UNFOLDING, PROCEDURAL AND FIXPOINT SEMANTICS OF LOGIC PROGRAMS

François DENIS Jean-Paul DELAHAYE
L.I.F.L., CNRS UA 369,
Université des Sciences et des Techniques de Lille, Flandres Artois
59655 Villeneuve d'Ascq CEDEX FRANCE
denis@lifl.lifl.fr delahaye@lifl.lifl.fr

Abstract

In this paper, we propose new notions related to unfolding semantics which clarify the relationship between procedural and fixpoint semantics of logic programs. We introduce 3 main new ideas :

- the unfolding rule has an **associativity property** that ensures the adequacy of unfolding semantics with the procedural semantics
- the **uniformly fair computation rules** which guarantees strong procedural equivalences
- the notion of **operator compatible with unfolding** which provides the connection between the unfolding and fixpoint semantics and gives new tools for proving old and new results of equivalence between procedural and fixpoints semantics of logic programs.

I. Introduction

Semantics of logic programs has recently been described by means of a program transformation technique based on an unfolding rule [Kan 87, Lev 88, Sek 89, Tam 84]. We define in section II the unfolding rule as an operation (denoted by Unf) on definite programs.

We show in section II that this operation has an interesting **associativity property**. For any definite programs P, Q and R, we have : Unf(P, Unf(Q, R)) = Unf(Unf(P, Q), R). This property has many consequences :

- Two natural ways to define the iterate programs P^n (namely, P^{n+1} = Unf(P, P^n) and P^{n+1} = Unf(P^n, P)) lead to identical results. This can be viewed as a new expression of the well-known fact that top-down and bottom-up resolutions are equivalent.
- Iterate programs P^n and P have the same procedural semantics (in a strong sense, that is, including computed answers and finite failure).

- We can define the infinite unfolding P^{ω} of a definite program P in a more general way than in [Lev 88]. This program contains only unit clauses and clauses on the form $A \leftarrow \infty$ and may be considered as "the procedural semantics" of P (Section III).

We show next that the equivalence of the two previous definitions of P^n may be seen as a particular case of a strong equivalence of some computation rules that we define in section IV : the **p-uniformly fair computation rules**. This class of computation rules is interesting in itself because it may be used to describe the implementation of Prolog which delays the selection of some atoms within a goal.

Many different aspects of the procedural semantics of a Prolog program (the success set [Emd 76], the finite failure set [Cla 78], the set of goals which have a computed answer equal to identity [Fal 89], the set of (G, θ) where G is a goal and θ is a computed answer [Fal 89]) have been described in terms of the least fixpoint of a continuous transformation T (or $T\uparrow\omega$ if T is not continuous) associated with it. For each operator the correspondence between $T\uparrow\omega$ and the procedural studied set has to be proved. These proofs, by induction on n, are often quite long and tedious because they have to show the connection between one step of the SLD-resolution and one step of the iteration of the operator T. We propose a new proof technique based on unfolding.

We say that an operator is **compatible with the unfolding rule** if for every definite program P and Q, $T_{Unf(P,Q)}(\varnothing) = T_P(T_Q(\varnothing))$. To prove this property needs no induction step and it is verified by all the usual operators. For such an operator, we have : $T_{P^n}(\varnothing) = T_P\uparrow n$ and $T_{P^{\omega}}(\varnothing) = T_P\uparrow\omega$ which is the expected result. We give a detailed proof for the T^S operator defined in [Fal 89].

These results may be extended to operators which take the negation as finite failure, or the interpreter search rule strategy, into account ([Den 89], [Del 88]).

II. Associativity of unfolding

Definition II.1: *Let* P *be a definite program and* $C : A \leftarrow B_1, ..., B_m$ $(m \geq 0)$ *be a definite clause. Let us define the unfolding of* C *with respect to* P *as in* [Lev 88]. Unf(C,P) *is the following set of clauses :*

i) {C} *if* $m = 0$ (C *is a unit clause*)
ii) $\{A\theta \leftarrow (L_1, ..., L_m)\theta$ *where*
- $D_i \leftarrow L_i$ *is a clause of* P, *for* $i = 1..m$ (L_i *is a list of atoms*)
- $\theta = mgu((B_1, ..., B_m),(D_1, ..., D_m))\}$ (mgu : *most general unifier*)

Remark : A mgu of two m-tuples $(B_1, ..., B_m)$ and $(C_1, ..., C_m)$ of atoms is a minimal substitution θ such that $B_i\theta = C_i\theta$ for $i = 1..m$. It is unique modulo variable renaming.
We mean by $\theta = mgu((B_1, ..., B_m),(C_1, ..., C_m))$ that
- $(B_1, ..., B_m)$ and $(C_1, ..., C_m)$ are unifiable and
- θ is a mgu of the m-tuples.

Definition II.2 : *Let* P = $\{C_1, ..., C_m\}$ *and* Q *be two definite programs. The unfolding of* P *wrt* Q *is* ([Lev 88]) :

$$Unf(P,Q) = \cup \{ Unf(C_i, Q) ; i = 1..m\}$$

Example : Let P be the program :
$$\begin{cases} A(x) \leftarrow B(x),C(x) & C_1 \\ B(a) \leftarrow B(a) & C_2 \\ B(b) \leftarrow & C_3 \\ C(a) \leftarrow & C_4 \\ C(b) \leftarrow & C_5 \end{cases}$$

$$Unf(A(x) \leftarrow B(x),C(x) ; P) = \begin{cases} A(a) \leftarrow B(a) \\ A(b) \leftarrow \end{cases} \qquad Unf(P,P) = \begin{cases} A(a) \leftarrow B(a) \\ A(b) \leftarrow \\ B(a) \leftarrow B(a) \\ B(b) \\ C(a) \\ C(b) \end{cases}$$

Notation : If E is a set of atoms, Var(E) denotes the set of variables occurring in the atoms of E.

We need the following associativity result to prove the next theorem.

Proposition II.1 : *Let* $\mathcal{A} = \{A_i\}_{1\le i\le n}$ *be a set of positive literals,* $\mathcal{B} = \{B_i \leftarrow B_{i,1}, ..., B_{i,n_i}\}_{1\le i\le n}$ *and* $\mathcal{C} = \{C_{i,j} \leftarrow C_{i,j,1}, ..., C_{i,j,p_{i,j}}\}_{1\le i\le n,\ 1\le j\le n_i}$ *be sets of definite clauses.*
Let us further assume that Var($\mathcal{A}$), Var($\mathcal{B}$) *and* Var($\mathcal{C}$) *are disjoint sets. Then,*
there exist substitutions α *and* β *such that :*

(I) $\begin{cases} \alpha = mgu((A_i)_{1\le i\le n},(B_i)_{1\le i\le n}) \\ \textit{and} \\ \beta = mgu((B_{i,j}\alpha)_{1\le i\le n,\ 1\le j\le n_i},\ (C_{i,j})_{1\le i\le n,\ 1\le j\le n_i}) \end{cases}$

iff

there exist substitutions γ *and* δ *such that :*

(II) $\begin{cases} \gamma = mgu((B_{i,j})_{1\le i\le n,\ 1\le j\le n_i},\ (C_{i,j})_{1\le i\le n,\ 1\le j\le n_i}) \\ \textit{and} \\ \delta = mgu((A_i)_{1\le i\le n},(B_i\gamma)_{1\le i\le n}). \end{cases}$

Moreover, if (I) *or* (II) *holds then* $\alpha\beta = \gamma\delta$ *(modulo variable renaming).*

Proof : We omit the range of indexes in the proof for easier reading.
a) Suppose first that (I) holds.
We can suppose that Var($\mathcal{C}$) remains invariant under substitution α.

For all i, j, we have : $B_{i,j}\alpha\beta = C_{i,j}\beta = C_{i,j}\alpha\beta$, then $(B_{i,j})$ and $(C_{i,j})$ are unifiable. Let γ be their mgu.
We can suppose that $Var(\mathcal{A} \cup \mathcal{B})$ remains invariant under substitution γ.
We have $\gamma \leq \alpha\beta$. Let σ be a substitution such that $\alpha\beta = \gamma\sigma$. For all i, we have : $A_i\sigma = A_i\gamma\sigma = A_i\alpha\beta = B_i\alpha\beta = B_i\gamma\sigma$, hence (A_i) and $(B_i\gamma)$ are unifiable and let δ be their mgu.
We have $\delta \leq \sigma$, therefore $\gamma\delta \leq \gamma\sigma = \alpha\beta$.

b) Suppose now that (II) holds.
We can suppose that $Var(\mathcal{A} \cup \mathcal{B})$ remains invariant under substitution γ.
For all i, we have $A_i\delta = A_i\gamma\delta = B_i\gamma\delta$ then (A_i) and (B_i) are unifiable. Let α be their mgu.
We can suppose that $Var(\mathcal{C})$ remains invariant under substitution α.
We have $\alpha \leq \gamma\delta$. Let τ be a substitution such that $\alpha\tau = \gamma\delta$.
For all i,j, we have : $B_{i,j}\alpha\tau = B_{i,j}\gamma\delta = C_{i,j}\gamma\delta = C_{i,j}\alpha\tau = C_{i,j}\tau$ then $(B_{i,j}\alpha)$ and $(C_{i,j})$ are unifiable.
Let β be their mgu. We have $\beta \leq \tau$ then $\alpha\beta \leq \alpha\tau = \gamma\delta$.

c) If (I) or (II) holds, we have $\alpha\beta \leq \gamma\delta \leq \alpha\beta$. That is, $\alpha\beta = \gamma\delta$ modulo variable renaming. □

Theorem II.1: *Let* P, Q *and* R *be definite programs.*

$$Unf(P, Unf(Q,R)) = Unf(Unf(P,Q), R) \text{ (modulo variable renaming)}$$

Proof : In order to build any of these two sets, we need to consider :

- every clause of P : $A \leftarrow A_1, ..., A_n$
- every n-tuple of clauses of Q : $B_i \leftarrow B_{i,1}, ..., B_{i,n_i}$
- every Σn_i-tuple of clauses of R : $C_{i,j} \leftarrow C_{i,j,1}, ..., C_{i,j,p_{i,j}}$.

We then get a clause of Unf(P, Unf(Q,R)) when $(B_{i,j})$ and $(C_{i,j})$ are unifiable (with mgu γ) and when (A_i) and $(B_i\gamma)$ are unifiable (with mgu δ) which is : $A\delta \leftarrow C_{i,j,k}\gamma\delta$ or $A\gamma\delta \leftarrow C_{i,j,k}\gamma\delta$.

We get a clause of Unf(Unf(P,Q), R) when (A_i) and (B_i) are unifiable (with mgu α) and when $(B_{i,j}\alpha)$ and $(C_{i,j})$ are unifiable (with mgu β) which is : $A\alpha\beta \leftarrow C_{i,j,k}\beta$ or $A\alpha\beta \leftarrow C_{i,j,k}\alpha\beta$.

The previous lemma shows that for any clause from Unf(P, Unf(Q,R)) there exists an equal (modulo variable renaming) clause from Unf(Unf(P,Q), R), and vice versa. Thus, the two sets of clauses are equal (modulo variable renaming). □

There are two natural ways to define inductively the "unfolding powers" of a definite program.

Let P be a definite program. Let us define :

$$\text{a) } P_a^1 = P;\ P_a^{n+1} = Unf(P_a^n, P) \quad \text{and} \quad \text{b) } P_b^1 = P;\ P_b^{n+1} = Unf(P, P_b^n).$$

The next corollary says that these two different definitions in fact define the same object.

Corollary II.1 : *For any definite program* P *and any integer* $n \geq 1$,

$$P_a^n = P_b^n \text{ (modulo variable renaming).}$$

Proof : Cases n=1 and n=2 are obvious.
Assume that $P_a^n = P_b^n$ for $n \geq 2$.
$P_a^{n+1} = Unf(P_a^n, P) = Unf(P_b^n, P) = Unf(Unf(P, P_b^{n-1}), P)$
$= Unf(P, Unf(P_b^{n-1}, P)) = Unf(P, Unf(P_a^{n-1}, P)) = Unf(P, P_a^n) = Unf(P, P_b^n) = P_b^{n+1}$. □

Remark : The definition contained in [Lev 88] is : $P^{n+1} = Unf(P^n, P^n)$. The definition b) is close to the notion of partial evaluation defined in [Llo 87].

Corollary II.2 : *For any definite program* P *and any integers* n, $m \geq 1$,

a) $Unf(P^n, P^m) = P^{n+m}$ b) $(P^n)^m = P^{nm}$

Proof :
a) By induction on m. Case m = 1 is obvious. Suppose that this relation holds for $m \geq 1$.
$Unf(P^n, P^{m+1}) = Unf(P^n, Unf(P^m, P)) = Unf(Unf(P^n, P^m), P)$
$= Unf(P^{n+m}, P) = P^{n+m+1}$.
b) $(P^n)^{m+1} = Unf(P^n, (P^n)^m) = Unf(P^n, P^{nm}) = P^{nm+n} = P^{n(m+1)}$. □

Corollary II.3 : *Let* C *be a definite clause and* P *be a definite program. Define for every integer* $n \geq 1$, *the definite program* C_n *by :* $C_1 = C$ *and* $C_{n+1} = Unf(C_n, P)$.
We have, for any integer $n \geq 1$: $C_{n+1} = Unf(C, P^n)$.

Proof : By induction on n. Case n = 1 is obvious. Suppose that this relation holds for $n \geq 1$.
$C_{n+2} = Unf(C_{n+1}, P) = Unf(Unf(C, P^n), P) = Unf(C, Unf(P^n, P)) = Unf(C, P^{n+1})$.

III. Unfolding and procedural semantics

We show in this section that unfolding preserves the procedural semantics of logic programs in a strong sense, i.e. taking finite failure and computed answers into account.

Definition III.1: *Let* P *be a definite program,* $\delta : G_0,...,G_n$ *be a* SLD-*derivation of* $P \cup \{G_0\}$, $i \in [1..n]$ *and let* A *be an atom of* G_i. *Let us define the* **depth of A** *in* G_i (*denoted by* $depth(A,G_i)$) *as* :

- $depth(A,G_i) = 0$ *if* $i = 0$.
- $depth(A,G_i) = depth(B, G_{i-1}) + 1$ *if* B *is the selected atom in* G_{i-1} *and* A *results from the resolution of* B *with a clause of* P.
- $depth(A,G_i) = depth(B,G_{i-1})$ *if* A *results from an instanciation of* B *in* G_{i-1} *and* B **is not** *the selected atom in* G_{i-1}.

Definition III.2 : *The* **depth of a SLD-derivation** δ *is the maximal depth of the atoms in the goals of* δ.

Remark : This notion of depth of a derivation δ corresponds to the usual notion of depth of the *proof tree* associated with δ [Der 87].

Definition III.3: *Let us denote by* $\mathfrak{B}$ *the computation rule which selects the first atom of minimal depth in the last goal of a* SLD-*derivation.*

Remark : The derivations computed by $\mathfrak{B}$ correspond to proof-trees computed using a breadth-first strategy [Der 87]. $\mathfrak{B}$ is a *fair* computation rule [Las 84] (i.e. for every infinite SLD-derivation, every atom A occurring in the derivation (or some further instanciated version of A) is selected by $\mathfrak{B}$.

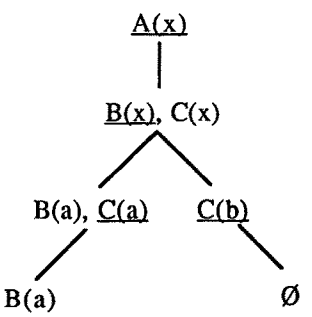
Example : For program P from the previous example,
$\leftarrow \underline{A(x)} \xrightarrow{C_1} \leftarrow \underline{B(x)},C(x) \xrightarrow{C_2} B(a), \underline{C(a)} \xrightarrow{C_4} \underline{B(a)} \xrightarrow{C_2} B(a)$ is a derivation computed by $\mathfrak{B}$ (where the underlined atom is the selected atom). The depth of this derivation is 3.

Definition III.4 : *Let* t(G) *be an* SLD-*tree of root* G. *Let us define, for any integer* n, *the* **n-depth subtree** $t_n(G)$ *of* t *as the maximal subtree of* t *of which all derivations have a depth* $\leq$ n. *Let us denote* $Res(t_n(G)) = \{ (L; \theta)$ *where* $(L; \theta)$ *is not a failed leaf of* $t_n(G)\}$

Remark : $t_n(G)$ is a finite tree.

Example : Let $t(\leftarrow A(x))$ be the SLD-tree of root $\leftarrow A(x)$ computed by $\mathfrak{B}$.

$t_2(\leftarrow A(x)) =$



$Res(t_2(\leftarrow A(x))) = \{(B(a);x=a), (\emptyset; x=b)\}$

Proposition III.1 : *Let* G *be a goal, let* C *be the clause* $X(x_1, ..., x_n) \leftarrow G$ *where* X *is a predicate symbol which does not occur in* G *and where* $x_1, ..., x_n$ *are the variables occurring in* G. *Denote by* t *the* SLD-*tree of root* G *computed by* $\mathfrak{B}$ *using a definite program* P. *Therefore for any integer* $n \geq 1$, *we have :*

$$C_n = \{X\theta \leftarrow L ; (L ; \theta) \in Res(t_{n-1}(G))\}$$

where C_n *is defined in corollary* II.3.

Proof : By induction on n. It is obvious if n = 1. Suppose that this relation holds for some $n \geq 1$.
$C_{n+1} = Unf(C_n, P) = \bigcup_{C' \in C_n} Unf(C', P) = \bigcup_{(L;\theta) \in Res(t_{n-1}(G))} Unf(X\theta \leftarrow L, P)$.

By definition of the unfolding and of the rule $\mathfrak{B}$, we have :
$Unf(X\theta \leftarrow L, P) = \{X\theta\theta' \leftarrow L' \text{ where } (L' ; \theta') \in Res(t_1(\leftarrow L))\}$ and
$Res(t_n(G)) = \{(L' ; \theta\theta') \text{ where } (L ; \theta) \in Res(t_{n-1}(G)) \text{ and } (L' ; \theta') \in Res(t_1(\leftarrow L))\}$.
Therefore, $C_{n+1} = \{X\theta\theta' \leftarrow L' ; (L ; \theta) \in Res(t_{n-1}(G)) \text{ and } (L' ; \theta') \in Res(t_1(\leftarrow L))\}$,
that is $C_{n+1} = \{X\theta \leftarrow L ; (L ; \theta) \in Res(t_n(G))\}$. □

Example : We verify, in the last example, that $C_3 = \{X \leftarrow, X \leftarrow B(a)\}$.

Proposition III.2 : *Let* G *be a goal, let* C *be the clause* $X(x_1, ..., x_n) \leftarrow G$ *where* X *is a predicate symbol which does not occur in* G *and where* $x_1, ..., x_n$ *are the variables occurring in* G.
$\leftarrow G$ *finitely fails under* $\mathfrak{B}$ *(or any fair computation rule)* [Las 84]
iff $\exists\, n \geq 1$ *such that* $C_n = \emptyset$
iff $\exists\, n \geq 1$ *such that* $Unf(C, P^{n-1}) = \emptyset$.

$\leftarrow G$ *succeeds with the computed answer* θ *under* $\mathfrak{B}$ *(or any computation rule)* [Apt 82] *iff* $\exists\, n \geq 1$ *such that* $X\theta \leftarrow \in C_n$
iff $\exists\, n \geq 1$ *such that* $X\theta \leftarrow \in Unf(C, P^{n-1})$.

Proof : Let t(G) be the SLD-tree of root G computed by $\mathfrak{B}$. Proposition III.2 is an immediate consequence of proposition III.1 and corollary II.3 since :
$\leftarrow G$ finitely fails under $\mathfrak{B}$ iff $\exists$ n such that $Res(t_n(G)) = \emptyset$ and
$\leftarrow G$ succeeds with the computed answer θ under $\mathfrak{B}$ iff $\exists$ n such that $(\emptyset;\theta) \in Res(t_n(G))$. □

Definition III.5 : *Definite programs* P *and* Q *have the same* **procedural semantics** *if the same goals finitely fail for* P *and* Q, *if the same goals succeed for* P *and* Q *and if they succeed with the same computed answers.*

Theorem III.1 : *For every integer* $n \geq 1$, *and for every definite program* P, P *and* P^n *have the same procedural semantics.*

Proof : It is an immediate consequence of proposition III.2 and corollary II.2. □

Let us define now the infinite unfolding P^ω of the definite program P.
As in [Lev 88, Kan 89], P^ω contains the unit clauses of each P^n. But, in order to take the finite failures into account, we also add to P^ω some non-unit clauses. More formally, we define :

Definition III.6 : $P^\omega = U^\omega \cup NU^\omega$ *where* $U^\omega = \{ A \leftarrow ; \exists n \in \omega$ *such that* $A \leftarrow$ *is a unit clause of* $P^n\}$ *and* $NU^\omega = \{ A \leftarrow \infty ; \forall n \in \omega, \exists m \geq n$, A *is the head of a clause of* P^m *with a not empty body and where* ∞ *is a new symbol*$\} \cup \{\infty \leftarrow \infty\}$.

Remark : P^ω may have an infinite number of clauses but, for these kinds of program, we can extend the notions of SLD-tree, branch of success, and finitely failed tree.

Theorem III.2 : *For any integer* $n \geq 1$, P^n *and* P^ω *have the same procedural semantics.*

Proof : It suffices to prove that P and P^ω have the same procedural semantics.
With notations of proposition III.1, we have :
$\leftarrow$ G finitely fails under $\mathfrak{B}$ (wrt P) iff $\exists n \geq 1$ such that $Unf(C, P^{n-1}) = \emptyset$
iff $Unf(C, P^\omega) = \emptyset$.
$\leftarrow$ G succeeds with the computed answer θ under $\mathfrak{B}$ (wrt P)
iff $\exists n \geq 1$ such that $X\theta \leftarrow \in Unf(C, P^{n-1})$
iff $X\theta \leftarrow \in Unf(C, P^\omega)$. □

Examples :

a) $\begin{cases} A(a) \leftarrow \\ A(s(x)) \leftarrow A(x) \\ A(b) \leftarrow A(b) \end{cases}$

$U^\omega = \{A(s^n(a)) \leftarrow ; n \geq 0\}$ and $NU^\omega = \{A(s^n(b)) \leftarrow \infty ; n \geq 0\} \cup \{\infty \leftarrow \infty\}$.

b) $\begin{cases} A(a) \leftarrow B(x) \\ B(s(x)) \leftarrow B(x) \end{cases}$ $U^\omega = \emptyset$ and $NU^\omega = \{A(a) \leftarrow \infty\} \cup \{\infty \leftarrow \infty\}$.

In fact we have the following result which is more precise :

Theorem III.3 :
$\leftarrow G = \leftarrow A_1, ..., A_p$ *finitely fails under* $\mathfrak{B}$ *(wrt* P*) iff* p*-tuple* $(B_1, ..., B_p)$ *of head of clauses of* P^ω *such that* $(A_1, ..., A_p)$ *and* $(B_1, ..., B_p)$ *are unifiable does not exist.*
$\leftarrow G = \leftarrow A_1, ..., A_p$ *succeeds with the computed answer* θ *iff there exists a* p*-tuple* $(B_1 \leftarrow, ..., B_p \leftarrow)$ *of unit clauses of* P^ω *such that* $(A_1, ..., A_p)$ *and* $(B_1, ..., B_p)$ *are unifiable with* mgu θ' *and* $(A_1, ..., A_p)\theta = (A_1, ..., A_p)\theta'$ *modulo variable renaming.*

Proof : straightforward. □

IV. Uniformly fair computation rules

We show that the equivalence between the two definitions of P^n may be seen as an equivalent property of some particular computation rules : $\mathfrak{B}$ and $\mathfrak{D}_p$ that we define now.

Definition IV.1: *Let* p *be an integer and let us denote by* $\mathfrak{D}_p$ *the computation rule which selects in the last goal* G *of a* SLD*-derivation the first atom of depth* < (k+1)p *if* kp ≤ depth(G) < (k+1)p.

Remark : The derivations computed by $\mathfrak{D}_p$ correspond to proof-trees computed using a depth-first strategy on blocks of depth p [Der 87].

Example : With the program P of the previous section,
$\leftarrow \underline{A(x)} \xrightarrow{C_1} \leftarrow \underline{B(x)},C(x) \xrightarrow{C_2} B(a), \underline{C(a)} \xrightarrow{C_4} \underline{B(a)} \xrightarrow{C_2} B(a)$ ... is a derivation computed by $\mathfrak{D}_2$.

$\leftarrow \underline{A(x)} \xrightarrow{C_1} \leftarrow \underline{B(x)},C(x) \xrightarrow{C_2} \underline{B(a)}, C(a) \xrightarrow{C_2} B(a),\underline{C(a)} \xrightarrow{C_4} B(a)$ ... is a derivation computed by $\mathfrak{D}_3$.

The corollary II.1 (identity of the two definitions of P^n) may be seen as a corollary of the next result :

Proposition IV.1 : *Let* t(G) *and* s(G) *be two* SLD*-trees of root* G *respectively computed by* $\mathfrak{B}$ *and* $\mathfrak{D}_p$. *Therefore,* $Res(t_p(G)) = Res(s_p(G))$ *(modulo variable renaming).*

This proposition is a specific case of a more general result on p-**uniformly fair computation rules** that we define now.
Roughly speaking, a uniformly fair computation rule computes SLD-derivation for which, at some stages, all atoms of the current goal have the same depth.

Definition IV.2 : *Let* $\delta : G_0,...,G_m, ...$ *be an* SLD*-derivation and let* $\delta' : G_0,...,G_m$ *be the maximal sub-derivation of* δ, *of which all atoms have a depth* < p. *The derivation* δ *is said to be* p-**uniformly fair** *if* $\delta = \delta'$ *or if all the atoms of* G_m *have a depth equal to* p **and** *(inductively) the derivation* : $G_{m+1}, G_{m+2}, ...$ *is* p*-uniformly fair.*
That is, an atom in a goal G *of* δ *cannot have a depth equal to* kp +1 *(where* k *is any integer) if* G *contains another atom of depth* < kp .

Example : An SLD-refutation (i.e. a finite SLD derivation whose last goal is ∅) of depth p is p-uniformly fair.

Definition IV.3 : *An* SLD*-tree* t *is* p-**uniformly fair** *if all derivations of* t *are* p-**uniformly fair.** *A computation rule is* p-**uniformly fair** *if all the* SLD*-trees that it computes are* p*-uniformly fair.*

Remark : A p-uniformly fair computation rule is fair.

Proposition IV.2 : *For any integer* $p \geq 1$, $\mathfrak{B}$ *and* $\mathfrak{D}_p$ *are* p-*uniformly fair computation rules.*

Proof : Straightforward from the definitions. □

Theorem IV.1 : *Let* t *and* s *be two* p-*uniformly fair* SLD-*trees of root* G. *For every integer* $k \geq 1$, $Res(t_{kp}(G)) = Res(s_{kp}(G))$ *(modulo variable renaming)*

Proof : Omitted here. It is a generalization of the proof of the "switching lemma" [Llo 87]. □

Example : For the program P above, the goal G : ← A(x), let t and s be the SLD-trees respectively computed by $\mathfrak{D}_2$ and $\mathfrak{B}$.
$Res(t_2(\leftarrow A(x))) = Res(s_2(\leftarrow A(x))) = \{(B(a);x=a), (\emptyset; x=b)\}$

Remarks :
1- This result shows that two p-uniformly fair computation rules are equivalent in a strong sense (i.e. intermediate stages of the computations are identical, not only the final results).
2- It may be seen as a generalization of the theorems of independence of the computation rules (for success) and fair computation rules (for finite failures).

V. Unfolding and transformations

We can now define a property on "immediate consequence operators" which will guarantee that their "iterate sets" describe accurately the procedural properties of the logic program.

Definition V.1 : *Let* L *be a first order language and* h *be the Herbrand base for* L. *Suppose that for any definite program* P *with associated language based on* L, *we can define a mapping* $T_P : 2^h \dashrightarrow 2^h$. *We say that this operator is* **compatible with the unfolding rule** *if for any definite programs* P, Q :

$$T_{Unf(P,Q)}(\emptyset) = T_P(T_Q(\emptyset))$$

Theorem V.1 : *If a monotonic operator* T *is compatible with the unfolding rule then for any definite program* P *and for any integer* $n \geq 1$:

$$T_{P^n}(\emptyset) = T_P \uparrow n$$

and, provided that T *is adaptated to infinite programs,*

$$T_{P^\omega}(\emptyset) = T_P \uparrow \omega$$

Proof : By induction on n. It is obvious if n=1. Suppose that the relation holds for some $n \geq 1$.
$T_{P^{n+1}}(\emptyset) = T_{Unf(P,\, P^n)}(\emptyset) = T_P(T_{P^n}(\emptyset)) = T_P(T_P \uparrow n) = T_P \uparrow (n+1)$. □

Definition V.2 : *Let* P *be a definite program and* I *be an Herbrand interpretation.*
$T_P(I) = \{A ;\ A \leftarrow A_1, ..., A_n$ *is a ground instance of a clause of* P *and* $\{A_1, ..., A_n\} \subset I\}$ [Apt 82].
T^δ *is defined in* [Del 88, Den 90]. *We give here a slightly different definition. If* I, J *are Herbrand interpretations,* $T^\delta_P(I,J) = (T_P(I), h\backslash T_P(hJ))$. *Then we have,* $T^\delta_P\uparrow n = (T_P\uparrow n, h\backslash T_P\downarrow n)$.

According to Theorem III.3, $T_{P\omega}(\varnothing) = SS$ and $T^\delta_{P\omega}(\varnothing) = (SS, FF)$ where SS (resp. FF) is the Success Set (resp. Finite Failure set) of P [Emd 76].

Definition V.3 : *Let* P *be a definite program on language* L. *The Herbrand universe* U_V *is the set of terms of* L, *modulo variable renaming. The Herbrand base* B_V *is the set of all atomic formulas* $p(t_1, ..., t_n)$ *where* p *is a predicate symbol of* L *and* $t_1, ..., t_n \in U_V$. *Let* $I \subset B_V$.
$T^S_P(I) = \{A\theta \in B_V ; \exists\ A \leftarrow B_1, ..., B_m \in P, \exists\ C_1, ..., C_m \in I, \theta = mgu((B_1, ..., B_m),(C_1, ..., C_m))\}$.
$T^C_P(I) = \{A\theta \in B_V ; \exists\ A \leftarrow B_1, ..., B_m \in P, B_1\theta, ..., B_m\theta \in I\}$ [Fal 89].

According to Theorem III.3, $T^S_{P\omega}(\varnothing) = U(P^\omega)$ and $T^C_{P\omega}(\varnothing) = \{A \in B_V ; \leftarrow A$ succeeds with the identity computed answer$\}$.

Theorem V.2 : *The immediate consequence operators* T, T^δ, T^C and T^S *are compatible with the unfolding rule.*

Proof : We prove this theorem only for the operator T^S. Let $I \subset B_V$.
$T^S_P(I) = \{A\theta \in B_V ; \exists\ A \leftarrow B_1, ..., B_m \in P, \exists\ C_1, ..., C_m \in I, \theta = mgu((B_1, ..., B_m),(C_1, ..., C_m))\}$.
We can notice that :
- $T^S_P(\varnothing) = U(P)$ (where U(P) denotes the heads of unit clauses of P).
- For any definite program Q, $T^S_{Unf(P,Q)}(\varnothing)$ depends only on P and U(Q).
- $T^S_P(I) = T^S_{Unf(P,Q)}(\varnothing)$ where $Q = \{A \leftarrow ; A \in I\}$.
- For any definite program Q, $T^S_{Unf(P,Q)}(\varnothing) = T^S_P(U(Q)) = T^S_P(T^S_Q(\varnothing))$.

Hence, T^S is compatible with the unfolding rule.
We also have : $T^S_{P\omega}(\varnothing) = U(P^\omega) = \bigcup_{n\in\omega} U(P^n) = \bigcup_{n\in\omega} T^S_{Pn}(\varnothing) = T^S\uparrow\omega$. □

Corollary V.1 : *Let* P *be a definite program.*
i) $SS = T_P\uparrow\omega =$ *least fixpoint of* T_P *(since* T_P *is continuous* [Apt 82]).
ii) $(SS, FF) = T^\delta_P\uparrow\omega$ [Del 88].
iii) $T^C_P\uparrow\omega = \{A \in B_V ; \leftarrow A$ *succeeds with the identity computed answer*$\}$ = *least fixpoint of* T^C_P *(since* T^C_P *is continuous* [Fal 89]).
iv) $\leftarrow A$ *succeeds with the computed answer* θ *iff there exists* $B \in T^S\uparrow\omega =$ *least fixpoint of* T^S_P *(since* T^S_P *is continuous* [Fal 89]) *such that* $\theta' = mgu(A,B)$ *and* $A\theta = A\theta'$ *modulo variable renaming .*

Proof : Straightforward. □

VI. Conclusion

The unfolding semantics appears as intermediary between procedural and fixpoint semantics. The fundamental property of associativity of the unfolding ensures its adequacy with the procedural semantics while the notion of "operator compatible with the unfolding rule" provides the connection with the fixpoint semantics. The ability to make "computations" on logic programs gives a quasi-algebraic characterization of the relationship between these two semantics which often allows proof to be more simpler than usual.

The theorem about the strong equivalence between p-uniformly fair computation rules gives an other explanation of the equivalence of the definitions of the iterated programs P^n. But it has a wider scope and it may be seen as a generalization of the usual results on the independence of computation rules (for success) and fair computation rules (for finite failures).

References

[Apt 82] K. APT and M. VAN EMDEN, *Contribution to the Theory of Logic Programming*, Journal of the Association for Computing Machinery, 29.3, 1982, p. 841-862.

[Cla 78] K. L. CLARK, *Negation as Failure. Logic and Data Bases*, H. GALLAIRE and J. MINKER ed., Plenum Press, New-York, 1978, p. 293-324.

[Del 88] J.-P. DELAHAYE, *Sémantique Logique et Dénotationnelle des Interprèteurs Prolog*, Informatique Théorique et Applications, vol 22, n°1, 1988, p. 3 - 42.

[Den 89] F. DENIS, *Approximations de la sémantique opérationnelle d'un interprèteur Prolog Standard par des sémantiques axiomatiques*, Rapport de recherche, septembre 1989, Laboratoire d'Informatique Fondamentale de Lille, Bât. M3, Université des Sciences et Techniques de Lille, 59655 Villeneuve d'Ascq, CEDEX.

[Den 90] F. DENIS, *Contribution à l'étude des sémantiques axiomatiques de Prolog*, Nouvelle thèse, LIFL, Bât. M3, Université des Sciences et Techniques de Lille, 59655 Villeneuve d'Ascq, CEDEX.

[Der 87] P. DERANSART, G. FERRAND, *Programmation en Logique avec Négation : Présentation Formelle*, INRIA-Université d'Orléans, Faculté des Sciences, BP 6759, 45067 ORLEANS Cédex 2.

[Emd 76] M. H. VAN EMDEN et R. A. KOWALSKI, *The Semantics of Predicate Logic as a Programming Language*, Journal of the Association for Computing Machinery, vol. 23-4, 1976, p 733-742.

[Fal 89] M. FALASCHI, G.LEVI, C. PALAMIDESSI, *Declarative Modeling of the Operational Behavior of Logic Languages*, Theorical Computer Science 69 (1989) 289-318, North-Holland.

[Kan 87] T. KANAMORI, K. HORIUCHI, *Construction of Logic programs Based on Generalized Unfold/Fold Rules*, Proc. 4th Int. Conf. on Logic Programming, The MIT Press, Series in Logic Programming (1988).

[Kan 89] K. KANCHANASUT, P. STUCKEY, *Eliminating negation from normal logic programs*, Department of Computer Science, University of Melbourne, Parkville 3052, Australia.

[Las 84] J.L. LASSEZ, M.J. MAHER, *Closures and Fairness in Semantics of Programming Logic*, T.C.S., 29, 1984, p. 167-184.

[Lev 88] G.LEVI, P. MANCARELLA, *The Unfolding Semantics of Logic Programs*, Technical report - 13/88, Università degli Studi di Pisa, Dipartimento di Informatica.

[Llo 84] J.W. LLOYD, *Foundations of Logic Programming*, Springer-Verlag (1984).

[Llo 87] J.W. LLOYD, J.C. SHEPHERDSON, *Partial evaluation in Logic Programming*. Technical report CS-87-09, Dept. of Computer Science, University of Bristol (1987).

[Sek 89] H. SEKI, *Unfold/Fold Transformation of Stratified Programs*, Extended Abstract, Institute for New Generation Computer Technology, 1-4-28, Mita, Minato-ku, Tokyo 108, JAPAN.

[Tam 84] H. TAMAKI, T. SATO, *Unfold/Fold Transformation of Logic Programs*, Proc. 2th Int. Conf. on Logic Programming (1984).