Logic Programming with Functions over Order-Sorted Feature Terms

Hassan Aït-Kaci and Andreas Podelski

Digital Equipment Corporation, Paris Research Laboratory
85, avenue Victor Hugo, 92500 Rueil-Malmaison, France

{hak,podelski}@prl.dec.com

Abstract. LIFE is an experimental programming language proposing to integrate logic programming, functional programming, and object-oriented programming. It replaces first-order terms with ψ-terms, data structures which allow computing with partial information. These are approximation structures denoting sets of values. LIFE further enriches the expressiveness of ψ-terms with functional dependency constraints. Whereas LIFE's relations defined as Horn-clauses use ψ-term unification for parameter-passing, LIFE's functions use ψ-term matching (*i.e.*, one-way unification). We 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.

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

This paper is an informal, albeit precise and detailed, overview of the operational mechanism underlying functional reduction in the context of a logic programming framework. It is formulated using order-sorted feature terms as data structures seen as constraints. This mechanism and data structure are used in the language LIFE [5].

1.1 The task

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

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

The first extension is based on ψ-terms which are attributed partially-ordered sorts denoting sets of objects [1, 2]. In particular, ψ-terms generalize first-order constructor terms in their rôle as data structures in that they are endowed with a unification operation denoting type intersection. This gives an elegant means to incorporate a calculus of multiple inheritance into symbolic programming. Importantly, the denotation-as-value of constructor terms is replaced by the denotation-as-approximation of ψ-terms. A consequence of this is that the notion of fully defined element, or ground term, is no longer available. Hence, such familiar tools as variable substitutions, instantiation, unification, *etc.*, must be reformulated in the new setting [5].

The second extension deals with building into the unification operation a means to reduce functional expressions using definitions of interpretable symbols over data patterns.[1] Our basic idea is that unification is no longer seen as an atomic operation by the resolution rule. Indeed, since unification amounts to normalizing a conjunction of equations, and since this normalization process commutes with resolution, these equations may be left in a normal form that is not a fully solved form. In particular, if an equation involves a functional expression whose arguments are not sufficiently instantiated to match a *definiens* of the function in question, it is simply left untouched. Resolution may proceed until the arguments are *proven* to match a definition from the accumulated constraints in the context [3]. This simple idea turns out invaluable in practice. Here are a few benefits.

- Such non-declarative heresies as the *is* predicate in Prolog and the *freeze* meta-predicate in some of its extensions [14, 10] 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 [9].
- More generally, it allows concurrent computation. Synchronization is obtained by checking entailment [13, 15].

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

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

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

1.2 The method

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

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

[1] Several patterns specifying a same function may possibly have overlapping denotations and therefore the order of the specified patterns define an implicit priority, as is usual in functional programming using first-order patterns (*e.g.*, [12]).

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

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

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.*, $\mathit{posint} \wedge \mathit{zero} = \perp, \mathit{negint} \wedge \mathit{zero} = \perp, \mathit{posint} \wedge \mathit{negint} = \perp$). This is declared in LIFE as $\mathit{int} := \{\mathit{posint}; \mathit{zero}; \mathit{negint}\}$. Furthermore, we assume the sort definition $\mathit{posint} := \{\mathit{posodd}; \mathit{poseven}\}$; *i.e.*, *posodd* and *poseven* are subsorts of *posint* and mutually incompatible.

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

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

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

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

What happens if both of these tests fail? Consider the query consisting of the conjunction:

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

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

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

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

This technique is relevant in the general framework of concurrent constraint logic programming, represented by, *e.g.*, the guarded Horn-clause scheme of Maher [13], Concurrent Constraint Programming (CCP) [15], and Kernel Andorra Prolog (KAP) [11]. These schemes are parameterized with respect to an abstract class of constraint systems. An incremental test for entailment and disentailment between constraints is needed for advanced control mechanisms such as delaying, coroutining, synchronization, committed choice, and deep constraint propagation. LIFE is formally an instance of this scheme, namely a CLP language using a constraint system based on order-sorted feature structures [6]. It employs a related, but limited, suspension strategy to enforce deterministic functional application. Roughly, these systems are concurrent thanks to a new effective discipline for procedure parameter-passing that we could describe as "call-by-constraint-entailment" (as opposed to Prolog's call-by-unification).

In this paper, we recall the basic terminology and notation of LIFE, unification and matching, and we sketch the essence of relative simplification. Formal material rewording everything in rigorous terms can be found in [4]. Nevertheless, in the final section we include the formal version of the simplification rules specifying the algorithms discussed informally in this paper, and state the theoretical results proven in [4].

2 LIFE Data Structures

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

2.1 Sorts and features

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

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

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

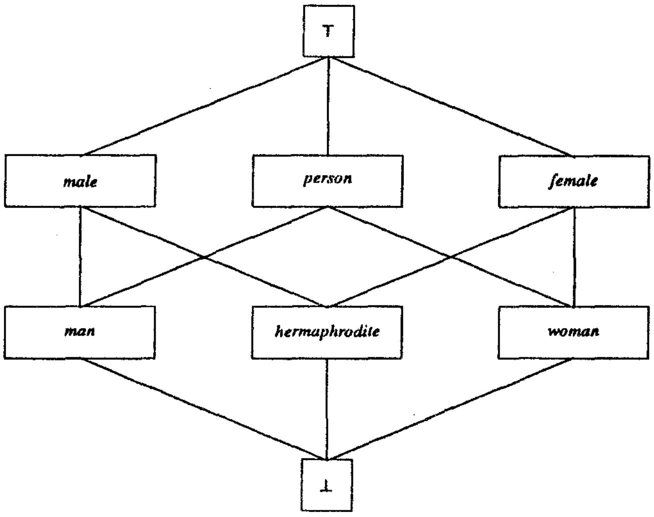

Fig. 1. A partial order of sorts

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

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

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

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

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

2.2 ψ-Terms

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

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

Stand-alone variables are always implicitly sorted by $\top$, and stand-alone sorts are always implicitly tagged by some variable occurring nowhere else. Thus, one might say that a basic ψ-term is always of the form *variable* : *sort*.

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

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

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

$$\begin{aligned} X : person(&age \Rightarrow I : int, \\ &spouse \Rightarrow Y : person(age \Rightarrow J : int)). \end{aligned}$$

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

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

$$\begin{aligned} X : person(&age \Rightarrow I : int, \\ &spouse \Rightarrow Y : person(age \Rightarrow I)). \end{aligned}$$

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

$$\begin{aligned} X : person(&age \Rightarrow I : int, \\ &spouse \Rightarrow Y : person(age \Rightarrow I, \\ &\qquad\qquad\qquad spouse \Rightarrow X)). \end{aligned}$$

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

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

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

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

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

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

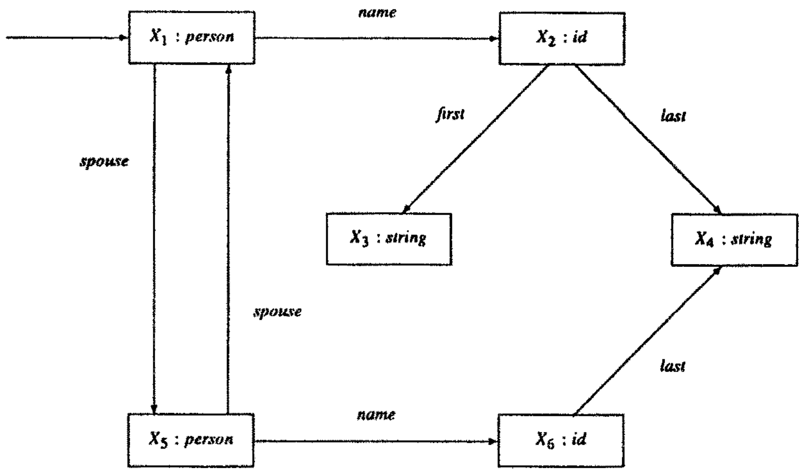

Fig. 2. An OSF-Graph

2.3 ψ-Terms as graphs

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

For example, the ψ-term:

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

corresponds to the OSF-graph shown in Figure 2.

2.4 ψ-Terms as values

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

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

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

2.5 Feature trees as values

We obtain two other examples of OSF-algebras when we "compress" the ψ-term domain by identifying values. In a first step, we say that two ψ-terms which are equal up to variable renaming represent the same value of the domain, or: two isomorphic graphs are identified. We call the OSF-algebra hereby obtained Ψ_0.

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

2.6 Unification of ψ-terms

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

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

2.7 Constraints and ψ-terms

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

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

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

is identified with the constraint:

$\psi \equiv X : person \ \& \ X.age \doteq Y \ \& \ Y : int \ \& \ X.likes \doteq X.$

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

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

2.8 Rules for Unification

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

Equality:

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

Sorts:

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

Features:

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

Clash:

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

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

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

3 Relative Simplification

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

3.1 Matching and entailment

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

In the Concurrent Constraint Logic Programming framework, the matching problem generalizes to the entailment problem; namely, whether the actual constraint, also called context, entails the formal constraint, also called guard [13, 15].

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

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

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

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

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

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

3.2 Entailment of general constraints

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

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

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

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

is equivalent to:

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

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

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

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

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

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

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

3.3 Towards relative simplification

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

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

Equality:

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

Sorts:

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

Features:

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

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

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

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

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

matches the ψ-term:

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

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

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

is normalized into:

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

that is,

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

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

3.4 Relative simplification for entailment

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

Equality:

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

Sorts:

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

Features:

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

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

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

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

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

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

3.5 Relative simplification for disentailment

If the result of the matching test $\psi_1 \leq \psi_2$ is negative, *i.e.*, the actual constraint does not entail the formal constraint, then we must know more; namely, whether the two terms are non-unifiable. Non-unifiability is equivalent to the fact that the actual parameter will not match the formal one even when further instantiated; *e.g.*, when further constraints are attached as conjuncts. Logically, this amounts to saying that a context formula ϕ *disentails* a guard constraint ψ if and only if the conjunction $\phi \ \&\ \psi$ is unsatisfiable. In terms of relative simplification, ϕ disentails ψ if and only if ψ simplifies to the *false* constraint $\perp$ relatively to ϕ.

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

Sorts:

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

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

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

[5] Refer to the sorts of Figure 1.

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

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

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

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

The ψ-term:[6]

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

is non-unifiable with the ψ-term:

$$\begin{aligned} \psi \;\equiv\; W : \top(&likes \Rightarrow U, \\ &friend \Rightarrow U) \end{aligned}$$

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

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

Then, using the rule:

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

where J is a new variable, to:

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

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

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

The ψ-term:

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

[6] We assume that $poseven \wedge posodd = \bot$.

is non-unifiable with the ψ-term:

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

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

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

Then, using the rule:

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

where J is a new variable, to:

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

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

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

3.6 Specifying the relative simplification algorithm

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

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

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

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

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

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

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

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

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

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

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

Feature Decomposition:

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

Sort Intersection:

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

Variable Elimination:

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

Inconsistent Sort:

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

Variable Clean-up:

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

Fig. 3. Basic Simplification

4 Simplification Rules

We give now the formal version of the simplification rules specifying the algorithms discussed informally in this paper, and state the theoretical results proven in [4]. Here, OSF-constraints are conjunctions of atomic sort constraints of the form $X : s$, atomic feature constraints of the form $X.\ell \doteq Y$, and equations $X \doteq Y$. Their interpretations over OSF-algebras are straightforward (*cf.*, Section 2.7).

4.1 Unification–Satisfiability

The first algorithm, called basic simplification, determines whether a constraint ϕ is consistent; *i.e.*, if it is satisfiable in some OSF-algebra $\mathcal{A}$—and, therefore, in particular in Ψ. Unification of two ψ-terms reduces to this problem.

Given an OSF-constraint ϕ, it can be normalized by choosing non-deterministically and applying any applicable rule among the transformations rules shown in Figure 3 until none applies. A rule transforms the numerator into the denominator. The expression $\phi[X/Y]$ stands for the formula obtained from ϕ after replacing all occurrences of Y by X.

Feature Decomposition:

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

Relative Feature Decomposition:

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

Relative Feature Equality:

(F.3) $$\frac{\psi \ \& \ U \doteq X_1 \ \& \ U \doteq X_2 \ \& \ V \doteq Y_1}{\psi \ \& \ U \doteq X_1 \ \& \ U \doteq X_2 \ \& \ V \doteq Y_1 \ \& \ V \doteq Y_2}$$ *if* $X_1.\ell \doteq Y_1 \in \phi, X_2.\ell \doteq Y_2 \in \phi$ *and* $V \doteq Y_2 \notin \psi$

Variable Introduction:

(F.4) $$\frac{\psi \ \& \ U \doteq X_1 \ \& \ U \doteq X_2}{\psi \ \& \ U \doteq X_1 \ \& \ U \doteq X_2 \ \& \ V \doteq Y_1 \ \& \ V \doteq Y_2}$$ *if* $X_1.\ell \doteq Y_1 \in \phi, X_2.\ell \doteq Y_2 \in \phi$ *and* $Y_1 \notin Var(\psi)$ *and* $Y_2 \notin Var(\psi)$ *where* V *is a new variable*

Fig. 4. Simplification relatively to ϕ : Features

The rules of Figure 3 are solution-preserving (*i.e.*, equivalence transformations), finite terminating, and confluent (modulo variable renaming).

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

Effectuality of basic-simplification *The constraint* ψ *is satisfiable if and only if the normal form* ψ' *of* ψ *is different from the false constraint, i.e.,* $\psi' \neq \bot$.

4.2 Matching–Entailment/Disentailment

The next algorithm, called relative simplification, determines whether a consistent constraint ϕ entails or disentails a constraint ψ which is the dissolved form of some ψ-term with root variable U. That is, it decides two problems simultaneously:

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

Matching of two ψ-terms reduces to the first of the two problems, non-unifiability to the second.

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

Relative Variable Elimination:

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

Equation Entailment:

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

Fig. 5. Simplification relatively to ϕ : Equations

The relative-simplification system preserves an important invariant property: the conjunction $\phi \;\&\; \psi$ is equivalent to the conjunction $\phi \;\&\; \psi'$. Again, the rules are finite terminating, and confluent (modulo variable renaming).

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

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

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

5 Conclusion

We have presented informally the essence of LIFE that is relevant to functions and explained the gist of our method for deciding incrementally entailment and disentailment of constraints. We call this new technique relative simplification. Reading this paper will provide an elaborate intuition of the technical details reported formally in [4]. There, we expound the concept of relative simplification as a general proof-theoretic method for proving guards in concurrent constraint logic languages using guarded rules. The specific relative simplification rules for OSF-constraints are given and proven correct and complete. Then, residuation is naturally explained using relative simplification. Finally, the operational semantics of function reduction is shown to be congruent with the semantics of ψ-terms as approximation structures.

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) $$\frac{\psi \;\&\; U : \perp}{\perp}$$

Fig. 6. Simplification relatively to ϕ : Sorts

References

1. Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351, 1986.
2. Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.
3. Hassan Aït-Kaci and Roger Nasr. Integrating logic and functional programming. *Lisp and Symbolic Computation*, 2:51–89, 1989.
4. Hassan Aït-Kaci and Andreas Podelski. Functions as passive constraints in LIFE. PRL Research Report 13, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France, June 1991. Revised, Novembre 1992.
5. Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. In Jan Maluszyński and Martin Wirsing, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming (Passau, Germany)*, pages 255–274. Springer-Verlag, LNCS 528, August 1991.
6. Hassan Aït-Kaci and Andreas Podelski. Towards a meaning of LIFE. PRL Research Report 11, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France, 1991. (Revised, October 1992; to appear in the Journal of Logic Programming).
7. Hassan Aït-Kaci, Andreas Podelski, and Gert Smolka. A feature-based constraint system for logic programming with entailment. In *Proceedings of the 5th International Conference on Fifth Generation Computer Systems*, pages 1012–1022, Tokyo, Japan, June 1992. ICOT.
8. Rolf Backofen and Gert Smolka. A complete and decidable feature theory. DFKI Research Report RR-30-92, German Research Center for Artificial Intelligence, Saarbrücken, Germany, 1992.
9. Staffan Bonnier and Jan Maluszyński. Towards a clean amalgamation of logic programs with external procedures. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium*, pages 311–326, Cambridge, MA, 1988. MIT Press.
10. Alain Colmerauer. Prolog II: Manuel de référence et modèle théorique. Rapport technique, Université de Marseille, Groupe d'Intelligence Artificielle, Faculté des Sciences de Luminy, Marseille, France, March 1982.
11. Seif Haridi and Sverker Janson. Kernel Andorra Prolog and its computation model. In David H. D. Warren and Peter Szeredi, editors, *Logic Programming, Proceedings of the 7th International Conference*, pages 31–46, Cambridge, MA, 1990. MIT Press.
12. Robert Harper, Robin Milner, and Mads Tofte. The definition of standard ML – Version 2. Report LFCS-88-62, University of Edinburgh, Edinburgh, UK, 1988.
13. Michael Maher. Logic semantics for a class of committed-choice programs. In Jean-Louis Lassez, editor, *Logic Programming, Proceedings of the Fourth International Conference*, pages 858–876, Cambridge, MA, 1987. MIT Press.
14. Lee Naish. *MU-Prolog 3.1db Reference Manual.* Computer Science Department, University of Melbourne, Melbourne, Australia, May 1984.
15. Vijay Saraswat. Concurrent constraint programming. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 232–245. ACM, January 1990.