The finiteness of logic programming derivations

Philippe Balbiani

The finiteness of logic programming derivations

Philippe Balbiani

The finiteness of logic programming derivations

Philippe Balbiani[1]

Institut de Recherche en Informatique de Toulouse

Abstract. The question of the termination of logic programming computations is studied from a semantical point of view. To every program are associated two first order formulas. Their valid consequences are respectively the finiteness and the infiniteness *SLDNF* sets of the logic programs considered. The non-existence of a recursive safe computation rule leading into an infinite *SLDNF* computation is proved.

1 Introduction

When does it stop? This question is common to many fields in computer science. To ask it in a logic programming setting does not make it easier: the restriction of first order predicate calculus to definite Horn clauses has the full power of recursion theory. Thus, because of the indecidability of the halting problem, there is no procedure to decide whether a computation will end or not. Non-termination is one of the most inefficient behaviour of programs. To force termination, programmers sometimes decide to insert control informations in their programs. This leads to imperative programs: from the point of view of logic programming, this is not an ideal option. Another way to limit the number of non-terminating execution is to allow the interpreter to cut some derivations [3] or to forbide multiple use of some clauses. Other methods consist in the analysis of the stream of informations between clauses and in the resolution of a linear system of equations [16]. For some classes of programs, these methods provide decision procedures for termination [1, 3, 7, 16, 18].

We do not propose here another method to decide termination in logic programming. We rather propose a semantical characterization of finite and infinite *SLDNF* computations.

To every program is firstly associated a first order classical formula: the completion of finiteness. As Clark's formula [10] offered a characterization of *SLDNF* finite failure, the ground valid consequences of this completion formula of finiteness will be the finiteness set of the logic program considered, that is to say: the set of ground atoms from which there cannot be any infinite *SLDNF* computation.

Secondly, a completion formula of infiniteness is defined. This formula will belong to

the language of first order modal logic of provability. Its ground valid consequences are the set of ground atoms from which it is possible to have an infinite *SLDNF* computation. Since the modal logic of provability [9] is not axiomatizable, this characterization does not lead to a decision procedure for infinite *SLDNF* computation. Consequently, we ask the question of the existence of an algorithm leading, every time it is possible, to an infinite computation. It is proved that such an algorithm cannot exist: for some logic programs, a safe computation rule leading to an infinite derivation every time it is possible cannot be recursive.

2 Normal Logic Programs

The first order language in which normal logic programms will be written, $\mathcal{L}$, is made of variables, constants, function symbols, predicate symbols[2], classical connectors $\neg$, $\wedge$, $\vee$ and $\leftarrow$, the quantifiers $\exists$ and $\forall$ and the usual punctuation symbols. Pre-interpretation, interpretation, variable assignment (with respect to some pre-interpretation), term assignment (with respect to some pre-interpretation and variable assignment) are defined as usual (see Lloyd [14] for precise definitions). Let J be a pre-interpretation of $\mathcal{L}$, V a variable assignment with respect to J and A an atom. We suppose that A is of the form $p(t_1,...,t_n)$ and that d_1, ..., d_n, elements of the domain of J, are the term assignments of t_1, ..., t_n. We call $A_{J,V} = p(d_1,...,d_n)$ the J-instance of A with respect to V. Let $[A]_J = \{A_{J,V}: V$ is a variable assignment with respect to $J\}$. Let I be an interpretation of domain D of $\mathcal{L}$ and V a variable assignment. To every formula of $\mathcal{L}$ is given a truth value, *true* or *false*, (with respect to I and V) following the classical rules of first order predicate calculus. A ground term (atom) is a term (atom) without variable. The Herbrand universe $U_{\mathcal{L}}$ is the set of ground terms of $\mathcal{L}$. The Herbrand base $B_{\mathcal{L}}$ is the set of ground atoms of $\mathcal{L}$. The Herbrand pre-interpretation is the pre-interpretation such that: (a) its domain is $U_{\mathcal{L}}$, (b) constants are assigned to themselves, (c) the mapping from $(U_{\mathcal{L}})^n$ to $U_{\mathcal{L}}$ defined with $(t_1,...t_n) \rightarrow f(t_1,...t_n)$ is assigned to any function symbol f of arity n. An Herbrand interpretation is an interpretation based on the Herbrand pre-interpretation.

Normal clause, normal program, normal goal, derived goal, safe computation rule, *SLDNF* derivation, *SLDNF* refutation and *SLDNF* tree are defined as in [14]. A safe computation rule always selects, if possible, a positive or a ground negative literal in a

given goal. An *SLDNF* computation flounders if this selection is not possible. Let P be a normal program and G be a normal goal. The *finiteness set* of P is the set of all $A \in B_{\mathcal{L}}$ such that there is no infinite *SLDNF* computation of $P \cup \{\leftarrow A\}$. The substitution θ is a *computed answer of finiteness* of rank n of $P \cup \{G\}$ if there is no *SLDNF* computation of $P \cup \{G\theta\}$ involving n or more than n goals. The substitution θ is a computed answer of finiteness of $P \cup \{G\}$ if there is no infinite *SLDNF* computation of $P \cup \{G\theta\}$. The *infiniteness set* of P is the set of all $A \in B_{\mathcal{L}}$ such that there is an infinite *SLDNF* computation of $P \cup \{\leftarrow A\}$. The substitution θ is a *computed answer of infiniteness* of rank n of $P \cup \{G\}$ if there is an *SLDNF* computation of $P \cup \{G\theta\}$ involving n or more than n goals. The substitution θ is a computed answer of infiniteness of $P \cup \{G\}$ if there is an infinite *SLDNF* computation of $P \cup \{G\theta\}$.

3 A Completion Formula

We define in this section a completion formula of finiteness for all normal programs. As Clark's formula [10], it will be obtained putting on the same side of an implication symbol the bodies of the clauses defining some predicate. Let $p(t_1,...,t_n) \leftarrow L_1,...,L_m$ be a clause of a program P. We will need a new predicate symbol: the equality predicate, $=$. The first step is to transform this clause into the formula $p(x_1,...,x_n) \leftarrow (L_1 \wedge ... \wedge L_m \leftarrow (x_1 = t_1) \wedge ... \wedge (x_n = t_n))$ where the variables $x_1, ..., x_n$ do not appear in P. If $y_1, ..., y_d$ are the variables of the original clause then this formula is transformed into $p(x1,...,xn) \leftarrow \forall y1 ... \forall yd\ (L_1^* \wedge ... \wedge L_m^* \leftarrow (x_1 = t_1) \wedge ... \wedge (x_n = t_n))$, where: if L_i is an atom then $L_i = L_i^*$ else L_i is a negative literal and $L_i = \neg L_i^*$. Let us suppose this transformation has been done for every clause appearing in the definition of p. Then we have k formulas of the form $p(x_1,...,x_n) \leftarrow E_1$, ..., $p(x_1,...,x_n) \leftarrow E_k$, where each E_i is of the form $\forall y_1 .. \forall y_d\ (L_1^* \wedge ... \wedge L_m^* \leftarrow (x_1 = t_1) \wedge ... \wedge (x_n = t_n))$. The *completed definition of finiteness* of p is then the formula $\forall x_1 ... \forall x_n\ (p(x_1,...,x_n) \leftarrow E_1 \wedge ... \wedge E_k)$.

It might be the case that some predicate symbols are not the head of any clauses in P. For such a predicate symbol q, we explicitly add the formula $\forall x_1 ... \forall x_n\ q(x_1,...,x_n)$. This formula is also called the completed definition of finiteness of q. Let P be a normal program. The *completion of finiteness* of P, $comp_F(P)$, is the collection of all the completed definition of finiteness of the predicate symbols of $\mathcal{L}$ together with *CET*,

Clark's equational theory that defines the predicate symbol of equality [10]. Our completion of finiteness possesses some properties possessed by P itself. Next result, for example, states that the set of Herbrand models of $comp_F(P)$ is a complete lattice.

proposition 3.1 Let P be a normal logic program and $\{M_i\}_{i\in I}$ a non-empty set of Herbrand models of $comp_F(P)$. *Then* $\cap_{i\in I}M_i$ *is a Herbrand model of* $comp_F(P)$.

Thus, since $B_{\mathcal{L}}$ is a Herbrand model of $comp_F(P)$, the intersection M_P of all Herbrand models of $comp_F(P)$ is still a model of $comp_F(P)$. It is the least Herbrand model of $comp_F(P)$. Unfortunately, it is not true that: $M_P = \{A\in B_{\mathcal{L}} : A$ is a valid consequence of $comp_F(P)\}$. As a matter of fact, if $P = \{A\leftarrow B(x), B(f(x))\leftarrow B(x)\}$ then the least Herbrand model of $comp_F(P)$ is $M_P = \{A\}\cup\{B(a), B(f(a)), ...\}$ but A is not a valid consequence of $CET\cup\{A\leftarrow \forall x\, B(x), \forall y\,(B(y)\leftarrow \forall x\,(B(x)\leftarrow y=f(x)))\}$.

To the procedural notion of a computed answer of finiteness is associated the semantical notion of a correct answer. Let P be a normal program, G the normal goal $\leftarrow L_1,...,L_n$ and θ a substitution of the variables of G. We say that θ is a *correct answer* for $comp_F(P)\cup\{G\}$ if $\forall((L_1{}^*\wedge...\wedge L_n{}^*)\theta)$ is a valid consequence of $comp_F(P)$. Theorem 3.5 will give a first relation between computed answers of finiteness and correct answers.

Now we define a mapping $T_P{}^J$ on the lattice of interpretations (based on some pre-interpretation J of the language) to itself. If J is a pre-interpretation of $\mathcal{L}$ and I is an interpretation based on J then $T_P{}^J(I)=\{B$: for every variable assignment V with respect to J and for every clause $A\leftarrow L_1,...,L_n$ in P, if $A_{J,V}=B$ then $L_1{}^*\wedge...\wedge L_n{}^*$ is true with respect to I and $V\}$. When J is the Herbrand pre-interpretation of $\mathcal{L}$, we will write T_P instead of $T_P{}^J$. The mapping $T_P{}^J$ possesses the usual properties.

proposition 3.2 For every pre-interpretation J of $\mathcal{L}$ and for every normal program P, *the mapping* $T_P{}^J$ *is monotonic.*

The mapping $T_P{}^J$ is not always continuous. Nevertheless its pre-fixpoints are models of the completion of finiteness of the program considered.

proposition 3.3 Let P be a normal program, J a pre-interpretation of $\mathcal{L}$ and I an interpretation based on J. Let us suppose that I, together with the identity relation assigned to $=$, is a model of the equality theory. We have: *I, together with the identity relation assigned to $=$, is a model of $comp_F(P)$ iff $T_P^J(I) \subseteq I$.*

As a corollary, we have:

proposition 3.4 *The least Herbrand model of $comp_F(P)$ is the least fixpoint of T_P.*

Note that, for every program P, $gfp(T_P)=B_{\mathcal{L}}$. A first important result is the soundness of our completion formula for *SLDNF* resolution (see theorem 3.5 below). Let P be a normal program and G be a normal goal. We say that $P \cup \{G\}$ is allowed whenever no *SLDNF* computation of $P \cup \{G\}$ flounders. We say that P is allowed whenever, for every ground atom A, no *SLDNF* computation of $P \cup \{\leftarrow A\}$ flounders.

theorem 3.5 Let P be a normal program and G be a normal goal. *If $P \cup \{G\}$ is allowed then every computed answer of finiteness of $P \cup \{G\}$ is a correct answer of $comp_F(P) \cup \{G\}$.*

proof Let θ be a computed answer of finiteness of $P \cup \{G\}$. Since there is no infinite *SLDNF* computation of $P \cup \{G\theta\}$, there is an integer *max* greater than the number of goals involved in any *SLDNF* computation of $P \cup \{G\theta\}^{(3)}$. The induction on *max* is straightforward.

As a corollary, we have:

corollary 3.6 Let P be a normal program and G a normal goal. *If $P \cup \{G\}$ is allowed and if there is no infinite SLDNF computation of $P \cup \{G\}$ then the empty substitution is a correct answer for $comp_F(P) \cup \{G\}$.*

Let P be a normal program. A consequence of corollary 3.6 is the inclusion of the set of finiteness of P in the least Herbrand model of $comp_F(P)$. However, it is not always equal to this least Herbrand model. Let us consider the program $P = \{A \leftarrow B(x), B(f(x)) \leftarrow B(x)\}$. The least Herbrand model of $comp_F(P)$ is $M_P = \{A\} \cup \{B(a), B(f(a)), ...\}$ but there is an infinite *SLDNF* computation of $P \cup \{\leftarrow A\}$. For that program, $T_P \uparrow \omega =$

$\{B(a), B(f(a)), ...\}$ and $M_P = T_P\uparrow\omega+1$. In other respects, if $A\in B_{\mathcal{L}}$, if $P\cup\{\leftarrow A\}$ is allowed and if there is no *SLDNF* computation of $P\cup\{\leftarrow A\}$ involving n or more than n goals then $A\in T_P\uparrow\omega$. If A is an atom, we define $[A]=\{A'\in B_{\mathcal{L}} : A'=A\theta$, for some substitution $\theta\}$. Then, $[A]$ is the set of ground instances of A.

theorem 3.7 Let P be a normal program and G the normal goal $\leftarrow L_1,...,L_m$. *If $P\cup\{G\}$ is allowed and if θ is a computed answer of finiteness of rank n of $P\cup\{G\}$ then* $\cup_{j=1..m}[L_j^*\theta]\subseteq T_P\uparrow\omega$.

proof The proof is a straightforward induction on the rank of the computed answer.

As a corollary, we have:

corollary 3.8 *The set of finiteness of P is included in $T_P\uparrow\omega$.*

The set of computed answer of finiteness of $P\cup\{G\}$ is not always finite (just consider the program containing one clause: $A(f(x))\leftarrow A(x)$). The possibility of its finiteness will not be studied here. Now we give the first completeness result of the finiteness of *SLDNF* resolution. Its proof is similar to the completeness proof of the negation as failure rule given by Lassez, Maher and Wolfram [13].

theorem 3.9 Let P be a normal program and G a normal goal. *Every correct answer for $comp_F(P)\cup\{G\}$ is a computed answer of finiteness of $P\cup\{G\}$.*

proof Suppose there is an infinite *SLDNF* computation of $P\cup\{G\}$. We show the empty substitution is not a correct answer for $comp_F(P)\cup\{G\}$. Let $G_0=G=\leftarrow l_1,...,l_m$, G_1, ... be the infinite *SLDNF* computation of $P\cup\{G\}$. Let θ_1, θ_2, ... be the *mgu* and C_1, C_2, ... the input clauses of this derivation. Let $^\circ$ be the relation defined on terms by $s^\circ t$ if and only if there is an integer n such that $s\theta_1...\theta_n=t\theta_1...\theta_n$. Of course, $^\circ$ is an equivalence relation on the set of terms of the language. For every term t, we note $[t]$ its class modulo $^\circ$. Let D be the set of equivalence classes modulo $^\circ$. Let J be the pre-interpretation of $\mathcal{L}$ with domain D assigning to each constant c its class $[c]$ and assigning to each function symbol f of arity n the function from D^n to D defined by: $([s_1],...,[s_n]) \rightarrow [f(s_1,...,s_n)]$. Let I be the interpretation based on J defined by: I =

$\{p([t_1],...,[t_n])$: for every element t'_1, ..., t'_n in $[t_1]$, ..., $[t_n]$, the set of proper successors of $p(t'_1,...,t'_n)$ in the computation is finite$\}$. We show that $T_P{}^J(I) \subseteq I$. If $p([t_1],...,[t_n]) \notin I$ then there are elements t'_1, ..., t'_n in $[t_1]$, ..., $[t_n]$ such that the set of the proper successors of $p(t'_1,...,t'_n)$ in the computation is infinite. Consequently, there is an integer i_0 such that the goal G_{i0} contains $p(t'_1,...,t'_n)$ as a subgoal, there is an integer i greater or equal to i_0, there is a clause $C_{i+1} = p(s_1,...,s_n) \leftarrow L_1,...,L_m$ in P, a substitution θ_{i+1} and an integer j in 1, ..., m such that $\theta_{i+1} = mgu(p(t'_1\theta_{i0+1}...\theta_i,...,t'_n\theta_{i0+1}...\theta_i), p(s_1,...,s_n))$ and the set of proper successors of $L_j\theta_{i+1}$ in the computation is infinite. Consequently, $[t_1]=[s_1]$, ..., $[t_n]=[s_n]$ and there is a clause $A \leftarrow L_1,...,L_m$ in P and a variable assignment V with respect to J such that $A_{J,V}=p([t_1],...,[t_n])$ and $L^*_1 \wedge ... \wedge L^*_m$ is false with respect to I and V. Consequently, $p([t_1],...,[t_n]) \notin T_P{}^J(I)$. Then: $T_P{}^J(I) \subseteq I$ and, according to proposition 3.3, I is a model of $comp_F(P)$. In others respects, it is not difficult to show that I is not a model of $\forall(l^*_1 \wedge ... \wedge l^*_m)$. Thus the empty substitution is not a correct answer for $comp_F(P) \cup \{G\}$.

The following theorem states the point of view of the fixpoint operator about the completeness of the finiteness of *SLDNF* derivation.

theorem 3.10 Let P be a normal program and G the normal goal $\leftarrow L_1,...,L_m$. *If* $\cup_{j=1..m}[L_j{}^*\theta] \subseteq T_P \uparrow n$ *then* θ *is a computed answer of finiteness of rank n of* $P \cup \{G\}$.

proof By induction on the integer n.

As a corollary we have:

corollary 3.11 *If P is allowed then the set of finiteness of P is equal to* $T_P \uparrow \omega$.

4 Infinite Derivations

Connectors used so far were *truth-functional*: the truth value of every formula formed by them only depends on the truth values of its subformulas. Now, our language will include a pair of intensional connectors (the *modal* connectors

□ and ◇) which will be no more truth-functional. Our language contains the following rule: if F is a formula then so is $\Box F$. The connector ◇ is defined by: $\Diamond F =_{def} \neg\Box\neg F$. We could give to □ and ◇ a variety of interpretations. Historically, modal logic is the logic of possibility and necessity: $\Box F$ and $\Diamond F$ are usualy read "F is necessary" and "F is possible". For us, □ will be used to denote the temporal relationship between bodies and heads of definite clauses. For example, we will formally represent through the modal formula $\Box(A \leftarrow \Box B)$ the procedural role of the clause $A \leftarrow B$.

A *pre-interpretation* of our first order modal language is made of: a non-empty set D, the domain; to each constant in $\mathcal{L}$ the assignment of an element in D; to each function symbol of arity n, the assignment of a function from D^n to D; a non-empty set M, the *universe* or set of *possible worlds* of the pre-interpretation; a binary relation R on M, the *accessibility relation* between possible worlds. We will require that this accessibility relation is transitive and reverse well-founded.

note This condition of well-foundedness is of fundamental importance for us. As a matter of fact, we will have to prove the equivalence between "$\Box A$ is a valid consequence of $comp_I(P)$" and "there is an *SLDNF* computation of $P \cup \{\leftarrow A\}$ involving an infinite number of goals", $comp_I(P)$ being some modal completion of infiniteness of P. On one hand, we will prove by induction on the longest *SLDNF* derivation of $P \cup \{\leftarrow A\}$ that $\Box A$ is not a valid consequence of $comp_I(P)$. On the other hand, we will prove by induction on the model of $comp_I(P) \cup \{\neg\Box A\}$ that there is no infinite *SLDNF* computation of $P \cup \{\leftarrow A\}$. This last induction holds because the accessibility relation between the possible worlds of a model of $comp_I(P) \cup \{\neg\Box A\}$ is well-founded.

An *interpretation* I of a first order modal language $\mathcal{L}$ over a pre-interpretation J with domain D and universe M is made of: for every predicate symbol of arity n, the assignment of a function from MxD^n to $\{true, false\}$. We thus say that I is *based on* J. Let J be a pre-interpretation with domain D and universe M. Let I be an interpretation based on J and V a variable assignment. To every possible world and formula can be attributed a truth value, *true* or *false*, (*with respect to* I and V) as follows: if the formula is of the form $\Box F$ ($\Diamond F$) then its truth value in w is *true* if and only if, in every (some) possible world accessible from w using R, the truth value of F is *true*. The *Herbrand pre-interpretation* is a pre-interpretation whose domain is $U_{\mathcal{L}}$. A *Herbrand*

interpretation of $\mathcal{L}$ is an interpretation based on the Herbrand pre-interpretation.

A formula is *satisfiable* (*valid*) if it is true in some (every) possible world of some (every) interpretation. It is a *valid consequence* of some set of formulas if it is true in every possible world (of every interpretation) satisfying every formula of this set. Let us consider the set of valid formulas. It is not recursively enumerable. As a matter of fact, validity (in the transitive and reverse well-founded interpretations we are considering) is highly indecidable: it is $\prod_2^1$-complete in the analytical hierarchy [9]. If the language is restricted to its propositional part then one gets *Pr*, the (decidable) propositional modal logic of provability. Its axiom schemata and inference rules are those of the classical propositional calculus plus:

(a) $(\Box A \leftarrow \Box B) \leftarrow \Box(A \leftarrow B)$.

(b) $\Box A \leftarrow \Box(A \leftarrow \Box A)$.

(c) if $\vdash_{Pr} A$ then $\vdash_{Pr} \Box A$.

This modal logic is of importance because of its relationship with provability in arithmetic. For further informations, we suggest the reader consult the book by Boolos [8]. As far as we know, the following results together with the previous ones presented in [4] and [5] are the first use of this modal logic for the semantical characterization of a programming language.

In [4] was defined a modal completion formula $comp_{CWA}(P)$ of any definite logic program *P*. It was proved that there was no *SLD* refutation of $P \cup \{\leftarrow A\}$ if and only if $\Box A$ is a valid consequence of $comp_{CWA}(P)$ in transitive and reverse well-founded interpretations. In [5] was defined a modal completion formula $comp_D(P)$ of any normal logic program *P*. It was proved that if the program is stratified then *A* belongs to its natural interpretation as it has been defined in [2] if and only if $\Box A$ is a valid consequence of $comp_D(P)$ in transitive and reverse well-founded interpretations.

We would like to define a modal completion formula $comp_I(P)$ of any normal logic program *P* such that there is an infinite *SLDNF* computation of $P \cup \{\leftarrow A\}$ if and only if $\Box A$ is a valid consequence of $comp_I(P)$ in transitive and reverse well-founded interpretations.

Let $p(t_1,...,t_n) \leftarrow L_1,...,L_m$ be a clause of a normal program *P*. The first step is to transform it into the formula $p(x_1,...,x_n) \leftarrow ((\Box L_1 \vee ... \vee \Box L_m) \wedge (x_1 = t_1) \wedge ... \wedge (x_n = t_n))$ where the variables x_1, ..., x_n do not appear in *P*. If y_1, ..., y_d are the variables of the original clause then we transform this formula into the formula $p(x_1,...,x_n) \leftarrow \exists y_1 ... \exists y_d$

$((\Box L_1^* \vee \ldots \vee \Box L_m^*) \wedge (x_1 = t_1) \wedge \ldots \wedge (x_n = t_n))$. Now suppose this transformation has been made for every clause in the definition of p. Then we have k transformed clauses of the form $p(x_1,\ldots,x_n) \leftarrow E_1$, ..., $p(x_1,\ldots,x_n) \leftarrow E_k$ where each E_i is of the form $\exists y_1 \ldots \exists y_d ((\Box L_1^* \vee \ldots \vee \Box L_m^*) \wedge (x_1 = t_1) \wedge \ldots \wedge (x_n = t_n))$. Then, the *completed definition of infiniteness* of p is the formula $\forall x_1 \ldots \forall x_n \, \Box(p(x_1,\ldots,x_n) \leftarrow E_1 \vee \ldots \vee E_k)$.

Furthemore we add the following modal equational theory:

1. $\Box(c \neq d)$, for every pair c, d of distinct constants.
2. $\Box(f(x_1,\ldots,x_n) \neq g(y_1,\ldots,y_m))$, for every pair f, g of distinct function symbols.
3. $\Box(f(x_1,\ldots,x_n) \neq c)$, for every constant c and every function symbol f.
4. $\Box(t[x] \neq x)$, for every term $t[x]$ containing x but distinct from x.
5. $\Box((x_1 \neq y_1) \vee \ldots \vee (x_n \neq y_n) \rightarrow f(x_1,\ldots,x_n) \neq f(y_1,\ldots,y_n))$, for every function symbol f.
6. $\Box(x = x)$.
7. $\Box((x_1 = y_1) \wedge \ldots \wedge (x_n = y_n) \rightarrow f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n))$, for every function symbol f.
8. $\Box((x_1 = y_1) \wedge \ldots \wedge (x_n = y_n) \rightarrow (p(x_1,\ldots,x_n) \rightarrow p(y_1,\ldots,y_n)))$, for every predicate symbol p (and for the predicate symbol of equality too).

Let P be a normal program. The *completion of infiniteness* of P, $comp_I(P)$, is the collection of all the completed definitions of infiniteness of the predicate symbols of $\mathcal{L}$ together with the modal equational theory. Let G be the normal goal $\leftarrow L_1,\ldots,L_n$ and θ a substitution of the variables of G. We will say that θ is a *correct answer* for $comp_I(P) \cup \{G\}$ if $\exists((\Box L_1^* \vee \ldots \vee \Box L_n^*)\theta)$ is a valid consequence of $comp_I(P)$. This notion of correct answer is the semantical counterpart of the procedural notion of a computed answer of infiniteness.

Now we define a mapping T_P^J on the lattice of interpretations (based on some pre-interpretation J of the language) to itself. If J is a classical pre-interpretation of $\mathcal{L}$ and if I is a classical interpretation based on J then $T_P^J(I) = \{B$: for some variable assignment V with respect to J, there is a clause $A \leftarrow L_1,\ldots,L_n$ in P such that $A_{J,V} = B$ and $L_1^* \vee \ldots \vee L_n^*$ is true with respect to I and $V\}$. When J is the classical Herbrand pre-interpretation, we will write T_P instead of T_P^J. The mapping T_P^J possesses some properties.

proposition 4.1 For every pre-interpretation J of $\mathcal{L}$ and for every normal program P, *the mapping* $T_P{}^J$ *is continuous.*

Note that, for every program P, $lfp(T_P{}^J) = \emptyset$. In other respects, it is not always true that $gfp(T_P) = T_P{\downarrow}\omega$. If $P = \{A \leftarrow B(x), B(f(x)) \leftarrow B(x)\}$, then $gfp(T_P) = \emptyset$ and $T_P{\downarrow}\omega = \{A\}$. A first important result is that the domain of interpretation of the terms of the language is not essential.

theorem 4.2 Let P be a normal program and G the normal goal $\leftarrow L_1,...,L_k$. Let θ be a substitution of the variables in G. *The following assertions are equivalent:*
(a) θ *is a correct answer* for $comp_I(P) \cup \{G\}$, *that is to say:* $\exists(\Box L_1{}^*\theta \vee ... \vee \Box L_k{}^*\theta)$ *is true in every model of* $comp_I(P)$.
(b) $\exists(\Box L_1{}^*\theta \vee ... \vee \Box L_k{}^*\theta)$ *is true in every Herbrand model of* $comp_I(P)$.

The proof of theorem 4.2 is not essential for our purpose. It could be done by induction on the model of $comp_I(P) \cup \{\neg\exists(\Box L_1{}^*\theta \vee ... \vee \Box L_k{}^*\theta)\}$. See [6] for the exact details. The result of theorem 4.2 will greatly simplified the presentation of future proofs. Especially the soundness proof of the infiniteness of *SLDNF* resolution we present now.

theorem 4.3 Let P be a normal program and G the normal goal $\leftarrow L_1,...,L_k$. *Every computed answer of infiniteness of* $P \cup \{G\}$ *is a correct answer for* $comp_I(P) \cup \{G\}$.

proof Let θ be a substitution of the variables of G which is not a correct answer for $comp_I(P) \cup \{G\}$. Theorem 4.2 says there is a Herbrand model of $comp_I(P) \cup \{\neg\exists(\Box L_1{}^*\theta \vee ... \vee \Box L_k{}^*\theta)\}$. Let I be a Herbrand interpretation with universe M and accessibility relation R which is a model of $comp_I(P) \cup \{\neg\exists(\Box L_1{}^*\theta \vee ... \vee \Box L_k{}^*\theta)\}$. Let w be a possible world of M where $comp_I(P) \cup \{\neg\exists(\Box L_1{}^*\theta \vee ... \vee \Box L_k{}^*\theta)\}$ is true and such that, for every possible world w' accessible from w using R, for every normal goal $\leftarrow l_1,...,l_q$ and for every substitution σ of the variables of that goal, if $comp_I(P) \cup \{\neg\exists(\Box l_1{}^*\sigma \vee ... \vee \Box l_q{}^*\sigma)\}$ is true in w' then σ is not a computed answer of infiniteness of $P \cup \{\leftarrow l_1,...,l_q\}$. Now for every $j=1,..., k$

and for every variable assignment V with respect to the Herbrand pre-interpretation, there is a possible world $w_{j,V}$ accessible from w and where $comp_I(P) \cup \{\neg L_j^* \theta\} \cup \{\forall x_1 ... \forall x_n (p_j(x_1, ..., x_n) \leftarrow E_1 \vee ... \vee E_k)\}$ is true with respect to V, p_j being the predicate symbol of the atom L_j^*. Let: $L_j^* \theta = p_j(t'_1, ..., t'_n)$. Thus, for every clause $p_j(t_1, ..., t_n) \leftarrow l_1, ..., l_q$ in P, $\exists y_1 ... \exists y_k ((\Box l_1 \vee ... \vee \Box l_q)) \wedge (t_1 = t'_1) \wedge ... \wedge (t_n = t'_n))$ is false in $w_{j,V}$ with respect to V. Let σ be an *mgu* of $p_j(t_1, ..., t_n)$ and $p_j(t'_1, ..., t'_n)$. $comp_I(P) \cup \{\neg \exists ((\Box l_1^* \sigma \vee ... \vee \Box l_q^* \sigma)\}$ is true in $w_{j,V}$ and, by induction hypothesis, σ is not a computed answer of infiniteness of $P \cup \{\leftarrow l_1, ..., l_q\}$. Consequently, θ is not a computed answer of infiniteness of $P \cup \{\leftarrow L_1, ..., L_k\}$.

corollary 4.4 *If there is an infinite SLDNF computation of $P \cup \{G\}$ then $comp_I(P) \cup \{\leftarrow \exists (\Box L_1^* \vee ... \vee \Box L_n^*)\}$ is insatisfaisable.*

corollary 4.5 *The infiniteness set of P is contained in the set $\{A \in B_{\mathcal{L}} : \Box A$ is a valid consequence of $comp_I(P)\}$.*

Now we give the point of view of the fixpoint operator.

theorem 4.6 Let P be a normal program and G the normal goal $\leftarrow L_1, ..., L_m$. *If θ is a computed answer of infiniteness of rank n of $P \cup \{G\}$ then $\cup_{j=1..m} [L_j^* \theta] \cap T_P \downarrow n \neq \emptyset$.*

proof The following induction on n proves that if $\cup_{j=1..m} [L_j^* \theta] \subseteq B_{\mathcal{L}} \setminus T_P \downarrow n$ then θ is a computed answer of finiteness of rank n of $P \cup \{G\}$. Suppose it is true for α-*1*. Let $n = \alpha$. If $\cup_{j=1..m} [L_j^* \theta] \subseteq B_{\mathcal{L}} \setminus T_P \downarrow n$ then, for every $L_j \theta$ in $G\theta$ and for every substitution σ such that $L_j \theta \sigma$ is ground, $L_j^* \theta \sigma \notin T_P \downarrow n$, that is to say: for every variable assignment V with respect to the Herbrand pre-interpretation and for every clause $B \leftarrow l_1, ..., l_q$ in P, if $B_V = B_j \theta \sigma$ then $l_1^* \vee ... \vee l_q^*$ is false with respect to $T_P \downarrow \alpha$- *1* and V. Thus, for every clause $B \leftarrow l_1, ..., l_q$ in P and for every substitution σ, if $\sigma = mgu(B, B_j \theta)$ then $\cup_{k=1..q} [l_k^* \sigma] \subseteq B_{\mathcal{L}} \setminus T_P \downarrow \alpha$-*1* and, by induction hypothesis, σ is a computed answer of finiteness of rank α-*1* of $P \cup \{\leftarrow l_1, ..., l_q\}$. Consequently, θ is a computed answer of finiteness of rank n of $P \cup \{G\}$.

note Theorem 4.6 does not imply that if there is an infinite *SLDNF* computation of $P\cup\{\leftarrow L_1,...,L_m\}$ then $\cup_{j=1..m}[L_j^*]\cap T_P\downarrow\omega \neq \varnothing$. As a matter of fact, if $P = \{A(s(x))\leftarrow A(x)\}$ and $G = \leftarrow A(y)$ then $\cup_{j=1..m}[L_j^*] = \{A(s^n(0)): n\ 0\}$ and $T_P\downarrow\omega = \varnothing$.

corollary 4.7 *The set of infiniteness of a normal program P is contained in* $T_P\downarrow\omega$.

As for the completeness proof of the infiniteness of *SLDNF* resolution, it has been done using the fact that *SLDNF* trees are finitely branching: if $P\cup\{G\}$ is allowed and if there is no infinite *SLDNF* computation of $P\cup\{G\}$ then every *SLDNF* tree of $P\cup\{G\}$ is finite. More precisely: some integer is greater than the depth of *SLDNF* trees of $P\cup\{G\}$.

theorem 4.8 Let P be a normal program and G the normal goal $\leftarrow L_1,...,L_m$. *If* $P\cup\{G\}$ *is allowed and if* $comp_I(P)\cup\{\leftarrow\exists(\Box L_1^*\vee...\vee\Box L_n^*)\}$ *is insatisfaisable then there is an infinite SLDNF computation of* $P\cup\{G\}$.

proof If there is no infinite derivation of $P\cup\{G\}$ then some integer *max* is greater than the length of any *SLDNF* derivation of $P\cup\{G\}$. The proof is straightforward and can be done by induction on *max*.

proposition 4.9 *The set of infiniteness of an allowed normal program P is equal to* $T_P\downarrow\omega$ *and is equal to the set* $\{A\in B_{\mathcal{L}}: \Box A$ *is a valid consequence of* $comp_I(P)\}$.

proof If $A \notin T_P\downarrow\omega$ then, for some integer n, $A \notin T_P\downarrow n$. The proof is straightforward and can be done by induction on n.

5 Recursive Computation Rules

We have just characterized in provability modal logic a property of infiniteness of *SLDNF* resolution: if P is allowed then there is an infinite *SLDNF* computation of $P\cup\{\leftarrow A\}$ if and only if $\Box A$ is a valid consequence of $comp_I(P)$ in the class of modal interpretations whose accessibility relation is transitive and reverse well-founded. This characterization does not give us a choice procedure of an atom in a goal such that if there is an infinite *SLDNF* computation of $P\cup\{G\}$ then there is an infinite *SLDNF*

derivation which uses this procedure: validity in the class of transitive and reverse well-founded interpretations is Π_2^1-complete [9]. Our characterization does not say however that such a procedure cannot exist. Now the question is to see whether there could be an algorithm of selection of atoms in goals always leading into an infinite derivation when such a derivation exists. Such an algorithm is a recursive maximal computation rule for the infiniteness of *SLDNF* resolution. Considering definite logic programs and *SLD* resolution, we will show that such a rule cannot exist.

A computation rule is *maximal for the refutation* of *SLD* resolution when, for every definite program P and for every definite goal G, if there is an *SLD* refutation of $P \cup \{G\}$ then there is an *SLD* refutation of $P \cup \{G\}$ using this rule. An essential result of the theory of logic programming is the independence of the computation rule for the refutation of *SLD* resolution, that is to say: every computation rule is maximal for the refutation of *SLD* resolution [14]. A computation rule is *maximal for the finite failure* of *SLD* resolution when if there is a finitely failed *SLD* tree of $P \cup \{G\}$ then the *SLD* tree of $P \cup \{G\}$ using this rule is finitely failed. An important result is the independence of the computation rule, as far as it is fair, for the finite failure of *SLD* resolution [12]. Similarly, it is not difficult to prove that every fair computation rule is maximal for the finiteness of *SLD* resolution.

Now we consider the maximality of a computation rule with respect to the infiniteness of *SLD* resolution. A computation rule is *maximal for the infiniteness* of *SLD* resolution when, for every program P and for every goal G, if there is an infinite *SLD* tree of $P \cup \{G\}$ then the *SLD* tree of $P \cup \{G\}$ using this rule is infinite. Some rules can be maximal for the infiniteness of *SLD* resolution. A computation rule is *recursive* if it is an algorithm for the selection of an atom in a goal. Such rules cannot be maximal for the infiniteness of *SLD* resolution.

theorem 5.1 *There is no recursive computation rule maximal for the infiniteness of SLD resolution.*

proof As a matter of fact, we prove that, for some definite program P, there can be no recursive computation rule maximal for the infiniteness of *SLD* resolution in P. The proof is based on an idea developed by Shepherdson [17] who proved that no recursive rule can be maximal for the refutation of *SLDNF* resolution. Let A and B be two recursively enumerable recursively inseparable disjoint sets [15]. Let f and g be two unary functions enumerating A and B. Let F and G be the partial recursive functions defined as follows: $F(x) = \mu y(f(y)=x)$; $G(x) = \mu y(g(y)=x)$. $F(x)$, respectively: $G(x)$, is,

when it exists, the least integer y such that $f(y) = x$, respectively: $g(y) = x$. Now, $F(x)$ is defined if and only if $x \in A$ and $G(x)$ is defined if and only if $x \in B$. Let P_F be the imperative program using the variables X_F and Y_F, and instructions like:

(a) $[i]\ X_F := X_F + 1$
(b) $[j]\ IF\ X_F\ \ 0\ THEN\ X_F := X_F - 1\ AND\ GOTO\ [j']$

for X_F and similar instructions for Y_F, and such that, for every input $(X_F, Y_F) = (x,0)$, the program stops if and only if $F(x)$ is defined. Let P_{F*} be the logic program obtained from P_F as follows:

(1) Replace every instruction of type (a) by the clause $p_{F,i}(X,Y) \leftarrow p_{F,i+1}(s(X),Y)$.
(2) Replace every instruction of type (b) by the clauses $p_{F,j}(s(X),Y) \leftarrow p_{F,j'}(X,Y)$ and $p_{F,j}(0,Y) \leftarrow p_{F,j+1}(0,Y)$.

Let P_{F**} and P_{G**} be the programs obtained from P_{F*} and P_{G*} respectively by the addition of the clauses $p_{F,0}(F^*(X),X) \leftarrow p_{F,1}(X,0)$ and $p_{G,0}(G^*(X),X) \leftarrow p_{G,1}(X,0)$. Let P be the program $P_{F**} \cup P_{G**}$. Let G_n be the goal $\leftarrow p_{F,0}(X,n), p_{G,0}(X,n)$. If we want to find an infinite *SLD* derivation, we have the choice of the selection of an atom at the first step of the computation only. If we choose the first atom then there is an infinite *SLD* derivation if and only if $n \notin A$. Otherwise, there is an infinite *SLD* derivation if and only if $n \notin B$. Let R be a recursive computation rule. It corresponds to a recursive set C such that R selects the first atom of G_n if and only if $n \in C$. Thus, if R is maximal for the infiniteness of *SLD* derivation then $A \subseteq \mathbb{N} \setminus C$ and $B \subseteq C$, which is impossible since A and B are recursively inseparable.

6 Conclusion

We have given a semantical characterization of finite and infinite *SLDNF* derivations. The completion formulas we have defined were both sound and complete for the finiteness and infineteness of *SLDNF* resolution: if P is allowed then there is no (an) infinite *SLDNF* computation of $P \cup \{\leftarrow A\}$ if and only if A ($\Box A$) is a valid consequence of $comp_F(P)$ ($comp_I(P)$) in classical first order predicate calculus (in transitive and reverse well-founded modal interpretations). The characterization of infinite *SLDNF* computations that was presented in theorems 4.3 and 4.8 constitutes a first step towards

a better understanding of the modal semantics of perpetual processes.

However, the result stated in theorem 5.1 is not very encouraging from the point of view of using *PROLOG* for concurrent applications. As a matter of fact, what is asked to a perpetual processes is to carry a computation which never ends. This computation has to be defined with the help of a recursive rule. As we have proved, such a rule cannot exist. Consequently, we have to circumscribe classes of definite logic programs for which recursive rules maximal for the infiniteness of *SLD* resolution exist. The notion of a perpetual processes makes sense for these classes of programs only. Now the question is the nature of these classes of programs.

Notes

(1) 58 avenue de la république, 93110 Rosny-sous-bois, France

(2) Including the binary predicate symbol of equality.

(3) This is a direct consequence of König's lemma.

References

1. K. R. Apt, G. Bezem: Acyclic programs. In: P. Szeredi, D. H. D. Warren (eds.): Proceedings of the Seventh International Conference on Logic Programming. Massachussets Institute of Technology Press 1990, pp. 617-633

2. K. R. Apt, H. A. Blair, A. Walker: Towards a theory of declarative knowledge. In: J. Minker (ed.): Foundations of Deductive Databases and Logic Programming. Los Altos: Morgan Kaufmann 1988, pp. 89-148

3. K. R. Apt, R. N. Bol, J. W. Klop: On the power of subsumption and context checks. In: A. Miola (ed.): Design and Implementation of Symbolic Computation Systems. Lecture Notes in Computer Science 429. Berlin: Springer 1990, pp. 131-140

4. P. Balbiani: A modal semantics for the negation as failure and the closed world assumption rules. In: C. Choffrut, M. Jantzen (eds.): Eighth Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science 480. Berlin: Springer 1990, pp. 523-534

5. P. Balbiani: A modal semantics of negation in logic programming. Fundamenta Informaticæ (to appear)

6. P. Balbiani: Sur la finitude des dérivations de la programmation en logique. rapport IRIT/92-3-R

7. M. Bezem: Characterizing termination of logic programs with level

mapping. In: E. L. Lusk, R. Overbeek (eds.): Proceedings of the North American Conference on Logic Programming. Massachussets Institute of Technology Press 1989, pp. 69-80

8. G. Boolos: The unprovability of inconsistency. Cambridge: Cambridge University Press 1979

9. G. Boolos, G. Sambin: Provability: the emergence of a mathematical modality. Studia Logica, 1-23 (1991)

10. K. L. Clark: Negation as failure. In: H. Gallaire et J. Minker (eds.): Logic and Data Bases. New York: Plenum Press 1978, pp. 293-322

11. F. Denis: Contribution à l'étude des sémantiques axiomatiques de Prolog. thèse de l'université des sciences et techniques de Lille Flandres Artois, 1990

12. J.-L. Lassez, M. J. Maher: Closures and fairness in the semantics of programming logic. Theoretical Computer Science 29, 167-184 (1984)

13. J.-L. Lassez, M. J. Maher, D. A. Wolfram: A unified treatment of resolution strategies for logic programs. In: S.-A. Tarnlund (ed.): Second International Conference on Logic Programming. Uppsala: Uppsala University Press 1984, pp. 263-276

14. J. W. Lloyd: Foundations of logic programming. Berlin: Springer 1987

15. P. Odifreddi: Classical recursion theory. Amsterdam: North-Holland 1989

16. L. Plümer: Termination proofs for logic programs. Lecture Notes in Artificial Intelligence 446. Berlin: Springer 1990

17. J. C. Shepherdson: Unsolvable problems for SLDNF resolution. Journal of Logic Programming 10, 19-22 (1991)

18. J. D. Ullman, A. van Gelder: Efficient tests for top-down termination of logical rules. Journal of the Association for Computing Machinery 35, 345-373 (1988)