Extending Horn clause logic with implication goals*

Laura Giordano, Alberto Martelli
Dipartimento di Informatica, Università di Torino, C.so Svizzera 185, 10149 Torino, Italy

Gianfranco Rossi
Dipartimento di Matematica e Informatica, Università di Udine, Via Zanon 6, 33100 Udine, Italy

Communicated by G. Levi
Received July 1989
Revised March 1990

Abstract

The paper deals with the problem of extending positive Horn clause logic by introducing implication in goals as a tool for program structuring. We allow a goal $G_i$ in a clause $G_1 \land \cdots \land G_n \rightarrow A$ to be not only an atom but also an implication $D \supset G$ (we shall call it an implication goal), where $D$ is a set of clauses and $G$ a goal. This extension of the language allows local definitions of clauses in logic programs. In fact, an implication goal $D \supset G$ can be thought of as a block $(D, G)$, where $D$ is the set of local clause declarations. In this paper we define a language with blocks in which, as in conventional block structured programming languages, static scope rules have been chosen for locally defined clauses. We analyze static scope rules, where a goal can refer only to clauses defined in statically surrounding blocks, and we compare this extension with other proposals in the literature. We argue, on account of both implementative and semantic considerations, that this kind of block structured language is a very natural extension of Horn clauses when used as a programming language. We show it by defining an operational, fixpoint and model-theoretic semantics which are extensions of the standard ones, and by proving their equivalence. We show also that static scope rules can be obtained by interpreting $\supset$ as classical and $\Rightarrow$ as intuitionistic implication with respect to Herbrand interpretations.

1. Introduction

A number of recent research efforts in logic programming have focused on the problem of introducing local definitions of clauses in logic languages. The main

* This work has been partially supported by MPI 40% and by CNR-Progetto Finalizzato “Sistemi Informatici e Calcolo Parallelo” under grant no. 89.00038.69.
motivation for such an extension is to provide an elegant solution to the lack of program structuring facilities, which is widely recognized as one of the main drawbacks of Horn clause logic as a programming language.

Different approaches have been advocated to face this problem in different proposals. For instance, Bowen and Kowalski [2] show how to introduce local definitions at the metalevel, whereas Warren [16] proposes a modal operator "assume". Gabbay and Reyle [3,4] present N_Prolog, an extension of logic programming which allows local definitions and which is designed mainly to deal with hypothetical reasoning. A similar extension is proposed by Miller [11] and more recently by McCarty [10]. Relying on a similar idea, Monteiro and Porto [14] propose "contextual logic programming" to develop a theory of modules in logic programming. Nait Abdallah [15] defines "ions" to deal with local definitions.

In this paper, we aim at defining a logic language with blocks in the style of conventional programming languages. To this purpose, we tackle the problem of local definitions of clauses by extending positive Horn clause logic with implication goals, that is by allowing implications of the form \( D \Rightarrow G \), where \( G \) is a goal and \( D \) is a set of clauses, to occur in goals and in clause bodies. In fact, an implication goal \( D \Rightarrow G \) can be considered as a block \((D, G)\), where \( D \) is a set of clause definitions local to \( G \) (that is, clauses in \( D \) can be used only to prove \( G \)).

The approach of using implication in goals as a structuring tool is analogous to that advocated by Gabbay and Reyle [3,4], Miller [11], McCarty [10] and Monteiro and Porto [14]; our proposal, however, differs from those in many respects. In fact, according to the semantics chosen for the implication goal, several different extensions of Horn clause logic can be obtained, each one characterized by different visibility rules for locally defined clauses.

In particular, in this paper we pursue the idea of defining a logic language with static scope rules for clause definitions, in which, as in most conventional programming languages, the rules for using a clause are determined by the static nesting of blocks in the program text. Indeed, static scope rules have the well-known advantage to allow efficient implementations of the language by means of compilation techniques. The problem of defining suitable visibility rules for locally defined clauses will be discussed in more detail in the next section, where we shall give an informal description of the language with blocks.

The rest of the paper is organized as follows. In Section 3 we define more precisely the language with blocks by means of its operational semantics. We show there that a unique implication connective is not sufficient for the language and thus we must introduce two different implications with different meanings: one in definite clauses \( (G_1 \land \cdots \land G_n \Rightarrow A) \) and the other one in goals \( (D \Rightarrow G) \).

In Section 4 we define the fixpoint semantics of the language as an extension of the standard fixpoint semantics for Horn clauses. The operational semantics is shown to be sound and complete with respect to the corresponding fixpoint semantics.
An interesting problem is to see if this kind of language allows also a model-theoretic semantics. In Section 5 we show that a very simple semantics can be given as an extension of standard semantics by defining satisfiability of formulas with respect to interpretations (subsets of \( B(P) \)). Satisfiability is defined as in the standard case (so \( \rightarrow \) is the classical implication) with the addition of the definition for \( D \Rightarrow G \), which is satisfiable in an interpretation \( I \) if and only if \( D \) implies \( G \) in all the interpretations \( I' \) which contain \( I \). Thus we do not need a general Kripke semantics (with worlds) as in the case of N-Prolog [3, 4] whose model-theoretic semantics is that of intuitionistic logic. In our case, it is sufficient to consider a subset of Kripke interpretations. This kind of semantics possesses properties analogous to those of the classical case, such as a minimal model which can be shown to be equal to the least fixpoint.

The model-theoretic semantics of \( \Rightarrow \) given in Section 5 is different in general form the intuitionistic semantics. However, in Section 6 we show that the two semantics are equivalent for the language of the paper with respect to Herbrand interpretations, and therefore we can consider \( \rightarrow \) and \( \Rightarrow \) as the classical and intuitionistic implication, respectively.

In Section 7 we introduce a more concrete operational semantics for the language and we give some hints on possible efficient implementations of this language in the style of conventional programming languages.

2. Visibility rules for locally defined clauses

An implication goal \( D \Rightarrow G \) can be considered as a block, where \( G \) is a goal and \( D \) is a set of local clauses. The set of clauses \( D \) corresponds to local procedure declarations in conventional programming languages. Since \( G \) can be itself an implication goal or can contain implication goals, it is possible to have nested blocks. Moreover, implication goals are allowed also in the bodies of the clauses. Indeed, a goal \( G_i \) in a clause \( G_1 \land \cdots \land G_n \Rightarrow A \) can be not only an atom but also an implication \( D \Rightarrow G \). In the following we describe different semantics which implication goals can be given.

2.1. Closed blocks

A natural meaning for the implication goal \( D \Rightarrow G \) is that of \( G \) being a logical consequence of \( D \). In other words, it is quite natural to define the derivability of an implication goal in the following way: \( D \Rightarrow G \) is derivable from \( P \) if \( G \) is derivable from \( D \), disregarding the content of \( P \). With such an informal semantics, the implication goal clearly defines closed environments of clause definitions (hereafter called closed blocks). In this case, the goal \( D \Rightarrow G \) clearly corresponds to the metapredicate \( \text{demo}("D", "G") \) defined by Bowen and Kowalski [2].
Example 1.

\[ G = s \]

\[ P = \{ q, \]

\[ q \land (((r \rightarrow p) \land r) \supset p) \rightarrow s \}. \]

The program \( P \) is composed of two clauses, the second one containing a block definition. The goal \( G \) succeeds from the program \( P \) since \( q \) is provable in \( P \) and \( p \) is provable in the closed block \(((r \rightarrow p) \land r)\). If, on the contrary, the second clause of \( P \) is replaced by the clause

\[ (((q \land r \rightarrow p) \land r) \supset p) \rightarrow s, \]

the goal \( G \) would fail. In fact, in this case, \( q \) is called from the inner block while its definition is in the outer program.

A block structured language requires the introduction of explicit quantifiers specifying the scope of variables. In fact, since a clause definition can occur in the body of another clause, it is necessary to distinguish between its local variables and variables coming from the external environment. The use of explicit quantifiers allows us to give static scope rules to variables.

Let us consider two simple examples of closed blocks with explicitly quantified variables. We shall use a Prolog-like syntactic notation in which a program is represented as a set of clauses separated by a dot, and ",," is used in place of "\&".

Example 2 (Miller [13]).

\[ \forall L, K \ (\text{rev}(L, K) :- \]

\[ \{ \forall K \ \text{rev1}([], K, K) \}

\[ \forall X, L, K, Acc \ (\text{rev1}([X | L], K, Acc) :- \text{rev1}(L, K, [X | Acc])) \]

\[ } \supset \text{rev1}(L, K, []). \]

In this program a predicate \( \text{rev}(L, K) \) is defined which is true when \( K \) is the reverse list of \( L \) (e.g. \( \text{rev}([a, b, c], [c, b, a]) \)). The procedure \( \text{rev} \) makes use of the predicate \( \text{rev1}(L1, L2, L3) \) which builds up in \( L3 \), element by element, the reverse list of \( L1 \) and then gives it back in \( L2 \). Since \( \text{rev1} \) is called only within \( \text{rev} \), it is convenient to define it locally to \( \text{rev} \) rather than in the global database.

Explicit quantifiers for variables allow nonlocal variables to occur in locally defined clauses. Hence closed blocks are closed with respect to clause definitions but not with respect to variables occurring in them (differently from theories in Bowen and Kowalski's proposal [2], which are closed also w.r.t. variables).
Example 3.

\[ \forall A, L_1, L_2 \text{(subset}(A \mid L_1), L_2) : \]
\[ \{ \forall L \text{ member} A(A \mid L) \} \]
\[ \forall X, L \text{ member} A([X \mid L]) \Rightarrow \text{member} A(L) \]
\[ \} \Rightarrow \text{member} A(L_2), \text{subset}(L_1, L_2) \).
\[ \forall L_2 \text{ subset}(\text{[]}, L_2) \).

The predicate subset \((L_1, L_2)\) is true if \(L_1\) is a subset of \(L_2\) (e.g., subset([a, c], [a, b, c])). A predicate member \(A(L)\) is defined locally to it and tests the occurrence of a given element \(A\) in the list \(L\). The variable \(A\) in the first clause of member \(A\) and is not local to that clause; it is quantified in the external clause defining subset. Therefore the predicate member \(A\) has one argument less than usual.

As noticed above, a goal \(D \rightarrow G\) with the semantics of closed blocks clearly corresponds to the metapredicate demo ("D", "G") defined within the object language. It must be mentioned that a first proposal for simulating demo in the object language has been presented by Gabbay and Reyle in [3, 4], where two primitives suspend and restore are introduced to simulate demo in N-Prolog as follows:

\[ \text{demo} \left(\text{"D"}, \text{"G"}\right) = \text{suspend} \left(D \rightarrow G\right) \text{ restore}, \]

where \(\rightarrow\) is the intuitionistic implication and suspend and restore are primitives which are used for suspending the data in the database (in order to compute \(D \rightarrow G\) in the empty context) and then restoring the data back, respectively. These primitives are no longer needed if the semantics of the implication in goals is defined in a suitable way so as to provide closed blocks.

This kind of semantics also makes implication goals suitable for supporting the introduction of module constructs in a logic language [6, 7]. Since modules are intended to be used mainly for "programming in the large", it seems reasonable to see them as closed environments (that is closed collections of clauses) with very limited and controlled communications with the external environment.

2.2. Open blocks

As a difference with closed blocks, blocks of conventional programming languages (hereafter called open blocks) allow local procedures to be defined in terms of other procedures occurring in externally nested blocks. In this way, the meaning of a block in a program turns out to be strictly dependent on its external environment. Therefore, open blocks seem to be a more adequate structuring tool for "programming in the small" rather than for defining a module facility. Open
blocks can be introduced in the language as a natural extension of closed blocks defined above by suitably modifying the semantics for the implication goal.

A structured language with open blocks requires scope rules for locally defined clauses. Two alternatives are feasible as usual, namely static or dynamic scope rules. As far as variables are concerned, the use of explicit quantifiers allows us to give them static scope rules, as we have seen for closed blocks. We are now mainly concerned with scope rules for clauses. It is worth noticing that scope rules for clauses can be more complex than scope rules for procedures in conventional programming languages, because in a logic program a predicate definition is usually given by means of several clauses which can occur in different blocks.

Visibility rules for local clause definitions depend on the semantics which is chosen for the implication goal. The operational semantics given to implication goals in the papers by Gabbay and Reyle, Miller and McCarty mentioned in the introduction is informally the following: A goal \( D \supset G \) can be solved in a program \( P \) if the goal \( G \) can be solved in the program \( P \cup D \). For instance, to solve the goal \( a \supset b \) in the program \{ \( a + b \) \}, we solve \( b \) in the program \{ \( a \rightarrow b, a \) \}. This semantics is adequate for open blocks and also to support hypothetical reasoning, i.e. by assuming \( a \) and knowing \( a + b \), we can deduce \( b \).

It is easy to see that dynamic scope rules are required by this semantics. Given the goal \( D \supset G \) to be proved in a program \( P \), after the clauses in \( D \) have been added to the program \( P \), they are no more distinguishable from other clauses of \( P \) and can be used in the subsequent refutation as global clauses. The added clauses are no more visible as soon as the proof of the goal \( G \) terminates (i.e. they are removed from the set of global clauses). Therefore, the set of clauses which can be used to solve a goal \( G \) depends on the sequence of goals generated till that moment in the proof containing \( G \). Of course, this set can be determined only dynamically.

**Example 4.**

\[ G = s \]

\[ P = \{ r \rightarrow q, \quad (((q \rightarrow p) \wedge r) \supset p) \rightarrow s \}. \]

The proof of the goal \( s \) in \( P \) yields

- goal \( (((q \rightarrow p) \wedge r) \supset p) \) in \( P \)
- goal \( p \) in \( P' = P \cup \{ q \rightarrow p, r \} \)
- goal \( q \) in \( P' \)
- goal \( r \) in \( P' \)

which succeeds. The proof of the goal \( q \) uses the clause \( r \) defined in the inner block, which is visible at that point since the block has been added to the program \( P \). If, on the contrary, the goal \( q \) is called directly from the outer environment its proof fails.
With this operational semantics, the implication $D \Rightarrow G$ cannot be the classical one. In fact, in classical logic

$$(a \rightarrow b) \rightarrow a = a,$$

while, with the above operational semantics, the proof of the goal $G = a$ from the program $P = \{(a \rightarrow b) \rightarrow a\}$ fails. Gabbay and Reyle [3] have shown instead, that having a unique implication symbol $\supset$ in goals and clauses, the interpretation of $\Rightarrow$ as the intuitionistic implication corresponds to the operational semantics above, and they have given the language a model-theoretic semantics based on "worlds". Miller [11] has given the language a fixpoint semantics for the same implication.

2.3. Open blocks with static scope rules

In this paper, on the contrary, we pursue the idea of defining a logic language with open blocks and static scope rules for clause definitions like those of conventional block structured programming languages. As a difference with the proposal discussed above, we preserve the distinction between the implication in goals ($\Rightarrow$) and in clauses ($\rightarrow$). The semantics of the implication $\rightarrow$ will be kept unaltered with respect to the semantics of the implication in Horn clause logic. Since scope rules are static, the set of clauses which can be used in the refutation of a goal depends only on the block structure of the program and can be statically determined. In this way to solve an atomic goal which comes from the body of a clause defined in a block, only the clauses defined in that block or in external enclosing blocks can be used. For instance, Example 4 would fail with static scope rules because the clause $r$, defined locally to the second clause of $P$, is not visible from the first clause of $P$. On the contrary, the following example will succeed with static scope rules.

Example 5.

$$G = s$$
$$P = \{q,
(((r \land q \rightarrow p) \land r) \supset p) \rightarrow s\}.$$

The goal $G$ succeeds from the program $P$, since in this case $r$ is used in the same block where it is defined and $q$ is used from an inner block.

This kind of open block appears to be a suitable extension of Horn clauses when used as a programming language. The choice of static scope rules is also justified from the implementation viewpoint, since they have the well-known advantages, to be discussed at the end of the paper, to allow more efficient implementations by allowing compilation of procedure calls. On the other hand, we remark that with our solution neither hypothetical reasoning nor dynamic program modification in general can be carried out.
As a final example of a program using blocks, let us consider the well-known logic program which implements the quicksort algorithm. First we present the usual Prolog implementation and then the corresponding implementation in the language we have defined so far. A Prolog-like notation like that of Example 2 will be employed in the example.

**Example 6.**

\[
\begin{align*}
\text{split}(H, [A|X], [A|Y], Z) & :- \\
& \text{order}(A, H), \text{split}(H, X, Y, Z).
\end{align*}
\]

\[
\begin{align*}
\text{split}(H, [A|X], Y, [A|Z]) & :- \\
& \text{order}(H, A), \text{split}(H, X, Y, Z).
\end{align*}
\]

\[
\begin{align*}
\text{split}(-, [], []). \\
\text{quicksort}([H|T], S) & :- \\
& \text{split}(H, T, A, B), \\
& \text{quicksort}(A, A1), \\
& \text{quicksort}(B, B1), \\
& \text{append}(A1, [H|B1], S).
\end{align*}
\]

Since predicate split is used only by quicksort, we can move its definition inside the body of quicksort in an inner block declaration as follows:

\[
\begin{align*}
\forall H, T, S \quad \text{quicksort}([H|T], S) & :- \\
\exists A, B, A1, B1
\end{align*}
\]

\[
\begin{align*}
& ((\forall A, X, Y, Z \quad \text{split}([A|X], [A|Y], Z) :- \\
& \quad \text{order}(A, H), \text{split}(X, Y, Z). \\
& \forall A, X, Y, Z \quad \text{split}([A|X], Y, [A|Z]) :- \\
& \quad \text{order}(H, A), \text{split}(X, Y, Z). \\
& \text{split}([], [], []). 
\end{align*}
\]

\[
\begin{align*}
& \Rightarrow \text{split}(T, A, B)), \\
& \text{quicksort}(A, A1), \\
& \text{quicksort}(B, B1), \\
& \text{append}(A, [H|B1], S)). \\
& \text{quicksort}([], []). \\
\end{align*}
\]

\[
\begin{align*}
\forall A, H \quad \text{order}(A, H) & :- \cdots
\end{align*}
\]

The scope of variable \( H \) is the whole clause defining quicksort; thus \( H \) can be used in the body of the split procedure as a *global variable* and must not be specified as a parameter of the split procedure itself (the procedure has now one parameter less than the previous definition). We have chosen to quantify variables used in the body of quicksort inside the body itself using an existential quantifier, whereas variables which are local to the definition of split are universally quantified in front of each clause of the procedure. Notice that it is not necessary to move the definition of
the predicate order inside the block in which split is defined. Nevertheless, order
can be used by split since order is defined in an external enclosing block. If, on the
contrary, closed blocks were used, the definition of order should be in the same
block as that of split.

3. The language and its operational semantics

In this section we define a logic language which extends positive Horn clause
logic by introducing open blocks with static scope rules. To describe the syntax and
the operational semantics of this language we shall use the notation of [11].

Let \( A, G \) and \( B \) be metalinguistic variables which represent atomic formulas,
goals and definite clauses, respectively, and let \( T \) be a propositional constant (true).
The syntax of the language is the following:

\[
G := T \mid A \mid G_1 \land G_2 \mid \exists xG \mid D \supset G \\
D := G \rightarrow A \mid D_1 \land D_2 \mid \forall xD.
\]

A program is defined as a set of closed definite clauses.

Notice that what we call clauses are actually not standard clauses, since they can
be composed of a conjunction of clauses and the left-hand part of a clause \( G \rightarrow A \)
is allowed to contain implications. Notice also that a clause of the form \( T \rightarrow A \) will
be simply written as \( A \).

Given a program \( P \) and a closed goal \( G \), we want now to define the meaning of
\( G \) being operationally derivable from \( P \), that is \( P \vdash G \). In order to avoid problems
with variable renaming and substitutions we follow [11] replacing universally quanti-
tified variables in a program with all their possible ground substitutions. Moreover,
conjunctions of clauses are replaced by the corresponding set of clauses. The program
which is obtained from \( P \) in such a way is denoted by \( [P] \). \([P]\) can be defined
recursively as the smallest set of formulas such that

(i) \( P \subseteq [P] \);

(ii) if \( D_1 \land D_2 \in [P] \) then \( D_1 \in [P] \) and \( D_2 \in [P] \);

(iii) if \( \forall xD \in [P] \) then \( [x/t]D \in [P] \) for all closed terms \( t \).

Let us consider first the case of closed blocks. With closed blocks, an implication
goal simply specifies a context switch; thus, to prove a goal \( D \supset G \) in \( P \) we can
simply prove \( G \) in \( D \), forgetting everything about \( P \) since the clauses defined in
externally nested blocks are visible neither from \( G \) not from \( D \). In this case, the
derivability of a closed goal \( G \) from a program \( P \) is defined by induction on the
structure of \( G \), by the following rules:

\( 1' \) \( P \vdash T \);

\( 2' \) if \( A \) is a closed atomic formula, \( P \vdash A \) iff there is a formula \( G \rightarrow A \in [P] \) and
\( P \vdash G \);

\( 3' \) \( P \vdash G_1 \land G_2 \) iff \( P \vdash G_1 \) and \( P \vdash G_2 \);

\( 4' \) \( P \vdash \exists xG \) iff there is some closed term \( t \) such that \( P \vdash [x/t]G \);

\( 5' \) \( P \vdash D \supset G \) iff \( D \vdash G \).
Let us turn to open blocks. As a difference with closed blocks, open blocks allow the external environment to remain visible when entering a block. With dynamic scope rules the union of clauses in \( P \) and \( D_i \) is considered for each goal \( D_i \gg G_i \), so that local definitions are no more distinguishable from global ones. In this case, the operational semantics (as defined for instance in [11]) can be obtained from the semantics defined above by replacing rule (5') with the rule

\[
P \vdash D \gg G \text{ iff } P \cup D \vdash G.
\]

From now on we shall consider open blocks with static scope rules. In this case in order to define an operational semantics for the language, we have to consider lists of programs of the form \( P_1 \cdot \cdots \cdot P_n \) instead of simply programs. In a list \( P_1 \cdot \cdots \cdot P_n \), \( P_1 \) is the initial program while each \( P_i \), for \( i > 1 \), is the conjunction of clauses contained in the \( D_i \) of a block \( D_i \gg G_i \). The higher the index \( i \), the deeper the nesting of the block \( D_i \gg G_i \). Thus the list \( P_1 \cdot \cdots \cdot P_n \) represents the static nesting of blocks in a program \( P \), at some step of the derivation of a goal \( G \) from \( P \).

We define the derivability of a closed goal \( G \) from a nonempty list of programs \( P_1 \cdot \cdots \cdot P_n \) by induction on the structure of \( G \), by the following rules:

1. \( P_1 \cdot \cdots \cdot P_n \vdash T \);

2. if \( A \) is a closed atomic formula, \( P_1 \cdot \cdots \cdot P_n \vdash A \) iff, for some \( i \), \( 1 \leq i \leq n \), there is a formula \( G \rightarrow A \in [P_i] \) and \( P_1 \cdot \cdots \cdot P_i \vdash G \);

3. \( P_1 \cdot \cdots \cdot P_n \vdash G_1 \land G_2 \) iff \( P_1 \cdot \cdots \cdot P_n \vdash G_1 \) and \( P_1 \cdot \cdots \cdot P_n \vdash G_2 \);

4. \( P_1 \cdot \cdots \cdot P_n \vdash \exists x G \) iff there is some closed term \( t \) such that \( P_1 \cdot \cdots \cdot P_n \vdash[x/t]G \);

5. \( P_1 \cdot \cdots \cdot P_n \vdash D \gg G \) iff \( P_1 \cdot \cdots \cdot P_n \vdash D \gg G \).

Consider first rule (2): when a clause \( G \rightarrow A \) in \( P_1 \) is used to refute an atomic goal \( A \), then the clauses in \( P_{i+1}, \ldots, P_n \) cannot be used any more to prove \( G \). This is because the blocks corresponding to \( P_{i+1}, \ldots, P_n \) do not contain the block from which \( G \) is called and therefore are not visible from \( G \). As we can see from rule (5), when the goal is a block \( D \gg G \), the set of local clause definitions \( D \) is added to the list of programs as the tail element and \( G \) is proved from the resulting list of programs. Thus the clauses in \( D \) can be used only to refute goals which come from \( D \) itself or from \( G \).

Notice that, in our proposal, the clauses that define a predicate \( p \) can occur in different blocks of a program and a matching clause for \( p \) is selected nondeterministically from the list of programs \( P_1, \ldots, P_n \) (rule (2)). Differently from conventional structured programming languages and from the proposal of Monteiro and Porto [14] no overriding of predicate definitions applies when entering a new block.

It is important to notice the different usage and meaning of the two implications, \( \rightarrow \) and \( \gg \), have in the language with open blocks and static scope rules. In particular, we point out that \( p \rightarrow q \neq p \gg q \).

A derivation of \( G \) from a nonempty list of programs \( P_1 \cdot \cdots \cdot P_n \) is defined as a finite sequence of pairs \( (W_1, G_1), \ldots, (W_m, G_m) \), where \( W_1 = P_1 \cdot \cdots \cdot P_n \), \( G_1 = G \), \( G_m = T \) and, for \( i = 1, \ldots, m \), \( W_i \vdash G_i \). The derivability of \( G \) from \( W_1 \) can be obtained from the pairs of the sequence which follow \( (W_i, G_i) \) using the above rules (how
many members of the sequence must be considered depends on the rule which is applied).

Example 7. Let us consider the program $P$ of Example 5. The following is a derivation of the goal $G = s$ from $P$:

$$
G_1 = s, \quad W_1 = \{T \rightarrow q, ((r \land q \rightarrow p) \land (T \rightarrow r)) \rightarrow p \rightarrow s\}, \\
G_2 = ((r \land q \rightarrow p) \land (T \rightarrow r)) \rightarrow p, \quad W_2 = P_1 \quad \text{by rule (2)}, \\
G_3 = p, \quad W_3 = P_1 | P_2, \quad P_2 = (r \land q \rightarrow p) \land (T \rightarrow r) \quad \text{by rule (5)}, \\
G_4 = r \land q, \quad W_4 = P_1 | P_2 \quad \text{by rule (2)}, \\
G_5 = q, \quad W_5 = P_1 | P_2 \quad \text{by rule (3)}, \\
G_6 = T, \quad W_6 = P_1 \quad \text{by rule (2)}, \\
G_7 = r, \quad W_7 = P_1 | P_2 \quad \text{by rule (3)} \quad \text{applied to } (G_4, W_4), \\
G_8 = T, \quad W_8 = P_1 | P_2 \quad \text{by rule (2)}.
$$

If the first clause of $P$ is replaced by $r \rightarrow q$, then the derivation of $G = s$ from $P$ is no longer feasible. In fact, in this case we have: $G_6 = r, \quad W_6 = P_1$, and $P_2$ does not contain the definition of $r$, which is defined in the inner block $P_2$ and, therefore, not visible at this point.

In the next sections we shall present the fixpoint and model-theoretic semantics for our language and we shall prove the equivalence between the operational semantics and the fixpoint semantics and between the fixpoint and the model-theoretic semantics. All these semantics are defined by extending the corresponding standard semantics given for positive Horn clause logic (see [1, 8]).

4. Fixpoint semantics

Given a program $P$, let $U(P)$ be the Herbrand universe for $P$, that is the set of all ground terms that can be formed out of the constant and functional symbols occurring in $P$. In the case $P$ does not contain any constant, we add some constant, say $c$, to form ground terms. Let $B(P)$ be the Herbrand base for $P$, that is, the set of all ground atoms which can be formed by using predicates of $P$ and terms in $U(P)$. An Herbrand interpretation for $P$ is a subset of $B(P)$. The set of all Herbrand interpretations for $P$ (the power set of $B(P)$) is a complete lattice under inclusion, with $B(P)$ as the top element and $\emptyset$ as the bottom element.

In the case of Horn clause logic, the fixpoint semantics of a program $P$ can be obtained by defining a mapping $T_P$ from the lattice of Herbrand interpretations to itself (called the “immediate consequences” transformation) and by proving it to be monotone and continuous; then the semantics of $P$ is the least fixpoint of $T_P$. In the case of the language with open blocks, a program $P$ (i.e., a set of clauses) can be regarded also as a block inside a larger program. In this case, $P$ must be
considered as an open context whose meaning may depend on its external environment, namely on the content of externally nested blocks. As a consequence, we replace the $T_P$ of the standard case with a mapping $T_{P,I}$ where $I$ is an Herbrand interpretation which is intended to convey all the necessary information about the enclosing environment of the program $P$. More precisely, $I$ is the set of the ground atomic formulas on the Herbrand Universe that are derivable from the external environment.

The mapping $T_{P,I}$ is defined as follows:

$$T_{P,I}(X) = I \cup \{A \in B(P): \text{there is a } G \rightarrow A \in [P] \text{ and } X > G\},$$

where $X$ is an Herbrand interpretation for $P$ and $>$ is the weak relation of satisfiability between Herbrand interpretations and closed goals and is defined as follows:

- $X > T_I$
- $X > A$ iff $A \in X$
- $X > G_1 \wedge G_2$ iff $X > G_1$ and $X > G_2$
- $X > \exists x G$ iff $X > [x/t]G$ for some $t \in U(P)$
- $X > D \Rightarrow G$ iff $T_{D,X}^\infty(\emptyset) > G$, where $T_{D,X}^\infty(\emptyset) = \bigcup_{k=1}^{\infty} T_{D,X}^k(\emptyset)$.

$T_{P,I}$ is well-defined although its definition seems to be circular. In fact, $T_{P,I}$ is defined by induction on the number $n$ of nestings of $\Rightarrow$ in $P$. When $n = 0$, the case $G = D \Rightarrow G'$ does not occur in the definition of $>$ and, therefore, $T_{P,I}$ is well-defined for $n = 0$. For $n > 0$, $T_{D,X}^\infty(\emptyset)$ is used in the definition of $T_{P,I}$. However, since $D$ contains at most $n - 1$ levels of nesting of $\Rightarrow$, $T_{D,X}$ is defined. Furthermore, since for every $k \geq 0$, $T_{D,X}^k(\emptyset) \in 2^{B(P)}$, and since $2^{B(P)}$ is a complete lattice, the least upper bound $T_{D,X}^\infty(\emptyset)$ does exist.

It can be proved (see Appendix A) that $T_{P,I}$ is monotone and continuous, and therefore that it has a least fixpoint $\text{lpf}(T_{P,I}) = \bigcup_{k=1}^{\infty} T_{P,I}^k(\emptyset)$ (short). The semantics of a program $P$, in this case, is a mapping from an interpretation $I$ (a subset of $B(P)$) to another interpretation, namely the least fixpoint of $T_{P,I}$, $T_{P,I}^\infty(\emptyset)$.

Notice that the mapping $T_{P,I}$ can clearly be defined in terms of $T_P$ as follows:

$$T_{P,I}(X) = I \cup T_P(X).$$

However, we have the following inequality:

$$T_{P,I}^\infty(\emptyset) \neq I \cup T_P^\infty(\emptyset).$$

We can consider such an interpretation $I$ as an environment which associates with each predicate symbol in the program a denotation, that is, the set of tuples of terms for which the predicate is true. Hence, the semantics of a program $P$ turns out to be defined as a mapping from an environment $I$, consisting of the set of ground atoms true in the external context, to another environment, the least fixpoint of $T_{P,I}$, consisting of the set of atoms $I$ plus the atoms derivable from $I$ using $P$. Thus, there is an immediate parallel with standard programming languages, whose denotational semantics is defined as a mapping between environments.
We shall prove that \( T_{P,0}^\infty(0) \) is the set of all ground atomic formulas operationally derivable from \( P \), and, more generally, by using the relation of weak satisfiability, that

\[
T_{P,0}^\infty(0) > G \iff P \vdash G,
\]

namely, that the fixpoint semantics is equivalent to the operational semantics. To prove this equivalence, we separately prove soundness and completeness of the operational semantics with respect to the fixpoint semantics.

**Theorem 1** (Soundness). Let \( P \) be a program and \( G \) be a closed goal. Then

\[
P \vdash G \Rightarrow T_{P,0}^\infty(0) > G.
\]

**Proof.** If \( P \vdash G \) then there is a derivation \((W_1, G_1), \ldots, (W_n, G_n)\), where \( W_1 = P \), \( G_1 = G \) and \( G_n = T \). We prove by induction on \( k < n \) that if \( W_{n-k} = P_1 \cdots P_m \) for some \( m \), then \( T_{P_{n-k}Y_{n-k}}^\infty(0) > G_{n-k} \), where \( Y_0 = \emptyset \) and \( Y_j = T_{P_{n-j}Y_{n-j}}^\infty(0) \). Thus for \( k = n-1 \) we have that \( T_{P,0}^\infty(0) > G \).

If \( k = 0 \) the thesis holds trivially.

We assume that the thesis holds for \( i < k \) and we prove it for \( i = k \) considering all possible cases for \( G \) (double induction).

- If \( G_{n-k} = T \), the thesis obviously holds.
- If \( G_{n-k} = A \) then, for some \( j \leq m \), there is a \( G \vdash A \in [P_j] \) and, for some \( h < k \), \( G_{n-h} = G \) and \( W_{n-h} = P_1 \cdots P_j \). By inductive hypothesis, \( T_{P_{j}Y_{j}}^\infty(0) > G \). So, by definition of \( T_{P_{j}Y_{j}}^\infty(0) \),

\[
A \in T_{P_{j}Y_{j}}^\infty(0) = T_{P_{j}Y_{j}}^\infty(0).
\]

Moreover, for all \( j \), \( Y_j \subseteq T_{P_{j}Y_{j}}^\infty(0) = Y_{j+1} \) and therefore, since \( j \leq m \),

\[
T_{P_{j}Y_{j}}^\infty(0) \subseteq T_{P_{n}Y_{n}}^\infty(0).
\]

Thus \( A \in T_{P_{n}Y_{n}}^\infty(0) \).

- If \( G_{n-k} = G_1 \wedge G_2 \) then there are two nonnegative integers \( h, j < k \) such that \( G_{n-h} = G_1 \), \( G_{n-j} = G_2 \) and \( W_{n-h} = W_{n-j} = W_{n-k} \). By inductive hypothesis, \( T_{P_{n-h}Y_{n-h}}^\infty(0) > G_1 \) and \( T_{P_{n-j}Y_{n-j}}^\infty(0) > G_2 \). Thus \( T_{P_{n}Y_{n}}^\infty(0) > G_1 \wedge G_2 \).

- If \( G_{n-k} = \exists x G' \) we proceed as in the previous case.

- If \( G_{n-k} = D \supset G' \) then, for some \( j < k \), \( G_{n-j} = G' \) and \( W_{n-j} = P_1 \cdots P_m \supset D \). By inductive hypothesis, \( T_{P_{n-j}Y_{n-j}}(0) > G' \), that is \( Y_m \supset D \supset G' \). Thus, by definition of \( Y_m \), \( T_{P_{n}Y_{n}}^\infty(0) > D \supset G' \). \( \square \)

To prove the completeness of the operational semantics with respect to the fixpoint semantics, we shall make use of the following lemmas. \( f^* \) denotes the set \( \{ T \to A : A \in I \} \), where \( I \) is a subset of \( B(P) \).
**Lemma 1.** Let \( I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots \) be a sequence of Herbrand interpretations. If \( G \) is a goal and \( \bigcup_{n=0}^{\infty} I_n \models G \) then there exists \( k \geq 0 \) such that \( I_k \models G \).

**Lemma 2.** Let \( D_1 \) and \( D_2 \) be programs, \( G \) a closed goal and \( W \) a list of programs (possibly empty). If \( D_1 \subseteq D_2 \) then
\[
D_1 \models W \models G \Rightarrow D_2 \models W \models G.
\]

**Lemma 3.** Let \( P \) be a program, \( G \) a closed goal, \( I \) a subset of \( B(P) \) and \( W \) a list of programs (possibly empty). Then
\[
I^* \cup P \models W \models G \Rightarrow I^* \models P \models W \models G.
\]

**Lemma 4.** Let \( P \) be a program, \( G \) a closed goal. Then
\[
\{ T \rightarrow A; \ P \models A \} \models G \Rightarrow P \models G.
\]

Lemma 1 is proved in Appendix A. Lemmas 2, 3 and 4 can be proved by double induction on the length of the derivation and on the structure of the goal \( G \).

**Theorem 2** (Completeness). Let \( P \) be a program, \( G \) a closed goal, \( I \) a subset of \( B(P) \). Then
\[
T_{P,I}^\omega(\emptyset) \models G \Rightarrow P \cup I^* \models G.
\]

**Proof.** It suffices to prove (by induction on the highest number \( n \) of levels of nesting of \( \Rightarrow \) in \( P \) and \( G \)) that for every \( k \geq 0 \),
\[
T_{P,I}^k(\emptyset) \models G \Rightarrow P \cup I^* \models G. \quad (*)
\]
In fact, if \( T_{P,I}^k(\emptyset) \models G \) then, by Lemma 1, there is a \( k \geq 0 \) such that \( T_{P,I}^k(\emptyset) \models G \). Therefore, we can conclude by (*) that \( P \cup I^* \models G \).

If \( n = 0 \) then there are no occurrences of \( \Rightarrow \) neither in \( P \) nor in \( G \). (*) can be easily proved by double induction on \( k \) and on the structure of \( G \).

Let us consider in detail the case \( n > 0 \). We assume, by inductive hypothesis, that (*) holds for at most \( n-1 \) levels of nesting of \( \Rightarrow \) in \( P \) and \( G \). Again we prove that it holds for \( n \) by double induction on \( k \) and on the structure of \( G \).

(a) Let us assume \( k = 0 \).

- If \( G = T \) then (*) obviously holds.
- If \( G = A \) the \( A \) is not in \( T_{P,I}^0(\emptyset) = \emptyset \).
- If \( G = G_1 \wedge G_2 \) then
\[
T_{P,I}^0(\emptyset) \models G_1 \wedge G_2 \Rightarrow T_{P,I}^0(\emptyset) \models G_1 \text{ and } T_{P,I}^0(\emptyset) \models G_2
\Rightarrow P \cup I^* \models G_1 \text{ and } P \cup I^* \models G_2
\]
(by inductive hypothesis since \( G_1 \) and \( G_2 \) are substructures of \( G \))
\[
\Rightarrow P \cup I^* \models G_1 \wedge G_2.
\]
- If \( G = \exists x G' \) we proceed in a way similar to the previous case.
Extending Horn clause logic with implication goals

If $G = D \supset G'$, then
\[
\begin{align*}
T^0_{P,I}(\emptyset) &> D \supset G' 
\Rightarrow T^\cap_{D,A}(\emptyset) > G', \quad \text{since } T^0_{P,I}(\emptyset) = \emptyset \\
&\Rightarrow T^h_{D,A}(\emptyset) > G', \quad \text{for some } h \geq 0 \text{ (by Lemma 1)} \\
&\Rightarrow D \cup \emptyset \vdash G', \quad \text{(by inductive hypothesis,}

\text{since } D \text{ and } G' \text{ can contain at most } n-1 \text{ levels of nesting of } \supset \\
&\Rightarrow D \vdash G' \\
&\Rightarrow \emptyset \vdash D \supset G' \\
&\Rightarrow P \cup I^* \vdash D \supset G' \quad \text{(by Lemma 2)}.
\end{align*}
\]

(b) For $k > 0$ we assume, by inductive hypothesis, that (*) holds for $k - 1$.

- If $G = T$ or $G = G_1 \land G_2$ or $G = \exists x G'$ we proceed as in the case $k = 0$.
- If $G = A$ then $A \in T^k_{P,I}(\emptyset)$, so either $A \in I$, or there is a $G \rightarrow A \in [P]$ such that $T^k_{P,I}(\emptyset) > G$. In the first case, $T \rightarrow A \in I^* \subseteq P \cup I^*$ and therefore $P \cup I^* \vdash A$. In the second one, since (*) holds for $k - 1$, we have, by inductive hypothesis, that $P \cup I^* \vdash A$.
- If $G = D \supset G'$, then
\[
\begin{align*}
T^k_{P,I}(\emptyset) &> D \supset G' 
\Rightarrow T^\cap_{D,A}(\emptyset) > G', \quad \text{where } X = T^k_{P,I}(\emptyset) \\
&\Rightarrow T^h_{D,A}(\emptyset) > G', \quad \text{for some } h \geq 0 \text{ (by Lemma 1)} \\
&\Rightarrow D \cup X^* \vdash G' \quad \text{(by inductive hypothesis,}

\text{since } D \text{ and } G' \text{ can contain at most } n-1 \text{ levels of nesting of } \supset \\
&\Rightarrow X^* \mid D \vdash G' \quad \text{(by Lemma 3)} \\
&\Rightarrow X^* \vdash D \Rightarrow G' \\
&\Rightarrow (T^k_{P,I}(\emptyset))^* \vdash D \Rightarrow G' \\
&\Rightarrow \{T \rightarrow A: P \cup I^* \vdash A\} \vdash D \Rightarrow G' \\
&\Rightarrow P \cup I^* \vdash D \Rightarrow G' \quad \text{(by Lemma 2, since}

\text{—see step } G = A— \\
A \in T^k_{P,I}(\emptyset) \text{ implies } P \cup I^* \vdash A \\
\text{and hence} \\
T^k_{P,I}(\emptyset) \subseteq \{A: P \cup I^* \vdash A\}) \\
&\Rightarrow P \cup I^* \vdash D \Rightarrow G' \text{(by Lemma 4).}
\end{align*}
\]

From Theorems 1 and 2, for $I = \emptyset$, we have $T^\cap_{P,I}(\emptyset) > G$ iff $P \vdash G'$; that is, the operational semantics is sound and complete with respect to the fixpoint semantics.

As a particular case, for $G = A$

\[
A \in T^\cap_{P,I}(\emptyset) \quad \text{iff } P \vdash A;
\]

that is,

\[
T^\cap_{P,I}(\emptyset) = \{A: P \vdash A\},
\]
As an example, let us consider once again the program $P$ of Example 5. We want to determine whether $G = s$ can be derived from $P$, that is, whether $s \in T_{P,0}(\emptyset)$.

Example 8.

$$
T_{P,0}^1(\emptyset) = \{q\}, \quad T_{P,0}^2(\emptyset) = T_{P,0}(\{q\}) = \{q, s\}, \quad T_{P,0}^3(\emptyset) = T_{P,0}(\{q, s\}) = \{q, s\} = \text{lfp}(T_{P,0}).
$$

Notice that to determine whether $s$ belongs to $T_{P,0}(X)$ it is necessary to determine whether $T_{P,0}^X(\emptyset) > p$ with $D = (q \land r \rightarrow p) \land (T \rightarrow r)$ (fifth rule of the definition of $>$).

For $X = \emptyset$ we have $\text{lfp}(T_{D,0}) = \{r\}$ so that $p$ is not satisfied in $T_{P,0}^1(\emptyset)$. On the contrary, for $X = \{q\}$ we have $\text{lfp}(T_{D,X}) = \{q, r, p\}$ so that $p$ is satisfied in $T_{P,0}^2(\emptyset)$ and $s \in T_{P,0}^3(\emptyset)$.

Finally notice that if we replace the first clause of $P$ with the clause $r \rightarrow q$, the goal $G = s$ is not satisfiable anymore by $P$ since in this case $T_{P,0}^\infty(\emptyset) = \emptyset$.

5. Model-theoretic semantics

In this section we define a model-theoretic semantics for the language with open blocks and we prove soundness and completeness of the fixpoint semantics with respect to the model-theoretic semantics.

We now want to define the satisfiability of a formula $\alpha$ in a given Herbrand interpretation $I$ (i.e., $I \models \alpha$), where $\alpha$ can be either a goal formula or a clause. Satisfiability can be defined as usual in classical logic, the only exception being the definition of satisfiability for implication goals. To understand the intuitive meaning of the semantics for implication goals, let us start considering the case of closed blocks. With closed blocks, an implication $D \supset G$ has the natural meaning “$G$ is a logical consequence of $D$”. Therefore, the definition of satisfiability for the implication goal is rather obvious,

$$
I \models D \supset G \iff \text{for all interpretations } I', I' \models D \Rightarrow I' \models G;
$$

that is, the satisfiability of $D \supset G$ does not depend on the interpretation $I$. In the case of open blocks, on the contrary, the content of the interpretation $I$ is not negligible and we can think of the interpretation $I$ as giving information about what is true in the external environment, that is, in the enclosing blocks. Since what is true in the external environment still remains true when entering a new block, we verify the satisfiability of an implication goal $D \supset G$ by considering all the interpretations which are supersets of $I$ and testing in each of them whether $G$ is true when $D$ is true.

Let us define more formally the model-theoretic semantics for open blocks. Let $\alpha$ be a closed formula of the language defined in Section 3, that is, a goal or a
definite clause. Given an Herbrand interpretation $I$ for $\alpha$ we define $I \models \alpha$ ($I$ satisfies $\alpha$) by induction on the structure of $\alpha$ as follows:

- $I \models T$,
- $I \models A$ if $A \in I$,
- $I \models G_1 \land G_2$ iff $I \models G_1$ and $I \models G_2$,
- $I \models \exists x G$ iff $I \models [x/t]G$ for some $t \in U(\alpha)$,
- $I \models D \supset G$ iff, for all interpretations $I'$, $(I \subseteq I'$ and $I' \models D) \Rightarrow I' \models G$,
- $I \models G \rightarrow A$ iff $I \models G \supset I \models A$,
- $I \models \forall x D$ iff $I \models [x/t]D$ for all $t \in U(\alpha)$,
- $I \models D_1 \land D_2$ iff $I \models D_1$ and $I \models D_2$.

Let $P$ be a program and $I$ an Herbrand interpretation for $P$. $I$ satisfies $P(I \models P)$, that is, $I$ is a model for $P$ if $I$ satisfies all clauses in $P$. Let us denote by $M(P)$ the set of all the Herbrand models of $P$. A closed goal formula $G$ is a logical consequence of $P$ ($P \models G$) iff, for all interpretations $I$ of $P$, $I \models P \Rightarrow I \models G$. Notice that, from the definition of $[P]$ given in Section 3, it follows that $I \models P$ iff for all $G \rightarrow A \in [P]$, $I \models G \rightarrow A$.

It must be noticed that the two different implications $\rightarrow$ and $\supset$ have been given different semantics. The implication $\rightarrow$ is the classical one, while the implication $\supset$ has a semantics similar to that of the implication of intuitionistic logic. Our model-theoretic semantics is, nevertheless, simpler than Kripke semantics for intuitionistic logic (see Section 6), since an Herbrand interpretation is defined to be a subset of the Herbrand base as in classical logic; we do not need to introduce the notion of worlds as in Kripke interpretations. From another point of view, as we shall see in the next section, this semantics can be considered as a Kripke semantics in which only a subset of the Kripke interpretations has to be taken into account. As a result, for every program $P$, there exists a least Herbrand model of $P$ and this gives us the possibility to prove the equivalence between the model-theoretic and fixpoint semantics in the same way as it has been done for Horn clause logic in [1].

**Example 9.** Let us consider again the program $P$ of Example 5:

$$P = \{q, \}
\begin{array}{c}
((r \land q \rightarrow p) \land r) \supset p \rightarrow s.
\end{array}$$

$I = \{q, s\}$ is a model of $P$, since $I$ satisfies both of the two clauses in $P$. In fact, $I \models q$ and, since $I \models s$, then $I \models (D \supset p) \rightarrow s$, where $D = ((r \land q \rightarrow p) \land r)$. Moreover, it can be proved that $I$ is a minimal model of $P$, since neither $I_1 = \emptyset$ nor $I_2 = \{q\}$ nor $I_3 = \{s\}$ are models of $P$. In fact, $I_1$ and $I_3$ do not satisfy the first clause of $P$, while $I_2$ does not satisfy the second one. To prove this, notice that $I_2 \not\models s$, but for every superset $I'$ of $I_2$, if $I' \models D$ then $I' \models p$. This obviously holds when $p \in I'$; the other possible cases are the following ones:

$$I' = \{q, s\} \not\models D, \quad I' = \{q, r\} \not\models D, \quad I' = \{q, s, r\} \not\models D.$$
Hence, \( I \) is the minimal model of \( P \) since \( P \) has a minimal model by the intersection property below. Therefore, \( G = s \) is a logical consequence of \( P \). On the contrary, \( I = \emptyset \) is a model of the program \( P \) of Example 4. In fact,

\[
\emptyset \vdash r \rightarrow q, \quad \emptyset \vdash ((q \rightarrow p) \land r) \rightarrow p \rightarrow s.
\]

Indeed, \( \emptyset \not\models ((q \rightarrow p) \land r) \rightarrow p \) since there is a superset \( \{r\} \) of \( \emptyset \) such that \( \{r\} \models (q \rightarrow r) \land r \) but \( \{r\} \not\models p \). Therefore \( I = \emptyset \models P \) but \( I \not\models s \), so that \( s \) is not a logical consequence of \( P \).

To prove the equivalence between fixpoint and model-theoretic semantics, we establish some lemmas first.

**Lemma 5.** Let \( P \) be a program, \( G \) a closed goal and \( I_1 \) and \( I_2 \) two Herbrand interpretations for \( P \). If \( I_1 \cap I_2 \models G \), then \( I_1 \models G \) and \( I_2 \models G \).

**Proof.** By induction on the structure of \( G \).

- If \( G = T \), it is obvious.
- If \( G = A \), then
  \[
  I_1 \cap I_2 \models A \Rightarrow A \in I_1 \cap I_2
  \Rightarrow A \in I_1 \quad \text{and} \quad A \in I_2
  \Rightarrow I_1 \models A \quad \text{and} \quad I_2 \models A.
  \]
- If \( G = G_1 \land G_2 \), then
  \[
  I_1 \cap I_2 \models G_1 \land G_2 \Rightarrow I_1 \cap I_2 \models G_1 \quad \text{and} \quad I_1 \cap I_2 \models G_2
  \Rightarrow I_1 \models G_1, I_2 \models G_1, I_1 \models G_2, I_2 \models G_2 \quad \text{(by ind. hypothesis)}
  \Rightarrow I_1 \models G_1 \land G_2 \quad \text{and} \quad I_2 \models G_1 \land G_2.
  \]
- If \( G = \exists x G' \), then
  \[
  I_1 \cap I_2 \models \exists x G' \Rightarrow I_1 \cap I_2 \models [t/x]G' \quad \text{for some closed term} \ t
  \Rightarrow I_1 \models [t/x]G' \quad \text{and} \quad I_2 \models [t/x]G' \quad \text{(by ind. hypothesis)}
  \Rightarrow I_1 \models \exists x G' \quad \text{and} \quad I_2 \models \exists x G'.
  \]
- If \( G = D \supset G' \), then
  \[
  I_1 \cap I_2 \models D \supset G' \Rightarrow \text{for all} \ I' \ (I_1 \cap I_2 \subseteq I' \quad \text{and} \quad I' \models D) \Rightarrow I' \models G'.
  \]
  Since \( I_1 \cap I_2 \subseteq I_1 \), we have that for all \( I' \)
  \[
  (I_1 \subseteq I' \quad \text{and} \quad I' \models D) \Rightarrow I' \models G',
  \]
i.e., \( I_1 \models D \supset G' \). Similarly, \( I_2 \models D \supset G' \).

**Lemma 6** (Model intersection property). Let \( P \) be a program and \( I_1 \) and \( I_2 \) two Herbrand interpretations for \( P \). If \( I_1 \) and \( I_2 \) are models of \( P \), then \( I_1 \cap I_2 \) is a model of \( P \).
Proof. It can be proved by induction on \( D \) that, for every clause \( D \) in \( P \), \( I_1 \models D \) and \( I_2 \models D \) imply \( I_1 \cap I_2 \models D \).

- If \( D = G \rightarrow A \), then
  \[
  I_1 \models G \rightarrow A \text{ and } I_2 \models G \rightarrow A \implies I_1 \models G \implies I_1 \models A \text{ and } I_2 \models G \implies I_2 \models A.
  \]

If we assume that \( I_1 \cap I_2 \models G \), we have, by Lemma 5, \( I_1 \models G \) and \( I_2 \models G \). Thus \( I_1 \models A \) and \( I_2 \models A \), that is, \( I_1 \cap I_2 \models A \). Therefore, \( I_1 \cap I_2 \models G \rightarrow A \).

- If \( D = \forall x D' \) then
  \[
  I_1 \models \forall x D' \text{ and } I_2 \models \forall x D' \implies I_1 \models [x/t]D' \text{ and } I_2 \models [x/t]D' \text{ for all } t \in B(P)
  \]
  \[
  \implies I_1 \cap I_2 \models \forall x D'.
  \]

As a consequence of Lemma 6 we have that the intersection \( \bigcap M(P) \) of all Herbrand models of \( P \) is a model of \( P \), namely the least Herbrand model of \( P \).

**Lemma 7.** \( \bigcap M(P) = \{ A : P \models A \} \).

**Proof.**

\( P \models A \) iff for all \( I, I \models P \implies I \models A \)

iff for all \( I, I \models P \implies A \in I \)

iff for all \( I, I \in M(P) \implies A \in I \), since \( M(P) = \{ I : I \models P \} \)

iff \( A \in \bigcap M(P) \) \hspace{1cm} \( \square \)

We now prove soundness and completeness of the fixpoint semantics with respect to the model-theoretic semantics.

**Theorem 3** (Soundness and completeness). Let \( P \) be a program, \( G \) a closed goal and \( I \) a subset of \( B(P) \) (remember that \( I^* = \{ T \rightarrow A, A \in I \} \)). Then

\[
T_{P, I}^\infty(\emptyset) > G \iff \bigcap M(P \cup I^*) = G.
\]
Proof. Let $I$ and $X$ be two Herbrand interpretations. We shall prove that

\[ X \succ G \iff X \models G \]  

(1)

and

\[ I \subseteq X \quad \text{and} \quad X \in M(P) \iff T_{P,I}(X) \subseteq X. \]  

(2)

From (2) it is easy to prove that

\[ T_{P,I}(\emptyset) = \bigcap \{ M(P \cup I*) \}. \]  

(3)

In fact,

\[
T_{P,I}(\emptyset) = \bigcap \{ X : T_{P,I}(X) \subseteq X \}
= \bigcap \{ X : X \in M(P) \text{ and } I \subseteq X \} \quad \text{by (3)},
= \bigcap \{ X : X \in M(P) \text{ and } X \models I^* \}
= \bigcap \{ X : X \models P \cup I^* \}
= \bigcap M(P \cup I^*).
\]

From (1) and (3) the thesis can be immediately derived.

Let us prove (1) and (2) by induction on the highest number $n$ of levels of nesting of $\Rightarrow$ in $P$ and $G$.

(a) If $n = 0$ then there are no occurrences of $\Rightarrow$ neither in $P$ nor in $G$. (1) can be easily proved by induction on the structure of $G$.

- If $G = T$, it is obvious.
- If $G = A$, $X \succ A$ iff $A \in X$ iff $X \models A$.
- If $G = G_1 \& G_2$,

\[ X \succ G_1 \& G_2 \iff X \succ G_1 \text{ and } X \succ G_2 \]

iff $X \models G_1$ and $X \models G_2$ (by inductive hypothesis)

iff $X \models G_1 \& G_2$.

- If $G = \exists x G'$, the proof is similar.

The case $G = D \Rightarrow G'$ does not occur, since $n = 0$.

We shall prove that (2) holds for $n = 0$.

(From left to right): Let us assume that $I \subseteq X$ and $X \in M(P)$. We want to show that $T_{P,I}(X) \subseteq X$. If $A \in T_{P,I}(X)$ then either $A \in I$ and then $A \in X$, or there is a $G \Rightarrow A \in [P]$ such that $X \succ G$. $G$ does not contain any occurrence of $\Rightarrow$, therefore $X \succ G$ implies $X \models G$. Since, in addition, $X \in M(P)$, that is, for all $G \Rightarrow A \in [P]$, $X \models G$ implies $X \models A$, we have that $X \models A$. Thus $A \in X$.

(From right to left): Let us assume that $T_{P,I}(X) \subseteq X$. We want to prove that $I \subseteq X$ and $X \in M(P)$.

\[ T_{P,I}(X) \subseteq X \Rightarrow \text{for all } A, A \in T_{P,I}(X) \text{ implies } A \in X \]

\[ \Rightarrow \text{for all } A, (A \in I \text{ or there is } G \Rightarrow A \in [P] \text{ such that } X \succ G) \]

implies $A \in X$. 

L. Giordano, A. Martelli, G. Rossi
Thus for all $A$ in $I$, $A \in I$ implies $A \in X$, that is, $I \subseteq X$; and if there exists a $G \rightarrow A \in [P]$ such that $X \models G$ then $A \in X$. Since $X \models G$ if $X \models G$, we have that, for all $G \rightarrow A \in [P]$, if $X \models G$ then $A \in X$, that is, $X \in M(P)$. Thus we have proved (2).

(b) If $n > 0$ we assume, by inductive hypothesis, that (1) and (2) (and thus the thesis, by the argument above) hold for at most $n - 1$ levels of nesting of $\rightarrow$ in $P$ and $G$. We prove that (1) and (2) hold for $n$ levels of nesting. Again, (1) is proved by induction on the structure of $G$.

- If $G = T$ or $G = A$ or $G = G_1 \land G_2$ or $G = \exists x G'$ we proceed as in the case $n = 0$.
- If $G = D \Rightarrow G'$,

$$X \Rightarrow D \Rightarrow G' \iff \bigcap M(D \cup X^*) \models G' \iff \text{for all } I, I \in M(D \cup X^*) \Rightarrow I \models G'$$

$$\iff \text{for all } I, I \models D \cup X^* \Rightarrow I \models G'$$

$$\iff \text{for all } I, X \subseteq I \text{ and } I \models D \Rightarrow I \models G'$$

$$\iff X \models D \Rightarrow G'.$$

(2) is proved as for $n = 0$ by showing that (3) holds. In doing this, we use the fact that each formula in $P$ contains at most $n$ levels of nesting of $\Rightarrow$ and that (1) holds for $n$.

From Theorem 3, for $I = \emptyset$, we have

$$T_{p=0}^\infty(\emptyset) > G \iff \bigcap M(P) \models G.$$  

Since

$$\bigcap M(P) \models G \iff \text{for all } I, I \in M(P) \Rightarrow I \models G$$

(by Lemma 5 and since $\bigcap M(P) \in M(P)$)

$$\iff \text{for all } I, I \models P \Rightarrow I \models G$$

$$\iff P \models G,$$

the following relation holds:

$$T_{p=0}^\infty(\emptyset) > G \iff P \models G,$$

that is, the fixpoint semantics is sound and complete with respect to the model-theoretic semantics.
6. An alternative model-theoretic semantics based on Kripke interpretations

In the last section we have defined the model-theoretic semantics of our language by employing Herbrand interpretations defined as subsets of the Herbrand base. We shall now define another model-theoretic semantics for the language, by making use of Kripke interpretations. Again we consider only interpretations defined on the Herbrand universe. The satisfiability relation is defined in the same way as in positive intuitionistic logic, with an extension due to the presence of the two different kinds of implication.

Let \( \alpha \) be a closed formula, that is a goal or a definite clause. A Kripke interpretation \( M \) for \( \alpha \) is a triple \( \langle W, \subseteq, I_0 \rangle \), where \( W \subseteq 2^{B(\alpha)} \) is a partially ordered set of worlds and \( I_0 \in W \) is a world of \( M \) such that \( I_0 \subseteq I \), for any world \( I \) of \( M \) (i.e. \( I_0 \) is the least world). We define the satisfiability relation between an interpretation \( M \) and a formula \( \alpha \) in a given world \( I \) of \( M \) by induction on the structure of \( \alpha \), as follows:

- \( M \models_I, T \),
- \( M \models_I, A \iff A \in I \),
- \( M \models_I, G_1 \land G_2 \iff M \models_I, G_1 \) and \( M \models_I, G_2 \),
- \( M \models_I, \exists xG \iff M \models_I, [x/t]G \) for some \( t \in U(\alpha) \),
- \( M \models_I, D \supset G \iff \), for each world \( I' \) of \( M \), \( (I \subseteq I' \) and \( M \models_{I'}, D \impliedby M \models_{I'}, G \),
- \( M \models_I, G \rightarrow A \iff M \models_I, G \impliedby M \models_I, A \),
- \( M \models_I, \forall xD \iff M \models_I, [x/t]D \) for all \( t \in U(\alpha) \),
- \( M \models_I, D_1 \land D_2 \iff M \models_I, D_1 \) and \( M \models_I, D_2 \).

An interpretation \( M = (W, \subseteq, I_0) \) satisfies a formula \( \alpha \) iff \( M \models_k, \alpha \). Let \( P \) be a program and \( M \) a Kripke interpretation for \( P \). \( M \) satisfies \( P \), if \( M \) satisfies all the clauses in \( P \). Let \( G \) be a closed goal formula. \( G \) is a logical consequence of \( P(P \models' G) \) iff, for all Kripke interpretations \( M \) for \( P \), \( M \models_k, P \impliedby M \models_k, G \) (we use a prime to distinguish between logical consequence in the two different model-theoretic semantics).

Notice that if we restrict the language to the propositional case and have a unique implication symbol (used both in goals and in clauses) with the semantics of \( \supset \), this semantics is the same (with a change of notation) as that presented in [4], which is the semantics of intuitionistic logic. On the other hand, if we restrict the language by eliminating blocks and so the implication \( \supset \), we have clearly a semantics for classical logic (only the least interpretation in a world is used). At a semantic level we can therefore consider \( \supset \) to be the intuitionistic implication, while \( \rightarrow \) is the classical one. However, if the two implications are considered altogether, the resulting semantics differs from that of both intuitionistic and classical logic. Indeed, the very weak logical equivalence

\[
D_1 \supset (D_2 \supset G) = (D_1 \land D_2 \supset G)
\]

which is held by intuitionistic and classical logic, is not satisfied by \( \supset \) (more precisely, the equivalence is satisfied if clauses in \( D_1 \) and \( D_2 \) are all of the form \( T \rightarrow A \), but not in the general case).
This model-theoretic semantics is not equivalent to that of the previous section in the general case. For example,

\[ a \supset b \models b \quad \text{and} \quad a \supset b \not\models b, \]

In fact, every interpretation \( I \) that satisfies \( a \supset b \) must satisfy \( b \) too, because, otherwise, there is an interpretation \( I' = I \cup \{a\} \) reachable from \( I \) which satisfies \( a \) but not \( b \) (against the definition of satisfiability for an implication goal). On the other hand, there are Kripke interpretations which satisfy \( a \supset b \) in their initial world but do not satisfy \( b \), such as, for instance, the interpretation \( M = \langle \{I_1, I_0\}, \preceq, I_0 \rangle \), where \( I_0 = \emptyset \) and \( I_1 = \{b\} \). Nevertheless, if we restrict ourselves to determine whether a goal is a logical consequence of a program \textit{in our language}, the two semantics are equivalent. In fact, it can be proved that, given a program \( P \) and a closed goal \( G \),

\[ P \models' G \iff P \models G. \quad (**) \]

In the example above this restriction is not satisfied since \( a \supset b \) is not a program in our language, so the equivalence (**) does not hold in this case.

To prove (**) we establish the following lemmas first.

Lemma 8. Let \( \alpha \) be a formula (that is, a goal or a definite clause), \( I_0 \) a subset of \( B(\alpha) \) and \( M \) the Kripke interpretation \( (W, \preceq, I_0) \), where \( W = \{I' : I' \in 2^{B(\alpha)} \} \) and \( I_0 \subseteq I' \). Then, for all \( I \in W \), \( I \models \alpha \iff M \models \alpha \).

\textbf{Proof.} By induction on the structure of the formula \( \alpha \).

- If \( \alpha = T \), it is obvious.
- If \( \alpha = A \), then \( I \models A \iff A \in I \iff M \models A \).
- If \( \alpha = G_1 \land G_2 \), then

\[ I \models G_1 \land G_2 \iff I \models G_1 \text{ and } I \models G_2 \]

\[ \iff M \models G_1 \text{ and } M \models G_2 \text{ (by inductive hypothesis)} \]

\[ \iff M \models G_1 \land G_2. \]

- If \( \alpha = \exists x G \) or \( \alpha = D_1 \land D_2 \) or \( \alpha = \forall x D \) we proceed as in the previous case.
- If \( \alpha = D \supset G \), we have to prove that, for all \( I' \) (\( I' \subseteq I' \) and \( I' \models D \))\( \Rightarrow I' \models G \) iff, for all worlds \( I' \) of \( M \), \( (I \subseteq I' \text{ and } M \models_I D) \Rightarrow M \models_I G \).

From left to right, for any \( I' \in W \) such that \( I \subseteq I' \),

\[ M \models_I D \Rightarrow I' \models D \quad \text{(by inductive hypothesis since } I' \in W) \]

\[ \Rightarrow I' \models G \quad \text{(since the left part of the equivalence holds)} \]

\[ \Rightarrow M \models_I G \quad \text{(by inductive hypothesis).} \]

From right to left, for any \( I' \) such that \( I \subseteq I' \) (clearly, \( I' \in W \) since \( I_0 \subseteq I \subseteq I' \)),

\[ I' \models D \Rightarrow M \models_I D \quad \text{(by inductive hypothesis)} \]

\[ \Rightarrow M \models_I G \quad \text{(since the right part of the equivalence holds)} \]

\[ \Rightarrow I' \models G \quad \text{(by inductive hypothesis).} \]
• If $\alpha = D \rightarrow G$, we have to prove that $I \models D \Rightarrow I \models G$ iff $M \models D \Rightarrow M \models i \cdot G$.

From left to right,

$M \models i \cdot D \Rightarrow I \models D$ (by inductive hypothesis)  
$\Rightarrow I \models G$ (since the left part of the equivalence holds)  
$\Rightarrow M \models i \cdot G$ (by inductive hypothesis).

From right to left,

$I \models D \Rightarrow M \models i \cdot D$ (by inductive hypothesis)  
$\Rightarrow M \models i \cdot G$ (since the right part of the equivalence holds)  
$\Rightarrow I \models G$ (by inductive hypothesis).  

Lemma 9. Let $D$ be a definite clause, $G$ a closed goal (we assume that $G$ contains only nonlogical symbols that occur in $D$), $I_0$ and $I$ subsets of $B(D)$ and $M$ the Kripke interpretation $(W, \subseteq, I_0)$, with $W \subseteq 2^{B(D)}$ and $I \in W$. Then

1. $I \models G \Rightarrow M \models i \cdot G$,
2. $I \not\models D \Rightarrow M \not\models i \cdot D$.

Proof. By induction on the number $n$ of levels of nesting of $\Rightarrow$ in $D$ and $G$.

(a) If $n = 0$ we prove (1) by structural induction on $G$ and (2) by structural induction on $D$. To prove (1), we proceed as follows:

• If $G = T$, it is obvious.
• If $G = A$, then $I \models A \Rightarrow A \in I \Rightarrow M \models i \cdot A$.
• If $G = G_1 \land G_2$, then

$I \models G_1 \land G_2 \Rightarrow I \models G_1$ and $I \models G_2$  
$\Rightarrow M \models i \cdot G_1$ and $M \models i \cdot G_2$ (by inductive hypothesis)  
$\Rightarrow M \models i \cdot G_1 \land G_2$.

• If $G = \exists x G'$, we proceed as in the previous case.

To prove (2), we will prove (by contraposition) $M \models i \cdot D \Rightarrow I \not\models D$.

• If $D = G \rightarrow A$, by assuming $M \models i \cdot G \Rightarrow A \in I \models i \cdot A$, that is $M \models i \cdot G \Rightarrow M \models i \cdot A$, we have

$I \models G \Rightarrow M \models i \cdot G$ (since (1) holds for $n = 0$)  
$\Rightarrow M \models i \cdot A$ (by the previous assumption)  
$\Rightarrow A \in I$  
$\Rightarrow I \models A$,

thus $I \not\models G \rightarrow A$.  

Extending Horn clause logic with implication goals

- If $D = D_1 \land D_2$ or $D = \forall x D'$, it is obvious by induction.

(b) Now we consider the case $n > 0$. We assume, by inductive hypothesis, that (1) and (2) hold for at most $n-1$ levels of nesting of $\supset$ in $G$ and $D$. First we shall prove that (1) holds by induction on the structure of $G$.

- If $G = T$ or $G = A$ or $G = G_1 \land G_2$ or $G = \exists x G'$, the proof is the same as for $n = 0$.

- If $G = D \supset G'$, then by assuming $I \models D \supset G'$ (that is, for all $I'$ such that $I \subseteq I'$, $I' \models D \supset I' \models G'$), we have, for all $I' \in W$ such that $I \subseteq I'$,

$$M \models _I D \Rightarrow I' \models D \quad \text{(by inductive hypothesis)}$$

$$\Rightarrow I' \models G' \quad \text{(by the previous assumption)}$$

$$\Rightarrow M \models _I G' \quad \text{(by inductive hypothesis),}$$

that is, $M \models _I D \supset G'$.

To prove (2) we prove (by contraposition) $M \models _I D \Rightarrow I \models D$, by induction on the structure of $D$.

- If $D = D_1 \land D_2$ or $D = \forall x D'$, the proof is the same as in the case $n = 0$.

- If $D = G \supset A$, by assuming $M \models _I G \supset A$, that is $M \models _I G \supset M \models _I A$, we have

$$I \models G \Rightarrow M \models _I G \quad \text{(since (1) holds for $n$)}$$

$$\Rightarrow M \models _I A \quad \text{(by the previous assumption)}$$

$$\Rightarrow A \in I$$

$$\Rightarrow I \models A,$$

thus $I \models G \supset A$. □

Since a program is a set of closed definite clauses, if $I \not\models P$ then there is a clause $D$ such that $I \not\models D$ and, by Lemma 9, we have that $M \not\models D$. Thus $I \not\models P$. Therefore, the relation (2) of Lemma 5 holds for any program $P$, not only for any definite clause $D$.

**Theorem 4.** Let $P$ be a program and $G$ be a goal. Then $P \models ^* G$ iff $P \models G$.

**Proof.** (From left to right): By hypothesis, for all the interpretations $M$, $M \models _{I_0} P \Rightarrow M \models _{I_0} G$. We want to prove that for all the interpretations $I$, $I \models P \Rightarrow I \models G$. Let $I$ be an interpretation such that $I \models P$. By Lemma 8, $M \models _I P$, where $M$ is the interpretation $(W, \subseteq, I)$ and $W = \{I': I' \in 2^{I_0} \text{ and } I \subseteq I'\}$. Thus, by hypothesis, $M \models _I G$ and, again by Lemma 8, $I \models G$.

(From right to left): By hypothesis, for all the interpretations $I$, $I \models P \Rightarrow I \models G$. We want to prove that, for all the interpretations $M$, $M \models _{I_0} P \Rightarrow M \models _{I_0} G$. Given an interpretation $M = (W, \subseteq, I_0)$, there are two possible cases for $I_0$: either $I_0 \models G$ or $I_0 \not\models P$. If $I_0 \models G$ then, by Lemma 9, $M \models _{I_0} G$; if $I_0 \not\models P$, then, by Lemma 9, $M \not\models _{I_0} P$. Thus we have that either $M \models _{I_0} G$ or $M \not\models _{I_0} P$, namely $M \models _{I_0} P \Rightarrow M \models _{I_0} G$. □
7. Towards a concrete implementation

The definition of the operational semantics given in Section 3 is very simple since, given a program $P$, it introduces the set $[P]$ of all ground instances of the clauses in $P$ and does not involve the notions of substitution, unification and variable renaming. We shall now present a less abstract operational semantics for the language with open blocks, which is clearly equivalent to the previous one and is defined using substitutions, unification and variable renaming.

Let $P_1 \cdots P_n$ be a nonempty list of programs, let $G$ be a closed goal and let $\theta$ be a substitution. We define derivability of $G$ from the list $P_1 \cdots P_n$, with substitution $\theta$ by induction on the structure of $G$ in the following way:

1. $P_1 \cdots P_n \vdash T$ with substitution $I$ (identity substitution);
2. if $A$ is a closed atomic formula, $P_1 \cdots P_n \vdash A$ with substitution $\theta$ iff for some $i \leq n$ there is a formula $\forall x_1 \cdots \forall x_k G \Rightarrow B \in P_i$ such that $\mu = \text{mgu}(A, B')$ and $P_i \mu \vdash P_i \mu \vdash \mu$ with substitution $\phi$ and $\theta = \mu \phi$, where $G' \Rightarrow B'$ is the clause obtained by renaming the universally quantified variables $x_1, \ldots, x_k$;
3. $P_1 \cdots P_n \vdash G_1 \land G_2$ with substitution $\theta$ iff $P_1 \cdots P_n \vdash G_1$, with substitution $\phi$ and $P_n \phi \vdash P_n \phi \vdash G_2 \phi$ with substitution $\mu$ and $\theta = \phi \mu$;
4. $P_1 \cdots P_n \vdash \exists x G$ with substitution $\theta$ iff $P_1 \cdots P_n \vdash \exists x G'$ with substitution $\theta$, where $G'$ is obtained from $G$ by renaming $x$;
5. $P_1 \cdots P_n \vdash D \supset G$ with substitution $\theta$ iff $P_1 \cdots P_n \vdash D \supset G$ with substitution $\theta$.

Notice that free variables can occur both in a goal and in programs of the list $P_1 \cdots P_n$. For this reason in rules (2) and (3) we apply substitutions not only to goals, but also to the programs in the list. Since the initial program $P_1$ is a set of closed clauses and the initial goal is a closed formula, they do not contain free variables. Free variables can be introduced into a goal by renaming the existentially quantified variables associated with the goal itself (rule (4)); free variables can be introduced in the list of programs by rule (5) whenever there is some free variable occurring in the set $D$ of a block goal $D \supset G$. Existential variables are renamed once, as soon as the existential quantifier is dropped by rule (4), whereas universal variables of a clause are renamed every time the clause is selected to resolve an atomic goal (rule (2)); notice that the free variables which possibly occur in the clause are not renamed.

Rule (3) is defined in such a way to preserve the sharing of variables between $G_1, G_2$ and the programs in the list and to prevent from an improper use of the free variables in the programs. For example, given the list of programs $L = \{q(a), q(b)\} \{q(x) \rightarrow p(x)\}$ where $x$ is a free variable, the goal $G = p(a) \land p(b)$ is not operationally derivable from $L$. Instead, the goals $p(a)$ and $p(b)$ are individually operationally derivable from that list.

The above operational semantics provides a rather abstract interpreter for the proposed language. More concrete interpreters can be obtained by applying a sequence of transformation steps to it, using, for instance, the methodology described in [9]. In that paper it is shown how to transform a nondeterministic recursive
extending Horn clause logic with implication goals

Interpretation for a logic language into a deterministic iterative one based on the data structures commonly used by Prolog interpreters and compilers. It is also shown that the extensions proposed in this paper can be dealt with efficiently by means of the well-known techniques used to implement blocks with static scope rules in conventional programming languages.

8. Conclusions and related work

In this paper we have presented an extension to positive Horn clause logic obtained by introducing implication in goals as a tool for structuring programs. Many of the results of this paper were first presented in the shorter paper [5]. The idea to add the standard programming language concept of block to logic programming languages was already proposed in [9], mainly from the implementation viewpoint. In this paper the concept of block has been defined more formally.

We have mentioned throughout this paper Miller's proposal for introducing modules in logic programming [11, 12]. Our approach is very similar to that of Miller, since the locality of clause definitions needed to define modules has been achieved essentially by introducing implication in goals. Since the implication goal has in our case a different semantics, our language results to be statically scoped instead of dynamically scoped as that of Miller. In [13] however, it is shown how a form of lexical scoping can be provided by using universal quantification on goals.

The proposal by Monteiro and Porto [14] of a "contextual logic programming" is very close to our proposal as far as the semantics chosen for the implication goal is concerned. In fact, they define a module (called a unit) as a set of context dependent predicate definitions, that is, in the terminology introduced above, as an open block. Moreover, like in our proposal, the external context of a unit is structured as a list of units with visibility rules for clauses similar to those of our language. The most remarkable difference of their proposal with respect to ours is that names can be associated with units so to have a module facility and therefore the same unit name can be used in different contexts. Moreover, the structure of a context in which a goal is proved (that is the set of clauses that can be used for its refutation) depends on the dynamic sequence of context extension calls preceding the goal.

The proposals mentioned above were put forward with the main purpose of adding a module facility to logic programming languages. However we believe that open blocks are more suitable for programming in the small, whereas modules should define closed environments or, anyway, interact with the external environment in a very limited and disciplined way through a well specified interface. Thus we are presently investigating extensions to logic programming with module constructs based on closed blocks [6, 7]. Besides adding an implication operator for proving a goal within a module, we have introduced another operator to give names to modules, which allows us to define parametric modules. We believe that the two extensions of Horn clause logic, namely (open) block constructs and module
constructs can be usefully integrated, providing a language suitable both for programming in the small and for programming in the large.

Appendix A

To prove that the mapping $T_{P,I}$ is monotone and continuous, we shall make use of the following lemmas. Let $I_1$, $I_2$ and $X$ be Herbrand interpretations.

Lemma 10. If $I_1 \subseteq I_2$, then $T_{P,I_1}(X) \subseteq T_{P,I_2}(X)$, for all $X$.

Proof. If $A \in T_{P,I_1}(X)$ then either $A \in I_1 \subseteq I_2$ and then $A \in T_{P,I_2}(X)$, or there is $G \Rightarrow A[P]$ such that $X \supseteq G$ and then $A \in T_{P,I_2}(X)$. □

As a consequence of this proposition we have that, for $I_1 \subseteq I_2$

$$\{X: T_{P,I_1}(X) \subseteq X\} \supseteq \{X: T_{P,I_2}(X) \subseteq X\}$$

and then

$$\bigcap\{X: T_{P,I_1}(X) \subseteq X\} \subseteq \bigcap\{X: T_{P,I_2}(X) \subseteq X\};$$

that is,

$$T_{P,I_1}^\infty(\emptyset) \subseteq T_{P,I_2}^\infty(\emptyset).$$

Lemma 11. If $I_1 \subseteq I_2$, then $T_{P,I_1}^\infty(\emptyset) \supseteq G \Rightarrow T_{P,I_2}^\infty(\emptyset) \supseteq G$.

Proof. By structural induction on $G$.

• If $G = T$, the proposition obviously holds.

• If $G = A$, then $A \in T_{P,I_i}(X) \subseteq T_{P,I_2}^\infty(\emptyset)$ (by Lemma 10).

• If $G = G_1 \land G_2$ or $G = \exists x G'$, the proposition holds trivially by induction.

• If $G = D \supseteq G'$, then

$$T_{P,I_1}^\infty(\emptyset) \supseteq D \supseteq G' \Rightarrow T_{D,X_1}^\infty(\emptyset) \supseteq G', \quad \text{with } X_1 = T_{P,I_1}^\infty(\emptyset)$$

$$\Rightarrow T_{D,X_2}^\infty(\emptyset) \supseteq G', \quad \text{with } X_2 = T_{P,I_2}^\infty(\emptyset)$$

(by inductive hypothesis since, as a consequence of Lemma 10, $X_1 \subseteq X_2$)

$$\Rightarrow T_{P,I_2}^\infty(\emptyset) \supseteq D \supseteq G'.$$ □

Lemma 12. If $X_1 \subseteq X_2$ then $X_1 > G \Rightarrow X_2 > G$.

Proof. By structural induction on $G$.

• If $G = T$, obvious.

• If $G = A$ then

$$X_1 > A \Rightarrow A \subseteq X_1 \subseteq X_2$$

$$\Rightarrow X_2 > A.$$  

• If $G = G_1 \land G_2$ or $G = \exists x G'$, the thesis obviously holds by inductive hypothesis.
If $G = D \supset G'$

\[ X_1 \supset D \supset G' \Rightarrow T_{D,X_1}^{\supset}(\emptyset) > G' \]

\[ \Rightarrow T_{D,X_2}^{\supset}(\emptyset) > G' \quad \text{(by Lemma 11)} \]

\[ \Rightarrow X_2 \supset D \supset G'. \]

**Theorem 5.** $T_{P,1}$ is monotone; that is, if $X_1 \subseteq X_2$ then $T_{P,1}(X_1) \subseteq T_{P,1}(X_2)$

**Proof.** Let $A \in T_{P,1}(X_1)$. If $A \in I$ then $A \in T_{P,1}(X_2)$, since $I \subseteq T_{P,1}(X_2)$. Otherwise there is a $G \supset A \in [P]$ and $X_1 \supset G$. By Lemma 12 we have that $X_2 \supset G$, so $A \in T_{P,1}(X_2)$.

**Lemma 1** (already stated in Section 4). Let $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ be a sequence of Herbrand interpretations. If $G$ is a goal and $\bigcup_{j=0}^{\infty} I_j \supset G$ then there exists a $k \geq 0$ such that $I_k \supset G$.

In order to prove Lemma 1, we shall first prove the following Lemma 13. Let us consider the following definitions:

\[ J_0(I) = \bigcup_{j=0}^{\infty} (I_j), \]

\[ J_i(I)(k_1, \ldots, k_i, D_1, \ldots, D_i) = T_{D_1, \ldots, D_i}^{(k_1, \ldots, k_i)}(I_1)(k_1-1, \ldots, k_i-1, D_1, \ldots, D_i)(\emptyset); \]

\[ J_i^0(I) = I_k, \]

\[ J_i^h(I)(k_1, \ldots, k_i, D_1, \ldots, D_i) = T_{D_1, \ldots, D_i}^{(k_1, \ldots, k_i)}(I_1)(k_1-1, \ldots, k_i-1, D_1, \ldots, D_i)(\emptyset). \]

**Lemma 13.** Let $G$ be a goal, $i$ a positive integer, $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ a sequence of Herbrand interpretations (we shall denote it by $\{J_i\}$) and $k_1, \ldots, k_i$ and $D_1, \ldots, D_i$ two tuples (of length $i$) of positive integers and of programs respectively. If $J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) \supset G$ then there exists a $k \geq 0$ such that $J_i^h(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) \supset G$.

**Proof.** By induction on the number $n$ of levels of nesting of $\supset$ in $G$ and $D_i$.

(a) If $n = 0$, we shall prove the thesis by double induction on $k_i$ and on the structure of $G$ ($G = D \supset G'$ does not occur in this case).

(a1) Let us consider the case $k_i = 0$.

- If $G = T$, obvious.

- If $G = A$ then

\[ A \in J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) = T_{D_1, \ldots, D_i}^{(k_1, \ldots, k_i)}(I_1)(k_1-1, \ldots, k_i-1, D_1, \ldots, D_i)(\emptyset) - \emptyset. \]

- If $G = G_1 \land G_2$ then

\[ J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) \supset G_1 \land G_2 \]

\[ \Rightarrow J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_1 \]

and $J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_2$

\[ \Rightarrow J_i^h(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_1 \]

and $J_i^h(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_2$ for some $l, h \geq 0$.
(by inductive hypothesis, since \( k_i \) does no change and \( G_1 \) and \( G_2 \) are substructures of \( G \)). Let \( k \) be the maximum between \( l \) and \( h \). By a generalization of Lemma 10, we have

\[
J^k_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_1 \quad \text{and} \quad J^k_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_2
\]

and therefore \( J^k_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > G_1 \land G_2 \).

(a2) For \( k_i > 0 \), if \( G = T \) or \( G = G_1 \land G_2 \) or \( G = \exists x G' \), the proof is the same as for \( k_i = 0 \).

If \( G = A \), then we prove that

\[
A \in J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i) \Rightarrow A \in J^k_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i)
\]

by induction on \( i \).

If \( i = 0 \) then

\[
A \in J_0(\{I_i\}) = \bigcup_{j=0}^{\infty} I_j \Rightarrow A \in I_k = J^k_0(\{I_i\}) \quad \text{for some } k \geq 0.
\]

If \( i > 0 \), then

\[
A \in J_i(\{I_i\})(k_1, \ldots, k_i, D_1, \ldots, D_i)
\]

\[
= T^k_{D_0, D_{l-1}}(J_i(\{I_i\})(k_1, \ldots, k_{l-1}, D_1, \ldots, D_{l-1}, \emptyset))
\]

\[
= T^k_{D_0, D_{l-1}}(J_{l-1}(\{I_i\})(k_1, \ldots, k_{l-1}, D_1, \ldots, D_{l-1}, \emptyset))
\]

There are two cases:

1. \( A \in J_{l-1}(\{I_i\})(k_1, \ldots, k_{l-1}, D_1, \ldots, D_{l-1}) \)

\[\Rightarrow A \in J^k_{l-1}(\{I_i\})(k_1, \ldots, k_{l-1}, D_1, \ldots, D_{l-1}) \quad \text{for some } k \geq 0\]

(by the inductive hypothesis on \( i \))

\[\Rightarrow A \in J^k_l(\{I_i\})(k_1, \ldots, k_l, D_1, \ldots, D_l) \quad \text{for some } k \geq 0.\]

2. There is a \( G' \rightarrow A \in [D_i] \) such that \( T^{k-1}_{D_0, D_{l-1}}(J_{l-1}(\{I_i\})(k_1, \ldots, k_{l-1}, D_1, \ldots, D_{l-1}, \emptyset)) > G' \)

\[\Rightarrow \text{there is a } G' \rightarrow A \in [D_i] \text{ such that } J_i(\{I_i\})(k_1, \ldots, k_{l-1}, k_i - 1, D_1, \ldots, D_i) > G'\]

\[\Rightarrow \text{there is a } G' \rightarrow A \in [D_i] \text{ such that } J^k_i(\{I_i\})(k_1, \ldots, k_{l-1}, k_i - 1, D_1, \ldots, D_i) > G'\]

for some \( k \geq 0 \) (since by inductive hypothesis the thesis holds for \( k_i - 1 \))

\[\Rightarrow A \in J^k_l(\{I_i\})(k_1, \ldots, k_l, D_1, \ldots, D_l) \quad \text{for some } k \geq 0.\]
(b) Now, let us consider the case \( n > 0 \). We assume, by inductive hypothesis, that the thesis holds for at most \( n - 1 \) levels of nesting of \( \supset \) in \( G \) and \( D \). Again we prove that the thesis holds by double induction on \( k \) and on the structure of \( G \).

(b1) Let \( k_0 = 0 \).

- If \( G = T \) or \( G = A \) or \( G = G_1 \land G_2 \) or \( G = \exists x G' \), the proof is the same as for \( n = 0 \).
- If \( G = D \supset G' \) then

\[
J_i(\{I_j\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > D \supset G'
\]

\[
\Rightarrow T_{D_jJ_i(\{I_j\})(k_1, \ldots, k_i, D_1, \ldots, D_i, \emptyset)}(\emptyset) > G'
\]

\[
\Rightarrow J^h_{i}(\{T_{D_jJ_i(\{I_j\})(k_1, \ldots, k_i, D_1, \ldots, D_i, \emptyset)}(\emptyset)\}) > G'
\]

for some \( h \geq 0 \) (by inductive hypothesis, since \( G' \) contains at most \( n - 1 \) levels of nesting of \( \supset \)).

\[
\Rightarrow T^{h}_{D_jJ_i(\{I_j\})(k_1, \ldots, k_i, D_1, \ldots, D_i, \emptyset)}(\emptyset) > G'
\]

for some \( h \geq 0 \) (by definition of \( J^h_i \)).

\[
J_{i+1}(\{I_j\})(k_1, \ldots, k_i, h, D_1, \ldots, D_i, D) > G'
\]

for some \( h \geq 0 \) (by inductive hypothesis, since \( D \) contain at most \( n - 1 \) levels of nesting of \( \supset \)).

\[
\Rightarrow T^{h}_{D_jJ_i(\{I_j\})(k_1, \ldots, k_i, D_1, \ldots, D_i, \emptyset)}(\emptyset) > G'
\]

for some \( h, k \geq 0 \) (by Lemma 12, since \( T^h \subseteq T^\infty \)).

\[
J^k_{i}(\{I_j\})(k_1, \ldots, k_i, D_1, \ldots, D_i) > D \supset G'.
\]

(b2) For \( k_0 > 0 \), if \( G = T \) or \( G = A \) or \( G = G_1 \land G_2 \) or \( G = \exists x G' \), the proof is the same as for \( n - 0 \). The case \( G = D \supset G' \) is the same as for \( k_i = 0 \).

The proof of Lemma 1 now follows immediately from Lemma 13, if we take \( i = 0 \).

**Theorem 6.** \( T_{P,1} \) is continuous; that is, if \( I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots \) is a sequence of interpretations, then

\[
\bigcup_{i=0}^{\infty} T_{P,1}(I_i) = T_{P,1} \left( \bigcup_{i=0}^{\infty} I_i \right).
\]

**Proof.** To prove this equality, we prove the inclusion in the two direction.

For all \( j \) we have \( I_0 \subseteq \bigcup_{i=0}^{\infty} I_i \), so, by the monotonicity of \( T_{P,1} \),

\[
T_{P,1}(I_j) \subseteq T_{P,1} \left( \bigcup_{i=0}^{\infty} I_i \right),
\]

and thus

\[
\bigcup_{i=0}^{\infty} T_{P,1}(I_i) \subseteq T_{P,1} \left( \bigcup_{i=0}^{\infty} I_i \right).
\]
If $A \in T_{P,I}(\bigcup_{i=0}^{\infty} I_i)$ then either $A \in I$ or not. If $A \in I$ then $A \in \bigcup_{i=0}^{\infty} T_{P,I}(I_i)$, since $I \subseteq T_{P,I}(I_i)$ for all $i > 0$. Otherwise, there exists a $G = A \in [P]$ and $\bigcup_{i=0}^{\infty} I_i > G$. By Lemma 1 there is a $k \geq 0$ such that $I_k > G$. Thus $A \in T_{P,I}(I_k) \subseteq \bigcup_{i=0}^{\infty} T_{P,I}(I_i)$. □

References