feature ℓ, *i.e.*, there is no outgoing edge from the root of g labeled ℓ, then ℓ is the ψ-term $Z_{\ell,\psi} : \top$, for a new variable $Z_{\ell,\psi}$ uniquely determined by the feature ℓ and the ψ-term ψ.

For example, taking $\psi = X : \top(\ell_1 \Rightarrow Y : s, \ell_2 \Rightarrow X)$, we have $\ell_1(\psi) = Y : s$, $\ell_2(\psi) = \psi$, and $\ell_3(\psi) = Z_{\ell_3,\psi} : \top$.

We obtain two other examples of OSF-algebras when we factorize the ψ-term domain by further identifying values. The first one identifies two ψ-terms which are equal up to variable renaming. The obtained domain obviously spans an OSF-algebra. We call this OSF-algebra $_0$.

The second one is obtained from $_0$ by further identifying two ψ-terms if their (possibly infinite) tree unfoldings are equal. A tree unfolding is obtained from a ψ-term by associating a unique node to every feature path. It is well known that a rooted directed graph represents a unique rational tree [14]. In our case, we obtain trees whose nodes are labeled by sorts and whose

[8] Refer to Figure 2 on Page 8 for an example.

[9] If an OSF-constraint is satisfiable in some interpretation, then it is also satisfiable in all canonical interpretations.

edges are labeled by features. We call these (rational) OSF-trees. It is again clear that the set of all OSF-trees spans an OSF-algebra $\mathcal{T}$.[10]

Formally, OSF-algebras can also be introduced as logical structures, namely models providing interpretations for the sort symbols as unary predicates and the feature symbols as unary functions, which satisfy the *Sort Axiom* saying, for all sorts s and s',

$$X : s \;\&\; X : s' \;\rightarrow\; X : s \wedge s'.$$

Furthermore, both $_0$ and $\mathcal{T}$ satisfy a *Constructibility Axiom* stating essentially the satisfiability of any OSF-constraint ϕ coming from dissolving a ψ-term ψ. More precisely, if $\mathcal{X} = \mathit{Var}(\phi)$ and, for $i = 1, \ldots, n$, $X_i.\ell_i \doteq Y \notin \phi$ for any variable Y, and $Y_i \notin \mathit{Var}(\phi)$, and $X_i \in \mathcal{X}$, then this axiom states the validity of:

$$\forall Y_1 \ldots \forall Y_n.\ \exists \mathcal{X}.\ \phi \;\&\; X_1.\ell_1 \doteq Y_1 \;\&\; \ldots \;\&\; X_n.\ell_n \doteq Y_n.$$

The constructibility axiom is a generalization of the axiom of functionality which is valid for first-order terms. Namely, the axiom which guarantees that, given a constructor symbol f of rank n, an individual $X = f(Y_1, \ldots, Y_n)$ exists if individuals Y_i exist, $i = 1, \ldots, n$. Formally, taking $\phi = X : f$,

$$\forall Y_1 \ldots \forall Y_n.\ \exists X.\ X : f \;\&\; X.1 \doteq Y_1 \;\&\; \ldots \;\&\; X.n \doteq Y_n.$$

The form we give for constructibility is indeed more general than plain functionality since it states the existence of something which is not valid for first-order terms; *e.g.*, self-referential individuals. For example, $\exists X.\ X.\ell \doteq X$ is obtained as an instance of our axiom by taking $n = 0$ and $\phi = X.\ell \doteq X$.

3.4 OSF-unification

We describe next how to determine whether an OSF-constraint ϕ is consistent; *i.e.*, if it is satisfiable in some OSF-algebra $\mathcal{A}$—and, therefore, in particular in . Unification of two ψ-terms reduces to this problem.

Definition 4 (Solved OSF-Constraints) *An OSF-constraint ϕ is called solved if for every variable X, ϕ contains:*

- *at most one sort constraint of the form $X : s$, with $\bot < s$;*
- *at most one feature constraint of the form $X.\ell \doteq Y$ for each ℓ; and,*
- *no other occurrence of the variable X if it contains the equality constraint $X \doteq Y$.*

[10] $\mathcal{T}$ is essentially the feature tree structure of [7] and [8, 25]. The difference lies in our using partially-ordered sorts and total, as opposed to partial, features.

In [6, 5], we show that an OSF-constraint in solved form is always satisfiable. Now, by Definition 3, the OSF-constraint obtained as the dissolved form of any ψ-term ψ is *de facto* in solved form.[11] Hence, such a constraint is always satisfiable. It is so, in particular, in the canonical interpretation with, interestingly enough, the valuation that assigns to each variable X in ψ the value in D that is the very ψ-term rooted in X in ψ. For this reason, a ψ-term can also be seen as a variable substitution.

Given an OSF-constraint ϕ, it can be normalized by choosing non-deterministically and applying any applicable rule among the transformations rules shown in Figure 3 until none

Feature Decomposition:

(B.1) $$\frac{\psi \;\&\; U.\ell \doteq V \;\&\; U.\ell \doteq W}{\psi \;\&\; U.\ell \doteq V \;\&\; W \doteq V}$$

Sort Intersection:

(B.2) $$\frac{\psi \;\&\; U : s \;\&\; U : s'}{\psi \;\&\; U : s \wedge s'}$$

Variable Elimination:

(B.3) $$\frac{\psi \;\&\; U \doteq V}{\psi[V/U] \;\&\; U \doteq V} \qquad \textit{if } U \in \mathit{Var}(\psi) \textit{ and } U \neq V$$

Inconsistent Sort:

(B.4) $$\frac{\psi \;\&\; X : \bot}{\bot}$$

Variable Clean-up:

(B.5) $$\frac{\psi \;\&\; U \doteq U}{\psi}$$

Figure 3. Basic simplification

applies. A rule transforms the numerator into the denominator. The expression $\phi[X/Y]$ stands for the formula obtained from ϕ after replacing all occurrences of Y by X.

Theorem 1 (OSF-Constraint Normalization) *The rules of Figure 3 are solution-preserving, finite terminating, and confluent (modulo variable renaming). Furthermore, they always result in a normal form that is either the* false *constraint* $\bot$ *or an OSF-constraint in solved form.*

[11] More precisely, this is true if we forget superfluous trivial sort constraints of the form $X : \top$.

For our purposes, the constraint ϕ to be normalized will be of the form $\psi_1 \;\&\; \psi_2 \;\&\; X_1 \doteq X_2$; *i.e.*, the conjunction of the dissolved ψ-terms ψ_1 and ψ_2 together with an equation identifying their root variables X_1 and X_2. If ϕ normalizes to the *false* constraint, then the two ψ-terms are non-unifiable. Otherwise, the resulting solved OSF-constraint is a conjunction of equality constraints and of the dissolved form of some ψ-term. This ψ-term is *the most general unifier* of ψ_1 and ψ_2, up to variable renaming. We shall see that this ψ-term has two equivalent order-theoretic characterizations (*cf.*, Propositions 3 and 4).

3.5 OSF-orderings and semantic transparency

In this section, we first introduce the notion of *endomorphic approximation* which captures precisely and elegantly object inheritance. We also show how it relates to the logic and type views.

Endomorphisms on a given OSF-algebra $\mathcal{A}$, *i.e.*, homomorphisms from $\mathcal{A}$ to $\mathcal{A}$, induce a natural partial ordering.

Definition 5 (Endomorphic Approximation) *On each OSF-algebra $\mathcal{A}$ an approximation preorder $\sqsubseteq_{\mathcal{A}}$ is defined such that, for two elements d and e in $D^{\mathcal{A}}$, d* approximates *e if and only if e is an endomorphic image of d. Formally,*

$$d \sqsubseteq_{\mathcal{A}} e \;\; \textit{iff} \;\; \gamma(d) = e \;\; \textit{for some endomorphism} \;\; \gamma : \mathcal{A} \mapsto \mathcal{A}.$$

We shall omit subscripting $\sqsubseteq_{\mathcal{A}}$ and write $\sqsubseteq$ when $\mathcal{A} = $. Notice that this ordering on ψ-terms as values of the domain of translates into an information-theoretic approximation ordering on ψ-terms as types.

We note that endomorphisms on are graph homomorphisms with the additional sort-compatibility property. A node labeled with sort s is always mapped into a node labeled with s or a subsort of s. An edge labeled with a feature is mapped into an edge labeled with the same feature. Thus, endomorphic approximation captures exactly object-oriented class inheritance. Indeed, if an attribute is present in a class, then it is also present in a subclass with a sort that is the same or refined. Since features are total functions, this also takes care of introducing a new attribute in a subclass: it refines $\top$. Note also, that the restriction of γ to the set of nodes defines a variable binding; it corresponds to the notion of a matching substitution for first-order terms.

The following fact was established in [6, 5].

Proposition 1 (ψ-Terms as Filters) *The denotation of a ψ-term in is the set of all ψ-terms it approximates; i.e.,*

$$[\![\psi]\!] = \{\psi' \in D \mid \psi \sqsubseteq \psi'\}.$$

The next ordering is the type ordering on ψ-terms which we informally called "more specific than" in Section 1.2 and Section 2.

Definition 6 (ψ-Term Subsumption) *A ψ-term ψ is* subsumed by *a ψ-term ψ' if and only if the denotation of ψ is contained in that of ψ' in all interpretations. Formally,*

$$\psi \leq \psi' \text{ iff } [\![\psi]\!]^{\mathcal{A}} \subseteq [\![\psi']\!]^{\mathcal{A}}$$

for all OSF-algebras $\mathcal{A}$.

In fact, it is sufficient to limit the above statement to the OSF-algebra only; *i.e.*, $[\![\psi]\!] \subseteq [\![\psi']\!]$.

The next and last ordering is a logical ordering on ψ-terms. We state it here in less general terms than in [6, 5].

Definition 7 (ψ-term Entailment) *A ψ-term ψ* entails *a ψ-term ψ' if and only if, as constraints, ψ implies the conjunction of ψ' and $X \doteq X'$; more precisely,*

$$\psi \succeq \psi' \text{ iff } \models \psi \rightarrow \exists \mathcal{U}\ (X \doteq X' \,\&\, \psi')$$

where X, X' are the roots of ψ and ψ' and $\mathcal{U} = Var(\psi')$.

It is again sufficient to state the validity of the implication in the OSF-algebra only (namely, using $\models$). This is not true in the more general wording and holds here only because the constraints are obtained by dissolving ψ-terms and their root variables are bound together.

Proposition 2 (Semantic Transparency of Orderings) *The following are equivalent:*

- $\psi \sqsubseteq \psi'$ — *ψ is an approximation of ψ';*
- $\psi' \leq \psi$ — *ψ' is a subtype of ψ;*
- $\psi' \succeq \psi$ — *ψ entails ψ';*
- $[\![\psi]\!] \subseteq [\![\psi']\!]$ — *the set of ψ-terms filtered by ψ is contained in that filtered by ψ'.*

The following two propositions are straightforward. Let ψ_1 and ψ_2 be two ψ-terms with variables renamed apart; *i.e.*, such that $Var(\psi_1) \cap Var(\psi_2) = \emptyset$. Let X_1 and X_2 be their respective root variables. Let ϕ be the normal form of the OSF-constraint $\psi_1 \,\&\, \psi_2 \,\&\, X_1 \doteq X_2$.

Proposition 3 (ψ-Term Unification) *The normal form ϕ is the* false *constraint if and only if $[\![\psi_1]\!]^{\mathcal{A}} \cap [\![\psi_2]\!]^{\mathcal{A}} = \emptyset$, for all OSF-algebras $\mathcal{A}$. Otherwise, ϕ is the conjunction of equality constraints and of the dissolved version of some ψ-term ψ. This ψ-term is the $\leq$-GLB of ψ_1 and ψ_2 up to variable renaming; i.e., $[\![\psi]\!]^{\mathcal{A}} = [\![\psi_1]\!]^{\mathcal{A}} \cap [\![\psi_2]\!]^{\mathcal{A}}$.*

Proposition 4 ($\sqsubseteq$-LUB of two ψ-terms) *The ψ-term ψ above is approximated by both ψ_1 and ψ_2 and is the least ψ-term for $\sqsubseteq$ (i.e., approximating all other ones) with this property.*

4 Entailment and disentailment of OSF-constraints

This section deals formally with all the apparatus presented and used informally in Section 2.2.

In the following, we use ϕ as the *context* formula. It is assumed to be an *OSF*-constraint in solved form, although not necessarily coming from dissolving a single ψ-term. The variables in ϕ are *global*. We shall use $\mathcal{X}$ to designate the set of global variables $Var(\phi)$ and the letters $X, Y, Z, \ldots$, for variables in $\mathcal{X}$. We use ψ, a dissolved ψ-term, as the *guard* formula. The variables in ψ are *local* to ψ; *i.e.*, $Var(\phi) \cap Var(\psi) = \emptyset$. We shall use $\mathcal{U}$ to designate the set of local variables $Var(\psi)$ and the letters $U, V, W, \ldots$, for variables in $\mathcal{U}$. The letter U will always designate the root variable of ψ. We also refer to ϕ as the *actual* parameter, and to ψ as the *formal* parameter. By extension, we will often use the qualifiers global/local, actual/formal, and context/guard, with all syntactic entities; *e.g.*, variables, formulae, constraints, or sorts.

We investigate a proof system which decides two problems simultaneously:

- the validity of the implication $\forall\mathcal{X}\ (\ \phi \rightarrow \exists\mathcal{U}.\ (\psi\ \&\ U \doteq X)\ )$;
- the unsatisfiability of the conjunction $\phi\ \&\ \psi\ \&\ U \doteq X$.

The first test is called a test for *entailment* of the guard by the context, and the second, a test for *disentailment*. This second test is equivalent to testing the validity of the implication $\forall\mathcal{X}\ (\ \phi \rightarrow \neg\exists\mathcal{U}.\ (\psi\ \&\ U \doteq X)\ )$.

Since both tests amount to deciding whether the context implies the guard or its negation, all local variables are existentially quantified and all global variables are universally quantified.

The *relative-simplification* system for OSF-constraints is given by the rules in Figures 4, 5, and 6. An OSF-constraint ψ simplifies to ψ' relatively to ϕ by a simplification rule ρ if $\frac{\psi}{\psi'}$ is an instance of ρ and the applicability condition (on ϕ and on ψ) is satisfied. We say that ψ simplifies to ψ' relatively to ϕ if it does so in a finite number of steps.

The relative-simplification system preserves an important invariant property: *a global variable never appears on the left of a variable equality constraint in the formula being simplified.* Thus, an equality $U \doteq X$ is a *directed* relation binding the local variable U to the global variable X. Furthermore, a global variable is never eliminated by a local one, or *vice versa*.

A set of bindings $U_i \doteq X_i$, $i = 1, \ldots, n$ is a *functional binding* if all the variables U_i are mutually distinct.

The effectuality of the relative-simplification system is summed up in the following statement:

> Effectuality of relative-simplification *The solved OSF-constraint ϕ entails (resp., disentails) the OSF-constraint $\exists U.\ (U \doteq X\ \&\ \psi)$ if and only if the normal form ψ' of $\psi\ \&\ U \doteq X$ relatively to ϕ is a conjunction of equations making up a functional binding (resp., is the* false *constraint $\psi' = \bot$).*

Feature Decomposition:

$$\text{(F.1)}\quad \frac{\psi \;\&\; U.\ell \doteq V \;\&\; U.\ell \doteq W}{\psi \;\&\; U.\ell \doteq V \;\&\; W \doteq V}$$

Relative Feature Decomposition:

$$\text{(F.2)}\quad \frac{\psi \;\&\; U \doteq X \;\&\; U.\ell \doteq V}{\psi \;\&\; U \doteq X \;\&\; V \doteq Y} \qquad \text{if } X.\ell \doteq Y \in \phi$$

Relative Feature Equality:

$$\text{(F.3)}\quad \frac{\psi \;\&\; U \doteq X_1 \;\&\; U \doteq X_2 \;\&\; V \doteq Y_1}{\psi \;\&\; U \doteq X_1 \;\&\; U \doteq X_2 \;\&\; V \doteq Y_1 \;\&\; V \doteq Y_2} \qquad \begin{array}{l}\text{if } X_1.\ell \doteq Y_1 \in \phi,\ X_2.\ell \doteq Y_2 \in \phi \\ \text{and } V \doteq Y_2 \notin \psi\end{array}$$

Variable Introduction:

$$\text{(F.4)}\quad \frac{\psi \;\&\; U \doteq X_1 \;\&\; U \doteq X_2}{\psi \;\&\; U \doteq X_1 \;\&\; U \doteq X_2 \;\&\; V \doteq Y_1 \;\&\; V \doteq Y_2} \qquad \begin{array}{l}\text{if } X_1.\ell \doteq Y_1 \in \phi,\ X_2.\ell \doteq Y_2 \in \phi \\ \text{and } Y_1 \notin Var(\psi) \text{ and } Y_2 \notin Var(\psi) \\ \text{where } V \text{ is a new variable}\end{array}$$

Figure 4. Simplification relatively to ϕ: Features

There are two technical remarks to be made. Firstly, observe that in our formulation of the entailment/disentailment problem, the implication contains *only one* equality $U \doteq X$ binding *only one* global variable. However, this is not a restriction. Equations $U_1 \doteq X_1, \ldots, U_n \doteq X_n$ can be equivalently replaced by adding $X_1 \doteq X.1 \;\&\; \ldots \;\&\; X_n \doteq X.1$ to the context ϕ and $U_1 \doteq U.1 \;\&\; \ldots \;\&\; U_n \doteq U.n \;\&\; U \doteq X$ to ψ, where X and U are new. That is, one obtains the conjunction of one equality $U \doteq X$ and a guard which, again, is a dissolved ψ-term.

Secondly, the fact that ψ is a dissolved ψ-term rooted in U ensures that the test of entailment of $\psi \;\&\; U \doteq X$ by ϕ does not depend on whether the implication holds in *all* OSF-interpretations, or only in , or $\mathcal{T}$. This is not necessarily so if U is not the root of ψ. Indeed, let us assume that U is *not* the root of ψ; for example, take ψ to be $V.\ell \doteq U$. Clearly, while $\forall X \left(\top \rightarrow \exists U \exists V \left(\psi \;\&\; U \doteq X\right)\right)$ holds in and $\mathcal{T}$, it does not hold in all OSF-algebras where it is not guaranteed that every element is the ℓ-image of some other element. In (and $\mathcal{T}$), this is the case since any element X is the ℓ-image of at least one element; namely, $\top(\ell \Rightarrow X)$.

Effectuality of relative-simplification is the central result of this section. We now proceed through the technical details aimed at establishing its claim in the form of two theorems: Theorem 2 and Theorem 3.

Sort Intersection:

$$\text{(S.1)}\quad \frac{\psi \;\&\; U : s \;\&\; U : s'}{\psi \;\&\; U : s \wedge s'}$$

Sort Containment:

$$\text{(S.2)}\quad \frac{\psi \;\&\; U \doteq X \;\&\; U : s}{\psi \;\&\; U \doteq X} \qquad \textit{if } X : s' \in \phi, \textit{ and } s' \leq s$$

Sort Refinement:

$$\text{(S.3)}\quad \frac{\psi \;\&\; U \doteq X \;\&\; U : s}{\psi \;\&\; U \doteq X \;\&\; U : s \wedge s'} \qquad \textit{if } X : s' \in \phi, \textit{ and } s \wedge s' < s$$

Relative Sort Intersection:

$$\text{(S.4)}\quad \frac{\psi \;\&\; U \doteq X \;\&\; U \doteq X'}{\psi \;\&\; U \doteq X \;\&\; U \doteq X' \;\&\; U : s \wedge s'} \qquad \begin{array}{l}\textit{if } X : s \in \phi, X' : s' \in \phi,\\ s \wedge s' < s, s \wedge s' < s',\\ \textit{and } U : s'' \notin \psi, \textit{ for any sort } s''\end{array}$$

Sort Inconsistency:

$$\text{(S.5)}\quad \frac{\psi \;\&\; U : \bot}{\bot}$$

Figure 5. Simplification relatively to ϕ: Sorts

Relative Variable Elimination:

$$\text{(E.1)}\quad \frac{\psi \;\&\; U \doteq X \;\&\; V \doteq X}{\psi[U/V] \;\&\; U \doteq X \;\&\; V \doteq X} \qquad \begin{array}{l}\textit{if } V \in \mathit{Var}(\psi), V \doteq X \notin \psi,\\ \textit{and } U \neq V\end{array}$$

Equation Entailment:

$$\text{(E.2)}\quad \frac{\psi \;\&\; U \doteq X \;\&\; U \doteq Y}{\psi \;\&\; U \doteq X} \qquad \textit{if } X = Y \textit{ or if } X \doteq Y \in \phi.$$

Figure 6. Simplification relatively to ϕ: Equations

4.1 Termination of relative simplification

For the purpose of showing that the relative simplification rules always terminate, we introduce an additional set of rules shown in Figure 7 extending basic simplification. These rules are *not* meant to be used in the effective operation of basic simplification, but only serve in our proof argument. The idea is that relative simplification of a guard ψ relatively to a context ϕ can be "simulated" by normalizing the formula $\phi \;\&\; \psi \;\&\; U \doteq X$ using basic simplification (Figure 3) together with the rules of Figure 7. It is not a real simulation, however, as Rules (B.1)–(B.5) have for side effect to destroy the context. The point is that one application of a relative simplification rule can be made to correspond to at least one application of one of Rules (B.1)–(B.5), (X.1)–(X.3). Since this latter system can be shown to terminate, then so can relative simplification.

Rules (X.1)–(X.3) perform essentially the same work as Rules (B.1) and (B.2) except that they do no erase parts of the formula. In Rule (X.1), we denote by $\sim_{\doteq}$ the reflexive, symmetric and transitive closure of $\doteq$ (that is, the equivalence relation on the variables occurring in the constraint which is generated by the $\doteq$-pairs between variables in the constraint).

Extended Feature Decomposition:

(X.1) $$\frac{\psi \;\&\; U.\ell \doteq U' \;\&\; U.\ell \doteq U''}{\psi \;\&\; U.\ell \doteq U' \;\&\; U.\ell \doteq U'' \;\&\; U'' \doteq U'} \quad \textit{if } U' \not\sim_{\doteq} U''$$

Extended Sort Intersection 1:

(X.2) $$\frac{\psi \;\&\; U : s \;\&\; U : s'}{\psi \;\&\; U : s \;\&\; U : s \sqcap s'} \quad \begin{array}{l}\textit{if } s \wedge s' < s'' \textit{ for any } s'' \\ \textit{such that } U : s'' \in \psi\end{array}$$

Extended Sort Intersection 2:

(X.3) $$\frac{\psi \;\&\; U : s \;\&\; U : s'}{\psi \;\&\; U : s \;\&\; U : s' \;\&\; U : s \sqcap s'} \quad \begin{array}{l}\textit{if } s \wedge s' < s'' \textit{ for any } s'' \\ \textit{such that } U : s'' \in \psi\end{array}$$

Figure 7. Rules extending basic simplification

Lemma 1 *The extended basic-simplification rules (B.1)–(B.5), (X.1)–(X.3) define equivalence transformations; furthermore, they are terminating.*

Proof: The first statement is clear. The proof of the second statement is an extension of the termination proof of the basic simplification rules (B.1)–(B.5) from [6, 5]: (X.1) can be applied only a finite number of times, since the number of equivalence classes partitioning the finite set of variables occurring in the constraint which is to be simplified decreases by 1 with each application. (X.2) and (X.3) can be applied only a finite number of times, since they can be applied at most once for every sort occurring in the constraint which is to be simplified. ∎

Lemma 2 *Let ψ & $U \doteq X$ simplify to ψ' relatively to ϕ by a relative-simplification step not using Rule (F.4). Then, ϕ & ψ & $X \doteq U$ simplifies to ϕ' & ψ'' by at most one extended basic-simplification step and a finite number of variable elimination (B.3), where ψ' and ψ'' are equal up to variable renaming.*

Proof: It can be seen that each relative simplification rule, except for (F.4), corresponds to one or several extended basic-simplification rules. Rules (F.1)–(F.3) correspond to Rules (B.1) and (X.1). Rules (S.1)–(S.4) correspond to Rules (B.2), (X.2) and (X.3). Rules (E.1)–(E.2) correspond to Rule (B.3). This, and the fact that extended basic-simplification rules are equivalence transformations, allow us to conclude. ∎

Lemma 3 *Let ψ simplify to ψ' of the form ψ & $U_1 \doteq X_1$ & $U_1 \doteq X_2$ by an application of Rule (F.4) relatively to ϕ. Then, ψ & $U_1 \doteq X_1$ simplifies to the same constraint ψ' by an application of Rule (F.3) relatively to ϕ.*

Proposition 5 *The relative-simplification rules are terminating.*

Proof: This is proved by induction on n, using Lemma 2 and Lemma 3. For every relative-simplification chain ψ_1 & $U_1 \doteq X_1, \ldots, \psi_n$ & $U_n \doteq X_n$ relatively to ϕ, there exists an extended-basic simplification chain of length $n + k$, where $k \geq 0$. This chain starts with the basic constraint ϕ & ψ & $X_1 \doteq U_1$ & $X \doteq U$, where $X \doteq U$ stands for the equations we have added so that each global variable X is bound to some local variable U (which, if necessary, is chosen new).

Since, according to Lemma 1, extended-basic-simplification chains are finite, so are relative-simplification chains. ∎

4.2 Correctness and completeness

We first note another consequence of the lemmata of the last section. Let $\mathcal{V}$ stand for the new local variables introduced by Rule (F.4).

Proposition 6 *Let ψ & $U \doteq X$ simplify to ψ' relatively to ϕ. Then, ϕ & ψ & $U \doteq X$ and $\exists \mathcal{V}.\ (\phi$ & $\psi')$ are equivalent.*

Proof: Let us first assume that ψ & $U \doteq X$ simplifies to ψ' relatively to ϕ, not using Rule (F.4). Then, ϕ & ψ & $U \doteq X$ and ϕ & ψ' are equivalent by Lemma 1 and Lemma 2. Let ψ & $U \doteq X$ simplify to ψ & $U \doteq X$ & $V \doteq X_1$ & $V \doteq X_2$ relatively to ϕ, by an application of Rule (F.4). Clearly, ϕ & ψ & $U \doteq X$ and ϕ & $V.\ (\psi$ & $U \doteq X$ & $V \doteq X_1)$ are equivalent. Thus, with Lemma 3, we can apply the first part of the proof on ψ & $U \doteq X$ & $V \doteq X_1$. ∎

The next corollary states a property which is important for showing that relative simplification can be used for proving entailment, the *invariance property*.

Corollary 1 (Invariance of Relative-Simplification) *If ψ & $U \doteq X$ simplifies to ψ' relatively to ϕ, then $\exists \mathcal{U}.\ (\phi$ & ψ & $U \doteq X)$ and $\exists \mathcal{U} \exists \mathcal{V}.\ (\phi$ & $\psi')$ are equivalent.*

It is helpful to list systematically the normal-form properties of the relative-simplification system.

Proposition 7 *The constraint ψ is in normal form relatively to ϕ if and only if the following conditions are satisfied:*

- *ψ is in solved-form;*
- *a global variable X may occur in ψ only in the form $\_ \doteq X$;*
- *if $X \doteq \_ \in \phi$, then X does not occur in ψ;*
- *if $V \doteq X \in \psi$, and $\_ \doteq X.\ell \in \phi$, then $\_ \doteq V.\ell \notin \psi$;*
- *if $V \doteq X \in \psi$, and $X : s \in \phi$, and $V : s' \in \psi$, then $s' < s$;*
- *if $\begin{matrix} V \doteq X, \\ V \doteq Y \end{matrix} \in \psi$, and $\begin{matrix} X' \doteq X.\ell, \\ Y' \doteq Y.\ell \end{matrix} \in \phi$, then $\begin{matrix} W \doteq X', \\ W \doteq Y' \end{matrix} \in \psi$,*

  for some variable W;

- *if $\begin{matrix} V \doteq X, \\ V \doteq Y \end{matrix} \in \psi$, and $\begin{matrix} X : s_1, \\ Y : s_2 \end{matrix} \in \phi$, then $V : s \in \psi$,*

  for some sort s such that $s \leq s_1$ and $s \leq s_2$.

Proof: by inspection of the relative-simplification rules. ∎

Proposition 8 *Let ψ' be a normal form of ψ & $U \doteq X$ relatively to ϕ. Let ϕ' be the constraint obtained from ϕ eliminating all redundancies according to the rules of Figure 8, and removing bindings $V \doteq \_$ of new variables introduced by (F.4). Then, the constraint ϕ' & ψ' is a solved-form of the constraint ϕ & ψ & $U \doteq X$, up to variable renaming.*

Proof: According to Proposition 6, ϕ & ψ & $U \doteq X$ is equivalent to . ϕ & ψ $'$, where stands for the new variables. According to the last three conditions of Proposition 7, Rules (R.1), (R.2) or (R.3) perform equivalence transformations. Thus, if applications of these rules modify ϕ' to ϕ'', then ϕ' & ψ' is equivalent to ϕ'' & ψ'.

According to the first four conditions of Proposition 7, ϕ'' & ψ' is in solved-form up to variable eliminations via Rule (B.3). More precisely, these variable eliminations are applications of Rule (B.3) using new equations of the form $V \doteq X$ introduced by Rule (F.4). They produce possibly equations of the form $X \doteq Y$ between global variables; then, further variable eliminations consist of applications of Rule (B.3) using these new equations. As a last step, these new equations are removed in order to obtain a constraint which is exactly equivalent to ϕ & ψ & $U \doteq X$, and not just up to existential quantification of new variables. ∎

Corollary 2 *If the normal form of ψ & $U \doteq X$ relatively to ϕ is not $\perp$, then ϕ & ψ & $U \doteq X$ is satisfiable.*

Redundant Sort Elimination:

(R.1) $\dfrac{\phi \;\&\; X : s}{\phi}$ *if* $U \doteq X \in \psi$, *and* $U : s' \in \psi$ *for some* $s' \leq s$

Redundant Feature Elimination:

(R.2) $\dfrac{\phi \;\&\; X'_1 \doteq X_1.\ell \;\&\; X'_2 \doteq X_2.\ell}{\phi \;\&\; X'_1 \doteq X_1.\ell}$ *if* $U \doteq X_1 \in \psi$, $U \doteq X_2 \in \psi$

Entailed Sort Redundancy Elimination:

(R.3) $\dfrac{\phi \;\&\; X_1 : s \;\&\; X_2 : s}{\phi \;\&\; X_1 : s}$ *if* $U \doteq X_1 \in \psi$, $U \doteq X_2 \in \psi$

Figure 8. Redundancy elimination rules

Proof: In [6, 5] we showed that a constraint is satisfiable if and only if it has a solved-form; that is, its basic normal form is different from . The statement then follows from Proposition 8. ∎

Theorem 2 (Disentailment) *Let ψ' be a normal form of $\psi \;\&\; U \doteq X$ relatively to ϕ. Then, ϕ disentails $\exists \mathcal{U}.\ (\psi \;\&\; U \doteq X)$ if and only if $\psi' = \bot$.*

Proof: If $\psi' =$, then $(\phi \quad . \psi \qquad\qquad ')$ is valid. From Corollary 1, it follows that $(\phi \quad . \psi \;\&\; U \qquad\qquad \doteq X)$ is valid, too. If $\psi' =$, then Corollary 2 can be applied. ∎

Proposition 9 *If the normal form ψ' of $\psi \;\&\; U \doteq X$ relatively to ϕ is not a conjunction of equations representing a functional binding, then $\phi \;\&\; \neg \exists \mathcal{U}.\ (\psi \;\&\; U \doteq X)$ is satisfiable.*

Proof: The assumption on the form of ψ' means that one of the three following cases is true, for some $V \;\; Var(\psi \;\; ')$ bound to some $X \;\; Var(\phi)$; *i.e.*, $V \;\; \doteq X \;\; \psi \;\; '$.

[(1)] $\psi \;\; '$ contains a sort constraint on V; say, $V : s$; or,
[(2)] $\psi \;\; '$ contains two equations on V; say, $V \doteq X \;\&\; V \doteq Y$; or,
[(3)] $\psi \;\; '$ contains a feature constraint on V, say, $V.\ell \doteq W$.

For each case, we can find a constraint ϕ' such that $\phi \;\&\; \phi'$ is satisfiable and disentails ψ'. Then, $\phi \;\&\; \phi'$ also disentails $. (\psi \;\&\; U \quad \doteq X)$; *i.e.*, $\phi \;\&\; \phi' \;\; U.\ (\psi \;\&\; U \qquad \doteq X)$ is valid. Clearly, this is sufficient to show that $\phi \;\&\; . (\psi \;\&\; U \qquad \doteq X)$ is satisfiable.

(1) $V : s \;\; \psi \;\; '$; then, according to the third condition of Proposition 7, ϕ contains either no sort constraint on X or one of the form $X : s'$ where $s < s'$. Thus, we set $\phi' = X : s''$, in the first case, for some sort s'' incompatible with s; *i.e.*, such that $s \;\; s \;\; '' =$. In the second case, we choose $s \;\; ''$ such that $s \;\; s \;\; '' =$ and $s \;\; '' \;\; s \;\; '$.

(2) $V \doteq X \;\&\; V \doteq Y \;\; \psi \;\; '$; then, either $V : s \;\; \psi \;\; '$ and we are in Case (2), or, according to the last condition of Proposition 7, at most one of X and Y is sorted in ϕ. If $Y : s \;\; \phi$, we set $\phi \;\; ' = X : s'$ for

some sort s' such that $s\ s\ ' = $. If none of X and Y is sorted in ϕ, we set $\phi\ ' = Y : s \ \&\ X : s'$ for some sorts s, s' such that $s\ s\ ' = $.

(3) $V.\ell_1 \doteq V_1\ \psi\ '$; then, ϕ contains no feature constraint $X.\ell_1 \doteq \_$ according to the fourth condition of Proposition 7. Without loss of generality, we can assume that ψ does not contain redundant conjuncts.[12] There exists a sort s such that ψ contains a conjunct of the form: $V.\ell_1 \doteq V_1 \ \&\ V_1.\ell_2 \doteq V_2 \ \&\ \ldots\ \&\ V_{n-1}.\ell_n \doteq V_n \ \&\ V_n : s$, for some $n \quad 1$. Thus, we set $\phi' = X.\ell_1 \doteq X_1 \ \&\ X_1.\ell_2 \doteq X_2 \ \&\ \ldots\ \&\ X_{n-1}.\ell_n \doteq X_n \ \&\ X_n : s'$, for some new variables $X_1, \ldots, X_n$ and some sort s' such that $s\ s\ ' = $. ∎

Theorem 3 (Entailment) *Let ψ' be a normal form of ψ relatively to ϕ. Then, ϕ entails $\exists\mathcal{U}.\ (\psi \ \&\ U \doteq X)$ if and only if ψ' is a functional binding. Moreover, $\phi \ \&\ \psi'$ is a solved OSF-constraint.*

Proof: If ψ' is a conjunction of equations representing a functional binding, then . $\psi\ \ '$ is valid; thus, so is $\phi\ \ U.\ \psi\ \ '$. By invariance of relative simplification (Corollary 1), it follows that $\phi\ \ .\ \psi$ is valid, too.

If ψ' has a different form then, either $\psi' = $, or $\psi\ '$ contains conjuncts that are not a functional binding. The fact that $\phi\ \ .\ \psi$ is not valid is trivial in the first case. In the other case, since the context ϕ is always assumed in solved form and, thus, satisfiable, then it follows from Proposition 9. ∎

Corollary 3 *Let ψ' be the relative-simplification normal form of $\psi \ \&\ U \doteq X$ relatively to ϕ. Then, the context entails the guard if and only if the conjunction $\phi \ \&\ \psi'$ is the solved-form of the conjunction $\phi \ \&\ \psi \ \&\ U \doteq X$.*

Proof: This is an immediate consequence of Theorem 3 and Proposition 8. ∎

4.3 Independence

The following theorem states that the OSF-constraint system has the independence property [19]. It is well-known that in any constraint system with this property it is possible to solve constraints which are conjunctions of constraints and negated constraints by testing entailment. Namely, $\phi \ \&\ \neg\exists\mathcal{U}_1\psi_1 \ \&\ \ldots \neg\exists\mathcal{U}_n\psi_n$ is satisfiable if and only if ϕ does not entail $\exists\mathcal{U}_i.\ \psi_i$, for every $i = 1, \ldots, n$. Here $\exists\mathcal{U}_i$ abbreviates the existential quantification of variables in $Var(\psi_i) - Var(\phi)$.

Clearly, ϕ entails $\exists\mathcal{U}_i.\ \psi_i$ if and only if ϕ entails $\exists\mathcal{U}_i\exists U_i.\ \psi_i[U_i/X_i] \ \&\ U_i \doteq X_i$, where we introduce a new variable U_i for every $X_i \in Var(\phi) \cap Var(\psi_i)$. Hence, given that the

[12] That is, we assume that every variable in ψ has at least one sort constraint and that redundant constraints in ψ are removed. A redundant constraint in ψ is one of the form $X.\ell \doteq Y \ \&\ Y : \top$ where Y does not occur elsewhere in ψ. Since we interpret features as total functions, this is not a proper restriction: redundant constraints can be moved into the functional expression or the body of the guarded clause without changing the declarative or the operational semantics. On the other hand, if this assumption is fulfilled, then the entailment of $\psi \ \&\ U \doteq X$ by ϕ does not depend on whether features are interpreted as total or partial functions.

independence property holds, we can use the relative-simplification algorithm in order to check satisfiability of conjunctions of positive and negative OSF-constraints.

For the formulation of the theorem, let us make a few assumptions that do not incur any loss of generality. First, we assume that $\mathcal{U}_i = Var(\psi_i)$, $U_i \in \mathcal{U}_i$, and $Var(\phi) \cap Var(\psi_i) = \emptyset$. Second, since they correspond to different existential quantification scopes, we will assume $\mathcal{U}_i \cap \mathcal{U}_j = \emptyset$ for $i \neq j$. Finally, we again assume that ψ_i does not contain redundant constraints (*cf.*, Footnote 12 on Page 33).

Theorem 4 (Independence) *A constraint ϕ entails the disjunction of the constraints $\exists \mathcal{U}_i.\ (\psi_i \ \& \ U_i \doteq X_i)$, for $i = 1, \ldots, k$, if and only if it entails one of them.*

Proof: The *if*-direction is trivial. It is sufficient to show that if $\phi \ \&$ $_i.\ (\psi_i \ \& \ U_i \doteq X_i)$ is satisfiable for every i, then $\phi \ \& \ \bigwedge_{i=1,\ldots,k}$ $_i.\ (\psi_i \ \& \ U_i \doteq X)$ is satisfiable.

Extending the proof technique of Proposition 9, we will find a constraint ϕ' such that $\phi \ \& \ \phi'$ is satisfiable and disentails ψ_i', for all $i = 1, \ldots, k$. As a consequence, $\phi \ \& \ \phi'$ also disentails $_i.\ (\psi_i \ \& \ U_i \doteq X_i)$. That is, $\phi \ \& \ \phi'$ $_i.\ (\psi_i \ \& \ U_i \doteq X_i)$ is valid. Clearly, this shows that $\phi \ \& \ \bigwedge_{i=1,\ldots,k}$ $_i.\ \psi_i \ \& \ U_i \doteq X$ is satisfiable.

According to Theorem 3, if $\phi \ \&$ $_i.\ (\psi_i \ \& \ U_i \doteq X_i)$ is satisfiable, then ψ_i', the normal form of $\psi_i \ \& \ U_i \doteq X_i$ relatively to ϕ is not a conjunction of equations representing a functional binding.

Thus, one of the three following cases is true, for some V_i $Var(\psi$ $_i')$ bound to some X_i $Var(\phi)$; *i.e.*, $V_i \doteq X_i$ ψ $_i'$:

[(1)] ψ $_i'$ contains a sort constraint on V_i; say, $V_i : s_i$; or,
[(2)] ψ $_i'$ contains two equations on V_i; say, $V_i \doteq X_i \ \& \ V_i \doteq Y_i$; or,
[(3)] ψ $_i'$ contains a feature constraint on V_i, say, $V_i.\ell_i \doteq W_i$.

(1) If $V_i : s_i$ ψ $_i'$, then ϕ contains either no sort constraint on X_i or one of the form $X_i : s_i'$ where $s_i < s_i'$, according to the third condition of Proposition 7. Let $U_{i_j} \doteq X_i$, for $i_j = 1, \ldots, m$, be the family of all equations occurring in the disjuncts binding a local variable U_{i_j} to that same global variable X_i. We add to ϕ the sort constraint $X_i : s_i''$ where s_i'' is some sort which is incompatible with those in the sort constraints $U_{i_j} : s_{i_j}$, and, in case $X_i : s_i'$ ϕ, is furthermore a subsort of s $_i'$, s_i'' s $_i'$.

(2) If $V_i \doteq X_i \ \& \ V_i \doteq Y_i$ ψ $_i'$, and $V_i : s_i$ ψ $_i'$ (otherwise we are in Case (2)), then we add to ϕ' the conjuncts $X_i.\ell_i \doteq Z_i \ \& \ Z_i$ $s \ \& \ Y$ $_i.\ell_i \doteq Z_i' \ \& \ Z_i'$ s $'$. Here s and s' are two incompatible sorts, and the ℓ_i's are pairwise different features which do not occur in ϕ and ψ_i, for $i = 1, \ldots, k$.

(3) Finally, we consider the set I of all indices i, $i = 1, \ldots, k$, for which Case (3), but neither Case (1) nor Case (2) applies. Thus, for i I, ψ $_i'$ contains a feature constraint of the form $V_i.\ell_i \doteq V_i^1$. According to our assumption this constraint is not a redundant conjunct; *i.e.*, there exists a sort s_i such that ψ_i contains, in fact, a conjunct of the form:

$$V_i.\ell_i \doteq V_i^1 \ \& \ V_i^1.\ell_i^2 \doteq V_i^2 \ \& \ \ldots \ \& \ V_i^{n-1}.\ell_i^n \doteq V_i^n \ \& \ V_i^n : s_i,$$

for some n 1. We add to ϕ $'$ the conjunct:

$$X_i.\ell_i^1 \doteq X_i^1 \ \& \ X_i^1.\ell_i^2 \doteq X_i^2 \ \& \ \ldots \ \& \ X_i^{n-1}.\ell_i^n \doteq X_i^n \ \& \ X_i^n : s_i',$$

for some new variables $X_i^1, \ldots, X_i^n$ and for some sort s_i' incompatible with s_i.

If there are several disjuncts ψ_{i_j}' with exactly the same chain of feature constraints starting in a variable bound to the same global variable, then s_i' must be chosen to be incompatible with the sorts in all of these chains. More precisely, if, for $i_j = 1, \ldots, m$, the disjunct ψ_{i_j}' contains the conjunct:

$$V_{i_j}.\ell_i \doteq V_{i_j}^1 \;\&\; V_{i_j}^1.\ell_i^2 \doteq V_{i_j}^2 \;\&\; \ldots \;\&\; V_{i_j}^{n-1}.\ell_i^n \doteq V_{i_j}^n \;\&\; V_{i_j}^n : s_{i_j},$$

then s_i' is chosen as some sort such that $s_{i_j}\ s\ _i' =$ for all $i\ \ _j$, $i_j = 1, \ldots, m$. ∎

5 A general residuation framework

Constraint Logic Programming (CLP) [18], the guarded Horn-clause scheme of Maher (ALPS) [20], Concurrent Constraint Programming (CCP) [23], and Kernel Andorra Prolog [15] (KAP) are recent logic programming frameworks that exploit the separation of relational resolution and constraint solving. They do so to a full extent by being parameterized with respect to an abstract class of constraint systems. In addition, ALPS, CCP, and KAP require a test for entailment and disentailment between constraints. This is needed for advanced control mechanisms such as delaying, coroutining, synchronization, committed choice, and deep constraint propagation. LIFE [6] is a CLP language using a constraint system based on order-sorted feature structures augmented with effective functional dependencies. Evaluating functional dependencies involves constraint entailment/disentailment since passing arguments to functions is done by *matching* as opposed to unification. Thus, LIFE employs a related, but limited, suspension strategy to enforce deterministic functional application.

In this work, extending the guarded Horn-clause scheme of Maher [20], we present an operational and denotational semantics of the general *residuation scheme* used, in a particular way, in LIFE.

The technique of residuation—delaying reduction and enforcing determinism by allowing only equivalence reductions—does not have to be limited to functions. Therefore, we explain it for the general case of relations. Intuitively, the arguments of a relation which are constrained by the guard are its input parameters and correspond to the arguments of a function.

Our scheme defines the denotational and operational meaning of *guarded Horn-clauses*, as formulated by Maher, using logical formulae called *guarded rules*. More precisely, a collection of n guarded Horn-clauses turns out to be syntactic sugar for the conjunction of $n + 1$ guarded rules. The quantification of the local variables (of the guard and the rule body) and their binding to global variables (of the context) turns out to be crucial for this formalism (*cf.*, Section 5.1).

We introduce a *compatibility condition* for guarded rules relaxing the requirement of Maher that the guards of one relation should be mutually exclusive. While this requirement is not part of the general ALPS scheme, it is essential for its completeness results. The compatibility condition is shown to be necessary and sufficient for the existence of a model of guarded Horn-clauses; *i.e.*, of the corresponding conjunction of guarded rules defining a relation.

Since adding guarded rules promotes determinate reduction, the possibility of doing so with possibly overlapping guards is important for efficiency. For example, the *and* predicate on three Boolean arguments can be specified with 11 guarded rules, instead of just two.

In contrast with our semantics, the scheme of Maher sees *guarded* Horn-clauses as defining a relation r by considering them as simple Horn-clauses; *i.e.*, by ignoring the operational meaning of the guard. This amounts to using Clark's completion, yielding a *definite equivalence* [10]. In the scheme of Smolka [24], a relation r is first defined by a definite equivalence defining the semantics of this relation, and only then guarded rules are added, helping to enforce deterministic derivations. Our improvement here is that one can define a predicate solely by ALPS guarded Horn clauses (*i.e.*, the corresponding guarded rules). Also, our guarded-rule reduction scheme extends the one of Smolka. Namely, it avoids useless redundancies in the syntactic formulation of guarded rules, as well as in the operational semantics as will be explained next.[13]

In every guarded-clause language, a resolution step produces a new environment; namely, the conjunction of the old environment, which is the constraint part of the resolvent (the context), and the guard. This conjunction affects the variables in the body (*viz.*, in LIFE, the right-hand side expression of a function definition) after successfully executing the corresponding guard; *i.e.*, it "constrains" them in a semantical sense.

For example, if (in the Herbrand constraint system) $Y = f(a)$ is the context and $Y = f(X)$ is the guard and $Z = X$ is the body, then X is constrained to be equal to a. Practically, the matching proof is done by unification which yields the *instantiation* of the body variable X, $X = a$. In order to compute the new environment, this unification is, of course, not repeated.

The example above can be generalized to constraint systems where the proof of the entailment/disentailment of the guard can be done by a new operational method that we call incremental *relative simplification* of the guard with respect to the context. In this method, the proof of entailment has as a consequence (somewhat like a side-effect) that the conjunction of the context and the guard is in solved form, as if normalized by the constraint solver. For example, relative simplification of the guard $Y = f(X)$ relatively to the context $Y = f(a)$ yields the constraint $X = a$. Hence, we say that an occurrence of the variable X in the body is then *instantiated*.

In contrast with Maher's and Smolka's, our scheme captures the practically relevant case where the variables in the body are already instantiated (in the operational sense above) through the corresponding guard's entailment proof. In particular, as made explicit in Section 4 this applies to the order-sorted feature (OSF) constraint system used in LIFE. So, one thing our scheme brings out formally is the justification and accommodation of the implementer's natural idea that repeated constraint-solving work should be avoided.

Independently of its benefits when used in a guarded language, relative simplification is an implementation strategy for entailment/disentailment proofs. As such, it formalizes and

[13] We mean "useless redundancy," not as a pleonasm, but as a deliberate opposition to "useful redundancy" serving a pragmatic purpose; *cf.*, Footnote 17.

justifies the standard approach of proving matching by doing unification and checking the bindings. Furthermore, it is operationally more powerful since it is incremental; *i.e.*, no redundant work is done. For example, the test of matching through unification is *not* incremental; bindings of global variables are effected for each test and have to be undone afterwards.

This section is organized as follows. In Section 5.1, we present our formulation of guarded Horn-clauses and guarded rules and establish their operational and denotational semantics. In Section 5.2, we briefly consider incremental relative-simplification systems in general. We exhibit some properties which indicate how they might be constructed from a unification system, or more generally, from a constraint solver. In Section 5.3, we put the results of the two previous sections together, to derive the operational semantics of residuation. In Section 6, we show the use of the general scheme on the specific instance of LIFE's functional applications.

5.1 Guarded Horn-clauses and guarded rules

We assume a ranked alphabet $\mathcal{R}$ of relational symbols. A relational atom is an expression of the form $r(X_1, \ldots, X_n)$ where $\in \mathcal{R}$ and the X_i's are mutually distinct variables.

Also, we assume a class of logical formulae (called constraints, noted ϕ, ψ, ..., closed under conjunction and including the *false* constant $\perp$) and a model or a class of models (possibly specified by axioms), to which satisfiability and validity will refer in the following.

A *guarded Horn-clause* is of the form H :- $G \parallel B$., where H, the head, is a relational atom; G, the guard, is a constraint formula; and, B, the body, is of the form $R \ \& \ \phi$, where R, the relational part, is a (possibly empty) conjunction of relational atoms, and ϕ, the constraint part, is a (possibly *true*) constraint formula. In the case of constraint systems with a relative simplification system, the guard G can be a conjunction of positive and negated constraints. We first consider the case where G is a conjunction of positive constraints.

Here is an example of a guarded Horn-clause defining deterministic list concatenation:

$$\begin{aligned} &\mathit{concat}(X, Y, Z) \text{ :- } X : \mathit{nil} \parallel Y \doteq Z. \\ &\mathit{concat}(X, Y, Z) \text{ :- } X : \mathit{cons} \ \& \ X.\mathit{hd} \doteq H \ \& \ X.\mathit{tl} \doteq T \parallel \\ &\qquad\qquad Z : \mathit{cons} \ \& \ Z.\mathit{hd} \doteq H \ \& \ Z.\mathit{tl} \doteq L \ \& \ \mathit{concat}(T, Y, L). \end{aligned}$$

Since any constraint system can be trivially augmented to express tuples,[14] we may assume the relational symbol r in the head to be a unary predicate. This amounts to replacing $r(U_1, \ldots, U_n)$:- $G \parallel B$ with $r(U)$:- $U \doteq (U_1, \ldots, U_n) \ \& \ G \parallel B$. Here, the constraint with tuple notation $U \doteq (U_1, \ldots, U_n)$ is just a shorthand for the specific constraint encoding multiple arguments in the system being considered. For instance, in our OSF-constraint system, $U \doteq (U_1, \ldots, U_n)$ stands for $U.1 \doteq U_1 \ \& \ \ldots \ \& \ U.n \doteq U_n$.

[14] Although doing so may increase significantly its expressive power, this is not important in the context of this presentation. Indeed, our considering only unary relations is not properly restrictive, but essentially a notational convenience.

A *guarded rule* is a logical sentence of the form:

$$\forall U \forall \mathcal{U}.\ \big(\ G\ \rightarrow\ \big(\ r(U)\ \leftrightarrow\ \exists \mathcal{V}.\ B\ \big)\ \big).$$

It is important to note that the existential quantification $\exists \mathcal{V}$ of the variables local to the body may *not* be pulled out; *i.e.*, the guarded rule may not be written $(\forall)\ \big(\ G\ \rightarrow\ \big(r(U)\ \leftrightarrow\ B\big)\big)$.

Let $H = r(U)$ where $r \in \mathcal{R}$ and U is a variable. Let $\mathcal{U} = \mathit{Var}(G) - \{U\}$ and $\mathcal{V} = \mathit{Var}(B) - (\mathcal{U} \cup \{U\})$. Then, the guarded Horn-clause H :- $G \parallel B$ corresponds to the above guarded rule. [15]

For example, the guarded rules corresponding to our foregoing definition of *concat* are:

$$\forall U \forall X.\ \big(\ (U.1 \doteq X\ \&\ X : \mathit{nil})\ \rightarrow$$
$$\big(\ \mathit{concat}(U)\ \leftrightarrow\ \exists\{Y, Z\}.\ (U.2 \doteq Y\ \&\ U.3 \doteq Z\ \&\ Y \doteq Z)\ \big)\big).$$

$$\forall U \forall\{X, H, T\}.\ \big(\ (U.1 \doteq X\ \&\ X : \mathit{cons}\ \&\ X.\mathit{hd} \doteq H\ \&\ X.\mathit{tl} \doteq T)\ \rightarrow$$
$$\big(\ \mathit{concat}(U)\ \leftrightarrow\ \exists\{Y, Z, L\}.\ \big(\ U.3 \doteq Z\ \&\ Z : \mathit{cons}\ \&\ Z.\mathit{hd} \doteq H\ \&\ Z.\mathit{tl} \doteq L\ \&$$
$$U.2 \doteq Y\ \&\ \mathit{concat}(T, Y, L)\ \big)\ \big)\ \big).$$

In the first rule, the variable X does not occur in the rule's body; thus, we can write it:

$$\forall U.\ \big(\ \exists X.\ (U.1 \doteq X\ \&\ X : \mathit{nil})\ \rightarrow$$
$$\big(\ \mathit{concat}(U)\ \leftrightarrow\ \exists\{Y, Z\}.\ (U.2 \doteq Y\ \&\ U.3 \doteq Z\ \&\ Y \doteq Z)\ \big)\ \big).$$

In the second rule, the scope of the variables H and T extends over the guard and the body.

A *(constrained) resolvent* $\boldsymbol{R}$ is a (possibly existentially quantified) formula of the form $R\ \&\ \phi$, where R consists of a (possibly empty) conjunction of relational atoms, and ϕ, its context, is a (possibly *true*) constraint formula. In the following, we will consider only the derivation of resolvents without quantification. Indeed, only the matrix of a quantified resolvent is rewritten (adding possibly more quantifications).

We will call the variables in $\mathit{Var}(\boldsymbol{R})$ *global* and denote them generically as X, Y, Z, *etc.* The variables in a rule are called *local*. Except for the case of explicit examples (*e.g.*, *concat*), local variables are generically named U, V, W, *etc.* The variables that are local to the body are within a quantification scope contained in that of those variables that are also in the guard. Local and global variables will always be assumed distinct, by implicit renaming if necessary, so as to avoid capture.

[15] It is interesting at this point to observe that our formulation of guarded rules is different from Smolka's [24] where the guarded rule above is written in the form:

$$\forall U.\ \big(\ \exists \mathcal{U}.\ G\ \rightarrow\ \big(\ r(U)\ \leftrightarrow\ \exists \mathcal{U} \exists \mathcal{V}.\ (G\ \&\ B)\ \big)\ \big).$$

We will compare our formulation to Smolka's in more detail in Section 5.3.

The next proposition characterizes the *reduction* of a resolvent by application of a guarded rule into an equivalent resolvent.

Proposition 10 *Given the guarded rule:*

$\forall U \forall \mathcal{U}.\ (\ G\ \rightarrow\ (\ r(U)\ \leftrightarrow\ \exists \mathcal{V}.\ B\ )\ ),$

the resolvent $\boldsymbol{R}\ =\ R\ \&\ r(X)\ \&\ \phi$ *is equivalent to the derived resolvent:*

$\exists U \exists \mathcal{U} \exists \mathcal{V}.\ (R\ \&\ B\ \&\ \phi\ \&\ G\ \&\ U \doteq X),$

if the context ϕ *of the resolvent entails the guard of the rule; i.e., if:*

$\phi\ \rightarrow\ \exists U \exists \mathcal{U}.\ (G\ \&\ U \doteq X)$

is valid.

Proof: The entailment condition says that the context ϕ is equivalent to its conjunction with the instantiated guard,

$$\begin{array}{ll} \phi & \phi\ \&\ U.\ (G\ \&\ U \doteq X) \\ & U.\ (\phi\ \&\ G\ \&\ U \doteq X). \end{array}$$

The resolvent $\boldsymbol{R}\ =\ R\ \&\ r(X)\ \&\ \phi$ is equivalent to:

$U.\ (R\ \&\ r(U)\ \&\ \phi\ \&\ G\ \&\ U \doteq X).$

Since the variable U and the variables in are universally quantified, the guarded rule can be written as :

$(r(U)\ \&\ G)\quad .\ (B\ \&\ G).$

It follows that $\boldsymbol{R}$ is equivalent to:

$U.\ (R\ \&\ \phi\ \&\ B\ \&\ G\ \&\ U \doteq X).$ ∎

After the application of a rule, local variables become variables of the derived resolvent and are, then (and only then), considered global.

Let us assume that the constraint ϕ entails the guard G. Then, although ϕ is equivalent to $\exists U \exists \mathcal{U}.\ (\phi\ \&\ G\ \&\ U \doteq X)$, the conjunction $B\ \&\ \phi$ is generally *not* equivalent to the quantified formula $\exists U \exists \mathcal{U}.\ (B\ \&\ \phi\ \&\ G\ \&\ U \doteq X)$. Namely, the guard G generally shares variables

with the body B of the guarded rule. Roughly, the conjunction $\phi \;\&\; G \;\&\; U \doteq X$ provides the instantiation of input parameters used in the body B of the guarded rule.[16]

We now consider the case of a guarded Horn-clause where the guard consists of a conjunction of positive and negated constraints.

For example, the guarded Horn-clause:

$$concat(X, Y, Z) \;\texttt{:-}\; \neg(X : cons) \;[\!]\; X : nil \;\&\; Y \doteq Z.$$

corresponds to the *guarded rule*:

$$\forall U.\ (\ \neg\exists X.\ (U.1 \doteq X \;\&\; X : cons)\ \rightarrow$$
$$(\ concat(U)\ \leftrightarrow\ \exists\{Y, Z\}.\ (X : nil \;\&\; U.2 \doteq Y \;\&\; U.3 \doteq Z \;\&\; Y \doteq Z)\ )\ ).$$

Generally, the guarded Horn-clause $H \;\texttt{:-}\; G \;\&\; \bigwedge_{j=1,\ldots,k} \neg G_j \;[\!]\; B.$ corresponds to the guarded rule:

$$\forall U \forall \mathcal{U}.\ (\ G \;\&\; \bigwedge_{j=1,\ldots,k} \neg\exists \mathcal{U}_j.\ G_j\ \rightarrow\ (\ r(U)\ \leftrightarrow\ \exists \mathcal{V}.\ B\ )\ ). \tag{1}$$

We will always assume that the sets $\mathcal{U}_j = Var(G_j) - \{U\}$ are pairwise disjoint, as well as disjoint from $\mathcal{U}$ and from $\mathcal{V}$.

Proposition 11 *Given the guarded rule (1), the resolvent* $\boldsymbol{R} = R \;\&\; r(X) \;\&\; \phi$ *is equivalent to the resolvent:*

$$\exists U \exists \mathcal{U} \exists \mathcal{V}.\ (R \;\&\; B \;\&\; \phi \;\&\; G \;\&\; U \doteq X),$$

if the context ϕ *of the resolvent entails the guard of the rule; i.e., if the implication:*

$$\phi\ \rightarrow\ \exists U \exists \mathcal{U}.\ (G \;\&\; U \doteq X)$$

is valid and the conjunctions:

$$\phi \;\&\; G_j \;\&\; U \doteq X$$

for $j = 1, \ldots, k$ *are unsatisfiable.*

[16] All conjuncts in the guard which do *not* share variables with the body of the guarded rule being applied, may be omitted in the derived resolvent.

Proof: The proof of Proposition 10 can be rephrased by replacing G with the new guard. Under the entailment assumption, the context ϕ is equivalent to ϕ & $_j$. $(G_j$ & $U \doteq X)$, and since G_j does not share variables with B, B & ϕ is equivalent to B & ϕ & $_j$. $(G_j$ & $U \doteq X)$. This means that the conjuncts $_j$. $(G_j$ & $U \doteq X)$ can be omitted from the derived resolvent. ∎

The collection of the guarded Horn-clauses $r(U)$:- $G_i \parallel B_i$ with the same head in a given program stands for the conjunction of the following $n+1$ guarded rules, where $\mathcal{U}_i = Var(G_i) - \{U\}$ and $\mathcal{V}_i = Var(B_i) - (\mathcal{U}_i \cup \{U\})$ for $i = 1, \ldots, n$:

$$\forall U \forall \mathcal{U}_i.\ (\ G_i\ \rightarrow\ (\ r(U)\ \leftrightarrow\ \exists \mathcal{V}_i.\ B_i\ )\ ),$$

for $i = 1, \ldots, n$, and:

$$\forall U.\ (\ \neg\exists \mathcal{U}_1 G_1\ \&\ \ldots\ \&\ \neg\exists \mathcal{U}_n G_n\ \rightarrow\ (\ r(U)\ \leftrightarrow\ \bot\ )\ ).$$

We assume the guards G_i to be of the general form, as in the guarded rule (1). In our examples, the $n+1st$ guarded rule (the "otherwise" rule) is always left implicit.

Whenever they are consistent, the $n+1$ guarded rules above define the predicate r. This follows from the next fact.

Proposition 12 *The following formula is a logical consequence of the guarded rules which stand for the guarded Horn-clauses* $r(U)$:- $G_i \parallel B_i$, $(i = 1, \ldots, n)$*:*

$$\forall U.\ (\ r(U)\ \leftrightarrow\ \bigvee_{i=1,\ldots,n} \exists \mathcal{U}_i \exists \mathcal{V}_i.\ (G_i\ \&\ B_i)\ ). \tag{2}$$

Proof: The proof for the -part of the formula is clear. For the -part we consider the two cases whether or not $r(U)$, and therefore $r(U)$, holds in an interpretation. In the first case, there is nothing to show. In the second case, we use the $n+1st$ guarded rule, the "otherwise" rule, by contraposition. ∎

It is important to observe that this is in contrast with [24], where, conversely, Formula (2) is called a *definite equivalence* and the guarded rules must be its logical consequences.

Not every conjunction of guarded rules has a model. In fact, in order to be a model an interpretation must satisfy the following *compatibility condition*:

$$\bigwedge_{i,\,j=1}^{n} \Big(\ \forall U \forall \mathcal{U}_i \forall \mathcal{U}_j.\ (G_i\ \&\ G_j\ \rightarrow\ (\exists \mathcal{V}_i.\ B_i\ \leftrightarrow\ \exists \mathcal{V}_j.\ B_j)\ )\ \Big). \tag{3}$$

This condition is trivially fulfilled if the guards are mutually exclusive.

Proposition 13 *Every model of the definite equivalence (2) and the compatibility condition (3) is a model of the conjunction of the $n+1$ guarded rules of the form (1), and vice versa.*

Proof: By (2), G_i & $\quad_i$. B_i implies $r(U)$. If G_i & $r(U)$ holds in an interpretation, then, by (2), there exists some j such that G_j & $\quad_j$. B_j holds. But then, by (3), $\quad_i$. B_i holds also. The $n+1$*st* guarded rule is an immediate consequence of (2). The other direction follows from Proposition 12 for (2) and from combining the guarded rules pairwise for (3). ∎

We call a model of a guarded Horn-clause program a model of the conjunctions of guarded rules which stand for the collections of guarded Horn-clauses with the same head in the program.

Corollary 4 *If the compatibility condition is valid, then a guarded Horn-clause program has a least model.*

Proof: It is a well-known fact that a system of predicate definitions such as (2) has a least model extending the model of the theory of the constraint domain (*cf.*, [18, 17]). The statement then follows from the assumption and Proposition 13. ∎

For the sake of completely relating our approach to others, let us mention one idea which is not (yet) implemented in LIFE. Given a program consisting of definite clauses, one can add explicit guarded rules which are logical consequences of the program [24]. Now, assume a relation r declared by the definite clauses $r(X) \leftarrow \exists U_i.\ \phi_i \ \&\ R_i$, $i = 1, \ldots, k$. Thus, the completed form of r is:

$$r(X) \leftrightarrow \bigvee_{i=1}^{k} \left(\exists U_i.\ \phi_i \ \&\ R_i \right).$$

Then, the following guarded rules are always immediate consequences of this definition:

$$\begin{array}{l} \neg\exists U_1.\ \phi_1 \ \&\ \ldots\ \&\ \neg\exists U_{i-1}.\ \phi_{i-1}\ \& \\ \quad \neg\exists U_{i+1}.\ \phi_{i+1} \ \&\ \ldots\ \&\ \neg\exists U_k.\ \phi_k \quad \rightarrow\ \left(r(X) \leftrightarrow \exists U_i.\ R_i \ \&\ \phi_i\right) \end{array}$$

for $i = 1, \ldots, k$. These guarded rules can be left implicit. Although semantically redundant, these additions are of great pragmatic use for efficient reductions. In fact, adding them is paramount to enabling the immediate reduction of a determinate goal; *i.e.*, one whose definition offers only one alternative in its context.[17] This appears to be related to what has been quoted to us as the "Andorra Principle" [15], a strategy of preferentially selecting goals which have at most one alternative, and is a basic principle underlying the Andorra Model [22].

[17] This is an example of a useful redundancy; *cf.*, Footnote 13.

5.2 Incremental relative-simplification systems

If G is a guard of the general form, as in the guarded rule (1), and ϕ is the context of a given resolvent, then we say that the context entails the guard if the validity condition and the unsatisfiability conditions in Proposition 11 are fulfilled. We say that the context disentails the guard if the implication $\phi \rightarrow \neg\exists U \exists \mathcal{U}.\ (G \,\&\, U \doteq X)$ is valid, or if one of the implications $\phi \rightarrow \exists U \exists \mathcal{U}_j.\ (G_j \,\&\, U \doteq X)$ is valid, for $j = 1, \ldots, k$. Again, disentailment is not the negation of entailment; *i.e.*, the two problems are not dual to each other. Thus, a guarded rule system needs to carry out two different tests.

If the context ϕ of a resolvent $\boldsymbol{R}$ entails the guard, then the context of any resolvent derived from $\boldsymbol{R}$ entails the guard, too. In other words, a context can only become stronger in each derivation step; *i.e.*, constraints are added as conjuncts. The same holds for disentailment.

If the context ϕ neither entails nor disentails the guard, there might still be a derivative of $\boldsymbol{R}$ whose context entails, or disentails, the guard. This is why incrementality is important. In the case where both tests fail, for the context ϕ of the current resolvent $\boldsymbol{R}$, the proof which has determined this will be continued by the proof for the strengthened context $\phi \,\&\, \phi'$ of a resolvent $\boldsymbol{R'}$ derived from $\boldsymbol{R}$, instead of starting from scratch. That is, the proof of the guard "stalls" in the context of $\boldsymbol{R}$; the proof of the guard in the context of $\boldsymbol{R'}$ "resumes" it.

The following observation is useful for deriving an entailment test from a constraint normalization system.

Proposition 14 *The context ϕ entails the guard G if and only if the conjunction $\phi \,\&\, (G \,\&\, U \doteq X)$ is equivalent to $\phi \,\&\, G'$ for some formula G' such that G' is valid.*

Proof: If G' is valid, then $\phi \quad G \quad '$ is also valid. Therefore, ϕ is equivalent to $\phi \,\&\, G'$. According to the assumption, $\phi \,\&\, (G \,\&\, U \doteq X) \quad \phi \,\&\, G \quad '$ is valid. Thus, ϕ is equivalent to $\phi \,\&\, (G \,\&\, U \doteq X)$. This shows that $\phi \quad U(G \,\&\, U \qquad \doteq X)$ is valid. For the other direction, it is sufficient to choose $G' = \big((G \,\&\, U \doteq X)\ \phi \quad \big)$. Clearly, then $\phi \,\&\, (G \,\&\, U \doteq X)$ is equivalent to $\phi \,\&\, G'$, and also G' is valid. ∎

The 'only if' direction in this proposition is crucial for practical purposes. Given ϕ and G, the formula G' has to be effectively found, and its validity has to be effectively determined.

In what follows, ϕ and ψ are two constraints where ϕ is a context formula assumed be consistent such that $Var(\psi) \cap Var(\phi) = \emptyset$.

Corollary 5 *If the guard consists of a positive constraint, say ψ, then the context entails the guard, i.e., $\phi \rightarrow \exists U \exists \mathcal{U}.\ (\psi \,\&\, U \doteq X)$ is valid, if and only if the conjunction $\phi \,\&\, \psi \,\&\, U \doteq X$ is equivalent to $\phi \,\&\, \psi'$ for some formula ψ' such that $\exists U \exists \mathcal{U}.\ \psi'$ is valid.*

Proof: The proof is a straightforward rephrasing of the previous proof. ∎

The corollary gives the idea about how one generally intends to obtain the formula G' from Proposition 14. Namely, by applying a suitable constraint normalization system on the conjunct $\phi \;\&\; \psi \;\&\; U \doteq X$ successively, as long as this is possible, without modifying ϕ. Clearly, the main difficulty is completeness; that is, whether under entailment, one can actually derive a constraint $\phi \;\&\; \psi'$ such that $\exists U \exists \mathcal{U}.\ \psi'$ is valid.

Corollary 6 *The context ϕ disentails the guard ψ, i.e., $\phi \;\rightarrow\; \neg\exists U \exists \mathcal{U}.\ (\psi \;\&\; U \doteq X)$ is valid if and only if $\phi \;\&\; \psi \;\&\; U \doteq X$ is equivalent to $\phi \;\&\; \bot$.*

Proof: We only need to note that if:

$\phi \;\&\; U.\ (\psi \;\&\; U \quad \doteq X) \quad \phi \;\&\; \quad '.\ \psi'$

is valid, then also:

$\phi \;\&\; U.\ (\psi \;\&\; U \quad \doteq X) \quad \phi \;\&\; \quad '.\ \psi'.$ ∎

Again, it is clear how one may try to obtain the disentailment proof. Namely, by applying the constraint solver on the conjunct $\phi \;\&\; \psi \;\&\; U \doteq X$ successively, as long as this is possible without modifying ϕ, or until one arrives at $\phi \;\&\; \bot$. Again, the difficulty is completeness. That is, whether under disentailment, one can actually derive $\bot$ in this way.

Definition 8 *We call a* relative-simplification *system a reduction system which, given the context-constraint ϕ and the guard-constraint ψ and the binding $U \doteq X$ of the variable U in ψ to the variable X in ϕ, reduces $\psi \;\&\; U \doteq X$ to a constraint ψ' with $\mathcal{V} = Var(\psi') - Var(\phi)$ such that:*

- *$\exists \mathcal{V}.\ \psi'$ is valid if and only if ϕ entails ψ; i.e., $\phi \;\rightarrow\; \exists U \exists \mathcal{U}.\ (\psi \;\&\; U \doteq X)$ is valid;*
- *$\psi' = \bot$ if and only if ϕ disentails ψ; i.e., $\phi \;\rightarrow\; \neg\exists U \exists \mathcal{U}.\ (\psi \;\&\; U \doteq X)$ is valid.*

Moreover, at each intermediate simplification step deriving a constraint ψ' with $\mathcal{V} = Var(\psi') - Var(\phi)$ the following relative-simplification invariant *holds:*

- *$\phi \;\&\; \exists U.\ (\psi \;\&\; U \doteq X)$ is equivalent to $\phi \;\&\; \exists \mathcal{V}.\ \psi'$.*

Proposition 15 (Confluence of Relative Simplification) *Any relative-simplification system can be transformed into an incremental one simply by closing the simplification relation with respect to the following condition. If ψ simplifies to ψ' relatively to ϕ, and ψ simplifies to ψ'' relatively to $\phi \;\&\; \phi'$, then also:*

- *ψ simplifies to ψ' relatively to $\phi \;\&\; \phi'$, and*

- *ψ' simplifies to ψ'' relatively to ϕ & ϕ'.*

The statement says that any relative-simplification system can be assumed to be already incremental.

Proof: The relative-simplification invariant still holds if one considers every simplification relatively to ϕ also a simplification relatively to ϕ & ϕ'. Namely, if $\phi \quad (\psi \quad \psi \quad ')$ is valid, then so is ϕ & $\phi' \quad (\psi \quad \psi \quad ')$. ∎

Generally, it is not evident how to transform the specification of a non-incremental relative-simplification system (*e.g.*, by rewrite-rules) into an incremental one (*e.g.*, by adding or modifying the rules). Our experience is limited to cases (essentially to the constraint systems over finite or rational first-order [11, 13] or feature trees [7, 25]) where incrementality came for free.

5.3 Operational semantics of residuation

We assume a constraint system with an incremental relative-simplification system as described in the previous section. Let the relation r be specified by n guarded Horn-clauses, each of the form $r(U)$:- $G \parallel B$, corresponding to $n+1$ guarded rules, each of the form (1). Let the guard G be of the form:

$$G = \psi_0 \,\&\, \bigwedge_{j=1,\ldots,k} \neg\exists\mathcal{U}_j.\ \psi_j.$$

Let us consider the hypothetical reduction of the resolvent $\boldsymbol{R} = R \,\&\, r(X) \,\&\, \phi$ to the new resolvent:

$$\boldsymbol{R}' = \exists\mathcal{U}_0'\exists\mathcal{V}.\ (R \,\&\, \phi \,\&\, B \,\&\, \psi_0' \,\&\, \bigwedge_{j=1,\ldots,k} \neg\exists\mathcal{U}_j'.\ \psi_j'),$$

where the constraints $\psi_j \,\&\, U \doteq X$ simplify to ψ_j' relatively to the context ϕ, with $Var(\psi_j) = \mathcal{U}_j$ and $Var(\psi_j') - Var(\phi) = \mathcal{U}_j'$ for $j = 0, 1, \ldots, k$.

Proposition 16 (Correctness of reduction) *The reduction step from the resolvent $\boldsymbol{R}$ to the resolvent $\boldsymbol{R}'$ is always a correct reduction step: $\boldsymbol{R}'$ implies $\boldsymbol{R}$; i.e., all solutions of $\boldsymbol{R}'$ are solutions of $\boldsymbol{R}$.*

Proof: This follows from Proposition 12 and the relative-simplification invariant. ∎

The reduction step from the resolvent $\boldsymbol{R}$ to the resolvent $\boldsymbol{R}'$ is also a complete reduction step: (with Proposition 16) $\boldsymbol{R}$ is equivalent to $\boldsymbol{R}'$. Equivalently,

Proposition 17 (Completeness of reduction) *The solutions of* $\boldsymbol{R}'$ *are* exactly *the solutions of* $\boldsymbol{R}$*, if:*

- $\exists \mathcal{U}'_0.\ \psi'_0$ *is valid, and,*
- $\psi'_j = \bot$*, for each* $j = 1, \ldots, k$*.*

Then, $\boldsymbol{R}'$ *is equivalent to* $R \,\&\, \phi \,\&\, \psi'_0 \,\&\, B$*.*

Proof: This follows from Proposition 11 and the relative-simplification invariant. ∎

In the case of relative-simplification systems based on constraint solvers (*e.g.*, implementing unification), $\phi \,\&\, \psi'_0$ is already essentially the solved form of $\phi \,\&\, \psi_0$. This is the case for OSF-constraints (*cf.*, Section 4). That is, our scheme captures the practically important case when the conjunction of the context and the guard has already been solved through the guard proof.

For comparison, let us consider the guarded-rule reduction defined by Smolka in [24]. There, the "commit condition" is that the conjunction of the context ϕ and the negated guard $\neg\psi$ be a constraint that simplifies to $\bot$, the inconsistent constraint. Under this condition, the resolvent $\phi \,\&\, r(x) \,\&\, R$ reduces to: $\phi'' \,\&\, R' \,\&\, R$ if the (renamed) guarded rule $\psi \rightarrow (r(X) \leftrightarrow \phi' \,\&\, R')$ is used, and the constraint $\phi \,\&\, \phi'$ simplifies to ϕ''.

A consequence of this on the syntactic formulation of guarded rules is that, in Smolka's scheme, the part of the guard which constrains variables in the body must be repeated in the constraint ϕ' in the body of the guarded rule. That is, the guarded rule:

$$\forall U \forall \mathcal{U}.\ (\ G \ \rightarrow\ (\ r(U)\ \leftrightarrow\ \exists \mathcal{V}.\ B\ )\ ).$$

must be written in the form:

$$\forall U.\ (\ \exists \mathcal{U}.\ G\ \rightarrow\ (\ r(U)\ \leftrightarrow\ \exists \mathcal{U} \exists \mathcal{V}.\ (G \,\&\, B)\ )\ ).$$

As a result, in Smolka's operational semantics of guarded-rule reduction is that the simplification of the constraint $\phi \,\&\, \phi'$ does more work than is necessary after relative simplification. Namely, it must repeat the simplification of the conjunction of the context and the guard.

Thus, for constraint systems with relative simplification, our formulation has an advantage in efficiency, although it is semantically equivalent to Smolka's. In our scheme, it is only necessary to normalize the constraints in B, but not those in G, in conjunction with the resolvent's context in the case where that guarded rule is applied.

The next proposition considers the case of disentailment. Here, of course, no instantiation is effectuated. It states that the reduction step from resolvent $\boldsymbol{R}$ to the resolvent $\boldsymbol{R}'$ can be excluded whenever $\boldsymbol{R}'$ is equivalent to $\bot$. Equivalently,

Proposition 18 (Failure of reduction) *The set of solutions of* $\boldsymbol{R}'$ *is empty, if:*

- $\psi_0' = \bot$, *or,*
- $\exists \mathcal{U}_j'.\ \psi_j'$ *is valid, for at least one of* $j = 1, \ldots, k$.

Proof: This follows from Proposition 12 and Definition 8. ∎

The foregoing propositions might suggest several possibilities for fine details of the operational semantics concerning resolvents with residuations; *i.e.*, relational atoms $r(X)$ for which none of the guards of the $n+1$ guarded rules for r is entailed. The answer of the query could be given by the residuated resolvent; *i.e.*, with the relational atom $r(X)$. Or, in order to make the answer more refined, it could be given by the disjunction of all resolvents $\boldsymbol{R}'$ which are not equivalent to $\bot$.

The constraint part of such a resolvent $\boldsymbol{R}'$ can be further tested for satisfiability. Possibly, it contains negated constraints. Assuming that the constraint system has the independence property (*cf.*, Theorem 4), such a constraint part can be tested for satisfiability by testing entailment of each of the negated constraints by the positive constraint.

6 Functional application over ψ-terms

We now show how the foregoing general residuation scheme can be used to explain functional application over ψ-terms. A ψ-term is a constrained data structure. Hence, as an expression, it can be further constrained by being conjoined with other functional and relational constraints. We will call such an expression a *constrained* ψ*-term*. For example, $X : cons(tl \Rightarrow T : list)$ & $length(T) \doteq L$ & $L : even$ is a constrained ψ-term specifying lists of odd length.

A constrained ψ-term is an expression of the form ψ & C where ψ is a ψ-term and C is a possibly empty conjunction of OSF-constraints and relational atoms.[18]

In LIFE a function f is defined by:

$$\begin{array}{l} f(p_1) \rightarrow e_1. \\ \quad \vdots \\ f(p_n) \rightarrow e_n. \end{array}$$

where $p_1, \ldots, p_n$ are ψ-terms and $e_1, \ldots, e_n$ are constrained ψ-terms. We assume that the variables occurring in each rule $f(p_i) \rightarrow e_i$ are different. We shall use $\mathcal{U}_i$ for $Var(p_i)$ and $\mathcal{V}_i$ for $Var(e_i)$. Again, for ease of notation and without loss of generality, we consider only the case of unary function symbols f.

[18] The concrete syntax in LIFE for a constrained ψ-term is: $\psi \mid C$. This is read as "ψ such that C."

The above form of function definition is in fact syntactic sugar for a collection of n guarded Horn-clauses of the form:

$$f_r(U,V) \text{ :- } U : p_i \;\&\; \bigwedge_{j=1}^{i-1} \neg\, U : p_j \;\|\; V : e_i.$$

for $i = 1, \ldots, n$; and thus, as seen in the previous section, for a conjunction $n+1$ guarded rules. The symbol f_r is a binary relation symbol associated to f. We shall also use the functional constraint notation $Y \doteq f(X)$ as sugaring for the relational atom $f_r(X, Y)$, and the constraint $Y : f(t)$ with the functional expression $f(t)$ as sugaring for $\exists X.\ X : t \;\&\; Y \doteq f(X)$.

We have everything ready now, with the general scheme of residuation of Section 5, to explain the operational semantics of functional reduction in LIFE as a matter of instance. Indeed, that scheme is sufficiently general to account for argument matching seen as constraint entailment and priority of rule order thanks to negative constraints imposing disentailment of previous patterns.

We make this explicit in the form of the following two propositions. They are immediate instances of Proposition 11 and Proposition 17, respectively.

Proposition 19 *The resolvent* $\;\&\; \phi \;\&\; Y : f(t)$ *is equivalent to the resolvent:*

$$\exists X \exists \mathcal{U}_i \exists \mathcal{V}_i.\ R \;\&\; \phi \;\&\; X : t \;\&\; Y : e_i \;\&\; X : p_i$$

if the context $\phi \;\&\; X : t$ *disentails the OSF-constraints* $X : p_j$ *for* $j = 1, \ldots, i-1$*, and if it entails the OSF-constraint* $X : p_i$*. That is, if the conjunctions* $\phi \;\&\; X : t \;\&\; X : p_j$ *are unsatisfiable for* $j = 1, \ldots, i-1$*, and the implication* $\phi \;\&\; X : t \;\rightarrow\; \exists \mathcal{U}_i.\ X : p_i$ *is valid.*

Proposition 20 *If, for* $j = 1, \ldots, i$*, the OSF-constraint* $X : p_j$ *simplifies to the OSF-constraint* ψ_j *relatively to* $\phi \;\&\; X : t$ *such that* $\psi_1 = \bot, \ \ldots\ , \psi_{i-1} = \bot$*, and* ψ_i *is a functional binding,*[19] *then the resolvent* $\;\&\; \phi \;\&\; Y : f(t)$ *is equivalent to the resolvent:*

$$\exists X \exists \mathcal{U}_i \exists \mathcal{V}_i.\ R \;\&\; \phi \;\&\; X : t \;\&\; Y : e_i \;\&\; \psi_i.$$

6.1 Functional application in the ψ-term calculus

Next, we express functional application in the framework of the calculus of subsumption and unification of ψ-terms.

We use a fact that follows directly from Proposition 2 and Proposition 3. Namely, the implication $X : t \rightarrow \exists \mathcal{U}_i.\ X : p_i$ is valid if and only if the ψ-term t is subsumed by the ψ-term

[19] Recall, from Section 4, that a functional binding is a conjunction of variable equalities $U_i \doteq X_i$, $i = 1, \ldots, n$ where all the variables U_i are mutually distinct.

p_i. The OSF-constraint $X : t \;\&\; X : p_i$ is unsatisfiable if and only if the ψ-term t is non-unifiable with the ψ-term p_i.

We will say that the equality $t \doteq p$ between two ψ-terms is satisfied under a valuation α in an interpretation $\mathcal{A}$, if and only if $\mathcal{A}, \alpha \models t \doteq p$ iff $[\![t]\!]^{\mathcal{A},} \quad = [\![p]\!]^{\mathcal{A},\alpha}$; *i.e.*, if the two ψ-terms have the same denotation under α.

Proposition 21 *If the ψ-term t is non-unifiable with the ψ-terms $p_1, \ldots, p_{i-1}$ and if it is subsumed by the ψ-term p_i, then the functional expression $f(t)$ is equivalent to the expression e_i constrained by $t \doteq p_i$. Formally,*

$$Y : f(t) \;\leftrightarrow\; \exists \mathcal{U}_i \exists \mathcal{V}_i .\; Y : e_i \;\&\; t \doteq p_i \tag{4}$$

is valid. If t is non-unifiable with the ψ-terms $p_1, \ldots, p_n$, then $f(t)$ is equivalent to $\bot$.

Proof: The statement follows from Proposition 19 and the fact that , $\alpha = X.\ (X : t \;\&\; X : p \quad _i)$ if and only if , $\alpha = t \quad \doteq p_i$. ∎

6.2 Endomorphisms and functional application

We have related functional reduction to the view of ψ-terms as constraints and as sets. In order to be complete with respect to the three (logical, term-as-set, and algebraic) characterizations of the information contents of ψ-terms, we now give an algebraic characterization of functional application as graph pattern-matching. This view generalizes the familiar notion of matching by computing substitutions.

If a function is defined over first-order terms, say, in the form $f(p) = e$, then the function applied to the term t yields the expression $\sigma(e)$ if the term t is matched by the pattern p via the matching substitution σ; *i.e.*, $f(t) = \sigma(e)$ if $\sigma(t) = p$. This is not so obvious for ψ-terms. Let us take, for example, the identity function on ψ-terms, which is defined in the form $f(X : \top) = (X : \top)$. When applied to the ψ-term $t \;=\; (X : s(\ell \Rightarrow X' : s))$, the function returns the same ψ-term. However, this does not exhibit a substitution σ such that $\sigma(X : \top) = (X : s(\ell \Rightarrow X' : s))$.

Recall that an approximation ordering $\sqsubseteq$ on ψ-terms is induced by the ordering on , the OSF-graph algebra (*cf.*, Section 3.5). An endomorphism γ is said to be principal in a set of endomorphisms if for every endomorphism γ' in this set, there exists an endomorphism ρ such that $\gamma \circ \rho = \gamma'$.

We define the application of an endomorphism on a constrained ψ-term of the form $\psi \;=\; \psi_0 \;\&\; \bigwedge_{k=1}^{m} (r_k(Y_k) \;\&\; Y_k : \psi_k)$ by:

$$\gamma(\psi) \;=\; \gamma(\psi_0) \;\&\; \bigwedge_{k=1}^{m} (r_k(Y_k) \;\&\; Y_k : \gamma(\psi_k)).$$

Let $f(p) \rightarrow e$ define the function f, and let t be a ψ-term such that $p \sqsubseteq t$. Let γ be a principal OSF-endomorphism among all those that map p into t. The next proposition states precisely the following fact: applying the rule means that $f(t) = f(\gamma(p)) = \gamma(e) = \gamma(f(p))$. In other words, principal OSF-endomorphisms preserve functional application (*i.e.*, functional evaluation and OSF-approximation commute).

Proposition 22 *If no ψ-term is approximated by both t and p_j for $j = 1, \ldots, i-1$, and t is approximated by p_i, then the functional expression $f(t)$ reduces to the ψ-term $\gamma(e_i)$, where γ is a principal endomorphism mapping p_i on t; i.e.,*

$$f(t) = \gamma(e_i), \quad \textit{if } \gamma(p_i) = t. \tag{5}$$

If no ψ-term is approximated by both t and p_i for $i = 1, \ldots, n$, then the functional ψ-term $f(t)$ is $\bot$.[20]

Proof: By Proposition 2, we know that the conditions in Proposition 22 on the OSF-graphs are equivalent to the conditions in Proposition 21 on the corresponding ψ-terms. In particular, this implies the existence of the principal endomorphism γ with $\gamma(p_i) = t$. From Propositions 3 and 4, Page 25, we know that $X : t \;\&\; X : p_i$ is equivalent to $X : \gamma(p_i) \;\&\; \phi$ where ϕ is a functional binding (of variables of p_i to variables of t). Moreover, the equivalence:

$$\bigwedge_{k=0}^{m} Y_k : \psi_k \;\&\; X : t \;\&\; X : p_i \qquad \bigwedge_{k=0}^{m} Y_k : \gamma(\psi_k) \;\&\; X : \gamma(p_i) \;\&\; \phi$$

is valid. Now, if e_i is of the form $\psi_0 \;\&\; \bigwedge_{k=1}^{m}(r_k(Y_k) \;\&\; Y_k : \psi_k)$, then $Y_0 : e_i \;\&\; X : t \;\&\; X : p_i$ is equivalent to $\gamma(Y_0 : e_i) \;\&\; X : \gamma(p_i) \;\&\; \phi$. Up to existential quantification of new variables occurring only in ϕ, this formula is equivalent to $\gamma(Y_0 : e_i) \;\&\; X : \gamma(p_i)$. Thus, Equation (5) follows from Proposition 21. ∎

The proposition above justifies the intuition of functional application over ψ-terms. The variables of the pattern p_i in the function definition are instantiated by variables of the calling term t, together with their sorts and their attached subterms, so that p_i becomes syntactically equal to t; then the variables in the expression e_i are instantiated accordingly, so that e_i becomes the expression which rewrites $f(t)$.

The variables in e_i which are not shared by the pattern p_i must not be instantiated; this is the reason why we require the endomorphism mapping p_i on t to be principal.

For example, let the function f be defined in the form $f(U : \top) \rightarrow U' : \top(\ell \Rightarrow U : \top)$. Applied to the ψ-term $t = X : s(\ell \Rightarrow X' : s)$, the function returns $f(t) = U' : \top(\ell \Rightarrow (X : s(\ell \Rightarrow X' : s)))$. Here, the principal endomorphism γ maps $(U : \top)$ on $(X : s(\ell \Rightarrow X' : s))$ and is the identity elsewhere. In particular, γ does not unnecessarily refine the sort of U'.

[20] Note that, in (5), we use the metalogical equal sign ($=$), as opposed to the logical one ($\doteq$). This means that in any resolvent we can replace the expression on the one side by the expression on the other side and obtain a resolvent which is equivalent up to existential quantification of new variables.

The endomorphic approximation ordering is very interesting when used on the graph representations of ψ-terms. It is in fact an immediate generalization of first-order term matching. More conveniently, if a graph ψ_1 approximates a graph ψ_2 with an endomorphism γ, this approximation is characterized exactly by a mapping $\gamma_{\mathcal{V}} : \mathit{Var}(\psi_1) \mapsto \mathit{Var}(\psi_2)$ that can be constructed inductively as follows:[21]

- $\gamma_{\mathcal{V}}(\mathit{Root}(\psi_1)) = \mathit{Root}(\psi_2)$;
- for every $X_1 \in \mathit{Var}(\psi_1)$ and for every feature $\ell \in \mathcal{F}$ such that $\ell(X_1) = Y_1$, then $\gamma_{\mathcal{V}}(Y_1) = \ell(\gamma_{\mathcal{V}}(X_1))$.

It is clear that this construction is well-defined by the very definition of endomorphic approximation. In fact, a mapping such as $\gamma_{\mathcal{V}}$ can be extended to all variables $\gamma_{\mathcal{V}} : \mathcal{V} \mapsto \mathcal{V}$; it can be defined simply from γ as $\gamma_{\mathcal{V}}(\mathit{Root}(\psi)) = \mathit{Root}(\gamma(\psi))$, for all ψ in .

For example, provided that $\mathit{married\_person} < \mathit{person}, \mathit{smith} < \mathit{name}, \mathit{male} < \mathit{gender}$, and $\mathit{female} < \mathit{gender}$, then the term:

$$
\begin{array}{l}
X_1 : \mathit{person}(\mathit{lastname} \Rightarrow X_2 : \mathit{name}, \\
\qquad \mathit{spouse} \Rightarrow X_3 : \mathit{person}(\mathit{lastname} \Rightarrow X_2, \\
\qquad\qquad\qquad \mathit{spouse} \Rightarrow X_4 : \mathit{person}), \\
\qquad \mathit{sex} \Rightarrow X_5 : \mathit{gender})
\end{array}
$$

approximates the term:

$$
\begin{array}{l}
Y_1 : \mathit{married\_person}(\mathit{lastname} \Rightarrow Y_2 : \mathit{smith}, \\
\qquad\qquad \mathit{spouse} \Rightarrow Y_3 : \mathit{married\_person}(\mathit{lastname} \Rightarrow Y_2, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{sex} \Rightarrow Y_4 : \mathit{female}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \mathit{spouse} \Rightarrow Y_1), \\
\qquad\qquad \mathit{sex} \Rightarrow Y_5 : \mathit{male})
\end{array}
$$

with the endormorphic mapping of variables: $\gamma_{\mathcal{V}}(X_1) = Y_1$, $\gamma_{\mathcal{V}}(X_2) = Y_2$, $\gamma_{\mathcal{V}}(X_3) = Y_3$, $\gamma_{\mathcal{V}}(X_4) = Y_1$, and $\gamma_{\mathcal{V}}(X_5) = Y_5$.

As for a matching algorithm, the basic unification rules of Figure 3 are sufficient. Evidently, if the basic unification yields $\bot$, then this shows disentailment. Otherwise, we will exhibit conditions on the obtained variable bindings which characterize entailment.

First, observe that after normalizing a consistent OSF-term using Rules (B.1)–(B.5), the variable equalities left in the solved form generate an equivalence relation on the variables. We call *variable coreference* this equivalence relation.

Given two ψ-terms ψ_1 and ψ_2, to decide whether $\psi_2 \sqsubseteq \psi_1$ and, if so, to compute the principal endomorphic mapping $\gamma_{\mathcal{V}}$ from $\mathit{Var}(\psi_2)$ to $\mathit{Var}(\psi_1)$ (the "matching substitution"), we proceed as follows:

[21] Given an OSF-graph ψ, we use the notation $\mathit{Root}(\psi)$ to designate its root variable, $\mathit{Sort}_{\psi}(X)$ to designate the sort of the variable X in ψ, and $\ell_{\psi}(X) = Y$ to express the fact that ψ has an arc labeled ℓ between nodes X and Y. (When no ambiguity may arise, we omit the subscript ψ.)

- [1.] let ψ_1' be the ψ-term obtained from ψ_1 by completing it with new variables sorted with $\top$ at any path occurrence of ψ_2 that is *not* in ψ_1;
- [2.] let ϕ be the normal form of $Root(\psi_1) \doteq Root(\psi_2)$ & ψ_1' & ψ_2;
- [3.] if ϕ is not $\bot$ then let γ_1 (resp., γ_2) be the canonical surjection of $Var(\psi_1')$ (resp.,$Var(\psi_2)$) onto the coreference classes of ϕ; *i.e.*, the function that maps a variable to its coreference class.

Then,

Theorem 5 *$\psi_2 \sqsubseteq \psi_1$ with principal OSF-endomorphism γ if and only if ϕ is not $\bot$ and γ_1 is a sort-preserving bijection.*[22] *Then, $\gamma_\nu = \gamma_1^{-1} \circ \gamma_2 : Var(\psi_2) \mapsto Var(\psi_1)$ is the corresponding endomorphic variable mapping.*

Proof: First of all, let us observe that completing ψ_1 into ψ_1' with feature occurrences of ψ_2 with new -sorted variables is an equivalence transformation thanks to totality of features. In other words, $\psi_{\ 1}$ and ψ_1' are equivalent. Let $_1 = Var(\psi_1')$ and $_2 = Var(\psi_2)$. The formula ϕ is of the form ψ & ε where ψ consists only of sort and feature constraints and ε consists only of equality constraints. These variable equalities generate the coreference relation. Let $[X]$ denote the coreference class of X.

If γ_1 is a sort-preserving bijection, then for every variable X of ϕ, $\gamma_1^{-1}([X])$ is the unique variable of ψ_1' which is element of this coreference class. Then, we can transform ϕ into an equivalent formula ϕ by replacing every variable X by $\gamma_1^{-1}([X])$ in ψ and replacing ε by $\varepsilon' = \bigwedge_{X \in \mathcal{X}_2} X \doteq \gamma_\nu(X)$. Note that this is an equivalence preserving transformation since $\bar{\phi}$ is, by construction, of the form ψ_1' & ε' and the coreference relation generated by ε and ε' are identical. It is important to realize that this statement would not be true if we had used ψ_1 instead of ψ_1'. Indeed, then, $\bar{\phi}$ would have been of the form ψ_1 & ψ' & ε' where ψ' consisted of additional feature constraints corresponding to occurrences of ψ_2 missing in ψ_1.

Clearly, it is true that $_1.\ (\psi_1'$ $_2.\ (\psi_1'$ & $\varepsilon'))$. This shows that $_1.\ (\psi_1'$ $_2.\ (Root(\psi_1) \doteq Root(\psi_2)$ & ψ_1' & $\psi_2))$ and, thus, $_1.\ (\psi_1'$ $_2.\ (Root(\psi_1) \doteq Root(\psi_2)$ & $\psi_2))$ is valid, and thus $\psi_2\ \psi_{\ 1}$.

Conversely, if $\psi_2\ \psi_{\ 1}$, then also $_1.\ (\psi_1'$ $_2.\ (Root(\psi_1) \doteq Root(\psi_2)$ & $\psi_2))$ is valid, and, thus, also $_1.\ (\psi_1'$ $_2.\ \phi)$. But this means that ϕ does not contain equalities binding two variables of ψ_1 to each other, and that ϕ does not contain a sort constraint stronger than the one in ψ_1 on the (same or corresponding) variable of ψ_1. ∎

Note that the completion of ψ_1 with occurrences from ψ_2 done in Step 1 is necessary to determine the bijection γ_1, and thus the mapping γ_ν, with no loss of information. For example, if $\psi_1 = f(a, h)$ and $\psi_2 = f(X, h(X))$, then $\psi_2 \not\sqsubseteq \psi_1$. However, using ψ_1 instead of the completed $\psi_1' = f(a, h(U))$ and normalizing does result in a sort-preserving bijection while, using ψ_1', it does not.

6.3 Semantics of functional application

If a function is defined over ψ-terms, then this means that it can be applied to set-denoting objects to return set-denoting objects. We will first consider the meaning of pointwise

[22] By sort-preserving, we mean: $\forall V \in Var(\psi_1), Sort_{\psi_1}(V) = Sort_\phi(\gamma_1(V))$.

functional application given an OSF-algebra $\mathcal{A}$ and a valuation α in $\mathcal{A}$. This extends naturally to the meaning of functional application on sets, given just an OSF-algebra $\mathcal{A}$.

The function $f^{\mathcal{A},\alpha}$ maps elements to elements of the domain $D^{\mathcal{A}}$ of $\mathcal{A}$. In fact, $f^{\mathcal{A},\alpha}$ describes a partial, namely at most n-point function:

$$f^{\mathcal{A},\alpha}(d) \;=\; d' \;\; \textit{if} \;\; d \in [\![p_i]\!]^{\mathcal{A},\alpha} - \bigcup_{j=1}^{i-1} [\![p_j]\!]^{\mathcal{A}} \;\; \textit{and} \; d' \in [\![e_i]\!]^{\mathcal{A},\alpha} \;\; \textit{for some } i.$$

The ψ-terms $p_1, \ldots, p_n$ are not necessarily disjoint. Instead of using an explicit negation operator, we give a deterministic meaning to the top-down order in the function definition in the above way. That is, we define the function $f^{\mathcal{A},\alpha}$ for only those valuations α where $[\![p_i]\!]^{\mathcal{A},\alpha}$ is disjoint from $[\![p_1]\!]^{\mathcal{A}}, \ldots, [\![p_{i-1}]\!]^{\mathcal{A}}$. Implicitly, we make the ψ-terms p_i disjoint by giving them the denotations $[\![p_i]\!]^{\mathcal{A},\alpha} - ([\![p_1]\!]^{\mathcal{A}} \cup \ldots \cup [\![p_{i-1}]\!]^{\mathcal{A}})$, for $i = 1, \ldots, n$. Note that, for two ψ-term ψ_1 and ψ_2, the set $[\![\psi_1]\!]^{\mathcal{A},\alpha}$ is disjoint with $[\![\psi_2]\!]^{\mathcal{A},\alpha} - [\![\psi_1]\!]^{\mathcal{A}}$, but generally not with $[\![\psi_2]\!]^{\mathcal{A},\alpha} - [\![\psi_1]\!]^{\mathcal{A},\alpha}$. For example, take $\psi_1 = X : \textit{int}$ and $\psi_2 = Y : \textit{real}$, and define some α where $\alpha(X) = 3,\ \alpha(Y) = 4$.

The function $f^{\mathcal{A}}$, *i.e.*, f interpreted in $\mathcal{A}$, maps elements (and, by extension, sets) to subsets of the domain $D^{\mathcal{A}}$,

$$f^{\mathcal{A}}(d) \;=\; \{d' \mid \exists \alpha \in \textit{Val}(\mathcal{A}).\, f^{\mathcal{A},\alpha}(d) = d'\}.$$

The denotation of the *functional application* of f on the ψ-term t under a valuation α in the interpretation $\mathcal{A}$ is:

$$[\![f(t)]\!]^{\mathcal{A},\alpha} \;=\; f^{\mathcal{A}}([\![t]\!]^{\mathcal{A},\alpha}).$$

Thus, $\mathcal{A}, \alpha \models Y : f(X : t)$ if and only if $\alpha(X) \in [\![t]\!]^{\mathcal{A},\alpha}$ and $\alpha(Y) = f^{\mathcal{A},\beta}(\alpha(X))$ for some $\beta \in \textit{Val}(\mathcal{A})$.

The denotation of the functional application of f on the ψ-term t in the interpretation $\mathcal{A}$ is $[\![f(t)]\!]^{\mathcal{A}} \;=\; f^{\mathcal{A}}([\![t]\!]^{\mathcal{A}})$.

Example 6.1 We define the identity function *id* on ψ-terms by the rule: $\textit{id}(X : \top) \rightarrow X : \top$. Then, $\textit{id}^{\mathcal{A}}(D) = D$ for any subset $D \subseteq D^{\mathcal{A}}$. If we confuse singletons and their elements, we may write $\textit{id}^{\mathcal{A}}(d) = d$ for elements d of the domain of $\mathcal{A}$. If s is any sort, then $[\![\textit{id}(X : s)]\!]^{\mathcal{A}} = [\![X : s]\!]^{\mathcal{A}} = s^{\mathcal{A}}$. In fact, the denotation of the function *id* applied on any ψ-term is equal to the denotation of the ψ-term. The denotation under a given valuation α is the value of the element on which the function is applied, $[\![\textit{id}(X : \top)]\!]^{\mathcal{A},\alpha} = [\![X : \top]\!]^{\mathcal{A},\alpha} = \{\alpha(X)\}$.

Example 6.2 We define the function *any* by the rule: $\textit{any}(X : \top) \rightarrow Y : \top$. The application of this function on a ψ-term ψ yields always the sort $\top$, $\textit{any}(\psi) = Y : \top = \top$.

Note that $[\![any]\!]^{\mathcal{A},\alpha}(\alpha(X)) = \alpha(Y)$. Thus, $any^{\mathcal{A}}(D) = D^{\mathcal{A}}$ for any subset $D \subseteq D^{\mathcal{A}}$, and $[\![any(X : s)]\!]^{\mathcal{A},\alpha} = D^{\mathcal{A}}$.

Example 6.3 For a fixed sort s, we define the function $sort_s$ by the rule: $sort_s(X : s) \to X : \top$. Now, $sort_s^{\mathcal{A}}([\![X : \top]\!]^{\mathcal{A},\alpha})$ yields $\{\alpha(X)\}$ if $\alpha(X) \in s^{\mathcal{A}}$ and $\emptyset$ otherwise. This function "type-checks" the variable X. Operationally, this means that the function call $sort_s(X)$ will residuate until X is known to be in the sort s and then fire; or, until it is known to be out of the sort s and fails.

What about the interpretation of the syntactic object f in an OSF-algebra $\mathcal{A}$? The function f is generally not completely specified in that not *one* function is singled out in every interpretation $\mathcal{A}$. Indeed, LIFE calculates with approximations of functions, just as it does for values of the universe. Thus, f denotes, under each interpretation $\mathcal{A}$, the set of all partial functions $\varphi : D^{\mathcal{A}} \mapsto D^{\mathcal{A}}$ such that, if $\varphi(d) = d'$, then there exists an $\mathcal{A}$-valuation α such that $f^{\mathcal{A},\alpha}(d) = d'$.

7 Conclusion

The original motivation of this paper was to provide a formal account of the precise manner in which functional application is used in the resolution scheme of LIFE. This involved doing three things essentially. Firstly, we have developed a correct and complete operational scheme for testing entailment and disentailment of order-sorted feature constraints. To that end, we have introduced a general technique, that we dubbed relative simplification, that amounts to normalization of a formula in the context of another. Secondly, we have developed a general residuation framework for guarded Horn-clauses over arbitrary constraint systems with an incremental relative simplification system. Doing so, we have given a logical reading of guarded rules as first-order formulae and exhibited operational and semantical properties of the framework. Lastly, we used this general residuation framework on the particular instance of functional application over the order-sorted features terms of LIFE. In particular, we characterized functional application over LIFE's structures in terms of their logical, set-theoretic, and algebraic accounts.

As for perspectives, one important issue begs the question. Namely, it would be interesting to build function denotations into the OSF-models. Indeed, while the framework of this paper gives a natural meaning to function symbols, it does not consider the latter as a "first-class" objects—*i.e.*, the OSF-interpretations used here are not functionally complete. We plan to study a means of construction using well-known techniques *à la* Dana Scott to extend domains of OSF-algebras to be functionally complete. That should involve the machinery of classical Scott-style constructions. Another dimension to that endeavor would be that of seeing all functions as features of objects. This intriguing perspective could indeed lead to interesting model constructions.

Another avenue for further work on the foundations that we have just cast is the use of the new discipline for procedure parameter-passing in concurrent systems described as "call-by-constraint-entailment." This is along the lines of what has been proposed in [20] and [23], and realized to some extent in AKL [15]. The novelty that our scheme suggests is the possibility to derive automatically an effective means to realize this from the operational semantics of a given constraint-solver. Then, it should be *practically* possible for concurrent constraint programming languages to use any constraint system to control suspension and resumption of execution.

References

1. Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351 (1986).

2. Hassan Aït-Kaci. *Warren's Abstract Machine, A Tutorial Reconstruction.* MIT Press, Cambridge, MA (1991).

3. Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215 (1986).

4. Hassan Aït-Kaci and Roger Nasr. Integrating logic and functional programming. *Lisp and Symbolic Computation*, 2:51–89 (1989).

5. Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. In Jan Maluszyński and Martin Wirsing, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming (Passau, Germany)*, pages 255–274. Springer-Verlag, LNCS 528 (August 1991).

6. Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. PRL Research Report 11, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France (1991). (Revised, October 1992; to appear in the Journal of Logic Programming).

7. Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. In *Proceedings of the 5th International Conference on Fifth Generation Computer Systems*, pages 1012–1022, Tokyo, Japan (June 1992). ICOT. (Full paper to appear in forthcoming special issue of *Theoretical Computer Science* on FGCS'92.).

8. Rolf Backofen and Gert Smolka. A complete and decidable feature theory. DFKI Research Report RR-30-92, German Research Center for Artificial Intelligence, Saarbrücken, Germany (1992).

9. Staffan Bonnier and Jan Maluszyński. Towards a clean amalgamation of logic programs with external procedures. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium*, pages 311–326, Cambridge, MA (1988). MIT Press.

10. Keith L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, NY (1978).

11. Alain Colmerauer. Prolog and infinite trees. In Keith Clark and Sten-Åke Tärnlund, editors, *Logic Programming*, pages 153–172, New York, NY (1982). Academic Press.

12. Alain Colmerauer. Prolog II: Manuel de référence et modèle théorique. Rapport technique, Université de Marseille, Groupe d'Intelligence Artificielle, Faculté des Sciences de Luminy, Marseille, France (March 1982).

13. Alain Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the Second International Conference on Fifth Generation Computer Systems*, pages 85–99, Tokyo, Japan (1984). ICOT.

14. Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169 (1983).

15. Seif Haridi and Sverker Janson. Kernel Andorra Prolog and its computation model. In David H. D. Warren and Peter Szeredi, editors, *Logic Programming, Proceedings of the 7th International Conference*, pages 31–46, Cambridge, MA (1990). MIT Press.

16. Robert Harper, Robin Milner, and Mads Tofte. The definition of standard ML – Version 2. Report LFCS-88-62, University of Edinburgh, Edinburgh, UK (1988).

17. Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Stuttgart, Germany (October 1988). To appear in the Journal of Logic Programming.

18. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of 14th Symposium of Principles of Programming Languages*, Munich (January 1987). ACM.

19. Jean-Louis Lassez, Michael Maher, and Kimball Mariott. Unification revisited. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 15, pages 587–625. Morgan-Kaufmann, Los Altos, CA (1988).

20. Michael Maher. Logic semantics for a class of committed-choice programs. In Jean-Louis Lassez, editor, *Logic Programming, Proceedings of the Fourth International Conference*, pages 858–876, Cambridge, MA (1987). MIT Press.

21. Lee Naish. *MU-Prolog 3.1db Reference Manual*. Computer Science Department, University of Melbourne, Melbourne, Australia (May 1984).

22. Vitor Santos Costa, David H. D. Warren, and Rong Yang. Andorra-I: A parallel Prolog system that transparently exploits both and- and or-parallelism. In *Proceedings of the 3rd ACM SIGPLAN Conference on Principles and Practice of Parallel Programming*, pages 83–93 (August 1991).

23. Vijay Saraswat and Martin Rinard. Concurrent constraint programming. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 232–245. ACM (January 1990).

24. Gert Smolka. Residuation and guarded rules for constraint logic programming. PRL Research Report 12, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France (1991).

25. Gert Smolka and Ralf Treinen. Records for logic programming. In Krzysztof Apt, editor, *Logic Programming, Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 240–254, Cambridge, MA (1992). MIT Press.

PRL Research Reports

The following documents may be ordered by regular mail from:

Librarian – Research Reports
Digital Equipment Corporation
Paris Research Laboratory
85, avenue Victor Hugo
92563 Rueil-Malmaison Cedex
France.

It is also possible to obtain them by electronic mail. For more information, send a message whose subject line is `help` to `doc-server@prl.dec.com` or, from within Digital, to `decprl::doc-server`.

Research Report 1: *Incremental Computation of Planar Maps*. Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. May 1989.

Research Report 2: *BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic*. Bernard Serpette, Jean Vuillemin, and Jean-Claude Hervé. May 1989.

Research Report 3: *Introduction to Programmable Active Memories*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. June 1989.

Research Report 4: *Compiling Pattern Matching by Term Decomposition*. Laurence Puel and Ascánder Suárez. January 1990.

Research Report 5: *The WAM: A (Real) Tutorial*. Hassan Aït-Kaci. January 1990.

Research Report 6: *Binary Periodic Synchronizing Sequences*. Marcin Skubiszewski. May 1991.

Research Report 7: *The Siphon: Managing Distant Replicated Repositories*. Francis J. Prusker and Edward P. Wobber. May 1991.

Research Report 8: *Constructive Logics. Part I: A Tutorial on Proof Systems and Typed λ-Calculi*. Jean Gallier. May 1991.

Research Report 9: *Constructive Logics. Part II: Linear Logic and Proof Nets*. Jean Gallier. May 1991.

Research Report 10: *Pattern Matching in Order-Sorted Languages*. Delia Kesner. May 1991.

---

† This report is no longer available from PRL. A revised version has now appeared as a book: "Hassan Aït-Kaci, *Warren's Abstract Machine: A Tutorial Reconstruction. MIT Press, Cambridge, MA (1991).*"

Research Report 11: *Towards a Meaning of LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, October 1992).

Research Report 12: *Residuation and Guarded Rules for Constraint Logic Programming*. Gert Smolka. June 1991.

Research Report 13: *Functions as Passive Constraints in LIFE*. Hassan Aït-Kaci and Andreas Podelski. June 1991 (Revised, November 1992).

Research Report 14: *Automatic Motion Planning for Complex Articulated Bodies*. Jérôme Barraquand. June 1991.

Research Report 15: *A Hardware Implementation of Pure Esterel*. Gérard Berry. July 1991.

Research Report 16: *Contribution à la Résolution Numérique des Équations de Laplace et de la Chaleur*. Jean Vuillemin. February 1992.

Research Report 17: *Inferring Graphical Constraints with Rockit*. Solange Karsenty, James A. Landay, and Chris Weikart. March 1992.

Research Report 18: *Abstract Interpretation by Dynamic Partitioning*. François Bourdoncle. March 1992.

Research Report 19: *Measuring System Performance with Reprogrammable Hardware*. Mark Shand. August 1992.

Research Report 20: *A Feature Constraint System for Logic Programming with Entailment*. Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. November 1992.

Research Report 21: *The Genericity Theorem and the Notion of Parametricity in the Polymorphic λ-calculus*. Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. December 1992.

Research Report 22: *Sémantiques des langages impératifs d'ordre supérieur et interprétation abstraite*. François Bourdoncle. January 1993.

Research Report 23: *Dessin à main levée et courbes de Bézier : comparaison des algorithmes de subdivision, modélisation des épaisseurs variables*. Thierry Pudet. January 1993.

Research Report 24: *Programmable Active Memories: a Performance Assessment*. Patrice Bertin, Didier Roncin, and Jean Vuillemin. March 1993.

Research Report 25: *On Circuits and Numbers*. Jean Vuillemin. April 1993.

Research Report 26: *Numerical Valuation of High Dimensional Multivariate European Securities*. Jérôme Barraquand. March 1993.

Research Report 27: *A Database Interface for Complex Objects*. Marcel Holsheimer, Rolf A. de By, and Hassan Aït-Kaci. March 1993.

Research Report 28: *Feature Automata and Sets of Feature Trees*. Joachim Niehren and Andreas Podelski. March 1993.

Research Report 29: *Real Time Fitting of Pressure Brushstrokes*. Thierry Pudet. March 1993.

Research Report 30: *Rollit: An Application Builder*. Solange Karsenty and Chris Weikart. April 1993.

Research Report 31: *Label-Selective λ-Calculus*. Hassan Aït-Kaci and Jacques Garrigue. May 1993.

Research Report 32: *Order-Sorted Feature Theory Unification*. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. May 1993.

13

Functions as Passive Constraints in LIFE

Hassan Aït-Kaci and Andreas Podelski

digital

PARIS RESEARCH LABORATORY

85, Avenue Victor Hugo
92563 RUEIL MALMAISON CEDEX
FRANCE