ERROR DIAGNOSIS IN LOGIC PROGRAMMING, 
AN ADAPTATION OF E. Y. SHAPIRO'S METHOD*

GÉRARD FERRAND

We study Shapiro's method of bug diagnosis in the theoretical framework of Horn clause logic programming. Within the framework of Clark's semantics (Herbrand's universe with variables, which is more general than the most usual semantics without variables) we extend the scope of fixpoint and declarative semantics of logic programming.

1. INTRODUCTION

The starting point of this paper is the "algorithmic program debugging" method of Shapiro [7, 8]. The idea is that of a system which can detect some semantics errors in a program. Clearly, the input of such a system must include at least the erroneous program (which has been actually written by the programmer) and some information on the intended semantics of the program (which is usually given by the programmer, as answers to some questions asked by the system). With this method, in order that the system may give as result some error of the erroneous program, it must also have as input another piece of information, which we call the error symptom. Intuitively, this symptom is something which goes wrong during the execution of the erroneous program.

Shapiro has defined appropriate notions for the errors of the programs for a class of languages, and the notion of an oracle which can be asked questions about the semantics of a program. He has described algorithms with which one can diagnose the program errors. He has implemented these algorithms as PROLOG programs (Edinburgh PROLOG-10) which can diagnose some errors of programs in

*This work has been supported by the "GRECO de programmation" (ATTRISEM project).

THE JOURNAL OF LOGIC PROGRAMMING

©Elsevier Science Publishing Co., Inc., 1987
52 Vanderbilt Ave., New York, NY 10017

0743-1066/87/$3.50
PROLOG-10. PROLOG is chosen by Shapiro as the most appropriate tool for error diagnosis, but the justifications are given in a framework of general programming.

In this paper we study the problem in the theoretical framework of logic programming (more precisely, Horn clause programming [4]). In this framework we can take advantage of a fixpoint semantics [1-3] that is rather simple to use (sets of atomic formulas). Moreover, because of its logical essence (notion of model), this framework is a privileged one to express clearly and precisely notions about errors. The idea is to set in the framework of this semantics not only the erroneous programs, but also the “error diagnoser program”, and to define what can be diagnosed, in order to adapt and to extend in this context Shapiro’s definitions and justifications.

We distinguish strictly between logic programming and general or even PROLOG programming. So a preliminary comment may be made about the comparison of this work with Shapiro’s one: in a way, some of our results are the same as Shapiro’s, but the justifications cannot be the same. For example, our semantics is declarative, but nevertheless we can describe precisely the link between an error symptom and an error which can be seen as the “cause” of the symptom and which must be diagnosed.

Moreover, apart from the differences of context, our definitions are extensions of Shapiro’s because of the use of variables in our semantics. The classical semantics of logic programs involves only terms without variables (Herbrand’s universe), while the “answers” given by an interpreter are substitutions, involving terms possibly with variables. Because of the lack of variables, some information potentially contained in a logic program is lost. For example, let us suppose that there are only two function symbols, i.e. the two constants 1 and 2, and let us consider a goal \( R(x) \). This semantics cannot formalize the difference between the two following cases:

- the interpreter outputs two substitutions \( \theta_1(x) = 1 \) and \( \theta_2(x) = 2 \);
- the interpreter outputs the substitution \( \theta(x) = x \).

Following Clark [2], we include terms with variables in the semantics of logic programs. But this extension limits the ability of a logic program to diagnose an error in a logic program; this limitation is an incompleteness result, which appears only in the framework with variables.

In our framework, we study two “dual” problems defined by Shapiro: *incorrectness* diagnosis and *insufficiency* diagnosis. Here are two simple examples: In these two examples the intended semantics for the relation \( \text{REV} \) is to reverse a list, and for \( \text{APP} \) it is to concatenate two lists. We are using the classical notation with “.” and \( \text{NIL} \).

**Example 1.** Let \( P \) be the following logic program:

\[
\begin{align*}
\text{REV}(\text{NIL}, \text{NIL}) & \leftarrow \\
\text{REV}(x \cdot y, z) & \leftarrow \text{REV}(y, t) \text{APP}(t, x \cdot \text{NIL}, z) \\
\text{APP}(\text{NIL}, x, x) & \leftarrow \\
\text{APP}(x \cdot y, z, t) & \leftarrow \text{APP}(y, z, t)
\end{align*}
\]
In the actual semantics of \( P \) there is the "erroneous" atom \( \text{REV}(u \cdot v \cdot \text{NIL}, u \cdot \text{NIL}) \), which we call an \textit{incorrectness symptom}. From the following tree, in which the incorrectness symptom is the root, we can clearly see why this atom is in the actual semantics of \( P \):

\[
\begin{align*}
\text{REV}(u \cdot v \cdot \text{NIL}, u \cdot \text{NIL}) \\
\text{REV}(v \cdot \text{NIL}, v \cdot \text{NIL}) & \quad \text{APP}(v \cdot \text{NIL}, u \cdot \text{NIL}, u \cdot \text{NIL}) \\
\text{REV}(\text{NIL}, \text{NIL}) & \quad \text{APP}(\text{NIL}, v \cdot \text{NIL}, v \cdot \text{NIL}) & \quad \text{APP}(\text{NIL}, u \cdot \text{NIL}, u \cdot \text{NIL}) \\
\end{align*}
\]

The "cause" of this symptom is the cause

\[
\text{APP}(v \cdot \text{NIL}, u \cdot \text{NIL}, u \cdot \text{NIL}) \leftarrow \text{APP}(\text{NIL}, u \cdot \text{NIL}, u \cdot \text{NIL})
\]

which can be seen on the right hand branch of the tree. We call it \textit{incorrectness}. It is an instance of the last clause of \( P \). In an incorrectness, the conclusion is "erroneous" while the premises are not.

\textbf{Example 2.} Let now \( P \) be the following logic program:

\[
\begin{align*}
\text{REV}(\text{NIL}, \text{NIL}) & \leftarrow \\
\text{REV}(x \cdot y, z) & \leftarrow \text{REV}(y, t), \text{APP}(t, x \cdot \text{NIL}, z) \\
\text{APP}(x \cdot y, z, x \cdot t) & \leftarrow \text{APP}(y, z, t)
\end{align*}
\]

Now in the actual semantics of \( P \) there is no atom of the form \( \text{REV}(u \cdot v \cdot \text{NIL}, w) \); however, the atom \( \text{REV}(u \cdot v \cdot \text{NIL}, v \cdot u \cdot \text{NIL}) \) should have been there. We call this last atom the \textit{insufficiency symptom}. From the following tree, in which the insufficiency symptom is the root, we can clearly see why this atom is not in the actual semantics of \( P \) (actually it is a little bit more difficult to verify the existence of an insufficiency symptom):

\[
\begin{align*}
\text{REV}(u \cdot v \cdot \text{NIL}, v \cdot u \cdot \text{NIL}) \\
\text{REV}(v \cdot \text{NIL}, v \cdot \text{NIL}) & \quad \text{APP}(v \cdot \text{NIL}, u \cdot \text{NIL}, u \cdot \text{NIL}) \\
\text{APP}(\text{NIL}, v \cdot \text{NIL}, v \cdot \text{NIL}) \\
\end{align*}
\]

The "cause" of this symptom is an atom, \( \text{APP}(\text{NIL}, v \cdot \text{NIL}, v \cdot \text{NIL}) \), which is a leaf of the tree. We call it \textit{insufficiency}. It is an atom which belongs to the intended semantics, but it is not a conclusion of an instance of a clause whose premises belong to the intended semantics.

In Example 1, a logic program "incorrectness diagnoser", applied to \( P \), will have in its semantics an atom \textit{incorrectness}(\( t_1, t_2 \)) which associates the symptom \( t_1 \) to the incorrectness \( t_2 \). Now \( t_1 \) is the atom \( \text{REV}(u \cdot v \cdot \text{NIL}, u \cdot \text{NIL}) \), but at this "language level" it must be seen as a term. \( t_2 \) must also be a term which represents the incorrectness. In the same way, in Example 2, a logic program "insufficiency diagnoser", applied to \( P \), will have in its semantics an atom \textit{insufficiency}(\( t_1, t_2 \)) which associates the symptom \( t_1 = \text{REV}(u \cdot v \cdot \text{NIL}, v \cdot u \cdot \text{NIL}) \) to the insufficiency \( t_2 = \text{APP}(\text{NIL}, v \cdot \text{NIL}, v \cdot \text{NIL}) \).
To set the two “language levels” (that of the erroneous program and that of the error diagnoser program) in our framework we must use a “representation” method. But above all we must extend the notion of semantics of a logic program, because the semantics of a fixed “error diagnoser” program must be relative to each “erroneous” program and to each intended semantics. So we will define the denotation of a logic program; this denotation may be relative; we will also define the intended denotation.

This paper is organized as follows: In Section 2 we justify our choice for the semantics of logic programs. For the clarity of the discussion we use a unified framework in which we can recall both the point of view of Apt, Van Emden, and Kowalski [1,3] and that of Clark [2]. In Section 3 we define the errors and the symptoms. In Section 4 we set up some technical preliminaries. In Sections 5 and 6, we study respectively the incorrectness and the insufficiency diagnosis. In Section 7, as a conclusion, we describe an application to PROLOG.

2. DENOTATION OF A LOGIC PROGRAM

2.1. Syntax

Ours is the usual syntax of logic programming. But because the functions of term and of atomic formula will be relative to a “level”, we give here some definitions and notation which we are going to use.

In what follows, there is a fixed denumerable infinite set \( V \) of variables denoted \( x, y, z, \ldots, u, v, \ldots \).

A function symbol system \( \Phi \) is defined by giving a set of symbols, each one having a fixed arity. The function symbols with arity 0 are the constants of \( \Phi \). The set \( \text{TERM}(\Phi) \) of the terms built with \( V \) and \( \Phi \) (\( V \) being fixed and implicit) is defined as usual.

A relation symbol system \( \Pi \) is defined by giving a set of symbols, each one having a fixed arity. The set \( \text{ATOM}(\Pi, \Phi) \) of the atoms (atomic formulas) built with \( V, \Phi, \) and \( \Pi \) is defined as usual.

In what follows, clause will mean the usual notion of definite clause: a clause is defined by giving an atom \( A \) and a finite set of atoms \( E \). We denote it by \( A \leftarrow E \), and by \( A \leftarrow B_1, \ldots, B_k \) if \( E = \{ B_1, \ldots, B_k \} \), and by \( A \leftarrow \) if \( E = \emptyset \).

A substitution is a map \( \theta: V \rightarrow \text{TERM}(\Phi) \). If \( e \) is a term, atom, set of atoms, clause, \ldots, then the instance \( e\theta \) of \( e \) by \( \theta \) is defined as usual.

Closed means without variables (so \( e\emptyset = e \)). \( \text{TERMC}(\Phi) \) and \( \text{ATOMC}(\Pi, \Phi) \) are respectively the set of all the closed terms and the set of all the closed atoms.

A logic program is a set of clauses.

2.2. Motivation

In order to express and to prove precise results we have to choose a definition for the semantics of a logic program. In our declarative framework there is already the usual “minimal model” [1,3], which is built out of the closed terms. But this model does not capture very well the effect of the substitutions on an interpretation of a
logic program. That is so because this model does not correspond exactly with the
notion of logical consequence. This question is studied technically and discussed in
the next section, but here is our intuitive motivation:

Following Clark [2], we say that a logic program $P$ is used for answering
questions: a question $Q$ is a set of atoms (possibly with variables). An answer of $P$
to the question $Q$ is a substitution $\theta$ (possibly with variables, $\theta(x)$) such that each
atom of the set $Q\theta$ is a logical consequence of $P$. Clark proves an equivalence
between this semantics and the procedural one based on resolution.

We have two aims: firstly, our formal definition of the semantics must be faithful
to the notion of logical consequence; secondly it must be a formal object as simple
as the usual minimal model, i.e. simply a set of atoms.

The minimal model in Clark's sense [2] is just what we need: it is (identified with)
a set of atoms (possibly with variables), and Clark proves that an atom $A$ is logical
consequence of $P$ if and only if $A$ is valid in this model (in the sense of the formal
logic, the variables being implicitly universally quantified). But we add (see the next
section) that it is also equivalent to the statement: $A$ is simply a member of this set
of atoms.

Another motivation for this semantics "with variables" is given by the following
remark: Suppose the question $Q$ has only one atom $R(x)$. Then we can interpret it
as the question:

"Are there objects satisfying $R(x)$, and if there are such objects, which ones?"

(formally, the first part of the question asks if the logical formula "there exists $x$
such that $R(x)$" is a logical consequence of $P$. For the second part, the only way to
denote these objects is by terms.

Fortunately, if the logical formula "there exists $x$ such that $R(x)$" is a logical
consequence of $P$, then there is a term $t$ in $\text{TERM}(\Phi)$ such that $R(t)$ is logical
consequence of $P$, i.e., there is an answer, $\theta(x) = t$, to the question. So logic
programming works! But it is not a trivial property of "exists": it is true only
because the clauses in $P$ are definite.

Suppose $P$ had only the clause (not definite) $R(a), R(b) \leftarrow$ (or equivalently the
logical formula "$R(a)$ or $R(b)$"); then "there exists $x$ such that $R(x)$" would be a
logical consequence of $P$. But there is no such term $t$, unless we add a new symbol.

Now, if $P$ is a logic program, we have a term $\theta(x) = t$ which is in $\text{TERM}(\Phi)$,
without adding any constant. Thus if there is no constant in $\Phi$, then $t$ must contain
variables, and if there is no function symbols, then $t$ must be a variable. The
minimal model in Clark's sense formalizes this in a natural way (compare with the
definition of the usual minimal model: to define Herbrand's universe, if there is no
constant, we add one).

The reason why terms without variables seem to be more attractive is that they
allow analogies with conventional programming: the objects involved in a problem
are represented by data structures; in logic programming the data structures are
terms, and in general the variables take data structures as values. However, in logic
programming the representation process is much more general, since it is the
problem itself (objects and relations between them) which is represented by means
of a logical language and which is processed using the notion of logical consequence.

The essential point in this discussion is that the logical function of a variable is not the representation of terms, it is the representation of individuals of the domain of any model (such an individual is not necessarily denotable by a term).

2.3. Absolute Denotation

An atom $A$ is logical consequence of $P$ if $A$ is valid in each model of $P$. A model of $P$ is an interpretation for the language defined by $\Phi$ and $\Pi$ in which all the clauses of $P$ are valid. It is the notion of interpretation which is usual in formal logic, but we call it an $L$-interpreretation in order to distinguish it from the notion of interpretation used in logic programming [1–3]. An interpretation is a set of atoms. The interpretations can be identified with some particular $L$-interpretations; the question here is to see if the interpretations are sufficient for defining the notion of logical consequence. For the clarity of the discussion we recall a few points.

Let $H$ be a fixed Herbrand universe. [Clark takes $H = \text{TERM}(\Phi)$; usually in the literature it is taken $H = \text{TERM}(\Phi)$. However, in general we could take $H$ to be a subset of $\text{TERM}(\Phi)$ satisfying an adequate closure condition.] We denote by $\text{BASE}(H)$ the set of all the atoms $R(t_1, \ldots, t_m)$ where $t_i \in H$. An interpretation is any subset of $\text{BASE}(H)$.

Let $I$ be an interpretation. There is an $L$-interpretation $\bar{I}$ which can be identified with $I$: we say that $\bar{I}$ is “termal on $H$”, because its domain is $H$, on which the function associated to each function symbol is the well-known canonical function. The relation on $H$ associated to a relation symbol $R$ is the set of the $m$-tuples $(t_1, \ldots, t_m)$ such that $R(t_1, \ldots, t_m) \in I$ [note that if $H = \text{TERM}(\Phi)$, this is not equivalent to $R(t_1, \ldots, t_m)$ valid in $\bar{I}$, because of the variables in the $t_i$].

In such an $L$-interpretation $\bar{I}$ which is termal, an assignment (which gives a value in the domain to each variable) is exactly the same operation as a substitution (with range in $H$). Then an atom $A$ is true in $\bar{I}$ for such an assignment $\theta$ if and only if the instance $A\theta$ of $A$ by the substitution $\theta$ is in the set $I$.

The usual logical notions are transposed by this identification of $M$ with $I$: so that for any atom $A$, $A$ is valid in $I$ if and only if all its instances $A\theta$ are in $I$, and a clause $A \leftarrow E$ is valid in $I$ if and only if $E\theta \subseteq I$ implies $A\theta \in I$. However, it must be specified what these $\theta$ are: they are any substitution if $H = \text{TERM}(\Phi)$, and only the closed ones if $H = \text{TERM}(\Phi)$.

$I$ is a model of $P$ if all the clauses of $P$ are valid in $I$. The intersection of all the models is the least model of $P$, and it is easy to see that $A$ is valid in this least model if and only if it is valid in each model of $P$.

Now these models correspond only to some $L$-interpretations, which are termal: some classic results show that $A$ is valid in each model of $P$ if and only if $A$ is logical consequence of $P$, but under the following conditions:

- if $H = \text{TERM}(\Phi)$, then $A$ must be assumed to be closed [1, 3];
- if $H = \text{TERM}(\Phi)$, then $A$ may be any atom [2].

It is easy to see that if $H = \text{TERM}(\Phi)$, then $A$ is valid in each model of $P$ if and only if it is valid in all the $L$-models of $P$ which are finitely generated, in the sense
that each element of the domain is the value of a closed term. Note that one should not restrict the $L$-interpretations only to those which are finitely generated, for if one did, the semantics could not explain, for example, why, if $P$ has only the two clauses $R(1) \leftarrow$ and $R(2) \leftarrow$, and if the question is $R(x)$, an interpreter gives only two answers $\theta_1(x) = 1$ and $\theta_2(x) = 2$ in place of a more general $\theta(x) = x$.

So we choose $H = \text{TERM}(\Phi)$, we call the least model of $P$ the absolute denotation of $P$ (absolute because of another definition in the next section) and denote it by $D(P)$. It is the minimal model in Clark's sense [2], and we know that for any atom $A$, $A$ is a logical consequence of $P$ if and only if $A$ is valid in $D(P)$.

If $I$ is any model of $P$ and $A$ any atom, the equivalence between $A$ valid in $I$ and $A \in I$ does not hold in general because of the variables (only one implication is trivially true, with the identity substitution). But the equivalence holds in the case $I = D(P)$.

In order to prove this quickly, let $C(P)$ be the set of atoms which are logical consequences of $P$. We know that $A \in C(P)$ if and only if $A$ is valid in $D(P)$, and we have to prove that $C(P) = D(P)$. It is easy to verify that, for any clause $A \leftarrow E$ in $P$, $E \theta \subseteq C(P)$ implies $A \theta \in C(P)$, i.e., $A \leftarrow E$ is valid in $C(P)$. So $C(P)$ is a model of $P$ and thus $D(P) \subseteq C(P)$. Conversely, $A \in C(P)$ implies $A$ valid in $D(P)$ and therefore $A \in D(P)$. So $C(P) \subseteq D(P)$ and finally $C(P) = D(P)$.

If things seem somehow complicated, it is because the same formal object $D(P)$ plays two roles at once, with different logical natures: the role of a model of $P$ and the role of a theory generated by $P$ (only the atomic theorems). To sum up, we have the

**Theorem 1.** For any atom $A$, the following three conditions are equivalent:

1. $A$ is valid in $D(P)$.
2. $A$ is an element of $D(P)$.
3. $A$ is a logical consequence of $P$.

### 2.4. Relative Denotation

The first motivation of the definition introduced in this section is the fact that the semantics of an "error diagnoser" program must be relative to the erroneous program and to its intended semantics. But more generally this definition is the natural way to extend the previous semantics to programs written in actual practice with "evaluable" predicates or in some cases with read procedures (to some extent the case is similar to that of relative recursiveness and oracle machines [6]).

If $E$ is any set of atoms, we define the denotation of $P$ relative to $E$, denoted $D(P|E)$, as the absolute denotation of the program $P$ to which we add all the clauses $A \leftarrow$ where $A \in E$. Obviously, $D(P|E) = D(P)$ if and only if $E \subseteq D(P)$.

### 3. Correctness and Sufficiency

There are several different notions about correctness in papers about logic programming. Some do not concern a logic program as such, but rather a program given with a relation [2]. We begin with some definitions which seem quite natural in our
declarative framework, in order to compare them with the definitions which are introduced in Shapiro's method.

Each definition must be a relation which compares the actual denotation $D(P)$ with an ideal denotation $I$. The idea of partial correctness is that each answer given by $P$ agrees with $I$. The idea of completeness is the converse: $P$ can give any answer which agrees with $I$.

This interpretation $I$ is not arbitrary (arbitrary interpretations are only auxiliaries for defining the notion of logical consequence); it must be such that it can be a denotation of a logic program. This justifies the following definition, which would be needless if we were using the usual minimal model without variables.

**Definition 1.** An interpretation $I$ is an intended denotation if it is closed by substitution, that is, each instance of an atom in $I$ is an atom in $I$ too.

**Definition 2.** Let $P$ be a logic program and $I$ an intended denotation.

- $P$ is partially correct in $I$ if $D(P) \subseteq I$;
- $P$ is complete for $I$ if $I \subseteq D(P)$;
- $P$ is totally correct in $I$ if $D(P) = I$.

Now the method for error diagnosis (Shapiro [7,8] is based on notions which are different from the previous ones. We transpose them into our theoretical framework.

Let $T_P$ be the well-known map (Clark [2]) which associates to any interpretation $I$ the interpretation $J$ defined as follows: An atom $A$ is in $J$ if and only if there exist a clause $B \leftarrow E$ in $P$ and a substitution $\theta$ such that $A = B\theta$ and $E\theta \subseteq I$.

**Definition 3.** $P$ is correct in $I$ if $I$ is a model of $P$, that is if $T_P(I) \subseteq I$.

$P$ is sufficient for $I$ if (dual notion) $I \subseteq T_P(I)$.

$D(P)$ is the least $I$ such that $T_P(I) \subseteq I$, so correct implies partially correct. But the converse is false: take $P = \{ A \leftarrow B \}$ and $I = \{ B \}$; then $D(P) = \emptyset \subseteq I$ and $A \in T_P(I) \setminus I$.

With these definitions, sufficiency and completeness are independent of each other: To have completeness without sufficiency take $P = \{ A \leftarrow B; B \leftarrow \}$ and $I = \{ A \}$; then $I \subseteq D(P) = \{ A, B \}$ and $A \in I - T_P(I)$. To have sufficiency without completeness take $P = \{ A \leftarrow A \}$ and $I = \{ A \}$; then $I = T_P(I)$ and $D(P) = \emptyset$.

Now we merely give a name to the errors we try to detect in programs:

**Definition 4.**

An incorrectness of $P$ in $I$ is an instance $A\theta \leftarrow E\theta$ of a clause $A \leftarrow E$ in $P$ such that $E\theta \subseteq I$ and $A\theta$ is not in $I$.

An insufficiency of $P$ for $I$ is an atom in $I - T_P(I)$.

So correct (sufficient) is equivalent to lack of incorrectness (insufficiency). An error will be an incorrectness or an insufficiency.
We give two other definitions with which we will be able to describe exactly what Shapiro's method is adapted to our declarative framework, and in particular to state the relations between different notions about errors.

**Definition 5a.** An *incorrectness symptom* of \( P \) in \( I \) is an atom in \( D(P) - I \).

From an intuitive point of view, an incorrectness symptom \( A \) arises when the program \( P \) gives an answer but the answer is not consistent with what was intended \([A \in D(P) \text{ but } A \notin I]\).

The existence of an incorrectness symptom implies the existence of an incorrectness, but the converse is false (previous counterexample). On the other hand, \( P \) is partially correct if and only if there is no incorrectness symptom.

For the dual notion, we recall that \( D(P) \) is also the least fixpoint of \( T_P \) and that in the same way (because \( T_P \) is monotonic) \( T_P \) has a greatest fixpoint, which we denote by \( \text{gfp}(T_P) \). It is also the greatest \( I \) such that \( I \subseteq T_P(I) \).

**Definition 5b.** An *insufficiency symptom* of \( P \) for \( I \) is an atom in \( I - \text{gfp}(T_P) \).

This notion comes natural from a formal point of view, but it has no immediate intuitive meaning. We shall come back to this point a little further on.

As previously, the existence of an insufficiency symptom implies the existence of an insufficiency, but the converse is false.

Globally we can compare the three conditions:

1. \( P \) totally correct in \( I \), i.e.,
   \( I = D(P) = \text{the least fixpoint} \).

2. \( P \) correct in \( I \) and sufficient for \( I \), i.e.,
   \( I = T_P(I) = \text{a fixpoint} \).

3. There is no symptom either of incorrectness or of insufficiency, i.e.,
   least fixpoint \( \subseteq I \subseteq \text{greatest fixpoint} \).

So it is clear that

(1) implies (2) implies (3),

but each converse is false. From an intuitive point of view, the method for error diagnosis starts from a symptom. Then it is possible that some errors cannot be diagnosed. The method is applicable only when \( I \) does not satisfy the relation \( D(P) \subseteq I \subseteq \text{gfp}(T_P) \).

As regards the intuitive meaning of an insufficiency symptom, it is interesting to note that the complement of the subset \( \text{gfp}(T_P) \) with respect to the set of all atoms contains the *finite failure set* of \( P \) (see [1], which we denote by \( \text{ffs}(P) \)). Roughly speaking, the idea is that starting from an atom in \( \text{ffs}(P) \) taken as a question, the program \( P \) "fails", i.e., the search ends without giving any answer. Then we can give to an atom \( A \) in \( I \cap \text{ffs}(P) \) the following intuitive meaning: there is an answer which ought to be confirmed, but \( P \) does not confirm it \([A \in I \text{ but } A \in \text{ffs}(P)]\).
Now such an atom is a particular case of an insufficiency symptom \( A \) is in \( \text{fs}(P) \) and therefore is not in \( \text{gfp}(T_P) \), so the method applies in this case. Strictly speaking, this comment goes beyond the scope of our declarative framework.

4. TECHNICAL PRELIMINARIES

4.1. Representation

In order to formalize the idea of a logic program \( P_1 \) diagnosing an error in a logic program \( P \), this error will be represented by a term in an atom of the denotation of \( P_1 \). This denotation will be relative to a set of atoms which in turn represents \( P \) and \( I \). There are two "language levels": the atoms and the clauses at the level of \( P \) must be seen as terms at the level of \( P_1 \), where some relation symbols must represent properties such as being an insufficiency, an incorrectness, a clause of \( P \), an atom valid in \( I \), etc.

More generally this idea of representation might be a formalization of some constructions using metalevel predicates in PROLOG.

We introduce three new function symbols, \( \text{EMPTY}, \text{AND}, \text{IF} \) with respective arities 0,2,2, and we define a new function symbol system \( \Phi_0 \) by gathering the symbols of \( \Phi \), those of \( \Pi \), and \( \text{EMPTY}, \text{AND}, \text{IF} \). By an obvious identification we have \( \text{TERM}(\Phi) \subseteq \text{TERM}(\Phi_0) \) and \( \text{ATOM}(\Pi, \Phi) \subseteq \text{TERM}(\Phi) \).

By convention we write \( t_1 \text{AND} t_2 \) for \( \text{AND}(t_1, t_2) \), \( t_1 \text{IF} t_2 \) for \( \text{IF}(t_1, t_2) \) and, if \( n \geq 3 \),

\[ t_1 \text{AND} t_2 \text{AND} \cdots \text{AND} t_n \text{ for } t_1 \text{AND} \{ t_2 \text{AND} \cdots \text{AND} t_n \} \).

We call the following elements of \( \text{TERM}(\Phi_0) \) conjunctions:

\( \text{EMPTY} \),

the elements of \( \text{ATOM}(\Pi, \Phi) \),

the terms \( A_1 \text{AND} \cdots \text{AND} A_n \), where \( n \geq 2 \) and \( A_i \in \text{ATOM}(\Pi, \Phi) \).

We define implications as the terms \( A \text{IF} Q \) where \( A \in \text{ATOM}(\Pi, \Phi) \) and \( Q \) is a conjunction.

We say that a conjunction represents a set of atoms defined as follows:

\( \text{EMPTY} \), \( A \), \( A_1 \text{AND} \cdots \text{AND} A_n \),

represent respectively the sets

\( \emptyset \), \( \{ A \} \), \( \{ A_1, \ldots, A_n \} \).

We say that an implication \( A \text{IF} Q \) represents the clause \( A \leftarrow E \) if \( Q \) represents \( E \); and that a set of implications represents the logic program consisting of the set of the clauses which are represented by the implications.

Let \( \text{RP} \) be any set of implications. Then we denote by \( \text{INST}(\text{RP}) \) the set of the instances of all the elements of \( \text{RP} \) by all the substitutions with range in \( \text{TERM}(\Phi) \).

So let \( P \) be the logic program which is represented by \( \text{RP} \), and \( I \subseteq \text{ATOM}(\Pi, \Phi) \). Then \( I \) is a model of \( P \) iff, for each implication \( A \text{IF} Q \in \text{INST}(\text{RP}) \), the following condition holds: if the set represented by \( Q \) is included in \( I \), then \( A \in I \).
4.2. Ramifications

The tool to describe exactly the link between error and symptom will be a forest in which the pieces of information at the nodes are atoms. We will use Pair's formalism [5], which is a handy method to express double inductions. So a forest will be formally a ramification [on the set ATOM(Π, Φ)]. We give here the terminology used.

A ramification can be seen as a finite sequence of (finite nonempty) trees: concatenation is an operation on the ramifications, denoted +, which is associative. There is an empty ramification, denoted by ε, which is the zero element of +. The root operation, denoted ρ, defines a ramification Ap where A is an atom and r is a ramification: Ap may be seen as a tree having A at the root, under which there is the sequence r of subtrees.

The ramification Aε is identified with the atom A, and each nonempty ramification r can be written r = Apr1 + r2, where A (an atom) and r1 and r2 (ramifications) are unique (ρ has priority on +). Definitions and proofs by induction are based on this property. So we can define the set of the roots of a ramification: ε has no root, and the set of the roots of Apr1 + r2 is the union of {A} with the set of the roots of r2. In a similar way we define the leaves of a ramification.

A tree is a ramification which can be written Apr. The subtrees of a ramification are trees defined as follows: ε has no subtree; the set of the subtrees of Apr1 + r2 is the union of {Apr1} with the sets of the subtrees of r1 and of r2.

The root conjunction of a ramification is defined as follows: the root conjunction of ε is EMPTY, that of Apr1 is A, and in the case where r2 is not ε, the root conjunction of Apr1 + r2 is A AND Q, where Q is the root conjunction of r2.

The root implication of a tree Apr is A IF Q, where Q is the root conjunction of r.

We say that a ramification r is conformable to a set of implication RP if the root implication of each subtree is in INST(RP).

**Proposition 1.** Let P be a logic program represented by a set of implications RP. Then each atom in D(P) is the root of a tree which is conformable to RP. Conversely, each ramification conformable to RP has all its atoms in D(P).

**Proof.** Firstly we apply induction on D(P) [using the fact that D(P) is included in every I such that Tp(I) ⊆ I]; then we use induction on the ramifications. □

5. THE PROBLEM OF NONCORRECTNESS

5.1. Meaning of Incorrectness Diagnosis by a Logic Program

The idea of a logic program "incorrectness diagnoser" (Shapiro [7, 8]) is that if we ask as a question an atom like INCORRECTNESS(A, x), where A is an incorrectness symptom, then we get as an answer θ(x) = an incorrectness. Formally, θ(x) is a term which represents an incorrectness, i.e. an implication B IF Q in our formalism.

Moreover, within our theoretical framework, we are going to be able to express (using the notion of ramification) the link between the symptom and what may be seen as its "cause", namely the incorrectness to be diagnosed.
Definition 6. Let RP be a set of implications and I an intended denotation. A tree $Bpr_1$ is said to be critical for RP in I if

- $Bpr_1$ is conformable to RP,
- all the atoms of $r_1$ are in I,
- the atom $B$ is not in I.

Let $r$ be a ramification. We say that $r$ has an implication $B \text{ IF } Q$ which is critical for RP in I if $r$ has a subtree $Bpr_1$ which is critical for RP in I and $B \text{ IF } Q$ is the root implication of $Bpr_1$.

Then, if RP represents the logic program $P$, an implication which is critical for RP in I represents a clause which is an incorrectness of $P$ in I.

Let us note that if $r$ is a ramification which itself is conformable to RP, then $r$ has an implication which is critical for RP in I if and only if $r$ has an atom which is not in I (take such an atom which is minimal in an obvious sense).

Now let I be an intended denotation, $P$ a logic program, and $A$ an incorrectness symptom of $P$ in I. Let RP be a set of implications which represents $P$. Then $A$ is the root of a tree which is conformable to RP [because $A \in D(P)$], and that tree has an implication which is critical for RP in I (because $A$ is not in I). Let $B \text{ IF } Q$ be this critical implication. It represents an incorrectness of $P$ in I, which we may see as a “cause” of the symptom $A$.

Formally, we would like the atom $\text{INCORRECTNESS}(A, B \text{ IF } Q)$ to be in the denotation of the program “incorrectness diagnoser”. This denotation must be relative to a set $S$ of atoms which depends on $P$ and I.

But all these atoms are not at the same “language level” as those of $P$ and I. Formally, the symbol $\text{INCORRECTNESS}$ will be in a new relation symbol system, just as IF and AND are in a new function symbol system.

5.2. Correctness and Completeness of the Diagnosis

Concerning any logic program that is a candidate for being an “incorrectness diagnoser”, there are two problems that arise naturally. The first one concerns the correctness of the diagnosis: whenever this program gives us an answer, is it always an incorrectness? The second concerns the completeness of the diagnosis: can all incorrectnesses be so diagnosed? Formally:

Problem 1 (Correctness of the diagnosis). For each atom $\text{INCORRECTNESS}(A, B \text{ IF } Q)$ in the (relative) denotation of the program “incorrectness diagnoser”, is it true that $B \text{ IF } Q$ represents an incorrectness of $P$ in I, and also that some tree with root $A$ has $B \text{ IF } Q$ as an implication which is critical for RP and I?

Problem 2 (Completeness of the diagnosis). If $A$ is an incorrectness symptom of $P$ in I, for each tree with root $A$ which is conformable to RP, and for each implication $B \text{ IF } Q$ of that tree which is critical for RP and I, is it true that the atom $\text{INCORRECTNESS}(A, B \text{ IF } Q)$ is in the (relative) denotation of the program “incorrectness diagnoser”?
These two problems could be seen as part of the problem of total correctness of the program “incorrectness diagnoser” itself, but we do not consider here this very formal aspect.

Now we can state a result of “limitation”, which is a case of incompleteness:

**Proposition 2.** There is no logic program “incorrectness diagnoser” for which we have completeness of the diagnosis and correctness at the same time, i.e., there is no program which, for any $P$ and $I$, gives a positive answer for the two previous problems.

This theoretical limitation is intrinsic to the problem, and it is independent of the diagnosis method and of the formalism for representing clauses by terms. It comes from the function of the variables in the strict framework of logic programming. Its proof is quite simple: an incorrectness can have an instance which is not an incorrectness, while if the atom $\text{INCORRECTNESS}(A, B, \text{IF } Q)$ is in the denotation of the “incorrectness diagnoser”, then all the instances of this atom must be also in this denotation; so a positive answer for the two previous problems gives a contradiction.

### 5.3. A Program Incorrectness Diagnoser

Let us say that an atom $A$ is **unsatisfiable** in an interpretation $I$ if no instance of $A$ is in $I$ (i.e., in usual logic, in the $L$-interpretation identified with $I$, $A$ is false for any assignment $\theta$).

Now we define a new function symbol system $\Phi_1$ by adding to $\Phi_0$ a new constant symbol denoted by $\text{NOT DEFINED}$. On the other hand, we define a new relation symbol system $\Pi_1$ consisting of the symbols $\text{INCORRECTNESS}, \text{CLAUSE}, \text{VALID}, \text{UNSATISFIABLE}$ with respective arities 2, 1, 1, 1.

It is with $\text{ATOM}(\Pi_1, \Phi_1)$ that we define the clauses of the logic program “incorrectness diagnoser” (see Figure 1). This program is an adaptation to our theoretical framework of a program in Shapiro [7] (there are some comments on the differences between the two programs further on).

Because of the recursive nature of the problem, we have to deal not only with atoms and trees but with conjunctions and ramifications as well. For the same reason we have to deal not only with $A$ which is an incorrectness symptom but with any atom $A$, so we need the symbol $\text{NOT DEFINED}$ for the case where there is no incorrectness to be associated with $A$.

Now let $I$ be an intended denotation [$I \subseteq \text{ATOM}(\Pi, \Phi)$], and $P$ a logic program the clauses of which are defined with atoms in $\text{ATOM}(\Pi, \Phi)$. Let $\mathbf{RP}$ be a set of implications representing $P$ [so $\mathbf{RP} \subseteq \text{TERM}(\Phi)$]. Then let $S(\mathbf{RP}, I)$ be the set of the following atoms [in $\text{ATOM}(\Pi_1, \Phi_1)$]:

- $\text{CLAUSE}(A \text{ IF } Q)$ where $A \text{ IF } Q$ is in $\mathbf{RP}$,
- $\text{VALID}(A)$ where $A$ is valid in $I$,
- $\text{UNSATISFIABLE}(A)$ where $A$ is unsatisfiable in $I$.

Let $\Delta(\mathbf{RP}, I)$ be the **denotation of the program “incorrectness diagnoser” relative to the set** $S(\mathbf{RP}, I)$. 
The following theorem gives a partial solution to Problem 2:

**Theorem 2.** Let $Q_1$ be the root conjunction of a ramification $r$ which is conformable to $R_P$.

If $r$ has all its atoms in $I$, then $\text{INCORRECTNESS}(Q_1, \text{NOT DEFINED})$ is in $\Delta(R_P, I)$.

If $r$ has an implication $B \text{ IF } Q$ which is critical for $R_P$ in $I$, with $B$ unsatisfiable in $I$, then $\text{INCORRECTNESS}(Q_1, B \text{ IF } Q)$ is in $\Delta(R_P, I)$.

**Proof.** A tedious but easy checking by induction on ramifications. □

Note that if the atom $B$ is without variables, then $B$ is unsatisfiable in $I$ if and only if $B$ is not (valid) in $I$. So if we restrict ourselves to diagnosis of incorrectness without variables, there is a total solution to Problem 2.

Now the following theorem gives a total solution to the Problem 1:

**Theorem 3.** Let $Q_1$ be a conjunction. If $\text{INCORRECTNESS}(Q_1, \text{NOT DEFINED})$ is in $\Delta(R_P, I)$ then $Q_1$ is the root conjunction of a ramification which is conformable to $R_P$ and which has all its atoms in $I$. If $B \text{ IF } Q$ is an implication and $\text{INCORRECTNESS}(Q_1, B \text{ IF } Q)$ is in $\Delta(R_P, I)$ then $Q_1$ is the root conjunction of a ramification (which is not necessarily conformable to $R_P$) which has the implication $B \text{ IF } Q$ critical for $R_P$ in $I$. 

FIGURE 1. "Incorrectness diagnoser".
ERROR DIAGNOSIS

PROOF. The basic idea is checking by induction on $\Delta(RE, I)$ as for Proposition 1 (Section 4.2). $\Delta(RE, I)$ is the (absolute) denotation of a program [let us denote it by $P(RE, I)$] which contains the seven clauses of the "incorrectness diagnoser" together with some other clauses, the heads of which are atoms in $S(RE, I)$ (see Section 2.4). The essential point would be to inspect the seven clauses, and to associate to each one of them a concatenation of ramifications or a root operation. More precisely, the induction requires a checking for each instance of each clause. But because of this notion of instance, things are not so simple: a proof by induction on $\Delta(RE, I)$ requires substitutions with range in $TERM(\Phi)$, while in conjunctions, implications, and ramifications the terms must be only in $TERM(\Phi)$. One must be careful about some "pathological" atoms which could appear in $\Delta(RE, I)$. The remainder of this proof is devoted to these tedious technical precautions.

(a) It is easy to see that we can choose a map

$$p : TERM(\Phi) \to TERM(\Phi)$$

such that $p(t) = t$ if $t$ is in $TERM(\Phi)$, and

$$p(f(t_1, \ldots, t_n)) = f(p(t_1), \ldots, p(t_n))$$

If $f$ is a symbol of $\Phi$ and the $t_i$ are in $TERM(\Phi)$. (For example take a fixed variable to be associated with the terms which cause problems.) Now for each substitution $\theta : V \to TERM(\Phi)$ we define $\theta^\sim : V \to TERM(\Phi)$ by $\theta^\sim(x) = p(\theta(x))$. Then we have

$$p(t\theta) = t\theta^\sim$$

if $t$ is in $TERM(\Phi)$.

(b) We define a generalized atom as a term $R(t_1, \ldots, t_m)$ in $TERM(\Phi)$ where $R$ is a symbol of $\Pi$, and we define

$$q(R(t_1, \ldots, t_m)) = R(p(t_1), \ldots, p(t_m))$$

which is in $ATOM(\Pi, \Phi)$. We have $q(A) = A$ if $A$ is in $ATOM(\Pi, \Phi)$. In the same way we define a generalized conjunction and a generalized implication (in conjunction and implication we replace atom by generalized atom). We extend $q$ by $q(\text{EMPTY}) = \text{EMPTY}$ and

$$q(A \text{ AND } B) = q(A) \text{ AND } q(B), \quad q(A \text{ IF } Q) = q(A) \text{ IF } q(Q).$$

$q$ is the identity on the true conjunctions and implications. Now, for any atom $A$ in $ATOM(\Pi, \Phi)$ and $\theta : V \to TERM(\Phi)$, $A\theta$ is a generalized atom, and $q(A\theta) = A\theta^\sim$. It is the same for any conjunction and any implication.

(c) For each atom in $\Delta(RE, I)$ which can be written in the form $\text{VALID}(t)$, we have $t = A\theta$ where $A$ is an atom valid in $I$ [$A$ being an atom of $ATOM(\Pi, \Phi)$] and $\theta : V \to TERM(\Phi)$. Thus $t$ is a generalized atom and $q(t) = A\theta^\sim$, which is valid in $I$. It is the same for $\text{UNSATISFIABLE}(t)$ and unsatisfiability in $I$. For each atom in $\Delta(RE, I)$ which can be written as $\text{CLAUSE}(t_1 \text{ IF } t_2)$, we have

$$t_1 \text{ IF } t_2 = (A \text{ IF } Q)\theta,$$

where $A \text{ IF } Q$ is in $RP$ and $\theta : V \to TERM(\Phi)$. Thus $t_1 \text{ IF } t_2$ is a generalized implication and $q(t_1 \text{ IF } t_2) \in \text{INST}(RP)$ because

$$q(t_1 \text{ IF } t_2) = (A \text{ IF } Q)\theta^\sim.$$
(d) Now let \( \Omega \) be a subset of \( \text{ATOM}(\Pi, \Phi_1) \) defined as follows: \( \Omega \) is the union of the five following sets \( \Omega_i \):

\( \Omega_0 \) is the set of the instances of all the atoms in \( S(\text{RP}, I) \) by all the substitutions with range in \( \text{TERM}(\Phi_1) \).

\( \Omega_1 \) is the set of atoms \( \text{INCORRECTNESS}(Q_1, \text{NOT DEFINED}) \) where \( Q_1 \) is a generalized conjunction with the following property: \( q(Q_1) \) is the root conjunction of a ramification which is conformable to \( \text{RP} \) and which has all its atoms in \( I \).

\( \Omega_2 \) is the set of atoms \( \text{INCORRECTNESS}(Q_1, B \text{ IF } Q) \) where \( Q_1 \) is a generalized conjunction and \( B \text{ IF } Q \) is a generalized implication with the following property: \( q(Q_1) \) is the root conjunction of a ramification which has the implication \( q(B \text{ IF } Q) \) critical for \( \text{RP} \) in \( I \).

\( \Omega_3 \) is the set of the atoms \( \text{INCORRECTNESS}(T_1, T_2) \) where \( T_1 \) is not a generalized conjunction.

\( \Omega_4 \) is the set of the atoms \( \text{INCORRECTNESS}(T_1, T_2) \) where \( T_2 \) is neither the symbol \text{NOT DEFINED} nor a generalized implication.

\( \Omega \) is an interpretation. What we need to prove is only that \( \Omega \) is a model of the program that we have denoted by \( P(\text{RP}, I) \) and that has \( \Delta(\text{RP}, I) \) as its (absolute) denotation, because then we have

\[ \Delta(\text{RP}, I) \subseteq \Omega. \]

(e) Now we can easily show by careful checking that \( \Omega \) is a model of \( P(\text{RP}, I) \). We have to associate to each clause of the program "incorrectness diagnoser" a construction of ramification, using results from (c) above for the last three clauses.

5.4. Extension to Relative Correctness

The problem of noncorrectness of \( P \) has been formulated with respect to the absolute denotation of \( P \). But we can extend it now by considering the denotation of \( P \) relative to a set of atoms \( E \).

We could restart with more general new definitions, but we can avoid that by using the fact that the relative denotation of \( P \) is the absolute denotation of some program (Section 2.4).

We make the definitions of correct, incorrectness, incorrectness symptom (Definitions 3, 4, 5 in Section 3) relative to \( E \) by adding to \( P \) the clauses \( A \leftarrow \) where \( A \in E \).

We suppose here that the noncorrectness problems are confined to \( P \) and do not concern \( E \). Formally we suppose \( E \subseteq I \) (for example, this can formalize the fact that evaluable predicates are considered \textit{a priori} without error). In this case,

1. an incorrectness symptom of \( P \) in \( I \) relative to \( E \) cannot be an instance of an atom of \( E \),

2. an incorrectness of \( P \) in \( I \) relative to \( E \) is always an instance of a clause of \( P \),

3. \( P \) is correct in \( I \) relative to \( E \) if and only if \( I \) is a model of \( P \).

In this case Theorems 2 and 3 (Section 5.3) remain true with the following
modifications:

We add to $\Pi_1$ a new relation symbol $\text{REL}$ with arity 1.

We add to the program "incorrectness diagnoser" the following clause:

$$\text{INCORRECTNESS}(x, \text{NOT} \_ \text{DEFINED}) \leftarrow \text{REL}(x).$$

We add to the set $S(\text{RP, I})$ all the atoms $\text{REL}(A)$ where $A \in E$.

[We can prove these extended theorems either by generalizing the previous proofs, or by applying the previous theorems to the new situation. With this last method, we have to deal with the atoms $\text{CLAUSE}(A \text{ IF EMPTY})$ where $A \in E$. But these atoms are eliminated using the previous adjunctions to the "incorrectness diagnoser" and to $S(\text{RP, I}).]

5.5. Comparison with Shapiro's Program

Apart from the comments already made in the introduction and concerning the theoretical framework, we can add some remarks.

If we transpose Shapiro's program [7], taken as a text, exactly into our framework, we must replace our third and fourth clauses with the following clause alone:

$$\text{INCORRECTNESS}(x \text{ AND } y, z) \leftarrow \text{INCORRECTNESS}(x, \text{NOT} \_ \text{DEFINED}), \text{INCORRECTNESS}(y, z).$$

It is easy to see that the denotation of this new program [relative to $S(\text{RP, I})$] is a subset of the denotation of our program. If we want Theorems 2 and 3 to hold also for this new program, we must replace "critical implication" by "first critical implication", where "first" refers to a "postorder traversal" on a ramification.

In fact this restriction is justified by the PROLOG framework of Shapiro and corresponds to the search strategy of the interpreter. More precisely, with this new program and this restriction, but within our formalism, the incorrectness which can be diagnosed depends on the set $\text{RP}$ chosen to represent $P$, i.e., it depends on the order of the atoms of $Q$ in $A \text{ IF } Q$. It is just a transposition of the effect, upon the search strategy of the interpreter, of a reordering of atoms in bodies of clauses. But these questions are outside the theoretical framework of this paper.

The relation symbol $\text{CLAUSE}$ is a parallel, in our framework, of a usual built-in procedure of PROLOG systems. The relation symbols $\text{VALID}$ and $\text{UNSATISFIABLE}$ are parallels, in our framework, of some procedures used by Shapiro for asking questions about the intended semantics. They work like an "encapsulation" of some in-out operations, allowing as much as possible the extension of the declarative semantics of logic programming.

6. THE PROBLEM OF NONSUFFICIENCY

This problem is, in a way, dual to the previous one (noncorrectness). We are going to use a similar method, outlining only the main points.
6.1. Insufficiency Diagnosis by a Logic Program

We say that an atom $B$ is a subroot of a ramification $r$ if there is a subtree $A_p r_1$ of $r$ such that $B$ is an atom of $r_1$.

Definition 7. Let $RP$ be a set of implications and $I$ an intended denotation. A ramification $r$ is said to be quasiconformable to $RP$ and $I$ if

all the subroots of $r$ are in $I$,

for each subtree $A_p r_1$ of $r$ such that $r_1$ is not empty, the root implication of $A_p r_1$ is in $INST(RP)$.

Lemma 1. Let $P$ be the logic program represented by $RP$, and $A$ an atom which is not in $gfp(T_P)$. Then $A$ is the root of a tree which is quasiconformable to $RP$ and $I$ and which has at least one leaf not in $T_p(I)$.

Proof. We suppose that all trees with root $A$ and which are quasiconformable to $RP$ and $I$ have all their leaves in $T_p(I)$, and we are going to prove that $A \in gfp(T_P)$.

Let $r_0 = A$; $r_0$ is a tree with root $A$, and it is quasiconformable to $RP$ and $I$. Now $A$ is a leaf of $r_0$, thus $A \in T_p(I)$; thus there exists $A \mathbin{\text{IF}} Q$ in $INST(RP)$ with all the atoms of $Q$ in $I$. By an obvious root operation with these atoms of $Q$ and $A$, we obtain a tree with root $A$ and which is again quasiconformable to $RP$ and $I$ [the case $Q = \text{EMPTY}$ is not excluded; then $r_1 = r_0 = A$ and thus $A \in D(P) \subseteq gfp(T_P)$].

So by induction we define a sequence $r_0, \ldots, r_n, \ldots$ of trees all with root $A$ and quasiconformable to $RP$ and $I$. Here $r_{n+1}$ is obtained by possibly attaching new atoms of $I$ to the leaves of $r_n$, using implications in $INST(RP)$; these leaves are in $T_p(I)$ by hypothesis [if we attach no more atoms, it is because $r_n$ is conformable to $RP$; then $r_n = r_{n+1} = \ldots$ and thus $A \in D(P) \subseteq gfp(T_P)$].

Let $J$ be the set that consists of all atoms of each $r_n$. By construction $A \in J \subseteq T_p(J)$; thus $J \subseteq gfp(T_P)$ [because $gfp(T_P)$ is also the greatest $J$ such that $J \subseteq T_p(J)$]; thus $A \in gfp(T_P)$, Q.E.D. □

Now let $I$ be an intended denotation, $P$ a logic program, and $A$ an insufficiency symptom of $P$ for $I$. Let $RP$ be a set of implications which represents $P$. Because $A$ is not in $gfp(T_P)$, $A$ is the root of a tree which is quasiconformable to $RP$ and $I$ and which has at least one leaf $B$ which is not in $T_p(I)$. Now all the atoms of this tree are in $I$ (even $A$ is in $I$), so we are sure that $B$ is in $I$. Then $B$ is an insufficiency of $P$ for $I$, which we may see as a "cause" of the symptom $A$.

Formally we would like an atom $\text{INSUFFICIENCY}(A, B)$ to be in the denotation of a program "insufficiency diagnoser", relative to a set $S$ of atoms which depends on $P$ and $I$.

6.2. Correctness and Completeness of the Diagnosis

Again naturally two problems arise:

Problem 3 (Correctness of the diagnosis). For each atom $\text{INSUFFICIENCY}(A, B)$ in the (relative) denotation of the program "insufficiency diagnoser", if $A$ is in $I$, is it...
true that $B$ is an insufficiency of $P$ for $I$, and that $B$ is a leaf of a tree with root $A$ and which is quasiconformable to $\text{RP}$ and $I$?

**Problem 4 (Completeness of the diagnosis).** If $A$ is an insufficiency symptom of $P$ for $I$, for each tree with root $A$ which is quasiconformable to $\text{RP}$ and $I$, and for each leaf $B$ of that tree which is an insufficiency of $P$ for $I$, is it true that the atom $\text{INSUFFICIENCY}(A, B)$ is in the (relative) denotation of the program "insufficiency diagnoser"?

Again we can state (the proof is similar) a result of "limitation", incompleteness in a sense, i.e. the impossibility of a method which is complete (and correct at the same time).

### 6.3. A Program Insufficiency Diagnoser

Let us say that an atom $A$ is an *impossibility* of $P$ for $I$ if no instance of $A$ is in $T_P(I)$. [So in particular $A$ is not in $T_P(I)$, and if $A \in I$, then $A$ is an insufficiency of $P$ for $I$.]

Now we define a new function symbol system $\Phi_2$ by adding to $\Phi_0$ the constant symbol $\text{NOT DEFINED}$ and a new function symbol with arity 1, denoted $\text{ATOM}$ (we do so in order to distinguish between an atom which is a true insufficiency and the constant $\text{NOT DEFINED}$; in the "incorrectness diagnoser" that function was fulfilled by $\text{IF}$). On the other hand, we define a new relation symbol system $\Pi_2$, the symbols of which are $\text{INSUFFICIENCY}$, $\text{CLAUSE}$, $\text{SATISFIABLE}$, $\text{IMPOSSIBLE}$ with respective arities $2, 1, 1, 1$. (The name $\text{SATISFIABLE}$ is chosen for intuitive reasons, and its correspondence with $\text{VALID}$ is another hint of the duality between insufficiency and incorrectness; but in our theoretical framework it will be used for exactly the same atoms as $\text{VALID}$.)

It is with $\text{ATOM}(\Pi_2, \Phi_2)$ that we define the clauses of the logic program “insufficiency diagnoser”: see Figure 2. This program is an adaptation to our theoretical framework of a program in Shapiro [7].

Let $I$ be as previously an intended denotation, $P$ a logic program, and $\text{RP}$ a set of implications representing $P$. Now let $S(\text{RP}, I)$ be the set of the following atoms:

- $\text{CLAUSE}(A \text{IF} Q)$ where $A \text{IF} Q$ is in $\text{RP}$,
- $\text{SATISFIABLE}(A)$ where $A$ is valid in $I$,
- $\text{IMPOSSIBLE}(A)$ where $A$ is an impossibility of $P$ for $I$.

And let $\Delta(\text{RP}, I)$ be the *denotation of the program “insufficiency diagnoser” relative to the set $S(\text{RP}, I)$.*

The following theorem gives a partial solution to Problem 4:

**Theorem 4.** Let $Q$ be the root conjunction of a ramification $r$ which is quasiconformable to $\text{RP}$ and $I$. If $r$ is conformable to $\text{RP}$, then $\text{INSUFFICIENCY}(Q, \text{NOT DEFINED})$ is in $\Delta(\text{RP}, I)$. If $r$ has a leaf $B$ which is an impossibility of $P$ for $I$, then $\text{INSUFFICIENCY}(Q, \text{ATOM}(B))$ is in $\Delta(\text{RP}, I)$.

**Proof.** As in Theorem 2 (Section 5.3).
INSUFFICIENCY(EMPTY, NOT_DEFINED) ←

INSUFFICIENCY(x AND y, ATOM(u))
← INSUFFICIENCY(x, ATOM(u))

INSUFFICIENCY(x AND y, ATOM(u))
← INSUFFICIENCY(y, ATOM(u))

INSUFFICIENCY(x AND y, NOT_DEFINED)
← INSUFFICIENCY(x, NOT_DEFINED),
INSUFFICIENCY(y, NOT_DEFINED)

INSUFFICIENCY(x, z)
← CLAUSE(x IF y),
SATISFIABLE(y),
INSUFFICIENCY(y, z)

INSUFFICIENCY(x, ATOM(x))
← IMPOSSIBLE(x)

SATISFIABLE(EMPTY) ←

SATISFIABLE(x AND y)
← SATISFIABLE(x),
SATISFIABLE(y)

Note that if B is without variables, then B is an impossibility of P for I if and only if B is not in T_p(I). So if we restrict ourselves to diagnosis of insufficiency without variable, there is a total solution to Problem 4.

Now the following theorem gives a total solution to Problem 3:

Theorem 5. Let Q be a conjunction. If INSUFFICIENCY(Q, NOT_DEFINED) is in Δ(RP, I) then Q is the root conjunction of a ramification which is conformable to RP and which has its subroots in I. If B is an atom and INSUFFICIENCY(Q, ATOM(B)) is in Δ(RP, I), then Q is the root conjunction of a ramification which is quasiconformable to RP and I and which has a leaf B not in T_p(I).

PROOF. As in Theorem 3 (Section 5.3). □

6.4. Extension to Relative Sufficiency

As in Section 5.4, we make the definitions of sufficient, insufficiency, and insufficiency symptom relative to a set of atoms E. For any E, an insufficiency and an insufficiency symptom of P for I relative to E cannot be instances of atoms in E.

The previous Theorems 4 and 5 still hold if we add the clause

INSUFFICIENCY(x, NOT_DEFINED) ← REL(x)

to the program “insufficiency diagnoser”, and the atoms REL(A), where A ∈ E, to the set S(RP, I).
6.5. **Comparison with Shapiro’s Program, and Finite Failure**

If we transpose exactly Shapiro’s program [7], taken as a text, in our framework we must replace our third and fourth clauses by the following clause alone:

\[
\text{INSUFFICIENCY}(x \text{ AND } y, z) \\
\leftarrow \text{INSUFFICIENCY}(x, \text{NOT}\_\text{DEFINED}), \\
\text{INSUFFICIENCY}(y, z)
\]

It is easy to see that the denotation of this new program [relative to \(S(RP, I)\)] is a subset of the denotation of our program. As in Section 5.5, we should change some definitions in order to specify what can be diagnosed; the latter would depend on the set \(RP\) chosen to represent \(P\). In fact it would be a restriction, natural in the PROLOG framework of Shapiro: in practice, if the programmer tries to diagnose an insufficiency, starting from a symptom \(A\), it is because a previous experiment has proved that \(A\) makes the program \(P\) “finitely fail”.

In our theoretical framework, we have already shown the existence of some link between insufficiency symptom and finite failure (end of Section 3). It should be noted that the two notions of “finite failure” are not the same: the notion used by Shapiro [7, 8] in the PROLOG framework is based on the search strategy of the interpreter, while the notion defined by Apt and Van Emden [1] for logic programming has the form “there is a finite failure search tree ...”. Again these questions are outside the framework of this paper.

7. **CONCLUSION**

We have tried to extend as much as possible the field where methods of fixpoint and declarative semantics can be applied, in order to express clearly and precisely some notions about errors and error diagnosis. Now we make some remarks about the applications of this work with an interpreter of logic programming, for example PROLOG (Shapiro [8] gives justifications for using PROLOG on the two “levels”, erroneous program and error diagnosis program):

(a) We can consider that in a way we have described a “theoretical envelope” of requirements that an implementation must meet, considering the difference between the theoretical interpreter “SLD resolution” and a PROLOG interpreter. This point of view may be convenient when the erroneous program is in “pure PROLOG”, or more generally when it has predicates which can be formalized with the notion of relative denotation. From this point of view, one of the things to do is to write procedures for \(\text{CLAUSE}(x \text{ IF } y)\), \(\text{VALID}(x)\), etc. Another is to take the search strategy of the interpreter into account.

(b) It would be useful to extend error diagnosis to programs with all PROLOG possibilities, including control predicates. Shapiro [8] examines this question from a pragmatic point of view. But this leads beyond the theoretical framework of this paper.

(c) Finally we have proved “limitation” (incompleteness) results (Sections 5.2 and 6.2) on the ability of a logic program to diagnose errors (with variables) in logic programs. But these results do not apply to the ability of a PROLOG program to diagnose errors in “pure PROLOG” programs; it may be possible to overcome this
limitation by using some "extralogical" possibilities of PROLOG. Again this leads beyond the scope of this paper and it is the topic of a work to come.

The author wishes to express his gratitude to Pierre Deransart who, at INRIA, introduced him to logic programming, gave him this research subject, and has advised and encouraged him. Thanks are due also to Bernard Lorho for his support, as well as to the members of the group "Langages et Traducteurs".

REFERENCES