# Functions as Passive Constraints in LIFE

HASSAN AÏT-KACI and ANDREAS PODELSKI
Digital Equipment Corporation

LIFE is a programming language proposing to integrate logic programming, functional programming, and object-oriented programming. It replaces first-order terms with $\psi$-terms, data structures that allow computing with partial information. These are approximation structures denoting sets of values. LIFE further enriches the expressiveness of $\psi$-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 $\psi$-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 $\psi$-terms in LIFE.

Categories and Subject Descriptors: D. [**Software**]: Programming Techniques; D.3.1 [**Programming Languages**]: Formal Definitions and Theory---*semantics*; *syntax*; D.3.2 [**Programming Languages**]: Language Classifications---*applicative languages*; *concurrent, distributed, and parallel languages*; *nonprocedural languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features---*concurrent programming structures*; *coroutines*; *data types and structures*; E. [**Data**]: Data Structures---*graphs*; *trees*

General terms: Design, Languages

Additional Key Words and Phrases: $\psi$-terms, committed-choice languages, concurrent constraint programming, coroutining, first-order terms, matching, relative simplification, residuation, unification

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

Logic programming frameworks that exploit the separation of relational resolution and constraint solving have recently been introduced. Among the preeminent, Constraint Logic Programming (CLP) [Jaffar and Lassez 1987], the guarded Horn clause scheme of Maher (ALPS) [Maher 1987], Concurrent Constraint Programming (CCP) [Saraswat and Rinard 1990], and Kernel Andorra Prolog (KAP) [Haridi and Janson 1990] do so to a full extent by being parameterized with respect to an abstract class of constraint systems. Additionally, ALPS, CCP, and KAP require a test for entailment and disentailment between constraints. This yields advanced control mechanisms such as coroutining and synchronization through

Authors' addresses: H. Aït-Kaci, School of Computing Science, Simon Fraser University, Burnaby, British Columbia, V5A 1S6, Canada; email: hak@cs.sfu.ca; A. Podelski, Digital Equipment Corporation, Paris Research Laboratory, 85 avenue Victor Hugo, 92563 Rueil-Malmaison Cedex, France; email: podelski@prl.dec.com.

committed choice and deep constraint propagation.

LIFE [Aït-Kaci and Podelski 1993b] is a CLP language over order-sorted feature constraints 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 scheme called *residuation* to enforce deterministic functional application. In this work, we extend the guarded Horn clause scheme of Maher [1987] and present a logical semantics of the general *residuation scheme* used in LIFE.

This will involve three things essentially. First, we will develop a general residuation framework for guarded Horn clauses over arbitrary constraint systems with an incremental constraint simplification system. Doing so, we will give a logical reading of guarded rules as first-order formulae and exhibit operational and semantical properties of the framework. Second, we will develop a correct and complete operational scheme for testing entailment and disentailment of order-sorted feature constraints. To that end, we will introduce a general technique, which we dub *relative simplification*, that amounts to normalizing a formula in the context of another. Last, we will use this general residuation framework on the particular instance of functional application over the order-sorted features terms of LIFE. In particular, we will characterize functional application over LIFE's structures in terms of their logical, set-theoretic, and algebraic accounts.

## 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 $\psi$-terms, which are attributed partially ordered sorts denoting sets of objects [Aït-Kaci 1986; Aït-Kaci and Nasr 1986]. In particular, $\psi$-terms generalize first-order constructor terms in their role as data structures in that they are endowed with a unification operation denoting set 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 $\psi$-terms. As a result, the notion of a fully defined element, or ground term, is no longer available. Hence, familiar tools such as variable substitutions, instantiation, unification, *etc.* must be reformulated in the new setting [Aït-Kaci and Podelski 1993b].

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 *definiendum* pattern 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

[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.,* Harper et al. [1988]).

[Aït-Kaci and Nasr 1989]. This simple idea turns out invaluable in practice. Here are a few benefits.

---Such nondeclarative heresies as the *is* predicate in Prolog and the *freeze* metapredicate in some of its extensions [Colmerauer 1982b; Naish 1986] are not needed.

---Functional computations are determinate and do not incur the overhead of the search strategy needed by logic programming.

---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 [Bonnier and Maluszyński 1988].

---More generally, it allows concurrent computation. Synchronization is obtained by checking entailment [Maher 1987; Saraswat and Rinard 1990].

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 article is concerned only with the first item, and therefore considers the case of first-order rules defining partial functions over $\psi$-terms.

## 1.2 The Method

The most direct way to explain the issue is with an example. In LIFE, we 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 *posint*, *zero*, *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 $Y = minus(X : string)$ 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:

$$Y = minus(X : int), X = minus(zero)?$$

Like Prolog, LIFE follows a left-to-right resolution strategy and examines the equation $Y = minus(X : int)$ 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 $Y = zero$.

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 the process of 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 that tests 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 article is that it is possible to build such a test, namely, an algorithm deciding simultaneously two problems in an incremental manner---entailment and disentailment. *Relative simplification* of constraints is the technique that we have devised to do that.

Consequently, although motivated by our concern for LIFE, this technique is relevant to general concurrent constraint logic programming, whose paradigm of concurrency rests on a new effective discipline for procedure parameter passing that can be described as ''call-by-constraint entailment'' (as opposed to Prolog's call-by-unification).

## 1.3 Relation to Other Work

It is important that we situate the contribution of this article with respect to those of others. Indeed, there are some subtle issues that may cause confusion and need to be explicated.

What we recount has close ties with the ALPS system of Maher [1987] and the work of Smolka [1991] characterizing residuation in terms of guarded rules. The main contribution of Maher was the formal insight explaining (1) the commit condition in the residuation mechanism as logical entailment of the guard constraint and (2) reduction via the residuation mechanism as equivalence transformations. As a result, he could give committed-choice languages a logical semantics. Smolka introduced the notion of a guarded rule. Such a rule expresses directly that its (atomic) head is equivalent to its body if and only if its guard is entailed by the context. Hence, the residuation mechanism itself is thereby justified logically.

In both Maher's and Smolka's schemes, a program has two components. The first component consists of a Horn clause program from which one derives the denotational semantics. The operational semantics is derived from the other component. This component consists either (1) of guarded Horn clauses (which one obtains from the syntax of Horn clauses by explicitly writing one conjunction symbol as the guard operator) or (2) of guarded rules (which one adds to the Horn clause program, in accordance with its denotational semantics). There is an issue as to whether this dichotomy is at all necessary. It would better to have a single view of a program having only one component (consisting of first-order logic formulae without special symbols for control annotations) which declares the denotational *and* the operational semantics. This is indeed what is achieved by our scheme.

It is well known that, in general, committed-choice programs do not have a declarative semantics.[2] That is, a predicate definition with two or more clauses cannot in general be translated to a consistent first-order formula (*i.e.*, with a nontrivial model). We give here a necessary and sufficient condition when this is yet the case. That is, we characterize committed-choice programs which do have a consistent denotational semantics. Roughly, this condition is a compatibility and a closed-world assumption on the guards. Importantly, this result extends the declarative semantics results of Maher [1987] for the case of incompatible guards to the more general case of compatible guards. Given a guarded Horn clause program satisfying the compatibility condition, one can give a program in our scheme with equivalent operational semantics (and with a denotational semantics).

The other contribution of our scheme is to describe formally a general implementation technique for proving guards incrementally, called relative simplification, and to incorporate it with the delay mechanism. We have abstracted its essential properties from the systems that we have developed for proving guards in two specific contexts: one for (unsorted) feature logic [Aït-Kaci et al. 1994] and the one we report in this article for the order-sorted case (*i.e.*, the system of Section 3, presented in abridged form by Aït-Kaci and Podelski [1993a]). In the operational semantics of their respective schemes, Maher [1987] and Smolka [1991] do not deal with practical proofs of guards. In particular, Smolka formulates the proof of entailment of the guard as the simplification of the conjunction of the context constraint and the negated guard constraint to *false*. Regarding implementation, this would lead to a nonincremental proof system. In contrast, we present an operational semantics for residuation that is based on an incremental algorithm that can be simultaneously used for doing both tests: entailment and disentailment, namely, in the case of terms, matching and (non)unification.

Finally, it is important to observe that neither Maher nor Smolka explicate goal reduction

[2] Formally, these are programs with guarded Horn clauses, guarded rules, or also the committed-choice combinator [Haridi et al. 1992].

in the presence of negation in guards, whereas we do.

Although our common work with Smolka [Aït-Kaci et al. 1994] was the first publication using relative simplification, the concept was first introduced by us as early as June 1991 [Aït-Kaci and Podelski 1991]. There, we developed the system we present in this article as a more general one than the one described by Aït-Kaci et al. [1994]. In fact, the work presented by Aït-Kaci et al. [1994] was the result of Smolka's suggestion that we work out the simpler case of unsorted partial-feature logic before we publicize the more complex and more general system we report here. Indeed, it turned out that taking into account the hierarchy on sorts is a nontrivial extension of the case of pairwise incompatible sorts treated by Aït-Kaci et al. [1994]. Subtle complexities arise, for example, when three sorts are incompatible, but pairwise compatible. Further, we present here the first truly incremental system in the sense that none of its rules introduces an operation that has to be undone upon residuation (when neither entailment nor disentailment can be proven). Thus, for example, two variables from the resolvent are never bound together. This property leads to a more complex system than the system of Aït-Kaci et al. [1994].

### 1.4 Organization of Article

In Section 2, we explain residuation in a general framework, introducing the concept of relative simplification as a general proof-theoretic method for proving guards in concurrent constraint logic languages using guarded rules. In Section 3, we develop a specific relative simplification system for order-sorted feature constraints, the data structures of LIFE. Section 4 ties the operational semantics of function reduction with the semantics of $\psi$-terms as approximation structures. Finally, we conclude with Section 5, giving a brief recapitulation of this article's contribution and a few perspectives.

Throughout, we use the terminology and notation introduced by Aït-Kaci and Podelski [1993b]. So, we provide an appendix where we recall all the necessary formalism accounting for LIFE's structures and operations. It is meant to make this document self-contained. The reader already familiar with those notions could ignore it altogether, although reading it may provide a timely summary.

## 2. A GENERAL RESIDUATION FRAMEWORK

### 2.1 Overview

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 that are constrained by the guard are its input parameters and correspond to the arguments of a function. Also, although in the current version of LIFE the guards corresponding to a function's arguments are mutually exclusive, we will develop a scheme that does not need this restriction.

A program in our scheme, which uses logical formulae called *guarded rules*, can characterize the denotational and the operational meaning of a program in the ALPS scheme of Maher [1987], which uses *guarded Horn clauses*. Namely, a definition by $n$ guarded Horn clauses corresponds to the conjunction of $n + 1$ guarded rules.

We will relax 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 the denotational-semantics results. 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 nine guarded Horn clauses, instead of just two, namely (we will introduce the syntax formally later):

$$
\begin{array}{lll}
\mathit{and}(X, Y, Z) \ \texttt{:-} & X = \mathit{false} & [\!] \ Z = \mathit{false};\\
& Y = \mathit{false} & [\!] \ Z = \mathit{false};\\
& X = \mathit{true} & [\!] \ Y = Z;\\
& Y = \mathit{true} & [\!] \ X = Z;\\
& Z = \mathit{true} & [\!] \ X = \mathit{true}, Y = \mathit{true};\\
& X = Y & [\!] \ X = Z;\\
& X \neq Y & [\!] \ Z = \mathit{false};\\
& X \neq Z & [\!] \ X = \mathit{true}, Y = \mathit{false}, Z = \mathit{false};\\
& Y \neq Z & [\!] \ X = \mathit{false}, Y = \mathit{true}, Z = \mathit{false}.
\end{array}
$$

If the constraint system provides a test of entailment that allows negated constraints in the context, we can add five more rules (this test is treated by Ramachandran and Van Hentenryck [1993]):

$$
\begin{array}{lll}
\mathit{and}(X, Y, Z) \ \texttt{:-} & X \neq \mathit{true} & [\!] \ X = \mathit{false}, Z = \mathit{false};\\
& Y \neq \mathit{true} & [\!] \ Y = \mathit{false}, Z = \mathit{false};\\
& X \neq \mathit{false} & [\!] \ X = \mathit{true}, Y = Z;\\
& Y \neq \mathit{false} & [\!] \ Y = \mathit{true}, X = Z;\\
& Z \neq \mathit{false} & [\!] \ X = \mathit{true}, Y = \mathit{true}, Z = \mathit{true}.
\end{array}
$$

The question is: when do we still obtain the denotational-semantics results, having dropped the requirement that Maher put on the guards? It is clear that generally this is not possible. We introduce a *compatibility condition* for guarded rules and show it to be necessary and sufficient for the existence of a model of guarded Horn clauses.

In the scheme of Maher [1987], the denotational semantics is obtained by ignoring the operational semantics. That is, the control construct *guard* is given a first-order logic reading (*i.e.*, conjunction) that is orthogonal to its operational significance. Namely, we see guarded Horn clauses as defining a relation *r* by considering them as simple Horn clauses. This amounts to using Clark's completion, yielding a *definite equivalence* [Clark 1978]. In the scheme of Smolka [1991], a relation *r* is first defined by a definite equivalence defining the semantics of this relation, and only then guarded rules are added, which define the derivations. In our scheme, the specification of a relation *r*, done solely by guarded rules, defines its semantics and the derivations.

Furthermore, the definition of reduction in our guarded-rule reduction scheme extends the one of Smolka. Namely, it avoids useless redundancies in the syntactic formulation of guarded rules and in the operational semantics of reduction, as will be explained next.[3]

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 (namely, 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

[3]We mean ''useless redundancy,'' not as a pleonasm, but as a deliberate opposition to ''useful redundancy'' serving a pragmatic purpose (see also footnote 6).

matching proof is done by unification, which yields the *instantiation* of the body variable $X$, namely, $X = a$. In order to compute the new environment, this unification should, of course, not be repeated. In fact, it is avoided in our scheme, whereas in Smolka's scheme the guard $Y = f(X)$ has to occur in the body for a second time, and the unification $f(a) = f(X)$ has to be repeated.

The example above can be generalized to all 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 $Y = f(X)$ relative 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*. This method applies in particular to the order-sorted feature (OSF) constraint system used in LIFE, as we show in Section 3.

In summary, our scheme captures the practically relevant case where the variables in the body are already instantiated through the corresponding guard's entailment proof. 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 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 followed by the check whether global variables have been bound is *not* incremental; bindings of global variables effected for a test have to be undone afterward.

The rest of this section details the general scheme that we have just overviewed. We first present guarded rules and give an operational and denotational semantics for them. We explain how they relate to guarded Horn clauses and give examples. Considering incremental relative-simplification systems in general, we exhibit some properties that indicate how these might be constructed from a unification system, or more generally, from a constraint solver. Finally, we combine all these notions to derive the operational semantics of residuation.

## 2.2 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 $r \in \mathcal{R}$ and the $X_i$'s are mutually distinct variables. For notational convenience, we will write such a relational atom simply as $r(X)$. Also, when equating tuples, we will write a sequence $X_1 \doteq U_1, \ldots, X_n \doteq U_n$ simply as $X \doteq U$.

Also, we assume a class of logical formulae (called constraints), noted $\phi, \psi, \ldots$, closed under conjunction and including the *false* constant $\bot$ and the *true* constant $\top$ and a model or a class of models (possibly specified by axioms), to which satisfiability and validity will refer in the following. As usual, an empty conjunct is considered the same as $\top$. We assume that the constraint systems come with a test of satisfiability and entailment (which is provided, *e.g.*, by a relative simplification system).

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

$$\forall U \, \forall \mathcal{U}. \, \big( \, G \; \rightarrow \; \big( \, r(U) \; \leftrightarrow \; \exists \mathcal{V}. \, B \, \big) \, \big) \qquad (1)$$

where

---the guard $G$ is a constraint;

---the head $r(U)$ is a relational atom;

---the body $B$ is of the form $R \;\&\; \phi$, where $R$, the relational part, is a (possibly empty) conjunction of relational atoms, and $\phi$, the constraint part, is a (possibly *true*) constraint formula;

---$\mathcal{U} = Var(G) - \{U\}$ and $\mathcal{V} = Var(B) - (\mathcal{U} \cup \{U\})$.

Later we will see that the guard $G$ can be a conjunction of a constraint and negated conjuncts. We first consider the case where $G$ is a constraint.

As an example, $X = \mathit{false} \rightarrow (\mathit{and}(X, Y, Z) \leftrightarrow Z = \mathit{false})$ is the guarded rule which corresponds to the first of the guarded Horn clauses specifying the *and* predicate given above. We will formalize this correspondence later.

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 $\phi$, its context, is a (possibly *true*) constraint formula. In the following, we will consider only the derivation of resolvents without quantifiers. Indeed, only the matrix of a quantified resolvent is rewritten (adding possibly more quantifications).

We will call the variables in $Var(\boldsymbol{R})$ *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, 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.

A guarded-rule *reduction* derives the resolvent of the form

$$\boldsymbol{R} \;=\; R \;\&\; r(X) \;\&\; \phi$$

by application of the guarded rule of the form above to the resolvent

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

if (and only if) the context $\phi$ of the resolvent $\boldsymbol{R}$ entails the guard of the rule, *i.e.*, if

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

is valid.[4]

PROPOSITION 1. *A guarded-rule reduction is an equivalence transformation (of a resolvent to the derived resolvent).*

PROOF. The entailment condition says that the context $\phi$ is equivalent to its conjunction with the instantiated guard,

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

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

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

Since the variable $U$ and the variables in $\mathcal{U}$ are universally quantified, the guarded rule can be written as

[4]Our definition of guarded rules is the same as, but our definition of guarded-rule reduction is different from, what is defined by Smolka [1991], which does not include the guard in the derived resolvent $\boldsymbol{R}'$. We will explain the effect of this difference in Section 2.4.

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

It follows that ***R*** is equivalent to

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

*Two Remarks.* First, 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 $(\forall)\ \big(\ G\ \rightarrow\ \big(r(U)\ \leftrightarrow\ B\big)\big)$. Second, let us assume that the constraint $\phi$ entails the guard $G$. Then, although $\phi$ 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.

All conjuncts in the guard that do *not* share variables other than $U$ (the variables in the head) with the body, may be omitted in the derived resolvent. We will exploit this next when we define guarded-rule reduction with guards which contain negations as conjuncts. It is clear that variables in the scope of a negated existential quantifier do not occur in the body.

Generally, a *guard G* in a guarded rule of the form (1) is a conjunction of the form

$$G\ =\ G_0\ \&\ \bigwedge_{j=1}^{k} \neg\exists\mathcal{U}_j.\ G_j \tag{2}$$

where $G_0, \ldots, G_n$ are constraints. 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 $\mathcal{V}$.

We now define guarded-rule reduction by application of the guarded rule (1) with the guard (2): The resolvent $\boldsymbol{R}\ =\ R\ \&\ r(X)\ \&\ \phi$ derives to the resolvent $\boldsymbol{R}'\ =\ \exists U\ \exists\mathcal{U}\ \exists\mathcal{V}.\ (R\ \&\ B\ \&\ \phi\ \&\ G_0\ \&\ U \doteq X)$ (hence, without the negated conjuncts of the guard) if (and only if) the context $\phi$ of the resolvent entails the guard. This means that (1) the implication

$$\phi\ \rightarrow\ \exists U\ \exists\mathcal{U}.\ (G_0\ \&\ U \doteq X)$$

is valid and (2) the conjunctions

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

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

Proposition 2. *Guarded-rule reduction is as well an equivalence transformation under the definition for guards with negations.*

Proof. The proof of Proposition 1 can be rephrased with the new form of $G$. Under the entailment assumption, the context $\phi$ is equivalent to $\phi\ \&\ \neg\exists\mathcal{U}_j.\ (G_j\ \&\ U \doteq X)$, and since $G_j$ does not share variables with $B$, $B\ \&\ \phi$ is equivalent to $B\ \&\ \phi\ \&\ \neg\exists\mathcal{U}_j.\ (G_j\ \&\ U \doteq X)$. This means that the conjuncts $\neg\exists\mathcal{U}_j.\ (G_j\ \&\ U \doteq X)$ can be omitted in the derived resolvent. $\Box$

A guarded-rule program can be given a logical semantics (and a reasonable operational semantics) only if every collection of $n$ guarded rules with the same head in the program, namely,

$$\forall U\ \forall\mathcal{U}_i.\ \big(\ G_i\ \rightarrow\ \big(\ r(U)\ \leftrightarrow\ \exists\mathcal{V}_i.\ B_i\ \big)\ \big), \tag{3}$$

(where $\mathcal{U}_i = Var(G_i) - \{U\}$ and $\mathcal{V}_i = Var(B_i) - \big(\mathcal{U}_i \cup \{U\}\big)$, for $i = 1, \ldots, n$) comes with a guarded rule called the ''otherwise'' rule (which might be left implicit) of the form

$$\forall U.\ \big(\neg\exists\mathcal{U}_1 G_1\ \&\ \ldots\ \&\ \neg\exists\mathcal{U}_n G_n\ \rightarrow\ \big(\, r(U)\ \leftrightarrow\ \bot\,\big)\,\big). \tag{4}$$

We assume the guards $G_i$ to be of the general form, as in (2).

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

PROPOSITION 3. *The following formula is a logical consequence of the* $n + 1$ *guarded rules given in (3) and (4):*

$$\forall U.\ \big(\, r(U)\ \leftrightarrow\ \bigvee_{i=1,\ldots,n}\ \exists\mathcal{U}_i\exists\mathcal{V}_i.\ (G_i\ \&\ B_i)\,\big). \tag{5}$$

PROOF. The proof for the $\leftarrow$ part of the formula (5) is clear. For the $\rightarrow$ part we consider the two cases whether or not $\neg r(U)$, and therefore $r(U) \leftrightarrow \bot$, holds in an interpretation. In the first case, there is nothing to show. In the second case, we use the $(n + 1)st$ guarded rule, the ''otherwise'' rule, by contraposition. □

Still, 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*.

$$\bigwedge_{i\,,\,j=1}^{n} \Big(\ \forall U\, \forall\mathcal{U}_i\forall\mathcal{U}_j.\ \big(G_i\ \&\ G_j\ \rightarrow\ (\exists\mathcal{V}_i.\ B_i\ \leftrightarrow\ \exists\mathcal{V}_j.\ B_j)\,\big)\ \Big). \tag{6}$$

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

PROPOSITION 4. *Every model of the definite equivalence (5) and the compatibility condition (6) is a model of the conjunction of the* $n + 1$ *guarded rules of the form (1), and vice versa.*

PROOF. By (5), $G_i\ \&\ \exists\mathcal{V}_i.\ B_i$ implies $r(U)$. If $G_i\ \&\ r(U)$ holds in an interpretation, then, by (5), there exists some $j$ such that $G_j\ \&\ \exists\mathcal{V}_j.\ B_j$ holds. But then, by (6), $\exists\mathcal{V}_i.\ B_i$ holds also. The $(n + 1)st$ guarded rule is an immediate consequence of (5). The other direction follows from Proposition 3 for (5) and from combining the guarded rules pairwise for (6). □

COROLLARY 1. *If the compatibility condition is valid, then a guarded-rules program has a least model.*

PROOF. It is a well-known fact that a system of predicate definitions such as (5) has a least model extending the model of the theory of the constraint domain (see Jaffar and Lassez [1987] and Höhfeld and Smolka [1988]). The statement then follows from the assumption and Proposition 4. □

A *guarded Horn clause* is of the form ''$H$ :- $G$ [] $B$'' 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 $\phi$, 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:[5]

$$\begin{aligned}
&\mathit{concat}(X,Y,Z) \;\text{:-}\; X : \mathit{nil} \;[]\; Y \doteq Z.\\
&\mathit{concat}(X,Y,Z) \;\text{:-}\; X : \mathit{cons} \;\&\; X.\mathit{hd} \doteq V \;\&\; X.\mathit{tl} \doteq W \;[]\\
&\qquad\qquad\qquad\quad Z : \mathit{cons} \;\&\; Z.\mathit{hd} \doteq V \;\&\; Z.\mathit{tl} \doteq L \;\&\; \mathit{concat}(W,Y,L).
\end{aligned}$$

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 $H$ :- $G \;[]\; B$ corresponds to the guarded rule of the form (1).

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

$$\begin{aligned}
&\forall U_1, U_2, U_3\\
&\big(\, (U_1 : \mathit{nil})\\
&\rightarrow \big(\mathit{concat}(U_1,U_2,U_3) \leftrightarrow U_2 \doteq U_3\big).
\end{aligned}$$

$$\begin{aligned}
&\forall U_1, U_2, U_3,\; V, W.\\
&\big(\, (U_1 : \mathit{cons} \;\&\; U_1.\mathit{hd} \doteq V \;\&\; U_1.\mathit{tl} \doteq W)\\
&\rightarrow \big(\; \mathit{concat}(U_1,U_2,U_3)\\
&\qquad \leftrightarrow \exists L.\, (U_3 : \mathit{cons} \;\&\; U_3.\mathit{hd} \doteq V \;\&\; U_3.\mathit{tl} \doteq L \;\&\; \mathit{concat}(W,U_2,L)\;)\;\big)\;\big).
\end{aligned}$$

Since any constraint system can be trivially augmented to express tuples (although doing so may increase its expressive power significantly), we may assume the relational symbol $r$ in the head to be a unary predicate. This amounts to replacing $r(U_1,\ldots,U_n)$ :- $G \;[]\; B$ with $r(U)$ :- $U \doteq (U_1,\ldots,U_n) \;\&\; G \;[]\; B$. 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 $U.1 \doteq U_1 \;\&\; \ldots \;\&\; U.n \doteq U_n$.

With unary relation symbols, the guarded rules corresponding to our foregoing definition of *concat* are:

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

$$\begin{aligned}
&\forall U\, \forall\{X,V,W\}.\\
&\big(\, (U.1 \doteq X \;\&\; X : \mathit{cons} \;\&\; X.\mathit{hd} \doteq V \;\&\; X.\mathit{tl} \doteq W)\\
&\rightarrow \big(\; \mathit{concat}(U)\\
&\qquad \leftrightarrow\; \exists\{Y,Z,L\}.\\
&\qquad\quad (\; U.2 \doteq Y \;\&\; U.3 \doteq Z \;\&\; Z : \mathit{cons} \;\&\; Z.\mathit{hd} \doteq V \;\&\; Z.\mathit{tl} \doteq L \;\&\\
&\qquad\qquad \mathit{concat}(W,Y,L)\;)\;\big)\;\big).
\end{aligned}$$

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

[5]In this example, we use the syntax of OSF constraints expressing the nature of the data expected by the rules. Namely, $V : s$ stands for ''variable $V$ is in sort $s$,'' $V.\ell \doteq W$ stands for ''feature $\ell$ of variable $V$ is equal to variable $W$,'' and $V \doteq W$ ''variable $V$ is equal to variable $W$.'' Hence the guards in the two rules of this example constrain $X$ to be, respectively, the emply list *nil* and a nonempty list *cons* with a head feature *hd* equal to $V$ and a tail feature *tl* equal to $W$.

$$\begin{array}{l}\forall U.\\ \big(\,\exists X.(U.1 \doteq X \;\&\; X : nil)\\ \rightarrow \big(\,concat(U)\\ \qquad \leftrightarrow\; \exists\{Y,Z\}.(\,U.2 \doteq Y \;\&\; U.3 \doteq Z \;\&\; Y \doteq Z\,)\,\big)\,\big).\end{array}$$

This is also true for $X$ in the second rule, but not for $V$ nor $W$, since the scope of these two variables extends over the guard and the body.

The collection of the guarded Horn clauses $r(U)$ :- $G_i \;[\!]\; B_i$ (where $i = 1, \ldots, n$) with the same head in a given program stands for the conjunction of $n+1$ guarded rules, namely, $n$ of the form 3 and one, the ''otherwise'' rule, of the form 4. In our examples, the $(n+1)st$ guarded rule (the ''otherwise'' rule) is always left implicit.

A model of a guarded Horn clause program is a model of the corresponding guarded-rules program, hence the following.

COROLLARY 2. *If the compatibility condition is valid, then a guarded Horn clause program has a least model.*

For example, this allows us to show that the program for the *and* predicate given above has a declarative semantics.

For the sake of completely relating our approach to others, let us mention one idea that is not (yet) implemented in LIFE. Given a program consisting of definite clauses, one can add explicit guarded rules that are logical consequences of the program [Smolka 1991]. 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

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

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

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

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.[6] This appears to be related to what has been quoted to us as the ''Andorra Principle'' [Haridi and Janson 1990], a strategy of preferentially selecting goals that have at most one alternative, and is a basic principle underlying the Andorra Model [Santos Costa et al. 1991].

## 2.3 Incremental Relative-Simplification Systems

If $G$ is a guard of the general form, as in (2), and $\phi$ 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 2 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.

[6]This is an example of a useful redundancy (see also footnote 3).

If the context $\phi$ 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 $\phi$ 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 $\phi$ of the current resolvent $\boldsymbol{R}$, the proof that 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 5. *The context $\phi$ 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 $\phi \;\rightarrow\; G'$ is also valid. Therefore, $\phi$ is equivalent to $\phi$ & $G'$. According to the assumption, $\phi$ & $(G$ & $U \doteq X) \;\leftrightarrow\; \phi$ & $G'$ is valid. Thus, $\phi$ is equivalent to $\phi$ & $(G$ & $U \doteq X)$. This shows that $\phi \;\rightarrow\; \exists U \exists \mathcal{U}\; (G$ & $U \doteq X)$ is valid. For the other direction, it is sufficient to choose $G' \;=\; \big((G$ & $U \doteq X) \vee \neg\phi\big)$. 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 $\phi$ and $G$, the formula $G'$ has to be effectively found, and its validity has to be effectively determined.

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

COROLLARY 3. *If the guard consists of a positive constraint, say $\psi$, 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 $\psi'$ 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 5, 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 $\phi$. 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 4. *The context $\phi$ disentails the guard $\psi$, 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

$$\phi \;\&\; \exists U\, \exists \mathcal{U}.\, (\psi \;\&\; U \doteq X) \;\leftrightarrow\; \phi \;\&\; \exists \mathcal{U}'.\, \psi'$$

is valid, then also

$$\phi \;\&\; \neg\exists U\, \exists \mathcal{U}.\, (\psi \;\&\; U \doteq X) \;\leftrightarrow\; \phi \;\&\; \neg\exists \mathcal{U}'.\, \psi'. \quad \Box$$

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 $\phi$, 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* 1. (*Relative-Simplification System*) A reduction system is a *relative-simplification system* if, given the context constraint $\phi$, the guard constraint $\psi$, and the binding $U \doteq X$ of the variable $U$ in $\psi$ to the variable $X$ in $\phi$, it reduces $\psi \;\&\; U \doteq X$ to a constraint $\psi'$ with $\mathcal{V} = Var(\psi') - Var(\phi)$ such that

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

---$\psi' = \bot$ if and only if $\phi$ disentails $\psi$; *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 $\psi'$ with $\mathcal{V} = Var(\psi') - Var(\phi)$ the following *relative-simplification invariant* must hold:

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

An important property for efficient implementation of the entailment and disentailment tests is *incrementality*. This means, intuitively, no *undoing*, *i.e.*, the ability to reuse work done in previous tests. Technically, this is the case if the constraint simplification steps used for the tests are the same whether the resolvent is given at once, or whether it is built up gradually at each goal reduction step (and the tests are applied following each reduction step). This is facilitated by the conjoining effect of goal reduction that builds a new context by conjunction of a rule's body constraints to the existing context. Formally, let $\phi_i$, $i \in \mathbb{N}$, be a sequence of contexts obtained by goal reduction, *i.e.*, such that $\phi_{i+1} = \phi_i \;\&\; \phi'_i$, for any $i$.

*Definition* 2. (*Incremental Constraint System*) A constraint system is said to be *incremental* if the sequence of constraint simplification steps used for $n$ successive applications of the entailment and disentailment tests applied to the $\phi_i$'s and the guard $\psi$, for $i = 0, \ldots, n$, is one of the nondeterministic paths of constraint simplification performing the tests directly on $\phi_n$ and $\psi$.

PROPOSITION 6. (CONFLUENCE OF RELATIVE SIMPLIFICATION) *It is possible to transform a relative-simplification system into an incremental one simply by closing the simplification relation with respect to the following condition. If $\psi$ simplifies to $\psi'$ relative to $\phi$, and $\psi$ simplifies to $\psi''$ relative to $\phi \;\&\; \phi'$, then also:*

---*$\psi$ simplifies to $\psi'$ relative to $\phi \;\&\; \phi'$, and*

---*$\psi'$ simplifies to $\psi''$ relative to $\phi \;\&\; \phi'$.*

PROOF. Observe that the relative-simplification invariant still holds if one considers every simplification relative to $\phi$ also as a simplification relative to $\phi \;\&\; \phi'$. Namely, if $\phi \rightarrow (\psi \leftrightarrow \psi')$ is valid, then so is $\phi \;\&\; \phi' \rightarrow (\psi \leftrightarrow \psi')$. The remaining conditions of Definition 1 are trivially fulfilled. $\square$

Generally, it is not evident how to transform the specification of a nonincremental 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 trees [Colmerauer 1982a; Colmerauer 1984] or feature trees [Aït-Kaci et al. 1994; Smolka and Treinen 1992]) where incrementality came for free.

## 2.4 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 $r(U)$ :- $G$ [] $B$, 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 $\psi'_j$ relative to the context $\phi$, with $Var(\psi_j) = \mathcal{U}_j$ and $Var(\psi'_j) - Var(\phi) = \mathcal{U}'_j$ for $j = 0, 1, \ldots, k$.

PROPOSITION 7. (CORRECTNESS OF REDUCTION) *The reduction transforming the resolvent* $\boldsymbol{R}$ *into 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 3 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 7) $\boldsymbol{R}$ is equivalent to $\boldsymbol{R}'$. Equivalently, we have the following.

PROPOSITION 8. (COMPLETENESS OF REDUCTION) *If* $\exists\mathcal{U}'_0.\; \psi'_0$ *is valid and* $\psi'_j = \perp$*, for each* $j = 1, \ldots, k$*, then the solutions of* $\boldsymbol{R}'$ *and* $\boldsymbol{R}$ *coincide. Moreover,* $\boldsymbol{R}'$ *is then equivalent to* $R \;\&\; \phi \;\&\; \psi'_0 \;\&\; B$.

PROOF. This follows from Proposition 2 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 (see Section 3), and also of Prolog terms. 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 [1991]. There, the ‘‘commit condition’’ is that the conjunction of the context $\phi$ and the negated guard $\neg\psi$ be a constraint that simplifies to $\perp$, 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 $\phi''$.

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

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

must be written in the form

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

As a result, in Smolka's operational semantics of guarded-rule reduction, the simplification of the constraint $\phi$ & $\phi'$ must repeat the simplification of the conjunction of the context and the guard.[7] Thus, for constraint systems with relative simplification, our formulation has an advantage in efficiency. Namely, 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, we have the following.

PROPOSITION 9. (FAILURE OF REDUCTION) *The set of solutions of* $\boldsymbol{R}'$ *is empty, if* $\psi'_0 = \bot$, *or if* $\exists \mathcal{U}'_j .\ \psi'_j$ *is valid, for at least one of* $j = 1, \ldots, k$.

PROOF. This follows from Proposition 3 and Definition 1. □

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 $\boldsymbol{R}'$ that 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 (see Theorem 3), such a constraint part can be tested for satisfiability by testing entailment of each of the negated constraints by the positive constraint.

## 3. ENTAILMENT AND DISENTAILMENT OF OSF CONSTRAINTS

In the following, we use $\phi$ as the *context* formula. It is assumed to be an OSF constraint in solved form, although not necessarily coming from dissolving a single $\psi$-term. The variables in $\phi$ 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 $\psi$, a dissolved $\psi$-term, as the *guard* formula. The variables in $\psi$ are *local* to $\psi$, *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 $\psi$. We also refer to $\phi$ as the *actual* parameter, and $\psi$ 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 thus must study a proof system that decides two problems simultaneously:

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

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 1, 2, and 3. An OSF constraint $\psi$ simplifies to $\psi'$ relative to $\phi$ by a simplification rule $\rho$ if $\frac{\psi}{\psi'}$

[7]Of course, this is because Smolka did not know about relative simplification at the time of his formulation (nor did we, as a matter of fact!).

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:

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

Relative Feature Equality:

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

Variable Introduction:

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

Fig. 1. Simplification relative to $\phi$: Features.

is an instance of $\rho$ and if the applicability condition (on $\phi$ and $\psi$) is satisfied. We say that $\psi$ simplifies to $\psi'$ relative to $\phi$ 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.

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

There are two technical remarks to be made. First, 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. Equality constraints $U_1 \doteq X_1, \ldots, U_n \doteq X_n$ can be equivalently replaced by adding $X_1 \doteq X.1 \ \& \ \ldots \ \& \ X_n \doteq X.1$ to the context $\phi$ and $U_1 \doteq U.1 \ \& \ \ldots \ \& \ U_n \doteq U.n \ \& \ U \doteq X$ to $\psi$, where $X$ and $U$ are new. That is, one obtains the conjunction of one equality $U \doteq X$ and a guard that, again, is a dissolved $\psi$-term.

Second, the fact that $\psi$ is a dissolved $\psi$-term rooted in $U$ ensures that the test of entailment of $\psi \ \& \ U \doteq X$ by $\phi$ does not depend on whether the implication holds in *all* OSF interpretations, or only in $\Psi$, or $\mathcal{T}$. This is not necessarily so if $U$ is not the root of $\psi$. Indeed, let us assume that $U$ is *not* the root of $\psi$; for example, take $\psi$ to be $V.\ell \doteq U$.

Sort Intersection:

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

Sort Containment:

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

Sort Refinement:

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

Relative Sort Intersection:

(S.4) $$\frac{\psi \ \& \ U \doteq X \ \& \ U \doteq X'}{\psi \ \& \ U \doteq X \ \& \ U \doteq X' \ \& \ U : s \wedge s'}$$ *if $X : s \in \phi$, $X' : s' \in \phi$, $s \wedge s' < s$, $s \wedge s' < s'$, and $U : s'' \notin \psi$, for any sort $s''$*

Sort Inconsistency:

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

Fig. 2. Simplification relative to $\phi$: Sorts.

Relative Variable Elimination:

(E.1) $$\frac{\psi \ \& \ U \doteq X \ \& \ V \doteq X}{\psi[U/V] \ \& \ U \doteq X \ \& \ V \doteq X}$$ *if $V \in Var(\psi)$, $V \doteq X \notin \psi$, and $U \neq V$*

Equation Entailment:

(E.2) $$\frac{\psi \ \& \ U \doteq X \ \& \ U \doteq Y}{\psi \ \& \ U \doteq X}$$ *if $X = Y$ or if $X \doteq Y \in \phi$.*

Fig. 3. Simplification relative to $\phi$: Equations.

Extended Feature Decomposition:

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

Extended Sort Intersection 1:

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

Extended Sort Intersection 2:

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

Fig. 4. Rules extending basic simplification.

Clearly, while $\forall X\ \big(\top \rightarrow \exists U \exists V\ (\psi \ \&\ U \doteq X)\big)$ holds in $\Psi$ and $\mathcal{T}$, it does not hold in all OSF algebras where it is not guaranteed that every element is the $\ell$-image of some other element. In $\Psi$ (and $\mathcal{T}$), this is the case since any element $X$ is the $\ell$-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: Theorems 1 and 2.

## 3.1 Termination of Relative Simplification

To show that relative simplification of OSF constraints always terminates, we introduce an additional set of rules, shown in Figure 4, 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 $\psi$ relative to a context $\phi$ can be ''simulated'' by normalizing the formula $\phi \ \&\ \psi \ \&\ U \doteq X$ using basic simplification (Figure 7 in the Appendix) together with the rules of Figure 4. It is not a real simulation, however, since Rules (B.1)--(B.5) destroy the context as a side effect. 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 that is generated by the $\doteq$-pairs between variables in the constraint).

LEMMA 1. *The basic-simplification rules (B.1)--(B.5) extended with rules (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) [Aït-Kaci and

Podelski 1993b]: (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 that 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 that is to be simplified. □

LEMMA 2. *Let $\psi$ & $U \doteq X$ simplify to $\psi'$ relative to $\phi$ by a relative-simplification step not using Rule (F.4). Then, $\phi$ & $\psi$ & $X \doteq U$ simplifies to $\phi'$ & $\psi''$ by at most one extended basic-simplification step and a finite number of variable elimination (B.3), where $\psi'$ and $\psi''$ 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 $\psi$ simplify to $\psi'$ of the form $\psi$ & $U_1 \doteq X_1$ & $U_1 \doteq X_2$ by an application of Rule (F.4) relative to $\phi$. Then, $\psi$ & $U_1 \doteq X_1$ simplifies to the same constraint $\psi'$ by an application of Rule (F.3) relative to $\phi$.*

PROPOSITION 10. *The relative-simplification rules are terminating.*

PROOF. This is proved by induction on $n$, using Lemmas 2 and 3. For every relative-simplification chain $\psi_1$ & $U_1 \doteq X_1, \ldots, \psi_n$ & $U_n \doteq X_n$ relative to $\phi$, there exists an extended basic-simplification chain of length $n + k$, where $k \geq 0$. This chain starts with the basic constraint $\phi$ & $\psi$ & $X_1 \doteq U_1$ & $X \doteq U$, 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. □

## 3.2 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 11. *Let $\psi$ & $U \doteq X$ simplify to $\psi'$ relative to $\phi$. Then, $\phi$ & $\psi$ & $U \doteq X$ and $\exists\mathcal{V}.\ (\phi$ & $\psi')$ are equivalent.*

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

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

COROLLARY 5. (INVARIANCE OF RELATIVE SIMPLIFICATION) *If $\psi$ & $U \doteq X$ simplifies to $\psi'$ relative to $\phi$, then $\exists\mathcal{U}.\ (\phi$ & $\psi$ & $U \doteq X)$ 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.

Redundant Sort Elimination:

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

Redundant Feature Elimination:

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

Entailed Sort Redundancy Elimination:

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

Fig. 5. Redundancy elimination rules.

PROPOSITION 12. *The constraint $\psi$ is in normal form relative to $\phi$ if and only if all the following conditions are satisfied:*

---*$\psi$ is in solved form;*

---*a global variable $X$ may occur in $\psi$ only in the form $\_ \doteq X$;*

---*if $X \doteq \_ \in \phi$, then $X$ does not occur in $\psi$;*

---*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 $\{V \doteq X, V \doteq Y\} \subseteq \psi$, and $\{X' \doteq X.\ell, Y' \doteq Y.\ell\} \subseteq \phi$, then $\{W \doteq X', W \doteq Y'\} \subseteq \psi$, for some variable $W$;*

---*if $\{V \doteq X, V \doteq Y\} \subseteq \psi$, and $\{X : s_1, Y : s_2\} \subseteq \phi$, 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 13. *Let $\psi'$ be a normal form of $\psi$ & $U \doteq X$ relative to $\phi$. Let $\phi'$ be the constraint obtained from $\phi$ eliminating all redundancies according to the rules of Figure 5, and removing bindings $V \doteq \_$ of new variables introduced by (F.4). Then, the constraint $\phi'$ & $\psi'$ is a solved form of the constraint $\phi$ & $\psi$ & $U \doteq X$, up to variable renaming.*

PROOF. According to Proposition 11, $\phi$ & $\psi$ & $U \doteq X$ is equivalent to $\exists\mathcal{V}.\ \phi$ & $\psi'$, where $\mathcal{V}$ stands for the new variables. According to the last three conditions of Proposition 12, Rules (R.1), (R.2), or (R.3) perform equivalence transformations. Thus, if applications of these rules modify $\phi'$ to $\phi''$, then $\phi'$ & $\psi'$ is equivalent to $\phi''$ & $\psi'$.

According to the first four conditions of Proposition 12, $\phi''$ & $\psi'$ 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 that is exactly equivalent to $\phi$ & $\psi$ & $U \doteq X$, and not just up to existential quantification of new variables. $\Box$

COROLLARY 6. *If the normal form of $\psi$ & $U \doteq X$ relative to $\phi$ is not $\bot$, then $\phi$ & $\psi$ & $U \doteq X$ is satisfiable.*

PROOF. We showed elsewhere that a constraint is satisfiable if and only if it has a solved form[Aït-Kaci and Podelski 1993b]. That is, its basic normal form is different from $\bot$. The statement then follows from Proposition 13. $\Box$

THEOREM 1. (DISENTAILMENT) *Let $\psi'$ be a normal form of $\psi$ & $U \doteq X$ relative to $\phi$. Then, $\phi$ disentails $\exists\mathcal{U}.\ (\psi$ & $U \doteq X)$ if and only if $\psi' = \bot$.*

PROOF. If $\psi' = \bot$, then $\forall\mathcal{X}\ (\phi \rightarrow \neg\exists\mathcal{U}\exists\mathcal{V}.\ \psi')$ is valid. From Corollary 5, it follows that $\forall\mathcal{X}\ (\phi \rightarrow \neg\exists\mathcal{U}.\ \psi$ & $U \doteq X)$ is valid, too. If $\psi' \neq \bot$, then Corollary 6 can be applied. $\Box$

PROPOSITION 14. *If the normal form $\psi'$ of $\psi$ & $U \doteq X$ relative to $\phi$ 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 $\psi'$ means that one of the three following cases is true, for some $V \in Var(\psi')$ bound to some $X \in Var(\phi)$, *i.e.*, $V \doteq X \in \psi'$:

(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 $\phi'$ such that $\phi$ & $\phi'$ is satisfiable and disentails $\psi'$. Then, $\phi$ & $\phi'$ also disentails $\exists\mathcal{U}.\ (\psi$ & $U \doteq X)$, *i.e.*, $\phi$ & $\phi' \rightarrow \neg\exists U.\ (\psi$ & $U \doteq X)$ is valid. Clearly, this is sufficient to show that $\phi$ & $\neg\exists\mathcal{U}.\ (\psi$ & $U \doteq X)$ is satisfiable.

(1) $V : s \in \psi'$; then, according to the third condition of Proposition 12, $\phi$ 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 $s \wedge s'' = \bot$. In the second case, we choose $s''$ such that $s \wedge s'' = \bot$ and $s'' \leq s'$.

(2) $V \doteq X$ & $V \doteq Y \in \psi'$; then, either $V : s \in \psi'$ and we are in Case (2), or, according to the last condition of Proposition 12, at most one of $X$ and $Y$ is sorted in $\phi$. If $Y : s \in \phi$, we set $\phi' = X : s'$ for some sort $s'$ such that $s \wedge s' = \bot$. If none of $X$ and $Y$ is sorted in $\phi$, we set $\phi' = Y : s$ & $X : s'$ for some sorts $s, s'$ such that $s \wedge s' = \bot$.

(3) $V.\ell_1 \doteq V_1 \in \psi'$; then, $\phi$ contains no feature constraint $X.\ell_1 \doteq \_$, according to the fourth condition of Proposition 12. Without loss of generality, we can assume that $\psi$ does not contain redundant conjuncts.[8] There exists a sort $s$ such that $\psi$ 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 $n \geq 1$.

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 $s \wedge s' = \bot$. $\Box$

[8]That is, we assume that every variable in $\psi$ has at least one sort constraint and that redundant constraints in $\psi$ are removed. A redundant constraint in $\psi$ is one of the form $X.\ell \doteq Y$ & $Y : \top$ where $Y$ does not occur elsewhere in $\psi$. 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 $\phi$ does not depend on whether features are interpreted as total or partial functions.

THEOREM 2. (ENTAILMENT) *Let $\psi'$ be a normal form of $\psi$ relative to $\phi$. Then, $\phi$ entails $\exists\mathcal{U}.\ (\psi \ \&\ U \doteq X)$ if and only if $\psi'$ is a functional binding. Moreover, $\phi \ \&\ \psi'$ is a solved OSF constraint.*

PROOF. If $\psi'$ is a conjunction of equations representing a functional binding, then $\exists\mathcal{U}\exists\mathcal{V}.\ \psi'$ is valid; thus, so is $\phi \rightarrow \exists U \exists\mathcal{V}.\ \psi'$. By invariance of relative simplification (Corollary 5), it follows that $\phi \rightarrow \exists\mathcal{U}.\ \psi$ is valid, too.

If $\psi'$ has a different form then, either $\psi' = \bot$, or $\psi'$ contains conjuncts that are not a functional binding. The fact that $\phi \rightarrow \exists\mathcal{U}.\ \psi$ is not valid is trivial in the first case. In the other case, since the context $\phi$ is always assumed in solved form and, thus, satisfiable, then it follows from Proposition 14. □

COROLLARY 7. *Let $\psi'$ be the relative-simplification normal form of $\psi \ \&\ U \doteq X$ relative to $\phi$. 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 2 and Proposition 13. □

## 3.3 Independence

The following theorem states that the OSF constraint system has the independence property [Lassez et al. 1988]. It is well known that in any constraint system with this property it is possible to solve constraints that 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 $\phi$ 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, $\phi$ entails $\exists\mathcal{U}_i.\ \psi_i$ if and only if $\phi$ 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 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 $\psi_i$ does not contain redundant constraints (see footnote 8).

THEOREM 3. (INDEPENDENCE) *A constraint $\phi$ 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 $\phi \ \&\ \neg\exists\mathcal{U}_i.\ (\psi_i \ \&\ U_i \doteq X_i)$ is satisfiable for every $i$, then $\phi \ \&\ \bigwedge_{i=1,\ldots,k}\neg\exists\mathcal{U}_i.\ (\psi_i \ \&\ U_i \doteq X)$ is satisfiable.

Extending the proof technique of Proposition 14, we will find a constraint $\phi'$ such that $\phi \ \&\ \phi'$ is satisfiable and disentails $\psi_i'$, for all $i = 1,\ldots,k$. As a consequence, $\phi \ \&\ \phi'$ also disentails $\exists\mathcal{U}_i.\ (\psi_i \ \&\ U_i \doteq X_i)$. That is, $\phi \ \&\ \phi' \rightarrow \neg\exists\mathcal{U}_i.\ (\psi_i \ \&\ U_i \doteq X_i)$ is valid. Clearly, this shows that $\phi \ \&\ \bigwedge_{i=1,\ldots,k}\neg\exists\mathcal{U}_i.\ \psi_i \ \&\ U_i \doteq X$ is satisfiable.

According to Theorem 2, if $\phi \ \&\ \neg\exists\mathcal{U}_i.\ (\psi_i \ \&\ U_i \doteq X_i)$ is satisfiable, then $\psi_i'$, the normal form of $\psi_i \ \&\ U_i \doteq X_i$ relative to $\phi$ is not a conjunction of equations representing a functional binding.

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

(1) $\psi_i'$ contains a sort constraint on $V_i$, say, $V_i : s_i$, or,

(2) $\psi'_i$ contains two equations on $V_i$, say, $V_i \doteq X_i$ & $V_i \doteq Y_i$, or,

(3) $\psi'_i$ contains a feature constraint on $V_i$, say, $V_i.\ell_i \doteq W_i$.

(1) If $V_i : s_i \in \psi'_i$, then $\phi$ 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 12. 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 $\phi$ the sort constraint $X_i : s''_i$ where $s''_i$ is some sort that is incompatible with those in the sort constraints $U_{i_j} : s_{i_j}$, and, in case $X_i : s'_i \in \phi$, is furthermore a subsort of $s'_i$, $s''_i \leq s'_i$. We can always add such a sort $s''_i$ to the signature without changing the meaning of the program thanks to our definition of OSF algebra. That is, we do not require that a sort denotes the union of the sets denoted by its subsorts (although the dual holds true: a sort denotes the intersection of the sets denoted by its supersorts).

(2) If $V_i \doteq X_i$ & $V_i \doteq Y_i \in \psi'_i$, and $V_i : s_i \notin \psi'_i$ (otherwise we are in Case (2)), then we add to $\phi'$ the conjuncts $X_i.\ell_i \doteq Z_i$ & $Z_i \in s$ & $Y_i.\ell_i \doteq Z'_i$ & $Z'_i \in s'$. Here $s$ and $s'$ are two incompatible sorts, and the $\ell_i$'s are pairwise different features that do not occur in $\phi$ and $\psi_i$, 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 $i \in I$, $\psi'_i$ 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 $\psi_i$ 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 $n \geq 1$. We add to $\phi'$ 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 $\psi'_{i_j}$ 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 $\psi'_{i_j}$ 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 $s_{i_j} \wedge s'_i = \bot$ for all $i_j$, $i_j = 1, \ldots, m$. $\square$

## 4. FUNCTIONAL APPLICATION OVER $\psi$-TERMS

In this section, we show the use of the general scheme of Section 2 on the specific instance of LIFE's functional applications. That is, we explicate how our general residuation scheme can be used to explain functional application over $\psi$-terms.

A $\psi$-term is a constraint describing a 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 $\psi$-term*. For example, $X : cons(tl \Rightarrow T : list)$ & $length(T) \doteq L$ & $L : even$ is a constrained $\psi$-term specifying lists of odd lengths.

A constrained $\psi$-term is an expression of the form $\psi$ & $C$ where $\psi$ is a $\psi$-term and $C$ a possibly empty conjunction of OSF constraints and relational atoms.[9]

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

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 $\psi$-terms and $e_1, \ldots, e_n$ are constrained $\psi$-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 $f$.

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 of $n+1$ 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 2, 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 2 and Proposition 8, respectively.

PROPOSITION 15. *The resolvent $R \,\&\, \phi \,\&\, Y : f(t)$ 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 16. *If, for $j = 1, \ldots, i$, the OSF constraint $X : p_j$ simplifies to the OSF constraint $\psi_j$ relative to $\phi \,\&\, X : t$ such that $\psi_1 = \bot, \;\ldots\;, \psi_{i-1} = \bot$, and $\psi_i$ is a functional binding,*[10] *then the resolvent $R \,\&\, \phi \,\&\, Y : f(t)$ is equivalent to the resolvent*

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

To express functional application in the framework of the calculus of subsumption and unification of $\psi$-terms, we use a fact that follows directly from Propositions 20 and 21. Namely, the implication $X : t \rightarrow \exists \mathcal{U}_i.\; X : p_i$ is valid if and only if the $\psi$-term $t$ is subsumed by the $\psi$-term $p_i$. The OSF constraint $X : t \,\&\, X : p_i$ is unsatisfiable if and only if the $\psi$-term $t$ is nonunifiable with the $\psi$-term $p_i$.

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

[10]Recall, from Section 3, 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.

PROPOSITION 17. *If the $\psi$-term $t$ is nonunifiable with the $\psi$-terms $p_1, \ldots, p_{i-1}$ and if it is subsumed by the $\psi$-term $p_i$, then the functional expression $f(t)$ is equivalent to the expression $e_i$ constrained by $t \doteq p_i$. Formally,*

$$Y : f(t) \;\leftrightarrow\; \exists\mathcal{U}_i \exists\mathcal{V}_i .\ Y : e_i \ \&\ t \doteq p_i \tag{7}$$

*is valid. If $t$ is nonunifiable with the $\psi$-terms $p_1, \ldots, p_n$, then $f(t)$ is equivalent to $\bot$.*

PROOF. The statement follows from Proposition 15 together with the fact that $\mathcal{A}, \alpha \models \exists X.\ (X : t \ \&\ X : p_i)$ if and only if $\mathcal{A}, \alpha \models t \doteq p_i$. □

## 4.1 Endomorphisms and Functional Application

We have related functional reduction to the view of $\psi$-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 $\psi$-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 $f(p) \rightarrow e$, then the function applied to the term $t$ yields the expression $\sigma(e)$ if the term $t$ matches the pattern $p$ via the matching substitution $\sigma$, *i.e.*, $f(t) = \sigma(e)$ if $\sigma(p) = t$. This is not so obvious for $\psi$-terms. Let us take, for example, the identity function on $\psi$-terms, which is defined in the form $f(X : \top) \rightarrow X : \top$. When applied to the $\psi$-term $t = X : s(\ell \Rightarrow X' : s)$, the function returns the same $\psi$-term. However, this does not exhibit, as expected for first-order terms, a substitution $\sigma$ such that $\sigma(X : \top) = X : s(\ell \Rightarrow X' : s)$. Rather, the instantiation map from $p$ to $t$ is expressed thanks to a more general notion of refinement that we describe next.

Recall that an approximation ordering $\sqsubseteq$ on $\psi$-terms is induced by the ordering on $\Psi$, the OSF graph algebra (see Section 5). An endomorphism $\gamma$ is said to be principal in a set of endomorphisms if for every endomorphism $\gamma'$ in this set there exists an endomorphism $\rho$ such that $\gamma' = \rho \circ \gamma$.

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

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

Let $f(p) \rightarrow e$ define the function $f$, and let $t$ be a $\psi$-term such that $p \sqsubseteq t$. Let $\gamma$ 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 18. *If no $\psi$-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 $\psi$-term $\gamma(e_i)$, where $\gamma$ is a principal endomorphism mapping $p_i$ on $t$, i.e.,*

$$f(t) = \gamma(e_i), \quad \textit{if } \gamma(p_i) = t. \tag{8}$$

*If no $\psi$-term is approximated by both $t$ and $p_i$ for $i = 1, \ldots, n$, then the functional $\psi$-term $f(t)$ is $\bot$.*[11]

[11]Note that, in (8), 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 that is equivalent up to existential quantification of new variables.

PROOF. By Proposition 20, we know that the conditions in Proposition 18 on the OSF graphs are equivalent to the conditions in Proposition 17 on the corresponding $\psi$-terms. In particular, this implies the existence of the principal endomorphism $\gamma$ with $\gamma(p_i) = t$. From Propositions 21 and 22, we know that $X : t$ & $X : p_i$ is equivalent to $X : \gamma(p_i)$ & $\phi$ where $\phi$ is a functional binding (of variables of $p_i$ to variables of $t$). Moreover, the equivalence

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

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 $\phi$, this formula is equivalent to $\gamma(Y_0 : e_i)$ & $X : \gamma(p_i)$. Thus, Equation (8) follows from Proposition 17. □

The proposition above justifies the intuition of functional application over $\psi$-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 to which $f(t)$ is rewritten.

The variables in $e_i$ that 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 $\psi$-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 $\gamma$ maps $U : \top$ on $X : s(\ell \Rightarrow X' : s)$ and is the identity elsewhere. In particular, $\gamma$ does not unnecessarily refine the sort of $U'$.

The endomorphic approximation ordering is very interesting when used on the graph representations of $\psi$-terms. It is in fact an immediate generalization of first-order term matching. More conveniently, if a graph $\psi_1$ approximates a graph $\psi_2$ with an endomorphism $\gamma$, this approximation is characterized exactly by a mapping $\gamma_{\mathcal{V}} : Var(\psi_1) \mapsto Var(\psi_2)$ that can be constructed inductively as follows:[12]

(1) $\gamma_{\mathcal{V}}(Root(\psi_1)) = Root(\psi_2)$;

(2) for every $X_1 \in Var(\psi_1)$ 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 $\gamma$ as $\gamma_{\mathcal{V}}(Root(\psi)) = Root(\gamma(\psi))$, for all $\psi$ in $\Psi$.

For example, provided that *married_person* $<$ *person*, *smith* $<$ *name*, *male* $<$ *gender*, and *female* $<$ *gender*, then the term

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

approximates the term

[12]Given an OSF graph $\psi$, we use the notation $Root(\psi)$ to designate its root variable, $Sort_\psi(X)$ to designate the sort of the variable $X$ in $\psi$, and $\ell_\psi(X) = Y$ to express the fact that $\psi$ has an arc labeled $\ell$ between nodes $X$ and $Y$. (When no ambiguity may arise, we omit the subscript $\psi$.)

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

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 7 are sufficient.[13] Evidently, if the basic unification yields $\perp$, then this shows disentailment. Otherwise, we will exhibit conditions on the obtained variable bindings that 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 $\psi$-terms $\psi_1$ and $\psi_2$, 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.

(1) let $\psi_1'$ be the $\psi$-term obtained from $\psi_1$ by completing it with new variables sorted with $\top$ at any path occurrence of $\psi_2$ that is *not* in $\psi_1$.

(2) Let $\phi$ be the normal form of the OSF clause: $\mathit{Root}(\psi_1) \doteq \mathit{Root}(\psi_2)\ \&\ \psi_1'\ \&\ \psi_2$.

(3) If $\phi$ is not $\perp$ then let $\gamma_1$ (resp., $\gamma_2$) be the canonical surjection of $\mathit{Var}(\psi_1')$ (resp.,$\mathit{Var}(\psi_2)$) onto the coreference classes of $\phi$, *i.e.*, the function that maps a variable to its coreference class.

Then,

THEOREM 4. *$\psi_2 \sqsubseteq \psi_1$ with principal OSF endomorphism $\gamma$ if and only if $\phi$ is not $\perp$ and $\gamma_1$ is a sort-preserving bijection.*[14] *Then, $\gamma_{\mathcal{V}} = \gamma_1^{-1} \circ \gamma_2 \ :\ \mathit{Var}(\psi_2) \mapsto \mathit{Var}(\psi_1)$ is the corresponding endomorphic variable mapping.*

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

If $\gamma_1$ is a sort-preserving bijection, then for every variable $X$ of $\phi$, $\gamma_1^{-1}([X])$ is the unique variable of $\psi_1'$ that is element of this coreference class. Then, we can transform $\phi$ into an equivalent formula $\phi$ by replacing every variable $X$ by $\gamma_1^{-1}([X])$ in $\psi$ and replacing $\varepsilon$ by $\varepsilon' = \bigwedge_{X \in \mathcal{X}_2} X \doteq \gamma_{\mathcal{V}}(X)$. Note that this is an equivalence-preserving transformation since $\phi$ is, by construction, of the form $\psi_1'\ \&\ \varepsilon'$, and the coreference relations generated by $\varepsilon$ and $\varepsilon'$ are identical. It is important to realize that this statement would not be true if we had used $\psi_1$ instead of $\psi_1'$. Indeed, then, $\phi$ would have been of the form $\psi_1\ \&\ \psi'\ \&\ \varepsilon'$ where $\psi'$ consisted of additional feature constraints corresponding to occurrences of $\psi_2$ missing in $\psi_1$.

[13]See Appendix.

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

Clearly, it is true that $\forall \mathcal{X}_1.\ (\psi'_1 \ \leftrightarrow\ \exists \mathcal{X}_2.\ (\psi'_1\ \&\ \varepsilon'))$. This shows that $\forall \mathcal{X}_1.\ (\psi'_1 \ \leftrightarrow\ \exists \mathcal{X}_2.\ (Root(\psi_1) \doteq Root(\psi_2)\ \&\ \psi'_1\ \&\ \psi_2))$ and, thus, $\forall \mathcal{X}_1.\ (\psi'_1 \ \rightarrow\ \exists \mathcal{X}_2.\ (Root(\psi_1) \doteq Root(\psi_2)\ \&\ \psi_2))$ is valid, and thus $\psi_2 \sqsubseteq \psi_1$.

Conversely, if $\psi_2 \sqsubseteq \psi_1$, then also $\forall \mathcal{X}_1.\ (\psi'_1 \ \rightarrow\ \exists \mathcal{X}_2.\ (Root(\psi_1) \doteq Root(\psi_2)\ \&\ \psi_2))$ is valid, and, thus, also $\forall \mathcal{X}_1.\ (\psi'_1 \ \leftrightarrow\ \exists \mathcal{X}_2.\ \phi)$. But this means (1) that $\phi$ does not contain equalities binding two variables of $\psi_1$ to each other and (2) that $\phi$ does not contain a sort constraint stronger than the one in $\psi_1$ on the (same or corresponding) variable of $\psi_1$. □

Note that the completion of $\psi_1$ with occurrences from $\psi_2$ done in Step 1 is necessary to determine the bijection $\gamma_1$, and thus the mapping $\gamma_\mathcal{V}$, 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 $\psi_1$ instead of the completed $\psi'_1 = f(a, h(U))$ and normalizing does result in a sort-preserving bijection while, using $\psi'_1$, it does not.

## 4.2 Semantics of Functional Application

If a function is defined over $\psi$-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 functional application given an OSF algebra $\mathcal{A}$ and a valuation $\alpha$ 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, 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 $\psi$-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 $\alpha$ where $[\![p_i]\!]^{\mathcal{A},\alpha}$ is disjoint from $[\![p_1]\!]^{\mathcal{A}}, \ldots, [\![p_{i-1}]\!]^{\mathcal{A}}$. Implicitly, we make the $\psi$-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 $\psi$-terms $\psi_1$ and $\psi_2$, 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 $\alpha$ 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 $D^{\mathcal{A}}$,

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

The denotation of the *functional application* of $f$ on the $\psi$-term $t$ under a valuation $\alpha$ 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 $\alpha(Y) = f^{\mathcal{A},\beta}\big(\alpha(X)\big)$ for some $\beta \in Val(\mathcal{A})$.

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

*Example* 1. We define the identity function *id* on $\psi$-terms by the rule $id(X : \top) \rightarrow X : \top$. Then, $id^{\mathcal{A}}(D) = D$ for any subset $D \subseteq D^{\mathcal{A}}$. If we confuse singletons and their elements, we may write $id^{\mathcal{A}}(d) = d$ for elements $d$ of the domain of $\mathcal{A}$. If $s$ is any sort, then $[\![id(X : s)]\!]^{\mathcal{A}} = [\![X : s]\!]^{\mathcal{A}} = s^{\mathcal{A}}$. In fact, the denotation of the function *id* applied on any $\psi$-term is equal to the denotation of the $\psi$-term. The denotation under a given valuation $\alpha$ is the value of the element on which the function is applied, $[\![id(X : \top)]\!]^{\mathcal{A},\alpha} = [\![X : \top]\!]^{\mathcal{A},\alpha} = \{\alpha(X)\}$.

*Example* 2. We define the function *any* by the rule: $any(X : \top) \rightarrow Y : \top$. The application of this function on a $\psi$-term $\psi$ yields always the sort $\top$, $any(\psi) = Y : \top = \top$. Note that $[\![any]\!]^{\mathcal{A},\alpha}\big(\alpha(X)\big) = \alpha(Y)$. 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* 3. For a fixed sort $s$, we define the function $sort_s$ by the rule $sort_s(X : s) \rightarrow X : \top$. 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 $\alpha$ such that $f^{\mathcal{A},\alpha}(d) = d'$.

## 5. CONCLUSION

Our original motivation 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. We developed a general residuation framework for guarded Horn clauses over arbitrary constraint systems with an incremental constraint 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. Second, we gave a correct and complete operational scheme for testing entailment and disentailment of order-sorted feature constraints. To that end, we introduced a general technique, that we dubbed relative simplification, that amounts to normalizing a formula in the context of another. Last, 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 article gives a natural meaning to function symbols, it does not consider the latter as ‘‘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 by Maher [1987] and Saraswat and Rinard [1990], and realized to some extent in AKL [Haridi and Janson 1990]. 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.

## APPENDIX

We give here a detailed summary of the technical terminology and notation used in this article. For a thorough investigation of these notions, the reader is referred to Aït-Kaci and Podelski [1993b].

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 that are part of a logical formalism, they are also quite primitive to constitute a low-level implementation logic.[15] We then formalize $\psi$-terms since 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 article.

### 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 $\perp$;

---$\leq$ is a decidable partial order on $\mathcal{S}$ such that $\perp$ is the least and $\top$ is the greatest element;

---$\langle \mathcal{S}, \leq, \wedge \rangle$ is a lower semilattice ($s \wedge s'$ 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 nonempty 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 $\perp^{\mathcal{A}} = \emptyset$;

---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 $\ell$ in $\mathcal{F}$, $\ell^{\mathcal{A}}$ is a total unary function from the domain into the domain, *i.e.*, $\ell^{\mathcal{A}} : D^{\mathcal{A}} \mapsto D^{\mathcal{A}}$.

[15] In fact, the reader familiar with implementation techniques of Prolog [Aït-Kaci 1991] 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 homomorphic tranformation preserving its structure appropriately. Namely,

*Definition* 3. (*OSF Homomorphism*) An OSF *homomorphism* $\gamma : \mathcal{A} \mapsto \mathcal{B}$ between two OSF algebras $\mathcal{A}$ and $\mathcal{B}$ is a function $\gamma : D^{\mathcal{A}} \mapsto D^{\mathcal{B}}$ such that

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

---$\gamma\big(s^{\mathcal{A}}\big) \subseteq s^{\mathcal{B}}$.

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* 4. (*OSF Constraint*) An *atomic* OSF constraint is one of (1) $X : s$, (2) $X \doteq X'$, or (3) $X.\ell \doteq X'$, where $X$ and $X'$ are variables in $\mathcal{V}$, $s$ is a sort in $\mathcal{S}$, and $\ell$ 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 $\ell$ of $X$.'' The set $Var(\phi)$ of variables occurring in an OSF constraint $\phi$ 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.

An OSF constraint $\phi$ is *satisfiable* in an OSF algebra $\mathcal{A}$, if there exists a valuation $\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}$ such that $\mathcal{A}, \alpha \models \phi$, where:

$$\begin{array}{lll}
\mathcal{A}, \alpha \models X : s & \text{if and only if} & \alpha(X) \in s^{\mathcal{A}};\\
\mathcal{A}, \alpha \models X \doteq Y & \text{if and only if} & \alpha(X) = \alpha(Y);\\
\mathcal{A}, \alpha \models X.\ell \doteq Y & \text{if and only if} & \ell^{\mathcal{A}}(\alpha(X)) = \alpha(Y);\\
\mathcal{A}, \alpha \models \phi \;\&\; \phi' & \text{if and only if} & \mathcal{A}, \alpha \models \phi \text{ and } \mathcal{A}, \alpha \models \phi'.
\end{array}$$

## $\psi$-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* 5. ($\psi$-*Term*) A $\psi$-term $\psi$ 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 $\psi$;

---$s$ 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 $\psi$-terms; and,

---no variable $Y$ occurring in $\psi$ is the root variable of more than one nontrivial $\psi$-term (*i.e.*, different than $Y : \top$).

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

We can associate to a $\psi$-term

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

the OSF constraint

$$\begin{array}{lllllll} \phi(\psi) = & X : s & \& & X.\ell_1 \doteq Y_1 & \& \ldots \& & X.\ell_n \doteq Y_n \\ & & \& & \phi(\psi_1) & \& \ldots \& & \phi(\psi_n) \end{array}$$

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 $\psi$-term $\psi$, and refer to the OSF constraint as the *dissolved $\psi$-term*. We will often deliberately confuse a $\psi$-term $\psi$ with its dissolved form $\phi(\psi)$ and refer to $\phi(\psi)$ simply as $\psi$.

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

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

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

A $\psi$-term $\psi$ with root $X$ corresponds to a unique rooted graph $g$ that is the direct translation of the constraint $\psi$ together with an indication of the root. The nodes of $g$ are exactly the variables of $\psi$. A node $Z$ is labeled by the sort $s$ if the conjunction $\psi$ contains a nontrivial 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 $(Y, Z)$ that is labeled by the feature $\ell$. The root of $g$ is the node $X$. Clearly, $g$ is the natural graphical representation of $\psi$. For example, the $\psi$-term

$$\begin{array}{l} X_1 : person(name \Rightarrow X_2 : id(first \Rightarrow X_3 : string, \\ \qquad\qquad\qquad\qquad\qquad\qquad last \Rightarrow X_4 : string), \\ \qquad\qquad spouse \Rightarrow X_5 : person(name \Rightarrow X_6 : id(last \Rightarrow X_4), \\ \qquad\qquad\qquad\qquad\qquad\qquad spouse \Rightarrow X_1)). \end{array}$$

corresponds to the OSF graph shown in Figure 6.

## 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 $\psi$-terms. They turn out to be *canonical interpretations* for OSF constraints.[16]

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

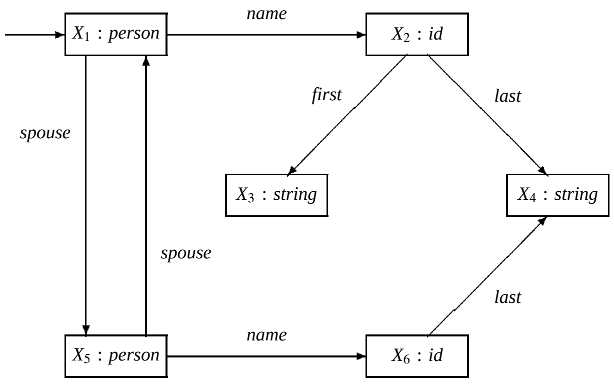

Fig. 6. An OSF graph.

The most immediate syntactic OSF interpretation is the OSF algebra $\Psi$ of $\psi$-terms. The domain of $\Psi$ is the set of all $\psi$-terms, up to graph representation. That is, we identify $\psi$-terms as values of $\Psi$ if they are represented by the same graph. For example, the two $\psi$-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

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

and features $\ell \in \mathcal{F}$ are interpreted as functions $\ell^{\Psi} : D^{\Psi} \mapsto D^{\Psi}$ as follows. Let $\psi$ be a $\psi$-term and $g$ its graph. If $(X, Y)$ is the edge of $g$ labeled by $\ell$, then $\ell^{\Psi}(g)$ is the $\psi$-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 that are not reachable by a directed path from the node $Y$.

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

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

We obtain two other examples of OSF algebras when we factorize the $\psi$-term domain by further identifying values. The first one identifies two $\psi$-terms that are equal up to variable renaming. The obtained domain obviously spans an OSF algebra. We call this OSF algebra $\Psi_0$.

The second one is obtained from $\Psi_0$ by further identifying two $\psi$-terms if their (possibly infinite) tree unfoldings are equal. A tree unfolding is obtained from a $\psi$-term by associating a unique node to every feature path. It is well known that a rooted directed graph represents a unique rational tree [Courcelle 1983]. In our case, we obtain trees whose nodes are labeled by sorts and whose 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 $\mathcal{T}$.[17]

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, that satisfy the *Sort Axiom* saying, for all sorts $s$ and $s'$,

[17] $\mathcal{T}$ is essentially the feature tree structure of Aït-Kaci et al. [1994] and Backofen and Smolka [1992, Smolka and Treinen [1992]. The difference lies in our using partially ordered sorts and total, as opposed to partial, features.

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

Furthermore, both $\Psi_0$ and $\mathcal{T}$ satisfy a *Constructibility Axiom* stating essentially the satisfiability of any OSF constraint $\phi$ coming from dissolving a $\psi$-term $\psi$. More precisely, if $\mathcal{X} = Var(\phi)$ and, for $i = 1, \ldots, n$, $X_i.\ell_i \doteq Y \notin \phi$ for any variable $Y$, and $Y_i \notin Var(\phi)$, 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 that 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 that 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 $\phi = X.\ell \doteq X$.

## OSF Unification

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

*Definition* 6. (*Solved OSF Constraints*) An OSF constraint $\phi$ is called solved if for every variable $X$, $\phi$ 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 $\ell$; and,
---no other occurrence of the variable $X$ if it contains the equality constraint $X \doteq Y$.

We can show that an OSF constraint in solved form is always satisfiable [Aït-Kaci and Podelski 1993b]. Now, by Definition 5, the OSF constraint obtained as the dissolved form of any $\psi$-term $\psi$ is *de facto* in solved form.[18] Hence, such a constraint is always satisfiable. It is so, in particular, in the canonical interpretation $\Psi$ with, interestingly enough, the valuation that assigns to each variable $X$ in $\psi$ the value in $D^{\Psi}$ that is the very $\psi$-term rooted in $X$ in $\psi$. For this reason, a $\psi$-term can also be seen as a variable substitution.

Given an OSF constraint $\phi$, it can be normalized by choosing nondeterministically and applying any applicable rule among the transformations rules shown in Figure 7 until none applies. A rule transforms the numerator into the denominator. The expression $\phi[X/Y]$ stands for the formula obtained from $\phi$ after replacing all occurrences of $Y$ by $X$.

THEOREM 5. (OSF CONSTRAINT NORMALIZATION) *The rules of Figure 7 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.*

For our purposes, the constraint $\phi$ to be normalized will be of the form $\psi_1 \;\&\; \psi_2 \;\&\; X_1 \doteq X_2$; *i.e.*, the conjunction of the dissolved $\psi$-terms $\psi_1$ and $\psi_2$ together with an equation

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

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) $$\frac{\psi \;\&\; U \doteq V}{\psi[V/U] \;\&\; U \doteq V}$$ *if $U \in Var(\psi)$ and $U \neq V$*

Inconsistent Sort:

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

Variable Clean-up:

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

Fig. 7. Basic simplification.

identifying their root variables $X_1$ and $X_2$. If $\phi$ normalizes to the *false* constraint, then the two $\psi$-terms are nonunifiable. Otherwise, the resulting solved OSF constraint is a conjunction of equality constraints and of the dissolved form of some $\psi$-term. This $\psi$-term is *the most general unifier* of $\psi_1$ and $\psi_2$, up to variable renaming. We shall see that this $\psi$-term has two equivalent order-theoretic characterizations (see Propositions 21 and 22).

## OSF Orderings

In this section, we first introduce the notion of *endomorphic approximation* that 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* 7. (*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,

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

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

We note that endomorphisms on $\Psi$ 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 $\gamma$ 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 holds [Aït-Kaci and Podelski 1993b].

PROPOSITION 19. ($\psi$-TERMS AS FILTERS) *The denotation of a $\psi$-term in $\Psi$ is the set of all $\psi$-terms it approximates, i.e.,*

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

The next ordering is the type ordering on $\psi$-terms that we informally called ''more specific than'' in Section 1.2.

*Definition* 8. (*$\psi$-Term Subsumption*) A $\psi$-term $\psi$ is *subsumed by* a $\psi$-term $\psi'$ if and only if the denotation of $\psi$ is contained in that of $\psi'$ in all interpretations. Formally,

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

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

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

The next and last ordering is a logical ordering on $\psi$-terms.

*Definition* 9. (*$\psi$-Term Entailment*) A $\psi$-term $\psi$ *entails* a $\psi$-term $\psi'$ if and only if, as constraints, $\psi$ implies the conjunction of $\psi'$ and $X \doteq X'$; more precisely,

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

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

It is important to realize that this formulation is not actually correct in general. Here, we limit the statement to the validity of the implication in the OSF algebra $\Psi$ only (namely, using $\models_{\Psi}$). This would not be sufficient in the more general case of arbitrary OSF constraints [Aït-Kaci and Podelski 1993b]. This weaker form is acceptable here only because the constraints in question are obtained by dissolving $\psi$-terms and because their root variables are bound together.

PROPOSITION 20. (SEMANTIC TRANSPARENCY OF ORDERINGS) *The four following statements are equivalent.*

| | | |
|---|---|---|
| (1) | $\psi \sqsubseteq \psi'$ | ($\psi$ is an approximation of $\psi'$); |
| (2) | $\psi' \leq \psi$ | ($\psi'$ is a subtype of $\psi$); |
| (3) | $\psi' \succeq \psi$ | ($\psi$ entails $\psi'$); |
| (4) | $[\![\psi]\!]^{\Psi} \subseteq [\![\psi']\!]^{\Psi}$ | (the set of $\psi$-terms filtered by $\psi$ is contained in the set of $\psi$-terms filtered by $\psi'$). |

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

PROPOSITION 21. ($\psi$-TERM UNIFICATION) *The normal form $\phi$ 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, $\phi$ is the conjunction of equality constraints and of the dissolved version of some $\psi$-term $\psi$. This $\psi$-term is the $\leq$-GLB of $\psi_1$ and $\psi_2$ up to variable renaming, i.e., $[\![\psi]\!]^{\mathcal{A}} = [\![\psi_1]\!]^{\mathcal{A}} \cap [\![\psi_2]\!]^{\mathcal{A}}$.*

PROPOSITION 22. (($\sqsubseteq$-LUB OF TWO $\psi$-TERMS)) *The $\psi$-term $\psi$ above is approximated by both $\psi_1$ and $\psi_2$ and is the least $\psi$-term for $\sqsubseteq$ (i.e., approximating all other ones) with this property.*

## ACKNOWLEDGMENTS

We are grateful to Michael Maher for detailed comments on the form and contents of the material of Section 2, and to Gert Smolka for stimulating discussions and his continuing collaboration on issues related to the topic of this work. Many thanks also to Ralf Treinen for reading all technical details thoroughly and reporting many errors and suggesting corrections.

We also wish to express our debt to the anonymous referees for their thorough and constructive criticisms. As well, we express our gratitude to Andrew Appel and John Mitchell for their thoughtful advice and the care they put in their editorial work.

Finally, we thank the members of the Paradise Project at PRL for many discussions related to the contents of this article. 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.

## REFERENCES

AT-KACI, H. 1991. *Warren's Abstract Machine, A Tutorial Reconstruction*. MIT Press, Cambridge, Mass.

AT-KACI, H. 1986. An algebraic semantics approach to the effective resolution of type equations. *Theor. Comput. Sci. 45*, 293--351.

AT-KACI, H. AND NASR, R. 1989. Integrating logic and functional programming. *Lisp Symb. Comput. 2*, 51--89.

AT-KACI, H. AND NASR, R. 1986. LOGIN: A logic programming language with built-in inheritance. *Logic Program. 3*, 185--215.

AT-KACI, H. AND PODELSKI, A. 1993a. Entailment and disentailment of order-sorted feature constraints. In *Proceedings of the 4th International Conference on Logic Programming and Automated Reasoning*. Lecture Notes in Computer Science, vol. 698. Springer-Verlag, New York, 1--18.

AT-KACI, H. AND PODELSKI, A. 1993b. Towards a meaning of LIFE. *J. Logic Program. 16,* 3-4 (July-Aug.), 195--234.

AT-KACI, H. AND PODELSKI, A. 1991. Functions as passive constraints in LIFE. PRL Res. Rep. 13 (June), Paris Research Laboratory, Digital Equipment Corporation, Rueil-Malmaison, France.

AT-KACI, H., PODELSKI, A., AND SMOLKA, G. 1994. A feature constraint system for logic programming with entailment. *Theor. Comput. Sci. 122*, 263--283.

BACKOFEN, R. AND SMOLKA, G. 1992. A complete and decidable feature theory. DFKI Res. Rep. RR-30-92, German Research Center for Artificial Intelligence, Saarbrücken, Germany.

BONNIER, S. AND MALUSZYNSKI, J. 1988. Towards a clean amalgamation of logic programs with external procedures. In *Logic Programming. Proceedings of the 5th International Conference and Symposium*. MIT Press, Cambridge, Mass., 311--326.

CLARK, K. L. 1978. Negation as failure. In *Logic and Data Bases*. Plenum Press, New York, 293--322.

COLMERAUER, A. 1984. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*. ICOT, Tokyo, 85--99.

COLMERAUER, A. 1982a. Prolog and infinite trees. In *Logic Programming*. Academic Press, New York, 153--172.

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

COURCELLE, B. 1983. Fundamental properties of infinite trees. *Theor. Comput. Sci. 25*, 95--169.

HARIDI, S. AND JANSON, S. 1990. Kernel Andorra Prolog and its computation model. In *Logic Programming, Proceedings of the 7th International Conference*. MIT Press, Cambridge, Mass., 31--46.

HARIDI, S., JANSON, S., AND PALAMIDESSI, C. 1992. Structural operational semantics for AKL. *Fut. Gen. Comput. Syst.*, 409--421.

HARPER, R., MILNER, R., AND TOFTE, M. 1988. The definition of standard ML -- Version 2. Rep. LFCS-88-62, Univ. of Edinburgh, Edinburgh, UK.

HOHFELD, M. AND SMOLKA, G. 1988. Definite relations over constraint languages. LILOG Rep. 53 (October), IWBS, IBM Deutschland, Stuttgart, Germany.

JAFFAR, J. AND LASSEZ, J.-L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*. ACM, New York, 111--119.

LASSEZ, J.-L., MAHER, M., AND MARIOTT, K. 1988. Unification revisited. In *Foundations of Deductive Databases and Logic Programming*. Morgan-Kaufmann, Los Altos, Calif., 587--625.

MAHER, M. 1987. Logic semantics for a class of committed-choice programs. In *Logic Programming, Proceedings of the 4th International Conference*. MIT Press, Cambridge, Mass., 858--876.

NAISH, L. 1986. Negation and quantifiers in NU-Prolog. In *Proceedings of the 3rd International Conference on Logic Programming*. Lecture Notes in Computer Science, vol. 225. Springer-Verlag, New York, 624--634.

RAMACHANDRAN, V. AND VAN HENTENRYCK, P. 1993. Incremental algorithms for constraint solving and entailment over rational trees. In *Proceedings of the 13th International Conference on Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science, vol. 761. Springer-Verlag, New York.

SANTOS COSTA, V., WARREN, D. H. D., AND YANG, R. 1991. 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*. ACM, New York, 83--93.

SARASWAT, V. AND RINARD, M. 1990. Concurrent constraint programming. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York, 232--245.

SMOLKA, G. 1991. Residuation and guarded rules for constraint logic programming. PRL Res. Rep. 12, Paris Research Laboratory, Digital Equipment Corporation, Rueil-Malmaison, France. To appear in *Constraint Logic Programming: Selected Research*. MIT Press, Cambridge, Mass.

SMOLKA, G. AND TREINEN, R. 1992. Records for logic programming. In *Logic Programming, Proceedings of the Joint International Conference and Symposium on Logic Programming*. MIT Press, Cambridge, Mass., 240--254.

Received January 1993; revised June 1993, September 1993, and October 1993; accepted December 1993