On the Declarative Semantics of Horn Clause Logic-Based Languages*

Catuscia Palamidessi
Dipartimento di Informatica
Corso Italia 40, 56100 Pisa, Italy

The paper presents a scheme for defining the declarative semantics of real logic languages whose operational behaviour can be modelled by partial orders. We delineate a uniform framework in which the construction of van Emden and Kowalski can be extended while preserving their basic results (existence of a minimal model, equivalence with the fixpoint semantics, and equivalence with the operational semantics based on SLD-resolution). In practice, we delineate a 'semantics theory scheme', parametric with respect to some notions depending on the particular language. It consists of some fixed properties to be verified in order to obtain the above mentioned results. We also investigate the possibility of getting stronger results. Then, we show some instances of this scheme: the semantics of a logic+functional language, the semantics of a class of concurrent logic languages, and the (extended) semantics of pure Horn Clause Logic.

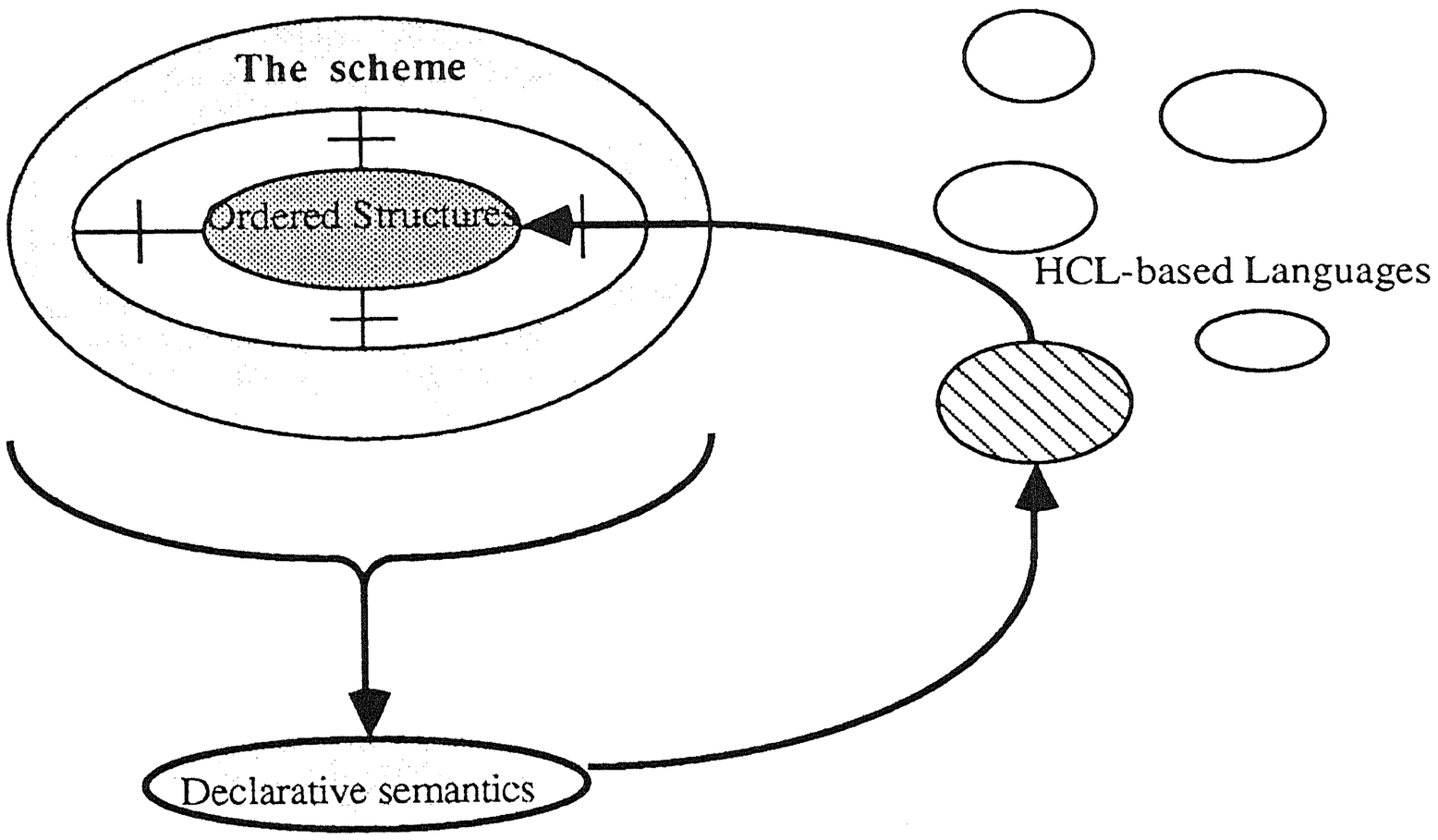
1. Introduction

Logic languages, i.e. languages based on Horn Clause Logic (HCL), have become very popular in the last few years, mainly because of their high level nature. In fact, HCL is based on assertions that have declarative interpretations. In other words, they can be seen as a formal description of the problem to be solved and they can be understood without any reference to the behaviour of any particular machine. Moreover, HCL has the advantage of being founded on a rigorous and well-established mathematical theory (a subset of the First Order Logic). This allowed van Emden and Kowalski to define a clear, simple and elegant formal semantics [7], based on the results of Tarski and Herbrand. Unfortunately, the real logic languages, such as Prolog, the concurrent logic languages, and the functional+logic languages, present some characteristics that make the above semantics inadequate. In this paper we analyse the possibility of extending the construction of van Emden and Kowalski to real logic languages. We use a two-level approach, in the sense that we use some methodologies typical of the denotational approach (in particular, partial orderings) in the framework of the declarative approach. The

* Part of this research was done while the author was staying at CWI. Her permanence at CWI was financially supported by the Consiglio Nazionale delle Ricerche.

Copyright © 1989, Stichting Mathematisch Centrum, Amsterdam
Printed in the Netherlands

behaviour of many real logic languages (but also of pure HCL), indeed, can be described in terms of 'ability to produce information', and therefore the mathematical notions of denotational semantics can be applied. We investigate a sort of 'general methodology' for defining the semantics in such a way that the standard results of van Emden and Kowalski (existence of a minimal model, equivalence with the fixpoint semantics, and equivalence with the operational semantics based on SLD-resolution) are preserved. Moreover, we study the possibility of deriving more powerfull results. In practice, we delineate a 'semantics theory scheme', parametric with respect to some notions (such as the ordered structures underlying the semantic domains, the notion of truth, etc.). This scheme can be applied to different languages, in order to obtain their semantics, by defining the above mentioned notions, and by verifying some fixed properties.



The basic ideas of our approach are similar to the ones developed in [11,12,13] for HCL on equalities theories, but they are more general. In fact, our approach is based on the notion of partial ordering, that includes the notion of equality as a subcase. This is particularly clear in the case of the logic+functional languages. As Section 4 shows, our scheme is applicable not only to languages interpreted on algebraic structures (i.e. equality models), but also to the ones interpreted on more complex structures (like algebraic Complete Partial Orders). In general, our scheme is applicable to all those languages in which the operational difference between goals can be described in terms of 'different amount of information that they can produce' and this can be modeled by an ordering. The concurrent logic languages whose

synchronization mechanism is based on the producer-consumer relation between atoms are instances of this paradigm. Finally, also in the case of the pure Horn Clause Logic, our scheme gives some interesting results. Namely, it gives a declarative semantics more powerful than the standard one, in the sense that it allows to get a stronger completeness theorem.

2. The Horn Clause Logic

We recall some basic notions about HCL. See [21,1] for details.

The language alphabet is given by a set D of *constructors symbols* $a,b,c,...$, a set P of *predicate symbols* $p,q,r,...$, and a set V of variable symbols $x,y,z,....$ We denote by T the set of all the *terms* built on D and V. The HCL basic construct is the *atomic formula* (or *atom*) $p(t_1,...,t_m)$, where $p \in P$, and $t_1,...,t_m \in T$. Let U be the subset of T consisting of all the terms containing no variables (ground terms). A definite clause is a construct of the form $A \leftarrow B_1,...,B_n$ $(n \geqslant 0)$, where A and the B_i's are atomic formulas. A is the *head* of the clause and $B_1,...,B_n$ is the *body*. If the body is empty the clause is a *unit clause*. An HCL program is a finite set of definite clauses $W=\{C_1,...,C_n\}$. A *goal statement* is a construct of the form $A_1,...,A_m$, where each A_i is an atomic formula. Definite clauses and goals can be seen as particular first order logic formulas: '$\leftarrow$' and ',' denote logic implication and conjunction respectively, and all variables are seen as universally quantified.

A *substitution* is a mapping $\theta : V \rightarrow T$ such that $Dom(\theta)$ is finite, where $Dom(\theta)$ is the set $\{x \in V | \theta(x) \neq x\}$. A substitution θ is called *renaming* if $\forall x(\theta(x) \in V)$ and θ is bijective. Let E be an expression (term or formula). We denote by $Var(E)$ the set of variables occurring in E. Given an expression E, $\theta_{|E}$ is the substitution whose domain is $D = Var(E) \cap Dom(\theta)$ and such that for all variables in $D, \theta_{|E}$ is equal to θ. The composition of substitutions is defined in the obvious way, and induces a *preorder* on substitutions

$$\theta_1 \leqslant \theta_2 \text{ iff } \exists \gamma(\theta_1 \gamma = \theta_2).$$

The *application* of a substitution θ to an expression E, denoted by $E\theta$, is defined as the simultaneous replacement of every variable x in E with $\theta(x)$. If θ is a renaming, then $E\theta$ is a *variant* of E. The application of substitutions induces a preorder on expressions, called more-generality preorder:

$$E_1 \leqslant E_2 \text{ iff } \exists \theta(E_1\theta = E_2).$$

Two expressions, E_1 and E_2 are *unifiable* iff $\exists \theta(E_1\theta = E_2\theta)$. If θ is a minimal substitution that makes E_1 and E_2 syntactically equal, then it is called the *mgu* (most general unifier) of E_1 and E_2 ($mgu(E_1, E_2)$).

The operational semantics of HCL programs is based on the notion of refutation. Let G be the goal , $A_1,...,A_m$ and let $C \equiv A \leftarrow B_1,...,B_n$ be a variant of a clause of W. Assume that A and A_i are unifiable, and let θ be their *mgu*. Then the goal

$$G' \equiv \leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_n, A_{i+1}, \ldots, A_m)\theta$$

is *derivable* from G, by using C with substitution θ. Briefly: $G \overset{\theta}{\underset{C}{\longmapsto}} G'$. By

repeated applications of this step we obtain a *derivation*:

$$G = G_1 \xrightarrow{\theta_1} G_2 \xrightarrow{\theta_2} \cdots G_{k-1} \xrightarrow{\theta_{k-1}} G_k.$$

Briefly: $G \overset{\theta}{\longmapsto}{}^* G_k$, where $\theta = \theta_1 \cdots \theta_{k-1}$. If G_k is empty (*null clause*, denoted by $\square$), then G is *refutable* in W and $\theta_{|G}$ is the *computed answer substitution*. The name SLD-resolution indicates this type of refutation together with a selection rule that chooses, for every goal, the atom to be resolved.

The operational meaning of a program W is defined as

$$SS = \{p(t_1, \ldots, t_n) | t_1, \ldots, t_n \in U \ \& \ \exists\theta(\leftarrow p(t_1, \ldots, t_n) \leftarrow A \overset{\theta}{\longmapsto}{}^* \square)\}.$$

The other standard semantics (model-theoretic and fixpoint), defined by van Emden and Kowalski [7], characterizes an HCL program W from a declarative point of view. Both of them are based on Herbrand interpretations (subsets of the Herbrand base B, defined as the set of the ground atoms). The model-theoretic semantics has to do with the notion of (standard) Herbrand model. A Herbrand model is a Herbrand interpretation which satisfies (á la Tarski, see, for instance, [24]) the program. The meaning of a program W is defined as the minimal Herbrand model M of W (i.e., the set of the ground atoms that are logical consequences of W).

The second semantics is given as the least fixpoint (*lfp*) of a transformation T (*one step inference operator*) on the Herbrand interpretations, defined as

$$T(I) = \{A \in B | \exists(A' \leftarrow B_1, \ldots, B_n) \in W, \exists\theta(B_1\theta, \ldots, B_n\theta \in I \ \& \ A'\theta = A)\}.$$

In [7] the equivalence of model-theoretic, fixpoint and operational semantics is proved ($M = lfp(T_W) = SS$). This result gives the soundness and completeness of SLD-resolution for HCL. However, it is worth noting that the operational semantics definition given above does not reflect entirely all the features of the language. In fact, the above set characterizes only the ground atoms which are refutable in (and which are logical consequences of) the program. A more adequate operational semantics should consider the refutability of non ground goals and the computed answer substitution (see [9]). Unfortunately, it turns out that M and $lfp(T_W)$ are not powerfull enough to model this notion. As a matter of fact, to formulate a stronger completeness result it is necessary to consider the general models of a program (i.e., all the models defined on any kind of domain) [4].

3. A MORE GENERAL APPROACH TO THE DECLARATIVE SEMANTICS. THE SCHEME

In this section we give a method to generate the declarative semantics of HCL-based languages. The method is applicable to languages whose operational behaviour can be described in terms of 'amount of information that can be produced', and then modelled by ordered structures. This construction is, of course, parametric with respect to the features of the particular language (and for this reason we call it schema). In particular, it depends on the particular ordered structure reflecting the operational ability to produce information. It depends also on the particular class of formulae to which we want to give meaning. Finally, the notion of truth and the notion of valuation can be

extended in different ways. The scheme consists of

- an extension **((D1)-(D5))** of the notions of the standard declarative semantics, in order to deal with more complex interpretation domains. **(D1)-(D5)** are the parameters of the scheme,
- some general properties **((C1)-(C7))** that express the conditions to be satisfied so that our construction is applicable,
- the properties **((P1)-(P5))** of the declarative semantics that is possible to define, if the above conditions are satisfied.

3.1. The models

We define the semantic structures suitable to represent the features of the language. See [24] for the basic terminology concerning Tarski's model-theoretic semantics (domain, interpretation, valuation, truth, etc.)

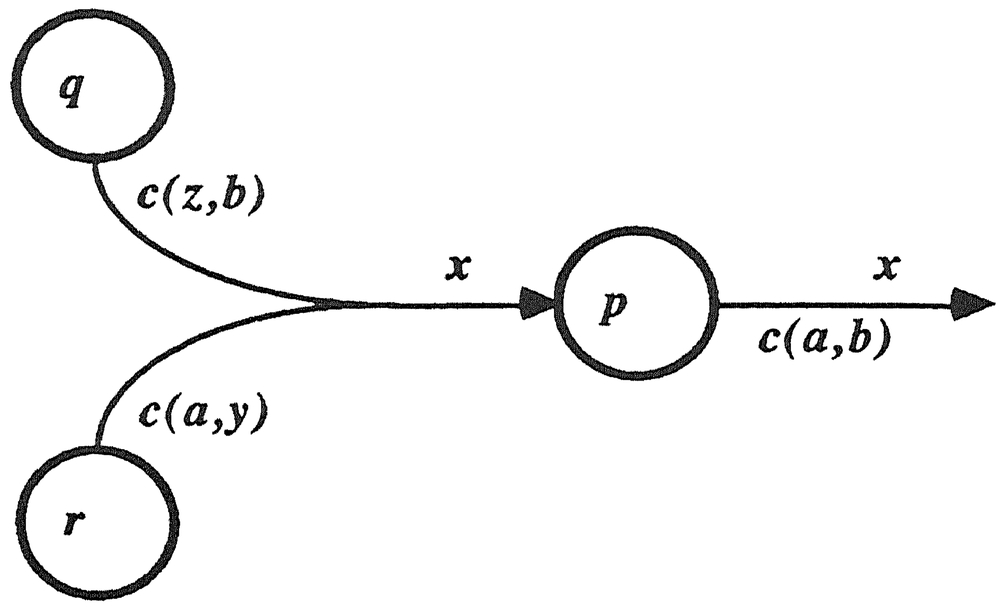
(D1) The first step is the choice of the ordering structure, that we will call *Ord*. The domains of the interpretations are required to have this structure. *Ord* must reflect the relation between the informations produced by different goals, in a given program. For instance, *Ord* can be the class of the Algebraic Complete Partial Orders, as in the case of K-LEAF (see Section 4).

(D2) Next, we must define a suitable notion of valuation. The standard one, in which each occurrence of a given variable is replaced by the same value, is usually too weak. In some concurrent logic languages, for instance, the shared variables can be seen as communication channels between processes, where more than one process is allowed to produce data. Then, these languages can better be modelled by allowing valuations to assign different values to different occurrences of the same variable. Of course, these values must be compatible (i.e. they must admit a common upper bound). This corresponds to saying that different occurrences of the same variable can give different approximations of the same value.

EXAMPLE 3.1. Let W be the concurrent logic program

$$\begin{aligned} W = \quad & \{p(x) \leftarrow q(x), r(x). \\ & r(c(a,y)) \leftarrow. \\ & q(c(z,b)) \leftarrow. \} \end{aligned}$$

where p,q and r are seen as processes able to produce values on the shared variable x. They produce on x different informations: $x:=c(a,b)$, $x:=c(a,y)$ and $x:=c(z,b)$ respectively. However, $c(a,y)$ and $c(z,b)$ can be seen as different parts of the same data structure $c(a,b)$.



According to the requirements expressed in **(D1)**, the interpretations of $c(a,y)$ and $c(z,b)$ (in *Ord*) are elements smaller than the interpretation of $c(a,b)$. Then, we could adopt the notion of compatible substitution, namely a function that assigns *compatible* values (i.e. having a common upper bound) to different occurrences of the same variable. □

(D3) The notion of truth for a formula can also be extended. Consider the following example.

EXAMPLE 3.2. Consider the first clause of the program in Example 3.1.

$$p(x) \leftarrow q(x), r(x).$$

The process p produces on x the data that can be obtained by combining the ones produced by q and r. Then, such a formula can be defined to be *true* iff

> *for all the possible valuations θ that assign to the occurrence of x in p the least upper bound of the values assigned to the ones in q and in r, if $q(x)\theta$ and $r(x)\theta$ are true, then also $p(x)\theta$ is true.*

We use the notation $I \models_{Ord} F$ to denote that the formula F is true (with respect to the notion defined in **(D3)**) in the interpretation I (satisfying **(D1)**). We write simply $I \models F$ when *Ord* is clear from the context. The following definition is the natural extension of the standard one.

DEFINITION 3.1. Let W be a program. An interpretation I is a *model* of W ($I \models_{Ord} W$) iff for every clause $C \in W$, $I \models_{Ord} C$ holds. A formula F is a *logic consequence* of W, ($W \models_{Ord} F$) iff, for every interpretation I, if $I \models_{Ord} W$ then $I \models_{Ord} F$ holds.

3.2. The special model

Now we characterize a model that represents the notion of being a logical consequence, at least with respect to a particular class Π of formulae (Π-formulae).

(D4) We consider a special domain D, a sort of new Herbrand Universe, that satisfies the order structure required in **(D1)**. Given a program W, the new definition of Herbrand interpretation (*D-interpretation*) and of Herbrand model (*D-model*) follows from **(D2)** and **(D3)**. If F is a formula, then $W \models_D F$ denotes that F is true in all the D-models of W. Given a class of formulae Π, the D-models characterize in this class the property of being a logical consequence, if the following variant of the Löwenheim-Skolem theorem holds.

(C1) $\forall F \in \Pi (W \models_{Ord} F \Leftrightarrow W \models_D F)$

It is not necessary to prove condition **(C1)** if the condition **(C5)** holds (see below).

(D5) Let DI denote the set of the D-interpretations for a given program W. DI has to be equipped with an ordering relation $\leqslant$ *consistent with the notion of truth* with respect to the Π-formulae. Namely, if I, I' are D-interpretations, and $I \leqslant I'$, then

$$\forall I, I' \in DI \ \forall F \in \Pi ((I \leqslant I' \& I \models F) \Rightarrow I' \models F).$$

The basic property that the ordering $\leqslant$ has to satisfy, is the existence of the minimal model. This generalizes the model-intersection property. Formally:

(C2) $\exists M_{\min} \in DI \ (\forall I \in DI (I \models W \Rightarrow M_{\min} \leqslant I) \ \& \ M_{\min} \models W)$.

The following result holds.

THEOREM 3.1. *Let W be a program. If properties* **(C1)** *and* **(C2)** *hold, then*

(P1) $\forall F \in \Pi (W \models_{Ord} F \Leftrightarrow M_{\min} \models F)$.

PROOF.

($\Rightarrow$) Let F be a formula in Π. If $W \models_{Ord} F$ then, by property **(C1)**, F is true in every D-model, and, in particular, in the minimal one.

($\Leftarrow$) Let F be a formula in Π, and assume $M_{\min} \models F$. By property **(C2)**, for each D-model I we have $M_{\min} \leqslant I$. Since the ordering among D-interpretations preserves the notion of truth, we have that $I \models F$. Therefore, $W \models_D F$ holds. Then, by property **(C1)**, $W \models_{Ord} F$ holds. □

This theorem shows that $M_{\min}$ characterizes the Π-formulae that are logical consequences of a program W. Then we can use $M_{\min}$ to define the declarative semantics of the Π-formulae in W. Formally, for $F \in \Pi$:

$$S_D(F) = \{F' \in \Pi \mid \exists \theta \text{valuation}(F' = F\theta \ \& \ F' \in M_{\min})\}.$$

3.3. The fixpoint semantics and the relation with the declarative semantics

We characterize now a mapping T_D on the set of the D-interpretations of a given program W. T_D extends (on the Π-formulae) the standard *one step inference operator*. The least fixpoint of this mapping gives the so-called fixpoint semantics.

DEFINITION 3.2. Let W be a program, and let I be an S-interpretation. Define

$$T_D(I)=\min\{I'\in DI \mid \forall F, F'\in\Pi((I\models F \,\&\, \exists C\in W(C\models(F\rightarrow F'))\Rightarrow I'\models F')\}.$$

The correctness of the definition of T_D, namely the existence of such a minimal interpretation I', derives from a property similar to **(C2)**. We give two sufficient conditions for the existence of the least fixpoint of T_D.

(C3) $(DI, \leqslant)$ is a Complete Partial Order (CPO).

(C4) T_D is continuous.

The following result is a general property of continuous functions defined on CPO's.

THEOREM 3.2. *If the properties* **(C3)** *and* **(C4)** *hold, then*

(P2) *There exists the least fixpoint of* T_D, $lfp(T_D)$.

(P3) $lfp(T_D)=lub_n T_D{}^n(I_\perp)$,

where lub means least upper bound and $I_\perp$ *is the minimal element in the CPO* $(DI, \leqslant)$.

Thanks to the result **(P2)** of the previous theorem, we can use T_D to define the (fixpoint) semantics of the Π-formulae in W. Formally, for $F\in\Pi$:

$$S_F(F)=\{F'\in\Pi \mid \exists\theta \text{ valuation } (F'=F\theta \,\&\, F'\in lfp(T_D))\}.$$

Note that if $(DI, \leqslant)$ is a complete lattice, then to ensure **(P2)** it would be sufficient to have the monotonicity of T_D (Knaster-Tarski theorem). On the other side, the continuity of T_D allows to obtain the property **(P3)**, that will be usefull in order to relate the fixpoint semantics to the operational one.

The equivalence between the fixpoint semantics and the declarative one is ensured by the following condition, whose structure is more general than the standard equivalence between Herbrand models and interpretations closed with respect to the one step inference operator. In fact, we only require the fixpoints of T_D to be D-models, and the D-models to be closed with respect to T_D (namely, to satisfy the condition $T_D(I)\leqslant I$). Formally:

(C5) $\forall I\in DI\ ((I=T_D(I)\Rightarrow I\models W) \,\&\, (I\models W\Rightarrow T_D(I)\leqslant I))$.

THEOREM 3.3. *If property* **(C5)** *holds, then*

(P4) $lfp(T_D)=M_{\min}$.

PROOF. Let I be a D-interpretation such that $T_D(I)\leqslant I$. Then, by monotonicity of T_D, we have, for each $n, T_D{}^n(I_\perp)\leqslant T_D{}^n(I)\leqslant I$. Therefore $lfp(T_D)=lub_n T_D{}^n(I_\perp)\leqslant I$. On the other side, $T_D(lfp(T_D))=lfp(T_D)$. Therefore $lfp(T_D)$ is the minimal interpretation closed with respect to T_D. By **(C5)** we have that $lfp(T_D)$ is a D-model of W and that it is the minimal one.

Note that this theorem also implies the existence of the minimal D-model of W. Therefore:

If property **(C5)** *is satisfied then it is not necessary to assume condition* **(C2)**.

3.4. The operational semantics and the relation with the declarative semantics

Now we define two conditions that are sufficient to ensure the soundness and completeness results. Define the success set SS_Π as

$$SS_\Pi = \{G \in \Pi \mid ((\leftarrow G \overset{\theta}{\longmapsto}{}^* \square) \;\&\; (\theta_{|G} = \epsilon))\}$$

where ϵ is the empty substitution.

(C6) $\forall G, G' \in \Pi((\leftarrow G \overset{\theta}{\longmapsto}{}^* \leftarrow G') \Rightarrow (C \vDash (G' \rightarrow G\theta)))$,

(C7) $\forall G \in \Pi(G \in TD^n(\varnothing)) \Rightarrow G \in SS)$.

The following result is an immediate consequence of **(P4)**.

THEOREM 3.4. *If properties* **(C6)** *and* **(C7)** *hold, then*

(P5) $SS_\Pi = \{G \in \Pi \mid M_{\min} \vDash G\}$.

Depending on the class Π we can get stronger results. For instance, if Π is the set of the ground atoms B, then we get the standard ones:

(P6) Each computed answer substitution is correct, and each correct answer substitution admits a more general substitution that can be computed. Formally:

1) $\forall A \in B((\leftarrow A \overset{\theta}{\longmapsto}{}^* \square) \Rightarrow (C \vDash_{Ord} A\theta))$, and

2) $\forall A \in B((C \vDash_{Ord} A\theta) \Rightarrow \exists \theta', \gamma\,(\theta\gamma = \theta' \;\&\; \leftarrow A \overset{\theta'}{\longmapsto}{}^* \square))$

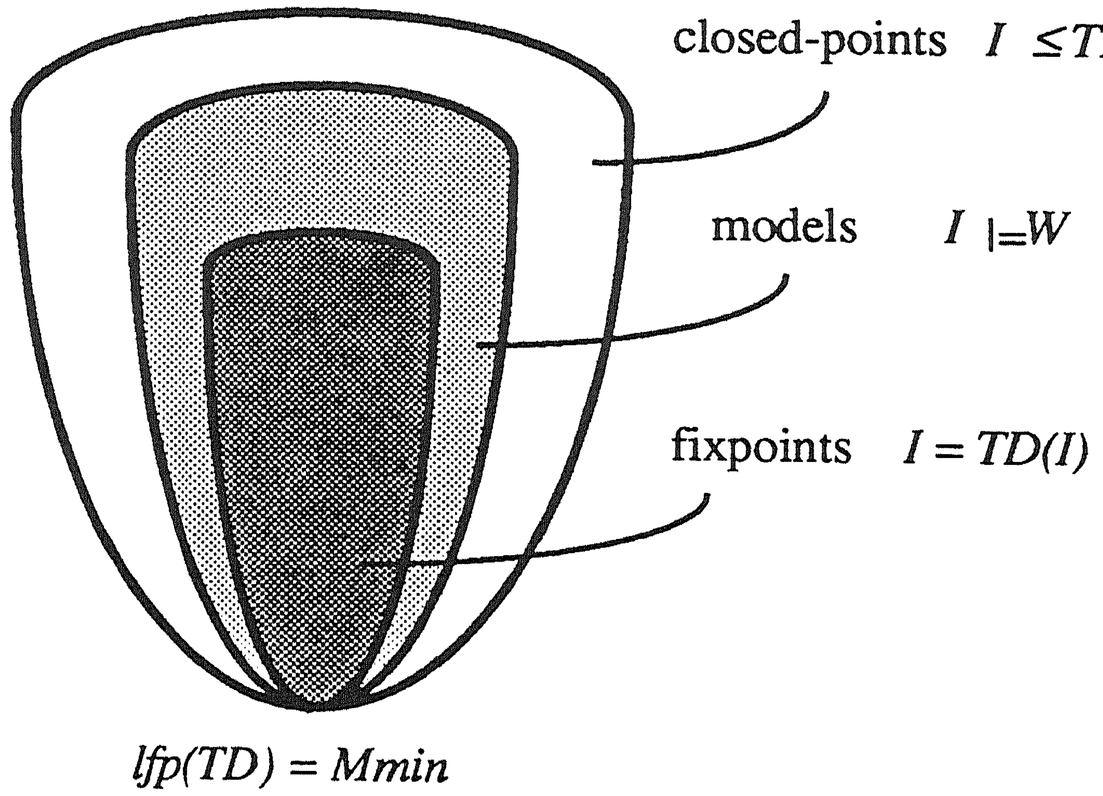
(P6)-(1) expresses the soundness, and **(P6)-(2)** is the so-called *strong completeness result*, due to [4].

If Π is the set of all the atoms At, then we get a result stronger than **(P6)**:

(P6′) The computed answer substitutions for a goal $\leftarrow A$ are the most general unifiers (*mgu*) of A and some of the atoms true in $M_{\min}$. Formally:

$$\forall A \in At,\ \forall \theta\ (\exists A' \in At\ (M_{\min} \vDash_{Ord} A' \;\&\; mgu(A, A')_{|A} = \theta)$$
$$\Leftrightarrow \exists \theta'\ ((\leftarrow A \overset{\theta}{\longmapsto}{}^* \square) \;\&\; (mgu(A, A')_{|A} = \theta)))$$

It is possible to show that **(P6′)** implies **(P6)**. Note that **(P6′)** allows to avoid the use of all the models (necessary to get the strong completeness result of Clark). Namely, the minimal model $M_{\min}$ is powerfull enough to characterize the declarative semantics of all the formulas with respect to which we want to get the completeness result.



4. Applications to existing languages

In this section we show some applications of the method described above. We discuss three cases, namely K-LEAF, pure HCL, and concurrent logic languages, whose semantics can be obtained as instances of the scheme we have defined.

4.1. The language K-LEAF

K-LEAF is a logic + functional (first order) language based on HCL + equality. Namely, the syntax is essentially HCL, enriched with the special symbol =. Its computational mechanism consists of two steps:

- transforming the program and the goal into a new program and goal, in which no more than one functional symbols occur in the terms. This procedure is called *flattening*
- applying SLD-resolution to the new program and goal.

For a detailed presentation of this language see [3,19,20]. The distinguishing feature of K-LEAF, with respect to other kinds of logic+functional languages, is that function calls are seen as possibly infinite processes. So, for example, if the function f is defined by the equation (unit clause) $f(x)=c(f(x))\leftarrow$, then the intended meaning of $f(a)$ is the infinite structure $c(c(c(...)))$. To model this kind of semantics, therefore, it is necessary to use structures in which the limit of an infinite computation can be adequately represented. We have chosen the algebraic Complete Partial Orders (see, for instance [2]). We summarize the basic choices, according to the scheme:

(D1) *Ord*: the class of the algebraic Complete Partial Orders.
(D2) *The notion of valuation*: the standard one.
(D3) *The notion of truth*: the standard one.
(D4) *The special domain D*: the Herbrand universe enriched with a new symbol

$\perp$, and with infinite terms. The ordering on D is obtained by imposing the following conditions:

- $\forall d \in D(\perp \leqslant d)$, and
- the constructors are monotonic.

(D5) *ordering on DI*: $I \leqslant I'$ iff the interpretation of each function symbol is smaller in I than in I' (with respect to the natural function ordering)

We indicate by D_{AM} the subset of the algebraic maximal elements of D. Conditions **(C1)-(C7)** can be proved to hold with respect to the following set of formulae:

$$\Pi = \{p(d_1, \ldots, d_n) | d_1, \ldots, d_n \in D_{AM}\}.$$

Then, the properties **(P1)-(P5)** hold for Π. A result similar to **(P6)** can also be proved.

4.2. Pure Horn Clause Logic

The operational semantics of HCL can be interpreted not only as a procedure to prove relations, but also as a method of generating the substitutions under which a given relation is derivable. The standard declarative semantics is not adequate to fully characterize this behaviour, namely the strong completeness result [4] (see Section 3) is still too weak. A more powerfull semantics is presented in [8] and [9]. The basic idea is to enrich the Herbrand domain with terms containing variables, in order to interpret 'syntactically' the universal quantifier. The construction is still an instance of our scheme. We consider the class Π of all the atoms. Then, according to Section 3.4, the step **(D1)** is not necessary. The other choices are:

(D2) *The notion of valuation (only on D, see (D4))*: the substitution.

(D3) *The notion of truth (only on DI)*: a clause is said to be true in I if whenever the body matches with elements in I, giving a most general unifier θ, then the head, instantiated by θ, belongs to I.

(D4) *The special domain D*: the set of all the terms, ordered by the more-generality relation, i.e. $d \leqslant d'$ iff there exists a substitution θ such that $d\theta = d'$.

(D5) *ordering on DI*: the set inclusion.

The properties **(C2)-(C7)** can be proved to hold with respect to Π. Therefore **(P2)-(P5)** hold. Moreover, the stronger result **(P6')** holds.

4.3. Concurrent logic languages

In this section we consider the concurrent logic languages whose synchronization mechanism is based on input-mode constraints, namely on some restrictions on unification. The most famous languages that belong to this class are Concurrent Prolog [22,23], PARLOG [5,6,10] and Guarded Horn Clauses [25,26]. These languages have a common structure, the basic construct being the *guarded clause*:

$$A \leftarrow G_1, \ldots, G_m \mid B_1, \ldots, B_n.$$

(where $G_1, \ldots, G_m$ is called *guard part*, and $B_1, \ldots, B_n$ is the *body part*). Some constraints can be specified on the clause. Their role is essentially to prevent the use of the clause, during the refutation procedure, if the arguments of the atom selected in the goal are not instantiated enough. According to the *process interpretation* [22,17] a goal can be seen as a network of processes communicating via the shared variables (channels), producing and consuming data on them. A constraint can then be seen as the requirement, for a consumer, to receive some data on certain variables.

In order to define the declarative semantics of these languages the notion of producer/consumer process has to be modeled. This can be done by ordering the atoms (processes) with respect to their ability to produce data. The semantics proposed in [18] meets these requirements, and can be seen as an instance of our schema (a variant of the instance described in Section 4.2).

(D2) *The notion of valuation* (*only on D*): the compatible substitution (see example 3.1).

(D3) *The notion of truth* (*only on DI*): similar to the one defined in example 3.2.

(D4) *The special domain D*: the set of all the terms, built on the annotated constructors. Constructors can be annotated '+', (produced) or '−' (consumed). *D* is ordered by the more-generality relation, and by the ordering induced by defining $c^- \leqslant c^+$ for each constructor c.

(D5) *Ordering on DI*: the set inclusion.

The properties **(C2)**-**(C7)** can be proved to hold with respect to the set Π of all the annotated atoms (with a notion of operational semantics slightly different from the original one). Therefore, for this class of formulae, **(P2)**-**(P5)** hold.

5. Conclusion and future works

This paper is a first step in the attempt to defining a scheme for the declarative semantics of languages based on HCL. The approach is analogous to (but more powerful than) the one defined in [11,12,13]. What is still missing is a theory allowing to derive the validity of the conditions **(C1)**-**(C7)** from few simple properties of the notions that are the parameters of the scheme. This is not so easy, because it is necessary to find a good compromise between the generality and flexibility of these notions, and the simplicity of the properties to be verified.

Possible relations with the theory developed in [14,15] (that generalizes the previous mentioned one) are also under investigation.

References

1. K.R. Apt (1988). *Introduction to Logic Programming,* Report CS-R8826, CWI, Amsterdam. To appear in *Handbook of Theoretical Computer Science,* North Holland.
2. H.P. Barendregt (1984). *The Lambda-Calculus: Its Syntax and Semantics,* Revised edition, North-Holland.

3. M. Bellia, P.G. Bosco, E. Giovannetti, G. Levi, C. Moiso, C. Palamidessi (1987). A two-level approach to logic plus functional programming integration. *Proc. PARLE Conference*, Springer-Verlag.
4. K. L. Clark (1979). *Predicate Logic as a Computational Formalism*, Research Report DOC 79/59, Department of Computing, Imperial College.
5. K.L. Clark, S. Gregory (1985). Notes on the implementation of PARLOG. *Journal of Logic Programming 2 (1)*, 17-42.
6. K. L. Clark, S. Gregory (1986). PARLOG: parallel programming in logic. *ACM Trans. on Progr. Lang. and Syst. 8*, 1-49.
7. M. H. van Emden, R. A. Kowalski (1976). The semantics of predicate logic as a programming language. *J. ACM 23*, 733-742.
8. M. Falaschi, G. Levi, M. Martelli, C. Palamidessi (1988). A new declarative semantics for logic languages. *Proc. 1988 International Conference and Symposium on Logic Programming*, Seattle, U.S.A.
9. M. Falaschi, G. Levi, M. Martelli, C. Palamidessi. *Declarative Modeling of the Operational Behaviour of Logic Languages.* To appear in *Theoretical Computer Science.*
10. S. Gregory (1987). *Parallel Logic Programming in PARLOG*, International Series in Logic Programming, Addison-Welsey Pub. Comp.
11. J. Jaffar, J.-L. Lassez, M.J. Maher (1984). A theory of complete logic programs with equality. *J. Logic Programming 1*, 211-223.
12. J. Jaffar, J.-L. Lassez, M.J. Maher (1986). A logic programming language scheme. D. de Groot, G. Lindstrom (eds.). *Logic Programming: Functions, Relations and Equations*, Prentice-Hall, 441-468.
13. J. Jaffar, J.-L. Lassez, M.J. Maher (1986). Some issues and trends in the semantics of logic programming. *Proc. of Third Int. Ul Conf. on Logic Programming*, LNCS 225, Springer-Verlag, 223-241.
14. J. Jaffar, J.-L. Lassez (1986). *Constraint Logic Programming*, Internal Report, Department of Computer Science, Monash University.
15. J. Jaffar, J.-L. Lassez (1987). Constraint Logic Programming. *Proc. of the SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ACM, 111-119.
16. J. Jaffar, J.-L. Lassez, M.J. Maher (1987). Prolog II as an instance of the logic programming language scheme. M. Wirsing (ed.). *Proc. IFIP Conf.*
17. G. Levi, C. Palamidessi (1985). The declarative semantics of logical read-only variables. *Proc. 1985 IEEE Symposium on Logic Programming*, IEEE Comp. Society Press, 128-137.
18. G. Levi, C. Palamidessi (1987). An approach to the declarative semantics of synchronization in logic languages. J.-L. Lassez (ed.). *Proc. 1987 Conference on Logic Programming*, MIT Press, 877-893.
19. G. Levi, C. Palamidessi, P.G. Bosco, E. Giovannetti, C. Moiso (1987). A complete semantics characterization of K-LEAF, a logic language with partial functions. *Proc. 1987 Symp. on Logic Programming*, IEEE Comp. Society Press.

20. G. LEVI, C. PALAMIDESSI, P.G. BOSCO E. GIOVANNETTI, C. MOISO. *Kernel LEAF: A Logic plus Functional Language*. Submitted for publication to the *Journal of Computer and System Science*.
21. J.W. LLOYD (1987). *Foundations of Logic Programming*, Second Edition, Springer-Verlag.
22. E.Y. SHAPIRO (1983). *A Subset of Concurrent Prolog and its Interpreter*, ICOT Technical Report TR-003.
23. E.Y. SHAPIRO (1986). Concurrent Prolog: a progress report. W. BIBEL, PH. JORRAND (eds.). *Fundamentals of Artificial Intelligence*, LNCS 232, Springer-Verlag, 277-313.
24. J. SHOENFIELD (1976). *Mathematical Logic*, Addison-Wesley, Reading, Mass.
25. K. UEDA (1985). *Guarded Horn Clauses*, ICOT Tech. Rep. TR-103.
26. K. UEDA (1986). *Guarded Horn Clauses*, Doctoral Thesis, Information Engineeering Course, Faculty of Engineering, University of Tokyo.