JOURNAL OF COMPUTER AND SYSTEM SCIENCES 42, 139–185 (1991)

# Kernel-LEAF: A Logic plus Functional Language

ELIO GIOVANNETTI*

*CSELT, via Reiss Romoli 274, 10148 Torino, Italy*

GIORGIO LEVI

*Dipartimento di Informatica, Università di Pisa, corso Italia 40, 56100 Pisa, Italy*

CORRADO MOISO

*CSELT, via Reiss Romoli 274, 10148 Torino, Italy*

AND

CATUSCIA PALAMIDESSI

*Dipartimento di Informatica, Università di Pisa, corso Italia 40, 56100 Pisa, Italy*

Received October 13, 1987; revised July 14, 1989

Kernel-LEAF is a logic plus functional language based on the flattening technique. It differs from other similar languages because it is able to cope with partial (undefined or non-terminating) functions. This is achieved by introducing the distinction between data structures and (functional) term structures, and by using two kinds of equality. The language has a clean model-theoretic semantics, where the domains of the interpretations are the algebraic CPOs. In these domains the difference between the two equalities corresponds to a different behaviour with respect to continuity. The operational semantics (based on SLD-resolution) is proved sound and complete with respect to the model-theoretic one. Finally, an outermost strategy, more efficient than unrestricted SLD-resolution, but still complete, is presented. © 1991 Academic Press, Inc.

## 1. INTRODUCTION

In the last years several efforts were devoted to the problem of integrating the most promising classes of declarative languages, i.e., the logic and the functional languages. Several different approaches have been proposed. For a discussion of the various approaches, see [5, 6, 41].

* Current address: Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy.

If the functional part is constrained to be first-order, the integration between functional programming and logic programming can be achieved in a purely logical framework, namely, Horn clause logic with equality. Since function definitions are basically *directed* equalities, i.e., rewrite rules, some important results available in the rewriting system domain can be exploited, possibly via some extensions, in the design of the integrated language.

The language we consider is a refinement and an evolution of LEAF [2, 3]; it is one example where the integration is based on extending a relational logic language (pure Horn clause logic) with equality. Other examples are the proposal in [39], FUNLOG [43], and EQLOG [25, 26]. A nice semantic characterization of complete logic programs with equality is given in [31].

The functional component (defined by equations) is given an operational semantics based on unification, which exhibits the same properties of standard logic programs, i.e., logical variables, search, partially determined data structures, and function invertibility. We can then consider the functional component separately as a logic language with functional notation. Examples are the language in [40], the language in [42], and SLOG [21, 22].

An other approach for the operational semantics of the functional component is to use *narrowing* [20, 29]. Narrowing a functional expression is applying to it the minimum substitution such that the resulting expression is reducible and then reduce it. The substitution is found by unifying the expression with the left-hand sides of equations. In general there will be several narrowings for an expression, one for each equation whose left-hand side is unifiable with the expression. An expression (or an equation) containing existentially quantified variables (logical variables) can be narrowed, possibly yielding a set of narrowing substitutions (answers). This is essentially the same situation of query evaluation in relational logic programs. Narrowing can be implemented as a proper extension of reduction, and is (the base for) a complete goal-solving algorithm for canonical, i.e., confluent and terminating, equational theories.

Some of the above-mentioned languages are based on some form of narrowing. This is the case of EQLOG [25, 26], where narrowing is applied to confuent and terminating equational theories, and of FUNLOG [43], whose semantic unification is essentially narrowing, of the language proposed by Reddy [40], based on equational theories with constructors.

Narrowing, however, is not the only inference rule which can be used for logic languages based on equational theories. A different approach, first suggested in LEAF [2], is based on the *transformation* of equations to *flat form* and on a SLD-resolution inference rule, which is applied to "flat" clauses, where function composition is eliminated and replaced by the logical operator *and*. Equations in flat form are used in SLOG [21, 22], which, however, uses a different inference system (the flattening technique was first introduced, in the theorem-proving domain, by [14]; more recently, it has been used in Surface Deduction [17]). SLD-resolution on flat equations has the advantage of being based on a (single) computational model, the one of Prolog, that is now quite standard and efficiently implemented. Another relevant advantage of flattening with respect to narrowing is the possibility to deal with conditional equations, without need of extensions.

The need of coping with infinite data structures and partial functions, which are typical functional programming concepts, leads to the introduction of two notions of equality, which can be characterized by two sets of Horn clauses. Strict-equality is only defined on finite and completely determined data structures. The other (non-strict) equality is defined on partially determined or infinite data structures.

Partial and infinite terms lead to the need of considering complete partial orders as interpretation domains. From the semantics viewpoint, the resulting logic language is therefore more complex than those definable by the general construction proposed in [32, 33], where extensions are essentially modeled by defining equivalence classes on the flat Herbrand Universe.

The paper is organized as follows. In Section 2 the basic design choices are explicitly motivated. Sections 3, 4, and 5 formally present the syntax of the language, its operational and its declarative semantics. Section 6 states the soundness and completeness theorems, which establish the equivalence between the operational and the declarative semantics. Finally, Section 7 presents an efficient computational model based on an outermost strategy. The reader is assumed to be familiar with the theory of logic programming [1, 36] and with the rewriting systems [8, 38]. (A short version of this paper has been presented in [35].)

## 2. The Rationale

### 2.1. *Logic plus Functional Languages, Equational Systems, and Properties of Narrowing*

As already mentioned, the functional component of a logic plus functional (L + F) language can be based on equational systems and narrowing can be used as the equation-solving algorithm. Narrowing algorithms, however, are complete only for equational systems satisfying suitable conditions. We will then first characterize the properties of equational systems which are adequate to model a sufficiently powerful L + F language and then we look into the problems of narrowing completeness.

The first relevant feature is related to termination. Since we want the language to be able to cope with infinite data structures and partial functions, we have to deal with (possibly) *nonterminating equational systems*.

An L + F integrated language, based on Horn clause logic with equality, should allow us to combine equational atoms and ordinary predicates (relational atoms). This can be achieved by *conditional equational systems*, where predicates are simply considered as boolean functions (i.e., a relational atom $p$ is read as a (directed) equation $p = true$). The pure logic (relational) component could thus be handled by the same computational mechanism used for the functional component. Note that, if conditional equations are used to represent definite Horn clauses, the *no extra-variables* condition [37] (all the vaiables occurring in the right-hand side (rhs) of the conditional equations must occur in the left-hand side (lhs)) does not hold in the general case.

We can now discuss the problem of narrowing completeness. A consequence of Hullot's proof [29] of the completeness of narrowing as equation-solving algorithm for canonical (i.e., confluent and terminating) equational systems, is that termination only plays a role in finding non-normalizable solutions. Therefore, if one is only interested in normal-form solutions, narrowing is complete for confluent theories even in absence of termination. However, if the system is not terminating, confluence cannot be reduced to local confluence. Hence the completion methods based on rule superposition cannot be exploited. Nevertheless, there is an important class of rewrite systems where confluence is ensured by syntactical constraints. Two conditions are sufficient: (1) *weak non-ambiguity*, i.e., 1-step local confluence; and (2) *left-linearity*, i.e., in the lhs of a rule each variable occurs once (see, for example, [28]). These conditions are usually satisfied in the framework of a L + F programming language, where the rules are function definitions in the ordinary deterministic sense. Narrowing is therefore complete for these systems with respect to normalizable solutions, thus allowing partial functions and infinite data structures.

In order to define an interpreter for an integrated L + F language, we must consider conditional equations. In [37] *conditional narrowing* is proved complete for *canonical conditional systems*, under the assumption that no *extra-variables* occur. For confluent but possibly non-terminating systems completeness only holds, once again, with respect to normalized solutions. Note that, in the case of relational definitions, weak non-ambiguity is always guaranteed. In fact, since they have the form $p = true \leftarrow rhs$, critical pairs can always be reduced in one step to the constant *true*.

If, however, extra-variables are present in the rhs's, confluence is no longer sufficient to guarantee completeness, even in presence of termination [10]. The slightly stronger notion of *level-confluence*, which is still implied by weak non-ambiguity and by left linearity, can be profitably introduced. Conditional narrowing is complete if the system is terminating and level-confluent [10]. However, in the case of level-confluent nonterminating systems, conditional narrowing may fail to find even normalizable solutions. For example, given the system

$$f(a) = c(f(a))$$

$$g(a) = b \leftarrow x = c(x)$$

(where $x$ is a variable). The solution (in normal form) $\{z := b\}$ of the equation $g(a) = z$ (where $z$ is a variable) cannot be found, because in order to rewrite $g(a)$ into $b$ one should solve the equation $x = c(x)$, whose solution $f(a)$, which is not normalizable and instantiates the two sides of the equation to the non-normalizable terms $f(a)$ and $c(f(a))$, cannot be computed by narrowing. In summary, there exists no syntactical characterization of nonterminating conditional equational systems with extra variables for which conditional narrowing is complete (with respect to normalizable solutions).

## 2.2. *Theories with Constructors and $III_n$ Systems*

An important design choice is the restriction to *equational theories with constructors*, where the distinction is made between functions and data constructors. This allows us to characterize theories which essentially define *functional programs*, ruling out equations that could better be viewed as program properties. For example, if *nil* and *cons* are considered data constructors, then the equations

$$append(nil, x) = x$$

$$append(cons(x, y), z) = cons(x, append(y, z)),$$

are function definitions, since the left parts contain only one (outermost) function symbol. On the other side, an equation like

$$append(append(x, y), z) = append(x, append(y, z))$$

would be considered a program property and would not be allowed as program component.

If this distinction between functions and data constructors is present, it is natural to consider only ground data terms, i.e., terms built on (constants and) constructors, as the individuals in the Herbrand models of the language. If the application of a function $f$ to an argument $a$ gives rise to a nonterminating computation we do not want to consider $f(a)$, or any of the functional terms generated in the infinite rewriting sequence of $f(a)$, as the "value" of $f(a)$, therefore the possible loss of $f(a)$ as a solution of some equation corresponds to the intended semantics.

In this framework, the fact that conditional narrowing may not be able to find some non-normalizable solutions, instead of being a drawback, would be a perfectly adequate behaviour. On the contrary, the algorithm exhibits the opposite flaw: it still produces undesired "non-data" solutions. For example the equation $c(f(a)) = c(x)$ would be solved by unification producing $\{x := f(a)\}$. Therefore, if soundness with respect to the intended semantics has to hold, the narrowing (syntactical) unification has to be allowed between data terms only, and forbidden between "true" functional terms. Also the narrowing final step, consisting in the syntactical unification between the two sides of the equation, has to verify the same constraint. This can be achieved by considering the equality symbol in the goal (and in the bodies of program clauses) as an ordinary predicate represented by a new symbol $\equiv$, whose definition is given by a clause

$$d(x_1, ..., x_m) \equiv d(y_1, ..., y_m) \leftarrow x_1 \equiv y_1, ..., x_m \equiv y_m \qquad (m \geqslant 0)$$

for each ($m$-adic) constructor $d$. This clause, as already noted, should be intended as

$$(d(x_1, ..., x_m), d(y_1, ..., y_m)) = true \leftarrow \equiv (x_1, y_1) = true, ..., \equiv (x_m, y_m) = true.$$

The constrained final step can be simulated by (usual) narrowing steps using these clauses.

If function definitions are constrained to be left-linear and weakly non-ambiguous, the language belongs to the the class called $\mathrm{III}_n$ in Bergstra-Klop's classification of rewriting systems [8]. In fact, this class is characterized by i) left-linearity, ii) weak non-ambiguity, and iii) the condition that body literals have the form $t = e$ where $e$ is a ground *unconditional normal form*, i.e., it is a ground term not unifiable with any of the lhs's of the heads. In our case, all the $e$'s simply coincide with the data constant *true*. $\mathrm{III}_n$ rewriting systems are level-confluent, and, moreover, every body equation, having the form $p = \mathit{true}$, or $\equiv (t_1, t_2) = \mathit{true}$, trivially satisfies the property that every solution instantiates the two sides of the equation to normalizable terms. The last property of $\mathrm{III}_n$ systems is clearly related to one of the causes of narrowing incompleteness, mentioned in section 2.1.

The language K-LEAF belongs to the class of $\mathrm{III}_n$ systems and has the already discussed distinction between functions and data constructors. As we will show in the following, the language has a complete proof procedure based on resolution. This procedure is also proved to be equivalent to conditional narrowing, which is therefore complete. K-LEAF is then one syntactical characterization of nonterminating conditional equational systems with extra-variables, for which conditional narrowing is complete.

### 2.3. *Flattening, Resolution, and Equalities*

A narrowing-based proof procedure can be transformed into an SLD-resolution procedure via the introduction of a *flattening* translation phase which was described in the earlier version of LEAF [3]. For every functional nesting $g(f(a))$ the flattening procedure introduces an equation $f(a) = x$ (where $x$ is a new variable) and replaces $f(a)$ with $x$ in $g$. Resolution of a flattened goal using the transformed program is able to exactly mimic narrowing, provided that suitable clauses are added to the program to handle the newly introduced equations. In other words, equalities introduced by flattening (denoted by $=$) have to be dealt with differently from those originally present in the bodies of the clauses and in the goal (denoted by $\equiv$). In particular, the clauses for $=$ must allow the elimination of $f(a) = x$, whenever $f(a)$ would not have been selected by narrowing (i.e., when its value is not required to reduce $g(f(a))$).

### 2.4. *Cpo Semantics vs Standard Semantics*

The semantics we are trying to define also must cope with the important role played by non-strict functions, i.e., functions which may terminate even on inputs from nonterminating functions. They are typically selectors which operate on infinite data structures, as in the following example

$$\mathit{nats}(n) = \mathit{cons}(n, \mathit{nats}(s(n)))$$

$$\mathit{first}(\mathit{cons}(x, l)) = x.$$

By composing the selector *first* with the generator *nats* we can write, for example, a term *first*(*nats*(*0*)) whose value is a finite data structure, i.e., *0*. Now if the compositional character of meaning has to be preserved, also non-normalizable terms, like *nats*(*0*), which may occur as subterms within normalizable expressions, have to be assigned a denotation, which is bound to be the class of all the partial results of the infinite computation along with the usual approximation ordering on them or, equivalently, the infinite data structure defined as the least upper bound of this class. This means that the Herbrand Universe has to be augmented with the indefinite constant $\perp$ (bottom) and with all the other "partial terms," i.e., data terms containing some occurrence of $\perp$, and then completed into an algebraic cpo by adding all the missing lubs, i.e., the infinite objects.

In the intended interpretations, $\equiv$ is the strict equality, i.e., equality between total finite terms, while $=$ is the non-strict equality, which may hold on infinite (or partial) terms also. It should be noted that the relation corresponding to ordinary semidecidable first-order-logic equality is basically $\equiv$. Non-strict equality is not semidecidable. This is not amazing because, even with respect to our new partial-order semantics, non-strict equality may be true in the Herbrand models, without being true in every model. Take, for example, the function definitions

$$f(a) = c(f(a))$$
$$g(a) = c(g(a)).$$

In every Herbrand model, the denotation of both $f(a)$ and $g(a)$ is the infinite term $c(c(c(\cdots)))$, which is the lub of the set of the partial results $\{\perp, c(\perp), c(c(\perp)), c(c(c(\perp))), ...\}$. Therefore, the non-strict equality $f(a) = g(a)$ is true in those models. In an algebraic-cpo model, on the other hand, the denotations of the two terms do not need to be the same. Hence the above equation is not a logical consequence of the previous function definitions. This corresponds to the fact that in the ordinary first-order-logic semantics $f(a)$ and $g(a)$ have two different denotations even in the Herbrand model, namely the two equivalence classes $\{f(a), c(f(a)), c(c(f(a))), ...\}$ and $\{g(a), c(g(a)), c(c(g(a))), ...\}$, which in the partial-order constructor-based semantic collapse into one, via the substitution of the elements $f(a)$ and $g(a)$ with $\perp$. From an algebraic point of view, theories with constructors are a particular case of *data theories*, i.e., algebraic theories based on data constraints. Hence the non-provability of $f(a) = g(a)$ corresponds to the *undecidablity of the word problem* in data theories.

The above argument also shows that in this new "functional-logic" semantics it is no longer true that if a set of clauses has a model, then it has a Herbrand model. For instance, there are algebraic-cpo models, but no Herbrand models, for the above set of clauses $\{f(a) = c(f(a)), g(a) = c(g(a)), \leftarrow f(a) = g(a)\}$. The loss of this fundamental property, however, does not bother us, because in the programming language the occurrence of the two different equality symbols is not free but restricted to perfectly precise roles. For example, a non-strict equation like $f(a) = g(a)$ is not an acceptable goal. Therefore a non-Herbrand satisfiable set of clauses like the one above could never show up. For acceptable sets of clauses, satisfiability always implies Herbrand satisfiability.

The advocated partial-order logic is, in conclusion, not so far from standard Horn clause logic with equality, which is of course restricted to weakly non-ambigous left linear systems. Every goal solution that can be found in the standard framework by a complete inference system, e.g., narrowing, is also valid, and can therefore be found in the new logic, with the exception of equations like $f(a)=g(x)$ for $f(a)$ non-defined or non-terminating. On the other hand, every acceptable goal—and therefore every equality between total finite terms—which is true with respect to the new semantics (and thus derivable) is also true in the standard semantics.

The design decisions emerging from the above argument have led us to the definition of a logic + functional programming system which, while at the user interface level is the same as LEAF [3], i.e., basically Horn clause logic with equality, is endowed with an operational and model-theoretic semantics which are deeply modified with respect to the earlier version, mainly owing to the explicit introduction of the two kinds of equality. A nice and rigorous completeness result can then be achieved, in contrast to most proposals of L + F-integrated languages.

## 3. Syntax of K-LEAF

K-LEAF is a logic + equational programming language, i.e., a language based on Horn clause logic (HCL) with equality. The concrete syntax of K-LEAF is essentially the same as the syntax of LEAF [3]. The term syntax is any *signature with constructors*, i.e., it is based on the distinction between constructors and functions, corresponding to the distinction, found in all the ordinary programming languages, between data structures and algorithms.

The *language alphabet* consists of a set $C$ of data constructor symbols, a set $F$ of function symbols, a set $P$ of predicate symbols, a set $V$ of variable symbols, and the special equality symbols $=$ and $\equiv$. The distinction between data constructor and function symbols leads to the distinction between (*general*) *terms* and *data terms*. A *data term* is:

(i) a variable symbol, or

(ii) a data constructor application $c(d_1, ..., d_n)$, where $c \in C$ and $d_1, ..., d_n$ are data terms.

A *term* is:

(i) a data term, or

(ii) a data constructor application $c(t_1, ..., t_n)$, where $c \in C$ and $t_1, ..., t_n$ are terms, or

(iii) a function application $f(t_1, ..., t_n)$, where $f \in F$ and $t_1, ..., t_n$ are terms. Terms of this form will be called *functional terms*.

The clauses of the language are defined in the usual way, with some constraints on the syntax of atoms. A *head atom* is:

(i) an *equation* $f(d_1, ..., d_n) < t$, where $f \in F$, $d_1, ..., d_n$ are data terms, $t$ is a term, or

(ii) a *relation* $p(d_1, ..., d_n)$, where $p \in P$ and $d_1, ..., d_n$ are data terms.

Moreover, the following two conditions have to be satisfied

(a) multiple occurrences of the same variable in $(d_1, ..., d_n)$ are not allowed (*left-linearity*)

(b) (in the equation case) all the variables occurring in $t$ must also occur in $(d_1, ..., d_n)$ (*definite outputs*).

A *body atom* is:

(i) a *strict equation* $t_1 \equiv t_2$, where $t_1$ and $t_2$ are terms, or

(ii) a *relation* $p(t_1, ..., t_n)$, where $p \in P$ and $t_1, ..., t_n$ are terms.

A *definite clause* is a formula of the form $A \leftarrow B_1, ..., B_n$ $(n \geqslant 0)$ where $A$ is a head atom and $B_1, ..., B_n$ are body atoms. If $n = 0$ then the clause $A \leftarrow$ is called a *unit clause* and is denoted by $A$. A *goal statement* is a formula of the form $\leftarrow B_1, ..., B_n$ $(n \geqslant 0)$, where $B_1, ..., B_n$ are body atoms. If $n = 0$ then the goal is called an *empty goal* and is denoted by $\leftarrow$.

A *program* $W$ is a set of define clauses $\{C_1, ..., C_m\}$ which are *weakly non-ambiguous*, i.e., such that for each pair of equational headers $t' = u'$ and $t'' = u''$, if there exists a replacement $\vartheta$ of variables to data terms (substitution) which makes identical $t'$ and $t''$, then $u'$ and $u''$ are data terms and they are identical under the same replacement.

EXAMPLE 1. The following is a K-LEAF program:

$\{plus(0, x) = x.$

$plus(s(x), y) = s(plus(x, y)).$

$nat(x) = cons(x, nat(s(x))).$

$odd(\ ) = odd1(nat(s(0)).$

$odd1(cons(x, cons(y, z))) = cons(x, odd1(z)).$

$sqrlist(\ ) = cons(0, sqrlist1(0, odd(\ )).$

$sqrlist1((x, cons(y, z)) = cons(plus(x, y), sqrlist1(plus(x, y), z)).$

$sqr(x) = extract(x, sqrlist(\ )).$

$extract(0, cons(x, y)) = x.$

$extract(s(x), cons(y, z)) = extract(x, z).$

$p(x, y, z) \leftarrow sqr(z) \equiv plus(sqr(x), sqr(y)).\}$

This program defines some non-terminating functions, such as *nat* (which computes the infinite list of naturals), *sqrlist* (the list of squares), *odd* (the list of odd numbers); sqr($x$) computes the square of $x$, and the relation $p(x, y, z)$ denotes all the triples $\langle x, y, z \rangle$, such that: $x^2 + y^2 = z^2$. *0*, *s*, and *cons* are data constructors.

Some comments about the definition of the language are in order:

- The definition of the language could be simplified by adding the special data constant *true* and by considering strict equations $t_1 \equiv t_2$ and relations $p(t_1, ..., t_n)$ as special equations, $t_1 \equiv t_2 = true$ and $p(t_1, ..., t_n) = true$, respectively. In this way the only basic construct would be the equation. The intended semantics is the same, and this will be used to simplify the proofs. We prefer to maintain the present notation in order to emphasize the difference between the logical and the functional part of a K-LEAF program.

- The distinction between data terms and terms leads traditionally to restrict the notion of "intended value" of a term. In rewriting languages usually the value of a term $t$ is the term $t'$ in normal form (if any) to which $t$ reduces. Here this notion is enforced by the requirement of $t'$ being a data term (note that a data term in K-LEAF is always in normal form).

- There are two equality symbols, $=$ and $\equiv$, with two different interpretations in the declarative semantics; they can be both defined through a set of Horn clauses (see Section 4.3). Their intuitive meaning is the following:

  $\equiv$, which occurs only in the clause bodies or in the goals, is a *strict-equality*. The atom $t_1 \equiv t_2$ is *true* if $t_1$ and $t_2$ have the same finite "data value," i.e., if they are reducible to the same finite data term. This kind of equality is the most definite one which preserves continuity (see Section 5), and, therefore, computability.

  $=$, which occurs only in the heads of K-LEAF programs, closely resembles the *reducibility* symbol in [44], and the clauses with an equational head can be interpreted as conditional rewrite rules [25, 30]. $=$ is *non-strict*, i.e., it can be true also between terms which have an undefined or an infinite data value. This kind of equality cannot be given a continuous interpretation (see Section 5).

- Weak non-ambiguity, definite outputs, and left linearity conditions (on equational heads) are necessary to ensure that the interpretation of any functional symbol is actually a function. Note that, in rewriting system terminology, the first two conditions do not guarantee the confluence if non-terminating functions are present [28]. Moreover, the left linearity condition (both on equational and relational heads) is necessary because multiple occurrences of the same variable would introduce an implicit non-continuous equality test, thus affecting the continuity of the predicates and the functions definable in the language. In our language every equality test must be realized through the predicate $\equiv$.

- In the user syntax left linearity on relational heads could be dropped, since a clause like $p(x, x) \leftarrow B_1, ..., B_n$ might automatically be transformed by the parser into $p(x, y) \leftarrow x \equiv y, B_1, ..., B_n$. In general, if there are $k+1$ occurrences of $x$ in the arguments of a head, they are replaced by the new variables $x_1, ..., x_k$, and the strict-equalities $x \equiv x_1, ..., x \equiv x_k$ are added to the body. Similar techniques have been introduced in Parlog [16]. This transformation could not apply to the functional heads case, because it could cause the loss of weak non-ambiguity.

## 4. The Flat Form and the Operational Semantics of K-LEAF

The operational semantics of K-LEAF is based on SLD-resolution without the addition of any E-unification algorithm, following the techniques already used in [2, 3, 44] and in agreement with the results of [12]. Function nestings are flattened into conjunctions of atoms, so that functional dependencies are explicitly represented. The flattened form of a K-LEAF program can be considered as a program in a different language on the same alphabet, which we call Flat-LEAF. A procedure will be described which transforms every K-LEAF program into a (semantically equivalent) Flat-LEAF program. Since compositions of functions are not allowed, the execution of Flat-LEAF programs is merely SLD-resolution. The operational semantics of K-LEAF is provided via the flattening procedure and the operational semantics of Flat-LEAF. The last can therefore be considered as an intermediate language to implement K-LEAF, with all the well-known advantages of having a declarative intermediate language.

### 4.1. *The Flat-LEAF*

In this section we define the syntax of the Flat-LEAF. The language alphabet is the same as the alphabet of K-LEAF (see Section 3). The main diffeences are that no more than one function occurrence—the most external one—can be present in a term (no function compositions), and (flat) equations are allowed in the body of clauses.

A *flat atom* is

(i) an equation $f(d_1, ..., d_n) = d$, where $f \in F$ and $d_1, ..., d_n$ and $d$ are data terms, or

(ii) a strict equation $d_1 \equiv d_2$, where $d_1$ and $d_2$ are data terms, or

(iii) a relation $p(d_1, ..., d_n)$, where $p \in P$ and $d_1, ..., d_n$ are data terms.

A *flat body atom* is a flat atom. The *flat heads* are the flat atoms of the kinds (i) and (iii). The atoms of the kind (ii) can be used as flat heads only in the special case of clauses defining the strict-equality (see Section 4.3). A *definite flat clause* is a clause $A \leftarrow B_1, ..., B_n$, where $A$ is a flat head, and $B_1, ..., B_n$ are flat body atoms.

Of course, some constraints, as in the K-LEAF case, are necessary in order to guarantee continuity of predicates and functions. Let us introduce the following definitions. For any expression (term, atom or clause) $E$, $\mathrm{Var}(E)$ is the set of variables occurring in $E$. If $A$ is an equation $t_1 = t_2$ then $\mathrm{In}(A)$ (*input variables, or consumed variables*) is $\mathrm{Var}(t_1)$, while $\mathrm{Out}(A)$ (*output variables, or produced variables*) is $\mathrm{Var}(t_2)$. The atom $t_1 = t_2$ is the producer of all the variables in $\mathrm{Out}(t_1 = t_2)$. If $A$ is a relational atom or a strict equation then $\mathrm{In}(A) = \mathrm{Var}(A)$ and $\mathrm{Out}(A) = \varnothing$. Note that this is in agreement with the interpretation of $A$ as the equation $A = \mathit{true}$. If $A_1, A_2$ are equational atoms, then the producer-consumer relationship $\angle$ is defined as follows: $A_1 \angle A_2$ iff $\mathrm{Out}(A_1) \cap \mathrm{In}(A_2) \neq \varnothing$ ($A_2$ consumes some of the variables produced by $A_1$). Note that $A_1 \angle A_2$ is possible only if $A_1$ is an equation.

Let $A \leftarrow B_1, ..., B_n$ be a Flat-LEAF clause. The additional constraints are

(a) $\angle$ is acyclic, that is $\sim\exists B_{j_1}, ..., B_{j_k}$ $(k \geqslant 1)$ $B_{j_1} \angle \cdots \angle B_{j_k} \angle B_{j_1}$ (or, equivalently, the transitive closure of $\angle$ is an ordering relation).

(b) If $i \neq j$, then $\mathrm{Out}(B_i) \cap \mathrm{Out}(B_j) = \varnothing$ (*no multiple productions*).

(c) For every variable $x \in \mathrm{Out}(A)$, either $x \in \mathrm{In}(A)$ or there exists $B_j$ *with definite inputs* such that $x \in \mathrm{Out}(B_j)$. $B_j$ has definite inputs iff for every variable $y \in \mathrm{In}(B_j)$, either $y \in \mathrm{In}(A)$ or, recursively, there exists a (distinct) $B_i$ with definite inputs such that $y \in \mathrm{Out}(B_i)$.

(d) For every equation $B_j$, $\mathrm{Out}(B_j) \cap \mathrm{In}(A) = \varnothing$.

(e) $A$ is *left-linear.*

(f) For every equation $t_1 = t_2$ in the body there are no multiple occurrences of the same variable in $t_2$ (*right linearity*).

A *flat goal* is a goal $\leftarrow B_1, ..., B_n$, where $B_1, ..., B_n$ are flat bodies and conditions (a), (b) and (f) in the previous definition hold. If $n = 0$ then the goal is an *empty goal*, denoted by $\leftarrow$. A *flat program* is a set of definite flat clauses satisfying the condition of weak non-ambiguity.

Few comments are in order. Constraint (c) replaces the *definite outputs* condition in K-LEAF equational heads. The other constraints avoid implicit (i.e., expressed via identity of variables) equality tests between two or more terms with possibly undefined or infinite value. Explicit tests of this kind are avoided by the requirement that only data terms can occur as the right part of body equations. Remember that the occurrence of these tests could affect the continuity of the functions defined in the program, or, from an operational point of view, the existence of a sound and complete computational mechanism. In Section 5.1 all of these constraints will be shown necessary for the definition of the fixed-point semantics (which traditionally represents the link between the operational and the model-theoretic semantics).

## 4.2. *Transformation to Flat Form (Flattening)*

In this section we formally define the transformation from K-LEAF programs into Flat-LEAF programs (*flattening*). Let us introduce some notions. A *data context* is [ ] (*hole*), or $c(e_1, ..., e_n)$, where $c \in C$ and $e_1, ...., e_n$ are data terms or data contexts. In other words a data context is a data term with some holes. For example, if $c$, $d \in C$ and $x$ is a variable, then $d([\ ], c(c(x, [\ ]), x))$ is a context with two holes. Data terms are data contexts with no holes. The *application* of a data context (with $n$ holes) $e$ to the terms $t_1, ..., t_n$ is the term $e[t_1, ..., t_n]$ obtained by replacing the holes in $e$ by $t_1, ..., t_n$, from left to right. For example, the application of the above data context to $f(x)$ and $g(x)$ gives $d(f(x), c(c(x, g(x)), x))$. Any term can be represented as an application of a data context to functional terms. Namely, $t$ is reresented as $e[t_1, ..., t_n]$, where $e$ is the external part of $t$ containing data constructors only (if any), and $t_1, ..., t_n$ are the subterms of $t$ with function symbols at the outermost level. In particular, if $t$ is functional then it can be obtained as the application of $[\ ]$ to $t$ itself. From now on the terms $t_1, ..., t_n$ in $e[t_1, ..., t_n]$, if not specified, are implicitly assumed to be functional.

In the following we give a top-down declarative description of flattening: *flat* denotes the flattening on programs; *flatc* the flattening on clauses; and *flatb* the flattening on body atoms. Note that *flatb* is defined on equations also, since unflattened equations (of the form $t = x$, where $x$ is a variable) can be introduced by *flatc* and *flatb*.

(1) *Flattening of K-LEAF programs.* If $W$ is $\{C_1, ..., C_m\}$, then $flat(W)$ is $\{flatc(C_1), ..., flatc(C_m)\}$.

(2) *Flattening of K-LEAF definite clauses and goal statements.*

(2.1) If $C$ is $p(d_1, ..., d_n) \leftarrow B_1, ..., B_n$ then $flatc(C)$ is $p(d_1, ..., d_n) \leftarrow flatb(B_1), ..., flatb(B_n)$.

(2.2) If $C$ is $f(d_1, ..., d_k) = e[t_1, ..., t_m] \leftarrow B_1, ..., B_n$, then $flatc(C)$ is $f(d_1, ..., d_k) = e[x_1, ..., x_m] \leftarrow flatb(t_1 = x_1), ..., flatb(t_m = x_m), flatb(B_1), ..., flatb(B_n)$, where $x_1, ..., x_m$ are new variables.

(2.3) If $C$ is $\leftarrow B_1, ..., B_n$ (goal statement), then $flatc(C)$ is $\leftarrow flatb(B_1), ..., flatb(B_n)$.

(3) *Flattening of body atoms.*

(3.1) If $B$ is $e'[t'_1, ..., t'_m] \equiv e''[t''_1, ..., t''_n]$, then $flatb(B)$ is $flatb(t'_1 = x'_1), ..., flatb(t'_m = x'_m), flatb(t''_1 = x''_1), ..., flatb(t''_m = x''_n), e'[x'_1, ..., x'_m] \equiv e''[x''_1, ..., x''_n]$, where $x'_1, ..., x'_m, x''_1, ..., x''_n$ are new variables.

(3.2) If $B$ is $p(e_1[t_{11}, ..., t_{1m_1}], ..., e_n[t_{n1}, ..., t_{nm_n}])$, then $flatb(B)$ is $flatb(t_{11} = x_{11}), ..., flatb(t_{1m_1} = x_{1m_1}), ..., flatb(t_{n1} = x_{n1}), ..., flatb(t_{nm_n} = x_{nm_n}), p(e_1[x_{11}, ..., x_{1m_1}], ..., e_n[x_{n1}, ..., x_{nm_n}])$ where $x_{11}, ..., x_{1m_1}, ..., x_{n1}, ..., x_{nm_n}$ are new variables.

(3.3) If $B$ is of the form $f(e_1[t_{11}, ..., t_{1m_1}], ..., e_n[t_{n1}, ..., t_{nm_n}]) = x$, then $flatb(B)$ is: $flatb(t_{11} = x_{11}), ..., flatb(t_{1m_1} = x_{1m_1}), ..., flatb(t_{n1} = x_{n1}), ..., flatb(t_{nm_n} = x_{nm_n}), f(e_1[x_{11}, ..., x_{1m_1}], ..., e_n[x_{n1}, ..., x_{nm_n}]) = x$ where $x_{11}, ..., x_{1m_1}, ..., x_{n1}, ..., x_{nm_n}$ are new variables.

Note that since the number of clauses in a program, the number of atoms in a clause, and the nested structure of an atom are finite, the flattening algorithm terminates in a finite number of steps.

EXAMPLE 2. Consider the program of Example 1. The corresponding Flat-LEAF program is

$\{plus(0, x) = x.$

$plus(s(x), y) = s(v) \leftarrow plus(x, y) = v.$

$nat(x) = cons(x, v) \leftarrow nat(s(x)) = v.$

$odd(\ ) = odd1(v) \leftarrow nat(s(0)) = v.$

$odd1(cons(x, cons(y, z))) = cons(x, v) \leftarrow odd1(z) = v.$

$sqrlist(\ ) = cons(0, v_1) \leftarrow sqrlist1(0, v_2) = v_1, odd(\ ) = v_2.$

$sqrlist1(x, cons(y, z)) = cons(v_1 . v_2) \leftarrow plus(x, y) = v_1, sqrlist1(v_3, z) = v_2, plus(x, y) = v_3.$

$sqr(x) = y \leftarrow extract(x, z) = y, sqrlist(\ ) = z.$

$extract(0, cons(x, y)) = x.$

$extract(s(x), cons(y, z)) = v \leftarrow extract(x, z) = v.$

$p(x, y, z) \leftarrow v_1 \equiv v_2, sqr(z) = v_1, plus(v_3, v_4) = v_2, sqr(x) = v_3, sqr(y) = v_4.\}$

Note that the only equations introduced by the flattening in the body are of the form $t = x$. Flattening does not affect weak non-ambiguity; together with the following theorem this guarantees that the flattening of K-LEAF programs actually results into Flat-LEAF programs.

THEOREM 1. *If $C$ is a K-LEAF clause [goal] then flatc($C$) is a Flat-LEAF clause [goal].*

*Proof.* It is easy to see that flatc($C$) satisfies conditions (i)–(iii) on flat atoms. Let us prove that it satisfies constraints (a)–(f).

(a) (by contradiction) Assume that, in the body of flat($C$), $\exists B_1, ..., B_n$ such that $B_1 \angle \cdots \angle B_n \angle B_1$. Clearly one of them, say $B_i$, is a new atom introduced by flattening. Let $j$, $k$ be the indexes such that $B_j \angle B_i \angle B_k \cdots \angle B_j$. Consider the flattening step which has generated $B_i$. There are two cases:

(1) $B_i$ and $B_k$ have been generated by *flat*($B'_k$). Then $B_j \angle B'_k \angle \cdots \angle B_j$ holds.

(2) $B_j$ and $B_i$ have been generated by *flat*($B'_i$). Then $B'_i \angle B'_k \angle \cdots \angle B'_i$ holds.

No other cases are possible; for example, if $B_j$ were introduced by flattening the head of $C$, then Out($B_j$) would be a variable occurring only in the (right part of the) head of *flatc*($C$). This argument shows that flattening steps cannot generate cycles, and therefore, by repeating it, we eventually infer the existence of one such cycle in $C$, which is a contradiction.

(b, d, f) Every equational body atom introduced by the flattening algorithm has the form $t = x$, where $x$ is a new variable and therefore $x$ cannot occur either in the output part of any other equational atom or in the input part of the head. Moreover, right linearity is trivially ensured.

(c) If *flatc*($C$) has an equational head, say $t = e[x_1, ..., x_m]$, then $C$ has an equational head of the form $t = e[t_1, ..., t_m]$, and the case (22) of flattening applies. Since $C$ has definite outputs, Var($e$) is contained in Var($t$). Moreover, the $x_i$'s occur also in the right part of body atoms introduced by flattening the head. It is easy to show (by induction) that these atoms have definite inputs.

(e) It is sufficient to note that flattening does not modify the left part of the equational heads. ▌

The correctness of flattening, that is, the semantic equivalence between the programs $W$ and *flat*($W$), is proved in Section 2.5.2.

## 4.3. *Equality and Strict-Equality*

Due to the flatness of atoms, a Flat-LEAF program $W$ can be executed by SLD-resolution (with syntactic unification) only. Anyway, the behaviour of strict equations has to be defined. Moreover, some problems can arise in handling equations. In this section we show that equations and strict equations can also be handled by SLD-resolution; it is sufficient to add two special sets of Horn clauses, $=(W)$ and $\equiv(W)$.

### 4.3.1. *Strict-Equality*

$\equiv$ is intended to model the strict equality on data terms. This relation can be defined, in a Horn clauses style, by the standard axioms which model the identity on ground data terms.

DEFINITION 1. (Clauses for $\equiv$). Given a program $W$, $\equiv(W)$ is the set of all the clauses

(i) $c \equiv c$, for every (0-adic) data constructor $c$ of $W$, and

(ii) $d(x_1, ..., x_n) \equiv d(y_1, ..., y_n) \leftarrow x_1 \equiv y_1, ..., x_n \equiv y_n$, for every ($n$-adic) data constructor $d$ of $W$.

For example, if $W$ is the program $\{plus(0, y) = x,\ plus(s(x), y) = s(plus(x, y))\}$, where $0$ and $s$ are data constructors, then $\equiv(W)$ is the set $\{0 \equiv 0,\ s(x) \equiv s(y) \leftarrow x \equiv y\}$. Note that $\equiv$ would not result strict if (ii) were replaced by a clause $d(x_1, ..., x_n) \equiv d(x_1, ..., x_n)$.

### 4.3.2. *Equality*

The intended meaning of equations is the (algebraic) congruence relation induced by the clauses (with equational headers) of $W$, viewed as conditional equivalence axioms. We want this relation to be non-strict, that is, it must hold even if the arguments are both undefined or partially defined (with the same values on the defined component). Note that this condition implies the relation to be non-monotonic. This form of non-strictness allows to handle functions which are (partially or totally) undefined, or functions whose output is infinite.

EXAMPLE 3. Let $W$ be the program $\{first0(cons(0, w)),\ nat(n) = cons(n, nat(s(n)))\}$, where $0$, $s$ and $cons$ are data constructors; $first0$ is a predicate which succeeds if the first element of its list argument is $0$ and does not depend on the rest of it ($w$); $nat(n)$ is a function which computes the infinite list of all the naturals starting from $n$. If we want $first0$ to be non-strict with respect to $w$, then the goal $G: \leftarrow first0(nat(0))$ should succeed. Let us now consider $flat(W) = \{first0(cons(0, w)),\ nat(n) = cons(n, v) \leftarrow nat(s(n)) = v\}$, $flatc(G): \leftarrow nat(0) = y,\ first0(y)$. After two resolution steps on $flatc(G)$ we obtain the goal $G': \leftarrow nat(s(0)) = v$. In order to refute $flatc(G)$ also the equation $nat(s(0)) = v$ should be solved.

In general, let $f(t_1, ..., t_n)$ occur in a K-LEAF body atom, and assume that $f$ does not depend (is not strict) on the $i$th argument. By definition, if $t_i$ is a functional term $g(u_1, ..., u_m)$ then the flattening introduces an equation $g(u_1, ..., u_m) = x$ in the body (and replaces $t_i$ by $x$ in $f(t_1, ..., t_n)$). We must allow this equation to succeed with an undefined value of $x$, since it represents the $i$th argument of $f$. In other words, every equation with a pure variable in the right part must be solvable whenever the value of the variable is not required from other atoms.

DEFINITION 2 (Clauses for $=$). Given a program $W$, $=(W)$ is the set of all the clauses $f(x_1, ..., x_n) = \perp$, where $f$ is an ($n$-adic) function symbol of $W$, $x_1, ..., x_n$ are variables, and $\perp$ is a special data constructor (constant) symbol, standing for the undefined value.

In practice, this definition establishes that every function has a special data constant ($\perp$) as default output value.

## 4.4. *Operational semantics of K-LEAF and Flat-LEAF*

The operational semantics of a K-LEAF program $W$ is based on that of the corresponding Flat-LEAF program, i.e., on SLD-resolution. $\equiv$ and $\equiv$ act as standard predicate symbols respectively defined by $\equiv(W)$ and by the clauses with equational head together with $=(W)$. The standard definitions have to be slightly modified to deal with the undefined symbol $\perp$ (introduced by $=(W)$). $D_\perp(V)$ is the set of *partial data terms*, i.e., the terms built on $C$, $V$, and $\perp$. $D_\perp$ is the set of *ground* (i.e., with no variable occurrences) *partial data terms*. $D(V)$ and $D$ are the subsets of $D_{\perp}(V)$ and $D_{\perp}$ containing terms with no occurrences of $\perp$. A *substitution* $\vartheta$ is a mapping $\vartheta: V \rightarrow D_{\perp}(V)$ whose domain $\mathrm{Dom}(\vartheta) = \{x \mid \vartheta(x) \neq x\}$ is finite. $\vartheta$ is said to bind a variable $x$ if $\vartheta(x) \neq x$. A *ground substitution* is a substitution which maps into elements of $D_{\perp}$. The application $E\vartheta$ of a substitution $\vartheta$ to an expression (term or atom) $E$, and the composition $\vartheta\gamma$ of the substitutions $\vartheta$ and $\gamma$, are defined as usual, see [36]. $E_1$ is a variant of $E_2$ if $\exists\vartheta$, $\gamma$ such that $E_1\vartheta = E_2$ and $E_1 = E_2\gamma$. $\vartheta$ is a *unifier* of $E_1$ and $E_2$ iff $E_1\vartheta = E_2\vartheta$. Moreover, $\vartheta$ is the *most general unifier* (m.g.u.) of $E_1$, $E_2$ iff for every other unifier $\sigma$ (of $E_1$, $E_2$) $\exists\gamma$ $\vartheta\gamma = \sigma$.

Given a flat goal $G: \leftarrow A_1, ..., A_m$ the computation of $G$ in a flat program $W$ is defined in the usual way and also involves the clauses of $=(W)$. Let $C: A \leftarrow B_1, ..., B_n$ be a variant of a clause of $W$, $=(W)$, or $\equiv(W)$, sharing no variables with $G$. If $A_i$ and $A$ are unifiable, with m.g.u. $\vartheta$, then the new goal $G': \leftarrow (A_1, ..., A_{i-1}, B_1, ..., B_n, A_{i+1}, ..., A_m)\,\vartheta$ is derived. Note that since $A_i$ and $A$ are in flat form, the m.g.u. can be determined by the standard unification algorithm. $G'$ is still a correct Flat-LEAF goal, as stated by the following theorem.

THEOREM 2. *Let $G$ be a Flat-LEAF goal, $W$ be a Flat-LEAF program, and $C$ be a clause of $W$, $=(W)$, or $\equiv(W)$. The goal $G'$, obtained from $G$ and $C$ by a resolution step, is a Flat-LEAF goal.*

The proof will be based on the following lemma.

LEMMA 1. *Let $d$, $d'$ be data terms, and assume that $d'$ is linear (i.e., $d'$ has no multiple occurrences of variables). Then the m.g.u. of $d$ and $d'$ (if any) assigns to the variables of $d$ terms which are linear and share no variables.*

*Proof.* This result can easily be proved by induction on the number of steps in Robinson's m.g.u. algorithm [1, 36]. ∎

*Proof of Theorem* 2. Let $G$ be $\leftarrow A_1, ..., A_m$, and $A_i$ be the atom chosen for the resolution step. Let $C$ be $A \leftarrow B_1, ..., B_n$, and $G'$ be $\leftarrow$ $(A_1, ..., A_{i-1}, B_1, ..., B_n, A_{i+1}, ..., A_m)\,\vartheta$.

(1) (Flatness) $A_1, ..., A_{i-1}, A_{i+1}, ..., A_m$ and $B_1, ..., B_n$ are flat, and $\vartheta$ binds to data terms only.

(2) (No cycles) We prove that from every $A_r\vartheta \angle B_t\vartheta$ we can infer $A_r \angle A_i$, and from every $B_s\vartheta \angle A_s\vartheta$ we can infer $A_i \angle A_s$. Therefore, if $G'$ has a cycle, then also $G$ has a cycle, which is a contradiction.

($A_r \angle A_i$) Let $x \in \mathrm{Out}(A_r\vartheta) \cap \mathrm{In}(B_t\vartheta)$, and $y$ be the variable in $\mathrm{Out}(A_r)$ such that $x \in \mathrm{Var}(y\vartheta)$. Since $G$ and $C$ share no variables, then $y \in \mathrm{Var}(A_i)$. Moreover, because of the absence of multiple productions in $C$, $y \notin \mathrm{Out}(A_i)$, hence $y \in \mathrm{In}(A_i)$, and, therefore, $A_r \angle A_i$ holds.

($A_i \angle A_s$) Let $x' \in \mathrm{Out}(B_u\vartheta) \cap \mathrm{In}(A_s\vartheta)$, and let $y'$ be the variable in $\mathrm{Out}(B_u)$ such that $x' \in \mathrm{Var}(y'\vartheta)$. Similarly to the previous case, we obtain $y' \in \mathrm{Var}(A)$.

Because of constraint (d), $y' \notin \mathrm{In}(A)$, hence $y' \in \mathrm{Out}(A)$. Let $z'$ be the variable in $\mathrm{In}(A_s)$ such that $x' \in \mathrm{Var}(z'\vartheta)$. Then $z' \in \mathrm{Var}(A_i)$. We want to show that $z' \in \mathrm{Out}(A_i)$. Assume that $z' \in \mathrm{In}(A_i)$ only. Consider the m.g.u. $\vartheta_1 \leqslant \vartheta$ of the left part of $A$ and $A_i$. Due to constraint (a), $\mathrm{In}(A_i) \cap \mathrm{Out}(A_i) = \emptyset$, therefore $\mathrm{Var}(z'\vartheta_1) \cap \mathrm{Out}(A_i\vartheta_1) = \emptyset$. Moreover, $\vartheta_1$ does not affect the right part of $A_i$, and then constraint (f) holds also on $A_i\vartheta_1$. Consider the m.g.u. $\vartheta_2$ of (the right part of) $A\vartheta_1$ and $A_i\vartheta_1$ such that $\vartheta_1\vartheta_2 = \vartheta$. Since $y' \notin \mathrm{In}(A)$, and by Lemma 1, variables of $z'\vartheta_1\vartheta_2$ are distinct from those of $y'\vartheta_1\vartheta_2$ and thus $\mathrm{Var}(z'\vartheta) \cap \mathrm{Var}(y'\vartheta) = \mathrm{Var}(z'\vartheta_1\vartheta_2) \cap \mathrm{Var}(y'\vartheta_1\vartheta_2) = \emptyset$ holds. This is not possible, since $x' \in \mathrm{Var}(z'\vartheta) \cap \mathrm{Var}(y'\vartheta)$ by hypothesis.

(3) (No multiple productions) Assume there are two equations, $E_1$ and $E_2$, in $G'$, such that $\mathrm{Out}(E_1) \cap \mathrm{Out}(E_2) \neq \emptyset$, and let us analyze the three possible cases.

(i) $E_1 = A_r\vartheta$ and $E_2 = A_s\vartheta$ for some $A_r$ and $A_s$ in $G$. Let $x \in \mathrm{Out}(A_r\vartheta) \cap \mathrm{Out}(A_s\vartheta)$. Let $y$, $z$ be the variables, in $\mathrm{Out}(A_r)$ and $\mathrm{Out}(A_s)$ respectively, such that $x \in \mathrm{Var}(y\vartheta) \cap \mathrm{Var}(z\vartheta)$. Clearly, $y, z \in \mathrm{Var}(A_i)$. Because of constraint (b), $y$, $z \notin \mathrm{Out}(A_i)$, and then $y$, $z \in \mathrm{In}(A_i)$. Hence, by constraint (e) and Lemma 1, $\mathrm{Var}(y\vartheta) \cap \mathrm{Var}(z\vartheta) = \emptyset$, which is a contradiction.

(ii) $E_1 = B_t\vartheta$ and $E_2 = B_u\vartheta$ for some body atoms $B_t$ and $B_u$ in $C$. Let $x \in \mathrm{Out}(B_t\vartheta) \cap \mathrm{Out}(B_u\vartheta)$. Let $y$, $z$ be the variables, in $\mathrm{Out}(B_t)$ and $\mathrm{Out}(B_u)$ respectively, such that $x \in \mathrm{Var}(y\vartheta) \cap \mathrm{Var}(z\vartheta)$. Clearly, $y, z \in \mathrm{Var}(A)$. Because of condition (d) $y$, $z \notin \mathrm{In}(A)$, and then $y, z \in \mathrm{Out}(A)$. Therefore, by constraint (f) and Lemma 1, $\mathrm{Var}(y\vartheta) \cap \mathrm{Var}(z\vartheta) = \emptyset$, thus contradicting the hypothesis.

(iii) $E_1 = A_r\vartheta$ and $E_2 = B_t\vartheta$ for some $A_r$ in $G$, and some body atom $B_t$ in $C$. Let $x \in \mathrm{Out}(A_r\vartheta) \cap \mathrm{Out}(B_u\vartheta)$. Let $y$ and $z$ be the variables, in $\mathrm{Out}(A_r)$ and $\mathrm{Out}(B_t)$ respectively, such that $x \in \mathrm{Var}(y\vartheta) \cap \mathrm{Var}(z\vartheta)$. As in the previous case, $y \in \mathrm{In}(A_i)$, and $z \in \mathrm{Out}(A)$. By applying the same argument as in (2), we can show that $\mathrm{Var}(z'\vartheta) \cap \mathrm{Var}(y'\vartheta) = \emptyset$, thus contradicting the hypothesis.

(4) (Right linearity) Assume there is in $G'$ an equational atom with two occurrences of the same variable in the right part. This atom can be one of the $A_j$'s or one of the $B_j$'s. In the first case the proof is similar to that of (3)(i), while in the second one it is similar to that of (3)(ii).

Theorem 2 shows that the set of Flat-LEAF goals is closed with respect to the resolution steps. Therefore SLD-resolution can be applied again to the derived goals. A *derivation* of a goal $G$ in a program $W$ is a (finite or infinite) sequence of goals $G = G_0, G_1, \ldots, G_n, \ldots$, and of substitutions $\vartheta_1, \vartheta_2, \ldots, \vartheta_n, \ldots$, such that $G_i$ is obtained by SLD-resolution from $G_{i-1}$ and from one of the clauses of $W$, $=(W)$, or $\equiv(W)$, and $\vartheta_i$ is the corresponding m.g.u. A finite derivation ending with $G_n$ is denoted by $G \mapsto^{\vartheta} G_n$, where $\vartheta$ is the composition of the m.g.u.'s used in the derivation. If $G_n$ is the empty goal ($\leftarrow$) then the derivation is *successful* (*refutation*), and $\vartheta|_G$ (the restriction of $\vartheta$ to the variables of $G$, see [36]) is the *computed answer substitution* (c.a.s.).

DEFINITION 3. Given a Flat-LEAF program $W$, the operational semantics of a Flat-LEAF atom $A$ is $O_F(A) = \{A' \text{ ground} \mid \leftarrow A \mapsto^{\vartheta} \leftarrow \text{and } \exists \vartheta' \colon V \to D_\perp \text{ such that } A\vartheta\vartheta' = A'\}$.

DEFINITION 4. Given a K-LEAF program $W$, the operational semantics of a K-LEAF atom $A$ is $O_K(A) = \{A' \text{ ground} \mid \leftarrow \mathit{flatb}(A) \mapsto^{\vartheta} \leftarrow \text{and } \exists \vartheta' \colon V \to D \text{ such that } A\vartheta\vartheta' = A'\}$, where the refutation of *flatb*$(A)$ is performed by using the clauses in *flat*$(W)$, $=(W)$, and $\equiv(W)$.

The following proposition shows that the atoms in $O_K(A)$ do not contain occurrences of $\perp$.

PROPOSITION 1. *Let $G$ be a K-LEAF goal, and $W$ be a K-LEAF program. Assume that flatc$(G)$ has a refutation in flat$(W)$ with computed answer substitution $\vartheta$. Then, $\vartheta|_G$ maps into $D(V)$.*

The proof will be based on the following lemma.

LEMMA 2. *Let $G$ be a K-LEAF goal, and $W$ be a K-LEAF program. Consider a derivation $\leftarrow$flatc$(G) \mapsto^{\vartheta} G_n$ in flat$(W)$. For each atom $A$ in $G_n$ we have* $\text{Out}(A) \cap \text{Var}(G\vartheta) = \varnothing$.

*Proof.* By induction on the length of the derivation.

$(n = 0)$ Every equation $t = x$ in flatc$(G)$ has been introduced by the flattening, then $x$ is a new variable (not occurring in $G$).

$(n > 1)$ Assume $\leftarrow$*flatc*$(G) \mapsto^{\vartheta} G_{n-1}$, and $G_n$ derivable from $G_{n-1}$. Let $G_{n-1}$ be $\leftarrow A_1, \ldots, A_m$, and $A_i$ be the selected atom. Let $A \leftarrow B_1, \ldots, B_n$ be the clause chosen for the resolution step, and $\phi$ be the m.g.u. of $A_i$ and $A$. Then the goal $G_n$ is $\leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\,\phi$.

(i) $\text{Out}(A_j\phi) \cap \text{Var}(G\vartheta\phi) = \varnothing$ for each $A_j$ in $G_{n-1}$. In fact, $\text{Var}(G\vartheta\phi) = \text{Var}(\text{Var}(G\vartheta)\,\phi)$, and $\text{Out}(A_j\phi) = \text{Var}(\text{Out}(A_j)\,\phi)$. By inductive hypothesis, $\text{Out}(A_j) \cap \text{Var}(G\vartheta) = \varnothing$. Moreover, because of constraints (a), (b), and (e), terms assigned by $\phi$ to $\text{Out}(A_j)$ cannot share variables with any other term assigned by $\phi$.

(ii) $\text{Out}(B_j\phi) \cap \text{Var}(G\vartheta\phi) = \varnothing$ for each $B_j$ in $C$. In fact, as before, $\text{Var}(G\vartheta\phi) = \text{Var}(\text{Var}(G\vartheta)\,\phi)$. By constraint (d), $\text{Var}(\text{Out}(B_j)\,\phi)$ is contained in $\text{Var}(\text{Out}(A)\,\phi) \cup \text{Out}(B_j)$. Of course, $\text{Out}(B_j) \cap \text{Var}(G\vartheta\phi) = \varnothing$. Moreover, $\text{Var}(\text{Out}(A)\,\phi) \cap \text{Var}(G\vartheta\phi) = \text{Out}(A_i\phi) \cap \text{Var}(G\vartheta\phi) = \varnothing$ by the same arguments used in the proof of (i). ∎

*Proof of Proposition* 1. Let flat$(G) = G_0, G_1, \ldots, G_n$ be the sequence of goals, and $\vartheta_1, \vartheta_2, \ldots, \vartheta_n$ be the sequence of m.g.u.'s obtained during the derivation. Assume that $\vartheta_i$ introduces a binding to $\perp$. Then the $i$th resolution step involves a clause in

$= (W)$ and an equation $t = x$, and the only variable bound to $\perp$ is $x$. By Lemma 2, $\mathrm{Var}(G\vartheta_1\vartheta_2 \cdots \vartheta_{i-1})$ does not contain $x$, then $(\vartheta_1\vartheta_2 \cdots \vartheta_{i-1}\vartheta_i)_{|G}$ maps into terms containing no occurrences of $\perp$. Since this holds for every $\vartheta_i$ in the sequence, the assertion is proved. ▮

## 5. Declarative Semantics

In this section we define the declarative (i.e., fixpoint and model-theoretic) semantics for K-LEAF and Flat-LEAF. To manage both cases homogeneously we introduce a superlanguage, called S-LEAF, or semantics LEAF, of which K-LEAF and Flat-LEAF are sublanguages. The definition of S-LEAF syntax is the same as the Flat-LEAF one, the only difference being the presence of functional nestings (body atoms can have functional terms in the left part and head atoms can have functional terms in the right part). S-LEAF without body equations is K-LEAF, while S-LEAF without functional nestings if Flat-LEAF. Thanks to the introduction of S-LEAF, we avoid the need for a double semantics (one for K-LEAF and the other for Flat-LEAF). Moreover, the proofs of the propositions relating the two program forms (e.g., the semantic equivalence of a user program with its flattened version) become easier. In fact, a K-LEAF program can "continuously" be transformed into a Flat-LEAF program through some intermediate S-LEAF program.

In the previous sections it has been shown that functions can have (partially or totally) undefined or even infinite results. As a consequence, functions, predicates, and equalities can have undefined or infinite arguments. The infinite values (obtained as outputs of functions) can be viewed as limits of chains of finite partially undefined values (their partial approximations). These arguments lead us to consider the notion of algebraic CPO as a natural setting to assign a meaning to functions.

Let us briefly recall the related notions. Let $S$ be a set and let $\leqslant$ be an *order relation* on $S$. A set $D \subseteq S$ is a *directed set* iff for any $a$, $b \in D$ there exists $c \in D$ such that $a, b \leqslant c$. $(S, \leqslant)$ is a complete partial order (CPO) iff there exists a minimal element $\perp_S$ (bottom) and every directed set $D \subseteq S$ has a least upper bound $\bigsqcup D \in S$; $a \in S$ is a *finite* (or *algebraic*) element iff for every directed set $D$, if a $\leqslant \bigsqcup D$, then $\exists d \in D$ such that $a \leqslant d$. Let $S_0$ be the set of algebraic elements of $S$, and, for $a \in S$, let $\hat{a}$ be the set $\{b \in S_0 \mid b \leqslant a\}$. $S$ is an *algebraic* CPO iff, for every $a \in S$, $\hat{a}$ is a directed set and $\bigsqcup \hat{a} = a$. If $(S, \leqslant)$ and $(S', \leqslant)$ are CPOs, and $f$ is a function from $S$ to $S'$, then $f$ is *continuous* iff for every directed set $D \subseteq S$ $f(\bigsqcup D) = \bigsqcup_{d \in D} f(d)$. If $(S, \leqslant)$ and $(S', \leqslant)$ are algebraic CPOs, then it is sufficient that $f(\bigsqcup D) = \bigsqcup_{d \in D} f(d)$ holds for every directed algebraic set $D \subseteq S_0$.

A simple algebraic CPO is the set $LL = \{\perp_{LL}, \mathit{true}, \mathit{false}\}$, with the ordering $\perp_{LL} \leqslant \mathit{true}$ and $\perp_{LL} \leqslant \mathit{false}$ (three-valued boolean CPO). Note that all the elements of $LL$ are algebraic. This CPO will have an important role in the definition of the interpretations for predicates.

In the following, $T_{\perp}(V)$ is the set of *partial terms*, i.e., the terms built on $C$, $\perp$,

$F$, and $V$. $T_\perp$ is the set of *ground partial terms*. For the sake of simplicity we introduce a notation for tuples. The tuple of expressions (terms or atoms) $E_1, ..., E_n$ will be represented by $\boldsymbol{E}$.

Let $W$ be a S-LEAF program. An *interpretation I* for $W$ consists of an algebraic CPO $(S, \leqslant)$, and a meaning function $[\![ \ ]\!]_I$ which assigns to every constructor or function symbol a continuous mapping on $S$ and to every predicate symbol a continuous mapping from $S$ to $LL$. The interpretation of non-ground terms and atoms involves the notion of environment. An environment is a mapping $\rho : V \to S$,. Thus, the interpretation of a variable is assigned by the environment, the interpretation of a constant is assigned by $I$, and the interpretation of more complex formulas is derived by imposing the structural compositionality. Note that the meaning of non-ground expressions (denoted by $[\![ \ ]\!]_{I\rho}$) functionally depends on the chosen environment $\rho$. The meaning of ground expressions does not depend on $\rho$ and it is simply denoted by $[\![ \ ]\!]_I$.

$$\begin{array}{rll}
[\![v]\!]_{I\rho} = v\rho & \text{for} & v \in V \\
[\![f(\boldsymbol{t})]\!]_{I\rho} = [\![f]\!]_I([\![\boldsymbol{t}]\!]_{I\rho}) & \text{for} & f \in C \cup F \text{ and } \boldsymbol{t} = t_1, ..., t_n \in T_\perp(V) \\
[\![p(\boldsymbol{t})]\!]_{I\rho} = [\![p]\!]_I([\![\boldsymbol{t}]\!]_{I\rho}) & \text{for} & p \in P \text{ and } \boldsymbol{t} = t_1, ..., t_n \in T_\perp(V) \\
[\![\boldsymbol{t}]\!]_{I\rho} = [\![t_1]\!]_{I\rho}, ..., [\![t_n]\!]_{I\rho} & \text{for} & \boldsymbol{t} = t_1, ..., t_n \in T_\perp.
\end{array}$$

Moreover, we impose interpretation of the elements of $D_\perp$ as algebraic elements, the elements of $D$ as maximal (algebraic) elements, and $\perp$ as the bottom element, i.e., $[\![\perp]\!]_I = \perp_S$.

The interpretation of equations and strict equations depends only on the chosen CPO $(S, \leqslant)$. The symbol $=$ is interpreted as the identity on $S$, $\text{eq}_S$; that is,

$$[\![t_1 = t_2]\!]_{I\rho} = [\![t_1]\!]_{I\rho} \,\text{eq}_S [\![t_2]\!]_{I\rho}, \qquad \text{where, for } a, b \in S,$$

$$a \,\text{eq}_S\, b = \begin{cases} \textit{true}, & \text{if } a \text{ and } b \text{ are the same element} \\ \textit{false}, & \text{otherwise.} \end{cases}$$

Note that this interpretation is non-strict and non-monotonic (and, therfore, non-continuous), since, for example, $[\![\perp = \perp]\!]_{I\rho} = \textit{true}$, while $[\![\perp = t]\!]_{I\rho} = \textit{false}$, for $[\![t]\!]_{I\rho}$ different from $\perp_S$.

On the contrary, the intended meaning of strict-equality is a decidable identity. If we look at the ordering $\leqslant$ as a relation of *less definiteness*, then the natural subset on which identity can be decided is the set of maximal algebraic elements $S_M$. Maximality ensures that equal elements can not become distinct by adding more information, or vice versa. Algebraicity guarantees that the comparison can be done in a finite time (by exploring a finite amount of information). We define

$$[\![t_1 \equiv t_2]\!]_{I\rho} = [\![t_1]\!]_{I\rho} \, \textit{eqd}_S [\![t_2]\!]_{I\rho}.$$

| *and* | *true* | *false* | $\perp_L$ |
|---|---|---|---|
| *true* | *true* | *false* | $\perp_L$ |
| *false* | *false* | *false* | *false* |
| $\perp_L$ | $\perp_L$ | *false* | $\perp_L$ |

| $\Leftarrow$ | *true* | *false* | $\perp_L$ |
|---|---|---|---|
| *true* | *true* | *true* | *true* |
| *false* | *false* | *true* | $\perp_L$ |
| $\perp_L$ | $\perp_L$ | *true* | $\perp_L$ |

FIGURE 1

Various definitions of $eqd_S$, satisfying the above requirements, are possible. For example, given $a$, $b \in S$,

(1) $a\ eqd_S b$

$$= \begin{cases} true & \text{if } a \text{ and } b \text{ are algebraic, maximal, and are the same element} \\ false & \text{if } a \text{ and } b \text{ are algebraic and maximal, but they are not the same element} \\ \perp_{LL} & \text{otherwise.} \end{cases}$$

Other possible interpretations would be[1]

(2) $eqd_S$ is *true* on algebraic, maximal, and identical elements; $\perp_{LL}$ otherwise.

(3) $eqd_S$ is *true* on algebraic, maximal, and identical elements; *false* on elements having no common upper bound; $\perp_{LL}$ otherwise.

[1] It is possible to show that, in the semantics proposed in [34], the interpretation (2) is the minimal model of $\equiv(W)$, while (3) corresponds to the minimal model of the completion of $\equiv(W)$ (i.e., $\equiv(W)$ augmented with the clauses for negation), see [15]).

All these interpretations are strict and continuous, and they have the value *true* on the same elements. Theorems and assertions in the following hold for each of these definitions.

*Remark* 1. For each $d_1, d_2$ in $D$, for each interpretation $I$ in $(S, \leqslant)$, $[\![d_1]\!]_I$ $eqd_S$ $[\![d_1]\!]_I$ holds.

The above definition concerns the logical value of atomic formulas. More complex formulas, such as the S-LEAF clauses, have a value depending on the interpretation of conjunction and implication symbols. Let $\boldsymbol{B} = B_1, ..., B_n$ be a conjunction of body atoms. Then:

$$[\![A \leftarrow \boldsymbol{B}]\!]_{I\rho} = [\![A]\!]_{I\rho} \Leftarrow [\![\boldsymbol{B}]\!]_{I\rho}$$

$$[\![\leftarrow \boldsymbol{B}]\!]_{I\rho} = \text{false} \Leftarrow [\![\boldsymbol{B}]\!]_{I\rho}$$

$$[\![\boldsymbol{B}]\!]_{I\rho} = [\![B_1]\!]_{I\rho} \text{ and} \cdots \text{and } [\![B_n]\!]_{I\rho}$$

*and* and $\Leftarrow$ represent the extensions of conjunction and implication in the *three-valued logic* theory. There are different possibilities. Our choice, shown in Fig. 1, is the non-strict monotonic extension, in both cases. These definitions imply that, for any conjunction of ground atoms $B_1, ..., B_n$, $[\![B_1, ..., B_n]\!]_I$ is *true* iff $[\![\leftarrow B_1, ..., B_n]\!]_I$ is *false*.

## 5.1. *Herbrand Universe and Herbrand Interpretations*

The Herbrand interpretations are a special kind of interpretation. They are based on a "purely syntactical" domain, the Herbrand universe. The standard Herbrand universe is the set $D$ of ground data terms (see Section 4.4). In order to cope with undefined and infinite values, we must extend this notion. In the following we define, according to the general setting, the Herbrand universe as an algebraic CPO. The mathematical construction is similar to the one given in [24].

DEFINITION 5 (Complete Herbrand universe). Let $N$ be the set of positive naturals, and let $N^*$ be the set of all the strings on $N$, denoted by $n_1 . n_2 \cdots n_k$, where $n_1, n_2, ..., n_k \in N$. Each symbol of $C$ is assumed to be provided with an *arity*; $\bot$ has *arity* 0. In the following, $u$ and $n$ denote an element of $N^*$ and of $N$, respectively.

- A *ground data term* is a partial function $d: N^* \rightarrow C \cup \{\bot\}$ such that:
  - (i) if $u.n \in \text{Dom}(d)$ then $u \in \text{Dom}(d)$ (prefix closure)
  - (ii) $\forall u \in \text{Dom}(d)$, $d(u)$ has arity $n$ iff $u.1$, $u.n \in \text{Dom}(d)$ (arity consistency)
  - (iii) if $u.n \in \text{Dom}(d)$ then $u.1, u.2, ..., u.n \in \text{Dom}(d)$.

The *complete Herbrand universe* $CU$ is the set of all ground data terms.

- A ground data term $d$ is *finite* iff $\text{Dom}(d)$ is finite. It can be proved that the set of finite data terms is isomorphic to the set $D_\perp$ (see Section 4.4), and we will use the same notation. Terms that are in $CU$ but not in $D_\perp$ are called *infinite.* ▮

The new notion of a term can better be understood, by representing a term as a *finitely-branching tree* whose nodes are labeled by elements of $C \cup \{\perp\}$. The correspondence is given by the following labeling on the arcs. For each node, the first outcoming arc is labeled by 0, the second one is labeled by 1, and so on. Then, the strings of naturals of $\mathrm{Dom}(d)$ correspond to the paths on the tree.

We can now define the structure of the algebraic CPO on $CU$. The interpretation of $\perp$ as the least defined data structure yields a (flat) ordering $\leqslant$ on $C \cup \{\perp\}$, reflecting the information content of each constructor symbol. $\leqslant$ is defined as the *reflexive closure* of the relation

$$\forall c \in C, \perp \leqslant c.$$

The ordering on $C \cup \{\perp\}$ induces a (generally non-flat) ordering on $CU$,

$$d \leqslant d' \qquad \text{iff} \quad (\forall u \in \mathrm{Dom}(d), d(u) \leqslant d'(u)).$$

THEOREM 3. *$(CU, \leqslant)$ is an algebraic CPO, and $D_\perp$ is the set of its algebraic elements. Moreover, for every set $A \subseteq D_\perp$, if the elements of $A$ are compatible (i.e., if there exists an upper bound of $A$) then $A$ has a least upper bound, and, if $A$ is finite, then $\bigsqcup A \in D_\perp$.*

*Proof.* The result is quite standard, see [24] for a similar proof. The details can be found in [23]. ▮

DEFINITION 6 (Herbrand interpretations). Let $W$ be a S-LEAF program. A *Herbrand interpretation* $I$ of $W$ is an interpretation of the form $I = ((CU, \leqslant), [\![\ ]\!]_I)$ such that for each $c \in C$, $[\![c]\!]_I = c$ (i.e., every data term is interpreted by itself).

Note that the Herbrand interpretations differ only in the meaning of function and predicate symbols. In the following, $HI$ denotes the set of Herbrand interpretations.

Let us point out the meaning of equality and strict-equality in Herbrand interpretations. Equality is simply the syntactic identity. As far as strict-equality is concerned, note that in $(CU, \leqslant)$ the only maximal algebraic elements are the finite data terms which contain no occurrences of $\perp$. Hence strict-equality is the syntactical identity only on the subset $D$ (the standard Herbrand universe) of $CU$.

Consider the set of the mappings from a poset $(S, \leqslant)$ into a poset $(S', \leqslant)$. This set is naturally ordered by the relation $g \leqslant g'$ iff $\forall x \in S$, $g(x) \leqslant g'(x)$. The minimal mapping $\Omega$ is the one which assigns to every element of $S$ the bottom element of $S'$ ($\forall x \in S$, $\Omega(x) = \perp_{S'}$). This functional ordering induces an ordering on the set of Herbrand interpretations, which results in a CPO.

DEFINITION 7 (Ordering on Herbrand interpretations). • Let $I, J \in HI$. Then $I \leqslant J$ iff $\forall f \in F$, $\forall p \in P$, $[\![f]\!]_I \leqslant [\![f]\!]_J$, and $[\![p]\!]_I \leqslant [\![p]\!]_J$.

• The minimal element of $HI$ is the interpretation $I_\perp$, which maps every function and predicate symbol to $\Omega$. More formally,

$$\forall f \in F, \forall p \in P, \forall \boldsymbol{d} = d_1, ..., d_n \in CU, \qquad [\![f]\!]_{I_\perp}(\boldsymbol{d}) = \perp, \qquad \text{and} \qquad [\![p]\!]_{I_\perp}(\boldsymbol{d}) = \perp_{LL}.$$

THEOREM 4. *$(HI, \leqslant)$ is a CPO.*

*Proof.* Let $L$ be a directed subset of $HI$. Then, for each $f \in F$, and given $\boldsymbol{d} = d_1, ..., d_n \in CU$, the set $\{[\![f]\!]_I(\boldsymbol{d}) \mid I \in L\}$ is a directed set on $CU$. Since $CU$ is a CPO, there exists its *lub*. Analogously, since $LL$ is a CPO, for each $p \in P$ there exists $lub\,\{[\![p]\!]_I(\boldsymbol{d}) \mid I \in L\}$. Then define $\bigsqcup L$ such that $[\![f]\!]_{\bigsqcup L}(\boldsymbol{d}) = \bigsqcup\{[\![f]\!]_I(\boldsymbol{d}) \mid I \in L\}$ and $[\![p]\!]_{\bigsqcup L}(\boldsymbol{d}) = \bigsqcup\{[\![p]\!]_I(\boldsymbol{d}) \mid I \in L\}$. It is easy to prove that $\bigsqcup L$ is actually the *lub* of $L$. ▌

The following definitions and results deal with some technical properties of S-LEAF clauses with respect to the interpretations and the substitutions. They are introduced only to define a powerful mathematical tool in order to prove the main results (continuity of declarative semantics and completeness of the operational semantics) of the following sections. It should be noted that the construction needs all the constraints (a)–(f) of the definition of S-LEAF. This can be considered as a proof that this set of constraints is the minimal one which guarantees the existence of a complete proof procedure (in nonterminating conditional equational systems with extra variables and non-strict equations in the bodies, see Section 2.1).

Let $X$ be a finite set of variables. Subst$(X, CU)$ is the set of all the mappings $\vartheta: X \to CU$, that is, the set of (ground) substitutions having domain $X$ and mapping into $CU$. Note that, on the Herbrand universe, we can use this notion instead of the notion of environment. Subst$(X, CU)$ is an algebraic CPO with respect to the natural functional ordering $\vartheta_1 \leqslant \vartheta_2$ iff $\forall x \in X$, $\vartheta_1(x) \leqslant \vartheta_2(x)$. In fact, if $X$ has $n$ elements, then (Subst$(X, CU)$, $\leqslant$) is isomorphic to $(CU^n, \leqslant)$, where $CU^n$ is the cartesian product of $CU$ with itself, $n$ times (see, for example, [4]). The substitution $\vartheta_\perp$, mapping every variable of $X$ in $\perp$, is the minimal element of Subst$(X, CU)$. In the following, we consider the poset $(S(X, CU), \leqslant)$ obtained by augmenting Subst$(X, CU)$ with the special element *fail*, and by adding the relation $\vartheta_\perp \leqslant fail$. Of course, $(S(X, CU), \leqslant)$ is still an algebraic CPO. Let $X$, $Y$ be finite disjoint sets of variables. The *union* of substitutions is a mapping $\smile : S(X, CU) \times S(Y, CU) \to S(X \cup Y, CU)$ such that

- $fail \smile \vartheta = \vartheta \smile fail = fail$
- if $\vartheta \neq fail$ and $\phi \neq fail$ then
  - $(\vartheta \smile \phi)(x) = \vartheta(x)$ for $x \in X$
  - $(\vartheta \smile \phi)(y) = \phi(y)$ for $y \in Y$.

It is easy to see that the *union* of substitutions is a continuous mapping.

Let $d$ be a linear (i.e., with no multiple occurrences of the same variable) and $\perp$-free (i.e., with no occurrences of $\perp$) data term, and let $d'$ be a ground data term. The mapping $\Delta(d, d')$ (possibly) computes the substitution on $d$ which unifies $d$ and $d'$:

$$\Delta(d, d') = \begin{cases} \{x := d'\} & \text{if } d \text{ is the variable } x \\ \Delta(d_1, d'_1) \smile \cdots \smile \Delta(d_n, d'_n) & \\ & \text{if } d \text{ is } c(d_1, ..., d_n) \text{ and } d' \text{ is } c(d'_1, ..., d'_n) \\ \mathit{fail} & \text{otherwise.} \end{cases}$$

Note that for the correctness of the application of $\smile$ the linearity of $d$ is necessary. It is easy to see that $\Delta(d, d')$ maps into $S(\mathrm{Var}(d), CU)$ and if $\Delta(d, d') = \vartheta \neq \mathit{fail}$ then $d\vartheta = d'$. Since it is defined recursively, and by means of continuous operators (if-then-else and $\smile$), and since $\perp$ does not occur in $d$, $\Delta(d, d')$ is continuous with respect to $d'$. $\Delta$ can be extended on tuples: if $\boldsymbol{d} = d_1, ..., d_n$, $\boldsymbol{d}' = d'_1, ..., d'_n$, where $\boldsymbol{d}$ is linear and $\perp$-free, and $\boldsymbol{d}'$ is ground, then define

$$\Delta(\boldsymbol{d}, \boldsymbol{d}') = \Delta(d_1, d'_1) \smile \cdots \smile \Delta(d_n, d'_n).$$

Let $E$ be an expression (term, or atom), let $\vartheta \in S(\mathrm{Var}(E), CU)$ and let $I$ be a Herbrand interpretation. The mapping $\Gamma(E, \vartheta, I) = [\![E]\!]_{I\vartheta}$ is continuous with respect to $\vartheta$ and $I$, since, by definition of interpretation, it is obtained by composition of continuous mappings.

Let $\boldsymbol{B} = B_1, ..., B_n$ be a conjunction of S-LEAF body atoms. Assume that the sets $\mathrm{In}(\boldsymbol{B}) = \mathrm{In}(B_1) \cup \cdots \cup \mathrm{In}(B_n)$ and $\mathrm{Out}(\boldsymbol{B}) = \mathrm{Out}(\boldsymbol{B}) = \mathrm{Out}(B_1) \cup \cdots \cup \mathrm{Out}(B_n)$ are disjoint. Remember that every $B_i$ can be viewed as an equation $t_i = d_i$. Let $\vartheta \in S(\mathrm{In}(\boldsymbol{B}), CU)$ and let $I$ be a Herbrand interpretation. The following mapping (possibly) computes the substitution on output variables which make the equations true in $I$, depending on the substitution on input variables

$$\Psi(\boldsymbol{B}, \vartheta, I) = \begin{cases} \Delta(d_1, [\![t_1]\!]_{I\vartheta}) \smile \cdots \smile \Delta(d_n, [\![t_n]\!]_{I\vartheta}) & \text{if } \vartheta \neq \mathit{fail} \\ \mathit{fail} & \text{otherwise.} \end{cases}$$

Note that the use of $\smile$ is correct because of the constraints (b) (no multiple productions) and (f) (right linearity) on S-LEAF body atoms. $\Psi(\boldsymbol{B}, \vartheta, I)$ maps into $S(\mathrm{Out}(\boldsymbol{B}), CU)$ and it is continuous with respect to $\vartheta$ and $I$, since it is defined by composition of continuous mappings. Note that if $\Psi(\boldsymbol{B}, \vartheta, I) = \vartheta' \neq \mathit{fail}$ then $[\![\boldsymbol{B}]\!]_{I\vartheta'} = \mathit{true}$.

The following definition extends $\Psi$ to any conjunction $\boldsymbol{B} = B_1, ..., B_n$ of S-LEAF body atoms. Let $X_1, ..., X_{k+1}$ and $\boldsymbol{B}_1, ..., \boldsymbol{B}_{k+1}$ be the sequences such that

- $X_1 = \mathrm{In}(\boldsymbol{B}) - \mathrm{Out}(\boldsymbol{B})$
- $X_{i+1} = X_i \cup \mathrm{Out}(\boldsymbol{B}_i)$
- $\boldsymbol{B}_1$ is the greatest subconjunction of $\boldsymbol{B}$ such that $\mathrm{In}(\boldsymbol{B}_1) = X_1$
- $\boldsymbol{B}_{i+1} = \boldsymbol{B}'_{i+1} - \boldsymbol{B}'_i$, where $\boldsymbol{B}'_i$ is the greatest subconjunction of $\boldsymbol{B}$ such that $\mathrm{In}(\boldsymbol{B}'_i)$ is contained in $X_i$.

The sequence ends with $X_{k+1} = \mathrm{Var}(\boldsymbol{B})$; note that $\boldsymbol{B}_{k+1}$ is empty. Now, let $I$ be a Herbrand interpretation, let $\vartheta \in S(X_1, CU)$ and define $\Phi(\boldsymbol{B}, \vartheta, I) = \Phi_k$, where

- $\Phi_1 = \Psi(\boldsymbol{B}_1, \vartheta, I)$
- $\Phi_{i+1} = \Phi_i \smile \Psi(\boldsymbol{B}_{i+1}, \Phi_i, I)$.

The correctness of the use of $\smile$ is due to the constraints (a) (no cycles), (b), and (f) on S-LEAF body atoms. $\Phi(\boldsymbol{B}, \vartheta, I)$ maps on $S(\mathrm{Out}(\boldsymbol{B}), CU)$ and it is continuous with respect to $\vartheta$ and $I$, since it is defined by means of continuous operators and by iterating the composition of continuous mappings. If $\Phi(\boldsymbol{B}, \vartheta, I) = \vartheta' \neq \mathit{fail}$ then $[\![\boldsymbol{B}]\!]_{I\vartheta'} = \mathit{true}$.

Let $C : A \leftarrow \boldsymbol{B}$ be a S-LEAF clause. Let $\vartheta \in S(\mathrm{In}(A), CU)$. Let $I$ be a Herbrand interpretation. Let $\boldsymbol{B}'$ be the conjunction of the atoms of the body with definite inputs. Note that $\mathrm{In}(\boldsymbol{B}') - \mathrm{Out}(\boldsymbol{B}')$ is contained in $\mathrm{In}(A)$. Consider a substitution $\phi$ which extends $\vartheta$ on $X_1 = \mathrm{In}(\boldsymbol{B}) - \mathrm{Out}(\boldsymbol{B})$. By constraint (c), $\mathrm{Out}(A)$ is contained in $\mathrm{In}(A) \cup \mathrm{Out}(\boldsymbol{B})$, therefore if $\Phi(\boldsymbol{B}, \phi, I) \neq \mathit{fail}$ then $\Phi(\boldsymbol{B}, \phi, I)_{|\mathrm{Out}(A)} = \Phi(\boldsymbol{B}', \vartheta, I)_{|\mathrm{Out}(A)}$. Then define

$$\Xi(C, \vartheta, I) = \begin{cases} \Phi(\boldsymbol{B}, \phi, I)_{|\mathrm{Out}(A)} & \text{if } \vartheta \neq \mathit{fail} \text{ and there exists } \phi \\ & \text{which extends } \vartheta \text{ on } X_1 \text{ such that } [\![\boldsymbol{B}]\!]_{I,\phi} = \mathit{true}. \\ \mathit{fail} & \text{otherwise.} \end{cases}$$

From the above remark, there exist at most one $\phi$ which satisfies this definition. As usual, $\Xi(\boldsymbol{B}, \vartheta, I)$ is continuous with respect to $\vartheta$ and $I$.

Eventually, we give the following definition. Let $W$ be a S-LEAF program and let $f \in F$ [$f \in P$]. Let $\boldsymbol{d} = d_1, ..., d_n, \boldsymbol{d}' = d'_1, ..., d'_n$ be data terms, where $\boldsymbol{d}$ is linear and $\perp$-free, and $\boldsymbol{d}'$ is ground. Let $I$ be a Herbrand interpretation. The following mapping computes the output value of $f$ in $W$, depending on the input values $\boldsymbol{d}'$ and on $I$:

$$\Theta(f, \boldsymbol{d}', I) = \begin{cases} [\![t]\!]_{I\vartheta'} & \text{if there exists } C : f(\boldsymbol{d}) = t \leftarrow \boldsymbol{B} \text{ in } W \text{ such that} \\ & \Xi(C, \vartheta, I) = \vartheta' \neq \mathit{fail}, \text{ where } \vartheta = \Delta(\boldsymbol{d}, \boldsymbol{d}'). \\ \perp & \text{otherwise.} \end{cases}$$

Note that the correct use of $\smile$ is guaranteed by constraint (e) (left linearity). Finally, we obtain the following result which shows that S-LEAF clauses (with equational heads) can be interpreted as definitions of continuous functions.

PROPOSITION 2. *$\Theta(f, \boldsymbol{d}, I)$ is a mapping and it is continuous with respect to $\boldsymbol{d}$ and $I$.*

*Proof.* The condition of weak non-ambiguity on $W$ guarantees that if there exist two clauses such that $\Delta(\boldsymbol{d}, \boldsymbol{d}') \neq \mathit{fail}$ then they define the same value for $[\![t]\!]_{I\vartheta'}$. The definition by composition of continuous functions guarantees the continuity. ∎

## 5.2. *Fixpoint Semantics*

In this section we define a transformation $T$, associated to a S-LEAF program $W$, which works on Herbrand interpretations. $T$ can be viewed as an inference operator which allows to build, starting from $I_\perp$, more and more refined interpretations. The limit of this sequence, which results to be a model of $W$ (the minimal one), is the least fixpoint of $T$, and it is used to define the fixpoint semantics of $W$.

Definition 8 (Mapping $T$). Let $W$ be a S-LEAF program. Let $I$ be a Herbrand interpretation for $W$. $T(I)$ is the Herbrand interpretation such that, for each $f \in F$ and $p \in P$,

$$[\![f]\!]_{T(I)}(\boldsymbol{d}') = \begin{cases} [\![t]\!]_{I\vartheta} & \text{if } \exists\vartheta \in \text{Subst}(V, CU), \exists f(\boldsymbol{d}) = t \leftarrow \boldsymbol{B} \in W, \\ & \text{such that } [\![\boldsymbol{d}]\!]_{I\vartheta} = \boldsymbol{d}' \text{ and } [\![\boldsymbol{B}]\!]_{I\vartheta} = \mathit{true} \\ \perp & \text{otherwise;} \end{cases}$$

$$[\![p]\!]_{T(I)}(\boldsymbol{d}') = \begin{cases} \mathit{true} & \text{if } \exists\vartheta \in \text{Subst}(V, CU), \exists p(\boldsymbol{d}) \leftarrow \boldsymbol{B} \in W, \\ & \text{such that } [\![\boldsymbol{d}]\!]_{I\vartheta} = \boldsymbol{d}' \text{ and } [\![\boldsymbol{B}]\!]_{I\vartheta} = \mathit{true} \\ \perp & \text{otherwise.} \end{cases}$$

Note that

- in both cases, $\boldsymbol{d}$ is a tuple of data terms, therefore $[\![\boldsymbol{d}]\!]_{I\vartheta} = \boldsymbol{d}\vartheta$ holds,
- the definition of $[\![p]\!]_{T(I)}$ could be derived from the definition of $[\![f]\!]_{T(I)}$ and from the reading of $p(\boldsymbol{t})$ as $p(\boldsymbol{t}) = \mathit{true}$
- the definitions of $[\![f]\!]_{T(I)}$ and of $[\![p]\!]_{T(I)}$ closely resemble the definition of $\Theta$ (see Section 5.1).

The following result proves the correctness of the definition of $T(I)$.

Proposition 3. *For each $f \in F$ and $p \in P$, $[\![f]\!]_{T(I)}$ and $[\![p]\!]_{T(I)}$ are continuous mappings.*

*Proof.* Immediate by Proposition 2, since $[\![f]\!]_{T(I)}(\boldsymbol{d}) = \Theta(f, \boldsymbol{d}, I)$ and $[\![p]\!]_{T(I)}(\boldsymbol{d}) = \Theta(p, \boldsymbol{d}, I)$. ▌

The following theorem is essential in order to define the least fixpoint of $T$, which is used to define the least fixpoint semantics of a program $W$.

Theorem 5. *$T$ is continuous on the CPO $(HI, \leqslant)$.*

*Proof.* The same as Proposition 3.

Corollary 1. *$T$ has a least fixpoint* $\text{lfp}(T) = \min\{I \mid T(I) = I\}$. *Moreover,* $\text{lfp}(T) = \bigsqcup_{n \geqslant 0} T^n(I_\perp)$.

*Proof.* Standard, by continuity of $T$. ▌

Note that the relevance of this result consists not only in the existence of $\mathrm{lfp}(T)$, but also in the implicit definition of an algorithm for effectively generating $\mathrm{lfp}(T)$.

EXAMPLE 4. Consider the program of Example 3. We have

- $[\![nat]\!]_{I_\perp}(0) = \perp$, $[\![nat]\!]_{I_\perp}(s(0)) = \perp$, ..., $[\![first0(nat(0))]\!]_{I_\perp} = \perp$.
- $[\![nat]\!]_{T(I_\perp)}(0) = cons(0, \perp)$,

$[\![nat]\!]_{T(I_\perp)}(s(0)) = cons(s(0), \perp)$, ..., $[\![first0(nat(0))]\!]_{T(I_\perp)} = true$.

...

- $[\![nat]\!]_{\mathrm{lfp}(T)}(0) = cons(0, cons(s(0), ...))$,

$[\![nat]\!]_{\mathrm{lfp}(T)}(0) = cons(s(0), cons(s(0)), ...))$, ..., $[\![first0(nat(0))]\!]_{\mathrm{lfp}(T)} = true$.

DEFINITION 9. If $f$, $p$ are function and predicate symbols, then their least fixpoint semantics is $[\![f]\!]_{\mathrm{lfp}(T)}$ and $[\![p]\!]_{\mathrm{lfp}(T)}$, respectively. The *least fixpoint semantics* of an atom $A$ is

$$F(A) = \{A' \text{ ground} \mid \exists \vartheta \in \mathrm{Subst}(V, CU), A\vartheta = A' \text{ and } [\![A']\!]_{\mathrm{lfp}(T)} = true\}.$$

## 5.3. *Model-Theoretic Semantics*

In this section we show that the usual model-theoretic approach [19] can be applied to S-LEAF. We define the notion of model, and we discuss the relevance of the Herbrand models and the existence of a minimal Herbrand model for S-LEAF programs. Finally, we show the relation with the fixpoint semantics.

DEFINITION 10. Let $W$ be a set of S-LEAF clauses (definite clauses or goal statements).

- An interpretation $M = ((S, \leqslant), [\![\ ]\!]_M)$ of $W$ is a *model* if for each clause $C$ in $W$, and for each environment $\rho$: $V \to S$, $[\![C]\!]_{M\rho} = true$ holds.
- A Herbrand model is a Herbrand interpretation which is a model. The set of Herbrand models of $W$ is denoted by $HM$.

Let $E$ be an atom or a clause, $W$ be a set of S-LEAF clauses and $I$ be an interpretation. $W$ is *consistent* if it has at least a model. $E$ is *true* in $I$ if for every $\rho$, $[\![E]\!]_{I\rho} = true$. $E$ is a *logical consequence* of $W$ ($W \models E$) if it is true in every model of $W$, and it is an *inductive consequence* of $W$ ($W \models_H E$) if it is true in every Herbrand model of $W$.

The following result points out the corespondence between the Herbrand models of a program $W$ and the fixpoints of the associated transformation $T$, and it will allow derivation of some relevant properties about model-theoretic semantics.

PROPOSITION 4. $\{I \mid T(I) = I\} \subseteq HM \subseteq \{I \mid T(I) \leqslant I\}$ (i.e., every fixpoint of $T$ is a (Herbrand) model, and every Herbrand model is a closed-point of $T$).

*Proof.* In the following, a predicate or strict equation $A$ is considered as an equation $A = \text{true}$:

$\{I \mid T(I) = I\} \subseteq HM$) Assume that $T(I) = I$. Let $f(\boldsymbol{d}) = t \leftarrow \boldsymbol{B} \in W$, let $\vartheta \in \text{Subst}(V, CU)$ and $[\![\boldsymbol{B}]\!]_{I\vartheta} = \mathit{true}$. Let $\boldsymbol{d}\vartheta = \boldsymbol{d}'$. Then, $[\![t]\!]_{I\vartheta} = [\![f]\!]_{T(I)}(\boldsymbol{d}') = [\![f]\!]_I(\boldsymbol{d}') = [\![f(\boldsymbol{d})]\!]_{I\vartheta}$.

$HM \subseteq \{I \mid T(I) \leqslant I\}$) Assume that $I$ is a Herbrand model of $W$. Let $f$ be a function symbol. We have to show that, for each $\boldsymbol{d}'$ ground $[\![f]\!]_{T(I)}(\boldsymbol{d}') \leqslant [\![f]\!]_I(\boldsymbol{d}')$. If $[\![f]\!]_{T(I)}(\boldsymbol{d}') = \bot$ the result is immediate. Otherwise, let $f(\boldsymbol{d}) = t \leftarrow \boldsymbol{B}$ be the clause of $W$ and $\vartheta$ be the substitution such that $\boldsymbol{d}\vartheta = \boldsymbol{d}'$ and $[\![\boldsymbol{B}]\!]_{I\vartheta} = \mathit{true}$. Then $[\![f]\!]_{T(I)}(\boldsymbol{d}') = [\![t]\!]_{I\vartheta} = [\![f(\boldsymbol{d})]\!]_{I\vartheta} = [\![f]\!]_I(\boldsymbol{d}')$. ▮

In general the inclusions of Proposition 4 are strict. Note that this proposition points out the mixed nature (i.e., functional and logical) of our language. In fact, in functional programming usually the (Herbrand) models are the same as the fixpoints, while in logic programming usually the (Herbrand) models are the same as the closed-points.

LEMMA 3. *There exists the minimal closed-point* $\min\{I \mid T(I) \leqslant I\} = \min\{I \mid T(I) = I\} = \text{lfp}(T)$.

*Proof.* By Corollary 1, there exists $\text{lfp}(T) = \bigsqcup_{n \geqslant 0} T^n(I_\bot)$. Clearly $\text{lfp}(T)$ is a closed-point. Let $I$ be a closed-point, we have only to show that $\text{lfp}(T) \leqslant I$. By definition, $T(I) \leqslant I$, then, by monotonicity of $T$, for every $nT^n(I_\bot) \leqslant T^n(I \leqslant I$ holds. Therefore $\bigsqcup_{n \geqslant 0} T^n(I_\bot) \leqslant I$. ▮

THEOREM 6. *Every S-LEAF program has at least a Herbrand model* (*and then every S-LEAF program is consistent*). *Moreover, there exists the minimal Herbrand model* $M_{\min} = \text{lfp}(T)$.

*Proof.* By Corollary 1, there exists the minimal fixpoint $\text{lfp}(T)$, and, by Proposition 4, $\text{lfp}(T)$ is a model. Moreover, every other model $M$ is a closed-point, then, by Lemma 3, $\text{lfp}(T) \leqslant M$. ▮

The following results show that the Herbrand models, and, in particular, the minimal one, can be considered as the representatives of all the models.

LEMMA 4. *Let $W$ be a S-LEAF program and $t$ be a ground term. If $M$ is a model of $W$ then* $[\![ [\![t]\!]_{M_{\min}} ]\!]_M \leqslant [\![t]\!]_M$ holds.

*Proof.* Let $(S, \leqslant)$ be the CPO on which $M$ is based. First of all we note that the set $SI$ of interpretations on $S$ can be given the structure of CPO, by the same construction used for the Herbrand interpretations. In addition, for every $S$ we can define the mapping $T_S$ on $SI$, thus extending to general interpretations the notion of the transformation $T$ on $HI$ (in practice $T_S$ is an inference operator which computes the set of the equalities deducible in $W$ from an $S$-interpretation). It can be easily proved that $T_S$ is continuous and that Proposition 4 still holds. In particular, if $M$ is a $S$-model then $T_S(M) \leqslant M$. Moreover, from the definition of $\Theta$ (and from its extension on $SI$) we can derive the *crossed monotonicity* of $T$ and $T_S$, i.e., for every Herbrand interpretation $I$ and for every $S$-interpretation $I'$:

$$\text{if} \quad \forall t \text{ ground}, [\![ [\![t]\!]_I ]\!]_{I'} \leqslant [\![t]\!]_{I'} \qquad \text{then} \quad [\![ [\![t]\!]_{T(I)} ]\!]_{I'} \leqslant [\![t]\!]_{T_S(I')}.$$

Since $[\![\bot]\!]_M = \bot_S$, then $\forall t$ ground, $\forall n$, $[\![ [\![t]\!]_{T^n(\bot_S)} ]\!]_M \leqslant [\![t]\!]_{T_S{}^n(M)} \leqslant [\![t]\!]_M$. Since $[\![\ ]\!]_M$ is continuous, eventually $[\![ [\![t]\!]_{M_{\min}} ]\!]_M = [\![ [\![t]\!]_{\mathrm{lfp}(T)} ]\!]_M \leqslant [\![t]\!]_M$. ▌

THEOREM 7. *Let $W$ be a program, $\boldsymbol{B}$ be a conjunction of body atoms and $\vartheta \in \mathrm{Subst}(\mathrm{Var}(\boldsymbol{B}), D)$. The following sentences are equivalent*:

(1) $W \models B\vartheta$

(2) $W \models_H \boldsymbol{B}\vartheta$

(3) $M_{\min} \models B\vartheta$.

*Proof.* Obviously, (1) implies (2) and (2) implies (3). In order to show that (3) implies (1), note that all the equations of $\boldsymbol{B}\vartheta$ have the form $t = d$, where $d \in D$. Since $\mathrm{lfp}(T) \models t = d$, then $[\![t]\!]_{\mathrm{lfp}(T)} = [\![d]\!]_{\mathrm{lfp}(T)} = d$. By Lemma 4, for every $S$-model $M$, $[\![ [\![t]\!]_{\mathrm{lfp}(T)} ]\!]_M \leqslant [\![t]\!]_M$ holds. Therefore $[\![d]\!]_M \leqslant [\![t]\!]_M$. Finally, since $[\![\ ]\!]_M$ maps elements of $D$ into maximal elements of $S$, $[\![d]\!]_M = [\![t]\!]_M$ holds. ▌

*Remark* 2. It is possible to show that if $D$ is not empty, and if $d$ contains occurrences of $\bot$, then $W \not\models_H t = d$. Therefore we can consider $W \models \boldsymbol{B}\vartheta$ and $W \models_H \boldsymbol{B}\vartheta$ to be equivalent for every $\vartheta \in \mathrm{Subst}(\mathrm{Var}(\boldsymbol{B}), D_\bot)$.

The existence and the properties of the minimal model allows us to give the following definition of the *model-theoretic semantics.*

DEFINITION 11. If $f$, $p$ are function and predicate symbols, then their model-theoretic semantics is $[\![f]\!]_{M_{\min}}$ and $[\![p]\!]_{M_{\min}}$, respectively. The model-theoretic semantics of an atom $A$ is

$$M(A) = \{A' \text{ ground} \mid \exists \vartheta \in \mathrm{Subst}(V, CU).\ A\vartheta = A' \text{ and } [\![A']\!]_{M_{\min}} = \mathit{true}\}.$$

THEOREM 8. *The model-theoretic and the fixpoint semantics of an atom $A$ coincide, i.e.,*

$$M(A) = F(A)$$

*Proof.* Immediate by Theorem 6. ▌

We conclude this section by showing the correctness of the flattening algorithm.

THEOREM 9. *Let $W$ be a S-LEAF program and let $W'$ be the flat program obtained by flattening $W$. Then $W$ and $W'$ have exactly the same Herbrand models.*

The proof is based on the following lemma. For the sake of simplicity, a tuple of equations $t_1 = u_1, \ldots, t_n = u_n$ will be denoted by $\boldsymbol{t} = \boldsymbol{u}$, and a conjunction $\mathit{flatb}(B_1), \ldots, \mathit{flatb}(B_n)$ by $\mathit{flatb}(\boldsymbol{B})$.

LEMMA 5. *If $B$ is a body atom, and $I$ is a Herbrand interpretation for $B$, then*

(1) *if $[\![B]\!]_{I\vartheta} = true$, then there exists $\vartheta'$ which extends $\vartheta$ and such that $[\![\mathit{flatb}(B)]\!]_{I\vartheta'} = true$*

(2) *if $[\![\mathit{flatb}(B)]\!]_{I\vartheta'} = true$, then $[\![B]\!]_{I\vartheta} = true$, where $\vartheta$ is the restriction of $\vartheta'$ to the variables of $B$.*

*Proof.* As usual, all the cases can be reduced to the equational one. Let $B$ be $f(e_1[\boldsymbol{t}_1], \ldots, e_n[\boldsymbol{t}_n]) = d$, where $e_1, \ldots, e_n$ are data contexts, and $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$ tuples of functional terms. The proof is by structural induction:

(**$\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$ are empty**) (No functional subterms) In this case, $\mathit{flat}(B) = B$, and the result follows immediately.

(**$\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$ are not empty**) In this case, $\mathit{flatb}(B)$ is

$$\mathit{flatb}(\boldsymbol{t}_1 = \boldsymbol{x}_1), \ldots, \mathit{flatb}(\boldsymbol{t}_n = \boldsymbol{x}_n), f(e_1[\boldsymbol{x}_1], \ldots, e_n[\boldsymbol{x}_n]) = d,$$

where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are tuples of new variable symbols.

(1) Assume $[\![f(e_1[\boldsymbol{t}_1], \ldots, e_n[\boldsymbol{t}_n]) = d]\!]_{I\vartheta} = true$. Let

$$\vartheta' = \vartheta \smile \{\boldsymbol{x}_1 := [\![\boldsymbol{t}_1]\!]_{I\vartheta}, \ldots, \boldsymbol{x}_n := [\![\boldsymbol{t}_n]\!]_{I\vartheta}\}.$$

Of course, $[\![\boldsymbol{t}_1 = \boldsymbol{x}_1]\!]_{I\vartheta'} = true, \ldots, [\![\boldsymbol{t}_n = \boldsymbol{x}_n]\!]_{I\vartheta'} = true$. By the inductive hypothesis, there exists a ground substitution $\vartheta''$ which extends $\vartheta'$, such that

$$[\![\mathit{flatb}(\boldsymbol{t}_1 = \boldsymbol{x}_1)]\!]_{I\vartheta''} = true, \ldots, [\![\mathit{flatb}(\boldsymbol{t}_n = \boldsymbol{x}_n)]\!]_{I\vartheta''} = true.$$

Moreover, $[\![\boldsymbol{x}_1]\!]_{I\vartheta''} = [\![\boldsymbol{t}_1]\!]_{I\vartheta''}, \ldots, [\![\boldsymbol{x}_n]\!]_{I\vartheta''} = [\![\boldsymbol{t}_n]\!]_{I\vartheta''}$. Therefore,

$$[\![f(e_1[\boldsymbol{x}_1], \ldots, e_n[\boldsymbol{x}_n]) = d]\!]_{I\vartheta''} = true.$$

(2) Assume $[\![\mathit{flatb}(B)]\!]_{I\vartheta''} = true$. Then

$$[\![f(e_1[\boldsymbol{x}_n], \ldots, e_n[\boldsymbol{x}_n]) = d]\!]_{I\vartheta''} = true,$$

and

$$[\![\mathit{flatb}(\boldsymbol{t}_1 = \boldsymbol{x}_1)]\!]_{I\vartheta''} = true, \ldots, [\![\mathit{flatb}(\boldsymbol{t}_n = \boldsymbol{x}_n)]\!]_{I\vartheta''} = true.$$

By the inductive hypothesis, $[\![\boldsymbol{t}_1 = \boldsymbol{x}_1]\!]_{I\vartheta'} = true, \ldots, [\![\boldsymbol{t}_n = \boldsymbol{x}_n]\!]_{I\vartheta'} = true$, where $\vartheta'$ is the restriction of $\vartheta''$ to $\mathrm{Var}(B) \cup \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. Then $[\![\boldsymbol{x}_1]\!]_{I\vartheta'} = [\![\boldsymbol{t}_1]\!]_{I\vartheta'}, \ldots, [\![\boldsymbol{x}_n]\!]_{I\vartheta'} = [\![\boldsymbol{t}_n]\!]_{I\vartheta'}$, therefore

$$[\![f(e_1[\boldsymbol{t}_1], \ldots, e_n[\boldsymbol{t}_n]) = d]\!]_{I\vartheta} = true,$$

where $\vartheta$ is the restriction of $\vartheta'$ to $\mathrm{Var}(B)$. ∎

*Proof of Theorem* 9. Consider a clause $C$ and an interpretation $I$. We have to show that $I$ is a model of $C$ iff $I$ is a model of flatc($C$). As usual, we consider only the case of equational heads. If $C$ has the form $f(\boldsymbol{d}) = e[\boldsymbol{t}] \leftarrow \boldsymbol{B}$ (where $e$ is a data context), then *flatc*($C$) is

$$f(\boldsymbol{d}) = e[\boldsymbol{x}] \leftarrow \mathit{flatb}(\boldsymbol{t} = \boldsymbol{x}), \mathit{flatb}(\boldsymbol{B}).$$

(**if part**) Assume $[\![\boldsymbol{B}]\!]_{I\vartheta} = \mathit{true}$. Let $\vartheta' = \vartheta \smile \{\boldsymbol{x} := \boldsymbol{t}\}$. By Lemma 5, $[\![\mathit{flatb}(\boldsymbol{t} = \boldsymbol{x}), \mathit{flatb}(\boldsymbol{B})]\!]_{I\vartheta''} = \mathit{true}$, where $\vartheta''$ extends $\vartheta'$. Since $I$ is a model of *flatc*($C$),

$$[\![f(\boldsymbol{d}) = e[\boldsymbol{x}]]\!]_{I\vartheta''} = [\![f(\boldsymbol{d}) = e[\boldsymbol{x}]]\!]_{I\vartheta'} = [\![f(\boldsymbol{d}) = e[\boldsymbol{t}]]\!]_{I\vartheta} = \mathit{true}.$$

(**only if part**) Assume $[\![\mathit{flatb}(\boldsymbol{t} = \boldsymbol{x}), \mathit{flatb}(\boldsymbol{B})]\!]_{I\vartheta''} = \mathit{true}$. By Lemma 5, $[\![\boldsymbol{B}]\!]_{I\vartheta''} = \mathit{true}$. Since $I$ is a model of $C$, $[\![f(\boldsymbol{d}) = e[\boldsymbol{t}]]\!]_{I\vartheta''} = \mathit{true}$. Moreover, $[\![\boldsymbol{x}_1]\!]_{I\vartheta''} = [\![\boldsymbol{t}_1]\!]_{I\vartheta''}, ..., [\![\boldsymbol{x}_n]\!]_{I\vartheta''} = [\![\boldsymbol{t}_n]\!]_{I\vartheta''}$. Therefore, $[\![f(\boldsymbol{d}) = e[\boldsymbol{x}]]\!]_{I\vartheta''} = \mathit{true}$. ∎

## 6. Equivalence Results

In this section we show the relation between the operational and the declarative (fixpoint and model-theoretic) semantics of K-LEAF programs and Flat-LEAF programs. In the following, a relation or a strict equation $B$ is considered as an equation $B = \mathit{true}$.

### 6.1. *Soundness*

The equivalence result for K-LEAF programs differs from the standard (logic programming) one [36]. In fact the refutability of an equation does not imply that the equation is true, but only that its first member is greater than the second one. This is due to the presence of the equality clauses, which can be used also when other clauses are applicable. In fact, equality clauses force the second member of an equational atom to take the value $\perp$, while the program clauses generally force the declarative semantics of the first member to be more defined. Of course, in the case of a computed answer substitution containing no occurrences of $\perp$, the standard result (truth of the refutable equalities) still holds. In fact, in this case, every right part is maximal, and therefore disequalities become equalities. Note that, in this case, the use of the equality clauses does not affect the goal variables. For the same reason, since the computed answer substitutions of K-LEAF goals are $\perp$-free, the soundness result for K-LEAF is similar to the standard one.

Lemma 6. *Let $d$ be a linear and $\perp$-free data term, let $\vartheta$ be a ground substitution and let $d'$ be a ground data term such that $d\vartheta \leqslant d'$. Then $\Delta(d, d') \neq \mathit{fail}$ and $\vartheta \leqslant \Delta(d, d')$.*

*Proof.* Immediate by monotoniciy of $\Delta$ with respect to $d'$ (see Section 5.1). ∎

THEOREM 10 (Soundness of Flat-LEAF). *Let $W$ be a flat program and let $G\colon \leftarrow B_1, ..., B_m$ be a flat goal. Let each $B_i$ have the form $t_i = d_i$. Assume that $G$ has a refutation with c.a.s. $\vartheta$. Then, for each $B_i$, $d_i\vartheta \leqslant [\![t_i]\!]_{M_{\min}\vartheta}$ holds, i.e., for every ground substitution $\vartheta'$,*

$$d_i\vartheta\vartheta' \leqslant [\![t_i]\!]_{M_{\min}\vartheta\vartheta'}.$$

*Proof.* From Theorem 6 and Corollary 1 we have that $T^n(I_\perp) \leqslant \mathrm{lfp}(T) = M_{\min}$. Then it is sufficient to show, by induction on the length $n$ of the refutation, that $d_i\vartheta \leqslant [\![t_i]\!]_{T^n(I_\perp)\vartheta}$ holds.

($n = 1$) If the clause used in the refutation belongs to $W$ or to $=(W)$, then the proof is immediate. If it belongs to $\equiv(W)$, then the result follows from Remark 1.

($n \geqslant 1$) Assume (without loss of generality) that $B_1$ is the selected atom, and let $C\colon A \leftarrow B'_1, ..., B'_m$ be the clause of $W$, $=(W)$, or $\equiv(W)$, used for the first step of the refutation. Let $\vartheta_1$ be the m.g.u. of $B_1$ and $A$. The derived goal is

$$G' : \leftarrow (B'_1, ..., B'_k, B_2, ..., B_m)\,\vartheta_1.$$

Let $\vartheta_2$ be the c.a.s. of the rest of the refutation. By inductive hypothesis, for each $i = 2, ..., m$, $d_i\vartheta_1\vartheta_2 \leqslant [\![t_i\vartheta_1]\!]_{T^{n-1}(I_\perp)\vartheta_2}$ holds, therefore $d_i\vartheta \leqslant [\![t_i]\!]_{T^n(I_\perp)\vartheta}$ holds. We have still to show that the same result holds for $i = 1$. If $C$ belongs to $=(W)$ the result is immediate. Otherwise, consider the subgoal $\leftarrow (B'_1, ..., B'_k)\,\vartheta_1$, and let each $B'_j$ have the form $t'_j = d'_j$. By inductive hypothesis, $d'_j\vartheta \leqslant [\![t'_j]\!]_{T^{n-1}(I_\perp)\vartheta}$ holds. If $C$ belongs to $\equiv(W)$, then $d'_j = d = true$. Since *true* is maximal, then $[\![t'_j]\!]_{T^{n-1}(I_\perp)\vartheta} = true$ holds, and then the result $([\![t_1]\!]_{T^n(I_\perp)\vartheta} = [\![t'_j]\!]_{T^{n-1}(I_\perp)\vartheta} = \mathrm{true})$ follows by Remark 1. Consider now the last case; i.e., let $C$ belong to $W$. Note that, if $d$ is a data term, then for every Herbrand interpretation $I$ and for every substitution $\vartheta$, $[\![d]\!]_{I\vartheta} = d\vartheta$ holds. Therefore, from the inductive hypothesis, we derive $[\![d'_j]\!]_{T^{n-1}(I_\perp)\vartheta} \leqslant [\![t'_j]\!]_{T^{n-1}(I_\perp)\vartheta}$. Let $\vartheta'$ be a ground substitution and let $\sigma = (\vartheta\vartheta')_{|\mathrm{In}(A)}$. By Lemma 6 and because of the construction of $\Xi$ (see Section 5.1), if we define $\phi = \Xi(C, \sigma, T^{n-1}(I_\perp))$, then $\phi \neq \mathrm{fail}$ and $(\vartheta\vartheta')_{|\mathrm{Out}(A)} \leqslant \phi$. Assume that $A$ has the form $t = d$. By definition of $T$, we have $[\![t]\!]_{T^n(I_\perp)\vartheta} = d\phi$. Therefore, since $t_1\vartheta = t\vartheta$, we have

$$d\vartheta\,\vartheta' \leqslant d\phi = [\![t]\!]_{T^n(I_\perp)\vartheta\vartheta'} = [\![t_1]\!]_{T^n(I_\perp)\vartheta\vartheta'}. \quad \blacksquare$$

COROLLARY 2. *Let $W$ be a flat program and let $G\colon \leftarrow \boldsymbol{B}$ be a flat goal, where $\boldsymbol{B} = B_1, ..., B_m$. If $G$ has a refutation with computed answer substitution $\vartheta \in \mathrm{Subst}(V, D)$, then, for every $\vartheta' \in \mathrm{Subst}(V, D)$, $W \models \boldsymbol{B}\vartheta\vartheta'$ holds.*

*Proof.* Let each $B_i$ have the form $t_i = d_i$. From Theorem 10, for each $B_i$, we have $d_i\vartheta\vartheta' \leqslant [\![t_i]\!]_{M_{\min}\vartheta\vartheta'}$. Since $d_i\vartheta\vartheta'$ is maximal, $[\![t_i]\!]_{M_{\min}\vartheta\vartheta'} = d_i\vartheta\vartheta'$, and (again by maximality of $d_i\vartheta\vartheta'$) $W \models_H B_i\vartheta\vartheta'$ holds. Then, from Theorem 7, $W \models \boldsymbol{B}\vartheta\vartheta'$ holds. $\blacksquare$

THEOREM 11 (Soundness of K-LEAF). *Let $W$ be a K-LEAF program and let* $G : \leftarrow \boldsymbol{B}$ be a *K-LEAF goal. If flatc($G$) has a refutation in flat($W$), with a c.a.s. (restricted to the variables of $G$) $\vartheta$, then for every $\vartheta' \in$* Subst($V$, $D$), *$W \models \boldsymbol{B}\vartheta\vartheta'$ holds.*

*Proof.* From Proposition 1, $\vartheta \in \text{Subst}(V, D(V))$. *Then Corollary* 2 *applies.* ∎

## 6.2. *Completeness*

Both for Flat-LEAF and K-LEAF we can give a completeness result which is similar to the standard one [15, 36]. If a conjunction of equation atoms, under the application of a finite substitution $\vartheta$, is a logic consequence, then the corresponding goal has refutation with a c.a.s. more general than $\vartheta$. This result is satisfactory for K-LEAF, but it is not completely adequate for the Flat-LEAF case. In fact, in K-LEAF the possible infinite-valued functions are "hidden" in other terms, while the external members of a K-LEAF "equation" have always a finite value. On the contrary, in Flat-LEAF, the "true" equational atoms, containing a functional left member, are the ordinary case. Therefore the hypothesis of finiteness is too restrictive. Of course we cannot expect to be able to obtain an infinite c.a.s. with a (finite) refutation. However, we could investigate the possibility to approximate by c.a.s.'s any infinite value. A more general result, which is given in the following, deals with this possibility.

Let first recall some lemmas for SLD-resolution (the proofs can be found in [36]).

M.G.U. LEMMA. *An* unrestricted refutation *with answer $\vartheta$ is a refutation with c.a.s. $\vartheta$ (see Section* 4.4) *except that we drop the requirement that the substitution $\vartheta_i$ obtained at the* $i$th *step be a m.g.u; it is only required to be a unifier. Let $W$ be a program and let $G$ be a goal. If $G$ has an unrestricted refutation with answer $\vartheta$ then $G$ has a refutation with c.a.s. $\vartheta'$ more general than $\vartheta$, i.e., there exists $\sigma$ such that $\vartheta'\sigma = \vartheta$.*

LIFTING LEMMA. *Let $W$ be a program, let $G$ be a goal, and let $\vartheta$ be a substitution. If $G\vartheta$ has a refutation with c.a.s. $\sigma$ then $G$ has a refutation with c.a.s. $\vartheta'$ more general than $\vartheta\sigma$, i.e., there exists $\gamma$ such that $\vartheta'\gamma = \vartheta\sigma$.*

INDEPENDENCE FROM THE COMPUTATION RULE. *A computation rule is a function from definite goals to atoms such that the value of the function on a goal is an atom, called the* selected atom, *in that goal. Given a computation rule $R$, a refutation via $R$ of a goal $G$ is a refutation of $G$ in which at every step the atom selected for the resolution is given by $R$. If $G$ has a refutation with c.a.s. $\vartheta$, then, for any computation rule $R$, $G$ has a refutation via $R$ with a c.a.s. $\vartheta'$ which is a variant of $\vartheta$.*

One reason why the proof of completeness for K-LEAF and Flat-LEAF programs is more complex than the standard one is the presence of strict equations. In fact, they represent a special case of atoms, because their truth value is established in the definition of interpretation, and does not come from the program clauses of $W$ (which are the only ones involved in the definition of $T$). In practice, we have to show that the clauses of $\equiv(W)$ represent a complete set of axioms for strict equations.

THEOREM 12. *Let $B$ a strict equality (i.e., an atom of the form $t \equiv u$ or, equivalently, $(t \equiv u) = true$. For every substitution $\vartheta$, if $\forall(B\vartheta)$ is true in $M_{\min}$ then*

(1) *$\vartheta \in \mathrm{Subst}(\mathrm{Var}(B), D)$ (i.e., $\vartheta$ is ground and it maps into maximal elements).*

(2) *the goal $\leftarrow B$ has a refutation with c.a.s. $\vartheta$.*

*Proof.* (1) Immediate, since $\equiv$ can have the value *true* only on finite maximal elements, and in the Herbrand interpretations the strict equations can have the value *true* only if they are ground (a variable could be instantiated to $\perp$).

(2) By structural induction on $B\vartheta$. If $B\vartheta$ is $c \equiv c$, where $c$ is a data constant, then $B$ matches with the head of the clause $c \equiv c \leftarrow$ of $\equiv(W)$ and the m.g.u. $\vartheta$ is obtained. If $B\vartheta$ is $c(d_1, ..., d_m) \equiv c(d_1, ..., d_m)$, where $c$ is a data constructor and $d_1, ..., d_m$ are ground data terms, then $B$ matches with the head of a clause of $\equiv(W)$ of the form

$$c(x_1, ..., x_m) \equiv c(y_1, ..., y_m) \leftarrow x_1 \equiv y_1, ..., x_m \equiv y_m.$$

Let $\vartheta_1$ be the m.g.u. of $B$ and $c(x_1, ..., x_m) \equiv c(y_1, ..., y_m)$. Let $\vartheta_2$ be the substitution such that $x_i\vartheta_1\vartheta_2 = y_i\vartheta_1\vartheta_2 = d_i$ for each $i$. The $d_i$'s are structurally simpler than $B\vartheta$, then the inductive hypothesis applies and $\leftarrow (x_1 \equiv y_1, ..., x_m \equiv y_m)\,\vartheta_1$ has a refutation with c.a.s. $\vartheta_2$. Therefore, we get a refutation for $\leftarrow B$ with computed answer substitution $\vartheta_1\vartheta_2$. Moreover,

$$B\vartheta_1\vartheta_2 = (c(x_1, ..., x_m) \equiv c(y_1, ..., y_m))\,\vartheta_1\vartheta_2 = (c(d_1, ..., d_m) \equiv c(d_1, ..., d_m)) = B\vartheta. \quad \blacksquare$$

LEMMA 7 (Ground completeness). *Let $W$ be a flat program, let $G: \leftarrow B$ be a ground flat goal, and let $\vartheta \in Subst(Var(B), D_\perp)$. If $[\![B\vartheta]\!]_{M_{\min}} = true$, then $G\vartheta$ has a refutation.*

*Proof.* From Theorem 6 and Corollary 1 we have $M_{\min} = \mathrm{lfp}(T) = \bigsqcup_{n \geq 0} T^n(I_\perp)$. Moreover, since $\boldsymbol{B\vartheta}$ is finite, $[\![\boldsymbol{B\vartheta}]\!]_{\mathrm{lfp}(T)} = \mathrm{true}$ implies that there exists a $n$ such that $[\![\boldsymbol{B\vartheta}]\!]_{T^n(I_\perp)} = \mathrm{true}$. Therefore it is sufficient to show that, for every $n$, if $[\![\boldsymbol{B\vartheta}]\!]_{T^n(I_\perp)} = \mathrm{true}$ then $\leftarrow \boldsymbol{B}$ has a refutation. Let $\boldsymbol{B\vartheta}$ be $B_1, ..., B_m$. By induction on $n$,

($\mathbf{n=1}$) If $[\![\boldsymbol{B\vartheta}]\!]_{T^1(I_\perp)} = \mathrm{true}$, then for each $B_i$ of the form $t_i = d_i$ we have three cases:

(a) $t_i$ is a strict equation and $d_i$ is *true*,

(b) $d_i$ is $\perp$,

(c) there exists a unit clause $C$ in $W$ whose head unifies with $t_i = d_i$.

Because of Theorem 12, the atoms of case (a) are refutable, the atoms of case (b) can be refuted by means of a suitable clause in $=(W)$, and, in case (c), we obtain a refutation by using $C$. By composing all these derivations we have the global refutation for $G\vartheta$.

**(n > 1)** If $[\![ \boldsymbol{B}\vartheta ]\!]_{T^n(I_\perp)} = \text{true}$, then for each $B_i$ of the form $t_i = d_i$ we have again three cases:

(a) $t_i$ is a strict equation and $d_i$ is *true*,

(b) $d_i$ is $\perp$,

(c) there exists a clause $C: A \leftarrow B'_1, ..., B'_k$ in $W$ such that $A$ is unifiable with $t_i = d_i$ and, if $\sigma$ is the m.g.u., then $[\![ (B'_1, ..., B'_k)\, \gamma ]\!]_{T^{n-1}(I_\perp)} = \mathit{true}$ for some $\gamma$ which extends $\sigma$.

The first two cases are analogous to the ones in the induction base. For the last case we have to show that $\leftarrow t_i = d_i$ has a refutation. By applying $C$ we obtain the goal $G' : \leftarrow (B'_1, ..., B'_k)\, \sigma$. Let $\sigma'$ be the substitution such that $\sigma\sigma' = \gamma$. By inductive hypothesis the goal $\leftarrow (B'_1, ..., B'_k)\, \sigma\sigma'$ has a refutation, and therefore, from the lifting lemma, also $G'$ is refutable. ▌

LEMMA 8. *Let $W$ be a flat program, let $G : \leftarrow \boldsymbol{B}$ be a flat goal and let $\vartheta$ belong to* Subst(Var($\boldsymbol{B}$), $D_\perp(V)$). *If $W \models \forall(\boldsymbol{B}\vartheta)$ then $G\vartheta$ has a refutation with an empty c.a.s.*

*Proof.* Let $x_1, ..., x_m$ be the variables occurring in $\boldsymbol{B}\vartheta$. Let $a_1, ..., a_m$ be new data constants, and consider the alphabet of $W$ extended with $a_1, ..., a_m$. Let $W'$ denote the program $W$ on the extended alphabet. Consider the ground substitution

$$\sigma = \{x_1 := a_1, ..., x_m := a_m\}$$

and consider the conjunction $\boldsymbol{B}\vartheta\sigma$. Since $W \models \forall(\boldsymbol{B}\vartheta)$, then $W' \models \boldsymbol{B}\vartheta\sigma$, because every model of $W'$ is also a model of $W$. In particular, $[\![ \boldsymbol{B}\vartheta\sigma ]\!]_{M_{\min}} = \mathit{true}$ (where $M_{\min}$ is the minimal Herbrand model of $W'$). Therefore, by Lemma 7, $G\vartheta\sigma$ has a refutation in $W'$ (of course, with an empty c.a.s.). By the lifting lemma, $G\vartheta$ has a refutation in $W'$, and then in $W$, with a c.a.s. $\delta$ more general than $\delta$; i.e.,

$$\delta = \{x_{i_1} := a_{i_1}, ..., x_{i_k} := a_{i_k}\}$$

for a subset $\{i_1, ..., i_k\}$ of $\{1, ..., m\}$. Since the $z_i$'s do not actually occur either in $W$ (in $W'$), or in $G\vartheta$, no m.g.u. used in the refutation can actually contribute to any of the $x_{i_j} := a_{i_j}$. Therefore $\delta$ is empty. ▌

THEOREM 13 (Completeness of Flat-LEAF). *Let $W$ be a flat program, let $G : \leftarrow \boldsymbol{B}$ a flat goal, and let $\vartheta \in$* Subst(Var($\boldsymbol{B}$), $D_\perp$). *If $W \models \forall(\boldsymbol{B}\vartheta)$, then $G$ has a refutation with computed answer substitution $\sigma$ and $\exists\gamma$ such that $\sigma\gamma = \vartheta$.*

*Proof.* By Lemma 8, $G\vartheta$ has a refutation, and therefore, by lifting lemma, $G$ has a refutation with c.a.s. $\sigma$ more general than $\vartheta$; i.e., there exists $\gamma$ such that $\sigma\gamma = \vartheta$. ▌

The following theorem extends these results to the infinite case.

THEOREM 14 (Completeness for infinite atoms). *Let $W$ be a flat program, let $G :\leftarrow \boldsymbol{B}$ be a flat goal and let $\vartheta \in \mathrm{Subst}(\mathrm{Var}(\boldsymbol{B}), CU(V))$. If $W \models \forall(\boldsymbol{B}\vartheta)$, then $G$ has a refutation. Moreover, let $\boldsymbol{B}$ be $B_1, ..., B_m$ and $B_i$ be $t_i = d_i$. For every $\rho \in \mathrm{Subst}(\mathrm{Var}(\boldsymbol{B}\vartheta), CU)$, for every $d'_1, ..., d'_m \in D_\perp$ such that $[\![d'_i]\!]_M \leqslant [\![t_i\vartheta\rho]\!]_M$, $G$ has a refutation with c.a.s. $\sigma$ and $\exists\gamma$ such that $\sigma\gamma \leqslant \vartheta\rho$ and $d'_i \leqslant [\![d_i\sigma\gamma]\!]_M \leqslant [\![t_i\sigma\gamma]\!]_M \leqslant [\![t_i\vartheta\rho]\!]_M$.*

*Proof.* The proof is similar to the one of Theorem 13. We start considering the extended alphabet, as in Theorem 12, and the corresponding $\mathrm{lfp}(T)$. It is possible to show, by continuity of $T$, that there exists a finite substitution $\delta$ such that $d'_i \leqslant [\![d_i\delta]\!]_{\mathrm{lfp}(T)} = [\![t_i\delta]\!]_{\mathrm{lfp}(T)}$ and $\delta \leqslant \vartheta\rho$. Similarly to Theorem 12, $G\delta$ has a refutation with an empty c.a.s. By lifting lemma, $G$ has a refutation with c.a.s. $\sigma$ more general than $\delta$. Therefore, for a suitable $\gamma$, *$\sigma\gamma \leqslant \vartheta\rho$ holds.* Moreover, by Theorem 10, $[\![d_i\sigma]\!]_{M_{\min}} \leqslant [\![t_i\sigma]\!]_{M_{\min}}$, and therefore $[\![d_i\sigma\gamma]\!]_M \leqslant [\![t_i\sigma\gamma]\!]_M$.

THEOREM 15 (Completeness of K-LEAF). *Let $W$ be a K-LEAF program, let $G : \leftarrow \boldsymbol{B}$ be a K-LEAF goal, and let $\vartheta \in \mathrm{Subst}(V, D)$. If $W \models \forall(\boldsymbol{B}\vartheta)$, then flatc$(G)$ has a refutation with computed answer substitution $\sigma$ and $\exists\gamma$ such that $\sigma\gamma = \vartheta$.*

*Proof.* If $W \models \forall(\boldsymbol{B}\vartheta)$, then, by Lemma 5 and Theorem 9, *flat*$(W) \models \forall($*flatc*$(\boldsymbol{B})\,\vartheta')$ for a $\vartheta'$ which extends $\vartheta$ on the new variables introduced by the flattening. By Theorem 14, given a substitution $\rho$, there exists a refutation for *flatc*$(\boldsymbol{B})$ with a c.a.s. $\delta$ and $\exists\gamma$ such that $\delta\gamma \leqslant \vartheta'\rho$ and $d'_i \leqslant [\![d_i\sigma\gamma]\!]_M \leqslant [\![t_i\sigma\gamma]\!]_M \leqslant [\![t_i\vartheta\rho]\!]_M$ for every $t_i = d_i$ in $\boldsymbol{B}$. Since the $d_i$'s in $\boldsymbol{B}$ are *true*, the restriction $\sigma$ of $\delta$ to $\mathrm{Var}(\boldsymbol{B})$ is maximal and therefore we can find a substitution $\phi$ such that $\sigma\phi = \vartheta\rho$. Since this is true for every $\rho$, we have that $\sigma$ is more general than $\vartheta$. ∎

## 7. EXECUTION OF K-LEAF PROGRAMS

In this section we discuss the features of flattening +SLD-resolution (flat SLD-resolution, in the following) as the execution mechanism for K-LEAF. Moreover, we define an inference system which is more efficient than the one described in Section 4, but equivalent to it; i.e., soundness and completeness are still preserved. It consists of an *outermost* computation rule and of two new inference rules which replace, in some special cases, the resolution with the equality axioms.

In general, the computational methods that have been proposed for HCL-with-equality languages are based on SLD-resolution and narrowing. Both of them are linear strategies, obtained as refinements of resolution and completion, respectively. In particular, it is worth mentioning conditional narrowing [18, 22] and SLDE-resolution (i.e., SLD-resolution with syntactic unification replaced by a E-unification [26, 43]).

Flat SLD-resolution behaves better than narrowing with respect to several aspects. In fact,

- SLD-resolution was shown to be equivalent to "refined" narrowing [12], with a considerable gain in efficiency with respect to "ordinary" narrowing (elimination of redundant solutions and, more generally, reduction of the search space);

- the full (relational plus functional) language can be supported by a single inference mechanism;
- conditional equations can easily be handled, without need of extensions.

A critical point in our approach concerns the (apparent) loss of the producer–consumer information contained in the functional notation, caused by flattening. For example, in an atom of the form $p(f(x))$, the notation shows that $p$ depends on the value of $f(x)$. The flattening transforms $p(f(x))$ into two atoms $p(y)$, $f(x)=y$, which (apparently) are at the same level. However, the producer–consumer information is still implicitly present in the flat form and can be exploited by the selection strategy. In fact, in a K-LEAF goal, equational atoms cannot be present, and therefore every equational atom in the flat goal has the form $f(t)=y$, and can be considered a producer on the variable $y$. For the same reason, any other atom in which $y$ occurs (in the input part) can be considered a consumer on $y$ (in the above example, $p(y)$ is a consumer on $y$).

A selection rule corresponding to the *innermost rule* (in the unflattened program) can be easily implemented through the usual leftmost selection rule of Prolog [12], provided that flat literals are put in the right order by the flattening procedure. However, this strategy has a serious drawback in the unbounded possibility of using the clauses of the form $f(x_1, ..., x_n) = \bot$, which results in a large amount of useless computations. Moreover, if $x$ and $y$ are not produced by any other subgoal, then the resolution of a subgoal of the form $x \equiv y$ with a strict equality clause generates an infinite search tree.

The use of the clauses of the form $f(x_1, ..., x_n) = \bot$ is necessary for the completeness, except for the case of totally defined functions, as in [22]. The problem of their inefficient use could be overcome by noting that the resolution of an equational atom $f(d_1, ..., d_n) = z$ with $f(x_1, ..., x_n) = \bot$ is useful only if the atoms (if any) in which $z$ occurs as argument do not require a value for $z$; i.e., their resolutions bind $z$ to variables only. Unfortunately, in general this is not the case, and it cannot be statically detected. On the other hand, there exists a selection rule, the *outermost strategy*, which naturally avoids the unnecessary uses of $f(x_1, ..., x_n) = \bot$, without needing any statical detection. The *outermost strategy* is a mechanism analogous to lazy evaluation in functional programming. In fact, it selects an equational atom only when its produced variable should be bound to a non-variable term by a consumer atom, and, in this case, it uses the program clauses only. According to this selection strategy, the resolution of an equational literal against a clause $f(x_1, ..., x_n) = \bot$ is only necessary to eliminate the producers of variables which are no longer consumed by (i.e., do not occur in) any other atom. It may, therefore, be implemented as an *elimination rule*.

In order to avoid the possibility of infinite branches in the resolution of $x \equiv y$, strict equations can be handled in an ad-hoc way. Namely, in the case where $x$ and

$y$ are not produced, we can apply a sort of "fake" $x \equiv x$ clause (*s-equality rule*). This rule does not affect the soundness. In fact, there are no other atoms which can bind the two sides of the strict equality to terms denoting infinite or undefined objects. In the following we describe the outermost strategy.

DEFINITION 12. (i) An *outermost atom* (in a flattened goal) is

(1) a relational atom $p(d_1, ..., d_n)$,

(2) a strict-equality atom $d_1 \equiv d_2$,

(3) an equational atom $f(d_1, ..., d_n) = x$ whose produced variable $x$ does not occur elsewhere in $G$.

(ii) Let $A$ be a literal of the form $p(d_1, ..., d_n)[f(d_1, ..., d_n) = x]$ and $P$ be a flattened L-LEAF program, then *clauses*$(A, P)$ denotes the set of all clauses defining $p[f]$ in $P$.

THE ALGORITHM. A Pascal-like description of the algorithm is given in the following. The italic words are either variables or names defined in the program or in the paper. The underlined words are keywords. Plain texts are informal sentences.

```
resolve(Goal: K-LEAF goal; Program: K-LEAF program);
  $G := flatc(Goal)$;
  $P := flat(Program)$;
  while $G$ is not empty and there is no failure do
  — select-don't-care an outermost atom $A$ in $G$.
  — case
    — $A$ is an equational atom then eliminate it, i.e., $G := G\text{-}\{A\}$
    — $A$ is a relational atom then
      execute demand-driven-resolution of $A$ using clauses$(A, P)$ w.r.t. $G$ and $P$
      if demand-driven-resolution returns a new goal $G'$ then $G := G'$;
                                                             else failure endif;
    — $A$ is a strict equality atom then
      execute strict-equality-resolution of $A$ w.r.t. $G$ and $P$
      if strict-equality-resolution returns a new goal $G'$ then $G := G'$;
                                                            else failure endif;
    end case
  end while
  if $G$ is empty
    then success, and return the solution (i.e., the answer substitution is the com-
          position of all the mgu's used in the resolution steps, restricted to
          variables in Goal)
    else return failure endif.
  end
```

```
Demand-driven-resolution (ddr)(A atom; G flat goal; P flat program);
  select-don't-know in clauses(A, P) a clause cl such that A and the head of cl are unifiable
    if there is no such clause
      then return failure
      else let $\sigma := mgu(A$, head of $cl)$;
        case
          $\sigma$ does not bind to non-variable terms variable produced by other atoms
            then apply ordinary resolution to A and cl, and return the obtained goal;
          $\sigma$ binds to non-variable term a variable $z$ produced by an equational atom $t = z$:
            then (resolution of A suspended)
              execute ddr of $t = z$ using $clauses(t = z, P)$ w.r.t. G and P
              case
                it returns failure then return failure;
                it returns a goal $G'$, where the atom A has been instantiated to $A\tau$
                  then
                    execute ddr of $A\tau$ using $\{cl\}$ w.r.t. $G'$ and P
                    case
                      it returns failure then return failure;
                      it returns a goal $G''$ then return $G''$
                    end case
              end case
        end case
    end if
end
```

```
strict-equality-resolution (ser) (A strict equality atom; G flat goal; P flat program);
  case
    A is $c(d_1, ..., d_n) \equiv c'(d'_1, ..., d'_m)$
      then apply ordinary resolution with $a \equiv$ -clause;
        if resolution fails then return failure else return the obtained goal end if
        (i.e., if $c$ is the same constructor as $c'$ then return the goal
        $G - \{c(d_1, ..., d_n) \equiv c'(d'_1, ..., d'_n)\} + \{d_1 \equiv d'_1, ..., d_n \equiv d'_n\}$; else failure);
    A is $x_1 \equiv x_2$ where both $x_1$ and $x_2$ are non-produced variables
      then unify them (that results in binding $\tau$ of $x_1$ to $x_2$ or vice versa)
        and eliminate A, i.e., return the goal $(G - \{A\})\tau$ (s-equality rule)
    A is $x \equiv c(d_1, ..., d_n)$ or $c(d_1, ..., d_n) \equiv x$, where $x$ is a non-produced variable
      then apply ordinary resolution to A and $a \equiv$ -clause
        and return the goal thus obtained;
    A is $x \equiv d$ or $d \equiv x$, where $x$ is variable produced by an equational atom $t = x$
      then execute ddr of $t = x$ using clauses $(t = z, P)$ w.r.t. G and P;
```

```
        case
          it returns failure then return failure
          it returns a new goal $G'$, where the atom $A$ has been instantiated to $A\tau$
            then execute set of $A\tau$ w.r.t. $G'$ and $P$
              case it returns failure then return failure;
                  it returns a goal $G''$ then return $G''$
              end case
        end case
    end case
end
```

Note that this strategy does not bind any variable to terms containing $\perp$, i.e., any substitution $\sigma$ computed by *resolve* is $\sigma: V \to D(V)$ instead of $\sigma: V \to D_\perp(V)$.

EXAMPLE. Consider the following K-LEAF program, which is already in flat form,

(1) $p(x, y) \leftarrow q(x), x \equiv a.$

(2) $q(a).$

(3) $f(b, x) = a,$

where $a$ and $b$ are constructors.

Consider the goal $\leftarrow p(f(x, y), g(x)), y \equiv r$. The flattened form is $\leftarrow p(v1, v2)$, $f(x, y) = v1$, $g(x) = v2, y \equiv r$. We can select the outermost atom $p(v1, v2)$, which, resolved with (1), gives the goal

$$\leftarrow g(v1), v1 \equiv a, \qquad f(x, y) = v1, \qquad g(x) = v2, y \equiv r.$$

The equational atom $g(x) = v2$ can be eliminated, since the produced variable $v2$ does not appear elsewhere in the goal. Then we obtain the goal

$$\leftarrow q(v1), v1 \equiv a, \qquad f(x, y) = v1, y \equiv r.$$

Resolution of $q(v1)$ is suspended, since (2) would bind the variable $v1$, produced by the atom $f(x, y) = v1$, to the non-variable term $a$. Then the execution of $f(x, y) = v1$ is selected, thus generating the goal

$$\leftarrow q(a), a \equiv a, y \equiv r.$$

Now resolution of $q(a)$ (i.e., $q(v1)$, where $v1$ is bound to $a$) with the clause (2) can be resumed, thus obtaining the goal

$$\leftarrow a \equiv a, y \equiv r.$$

After the trivial strict equality resolution of $a \equiv a$, we obtain

$$\leftarrow y \equiv r.$$

Finally, we resolve $y \equiv r$ by the $s$-equality rule, thus obtaining the empty goal. The computed answer substitution is $\{x := b, y := r\}$.

The proposed strategy is an extension to the conditional case of Reddy's *lazy narrowing* [40]. One difference is in the computation of an (unflattened) atom $e \equiv x$. Flattening + outermost SLD-resolution evaluates $e$ until it becomes a data-term. On the contrary, lazy narrowing immediately produces the substitution $\{x := e\}$, which may not be a solution of the equation $e \equiv x$ in the intended semantics.

An advantage of outermost SLD-resolution with respect to lazy narrowing is that it gets for free the sharing of subexpressions derived from a common expression. In the previous example the equational atom $f(x, y) = v1$ is shared by the relational atom $q(v1)$ and the strict equality atom $v1 \equiv a$. Neither completeness nor soundness of that SLD-resolution hold for *resolve*.

THEOREM 16 (Completeness of outermost resolution). *Let $W$ be a K-LEAF program and $G$ a K-LEAF goal. If $G\sigma$, with $\sigma: V \rightarrow D$, is a logic consequence of $W$, then there is a substitution $\sigma'$ computed by resolve$(G, W)$, such that $\sigma'$ is not less general than $\sigma$, i.e., for some substitution $\tau$, $\sigma'_{|\mathrm{Var}(G)}\tau = \sigma$.*

*Proof.* The proof can be derived in two steps. The first one considers the atom selection strategy only, where resolutions against equality and strict equality axioms are not yet replaced with elimination and $s$-equality rules. Since it concerns only the computation rule, the completeness and correctness results for SLD-resolution in Section 6 can be applied. The second step considers the two new inference rules. It is easy to show (see the table below) that completeness is not affected, since the substitutions computed by these rules are more general than those computed by the equality axioms.

| Atom | New inference rules | Equality axioms |
|---|---|---|
| $f(d_1, ..., d_n) = x$ | $\sigma$ is empty | $\sigma$ is (at least) $\{x := \perp\}$ |
| $x \equiv y$ | $\sigma$ is $\{x := y\}$ | We obtain a $\sigma = \{x := d; y := d\}$ for any $d$ in $D$ ∎ |

THEOREM 17 (Soundness of outermost resolution). *Let $W$ be a K-LEAF program and $G$ a K-LEAF goal. If $\rho$ is an answer computed by resolve$(G, W)$, then for each Herbrand model $I$ of $W$, and for each substitution $\vartheta$: $\mathrm{Var}(G\rho) \rightarrow D$, $G\rho\vartheta$ is* true *in $I$.*

*Proof.* As in the previous theorem, the proof can be derived in two steps. The first one is analogous to the completeness case. The second one is less trivial than the one of completeness because the answers computed by the outermost strategy are more general. We need the following lemma.

LEMMA 9. *Let W be a K-LEAF program and G a K-LEAF goal. Let G' be the current goal and $\sigma$ be the composition of all the mgu's computed at any step of the computation of resolve(G, W). Then the following property holds*:

> *For any variable x not produced by any equational atom occurring in the previous current-goals, for any variable y in* $\mathrm{Var}(\sigma(x))$, *y is still not produced.*

*Proof.* By induction on the number of steps. The induction step is based on

- left-linearity of patterns in clause heads,
- the condition (d) on the syntax of Flat-LEAF clauses,
- the condition that the $s$-equality rule can be applied only if the two terms are non-produced variables. ▌

COROLLARY 3. *Let W be a K-LEAF program and G a K-LEAF goal. Let G' and $\sigma$ be the current goal and the composition of all the mgu's computed at any step of the computation by resolve(G, W). The following property holds*:

> *for any x in* $\mathrm{Var}(G)$, *for any y in* $\mathrm{Var}(\sigma(x))$, *y is not a produced variable.*

*Proof.* Immediate from the previous lemma and by noting that the only produced variables in flatc($G$) are the new variables introduced by the flattening procedure. ▌

*Proof of Theorem* 17.

ELIMINATION RULE. We can apply this rule to $f(d_1, ..., d_n) = x$ only if $x$ does not occur in the rest of the goal. Therefore the computed answer is equal to that computed by resolving all the eliminated goals of the form $f(d_1, ..., d_n) = x$ with $f(x_1, ..., x_n) = \perp$, because the binding $x := \perp$ is not "visible" to the non-produced variables, and then, because of the previous lemma, it is not "visible" to the variables in the original (unflattened) goal. See also property 1.

$s$-EQUALITY RULE. Let $\sigma$ be a solution computed by the procedure *resolve*, then $\sigma(x) = \sigma(y)$. By replacing each $s$-equality rule application with a resolution step against strict-equality-clauses for any $\vartheta$: $\{x, y\} \to D$ such that $\vartheta(x) = \vartheta(y)$ and $\vartheta \leqslant \sigma$, we obtain the solution $\vartheta\sigma'$, where $\sigma' = \sigma$ restricted to $\mathrm{Dom}(\sigma) - \{x, y\}$. Since $\vartheta$ is ground, $\vartheta\sigma' = \vartheta\sigma \leqslant \sigma$. Therefore all the ground instances (in $D$, and with respect to $\{x, y\}$) of $\sigma$ are correct, and then $\sigma$ is correct (with respect to $\{x, y\}$). ▌

## 8. CONCLUSIONS

In this paper we have presented the syntax, the operational, and the declarative semantics of the logical plus fuctional language K-LEAF. We have discussed the advantages of the technique on which this language is based (flattening instead of narrowing) and we have shown how it is possible to describe infinite processes, i.e., programs producing and/or consuming infinite data structures. This feature is carefully formalized, from a semantics point of view, by means of domains having a CPO structure. The soundness and completeness of the operational semantics are then proved. Finally, we have given an optimized algorithm for computing K-LEAF programs, based on an outhermost strategy.

The sequential (outermost) computational model of K-LEAF has been implemented on an (extended) Warren Abstract Machine [13]. An OR-parallel implementation for a distributed architecture is now under development, and we are also investigating an AND-parallel computational model [23].

K-LEAF has been developed in the context of the subtask D of the ESPRIT Project 415. The main purpose of this subtask is the design and the implementation of a higher-order logic plus functional language, called IDEAL [7, 9, 11]. This language should offer in a unified and coherent environment the most appealing features of Prolog and of the modern functional languages: full invertibility, nondeterminism, higher-order functions and predicates, lazy-evaluation, and typing. Besides being a rich extension of Prolog, K-LEAF has been regarded as a high-level intermediate language for the implementation of IDEAL.

## Acknowledgments

We would like to thank Peter Weijland and Maurizio Martelli for his detailed comments on this paper. This work has been partially supported by EEC within the ESPRIT Project 415, "Parallel Architectures and Languages for Advanced Information Processing: a VLSI-Directed Approach."

## References

1. K. R. Apt, "Introduction to Logic Programming," Tech. Rep. CS-R8741, CWI, Amsterdam, 1987; Handbook of Theoretical Computer Science," North-Holland, Amsterdam, to appear.
2. R. Barbuti, M. Bellia, G. Levi, and M. Martelli, On the integration of logic programming and functional programming, "Proc. 1984 Int. Symp. on Logic Programming," *in* IEEE Comput. Soc. Press, Rockville, MD, 1984. pp. 160–166.
3. R. Barbuti, M. Bellia, G. Levi, and M. Martelli, LEAF: A language which integrates logic, equations and functions, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), Englewood Clifts, NJ, pp. 201–238, 1986.
4. H. P. Barendregt, "The Lambda-Calculus: Its Syntax and Semantics," rev. ed., North-Holland, Amsterdam, 1984.
5. M. Bellia, E. Giovannetti, G. Levi, and C. Moiso, "The Relation between Logic and Functional Languages," ESPRIT Project 415, First year report, 1985.
6. M. Bellia and G. Levi, The relation between logic and functional languages: A survey, *J. Logic Programming* **3** (1986), 185–215.
7. M. Bellia, P. G. Bosco, E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi, A two level approach to logic plus functional programming integration, *in* "Proc. Conference on Parallel Architectures and Languages Europe (PARLE)," pp. 374–393, Springer-Verlag, New York/Berlin, 1987.

8. J. A. BERGSTRA AND J. W. KLOP, Conditional rewrite rules: Confluence and termination, *J. Comput. System Sci.* **32** (1986), 323–362.
9. P. G. BOSCO AND E. GIOVANNETTI, IDEAL, An ideal deductive applicative language, *in* "Proc. 1986 Symp. on Logic Programming," pp. 89–94, IEEE Comput Soc. Press, Rockville, MD, 1986.
10. P. G. BOSCO, E. GIOVANNETTI, AND C. MOISO, A completeness result for a semantic unification algorithm based on conditional narrowing, *in* "Foundations of Logic and Functional Programming" (M. BOSCAROL, M. CARLUCCI AIELLO, AND G. LEVI, Eds.) Lect. Notes in Comput. Sci., Vol. **306**, pp. 157–167, Springer-Verlag, New York/Berlin, 1987.
11. P. G. BOSCO AND E. GIOVANNETTI, A Prolog-compiled higher-order functional and logic language, *in* "Proc. AIMSA '86," North-Holland, Amsterdam, 1986.
12. P. G. BOSCO, E. GIOVANNETTI, AND C. MOISO, Refined strategies for semantic unification, *in* "Proc. TAPSOFT '87," Lect. Notes in Comput. Sci., Vol. **150**, pp. 276-290, Springer-Verlag, 1987; *Theoret. Comput. Sci.* **59** (1988), 3–23.
13. P. G. BOSCO, C. CECCHI, AND C. MOISO, An extension of WAM for K-LEAF: A WAM-based compilation of conditional narrowing, *in* "Proc. Sixth Conf. on Logic Programming, Lisbon, 1989," pp. 318–333.
14. D. BRAND, Proving theorems with the modification method, *SIAM J. Comput.* **4** (1975), 412–430.
15. K. L. CLARK, Negation as failure, *in* "Logic and Data Bases" (H. Gallaire and J. Minker Eds.), pp. 293–322, Plenum, New York, 1978.
16. K. L. CLARK AND S. GREGORY, PARLOG: Parallel programming in logic, *ACM Trans. Programm. Lang. Syst.* **8** (1986), 1–49.
17. P. T. COX AND T. PIETRZYKOWSKI, Surface deduction: A uniform mechanism for logic programming, *in* "Proc. 1985 Symp. on Logic Programming," pp. 220–227, IEEE Comput. Soc. Press, Rockville, MD, 1985.
18. N. DERSHOWITZ AND D. A. PLAISTED, Logic programming cum applicative programming, *in* "Proc. 1985 Symp. on Logic Programming," pp. 54–66, IEEE Comput. Soc. Press, Rockville, MD, 1985.
19. M. H. VAN EMDEN AND R. A. KOWALSKI, The semantics of predicate logic as a programming language, *J. Assoc. Comput. Mach.* **23** (1976), 733–742.
20. M. FAY, First order unification in an equational theory, *in* "Proc. 4th Workshop on Automated Deduction," pp. 161–167.
21. L. FRIBOURG, Oriented equational clauses as a programming language, *J. Logic Programming* **1** (1984), 165–177.
22. L. FRIBOURG, SLOG: A logic programming language interpreter based on clausal superposition and rewriting, *in* "Proc. 1985 Symp. on Logic Programming," pp. 172–184, IEEE Comput. Soc. Press, Rockville, MD, 1985.
23. E. GIOVANNETTI, G. LEVI, C. MOISO, AND C. PALAMIDESSI, "Kernel LEAF: An Experimental Logic plus Functional Language—Its Syntax, Semantics and Computational Model," ESPRIT Project 415, Second year report, 1986.
24. J. A. GOGUEN, J. W. THATCHER, E. WAGNER, AND J. B. WRIGHT, Initial algebra semantics and continuous algebras, *J. Assoc. Comput. Mach.* **24** (1977), 68–95.
25. J. A. GOGUEN AND J. MESEGUER, Equality, types, modules and (why not?) generics for logic programming, *J. Logic Programming* **1** (1984), 179–210.
26. J. A. GOGUEN AND J. MESEGUER, Quality, types and generic modules for logic programming, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), Englewood Cliffs, NJ, Prentice-Hall, pp. 295–364, 1986.
27. J. A. GOGUEN, One, none, a hundred thousand specification languages, *in* "Proc. IFIP '86," pp. 995–1004, Elsevier Science, New York, 1986.
28. G. HUET, Formal structures for computation and deduction, draft, May 1986.
29. J.-M. HULLOT, Canonical forms and unification, *in* "Proc. 5th Conf. on Automated Deduction," Lect. Notes in Comput. Sci., Vol. **87**, pp. 318–334, Springer Verlag, New York/Berlin, 1980.

30. H. HUSSMANN, "Unification in Conditional-Equational Theories," Report MIP-8502, Universitaet Passau, 1985.
31. J. JAFFAR, J.-L. LASSEZ, AND M. J. MAHER, A theory of complete logic programs with equality, *J. Logic Programming* **1** (1984), 211–223.
32. J. JAFFAR, J.-L. LASSEZ, AND M. J. MAHER, A logic programming language scheme, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), pp. 441–468, Prentice-Hall, Englewood Cliffs, NJ, 1986.
33. J. JAFFAR, J.-L. LASSEZ, AND M. J. MAHER, Some issues and trends in the semantics of logic programming, *in* "Proc. of Third Int'l Conf. on Logic Programming," Lect. Notes in Comput. Sci., Vol. **225**, pp. 223–241, Springer-Verlag, New York/Berlin, 1986.
34. J.-L. LASSEZ AND M. J. MAHER, Optimal fixedpoints of logic programs, *Theoret. Comput. Sci.* **39** (1985), 15–25.
35. G. LEVI, C. PALAMIDESSI, P. G. BOSCO, E. GIOVANNETTI, AND C. MOISO, A complete semantics characterization of K-LEAF, a logic language with partial functions, *in* "Proc. 1987 Symp. on Logic Programming," pp. 318–327, IEEE Comput. Soc. Press, Rockville, MD, 1987.
36. J. W. LLOYD, "Foundations of Logic Programming," rev. ed., Springer-Verlag, New York/Berlin, 1987.
37. S. KAPLAN, "Fair Conditional Term Rewriting Systems: Unification, Termination, and Confluence," Technical Report 194, University of Orsay, 1984.
38. J. W. KLOP, Working material for the seminar on reduction machines, Ustica, Italy, Sept. 1985.
39. W. A. KORNFELD, Equality for PROLOG, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), pp. 279–294, Prentice-Hall, Englewood Cliffs, NJ, 1986.
40. U. S. REDDY, Narrowing as the operational semantics of functional languages, *in* "Proc. 1985 Symp. on Logic Programming," pp. 138–151, IEEE Comput. Soc. Press, Rockville, MD, 1985.
41. U. S. REDDY, On the relationship between logic and functional languages, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), pp. 3–36, Prentice-Hall, Englewood Cliffs, NJ, 1986.
42. P. RETY, C. KIRCHNER, H. KIRCHNER, AND P. LESCANNE, NARROWER: A new algorithm for unification and its application to logic programming, *in* "Proc. First Int. Conf. on Rewriting Techniques and Applications, Lect. Notes in Comput. Sci., Vol. **202**, pp. 141–157, Springer-Verlag, New York/Berlin, 1985.
43. P. A. SUBRAHMANYAM AND J.-H. YOU, FUNLOG: A computational model integrating logic programming and functional programming, *in* "Logic Programming: Functions, Relations and Equations" (D. DeGroot and G. Lindstrom, Eds.), pp. 157–198, Prentice-Hall, Englewood Cliffs, NJ, 1986.
44. H. TAMAKI, Semantics of a logic programming language with a reducibility predicate, *in* "Proc. of the 1984 Symp. on Logic Programming," pp. 259–264, IEEE Comput. Soc. Press, Rockville, MD, 1984.