TOWARDS A MEANING OF LIFE[†]

HASSAN AÏT-KACI AND ANDREAS PODELSKI

▷ LIFE is an experimental programming language proposing to integrate three orthogonal programming paradigms proven useful for symbolic computation. From the programmer's standpoint, it may be perceived as a language taking after logic programming, functional programming, and object-oriented programming. From a formal perspective, it may be seen as an instance (or rather, a composition of three instances) of a Constraint Logic Programming scheme due to Höhfeld and Smolka refining that of Jaffar and Lassez.

We start with an informal overview demonstrating LIFE as a programming language, illustrating how its primitives offer rather unusual, and perhaps (pleasantly) startling, conveniences. The second part is a formal account of LIFE's object unification seen as constraint-solving over specific domains. We build on work by Smolka and Rounds to develop type-theoretic, logical, and algebraic renditions of a calculus of order-sorted feature approximations. ◁

... the most succinct and poetic definition: *'Créer, c'est unir'* ('To create is to unify'). This is a principle that must have been at work from the very beginning of life.

KONRAD LORENZ, *Die Rückseite des Spiegels*

1. INTRODUCTION

As an acronym, 'LIFE' means Logic, Inheritance, Functions, and Equations. LIFE also designates an experimental programming language designed after these four precepts for specifying structures and computations. As for what LIFE means as a

[†]This article is a revision of [9]. A short form of [9] was also published as [10].

Address correspondence to Dr. H. Aït-Kaci, Digital Equipment Corporation, Paris Research Laboratory, 85 Avenue Victor Hugo, 92563 Rueil-Malmaison Cedex, France. Email: {hak, podelski}@prl.dec.com.

Received December 1991; accepted March 1993.

programming language, it is the purpose of this document to initiate the presentation of a complete formal semantics for LIFE. We shall proceed by characterizing LIFE as a specific instantiation of a Constraint Logic Programming (CLP) scheme with a particular constraint language. In its most primitive form, this constraint language constitutes a logic of record structures that we shall call Order-Sorted Feature logic—or, more concisely, OSF logic.

In this document, we mean to do two things: first, we overview informally the functionality of LIFE and the conveniences that it offers for programming; then, we develop the elementary formal foundations of OSF logic. We shall call this *basic* OSF logic. Although, in the basic form that we give here, the OSF formalism does not account for all overviewed aspects of LIFE (*e.g.*, functional reduction, constrained sort signature), it constitutes the kernel to be extended when we address those more elaborate issues later elsewhere. Showing how basic OSF logic fits as an argument constraint language of a CLP scheme is therefore a useful and necessary exercise. The CLP scheme that we shall use has been proposed by Höhfeld and Smolka [15] and is a generalization of that due to Jaffar and Lassez [16].

We shall define a class of interpretations of approximation structures adequate to represent basic LIFE objects. We call these OSF interpretations. As for syntax, we shall describe three variant (first-order) formalisms: (1) a type-theoretic term language, (2) an algebraic language, and, (3) a logical (clausal) language. All three will admit semantics over OSF interpretations structures. We shall make rigorously explicit the mutual syntactic and semantic equivalence of the three representations. This allows us to shed some light on, and reconcile, three common interpretations of multiple inheritance as, respectively, (1) set inclusion; as (2) algebraic endomorphism; and, (3) as logical implication.

Our approach centers around the notion of an OSF-algebra. This notion was already used implicitly in [1, 2] to give a semantics to ψ-terms. Gert Smolka's work on Feature Logic [18, 19] made the formalism emerge more explicitly, especially in the form of a "canonical *OSF*-graph algebra," and was used by Dörre and Rounds in recent work showing undecidability of semiunification of cyclic structures [14].[1]

This document is organized as follows. We first give an informal tour of some of LIFE's unusual programming conveniences. We hope by this to illustrate for the reader that some original functionality is available to a LIFE user. We do this by way of small yet (pleasantly) startling examples. Following that, in Section 3, we proceed with the formal account of basic OSF logic. There, OSF interpretations are introduced together with syntactic forms of terms, clauses, and graphs taking their meaning in those interpretations. It is then made explicit how these various forms are related through mutual syntactic and semantic correspondences. In Section 3.4, we show how to tie basic OSF logic into a CLP scheme. (For the sake of making this work self-contained, we briefly summarize, in Appendix A, the essence of the general Constraint Logic Programming scheme that we use

[1] Dörre and Rounds do not consider order-sorted graphs and focus only on features, whereas Smolka considers both the order-sorted and the unsorted case. However, Smolka does not make explicit the mutual syntactic and semantic mappings between the algebraic, logical, and type-theoretic views. On the other hand, the logics considered in [18, 19] are richer than the basic formalism to which we limit ourselves here, allowing explicit negation and quantification. Naturally, all these extensions can as well be considered in our framework.

explicitly. It is due to Höhfeld and Smolka [15].) Finally, we conclude anticipating on the necessary extensions of basic OSF logic to achieve a full meaning of LIFE.

2. LIFE, Informally

LIFE is a trinity. The function-oriented component of LIFE is directly derived from functional programming languages with higher-order functions as first-class objects, data constructors, and algebraic pattern-matching for parameter-passing. The convenience offered by this style of programming is one in which expressions of any order are first-class objects and computation is determinate. The relation-oriented component of LIFE is essentially one inspired by the Prolog language [13, 17]. Unification of first-order patterns used as the argument-passing operation turns out to be the key of a quite unique and hitherto unusual *generative* behavior of programs, which can construct missing information as needed to accommodate success. Finally, the most original part of LIFE is the structure-oriented component which consists of a calculus of type structures—the ψ-calculus [1, 2]—and accounts for some of the (multiple) inheritance convenience typically found in so-called object-oriented languages.

Under these considerations, a natural coming to LIFE has consisted in first studying pairwise combinations of each of these three operational tools. Metaphorically, this means realizing edges of a triangle (see Figure 1) where each vertex is some essential operational rendition of the appropriate calculus. LOGIN is simply Prolog where first-order constructor terms have been replaced by ψ-terms, with type definitions [5]. Its operational semantics is the immediate adaptation of that of Prolog's SLD resolution. Le Fun [6, 7] is Prolog where unification may reduce functional expressions into constructor form according to functions defined by pattern-oriented functional specifications. Finally, FOOL is simply a pattern-oriented functional language where first-order constructor terms have been replaced by ψ-terms, with type definitions. LIFE is the composition of the three with the additional capability of specifying arbitrary functional and relational constraints on objects being defined. The next subsection gives a very brief and informal account of the calculus of type inheritance used in LIFE (ψ-calculus). The reader is assumed familiar with functional programming and logic programming.

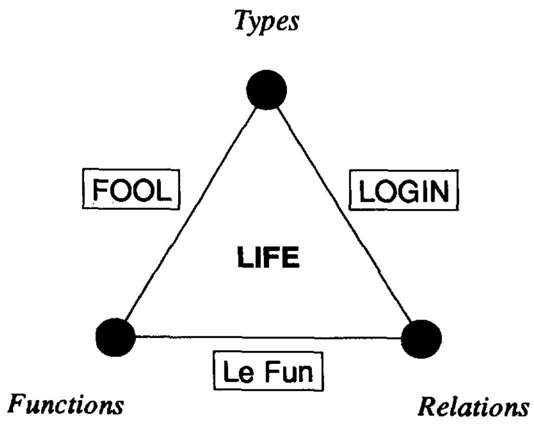

FIGURE 1. The LIFE molecule.

2.1. ψ-Calculus

In this section, we give an informal but informative introduction of the notation, operations, and terminology of the data structures of LIFE. It is necessary to understand the programming examples to follow.

The ψ-calculus consists of a syntax of structured types called ψ-terms together with subtyping and type intersection operations. Intuitively, as expounded in [5], the ψ-calculus is a convenience for representing record-like data structures in logic and functional programming more adequately than first-order terms do, without loss of the well-appreciated instantiation ordering and unification operation.

Let us take an example to illustrate. Let us say that one has in mind to express syntactically a type structure for a *person* with the property, as expressed for the underlined symbol in Figure 2, that a certain functional diagram commutes.

The syntax of ψ-terms is one simply tailored to express as a term this kind of approximate description. Thus, in the ψ-calculus, the information of Figure 2 is unambiguously encoded into a formula, perspicuously expressed as the ψ-term:

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

It is important to distinguish among the three kinds of symbols participating in a ψ-term. We assume given a set $\mathcal{S}$ of sorts or *type constructor symbols*, a set $\mathcal{F}$ of

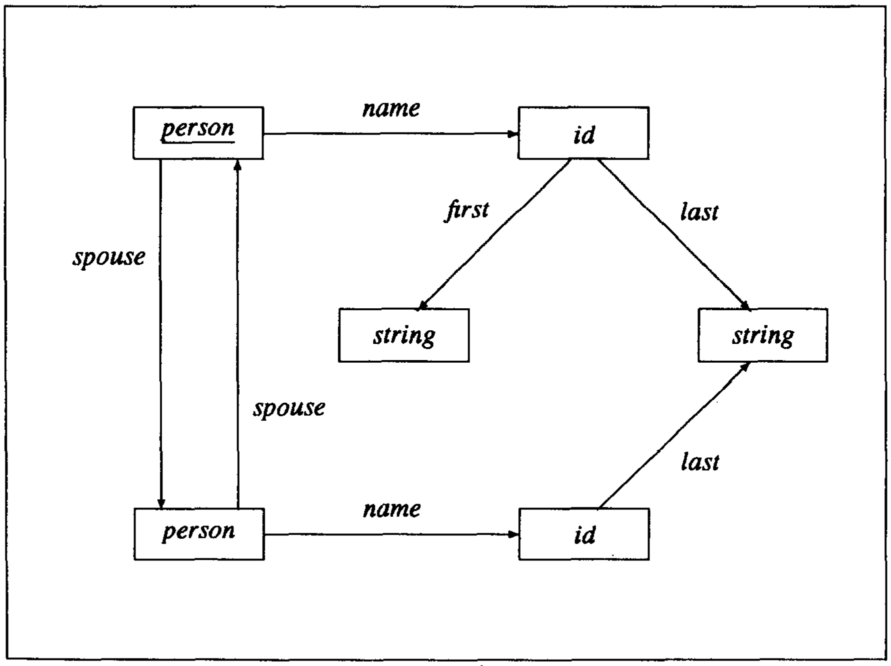

FIGURE 2. A commutative functional diagram.

features, or attributes symbols, and a set V of *variables* (or *coreference tags*). In the ψ-term above, for example, the symbols *person*, *id*, *string* are drawn from $\mathcal{S}$, the symbols *name*, *first*, *last*, *spouse* from $\mathcal{F}$, and the symbols X, S from V. (We capitalize variables, as in Prolog.)

A ψ-term is either *tagged* or *untagged*. A tagged ψ-term is either a variable in V or an expression of the form $X : t$ where $X \in V$ is called the term's *root variable* and t is an untagged ψ-term. An untagged ψ-term is either *atomic* or *attributed*. An atomic ψ-term is a sort symbol in $\mathcal{S}$. An attributed ψ-term is an expression of the form $s(\ell_1 \Rightarrow t_1, \ldots, \ell_n \Rightarrow t_n)$ where the root variable's sort symbol $s \in \mathcal{S}$ and is called the ψ-term's *principal* type, the ℓ_i's are mutually distinct attribute symbols in $\mathcal{F}$, and the t_i's are ψ-terms ($n \geq 0$).

Variables capture coreference in a precise sense. They are coreference tags and may be viewed as typed variables where the type expressions are untagged ψ-terms. Hence, as a condition to be *well-formed*, a ψ-term must have all occurrences of each coreference tag consistently refer to the same structure. For example, the variable X in

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

refers consistently to the atomic ψ-term *string*. To simplify matters and avoid redundancy, we shall obey a simple convention of specifying the sort of a variable at most once and understand that other occurrences are equally referring to the same structure, as in:

$$\begin{aligned}&\mathit{person}(\mathit{id} \Rightarrow \mathit{name}(\mathit{first} \Rightarrow \mathit{string},\\ &\qquad\qquad\qquad\qquad \mathit{last} \Rightarrow X : \mathit{string}),\\ &\qquad\quad \mathit{father} \Rightarrow \mathit{person}(\mathit{id} \Rightarrow \mathit{name}(\mathit{last} \Rightarrow X))).\end{aligned}$$

In fact, since there may be circular references as in $X : \mathit{person}(\mathit{spouse} \Rightarrow \mathit{person}(\mathit{spouse} \Rightarrow X))$, this convention is necessary. Finally, a variable appearing nowhere typed, as in $\mathit{junk}(\mathit{kind} \Rightarrow X)$ is implicitly typed by a special greatest initial sort symbol $\top$ always present in $\mathcal{S}$. This symbol will be left invisible and not written explicitly as in $(\mathit{age} \Rightarrow \mathit{integer}, \mathit{name} \Rightarrow \mathit{string})$, or written as the symbol @ as in $@(\mathit{age} \Rightarrow \mathit{integer}, \mathit{name} \Rightarrow \mathit{string})$. In the sequel, by ψ-term we shall always mean well-formed ψ-term and call such a form a (ψ)-*normal form*.

Generalizing first-order terms,[2] ψ-terms are ordered up to variable renaming. Given that the set $\mathcal{S}$ is partially-ordered (with a greatest element $\top$), its partial ordering is extended to the set of attributed ψ-terms. Informally, a ψ-term t_1 is subsumed by a ψ-term t_2 if (1) the principal type of t_1 is a subtype in $\mathcal{S}$ of the principal type of t_2; (2) all attributes of t_2 are also attributes of t_1 with ψ-terms which subsume their homologues in t_1; and, (3) all coreference constraints binding in t_2 must also be binding in t_1.

[2] In fact, if a first-order term is written $f(t_1, \ldots, t_n)$, it is nothing other than syntactic sugar for the ψ-term $f(1 \Rightarrow t_1, \ldots, n \Rightarrow t_n)$.

For example, if *student* < *person* and *paris* < *cityname* in $\mathcal{S}$ then the ψ-term:

$$\begin{aligned}
&student(id \Rightarrow name(first \Rightarrow string,\\
&\qquad\qquad\qquad\quad last \Rightarrow X : string),\\
&\qquad lives\_at \Rightarrow Y : address(city \Rightarrow paris),\\
&\qquad father \Rightarrow person(id \Rightarrow name(last \Rightarrow X),\\
&\qquad\qquad\qquad\qquad lives\_at \Rightarrow Y))
\end{aligned}$$

is subsumed by the ψ-term:

$$\begin{aligned}
&person(id \Rightarrow name(last \Rightarrow X : string),\\
&\qquad lives\_at \Rightarrow address(city \Rightarrow cityname),\\
&\qquad father \Rightarrow person(id \Rightarrow name(last \Rightarrow X))).
\end{aligned}$$

In fact, if the set $\mathcal{S}$ is such that *greatest lower bounds* (GLB's) exist for any pair of type symbols, then the subsumption ordering on ψ-term is also such that GLB's exist. (See Appendix B for the case when GLB's are not unique.) Such are defined as the *unification* of two ψ-terms. A detailed unification algorithm for ψ-terms is given in [5].

Consider for example the poset displayed in Figure 3 and the two ψ-terms:

$$\begin{aligned}
&X : student(advisor \Rightarrow faculty(secretary \Rightarrow Y : staff,\\
&\qquad\qquad\qquad\qquad\qquad\qquad assistant \Rightarrow X),\\
&\qquad\qquad roommate \Rightarrow employee(representative \Rightarrow Y))
\end{aligned}$$

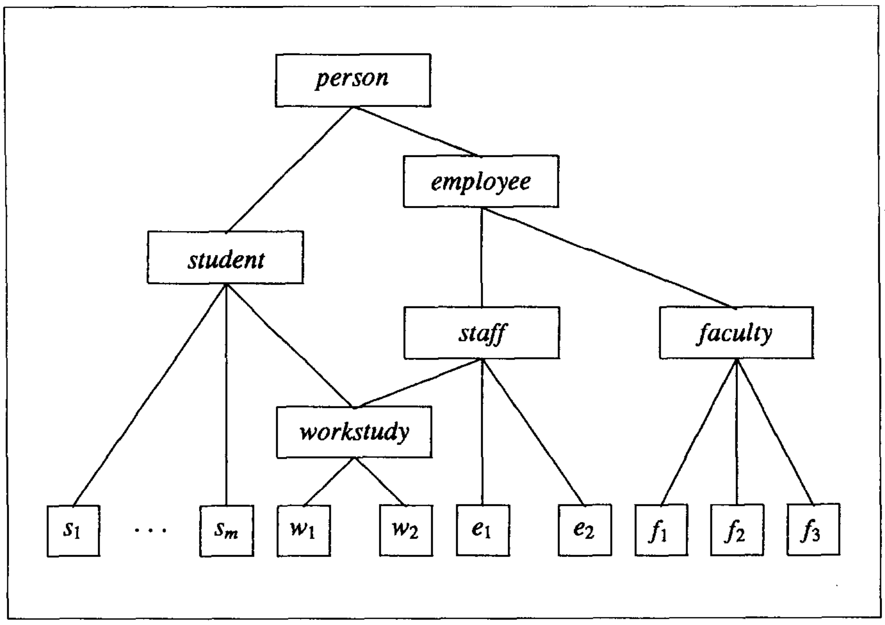

FIGURE 3. A lower semi-lattice of sorts.

and:

$$\begin{aligned}&\mathit{employee}(\mathit{advisor} \Rightarrow f_1(\mathit{secretary} \Rightarrow \mathit{employee},\\ &\qquad\qquad\qquad\qquad \mathit{assistant} \Rightarrow U : \mathit{person}),\\ &\qquad \mathit{roommate} \Rightarrow V : \mathit{student}(\mathit{representative} \Rightarrow V),\\ &\qquad \mathit{helper} \Rightarrow w_1(\mathit{spouse} \Rightarrow U)).\end{aligned}$$

Their unification (up to tag renaming) yields the term:

$$\begin{aligned}&W : \mathit{workstudy}(\mathit{advisor} \Rightarrow f_1(\mathit{secretary} \Rightarrow Z : \mathit{workstudy}(\mathit{representative} \Rightarrow Z),\\ &\qquad\qquad\qquad\qquad\qquad \mathit{assistant} \Rightarrow W),\\ &\qquad\quad \mathit{roommate} \Rightarrow Z,\\ &\qquad\quad \mathit{helper} \Rightarrow w_1(\mathit{spouse} \Rightarrow W)).\end{aligned}$$

Last in this brief introduction to the ψ-calculus, we explain type definitions. The concept is analogous to what a global store of constant definitions is in a practical functional programming language based on λ-calculus. The idea is that types in the signature may be specified to have attributes in addition to being partially-ordered. Inheritance of attributes from all supertypes to a subtype is done in accordance with ψ-term subsumption and unification. For example, given a simple signature for the specification of linear lists $\mathcal{S} = \{\mathit{list}, \mathit{cons}, \mathit{nil}\}$ with $\mathit{nil} < \mathit{list}$ and $\mathit{cons} < \mathit{list}$, it is yet possible to specify that *cons* has an attribute $\mathit{tail} \Rightarrow \mathit{list}$. We shall specify this as:

$$\mathit{list} := \{\mathit{nil}; \mathit{cons}(\mathit{tail} \Rightarrow \mathit{list})\}.$$

From which the appropriate partial-ordering is inferred.

As in this *list* example, such type definitions may be recursive. Then, ψ-unification *modulo* such a type specification proceeds by unfolding type symbols according to their definitions. This is done by need as no expansion of symbols need be done in case of (1) failures due to order-theoretic clashes (*e.g.*, $\mathit{cons}(\mathit{tail} \Rightarrow \mathit{list})$ unified with *nil* fails; *i.e.*, gives $\bot$); (2) symbol subsumption (*e.g.*, *cons* unified with *list* gives just *cons*), and (3) absence of attribute (*e.g.*, $\mathit{cons}(\mathit{tail} \Rightarrow \mathit{cons})$ unified with *cons* gives $\mathit{cons}(\mathit{tail} \Rightarrow \mathit{cons})$). Thus, attribute inheritance may be done "lazily," saving much unnecessary expansions [11].

In LIFE, a basic ψ-term denotes a functional application in FOOL's sense if its root symbol is a defined function. Thus, a *functional expression* is either a ψ-term or a conjunction of ψ-terms denoted by $t_1 : t_2 : \cdots : t_n$.[3] An example of such is $\mathit{append}(\mathit{list}, L) : \mathit{list}$, where *append* is the FOOL function defined as:

$$\begin{aligned}&\mathit{list} := \{[\,]; [@ \mid \mathit{list}]\}.\\ &\mathit{append}([\,], L : \mathit{list}) \rightarrow L.\\ &\mathit{append}([H \mid T : \mathit{list}], L : \mathit{list}) \rightarrow [H \mid \mathit{append}(T, L)].\end{aligned}$$

[3]In fact, we propose to see the notation -:- simply as a dyadic operation resulting in the GLB of its arguments since, for example, the notation $X : t_1 : t_2$ is shorthand for $X : t_1$, $X : t_2$. Where the variable X is not necessary, (*i.e.*, not otherwise shared in the context), we may thus simply write $t_1 : t_2$.

This is how functional dependency constraints are expressed in a ψ-term in LIFE. For example, in LIFE the ψ-term $foo(bar \Rightarrow X : list,\ baz \Rightarrow Y : list,\ fuz \Rightarrow append(X,Y) : list)$ is one in which the attribute *fuz* is derived as a list-valued function of the attributes *bar* and *baz*. Unifying such ψ-terms proceeds as before modulo suspension of functional expressions whose arguments are not sufficiently refined to be provably subsumed by patterns of function definitions.

As for relational constraints on objects in LIFE, a ψ-term t may be followed by a *such-that* clause consisting of the logical conjunction of (relational) literals $C_1, \ldots, C_n$, possibly containing functional terms. It is written as $t | C_1, \ldots, C_n$. Unification of such relationally constrained terms is done modulo proving the conjoined constraints. We will illustrate this very intriguing feature with two examples: `prime.life` (Section 2.5) and `quick.life` (Section 2.4). In effect, this allows specifying *daemonic constraints* to be attached to objects. Such a (renamed) "daemon-constrained" object's specified relational and (equational) functional formula is normalized by LIFE, its proof being triggered by unification at the object's creation time.

We give next some LIFE examples.

2.2. *Order-sorted logic programming:* `happy.life`

The first example illustrates a use of partially-ordered sorts in logic programming. The ψ-terms involved here are only atomic ψ-terms; *i.e.*, unattributed sort symbols. This example shows the advantage of summarizing the extent of a relation with predicate's arguments ranging over types rather than individuals.

Peter, Paul and Mary are students, and students are persons.

```
student := {peter;paul;mary}.
student <| person.
```

Grades are good grades or bad grades. A and B are good grades, while C, D and F are bad grades.

```
grade := {goodgrade;badgrade}.
goodgrade := {a;b}.
badgrade := {c;d;f}.
```

Goodgrades are good things.

```
goodgrade <| goodthing.
```

Every person likes herself. Every person likes every good thing. Peter likes Mary.

```
likes(X:person,X).
likes(person,goodthing).
likes(peter,mary).
```

Peter got a C, Paul an F and Mary an A.

```
got(peter,c).
got(paul,f).
got(mary,a).
```

A person is happy if s/he got something that s/he likes, or, if s/he likes something that got a good thing.

```
happy(X:person) :- got(X,Y),likes(X,Y).
happy(X:person) :- likes(X,Y),got(Y,goodthing).
```

To the query '`happy(X:student)?`' LIFE answers `X` = mary (twice—see why?), then gives `X=peter`, then fails. (It helps to draw the sort hierarchy order diagram.)

2.3. *Passive constraints:* `lefun.life`

The next three examples illustrate the interplay of unification and interpretable functions. The first two do not make any specific use of ψ-terms. Again, the first-order term notation is used as implicit syntax for ψ-terms with numerical features.

Consider first the following:

```
p(X, Y) :- q(X, Y, Z, Z), r(X, Y).
q(X, Y, X+Y, X*Y).
q(X, Y, X+Y, (X*Y)-14).
r(3, 5).
r(2, 2).
r(4, 6).
```

Upon a query '`p(X,Y)?`' the predicate `p` selects a pair of expressions in X and Y whose evaluations must unify, and then selects values for X and Y. The first solution selected by predicate `q` sets up the residual equation (or *residuation*, or *suspension*) that $X+Y=X*Y$ (more precisely that both $X+Y$ and $X*Y$ should unify with Z), which is not satisfied by the first pair of values, but is by the second. The second solution sets up $X+Y=(X*Y)-14$, which is satisfied by $X=4$, $Y=6$.

The next two examples show the use of higher-order functions such as `map`:

```
map(@, [ ]) → [ ].
map(F, [H|T]) → [F(H)|map(F,T)].
inc_list(N:int, L:list, map(+(N),L)).
```

To the query ‘`inc_list(3,[1,2,3,4],L)?`’,LIFE answers `L=[4,5,6,7]`. In passing, note the built-in constant @ as the primeval LIFE object (formally written ⊤) which approximates anything in the universe.

Note that it is possible, since LIFE uses ψ-terms as a universal object structure, to pass arguments to functions by keywords and obtain the power of partial application (currying) in all arguments, as opposed to λ-calculus which requires left-to-right currying [3]. For example of an (argument-selective) currying, consider the (admitted pathological) LIFE program:

```
curry(V) :- V=G(2⇒1), G=F(X), valid(F), pick(X), p(sq(V)).
sq(X) → X*X.
twice(F,X) → F(F(X)).
valid(twice).
p(1).
id(X) → X.
pick(id).
```

What does LIFE answer when ‘`curry(V)?`’ is the query? The relation `curry` is the property of a variable V when this variable is the result of applying a variable function G to the number 1 as its second argument. But G must also be the value of applying a variable function F to an unknown argument X. The predicate `valid` binds F to `twice`, and therefore binds V to `twice(X,1)`. Then, `pick` binds X to the identity function. Thus, the value of G, `twice(X)`, becomes `twice(id)` and V becomes now bound to 1, the value of `twice(id,1)`. Finally, it must be verified that the square of V unifies with a value satisfying property `p`.

2.4. *Functional programming with logical variables:* `quick.life`

This is a small LIFE module specifying (and thus, implementing) C.A.R. Hoare's “Quick Sort” algorithm functionally. This version works on lists which are not terminated with `[ ]` (nil) but with uninstantiated variables (or partially instantiated to a non-minimal list sort). Therefore, LIFE makes difference-lists *bona fide* data structures in functional programming.

```
q_sort(L,order⇒O) → undlist(dqsort(L,order⇒O)).

undlist(X\Y) → X|Y=[ ].

dqsort([ ]) → L\L.
dqsort([H|T],order⇒O)
    → (L1\L2) : where
                ((Less,More) : split(H,T,([ ],[ ]),order⇒O),
                 (L1\[H|L3]) : dqsort(Less,order⇒O),
                 (L3\L2)     : dqsort(More,order⇒O)).

where → @.
```

```
split(@,[ ],P) → P.
split(X,[H|T],(Less,More),order⇒O) →
        cond(O(H,X),
             split(X,T,([H|Less],More),order⇒O),
             split(X,T,(Less,[H|More]),order⇒O)).
```

The function dqsort takes a regular list (and parameterized comparison boolean function ∘) into a difference-list form of its sorted version (using Quick Sort). The function undlist yields a regular form for a difference-list. Finally, notice the definition and use of the (functional) constant where which returns the most permissive approximation (@). It simply evaluates its arguments (*a priori* unconstrained in number and sorts) and throws them away. Here, it is applied to three arguments at (implicit) positions (attributes) 1 (a pair of lists), 2 (a difference-list), and 3 (a difference-list). Unification takes care of binding the local variables Less,More,L1,L2,L3, and exporting those needed for the result (L1,L2). The advantage (besides perspicuity and elegance) is performance: replacing where with @ inside the definition of dqsort is correct but keeps around three no-longer needed argument structures at each recursive call.

Here are some specific instantiations:

```
number_sort(L:list) → q_sort(L, order ⇒ < ).
string_sort(L:list) → q_sort(L, order ⇒ $ < ).
```

such that to the query:

```
L=string_sort(["is","This","sorted","lexicographically"])?
```

LIFE answers:

```
L=["This","is","lexicographically","sorted"].
```

2.5. *High-school math specifications:* prime.life

This example illustrates sort definitions using other sorts and constraints on their structure. A prime number is a positive integer whose number of proper factors is exactly one. This can be expressed in LIFE as:

```
posint:=I:int|I>0=true.
prime:=P:posint|number_of_factors(P)=one.
```

where:

```
number_of_factors(N:posint)
   → cond(N⇐1, { }, factors_from(N,2)).
factors_from(N:int,P:int)
```

```
    → cond(P*P>N,
           one,
           cond(R:(N/P)=:=floor(R),
                many,
                factors_from(N,P+1))).
posint_stream→ {1;1+posint_stream}.
list_all_primes:- write(posint_stream:prime), nl, fail.
```

As for @, the dual built-in constant { } is the final LIFE object (formally written ⊥) and is approximated by anything in the universe. Operationally, it just causes failure equivalent to that due to an inconsistent formula. Any object that is not a non-strict functional expression (such as cond) in which { } occurs will lead to failure (⊥ as an object or the inconsistent clause as a formula). Also, LIFE's functions may contain infinitely disjunctive objects such as streams. For instance, posint_stream is such an object (a 0-ary function constant) whose infinitely many disjuncts are the positive integers enumerated from 1. Or, if a limited stream is preferred:

```
posint_stream_up_to(N:int)
    → cond(N<1,
           { },
           {1;1+posint_stream_up_to(N-1)}).
list_primes_up_to(N:int)
    :- write(posint_stream_up_to(N):prime), nl, fail.
```

This last example concludes our informal overview of some of the most salient features of LIFE. Next, with a slight change of speed, we shall undertake casting its most basic components into an adequate formal frame.

3. Formal LIFE

This section makes up the second part of this paper and sets up formal foundations upon which to build a full semantics of LIFE. The gist of what follows is the construction of a logical constraint language for LIFE type structures with the appropriate semantic structures. In the end of this section, we will use this constraint language to instantiate the Höhfeld-Smolka CLP scheme (see Appendix Section A for a summary of the scheme). We hereby give a complete account essentially of that part of LIFE which makes up LOGIN [5] without type definitions. Elsewhere, using the same semantic framework, we account for type definitions [11] and for functions as passive constraints [8].

Thus, the point of this section is to elucidate how the core constraint system of LIFE (namely, ψ-terms with unification) is an instance of CLP. The main difficulty faced here is the absence of element-denoting terms since ψ-terms denote sets of values. It is still possible, however, to compute "answer substitutions," and we will make explicit their formal meaning. A concrete representation of ψ-terms is given in term of order-sorted feature (*OSF*) graphs. One main insight is that *OSF*-graphs make a canonical interpretation. In addition, they enjoy a nice "schizophrenic" property; *OSF*-graphs denote *both elements* of the domain of interpretation *and sets* of values. Indeed, an *OSF*-graph may be seen as the generator of a principal filter for an approximation ordering (namely, of the set of all graphs it approximates). What we also exhibit is that a most general solution as a variable valuation is immediately extracted from an *OSF*-graph. All other solutions are endomorphic refinements (*i.e.*, instantiations) of this most general one, generating all and only the elements of the set denotation of this *OSF*-graph.

Lest the reader, faring through this dense and formal section, feel a sense of loss and fail to see the forest from the trees, here is a road map of its contents. Section 3.1 introduces the semantic structures needed to interpret the data structures of LIFE. Then, Section 3.2 describes three alternative syntactic presentations of these data structures: Section 3.2.1 defines a term syntax, Section 3.2.2 defines a clausal syntax, and Section 3.2.3 defines a graph syntax. In each case, a semantics is given in terms of the algebraic structures introduced in Section 3.1. The three views are important since the term view is the abstract syntax used by the user; the clausal view is the syntax used in the normalization rules presenting the operational semantics of constraint-solving; and, the graph view is the canonical representation used for implementation. Then, all these syntaxes are formally related thanks to explicit correspondences. Following that, Section 3.3 shows that each syntax is endowed with a natural ordering. The terms are ordered by set-inclusion of their denotations; the clauses by implications; and, the graphs by endomorphic approximation. It is then established in a semantic transparency theorem that these orderings are semantically preserved by the syntactic correspondences. The last part, Section 3.4, integrates the previous constructions into a relational language of definite clauses and ties everything together as an explicit instance of the Höhfeld-Smolka CLP scheme. Section 3.4.1 deals with definite clauses and queries over *OSF*-terms, Section 3.4.2 deals with definite clauses of *OSF*-constraints; and, Section 3.4.3 deals with *OSF*-graphs computed by a LIFE program.

3.1. The Interpretations: OSF-algebras

The formulae of basic OSF logic are *type formulae* which restrict variables to range over sets of objects of the domain of some interpretation. Roughly, such types will be used as approximations of elements of the interpretation domains when we may have only partial information about the element or the domain. In other words, specifying an object to be of such a type does in no way imply that this object can be singled out in every interpretation. Furthermore, it will not be necessary to consider a single fixed interpretation domain, reflecting situations when the domain of discourse can not be specified completely, as is often the case in knowledge representation. Instead, it can be sufficient to specify a *class* of admissible interpretations. This is done by means of a *signature*. We shall consider domains which are coherently described by classifying symbols (*i.e.*, partially-ordered sorts) and whose elements may be functionally related with one another through features (*i.e.*, labels or attributes). Thus, our specific signatures will comprise the symbols for sorts and features and regulate their intended interpretation.

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

$\wedge, \mathcal{F}\rangle$ such that:

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

A signature as above has the following interpretation. An *order-sorted feature algebra* (or simply *OSF*-algebra) over the signature $\langle \mathcal{S}, \leq, \wedge, \mathcal{F}\rangle$ is a structure

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

such that:

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

Thanks to our interpretation of features as functions on the domain, a natural monoid homomorphism extends this between the free monoid $\langle \mathcal{F}^*, \cdot, \varepsilon\rangle$ and the endofunctions of $D^{\mathcal{A}}$ with composition, $\langle (D^{\mathcal{A}})^{(D^{\mathcal{A}})}, \circ, Id_{D^{\mathcal{A}}}\rangle$. We shall refer to elements of either of these monoids as attribute (or feature) compositions.

In the remainder of this paper, we shall implicitly refer to some fixed signature $\langle \mathcal{S}, \leq, \wedge, \mathcal{F}\rangle$.

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

Definition 1 (*OSF*-**Homomorphism**). *An OSF-algebra homomorphism* $\gamma : \mathcal{A} \mapsto \mathcal{B}$ *between two OSF-algebras* $\mathcal{A}$ *and* $\mathcal{B}$ *is a function* $\gamma : D^{\mathcal{A}} \mapsto D^{\mathcal{B}}$ *such that:*

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

It comes as a straightforward consequence that *OSF*-algebras together with *OSF*-homomorphisms form a category. We call this category **OSF**.

Let D be a non-empty set and $(l^D \in D^D)_{\ell\in\mathcal{F}}$ an $\mathcal{F}$-indexed family of total endofunctions of D. To any feature composition $\omega = \ell_1, \ldots, \ell_n$, $n \geq 0$ in the free monoid $\mathcal{F}^*$, there corresponds a function composition $\omega^D = \ell_n^D \circ \cdots \circ \ell_1^D$ in D^D (for $n = 0$, $\varepsilon^D = 1_D$). Then, for any non-empty subset S of D, we can construct the $\mathcal{F}$-*closure* of S, the set $\mathcal{F}^*(S) = \bigcup_{\omega\in\mathcal{F}^*} \omega^D(S)$. This is the smallest set containing S which is closed under feature application. Using this, the familiar notion of least algebra generated by a set can naturally be given for *OSF*-algebras as follows.

Proposition 1 (**Least subalgebra generated by a set**). *Let D be the domain of an*

OSF-algebra $\mathscr{A}$, then for any non-empty subset S of D, the $\mathscr{F}$-closure of S is the domain of $\mathscr{A}[S]$, the least OSF-algebra subalgebra of $\mathscr{A}$ containing S; i.e., $D^{\mathscr{A}[S]} = \mathscr{F}^*(S)$.

Proof. $\mathscr{F}^*(S)$ is closed under feature application by construction. As for sorts, simply take $s^{\mathscr{A}[S]} = s^D \cap \mathscr{F}^*(S)$. It is straightforward to verify that this forms a subalgebra which is the smallest containing S. □

3.2. *The syntax*

3.2.1. OSF-terms. We now introduce the syntactic objects that we intend to use as type formulae to be interpreted as subsets of the domain of an *OSF*-algebra. Let $\mathscr{V}$ be a countably infinite set of variables.

Definition 2 (*OSF*-Term). *An order-sorted feature term (or, OSF-term) ψ is an expression of the form*

$$\psi = X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n) \tag{1}$$

where X is a variable in $\mathscr{V}$, s is a sort in $\mathscr{S}$, $\ell_1, \ldots, \ell_n$ are features in $\mathscr{F}$, $n \geq 0$, and $\psi_1, \ldots, \psi_n$ are OSF-terms.

Note that the equation above includes $n = 0$ as a base case. That is, the simplest *OSF*-terms are of the form $X : s$. We call the variable X in the above *OSF*-term the *root* of ψ (noted $Root(\psi)$), and say that X is "sorted" by the sort s and "has attributes" $\ell_1, \ldots, \ell_n$. The set of variables occurring in ψ is given by $Var(\psi) = \{X\} \cup \bigcup_{j \leq n} Var(\psi_j)$.

Example 1. The following is an example of the syntax of an *OSF*-term:

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

Note that, in general, an *OSF*-term may have redundant attributes (e.g., *name*) or the same variable sorted by different sorts (e.g., X and S).

Intuitively, such an *OSF*-term as given by Equation 1 is a syntactic expression intended to denote sets of elements in some appropriate domain of interpretation under all possible valuations of its variables in this domain. Now, what is expressed by an *OSF*-term is that, for a given fixed valuation of the variables in such a domain, the element assigned to the root variable must lie within the set denoted by its sort. In addition, the function that denotes an attribute must take it into the denotation of the corresponding subterm, under the same valuation. The same scheme then applies recursively for the subterms. Clearly, an *OSF*-algebra forms an adequate structure to capture this precisely as shown next.

Given the interpretation $\mathscr{A}$, the *denotation* $[\![\psi]\!]^{\mathscr{A},\alpha}$ of an *OSF*-term ψ of the form given by equation 1, *under a valuation* $\alpha : \mathscr{V} \mapsto D^{\mathscr{A}}$ is given inductively by:

$$[\![\psi]\!]^{\mathscr{A},\alpha} = \{\alpha(X)\} \cap s^{\mathscr{A}} \cap \bigcap_{1 \le i \le n} (\ell_i^{\mathscr{A}})^{-1}([\![\psi_i]\!]^{\mathscr{A},\alpha}) \tag{2}$$

where an expression such as $f^{-1}(S)$, when f is a function and S is a set, stands for $\{x | \exists y\ y = f(x)\}$; i.e., denotes the set of all elements whose images by f are in S.

Without further context with which variable names may be shared, we shall usually use a lightened notation for *OSF*-terms whereby any variable occurring without a sort is implicitly sorted with $\top$ and all variables which do not occur more than once are not given explicitly. This is justified in some manner by our *OSF*-term semantics is the sense that the *OSF*-term recovered from the lightened notation, by introducing a new distinct variable anywhere one is missing and introducing the sort $\top$ anywhere a sort is missing, denotes precisely the same set, irrespective of the name of single occurrence variables.

Example 2. Using this light notation, the *OSF*-term of Example 1 becomes:

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

Observe that Equation 2 reflects the meaning of an *OSF*-term for only one valuation and therefore always specifies a singleton or possibly the empty set. Also, note that this definition does include the base case (*i.e.*, $n = 0$), owing to the fact that intersection over the empty set is the universe ($\bigcap\{\ldots | 1 \le i \le n\} = \bigcap\emptyset = D^{\mathscr{A}}$).

Since we are interested in all possible valuations of the variables in the domain of an *OSF*-algebra interpretation $\mathscr{A}$, the *denotation* of an *OSF*-term $\psi = X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n)$ is defined as the *set* of domain elements:

$$[\![\psi]\!]^{\mathscr{A}} = \bigcup_{\alpha \in Val(\mathscr{A})} [\![\psi]\!]^{\mathscr{A},\alpha}. \tag{3}$$

The syntax of *OSF*-term allows some to be in a form where there is apparently ambiguous or even implicitly inconsistent information. For instance, in the *OSF*-term of Example 1, it is unclear what the attribute *name* could be. Similarly, if *string* and *number* are two sorts such that $string \wedge number = \bot$, it is not clear what the *ssn* attribute is for the *OSF*-term $X : \top\ (ssn \Rightarrow string,\ ssn \Rightarrow number)$, and whether indeed such a term's denotation is empty or not. The following notion is useful to this end.

Definition 3 (ψ-term). *A normal OSF-term* ψ *is of the form* $\psi = X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n)$ *where*:

- *there is at most one occurrence of a variable* Y *in* ψ *such that* Y *is the root variable of a non-trivial OSF-term* (*i.e., different than* Y: $\top$);
- s *is a non-bottom sort in* $\mathscr{S}$;
- $\ell_1, \ldots, \ell_n$ *are pairwise distinct features in* $\mathscr{F}$, $n \ge 0$,
- $\psi_1, \ldots, \psi_n$ *are normal OSF-terms.*

We call Ψ the set that they constitute.

Example 3. One could verify easily that the *OSF*-term:

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

is a ψ-term and always denotes exactly the same set as the one of Example 1.

Given an arbitrary *OSF*-term ψ, it is natural to ask whether there exists a ψ-term ψ' such that $[\![\psi]\!]^{\mathscr{A}} = [\![\psi']\!]^{\mathscr{A}}$ in every *OSF*-interpretation $\mathscr{A}$. We shall see in the next subsection that there is a straightforward normalization procedure that allows either to determine whether an *OSF*-term denotes the empty set or produce an equivalent ψ-term form for it.

Before we do that, let us make a few general but important observations about *OSF*-terms. First, the *OSF*-terms generalize first-order terms in many respects. In particular, if we see a first-order term as an expression denoting the set of all terms that it subsumes, then we obtain the special case where *OSF*-terms are interpreted as subsets of a free term algebra $\mathscr{T}(\Sigma, V)$, which can be seen naturally as a special *OSF*-algebra where the sorts form a flat lattice and the features are (natural number) positions. Recall that the first-order term notation $f(t_1, \ldots, t_n)$ is syntactic sugar for the ψ-term notation $f(1 \Rightarrow t_1, \ldots, n \Rightarrow t_n)$.[4]

Second, observe that since Equation 3 takes the union over all admissible valuations, it is natural to construe all variables occurring in an *OSF*-term to be implicitly existentially quantified at the term's outset. However, this latter notion is not very precise as it is only relative to *OSF*-terms taken out of external context. Indeed, it is not quite correct to assume so in the particular use made of them in definite relational clauses where variables may be shared among several goals. There, it will be necessary to relativize carefully this quantification to the global scope of such a clause.[5] Nevertheless, *assuming no further context*, the foregoing *OSF*-term semantics given above is one in which all variables are implicitly existential. To convince herself, the reader need only consider the equality $[\![X: s]\!]^{\mathscr{A}} = s^{\mathscr{A}}$ (which follows since $\bigcup_{\alpha \in Val(\mathscr{A})}(\{\alpha(X)\} \cap s^{\mathscr{A}}) = s^{\mathscr{A}}$). A corollary of this equality, is that it is natural to view sorts as particular (basic) *OSF*-terms. Indeed, their interpretations as either entities coincide.

Third, another important consequence of this type semantics is that the denotation of an *OSF*-term ψ is the empty set in all interpretations if ψ has an occurrence of a variable sorted by the empty sort $\bot$.[6] We shall call any *OSF*-term of the form $X: \bot$ an *empty OSF*-term. As observed above, any empty *OSF*-term denotes exactly the empty set. Dually, it is also clear that $[\![\psi]\!] = D^{\mathscr{A}}$ in all

[4]To render exactly first-order terms, feature positions should be such that $i(f(t_1, \ldots, t_n)) = t_i$ is defined only for $1 \leq i \leq n$. That is, feature positions should be partial functions. In our case, they are total so that if $i > n$ then $i(f(t_1, \ldots, t_n)) = \top$. Therefore, the terms that we consider here are "loose" first-order terms.

[5]See Section 3.4 for precise details.

[6]As a direct consequence of the universal set-theoretic identity: $f^{-1}(A \cap B) = f^{-1}(A) \cap f^{-1}(B)$, for any function f and sets A, B.

interpretations $\mathscr{A}$ if and only if all variables in ψ are sorted by $\top$. If ψ is of the form $Z : \top$, we call ψ a *trivial OSF-term.*

Fourth, it is important to bear in mind that we treat features as *total* functions. There are fine differences addressing the more general case of partial features and such deserves a different treatment. We limit ourselves to total features for the sake of simplicity.[7] This is equivalent to saying that, given an *OSF*-term,

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

and a variable $Z \notin Var(\psi)$, we have:

$$[\![\psi]\!]^{\mathscr{A},\alpha} = [\![X : s(l_1 \Rightarrow \psi_1, \ldots, l_n \Rightarrow \psi_n, l \Rightarrow Z : \top)]\!]^{\mathscr{A},\alpha}$$

for any feature symbol $\ell \in \mathscr{F}$, any *OSF*-interpretation $\mathscr{A}$ and valuation $\alpha \in Va\ell(\mathscr{A})$.

Finally, note that variables occurring in an *OSF*-term denote essentially an equality among attribute compositions as made clear by, say:

$$[\![X : \top(\ell_1 \Rightarrow Y : \top, \ell_2 \Rightarrow Y : \top)]\!]^{\mathscr{A}} = \{d \in D^{\mathscr{A}} | \ell_1^{\mathscr{A}}(d) = \ell_2^{\mathscr{A}}(d)\}.$$

This justifies semantically why we sometimes refer to variables as *coreference tags.*

3.2.2. OSF-clauses. An alternative syntactic presentation of the information conveyed by *OSF*-terms can be given using logical means as an *OSF*-term can be translated into a constraint formula bearing the same meaning. This is particularly useful for proof-theoretic purposes. A constraint normalization procedure can be devised in the form of semantics preserving simplification rules. A special syntactic form called *solved form* may be therefore systematically exhibited. This is the key allowing the effective use of types as constraints formulae in a Constraint Logic Programming context.

Definition 4 (*OSF*-Constraint). *An order-sorted feature constraint (OSF-constraint) is an atomic expression of either of the following forms:*

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

where X and Y are variables in $\mathscr{V}$, s is a sort in $\mathscr{S}$, and ℓ is a feature in $\mathscr{F}$. An order-sorted feature clause (OSF-clause) $\phi_1 \& \ldots \& \phi_n$ is a finite, possibly empty conjunction of OSF-constraints $\phi_1, \ldots, \phi_n (n \geq 0)$.

One may read the three atomic forms of *OSF*-constraints as, respectively, "X lies in sort s," "X is equal to Y," and "Y is the feature ℓ of X." The set $Var(\phi)$ of (free) variables occurring in an *OSF*-clause ϕ is defined in the standard way. *OSF*-clauses will always be considered equal if they are equal modulo the commutativity, associativity and idempotence of conjunction "&." Therefore, a clause can also be formalized as the set consisting of its conjuncts.

[7]Furthermore, this is what is realized in our implementation prototype [4].

The definition of the interpretation of *OSF*-clauses is straightforward. If $\mathscr{A}$ is an *OSF*-algebra and $\alpha \in Val(\mathscr{A})$, then $\mathscr{A}, \alpha \models \phi$, the *satisfaction* of the clause ϕ in the interpretation $\mathscr{A}$ under the valuation α, is given by:

- $\mathscr{A}, \alpha \models X : s$ iff $\alpha(X) \in s^{\mathscr{A}}$;
- $\mathscr{A}, \alpha \models X \doteq Y$ iff $\alpha(X) = \alpha(Y)$;
- $\mathscr{A}, \alpha \models X.\ell \doteq Y$ iff $\ell^{\mathscr{A}}(\alpha(X)) = \alpha(Y)$;
- $\mathscr{A}, \alpha \models \phi \,\&\, \phi'$ iff $\mathscr{A}, \alpha \models \phi$ and $\mathscr{A}, \alpha \models \phi'$.

Note that the empty clause is trivially valid everywhere.

We can associate an *OSF*-term $\psi = X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n)$ with a corresponding *OSF*-clause $\phi(\psi)$ as follows:

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

where $Y_1, \ldots, Y_n$ are the roots of $\psi_1, \ldots, \psi_n$, respectively. We say that the *OSF*-clause $\phi(\psi)$ is obtained from "dissolving" the *OSF*-term ψ.

Example 4. Let ψ be the *OSF*-term of Example 1. Its dissolved form $\phi(\psi)$ is the following *OSF*-clause:

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

Proposition 2. *If the OSF-clause $\phi(\psi)$ is obtained from dissolving the OSF-term ψ, then, for every OSF-algebra interpretation $\mathscr{A}$ and every $\mathscr{A}$-valuation α,*

$$[\![\psi]\!]^{\mathscr{A},\alpha} = \begin{cases} \{\alpha(X)\} & \text{if } \mathscr{A}, \alpha \models \phi(\psi), \\ \emptyset & \text{otherwise,} \end{cases}$$

and therefore,

$$[\![\psi]\!]^{\mathscr{A}} = \{\alpha(X) \mid \alpha \in Val(\mathscr{A});\, \mathscr{A}, \alpha \models \phi(\psi)\}.$$

Proof. This is immediate, from the definitions of the interpretations of *OSF*-terms and *OSF*-clauses. □

We will now define *rooted OSF-clauses* which, when solved, are in one-one correspondence with *OSF*-terms.

Given an *OSF*-clause ϕ, we define a binary relation on $Var(\phi)$, noted $X \overset{\phi}{\rightsquigarrow} Y$ (read, "Y is reachable from X in ϕ"), and defined inductively as follows. For all $X, Y \in Var(\phi)$:

- $X \overset{\phi}{\rightsquigarrow} X$;
- $X \overset{\phi}{\rightsquigarrow} Y$ if $Z \overset{\phi}{\rightsquigarrow} Y$ where $X.\ell \doteq Z$ is a constraint in ϕ.

A *rooted OSF-clause* ϕ_X is an *OSF*-clause ϕ together with a distinguished variable X (called its root) such that every variable Y occurring in ϕ is explicitly sorted (possibly as $Y: \top$), and reachable from X. We use ϕ^R for the injective (!) assignment of *rooted OSF*-clauses to *OSF*-terms ψ, *i.e.*, $\phi^R(\psi) = \phi(\psi)_{Root(\psi)}$.

Conversely, it is not always possible to assign a (unique) *OSF*-term to a (rooted) *OSF*-clause (*e.g.*, $X: s \,\&\, X: s'$). However, we see next that such a thing is possible in an important subclass of rooted *OSF*-clauses.

Given an *OSF*-clause ϕ and a variable X occurring in ϕ, we say that a conjunct in ϕ *constrains* the variable X if it has an occurrence of a variable which is reachable from X. One can thus construct the *OSF*-clause $\phi(X)$ which is rooted in X and consists of all the conjuncts of ϕ constraining X. That is, $\phi(X)$ is the maximal subclause of ϕ rooted in X.

Definition 5 (Solved OSF-Constraints). *An OSF-clause ϕ is called solved if for every variable X, ϕ contains*:

- *at most one sort constraint of the form $X: s$, with $\perp < s$;*
- *at most one feature constraint of the form $X.\ell \doteq Y$ for each ℓ; and,*
- *no equality constraint of the form $X \doteq Y$.*

We call Φ the set of all OSF-clauses in solved form, and Φ_R the subset of Φ of rooted solved OSF-clauses.

Given an *OSF*-clause ϕ, it can be normalized by choosing non-deterministically and applying any applicable rule among the four transformations rules shown in Figure 4 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. We also refer to any clause of the form $X: \perp$ as the `fail` clause.)

Theorem 1 (*OSF*-Clause Normalization). *The rules of Figure 4 are solution-preserving, finite terminating, and confluent (modulo variable renaming). Further-*

(*Inconsistent Sort*)
$$\frac{\phi \,\&\, X : \perp}{X : \perp}$$

(*Sort Intersection*)
$$\frac{\phi \,\&\, X : s \,\&\, X : s'}{\phi \,\&\, X : s \wedge s'}$$

(*Feature Decomposition*)
$$\frac{\phi \,\&\, X.\ell \doteq Y \,\&\, X.\ell \doteq Y'}{\phi \,\&\, X.\ell \doteq Y \,\&\, Y \doteq Y'}$$

(*Variable Elimination*)
$$\frac{\phi \,\&\, X \doteq Y}{\phi[X/Y] \,\&\, X \doteq Y} \qquad \text{if } X \in Var(\phi)$$

FIGURE 4. *OSF*-Clause Normalization Rules.

more, they always result in a normal form that is either the inconsistent clause or an OSF-clause in solved form together with a conjunction of equality constraints.

Proof. Solution preservation is immediate as each rule transforms an *OSF*-clause into a semantically equivalent one.

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

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

Given ϕ in normal form, we will refer to its part in solved form as *Solved*(ϕ), *i.e.*, ϕ without its variable equalities.

Example 5. The normalization of the *OSF*-clause given in Example 4 leads to the solved *OSF*-clause which is the conjunction of the equality constraint $N \doteq M$ and the following solved *OSF*-clause:

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

Given a rooted solved *OSF*-clause ϕ_X, we define the *OSF*-term $\psi(\phi_X)$ by:

$$\psi(\phi_X) = X: s(\ell_1 \Rightarrow \psi(\phi(Y_1)), \ldots, \ell_n \Rightarrow \psi(\phi(Y_n))),$$

where ϕ contains the constraint $X: s$ (if these are none of this form given explicitly, we can assume the implicit existence of $X: \top$ in ϕ, according to our convention of identifying *OSF*-clauses), and $X.\ell_1 \doteq Y_1, \ldots, X.\ell_n \doteq Y_n$ are all other constraints in ϕ with an occurrence of the variable X on the left-hand side.

3.2.3. OSF-graphs. We will now introduce the notion of *order-sorted feature graph* (*OSF*-graph) which is closely related to those of normal *OSF*-term and of rooted solved *OSF*-clause. The exact syntactic and semantic mutual correspondence between these three notions is to be established precisely.

Definition 6 (*OSF*-**graph**). *The elements g of the domain $D^{\mathscr{G}}$ of the order-sorted feature graph algebra $\mathscr{G}$ are directed labeled graphs $g = (N, E, \lambda_N, \lambda_E, X)$, where $\lambda_N : N \to \mathscr{S}$ and $\lambda_E : E \to \mathscr{F}$ are (node and edge, resp.) labelings and $X \in N$ is a distinguished node called the root, such that:*

- *each node of g is denoted by a variable X, i.e., $N \subseteq \mathscr{V}$;*
- *each node X of g is labeled by a non-bottom sort s, i.e., $\lambda_N(N) \subseteq \mathscr{S} - \{\bot\}$;*
- *each (directed) edge $\langle X, Y \rangle$ of g is labeled by a feature, i.e., $\lambda_E(E) \subseteq \mathscr{F}$;*
- *no two edges outgoing from the same node are labeled by the same feature, i.e., if $\lambda_E(\langle X, Y \rangle) = \lambda_E(\langle X, Y' \rangle)$, then $Y = Y'$ (g is deterministic);*
- *every node lies on a directed path starting at the root (g is connected).*

In the interpretation $\mathcal{G}$, the sort $s \in \mathcal{S}$ denotes the set $s^{\mathcal{G}}$ of *OSF*-graphs g whose root is labeled by a sort s' such that $s' \leq s$; that is,

$$s^{\mathcal{G}} = \{g = (N, E, \lambda_N, \lambda_E, X) | \lambda_N(X) \leq s\}.$$

The feature $\ell \in \mathcal{F}$ has the following denotation in $\mathcal{G}$. Let $g = (N, E, \lambda_N, \lambda_E, X)$. If there exists an edge $\langle X, Y \rangle$ labeled ℓ for some node Y of g, then Y is the root of $\ell^{\mathcal{G}}(g)$, and the (labeled directed) graph underlying $\ell^{\mathcal{G}}(g)$ is the maximally connected subgraph of g rooted at the node Y, $g' = (N_{|Y}, E_{|Y}, \lambda_N, \lambda_E, Y)$. If there is no edge outgoing from the root of g labeled ℓ, then $\ell^{\mathcal{G}}(g)$ is the *trivial graph* of $D^{\mathcal{G}}$ whose only node is the variable $Z_{\ell,g}$ labeled $\top$, where $Z_{\ell,g} \in \mathcal{V} - N$ is a new variable uniquely determined by the feature ℓ and the graph g; that is, if $\ell \neq \ell'$ or $g \neq g'$ then $Z_{\ell,g} \neq Z_{\ell',g'}$. In summary, if $g = (N, E, \lambda_N, \lambda_E, X)$, then:

$$\ell^{\mathcal{G}}(g) = \begin{cases} (N_{|Y}, E_{|Y}, \lambda_N, \lambda_E, Y), & \text{if } \lambda_E(\langle X,Y \rangle) = \ell \text{ for some } \langle X,Y \rangle \in E; \\ (\{Z_{\ell,g}\}, \emptyset, \{\langle Z_{\ell,g}, \top \rangle\}, \emptyset, Z_{\ell,g}) & \text{where } Z_{\ell,g} \in \mathcal{V} - N, \text{otherwise.} \end{cases}$$

We will present two concise ways of describing *OSF*-graphs. The first one assigns to a normal *OSF*-term ψ a (unique) *OSF*-graph $G(\psi)$. If $\psi = X : s$, then $G(\psi) = (\{X\}, \emptyset, \{\langle X, s \rangle\}, \emptyset, X)$. If $\psi = X : s(\ell_1 \Rightarrow \psi_1, \ldots, \ell_n \Rightarrow \psi_n)$, and $G(\psi_i) = (N_i, E_i, \lambda_{N_i}, \lambda_{E_i}, X_i)$, then $G(\psi) = (N, E, \lambda_N, \lambda_E, X)$ where:

- $N = \{X\} \cup N_1 \cup \ldots \cup N_n$;
- $E = \{\langle X, X_1 \rangle, \ldots, \langle X, X_n \rangle\} \cup E_1 \cup \ldots \cup E_n$;
- $\lambda_N(U) = \begin{cases} s & \text{if } U = X, \\ \lambda_{N_i}(U) & \text{if } U \in N_i - \left(\{X\} \cup \bigcup_{j=1}^{i-1} N_j\right); \end{cases}$
- $\lambda_E(e) = \begin{cases} \ell_i & \text{if } e = \langle X, X_i \rangle, \\ \lambda_{E_i}(e) & \text{if } e \in E_i. \end{cases}$

Conversely, we construct a (unique, normal) *OSF*-term $\psi(g)$ for any *OSF*-graph g. If X is the root of $g \in D^{\mathcal{G}}$, labeled with the sort $s \in \mathcal{S}$, and $\ell_1, \ldots, \ell_n$ are the (pairwise distinct) features in $\mathcal{F}$, $n \geq 0$, labeling all the edges outgoing from X, then there exists an *OSF*-term:

$$\psi(g) = X : s(\ell_1 \Rightarrow \psi(g_1), \ldots, \ell_n \Rightarrow \psi(g_n))$$

where $\ell_1^{\mathcal{G}}(g) = g_1, \ldots, \ell_n^{\mathcal{G}}(g) = g_n$. If, in this recursive construction, the root variable Y of $\psi(g')$ has already occurred earlier in some predetermined ordering of $\mathcal{F}^*$ then one has to put $Y : \top$ instead of $\psi(g')$. The uniqueness of $G(\psi)$ follows from the fixed choice of an ordering over $\mathcal{F}^*$ for normal *OSF*-terms.[8]

Corollary 1 (Graphical Representation of ψ-Terms). *The correspondences $\psi : D^{\mathcal{G}} \to \Psi$ and $G : \Psi \to D^{\mathcal{G}}$ between normal OSF-terms (ψ-terms) and OSF-graphs are*

[8]Without any loss of generality, we may assume an ordering on $\mathcal{F}$ which induces a lexicographical ordering on $\mathcal{F}^*$. We require that, in a *normal OSF*-term ψ of the form above, the features $\ell_1, \ldots, \ell_n$ be ordered, and that the occurrence of a variable Y as root of a non-trivial *OSF*-term is the least of all occurrences of Y in ψ according to the ordering on $\mathcal{F}^*$.

bijections. Namely,

$G \circ \psi = 1_{D^{\mathscr{G}}}$ and $\psi \circ G = 1_{\Psi}$.

Using this one-one correspondence, we can formally characterize the *OSF*-graph algebra as follows.

- $D^{\mathscr{G}} = \{G(\psi) | \psi$ is a normal *OSF*-term$\}$;
- $s^{\mathscr{G}} = \{G(X : s'(\ldots)) | s' \leq s\}$,
- $\ell^{\mathscr{G}}(G(X : s(\ldots, \ell \Rightarrow \psi', \ldots)))$
  $$= \begin{cases} G(X : s(\ldots, \ell \Rightarrow \psi', \ldots)) & \text{if } Root(\psi') = X, \\ G(\psi') & \textit{otherwise}; \end{cases}$$
- $\ell^{\mathscr{G}}(G(\psi)) = G(Z_{\ell, G(\psi)} : \top)$, otherwise; where $Z \notin Var(\psi)$.

Note that, in particular, $\ell^{\mathscr{G}}(G(X : s(\ell \Rightarrow X : \top))) = G(X : s(\ell \Rightarrow X : \top))$.

We have defined the following mappings:

$$\begin{aligned} &\psi_\phi : \Phi_R \to \Psi \\ &\psi_G : D^{\mathscr{G}} \to \Psi \\ &G \;: \Psi \to D^{\mathscr{G}} \\ &\phi \;: \Psi \to \Phi_R \end{aligned}$$

somehow "overloading" the notation of mapping ψ $(= \psi_\phi + \psi_G)$ to work either on rooted solved *OSF*-clauses or *OSF*-graphs.

It follows that Corollary 1 can be extended and reformulated as

Proposition 3 (Syntactic Bijections). *There is a one-one correspondence between OSF-graphs, normal OSF-terms, and rooted solved OSF-clauses as the syntactic mappings* $\psi : (\Phi_R + D^{\mathscr{G}}) \to \Psi$, $G : \Psi \to D^{\mathscr{G}}$, *and* $\phi : \Psi \to \Phi_R$ *put the syntactic domains* Ψ, $D^{\mathscr{G}}$, *and* Φ_R *in bijection. That is,*

$1_\Psi = \psi_G \circ G$ and $G \circ \psi_G = 1_{D^{\mathscr{G}}}$,

$1_{\Phi_R} = \phi \circ \psi_\phi$ and $\psi_\phi \circ \phi = 1_\Psi$.

Proof. This is clear from the considerations above. The bijection between *OSF*-graphs and rooted solved *OSF*-clauses can be defined via *OSF*-terms. Therefore, we shall take the freedom of cutting the intermediate step in allowing notations such as $\phi(g)$ or $G(\phi)$. It is interesting, however, to see how a solved clause ϕ with the root X corresponds uniquely to an *OSF*-graph $G(\phi_X)$ which is rooted at the node X. A constraint $X : s$ "specifies" the labeling of the node X by the sort s, and a constraint $X.\ell \doteq Y$ specifies an edge $\langle X, Y \rangle$ labeled by the feature ℓ. If, for a variable Z, there is no constraint of the form $Z : s$, then the node Z of $G(\phi)$ is labeled $\top$. Conversely, every clause $\phi(g)$ together with the root X of the *OSF*-graph g is a rooted solved clause, since the reachability of variables corresponds directly to the graph-theoretical reachability of nodes. □

As for meaning, we shall presently give three independent semantics, one for each syntactical representation. Each semantics allows an apparently different formalization of a multiple-inheritance ordering. We show then that they all coincide _rt to semantic transparency of the syntactic mappings G, ψ, and ϕ.

3.3. OSF-orderings and semantic transparency

Endomorphisms on a given *OSF*-algebra $\mathcal{A}$ induce a natural partial ordering.

Definition 7 (Endomorphic Approximation). *On each OSF-algebra $\mathcal{A}$ a preorder $\sqsubseteq_{\mathcal{A}}$ is defined by saying that, for two elements d and e in $d^{\mathcal{A}}$, d approximates e,*

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

We remark that all *OSF*-graphs are approximated by the trivial *OSF*-graph $G(Z : \top)$ consisting of one node Z labeled $\top$, *i.e.*, for all $g \in D^{\mathcal{G}}$, $G(Z : \top) \sqsubseteq_{\mathcal{G}} g$. Clearly an endomorphism $\gamma : D^{\mathcal{G}} \mapsto D^{\mathcal{G}}$ can be extended from $\gamma(Z : \top) = g$ by setting $\gamma(Z_i : \top) = g_i$, if $\ell_i^{\mathcal{G}}(g) = g_i$ and $\ell_i^{\mathcal{G}}(Z : \top) = Z_i : \top$ for some "new" variable Z_i, *etc.* ...

The following results aim at characterizing the solutions of a solved (not necessarily connected) clause in an *OSF*-algebra. The essential point is to demonstrate that all solutions in any *OSF*-algebra of a set of *OSF*-constraints can be obtained as homomorphic images from one solution in one particular subalgebra of *OSF*-graphs—the canonical graph algebra induced by ϕ.

Definition 8 (Canonical Graph Algebra). *Let ϕ be an OSF-formula in solved-form. The subalgebra $\mathcal{G}[D^{\mathcal{G},\phi}]$ of the OSF-graph algebra $\mathcal{G}$ generated by $D^{\mathcal{G},\phi} = \{G(\phi(X)) | X \in Var(\phi)\}$ of all maximally connected subgraphs of the graph form of ϕ is called the canonical graph algebra induced by ϕ.*

It is interesting to observe that, for ϕ an *OSF*-formula in solved-form, the set $D^{\mathcal{G},\phi}$ is *almost* an *OSF*-algebra. More precisely, it is closed under feature application up to trivial graphs, in the sense that for all $\ell \in \mathcal{F}$, $\ell^{\mathcal{G}}(g) \notin D^{\mathcal{G},\phi} \Rightarrow \ell^{\mathcal{G}}(g) = G(Z_{\ell,g} : \top)$. In other words, the $\mathcal{F}$-closure of $D^{\mathcal{G},\phi}$ adds only mutually distinct trivial graphs with root variables outside $Var(\phi)$.

Definition 9 (ϕ-Admissible Algebra). *Given an OSF-clause in solved form ϕ, any OSF-algebra $\mathcal{A}$ is said to be ϕ-admissible if there exists some $\mathcal{A}$ valuation α such that $\mathcal{A}, \alpha \models \phi$.*

It comes as no surprise that the canonical graph algebra induced by any solved *OSF*-clause ϕ is ϕ-admissible, and so is any *OSF*-algebra containing it—$\mathcal{G}$, in particular. The following is a direct consequence of this fact.

Corollary 2 (Canonical Solutions). *Every solved OSF-clause $\phi(X)$ is satisfiable in the OSF-graph algebra $\mathcal{G}$ under any $\mathcal{G}$-valuation α such that $\alpha(X) = G(\phi(X))$.*

In other words, according to the observation made above the set $D^{\mathcal{G},\phi}$ contains all the non-trivial graphs solutions. In fact, the canonical graph algebra induced by ϕ is weakly initial in **OSF**(ϕ), the full subcategory of ϕ-admissible *OSF* algebras.[9] This is expressed by the following proposition.

[9]An object o is weakly initial (resp., final) in a category if there is at least one arrow $a : o \to o'$ (resp., $a : o' \to o$) for any other object o' in the category. Weakly initial (resp., final) objects are not necessarily mutually isomorphic. If the object o admits *exactly one* such arrow, it is initial (resp., final). Initial (resp., final) objects are necessarily mutually isomorphic.

Theorem 2 (Extracting Solutions). *The solutions of a solved OSF-clause ϕ in any ϕ-admissible OSF-algebra $\mathcal{A}$ are given by OSF-algebra homomorphisms from the canonical graph algebra induced by ϕ in the sense that for each $\alpha \in Val(\mathcal{A})$ such that $\mathcal{A}, \alpha \models \phi$ there exists an OSF-algebra homomorphism $\gamma : \mathcal{G}[D^{\mathcal{G},\phi}] \mapsto \mathcal{A}$ such that:*

$$\alpha(X) = \gamma(G(\phi(X))).$$

Proof. Let α be a solution of ϕ in $\mathcal{A}$; *i.e.*, such that $\mathcal{A}, \alpha \models \phi$. We define a homomorphism $\gamma : \mathcal{G}[D^{\mathcal{G},\phi}] \mapsto \mathcal{A}$ by setting $\gamma(G(\phi(X))) = \alpha(X)$, and extending from there homomorphically. This is possible since the two compatibility conditions are satisfied for any graph $g = G(\phi(X))$. Indeed, if $\ell^{\mathcal{G}}(g) = g'$, then there are two possibilities: (1) $g' = G(Z : \top)$ where $Z \notin Var(\phi)$, or (2) $g' = G(\phi(Y))$ for some variable Y occurring in ϕ; namely, in a constraint of the form $X.\ell \doteq Y$. Then, $\lambda^{\mathcal{A}}(\alpha(X)) = \alpha(Y)$. This means that for all $g \in D^{\mathcal{G},\phi}$ of the form $g = G(\phi(X))$, it is the case that $\gamma(f^{\mathcal{G}}(g) = \ell^{\mathcal{A}}(\gamma(g))$. If $G(\phi(X)) \in s^{\mathcal{G}}$ (*i.e.*, if $G(\phi(X))$ is labeled by a sort s' such that $s' \leq s$), then ϕ contains a constraint of the form $X : s'$, and therefore $\alpha(X) \in s'^{\mathcal{A}}$. This means that if $g \in s^{\mathcal{G}}$ then $\gamma(g) \in s^{\mathcal{A}}$ and the second condition is also satisfied (if $g = G(Z : \top)$, then this is trivially true). □

Some known results are easy corollaries of the above proposition. The first one is a result in [19], here slightly generalized from so-called set-descriptions to clauses.

For a solved clause ϕ, Theorem 2 can be used to infer that the image of a solution in one *OSF*-algebra under an *OSF*-homomorphism (sufficiently defined) is a solution in the other: If $\alpha \in Val(\mathcal{A})$ with $\mathcal{A}, \alpha \models \phi$ and $\alpha' \in Val(\mathcal{B})$ is defined by $\alpha'(X) = \gamma(\alpha(X))$ for some $\gamma : \mathcal{A} \mapsto \mathcal{B}$, then simply let $\gamma' : \mathcal{G} \mapsto \mathcal{A}$ be the homomorphism existing according to Theorem 2 (*i.e.*, such that $\alpha(X) = \gamma(G(\phi(X)))$) and then $\alpha'(X) = (\gamma \circ \gamma')(G(\phi(X)))$, and thus $\mathcal{B}, \alpha' \models \phi$. This fact, a standard property expected from homomorphisms in other formalisms, holds also for a not necessarily solved clause.

Proposition 4 (Extending Solutions). *Let $\mathcal{A}$ and $\mathcal{B}$ be two OSF-interpretations, and let $\gamma : \mathcal{A} \mapsto \mathcal{B}$ be an OSF-homomorphism between them. Let ϕ be any OSF-clause such that $\mathcal{A}, \alpha \models \phi$ for some $\mathcal{A}$-valuation α. Then, for any $\mathcal{B}$-valuation β obtained as $\beta = \gamma \circ \alpha$ it is also the case that $\mathcal{B}, \beta \models \phi$.*

Proof. $\mathcal{A}, \alpha \models \phi$ means that $\mathcal{A}, \alpha \models \phi'$ for every atomic constraint conjunct ϕ' of ϕ. If ϕ' is of the form $X.\ell \doteq Y$, then $\ell^{\mathcal{B}}(\beta(X)) = \ell^{\mathcal{B}}(\gamma(\alpha(X))) = \gamma(\ell^{\mathcal{A}}(\alpha(X))) = \gamma(\alpha(Y)) = \beta(Y)$. If ϕ' is of the form $X : s$, this means that $\alpha(X) \in s^{\mathcal{A}}$; and then, $\beta(X) = \gamma(\alpha(X)) \in s^{\mathcal{B}}$. Therefore, all atomic constraints in ϕ are also true in $\mathcal{B}$ under β and so is ϕ. □

Theorem 3 (Weak Finality of $\mathcal{G}$). *There exists a totally defined homomorphism γ from any OSF-algebra $\mathcal{A}$ into the OSF-graph algebra $\mathcal{G}$.*

Proof. For each $d \in D^{\mathcal{A}}$ we choose some variable $X_d \in Var$ to denote a node. There is an edge $\langle X_d, X_{d'} \rangle$ labeled ℓ if $\ell^{\mathcal{A}}(d) = d'$. Each node X_d is labeled with the greatest common subsort of all sorts such that $d \in s^{\mathcal{A}}$ (which exists since we assume $\mathcal{S}$ to be finite). We thus obtain a graph g whose nodes are denoted by variables and labeled by sorts and whose (directed) edges are labeled by features.

We define $\gamma(d)$ to be the OSF-graph which is the maximally connected subgraph of g rooted in X_d and whose root is X_d. Obviously, we obtain a homomorphism. □

In other words, the *OSF*-graph algebra $\mathcal{G}$ is a weakly final object in the category **OSF** of *OSF*-algebras with *OSF*-homomorphisms. Therefore, we have the interesting situation where, if in the *OSF*-algebra $\mathcal{A}$ a solution $\alpha \in Val(\mathcal{A})$ of an *OSF*-clause ϕ exists, it is given by a homomorphism from the *OSF*-graph algebra $\mathcal{G}$ into $\mathcal{A}$, and a solution of ϕ in $\mathcal{G}$ can always be obtained as the image of α under a homomorphism from $\mathcal{A}$ into $\mathcal{G}$.

Therefore, we may obtain purely semantically as a corollary the following result due to Smolka which establishes that the *OSF*-algebra $\mathcal{G}$ is a "canonical model" for *OSF*-clause logic [18]:

Corollary 3 (Canonicity of $\mathcal{G}$). *An OSF-clause is satisfiable iff it is satisfiable in the OSF-graph algebra.*

Proof. This is a direct consequence of Theorem 2 and Theorem 3. □

This canonicity result was originally proven proof-theoretically by Smolka [18], and then by Dörre and Rounds [14], directly, for the case of feature graph algebras without sorts.

Corollary 4 (Principal Canonical Solutions). *The OSF-graph $G(\phi(X))$ approximates every other graph g assigned to the variable X by a solution of an OSF-clause ϕ; i.e., the solution $\alpha \in Val(\mathcal{G})$, $\alpha(X) = G(\phi(X))$ is a principal solution of ϕ in the OSF-algebra $\mathcal{G}$.*

Proof. This is a specialization of Theorem 2 for the case of $\mathcal{A} = \mathcal{G}$. □

That is, graph solutions are most general. A related fact—the existence of principal solutions in the feature graph algebra (without sorts)—has already been proven by Smolka (directly; the generalization in Theorem 2 seems to be new).

The following fact comes from Proposition 3 for the special case of a rooted solved *OSF*-clause, since from $\phi(G(\psi)) = \phi(\psi)$ and from Proposition 2 we know that $[\![\psi]\!]^{\mathcal{A}} = \{\alpha(X) | \mathcal{A}, \ \alpha \models \phi(G(\psi))\}$. It states that the elements of the set denoted by an *OSF*-term in any *OSF*-algebra can be obtained by "instantiating" *one* element in the set denoted by this *OSF*-term in one particular *OSF*-algebra (namely, its principal element).

Theorem 4 (Interpretability of Canonical Solutions). *If the normal OSF-term ψ corresponds to the OSF-graph $G(\psi) \in D^{\mathcal{G}}$, then its denotation can be characterized by*:

$$[\![\psi]\!]^{\mathcal{A}} = \{\gamma(G(\psi)) | \gamma : \mathcal{G} \mapsto \mathcal{A} \text{ is an OSF algebra homomorphism}\}. \qquad (4)$$

The following corollary expresses the intuitive idea that some of the solutions of a clause are solutions to stronger clauses (which are obtained via *OSF*-graph algebra endomorphisms; *cf.* also, Corollary 8).

Corollary 5 (Homomorphism Refinability of Solutions). *If the normal OSF-term ψ corresponds to the OSF-graph $g = G(\psi) = G(\phi(\psi))$, then its denotation can be characterized by*:

$$[\![\psi]\!]^{\mathcal{A}} = \{\alpha(X) | \mathcal{A}, \alpha \models \phi(\gamma(g)); \gamma : \mathcal{G} \mapsto \mathcal{G} \text{ is an endomorphism}\}. \qquad (5)$$

Proof. The mapping $\gamma_1 : \mathscr{G} \mapsto \mathscr{A}$ given by $\alpha'(x) \mapsto \alpha(X)$ is clearly an *OSF*-algebra homomorphism; so is the mapping $\gamma_2 : \mathscr{G} \mapsto \mathscr{G}$ given by $G(\phi(X)) \mapsto \alpha'(X)$. The homomorphisms γ of equation (5) are of the form $\gamma_2 \circ \gamma_1$. □

Corollary 6 (ψ-Types as Graph Filters). *The denotation of a normal OSF-term in the OSF-graph algebra is the set of all OSF-graphs which the corresponding OSF-graph approximates; i.e.,*

$$[\![\psi]\!]^{\mathscr{G}} = \{G \in D^{\mathscr{G}} | G(\psi) \sqsubseteq_{\mathscr{G}} G\}.$$

Proof. This is a simple reformulation of (4) for the case of $\mathscr{A} = \mathscr{G}$. □

In lattice-theoretic terms, this result characterizes the canonical type denotation of a ψ-term as the principal approximation filter generated by its graph form.

We readily obtain the following result established in [14] as an immediate consequence of Theorem 4.

Corollary 7 (Dörre-Rounds). *The approximation relation between two elements d and d' in an OSF-algebra $\mathscr{A}$ can be characterized on OSF-terms as:*

$$d \sqsubseteq_{\mathscr{A}} d' \textit{ iff for all OSF-terms } \psi,\ d' \in [\![\psi]\!]^{\mathscr{A}} \textit{ whenever } d \in [\![\psi]\!]^{\mathscr{A}}.$$

Proof. If $\gamma(G(\psi)) = d$ for some $\gamma : \mathscr{G} \mapsto \mathscr{A}$ according to (4) assuming $d \in [\![\psi]\!]^{\mathscr{A}}$, and $\gamma'(d) = d'$ according to the assumption $d \sqsubseteq_{\mathscr{A}} d'$, for some endomorphism $\gamma' : \mathscr{A} \mapsto \mathscr{A}$, then $d' = (\gamma \circ \gamma')(G(\psi))$, and one can apply (4) again—In the other direction, the condition on all *OSF*-terms says exactly that from $\gamma(d) = d'$ a homomorphic extension $\gamma : \mathscr{A} \mapsto \mathscr{A}$ can be defined. □

Besides the approximation ordering on *OSF*-graphs, there are two other natural partial orders that can be defined over *OSF*-terms and *OSF*-clauses. Namely, subsumption and implication, respectively.

Definition 10 (OSF-Term Subsumption). *Let ψ and ψ' be two OSF-terms, then, $\psi \leq \psi'$ ("ψ is subsumed by ψ'") iff, for all OSF-algebras $\mathscr{A}$, $[\![\psi]\!]^{\mathscr{A}} \subseteq [\![\psi']\!]^{\mathscr{A}}$.*

Definition 11 (OSF-Clause Implication). *Let ϕ and ϕ' be two OSF-clauses; then, $\phi \succeq \phi'$ ("ϕ implies ϕ'") iff, for all $\mathscr{A}$ and α such that $\mathscr{A}, \alpha \models \phi$, there exists α' such that $\forall X \in Var(\phi) \cap Var(\phi')$, $\alpha'(X) = \alpha(X)$, and $\mathscr{A}, \alpha' \models \phi'$.*

Definition 12 (Rooted OSF-Clause Implication). *Let ϕ_X and $\phi'_{X'}$, be two rooted OSF-clauses with no common variables; then, $\phi_X \succeq \phi'_{X'}$, iff $\phi \succeq \phi'[X/X']$.*

Theorem 5 (Semantic Transparency of Orderings). *If the normal OSF-terms ψ, ψ', the OSF-graphs g, g' and the rooted solved OSF-clauses ϕ_X, ϕ'_X respectively correspond to one another through the syntactic mappings, then the following are equivalent statements:*

- $g \sqsubseteq_{\mathscr{G}} g'$; *"g is a graph approximation of g';"*
- $\psi' \leq \psi$; *"ψ' is a subtype of ψ;"*
- $\phi'_X \succeq \phi_X$; *"ϕ is true of X whenever ϕ' is true of X;"*
- $[\![\psi]\!]^{\mathscr{G}} \subseteq [\![\psi']\!]^{\mathscr{G}}$. *"the set of graphs filtered by ψ is contained in that filtered by ψ'."*

Proof. This follows from Proposition 2, Theorem 4 and Corollary 6.[10] □

We want to exhibit the following direct consequence of the above considerations.

Corollary 8 (Endomorphic Entailment). *If one rooted solved OSF-clause ϕ is implied by another, ϕ' ($\phi' \succeq \phi$), then it is a homomorphic image of ("more instantiated than") ϕ' in the following way:*

$$\phi = \phi(\gamma(G(\phi')))$$

for some OSF-graph algebra endomorphism γ.

The following two theorems are immediate and tie back our setting to unification as constraint-solving and principal solution computation.

Theorem 6 (ψ-Term Unification). *Let ψ_1 and ψ_2 be two ψ-terms. Let ϕ be the normal form of the OSF-clause $\phi(\psi_1) \& \phi(\psi_2) \& Root(\psi_1) \doteq Root(\psi_2)$. Then, ϕ is the inconsistent clause iff their GLB with respect to $\leq$ is $\perp$ (i.e., iff their denotations in all interpretations have an empty intersection). If ϕ is not the inconsistent clause, then their GLB (modulo variable renaming) $\psi_1 \wedge \psi_2$ is given by the normal OSF-term $\psi(Solved(\phi))$.*

Theorem 7 (Computing the LUB of two *OSF*-graphs). *Let g_1 and g_2 be two OSF-graphs. Let g be the OSF-graph, if it exists, given by $g = G(Solved(\phi(g_1) \& \phi(g_2)))$. Then, g is approximated by both g_1 and g_2 and is the principal OSF-graph for $\sqsubseteq_{\mathscr{G}}$ (i.e., approximating all other ones) with this property.*

3.4. Definite Clauses over OSF-algebras

In this section, we assume familiarity with the Höhfeld–Smolka CLP scheme. The reader in need of background will find all essential material necessary for understanding what follows in Appendix A.

3.4.1. Definite clauses and queries over OSF-terms. A LIFE program of the form considered here consists of a conjunction of definite clauses $\mathscr{C}$ over ψ-terms of the form:

$$\mathscr{C} \equiv r(\psi_0) \leftarrow r_1(\psi_1) \& \ldots \& r_m(\psi_m). \tag{6}$$

We denote by $\mathscr{R}$ the set of all relation (predicate) symbols occurring in a given program. For simplicity of notation, we consider all relation symbols $r \in \mathscr{R}$ to be monadic.

Given an *OSF*-algebra $\mathscr{A}$, an interpretation of the program is a structure $\mathscr{M} = \langle \mathscr{A}, (r^{\mathscr{M}})_{r \in \mathscr{R}} \rangle$ consisting of $\mathscr{A}$ and relations $r^{\mathscr{M}}$ over $D^{\mathscr{A}}$ interpreting every symbol r occurring in the program. Such a structure $\mathscr{M}$ extending $\mathscr{A}$ models a

[10] Strictly speaking, our *OSF*-orderings are preorders rather than orderings. It really does not matter, in fact. Recall that a preorder (reflexive, transitive) o is a "looser" structure than either an order (anti-symmetric preorder) or an equivalence (symmetric preorder). It may be tightened into an order by factoring over its underlying equivalence ($\equiv_o = o \cap o^{-1}$), its "symmetric core." Then, the quotient set over $\equiv_o$ is partially ordered by o. Hence, if we define, in all three frameworks, *equivalence* as the symmetric core ($\equiv_o$) of the corresponding preorder ($o = \sqsubseteq, \leq, \succeq$), then Theorem 5 extends readily to these equivalence relations, and therefore the quotients are in order-bijection.

definite clause $\mathscr{C}$ in the program of the form of Expression (6) if $r^{\mathscr{M}}(d)$ holds whenever $r_1^{\mathscr{M}}(d_1)$ and ... and $r_m^{\mathscr{M}}(d_m)$ holds, for all elements $d, d_1, \ldots, d_n$ of $D^{\mathscr{A}}$ such that $\langle d, d_1, \ldots, d_n \rangle \in [\![\langle \psi_0, \psi_1, \ldots, \psi_n \rangle]\!]^{\mathscr{A}}$ (where the notation $[\![\langle \psi_1, \ldots, \psi_n \rangle]\!]^{\mathscr{A}}$ is shorthand for $\bigcup_{\alpha \in Val(\mathscr{A})} [\![\psi_1]\!]^{\mathscr{A},\alpha} \times \ldots \times [\![\psi_n]\!]^{\mathscr{A},\alpha}$).

The structure $\mathscr{M}$ is a model of the program if $\mathscr{M}$ models every definite clause $\mathscr{C}$ in the program. The meaning of a program is the class of minimal models extending the *OSF*-algebras over a given *OSF*-signature.[11]

A query, or resolvent, is a conjunction of atomic formulae of the form $r(\psi)$ and of *typing constraints* of the form $X \doteq \psi$, where r is a relational symbol and ψ is an *OSF*-term. Such an expression has for interpretation: $\mathscr{A}, \alpha \models X \doteq \psi$ if and only if $\alpha(X) \in [\![\psi]\!]^{\mathscr{A},\alpha}$.

Definition 13 (LIFE Resolution Rule). *A resolvent over OSF-terms* $\boldsymbol{R} \equiv R \;\&\; r(\psi)$ *reduces in one* resolution step, *choosing the query conjunct* $r(\psi)$ *and the (renamed) program clause* $\mathscr{C} \equiv r(\psi_0) \leftarrow r_1(\psi_1) \;\&\ldots\&\; r_m(\psi_m)$ *non-deterministically, to the resolvent* $\boldsymbol{R}' \equiv R \;\&\; r_1(\psi_1) \;\&\ldots\&\; r_m(\psi_m) \;\&\; X \doteq (\psi \wedge \psi_0)$, *where* $X = Root(\psi)$.

If the GLB of ψ and ψ_0 is $\perp$ ("bottom"), then $\boldsymbol{R}'$ is equivalent to the `fail` constraint. Iterated application of this rule yields a derivation sequence of the query $\boldsymbol{R}$. The answer to the query

$$\boldsymbol{R} \equiv r_1(\psi_1) \;\&\ldots\&\; r_m(\psi_m)$$

computed in a (terminating) derivation sequence is either the `fail` constraint or a conjunction of typing constraints

$$X_1 \doteq \psi_1' \;\&\ldots\&\; X_n \doteq \psi_n' \;\&\; Z_1 \doteq \psi_1'' \;\&\ldots\&\; Z_m \doteq \psi_m''.$$

Here, X_i is the root variable of the query *OSF*-term ψ_i, as well as of the answer *OSF*-term ψ' (which is subsumed by ψ_i). The *OSF*-terms ψ_j'' are rooted in new variables Z_j; *i.e.*, $Z_j \notin Var(\boldsymbol{R})$. All the new variables are implicitly existentially quantified. We say that "the answer *OSF*-terms interpreted in $\mathscr{A}$ contain the elements $d_1', \ldots, d_n'$," in order to abbreviate the fact that there exist elements $d_1'', \ldots, d_m''$ such that $\langle d_1', \ldots, d_n', d_1'', \ldots, d_m'' \rangle \in [\![\langle \psi_1', \ldots, \psi_n', \psi_1'', \ldots, \psi_m'' \rangle]\!]^{\mathscr{A}}$.

Theorem 8 (Correctness of LIFE Resolution). *The resolution rule for definite clauses over OSF-terms is sound and complete.*

That is, given the query $r_1(\psi_1), \ldots, r_n(\psi_n)$, the relations $r_1^{\mathscr{M}}(d_1), \ldots, r_n^{\mathscr{M}}(d_n)$ hold in the minimal model $\mathscr{M}$ of the program extending the *OSF*-algebra $\mathscr{A}$ for elements $d_1, \ldots, d_n$ in the sets denoted by the query *OSF*-terms $\psi_1, \ldots, \psi_n$ if and only if there exists a derivation of the query yielding an answer such that the answer *OSF*-terms interpreted in $\mathscr{A}$ contains these elements $d_1, \ldots, d_n$.

Proof. This is an immediate consequence of Proposition 5, Theorem 9, and Proposition 6 in the next section. □

[11] Minimality is with respect to set-inclusion on the relations $r^{\mathscr{M}}$.

3.4.2. Definite clauses over OSF constraints.

Proposition 5. *The definite clause $\mathscr{C} \equiv r(\psi_0) \leftarrow r_1(\psi_1) \& \ldots \& r_m(\psi_m)$ over ψ-terms has the same meaning as the following definite clause over OSF-clause constraints*:

$$r(X) \leftarrow r_1(X_1) \& \ldots \& r_m(X_m) \& \phi(\psi_1) \& \ldots \phi(\psi_m) \& \phi(\psi_0).$$

The resolvent over ψ-terms $r_1(\psi_1) \& \ldots \& r_m(\psi_m)$ is equivalent to the OSF-constraint resolvent $r_1(X_1) \& \ldots \& r_m(X_m) \& \phi(\psi_1) \& \ldots \phi(\psi_m)$.

Proof. We do not change the meaning of $\mathscr{C}$ if we replace it by a definite clause over typing constraints; *i.e.*, of the form:

$$X \doteq \psi \& X_1 \doteq \psi_1 \& \ldots X_m \doteq \psi_m \to (r_1(X_1) \& \ldots \& r_m(X_m) \to r(X)).$$

Of course, this clause can be written as the definite clause:

$$r(X) \leftarrow r_1(X_1) \& \ldots \& r_m(X_m) \& X_1 \doteq \psi_1 \& \ldots X_m \doteq \psi_m \& X \doteq \psi.$$

Here, $X_1, \ldots, X_m$, X can be chosen as the root variables of, respectively, $\psi_1, \ldots, \psi_m$, ψ or, equivalently, as new variables. In the first case, $X_i \doteq \psi_i$ is, after dissolving the OSF-term, exactly the solved OSF-clause $\phi(\psi_i)$ which corresponds (uniquely) to ψ_i, and the definite clause becomes the one in the first statement. The second statement follows similarly. □

The resolution rule for OSF-constraint resolvents is stated as follows. The resolvent $\boldsymbol{R} \equiv R \& \phi$ reduces to $\boldsymbol{R}' \equiv R \& R' \& \phi \& \phi$, by choosing the conjunct $r(X)$ in R and the (renamed) program clause $r(X) \leftarrow R' \& \phi'$ non-deterministically (R and R' are conjunctions of relational atoms of the form $r(X)$, and ϕ and ϕ' are OSF-clauses).

Theorem 9 (Soundness and completeness of OSF-constraint resolution). *For every interpretation $\mathscr{A}$ and valuation such that an OSF-constrained resolvent $\boldsymbol{R}$ holds, then so does a resolvent derived from it. If $\mathscr{M}$ is a minimal model of the program, and formula α is a solution of $\boldsymbol{R}$ in $\mathscr{M}$, then there exists a sequence of reductions of $\boldsymbol{R}$ to a solved OSF-clause constraint ϕ exhibiting α as its solution.*

Proof. This follows from instantiating the CLP scheme of [15] (*cf.*, Appendix Section A). The role of the constraint language in this scheme is taken by OSF-clauses as constraints together with OSF-algebras as interpretations.

The soundness of the resolution rule is clear: Under every interpretation $\mathscr{A}$ and every valuation such that R holds, then so does R'; *i.e.*, $[\![R']\!]^{\mathscr{A}} \subseteq [\![R]\!]^{\mathscr{A}}$. It is also not difficult to prove its completeness: If $\mathscr{M}$ is a minimal model of the program, and $\alpha \in [\![R]\!]^{\mathscr{M}}$ is a solution of the formula R in $\mathscr{M}$, then there exists a sequence of reductions of R to a solved OSF-clause ϕ such that $\alpha \in [\![\phi]\!]^{\mathscr{M}}$. □

Now we can look at the connection with the previous section: Let ϕ' be the solved-form OSF-clause constituting an answer of the query:

$$\boldsymbol{R} \equiv r_1(X_1) \& \ldots \& r_m(X_m) \& \phi(\psi_1) \& \ldots \phi(\psi_m).$$

If ϕ'' is the conjunction of all OSF-constraints in ϕ' constraining the (new) variables $Z_1, \ldots, Z_m$ which are not reachable from $X_1, \ldots, X_n$, ϕ' can be written as:

$$\phi'(X_1)\,\&\ldots\&\,\phi'(X_n)\,\&\,\phi''(Z_1)\,\&\ldots\&\,\phi''(Z_m).$$

Let us call $\psi'_1,\ldots,\psi'_n$, and $\psi''_1,\ldots,\psi''_m$ the normal *OSF*-terms which correspond uniquely to the rooted solved *OSF*-clauses in this conjunction. Then we say that ϕ' *corresponds* to the typing constraint $X_1 \doteq \psi'_1\,\&\ldots\&\,X_n \doteq \psi'_n\,\&\,Z_1 \doteq \psi''_1\,\&\ldots\&\,Z_m \doteq \psi''_m$. Clearly, the two constraints are equivalent.

Proposition 6. *Every answer of a query over OSF-terms (obtained by ψ-term resolution) corresponds to an answer of an OSF-constrained query (obtained by OSF-constrained resolution) and vice versa.*

Proof. *This follows from the above and Theorem* 5. □

3.4.3. OSF-graphs computed by a LIFE program. Let us call query-*OSF*-graphs those *OSF*-graphs $G(\psi_1),\ldots,G(\psi_n)$ which correspond uniquely to the *OSF*-terms $\psi_1,\ldots,\psi_n$ in a query $\boldsymbol{R}$. Note that a solution of $\boldsymbol{R}$ in the *OSF*-graph algebra $\mathscr{G}$ consists of *OSF*-graphs g_i which (1) approximate the query-*OSF*-graphs, *i.e.*, $g_i \sqsubseteq_{\mathscr{G}} G(\psi_i)$ and (2) satisfy the relation r_i, that is, $r_i^{\mathscr{M}}(g_i)$ holds in the minimal model $\mathscr{M}$ of the program extending the *OSF*-algebra $\mathscr{G}$. Every *OSF*-graph approximated by a solution (*i.e.*, lying in its graph filter; *cf.* Corollary 6) is also a solution.

Theorem 10 (*OSF*-graph Resolution and Endomorphic Refinement). *Every terminating non-failing derivation sequence of a query **R** yields a unique OSF-graph algebra endomorphism γ_0. The images of the query-OSF-graphs (under these endomorphisms γ_0) are principal solutions in the OSF-graph algebra of **R**. Every solution of the query is approximated by one of the principal solutions thus obtained.*

More precisely, the images are the principal elements for which the query relations hold in the *OSF*-graph algebra, and the principal solutions are given by assigning these elements to the root variables of the query *OSF*-terms.

Proof. Let $\phi' \equiv \phi'(X_1)\&\ldots\&\,\phi'(X_n)\&\,\phi''$ be the solved form of the *OSF*-clause ϕ which is a resolution-normal form of the query $\boldsymbol{R} \equiv r_1(\psi_1)\&\ldots\&\,r_n(\psi_n)$. All variables in ϕ'' are different from the ones in $\phi'(X_1),\ldots,\phi'(X_n)$ (and existentially quantified).

Since ϕ' is the solved form of a conjunction of $\phi(\psi_1),\ldots,\phi(\psi_n)$ and other *OSF*-clauses (added successively as conjunctions by the resolution procedure), it is clear that the answer constraint ϕ' implies the query constraint $\phi_{\boldsymbol{R}} \equiv \phi(\psi_1),\ldots,\phi(\psi_n)$. By applying Theorem 5 one infers that there exists an *OSF*-graph algebra homomorphism $\gamma_0 : \mathscr{G} \mapsto \mathscr{G}$ mapping the graph representing (uniquely) the query constraint on the graph representing (uniquely) the answer constraint, *i.e.*, $\gamma(G(\phi_{\boldsymbol{R}})) = G(\phi')$. Since $G(\phi(\psi_i)) = G(\psi_i)$, this and the homomorphism property imply that $\gamma_0(G(\psi_1)) = G_1,\ldots,\gamma_0(G(\psi_n)) = G_n$ where we set

$$G_1 \equiv G(\phi'(X_1)),\ldots,G_n \equiv G(\phi'(X_n)).$$

That is, definite clause resolution computes an endomorphic refinement γ_0 of the query arguments, which is the first statement of the proposition.

From Corollary 4 follows that a valuation α with $\alpha(X_i) = G(\phi'(X_i))$ is a principal solution of ϕ'. Note that, since $\mathcal{G}$ is a canonical *OSF*-algebra, ϕ'' is always satisfiable in $\mathcal{G}$ (!). □

Corollary 9. *The solutions of a query in the OSF-algebra $\mathcal{A}$ are exactly the images of the OSF-graphs which represent the query OSF-terms, under the homomorphisms $\gamma \circ \gamma_0$ obtained by composing a homomorphism $\gamma : \mathcal{G} \mapsto \mathcal{A}$ with a homomorphism γ_0 from a derivation sequence as in Theorem* 10.

Proof. This follows directly from Theorem 5. If $\mathcal{A}$, $\alpha \models \phi'$—and thus, by the soundness of the resolution procedure, $\mathcal{A}$, $\alpha \models r_1(\psi_1) \& \dots \& r_n(\psi_n)$,—then there exists a homomorphism $\gamma : \mathcal{G} \mapsto \mathcal{A}$ such that:

$$\alpha(X_1) = \gamma(G_1), \dots, \alpha(X_n) = \gamma(G_n),$$

and the converse holds as well; namely, every *OSF*-homomorphism $\mathcal{G} \mapsto \mathcal{A}$ which is defined on all of:

$$G_1, \dots, G_n$$

defines a solution α in this way, and therefore,

$$\{\langle \gamma(G_1), \dots, \gamma(G_n) \rangle | \gamma : \mathcal{G} \mapsto \mathcal{A}\} \subseteq \{\langle d_1, \dots, d_n \rangle | r_1^{\mathcal{A}}(d_1), \dots, r_n^{\mathcal{A}}(d_n)\}.$$

In other words,

$$\{\langle (\gamma \circ \gamma_0)(G(\psi_1)), \dots, (\gamma \circ \gamma_0)(G(\psi_n)) \rangle | \gamma : \mathcal{G} \mapsto \mathcal{A}\} \subseteq r_1^{\mathcal{A}} \times \dots \times r_n^{\mathcal{A}}. \quad \square$$

That is, definite clause resolution computes an endomorphic refinement γ_0 of the query arguments. *Any* further refinement of this graph "instantiation" through a homomorphism γ into an *OSF*-algebra $\mathcal{A}$, model of the program, yields elements $d_1, \dots, d_n$ in the relations (of $\mathcal{A}$) denoted by the query predicates as directed by the definite clauses defining the predicates of the program.

In particular, *if* the homomorphism $\gamma \circ \gamma_0$ from the subalgebra generated by the query *OSF*-graphs into the *OSF*-algebra $\mathcal{A}$ can be defined, then the query has a solution in $\mathcal{A}$.

This leads to an essential difference between query languages over first-order terms (such as PROLOG) and LIFE, a query language over *OSF*-terms: In the first case, an answer of a query states the existence of solutions in the initial algebra and, thus, in *all* models of the program. In the second case, however, an answer of a query over *OSF*-terms states the existence of solutions in the (weakly) *final* algebra $\mathcal{G}$ of *OSF*-graphs only.

4. CONCLUSION

There are many benefits to seeing LIFE's constraints algebraically, especially if the view is in complete and natural coincidence with logical and implementational views. One nice outcome of this approach is our understanding that sorted and labeled graph-structures used in our implementation of LIFE's approximation structures form a particularly useful *OSF*-algebra which happens to be a canonical interpretation (in the sense of Herbrand) for the satisfiability of *OSF*-clauses. This is important as there is no obvious initiality result, our setting having no values but only approximations. Indeed, approximation chains in *OSF*-algebras can very well be infinitely strictly ascending (getting better and better...), and this is the case of our version presented here—*all* approximation chains are non Noetherian! We do not care, as only "interesting" approximations, in a sense to be made rigorously precise, are of any use in LIFE.

With this generalizing insight, we can give a crisp interpretation of Life's approximation structures as principal filters in *OSF*-interpretations for the information-theoretic approximation ordering ($\sqsubseteq$) derived from the existence of (*OSF*-)endomorphisms. Thereby, they may inherit a wealth of lattice-theoretic results such as that of being closed under *joins* ($\sqcup$), or equivalently, set-intersection ($\cap$) in the type interpretation (Ψ) with the inclusion ordering ($\subseteq$), conjunction (&) in the logical interpretation (Φ) with the implication ordering ($\succeq$), and graph-unification ($\wedge$) in the canonical (graph) interpretation with the (graph) approximation ordering.

The work we have reported is a step towards a complete semantics of LIFE as suggested by this article's title. A full constraint language for LIFE has not been given here. We have merely laid the formal foundations for computing with partial knowledge in the form of approximations expressed as relational, functional, or type constraints, and explored their syntactic and semantic properties as type-theoretic, logical, and algebraic formulations. We have made explicit that these are in mutual correspondence in the clearest possible way and thence reconciled many common and apparently different formal views of multiple inheritance. A full meaning of LIFE is being dutifully completed by us authors in terms of the foundations cast here and to be reported soon. That includes functional beings, daemons, and many other unusual LIFE forms [8, 12, 11]. Finally, we must mention that quite a decent C implementation of a LIFE interpreter for experimentation has been realized by Richard Meyer, and further completed and extended by Peter Van Roy. It is called *Wild_LIFE* [4], and is in the process of being released as public domain software by Digital's Paris Research Laboratory. We hope to share it soon with the programming community at large so that LIFE may benefit from the popular wisdom of real life users, and hopefully contribute a few effective conveniences to computer programming, then perhaps evolve into *Real_LIFE*.

APPENDIX

A. THE HÖHFELD-SMOLKA SCHEME

Recently, Höhfeld and Smolka [15] proposed a refinement of the Jaffar–Lassez's scheme [16]. It is more general than the original Jaffar–Lassez scheme in that it abstracts from the syntax of constraint formulae and relaxes some technical demands on the constraint language—in particular, the somewhat baffling "solution-compactness" requirement.

The Höhfeld–Smolka constraint logic programming scheme requires a set $\mathcal{R}$ of *relational symbols* (or, predicate symbols) and a *constraint language* $\mathcal{L}$. It needs very few assumptions about the language $\mathcal{L}$, which must only be characterized by:

- $\mathcal{V}$, a countably infinite set of *variables* (denoted as capitalized $X, Y, \ldots$);
- Φ, a set of *formulae* (denoted $\phi, \phi', \ldots$) called *constraints*.
- a function *Var*: $\Phi \mapsto \mathcal{V}$, which assigns to every constraint ϕ the set $Var(\phi)$ of *variables constrained by* ϕ;

- a class of "admissible" *interpretations* $\mathcal{A}$ over some domain $D^{\mathcal{A}}$;
- the set $Val(\mathcal{A})$ of ($\mathcal{A}$-)*valuations, i.e.*, total functions, $\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}$.

Thus, $\mathcal{L}$ is not restricted to any specific syntax, *a priori*. Furthermore, nothing is presumed about any specific method for proving whether a constraint holds in a given interpretation $\mathcal{A}$ under a given valuation α. Instead, we simply assume given, for each admissible interpretation $\mathcal{A}$, a function $[\![\_]\!]^{\mathcal{A}} : \Phi \mapsto 2^{(Val(\mathcal{A}))}$ which assigns to a constraint $\phi \in \Phi$ the set $[\![\phi]\!]^{\mathcal{A}}$ of valuations which we call the *solutions* of ϕ under $\mathcal{A}$.

Generally, and in our specific case, the constrained variables of a constraint ϕ will correspond to its free variables, and α is a solution of ϕ under the interpretation $\mathcal{A}$ if and only if ϕ holds true in $\mathcal{A}$ once its free variables are given values α. As usual, we shall denote this as "$\mathcal{A}, \alpha \models \phi$."

Then, given $\mathcal{R}$, the set of relational symbols (denoted $r, r_1, \ldots$), and $\mathcal{L}$ as above, the language $\mathcal{R}(\mathcal{L})$ of *relational clauses* extends the constraint language $\mathcal{L}$ as follows. The syntax of $\mathcal{R}(\mathcal{L})$ is defined by:

- the same countably infinite set $\mathcal{V}$ of *variables*;
- the set $\mathcal{R}(\Phi)$ of formulae ρ from $\mathcal{R}(\mathcal{L})$ which includes:
  - all $\mathcal{L}$-constraints, *i.e.*, all formulae ϕ in Φ;
  - all relational atoms $r(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n \in \mathcal{V}$, mutually distinct;

  and is closed under the logical connectives & (conjunction) and $\rightarrow$ (implication); *i.e.*,
  - $\rho_1 \& \rho_2 \in \mathcal{R}(\Phi)$ if $\rho_1, \rho_2 \in \mathcal{R}(\Phi)$;
  - $\rho_1 \rightarrow \rho_2 \in \mathcal{R}(\Phi)$ if $\rho_1, \rho_2 \in \mathcal{R}(\Phi)$;
- the function $Var : \mathcal{R}(\Phi) \mapsto \mathcal{V}$ extending the one on Φ in order to assign to every formula ρ the set $Var(\rho)$ of the *variables constrained by* ρ:
  - $Var(r(X_1, \ldots, X_n)) = \{X_1, \ldots, X_n\}$;
  - $Var(\rho_1 \& \rho_2) = Var(\rho_1) \cup Var(\rho_2)$;
  - $Var(\rho_1 \rightarrow \rho_2) = Var(\rho_1) \cup Var(\rho_2)$;
- the class of "admissible" *interpretations* $\mathcal{A}$ over some domain $D^{\mathcal{A}}$ such that $\mathcal{A}$ extends an admissible interpretation $\mathcal{A}_0$ of $\mathcal{L}$, over the domain $D^{\mathcal{A}} = D^{\mathcal{A}_0}$ by adding relations $r^{\mathcal{A}} \subseteq D^{\mathcal{A}} \times \ldots \times D^{\mathcal{A}}$ for each $r \in \mathcal{R}$;
- the same set $Val(\mathcal{A})$ of *valuations* $\alpha : \mathcal{V} \mapsto D^{\mathcal{A}}$.

Again, for each interpretation $\mathcal{A}$ admissible for $\mathcal{R}(\mathcal{L})$, the function $[\![\_]\!]^{\mathcal{A}} : \mathcal{R}(\Phi) \mapsto 2^{(Val(\mathcal{A}))}$ assigns to a formula $\rho \in \mathcal{R}(\Phi)$ the set $[\![\rho]\!]^{\mathcal{A}}$ of valuations which we call the *solutions* of ρ under $\mathcal{A}$. It is defined to extend the interpretation of constraint formulae in $\Phi \subseteq \mathcal{R}(\Phi)$ inductively as follows:

- $[\![r(X_1, \ldots, X_n)]\!]^{\mathcal{A}} = \{\alpha \mid \langle \alpha(X_1), \ldots, \alpha(X_n) \rangle \in r^{\mathcal{A}}\}$;
- $[\![\rho_1 \& \rho_2]\!]^{\mathcal{A}} = [\![\rho_1]\!]^{\mathcal{A}} \cap [\![\rho_2]\!]^{\mathcal{A}}$;
- $[\![\rho_1 \rightarrow \rho_2]\!]^{\mathcal{A}} = (Val(\mathcal{A}) - [\![\rho_1]\!]^{\mathcal{A}}) \cup [\![\rho_2]\!]^{\mathcal{A}}$.

Note that an $\mathcal{L}$-interpretation $\mathcal{A}_0$ corresponds to an $\mathcal{R}(\mathcal{L})$-interpretation $\mathcal{A}$, namely where $r^{\mathcal{A}_0} = \emptyset$ for every $r \in \mathcal{R}$.

As in Prolog, we shall limit ourselves to *definite relational clauses* in $\mathcal{R}(\mathcal{L})$ that we shall write in the form:

$$r(\vec{X}) \leftarrow r_1(\vec{X}_1) \& \ldots \& r_m(\vec{X}_m) \& \phi,$$

$(0 \leq m)$, making its constituents more conspicuous and also to be closer to 'standard' Logic Programming notation, where:

- $r(\vec{X}), r_1(\vec{X}_1), \ldots, r_m(\vec{X}_m)$ are relational atoms in $\mathcal{R}(\mathcal{L})$; and,
- ϕ is a conjunction of constraint formulae in $\mathcal{L}$.

Given a set $\mathscr{C}$ of definite $\mathcal{R}(\mathcal{L})$-clauses, a *model* of $\mathscr{C}$ is an $\mathcal{R}(\mathcal{L})$-interpretation such that every valuation $\alpha : \mathscr{V} \mapsto D^{\mathcal{M}}$ is a solution of every formula ρ in $\mathscr{C}$, *i.e.*, $[\![\rho]\!]^{\mathcal{M}} = Val(\mathcal{M})$. It is a fact established in [15] that any $\mathcal{L}$-interpretation $\mathcal{A}$ can be extended to a *minimal model* $\mathcal{M}$ of $\mathscr{C}$. Here, minimality means that the added relational structure extending $\mathcal{A}$ is minimal in the sense that if $\mathcal{M}'$ is another model of $\mathscr{C}$, then $r^{\mathcal{M}} \subseteq r^{\mathcal{M}'} (\subseteq D^{\mathcal{A}} \times \ldots \times D^{\mathcal{A}})$ for all $r \in \mathcal{R}$.

Also established in [15], is a fixed-point construction. The minimal model $\mathcal{M}$ of $\mathscr{C}$ extending the $\mathcal{L}$-interpretation $\mathcal{A}$ can be constructed as the limit $\mathcal{M} = \bigcup_{i \geq 0} \mathcal{A}_i$ of a sequence of $\mathcal{R}(\mathcal{L})$-interpretations $\mathcal{A}_i$ as follows. For all $r \in \mathcal{R}$ we set:

$$r^{\mathcal{A}_0} = \emptyset;$$

$$r^{\mathcal{A}_{i+1}} = \{\langle \alpha(x_1), \ldots, \alpha(x_n) \rangle | \alpha \in [\![\rho]\!]^{\mathcal{A}_i}; r(x_1, \ldots, x_n) \leftarrow \rho \in \mathscr{C}\};$$

$$r^{\mathcal{M}} = \bigcup_{i \geq 0} r_i^{\mathcal{A}}.$$

A *resolvent* is a formula of the form $\rho \,[\!] \, \phi$, where ρ is a possibly empty conjunction of relational atoms $r(X_1, \ldots, X_n)$ (its *relational part*) and ϕ is a possibly empty conjunction of $\mathcal{L}$-constraints (its *constraint part*). The symbol $[\!]$ is in fact just the symbol & in disguise. It is simply used to emphasize which part is which. (As usual, an empty conjunction is assimilated to *true*, the formula which takes all arbitrary valuations as solution.)

Finally, the Höhfeld–Smolka scheme defines constrained *resolution* as a reduction rule on resolvents which gives a sound and complete interpreter for *programs* consisting of a set $\mathscr{C}$ of definite $\mathcal{R}(\mathcal{L})$-clauses. The reduction of a *resolvent* R of the form:

- $B_1 \& \ldots \& r(X_1, \ldots, X_n) \& \ldots B_k [\!] \phi$

by the (renamed) program clause:

- $r(X_1, \ldots, X_n) \leftarrow A_1 \& \ldots \& A_m \& \phi'$

is the new resolvent R' of the form:

- $B_1 \& \ldots \& A_1 \& \ldots \& A_m \& \ldots B_k [\!] \phi \& \phi'$.

The soundness of this rule is clear: under every interpretation $\mathcal{A}$ and every evaluation such that R holds, then so does R', *i.e.*, $[\![R']\!]^{\mathcal{A}} \subseteq [\![R]\!]^{\mathcal{A}}$. It is also not difficult to prove its completeness: if $\mathcal{M}$ is a minimal model of $\mathscr{C}$, and $\alpha \in [\![R]\!]^{\mathcal{M}}$ is a solution of the formula R in $\mathcal{M}$, then there exists a sequence of reductions of (the $\mathcal{R}(\mathcal{L})$-formula) R to an $\mathcal{L}$-constraint ϕ such that $\alpha \in [\![\phi]\!]^{\mathcal{M}}$.

B. DISJUNCTIVE OSF TERMS

A technicality arises if $\mathcal{S}$ is not a lower semi-lattice. For example, given the (non-lattice) set of sorts of Figure 5, the GLB of *student* and *employee* is not uniquely defined, in that it could be *john* or *mary*. That is, the set of their common lower bounds does not admit *one* greatest element. However, the set of their *maximal* common lower bounds offers the most general choice of candidates. Clearly, the *disjunctive* type {*john*; *mary*} is an adequate interpretation. In this way, the *OSF*-term syntax may be enriched with disjunction denoting type union.

Informally a *disjunctive OSF*-term is a set of *OSF*-terms, written $\{t_1; \ldots; t_n\}$ where the t_i's are *OSF*-terms. The subsumption ordering is extended to disjunctive (sets of) *OSF*-terms such that $D_1 \leq D_2$ *iff* $\forall t_1 \in D_1$, $\exists t_2 \in D_2$ such that $t_1 \leq t_2$. This informally justifies the convention that a singleton $\{t\}$ is the same as t, and that the empty set is identified with $\perp$. Unification of two disjunctive *OSF*-terms consists in the enumeration of the set of all maximal *OSF*-terms obtained from unification of all elements of one with all elements of the other. For example, limiting ourselves to disjunctions of atomic *OSF*-terms in the context of signature in Figure 3, the unification of {*employee*; *student*} with {*faculty*; *staff*} is {*faculty*; *staff*}. It is the set of maximal elements of the set {*faculty*; *staff*; $\perp$; *workstudy*} of pairwise GLB's. In practice, it is convenient and more effective to allow nesting disjunctions in the structure of *OSF*-terms.

Formally, the syntax of a disjunctive *OSF*-term is:

$$X : \{t_1; \ldots; t_n\}$$

where $X \in \mathcal{V}$, the t_i's are (possibly disjunctive) *OSF*-terms, and $n \geq 0$. Again, where X is not shared in the context, it may be omitted and not written explicitly.

Example 6. In order to describe a person whose friend may be an astronaut with same first name, or a businessman with same last name, or a charlatan with first and last names inverted, we may write such expressions as:

$$
\begin{aligned}
&\textit{person}(\textit{id} \Rightarrow \textit{name}(\textit{first} \Rightarrow X : \textit{string},\\
&\qquad\qquad\qquad\quad \textit{last} \Rightarrow Y : \textit{string}),\\
&\quad \textit{friend} \Rightarrow \{\textit{astronaut}(\textit{id} \Rightarrow \textit{name}(\textit{first} \Rightarrow X))\\
&\qquad\qquad ; \textit{businessman}(\textit{id} \Rightarrow \textit{name}(\textit{last} \Rightarrow Y))\\
&\qquad\qquad ; \textit{charlatan}(\textit{id} \Rightarrow \textit{name}(\textit{first} \Rightarrow Y,\\
&\qquad\qquad\qquad\qquad\qquad\qquad \textit{last} \Rightarrow X))\}).
\end{aligned}
$$

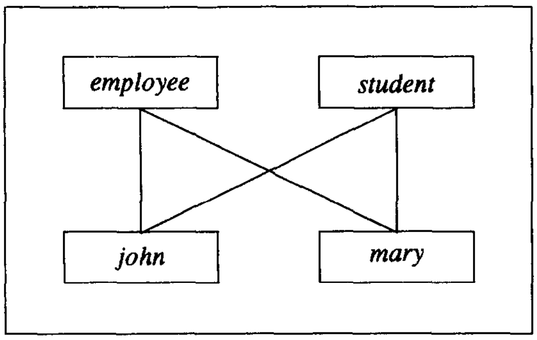

FIGURE 5. Disjunctive-Clause Normalization Rules.

Note that variables may even be chained or circular within disjunctions as in:

person(*partner* ⇒ P : {*crook*; F},
    friend ⇒ F : {*artist*; P})

which may be used to describe a person whose partner is a crook or whoever his/her friend is, and whose friend is an artist or whoever his/her partner is. These are no longer graphs but hypergraphs.

The denotation of a disjunctive *OSF*-term in an *OSF*-interpretation $\mathscr{A}$ with variable valuation $\alpha \in Val(\mathscr{A})$ is simply given by:

$$[\![X:\{t_1;\ldots;t_n\}]\!]^{\mathscr{A},\alpha} = \{\alpha(X)\} \cap \bigcup_{i=1}^{n} [\![t_i]\!]^{\mathscr{A},\alpha} \tag{7}$$

and again, as before;

$$[\![t]\!]^{\mathscr{A}} = \bigcup_{\alpha \in Val(\mathscr{A})} [\![t]\!]^{\mathscr{A},\alpha}. \tag{8}$$

Observe that with this definition, our syntactic convention dealing with the degenerate cases that, for $n = 0$, identifies { } with $\perp$, and for $n = 1$, identifies $\{t\}$ with t, is now formally justified on semantic grounds.

Also, note that in Equation (7), the *same* valuation is used in all parts of the union. As a result, for a given α, $[\![t]\!]^{\mathscr{A},\alpha}$ still denotes either the empty set or a singleton, even if t is a non-degenerate disjunctive term. This may appear strange as one would expect that variables in disjuncts that are not shared with the global context be independently valuated. They are, in fact, but thanks to Equation (8), not Equation (7). Taking, for example, $t = \{X : int;\ X : string\}$, where $int^{\mathscr{A}} = \mathbf{Z}$ is the set of all integers and $string^{\mathscr{A}} = \mathbf{S}$ is the set of all finite strings of ASCII characters, with α and β such that $\alpha(X) = 1$ and $\beta(X) =$ "`hello`", then $[\![t]\!]^{\mathscr{A},\alpha} = \{1\} \cup \emptyset = \{1\}$ and $[\![t]\!]^{\mathscr{A},\beta} = \emptyset \cup \{$"`hello`"$\} = \{$"`hello`"$\}$. However, as expected, we do have $[\![t]\!]^{\mathscr{A}} = \mathbf{Z} \cup \mathbf{S}$.

Example 7. The disjunctive term

P:{*charlatan*
  ; *person*(*id* ⇒ *name*(*first* ⇒ X : "*John*",
                  last ⇒ Y : {"Doe"; X}),
      friend ⇒ {P; *person*(*id* ⇒ *name*(*first* ⇒ Y,
                              last ⇒ X))})}

describes either a charlatan, or a person named either "John Doe" or "John John" whose friend is himself, or a person with his first and last names inverted. It does *not* specify that that person's friend is either a charlatan or himself or a person... since it is semantically equivalent to the term:

{*charlatan*
; P : *person*(*id* ⇒ *name*(*first* ⇒ X : "*John*",
                  last ⇒ Y : {"*Doe*"; X}),

(*Bottom Elimination*) $$\frac{\phi \vee X : \bot}{\phi}$$

(*Distributivity*) $$\frac{\phi \,\&\, (\phi_1 \vee \phi_2)}{(\phi \,\&\, \phi_1) \vee (\phi \,\&\, \phi_2)}$$

FIGURE 6. Disjunctive clause normalization rules.

$$\begin{aligned} &\mathit{friend} \Rightarrow \{P;\ \mathit{person}(\mathit{id} \Rightarrow \mathit{name}(\mathit{first} \Rightarrow Y,\\ &\qquad\qquad\qquad\qquad \mathit{last} \Rightarrow X))\})\}. \end{aligned}$$

Similarly, *OSF*-clauses are extended to be possibly disjunctive as well. Hence, an *OSF*-clause is now, either of the following forms:

- $X : s$
- $X \doteq Y$
- $X.\ell \doteq Y$
- $\phi_1 \,\&\, \phi_2$
- $\phi_1 \vee \phi_2$

where ϕ_1, ϕ_2 are *OSF*-clauses.

The interpretation of atomic and conjunctive *OSF*-constraints is as before; and as for disjunctions, we have simply:

$$\mathscr{A}, \alpha \models \phi \vee \phi' \quad \text{iff} \quad \mathscr{A}, \alpha \models \phi \text{ or } \mathscr{A}, \alpha \models \phi'.$$

Converting from *OSF*-terms to *OSF*-clauses is done by extending the dissolution mapping ϕ to be:

$$\phi(X : \{t_1; \ldots ; t_n\}) = (X \doteq Root(t_1) \,\&\, \phi(t_1)) \vee \ldots \vee (X \doteq Root(t_n) \,\&\, \phi(t_n)).$$

Example 8. Let us reconsider the second term of Example 6 again. Namely, writing explicitly all omitted (since unshared) variables:

$$\begin{aligned} t = X : \mathit{person}(&\mathit{partner} \Rightarrow P : \{C : \mathit{crook};\ F\},\\ &\mathit{friend} \Rightarrow F : \{A : \mathit{artist};\ P\}) \end{aligned}$$

its dissolved form is:

$$\begin{aligned} \phi(t) = X : \mathit{person} \,\&\, X.\mathit{partner} \doteq P \,\&\, ((P \doteq C \,\&\, C : \mathit{crook}) \vee P \doteq F)\\ \&\, \mathrm{X.friend} \doteq \mathrm{F} \,\&\, ((\mathrm{F} \doteq \mathrm{A} \,\&\, \mathrm{A} : \mathrm{artist}) \vee \mathrm{F} \doteq \mathrm{P}). \end{aligned}$$

Finally, the *OSF*-clause normalization rules are also extended with two additional ones shown in (making the similar associativity and commutativity conventions for $\vee$ that we did for &), and we leave it as an exercise to the reader to show that these two rules together with the four rules shown in Figure 4 enjoy a straightforward extension of Theorem 1. Namely,

Theorem 11 (Disjunctive *OSF*-Clause Normalization). *The six OSF-clause normalization rules of Figures 4 and 5 are solution-preserving, finite terminating, and confluent (modulo variable renaming). Furthermore, they always result in a normal form that is either the inconsistent clause or a disjunction of conjunctive OSF-clauses in solved form with associated conjunctions of equality constraints.*

Note that the normalization rules of Figure 5 contribute essentially to putting the dissolved form in disjunctive normal form. In particular, they do not eliminate disjuncts that are subsumed by other disjuncts in the same disjunction. In the following example, the second and third disjuncts are subsumed by the fourth and are therefore non-principal solutions. Only the first and fourth disjuncts are principal solutions.

Example 9. Normalizing the dissolved form of Example 8, we obtain a disjunction of four conjunctions:

$$
\begin{aligned}
((X: \mathit{person}\ \&\ X.\mathit{partner} \doteq P \& P: \mathit{crook}\ \&\ P \doteq C \\
\&X.\mathit{friend} \doteq F\ \&\ F: \mathit{artist}\ \&\ F \doteq A) \\
\vee (X: \mathit{person}\ \&\ X.\mathit{partner} \doteq P\ \&\ P: \mathit{artist}\ \&\ P \doteq A \\
\&X.\mathit{friend} \doteq P\ \&\ P \doteq F) \\
\vee (X: \mathit{person}\ \&\ X.\mathit{partner} \doteq P\ \&\ P: \mathit{crook}\ \&\ P \doteq C \\
\&X.\mathit{friend} \doteq P\ \&\ P \doteq F) \\
\vee (X: \mathit{person}\ \&\ X.\mathit{partner} \doteq P \\
\&X.\mathit{friend} \doteq P\ \&\ P \doteq F)).
\end{aligned}
$$

ACKNOWLEDGEMENTS

We acknowledge first and foremost Gert Smolka for his enlightening work on feature logic and for mind-opening discussions. He pointed out that ψ-terms were solved formulae and he also came up, in conjunction with work by Jochen Dörre and Bill Rounds, with the notion of feature algebras. Bill Rounds has also been a source of great inspiration. In essence, our quest for the meaning of LIFE has put their ideas and ours together. To these friends, we owe a large part of our understanding.

We also wish to express our thanks to Kathleen Milsted and Jean-Christophe Patat for precious help kindly proofreading the penultimate version of the manuscript.

Each of us authors has enjoyed tremendously the excitement of seeking together a meaning for LIFE and, of course, each shamelessly blames the other for all remaining mistakes...

REFERENCES

1. Hassan Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Types.* PhD thesis, University of Pennsylvania, Philadelphia, PA (1984).

2. Hassan Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351 (1986).
3. Hassan Aït-Kaci and Jacques Garrigue. Label-selective λ-calculus. PRL Research Report (forthcoming), Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison (1993).
4. Hassan Aït-Kaci, Richard Meyer, and Peter Van Roy. Wild_LIFE, a user manual. PRL Technical Report (forthcoming), Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France (1993).
5. Hassan Aït-Kaci and Roger Nasr. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215 (1986).
6. Hassan Aït-Kaci and Roger Nasr. Integrating logic and functional programming. *Lisp and Symbolic Computation*, 2:51–89 (1989).
7. Hassan Aït-Kaci, Roger Nasr, and Patrick Lincoln. Le Fun: Logic, equations, and Functions. In *Proceedings of the Symposium on Logic Programming* (*San Francisco, CA*), pages 17–23, Washington, DC (1987). IEEE, Computer Society Press.
8. 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, November 1992).
9. 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).
10. 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).
11. Hassan Aït-Kaci, Andreas Podelski, and Seth Copen Goldstein. Order-sorted feature theory unification. PRL Research Report (forthcoming), Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France (1993).
12. 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.
13. William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, Germany, 2nd edition (1984).
14. Jochen Dörre and William C. Rounds. On subsumption and semiunification in feature algebras. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, (*Philadelphia, PA*), pages 301–310, Washington, DC (1990). IEEE, Computer Society Press.
15. Markus Höhfeld and Gert Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Stuttgart, Germany (October 1988).
16. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, Munich, W. Germany (January 1987).
17. Richard O'Keefe. *The Craft of Prolog*. MIT Press, Cambridge, MA (1990).
18. Gert Smolka. A feature logic with subsorts. LILOG Report 33, IWBS, IBM Deutschland, Stuttgart, Germany (May 1988).
19. Gert Smolka. Feature constraint logic for unification grammar. *Journal of Logic Programming*, 12:51–87 (1992).