Semantics of normal logic programs with embedded implications

Fernando Orejas[1] Edelmira Pasarella[1,2] Elvira Pino[1]

Abstract

The aim of our work is the definition of a model-theoretic semantics of normal logic programs with embedded implications. We first propose a quite simple operational semantics for this class of programs whose negation mechanism is the constructive negation. This semantics is used to prove the adequacy of the model-theoretic semantics. Then we define a declarative semantics for this class of programs in terms of Beth models and show that in the model class associated to every program there is a least model that can be seen as the semantics of the program, which may be built upwards as the least fixpoint of a continuous immediate consequence operator. Finally, it is proved that the operational semantics is sound and complete with respect to the least fixpoint semantics.

1 Introduction

There are two main approaches to decompose large logic programs into manageable units [2]. On the one hand, different kinds of modular units, similar to the module notions existing in other programming paradigms, have been proposed and studied together with the corresponding composition operations. This kind of modularization can be considered "external" to the logic programming paradigm, since it is based on the use of constructions which are, to a certain extent, independent of any programming formalism [10]. Conversely, the second approach provides a structuring mechanism in terms of a logical connective. In particular, this approach originated in the work of Miller [9]. The idea is that intuitionistic implication may be used to structure a logic program into blocks, as it is done in imperative programming languages. For more details on the use of this connective the reader may look, for instance, at [2].

Providing semantics to normal logic programming units is, in general, a difficult task, due to the non-monotonic nature of negation that hinders the definition of a compositional semantics. Nevertheless, a certain amount of work has been done

[1]Dpto de L.S.I., Universidad Politècnica de Catalunya,Campus Nord, Mòdul C6, Jordi Girona 1-3, 08034 Barcelona, Spain, email: {pino,orejas}@lsi.upc.es

[2]Dpto de Computación y Tecnología de la Información, Universidad Simón Bolívar, Aptdo Postal 89000, Caracas 1089, Venezuela, email: edelmira@lsi.upc.es

for defining the semantics of normal logic program modules according to the first approach mentioned above (see, e.g., [6, 4]). This is not quite true for the second approach. To our knowledge, only [1] and [7] consider this case. The reason for this lack of work is, probably, the apparent difficulty in mixing two semantically very different connectives such as negation and intuitionistic implication. Intuitionistic connectives seem to ask for intuitionistic models, like Kripke or Beth models, while negation seems to ask for some kind of 3-valued models. The semantics presented in [1] and [7] is defined in terms of some sort of Kripke models. The results presented in [1] are very restrictive. Only the case of negation as failure is considered, programs must be stratified and signatures may only contain predicate symbols, i.e. function symbols are not allowed. The work presented in [7] also deals with negation as failure, but the other restrictions are not present. In addition, we found the model-theoretic semantics quite ad-hoc. The kind of Kripke models used are not really intuitionistic: first, the interpretation associated to a given world is a three-valued structure and, second and more importantly, the order relation between worlds is not monotonic, in contrast with the intuition underlying intuitionistic Kripke models. This means that if an atom is satisfied by the interpretation associated to a given world, then it may not be satisfied by the interpretation associated to a greater world.

In [8] we defined a new declarative compositional semantics for a general class of normal logic program units, in terms of a class of models that we called ranked. As we pointed out in that paper, ranked models are, intuitively, quite close to Beth models. This lead us to think that both connectives could have a natural and reasonably simple semantics in terms of intuitionistic (Beth) models. Moreover, this semantics would make more explicit the intuitionistic nature of negation in logic programming already pointed out by other authors (e.g. [12]). In this paper we follow these ideas. We first propose a quite simple operational semantics for this class of programs whose negation mechanism is the constructive negation [3, 5, 13]. This semantics is used to prove the adequacy of the model-theoretic semantics. Then we define a declarative semantics for this class of programs in terms of Beth models and show that in the model class associated to every program there is a least model that can be seen as the semantics of the program, which may be built upwards as the least fixpoint of a continuous immediate consequence operator. Finally, it is proved that the operational semantics is sound and complete with respect to the least fixpoint semantics.

This paper is organized as follows. In the following section we present some basic concepts and notation used in the paper. Section 3 introduces the operational semantics of our programs. Section 4 defines the Beth models used in the paper and their associated forcing relation. The next section introduces the immediate consequence operator showing that it is monotonic and continuous. Section 6, defines an ordering relation on the class of models of a program and shows that the least model coincides with the least fixpoint of the immediate consequence operator. Finally, in Section 7, the operational semantics is proved to be sound and complete with respect to the least model semantics.

2 Preliminaries

A *countable signature* Σ consists of a pair of sets (FS_Σ, PS_Σ) of function and predicate symbols with some associated arity. Terms, atoms or first order formulas built by using functions and/or predicates from Σ and, also, variables from a fixed countable set X of variable symbols are called Σ-terms, Σ-atoms and Σ-formulas, respectively. The Herbrand universe, denoted by H_Σ, is the set of all ground Σ-terms constructed by using Σ-functions. Terms are denoted by $t, s, \ldots$, predicate symbols by $p, q, \ldots$ and function symbols by $f, g, \ldots$. Letters a, b denote atoms and the character ℓ is used for literals. A formula whose subterms are variables is called a *flat* formula. $\varphi^\forall$ and $\varphi^\exists$ are the universal and existential closure of φ, respectively. The logical constants are denoted by $\underline{\mathtt{t}}$ and $\underline{\mathtt{f}}$. Programs are denoted by using the letters P and Q. In general, subscripts and superscripts will be used if needed and a bar is used to denote (finite) sequences of objects. *Normal logic programs with embedded implications* over a signature Σ are finite sets of clauses $a :- G_1, \ldots, G_k$, where a is a Σ-atom and $\forall i \in \{1, \ldots, k\}$, G_i is an *intuitionistic Σ-literal*, that is, either a Σ-literal, b or $\neg b$, where b is a Σ-atom; or an *intuitionistic Σ-expression*, $P_i \supset G_i'$, where P_i is a Σ-program and G_i' is an intuitionistic Σ-literal. The idea behind this kind of goals is that, when evaluating G_i', one may use the definitions in P_i as auxiliary local definitions (in addition to other global definitions in the given program). Clauses whose head is empty correspond to goals of this class of language, called *intuitionistic Σ-goals*.

Free variables in a clause are assumed to be implicitly quantified universally. This means that the scope of a variable is the clause where it is defined. In particular, given the goal $\{p(x).\} \supset p(f(x))$, x in $p(x)$ is not considered to be bound to x in $p(f(x))$ and, as a consequence, the goal should succeed.

We consider that clauses are written following the structure of constraint normal clauses with flat head. That is, $p(t_1, \ldots, t_n) :- G_1, \ldots, G_k$ is written as the constrained clause $p(x_1, \ldots, x_n) :- G_1, \ldots, G_k \square x_1 = t_1, \ldots, x_n = t_n$

Moreover, we suppose that the identical tuple $x_1, \ldots, x_n$ of *fresh variables* occurs in all clauses (in a program) with predicate p in its heads. Also, just to simplify, clauses of the form $a :- \square\underline{\mathtt{t}}$ are written as a.

The *set of definitions of a predicate* p in a program P is defined as usual:

$$Def(P, p) \equiv \{p(\overline{x}) :- \overline{G}^k \square c^k \in P\}$$

Constraints occurring in programs are *equality Σ-constraints*, that is, arbitrary first order Σ-formulas in which the only relational symbol occurring in atoms is the equality (formulas composing equality atoms with the connectives $\neg, \wedge, \vee, \rightarrow$, and the quantifiers $\forall, \exists$). Constraints are denoted by using the letters c and d (possibly with sub or super-scripts). We will handle constraints in a logical way, using logical consequence of the *free equality theory*, FET_Σ (see, e.g., [12]).

A constraint c is *satisfiable* (resp. unsatisfiable) if, and only if, $FET_\Sigma \models c^\exists$ (resp. $FET_\Sigma \models \neg(c^\exists)$); a constraint d is *less general* than c if, and only if, $FET_\Sigma \models (d \rightarrow c)^\forall$. A ground substitution $\overline{x} = \overline{t}$ (where t_i are closed terms) is called a *solution of a constraint* c if, and only if, $FET_\Sigma \models (\overline{x} = \overline{t} \rightarrow c)^\forall$.

A constrained Σ-atom is a pair $p(\overline{x})\square c(\overline{x})$, where $p \in PS_\Sigma$ and $c(\overline{x})$ is a satisfiable Σ-constraint. The set of all the constrained Σ-atoms is denoted $\mathcal{L}_\Sigma(X)$.

3 Operational semantics

In this section we introduce an operational semantics for the class of normal logic programs with embedded implication. This semantics is presented in terms of a derivation relation over sequents of the form $P \vdash \overline{G}\square c$, where P is a Σ-program and $:- \overline{G}\square c$ is a Σ-goal. It may be noted that our semantics is very simple, but not very useful for practical purposes, since it is too non-deterministic to be directly implemented. Its main aim is to show the adequacy of the model-theoretic semantics defined below. In particular, our treatment of negation can be seen as a more abstract and simple variation of [3, 5, 13]. Instead, we could have introduced an operational semantics closer to implementation. For instance, a variation of the BCN semantics [11]. However, the proofs for the main results of this paper would have been slightly more complex.

The following mutually recursive definitions establish our semantics.

Definition 1 *Let P be a Σ-program and $:- \overline{G}\square c$ a Σ-goal. $P \vdash \overline{G}\square c$ can be proved with computed answer c' if and only if there exists a finite derivation of $P \vdash \overline{G}\square c$, that is, $P \vdash \overline{G}\square c \overset{n}{\rightsquigarrow} P \vdash \square c'$, $n \geq 0$, $FET_\Sigma \models c'^{\exists}$ where $\overset{n}{\rightsquigarrow}$ corresponds to n applications (derivation steps) of the relation $\rightsquigarrow$ over sequents.*

Definition 2 *The derivation relation $\rightsquigarrow$ over sequents is defined as follows:*

1. $P \vdash \overline{G}_1, p(\overline{x}), \overline{G}_2\square c \rightsquigarrow P \vdash \overline{G}_1, \overline{G}, \overline{G}_2\square c \wedge d$ *if there exists a (renamed apart) clause $p(\overline{x}) :- \overline{G}\square d \in Def(P,p)$ and $FET_\Sigma \models (c \wedge d)^{\exists}$*
2. $P \vdash \overline{G}_1, \neg p(\overline{x}), \overline{G}_2\square c \rightsquigarrow P \vdash \overline{G}_1, \overline{G}_2\square c'$ *if for every (renamed apart) clause $p(\overline{x}) :- G_1, \ldots, G_m\square d$ there exists $J \subseteq \{1, \ldots, m\}$ such that $\forall j \in J : P \vdash \neg G_j\square d$ can be proved with computed answer d_j and $FET_\Sigma \models (c' \rightarrow \neg d \vee \bigvee_{j \in J} d_j)^{\forall}$.*
3. $P \vdash \overline{G}_1, Q \supset G, \overline{G}_2\square c \rightsquigarrow P \vdash \overline{G}_1, \overline{G}_2\square c'$ *if $P \cup Q \vdash G\square c$ can be proved with computed answer c'.*
4. $P \vdash \overline{G}_1, \neg(Q \supset G), \overline{G}_2\square c \rightsquigarrow P \vdash \overline{G}_1, \overline{G}_2\square c'$ *if $P \cup Q \vdash \neg G\square c$ can be proved with computed answer c'.*

We assume that whenever a constrained Σ-atom $\neg\neg a\square c$ occurs in the right part of a sequent, it denotes $a\square c$.

Example 3 *Given the programs $P = \{p(x) :- p(x)\square x = a, q(x) :- \square x = a\}$ and $Q = \{r :- p(x), \neg q(x)\}$*

$P \vdash Q \supset \neg r \rightsquigarrow P \vdash \square\underline{\mathtt{t}}$ because (def. 2.3):
$P \cup Q \vdash \neg r \rightsquigarrow P \cup Q \vdash \square\underline{\mathtt{t}}$ (def. 2.2)
$P \cup Q \vdash \neg p(x) \rightsquigarrow P \cup Q \vdash \square x \neq a$ (def. 2.2) and
$P \cup Q \vdash q(x) \rightsquigarrow P \cup Q \vdash \square x = a$ (def. 2.1)
and $FET_\Sigma \models (\underline{\mathtt{t}} \rightarrow x \neq a \vee x = a)^{\forall}$

4 Model theory semantics

In this section, we introduce a class of Beth models, and an associated forcing relation, to define the semantics of our programs. Beth models and Kripke models are based on a similar intuition [14]. Both kinds of models are defined as a family of logical structures, where each structure, (corresponding to a *world*) denotes the amount of knowledge one has at a certain moment. Worlds are (partially) ordered, where the ordering relation denotes the increase of knowledge. In these models a *forcing* relation plays the role of satisfaction. Forcing is defined for each world and defines what one can expect to be true in the given world. The key difference between Kripke and Beth models is in the definition of the forcing relation. In Kripke models an atomic formula is forced in a world w if it is satisfied by the associated structure. In Beth models, an atomic formula is forced in w if we may be sure that it will be satisfied in the future. This is formalized saying that the formula is satisfied by all the structures in a bar for that world. Where a bar for w is a set of worlds such that any increasing sequence of worlds starting in w would contain a world in the bar.

In our case, worlds are pairs (P, L), where P is a program and L is a set of constrained atoms. The structure associated to a world is also represented (as a variation of Herbrand structures) as a set of constrained atoms. The intuition is that for a given world (P, L), assuming that the given program includes all the clauses in P, we know that the atoms in L are false and the ones in the associated structure are true. Worlds can be seen as stages in computation, where additional computation provides additional knowledge. The idea is that the atoms in L, for a world (P, L), should be supported by the given program and by the knowledge in the previous worlds. In this context, forcing should be defined like for Beth models: an atomic formula is forced in a world if it will hold in the future.

Example 4 *To illustrate the ideas above, given* $P = \{p :- \neg q, q :- \neg r, s :- \neg p\}$, *a model for* P *may include, for instance, the worlds:* $(\emptyset, \emptyset), (\emptyset, \{r\}), (\emptyset, \{p, r\})$ *with the associated structures* $(\emptyset), (\{q\}), (\{q, s\})$ *In this model, the world* $(\emptyset, \{r\})$ *together with its associated structure* $\{q\}$ *represent that, at certain stage, we may know that* q *is true but* r *is not. The fact that the program in these worlds is empty means that we may have this knowledge without assuming that we have additional clauses (others than the ones in* P*). However, we may consider that the atom* s *is forced in this world, because it holds in the following (larger) world.*

Definition 5 *A* Σ*-world* w *is a pair* (P_w, L_w) *where* P_w *is a* Σ*-program up to variable renaming and* $L_w \subseteq \mathcal{L}_\Sigma(X)$. *The set of all* Σ*-worlds is denoted* W_Σ. *A* Σ*-structure is a 3-tuple* $\mathcal{B} = (W, \preceq, I)$, *where* $W \subseteq W_\Sigma$ *and*

1. *For every* Σ*-program* P, $(P, \emptyset) \in W$
2. $\preceq$ *is a partial order on* W, *such that* $\forall v, w \in W : v \preceq w$ *if, and only if,* $P_v = P_w$ *and* $L_v \subseteq L_w$. *The strict order associated to* $\preceq$ *is denoted* $\prec$.
3. *The interpretation function* $I : W \to 2^{\mathcal{L}_\Sigma(X)}$ *satisfies the following properties:*
   (a) $\forall v \in W$, $\underline{\mathtt{t}}\Box c \in I(v)$ *if* $FET_\Sigma \models c^\exists$

(b) (Monotonicity) $\forall v, w \in W$, *if* $v \preceq w$ *then* $I(v) \subseteq I(w)$

The collection of the all Σ-structures defined in this way is denoted $Struct(\Sigma)$. In addition, we consider that, $\forall w \in W$, the sets of constrained atoms L_w and $I(w)$ are closed under renaming of variables, disjunction of constraints and less general constraints. Also, $\mathcal{B}(P) = \langle W(P), \preceq, I(P)\rangle$ corresponds to the ordered structure associated to P occurring in $\mathcal{B}$ such that $W(P) = \{w \in W \mid (P, \emptyset) \preceq w\}$ and $I(P) = \{I(w) \mid w \in W(P)\}$.

For simplicity, we will assume the following notational conventions: $\ell \square c \in (I(w), L_w)$ means $a \square c \in I(w)$ if $\ell = a$, and $a \square c \in L_w$ if $\ell = \neg a$. Conversely, we write $\neg \ell \square c \in (I(w), L_w)$ to denote $a \square c \in L_w$ if $\ell = a$, and $a \square c \in I(w)$ if $\ell = \neg a$.

Definition 6 (forcing) *Let* $\mathcal{B} = (W, \preceq, I)$ *be a* Σ*-structure. We say that* $B \subseteq W$ *is a bar w.r.t. a world* $v \in W$ *iff for each* $\preceq$*-increasing chain of worlds* $v_0, v_1, \ldots$ *in* W *such that* $v_0 = v$, *there exists* $k \geq 0$ *and* $w \in B$ *such that* $v_k \preceq w \preceq v_{k+1}$. *The bar* B *is strict iff* $\forall v, w \in B$, $v \not\preceq w$ *and* $w \not\preceq v$. *Then, the forcing relation* $\Vdash$, *on a* Σ*-structure* $\mathcal{B}$ *is inductively defined for every world as follows. Let* $v \in W$*:*

1. *For all satisfiable constraints* c *and* d: $v, \mathcal{B} \Vdash c \square d$, *iff* $I(v) \models (c \wedge d)^{\exists}$.
2. $v, \mathcal{B} \Vdash \ell \square c$, *iff there exists a bar* $B \subseteq W$ *with respect to* v *such that for all* $w \in B : \ell \square c \in (I_{\mathcal{B}}(w), L_w)$.
3. $v, \mathcal{B} \Vdash \overline{G}_1, \overline{G}_2 \square c$ *iff* $v, \mathcal{B} \Vdash \overline{G}_1 \square c$, $v, \mathcal{B} \Vdash \overline{G}_2 \square c$.
4. $v, \mathcal{B} \Vdash \neg(G_1, \ldots, G_m) \square c$ *iff* $v, \mathcal{B} \Vdash \neg G_j \square c_j$ *for some* $j \in J \subseteq \{1, \ldots, m\}$ *and* $FET_\Sigma \models (c \rightarrow \bigvee_{j \in J} c_j)^{\forall}$.
5. $v, \mathcal{B} \Vdash P \supset G \square c$ *iff there exists a satisfiable constraint* c', $FET_\Sigma \models (c' \rightarrow c)^{\forall}$ *such that* $(P_v \cup P, \emptyset), \mathcal{B} \Vdash G \square c'$
6. $v, \mathcal{B} \Vdash \neg(P \supset G \square c)$ *iff there exists a satisfiable constraint* c', $FET_\Sigma \models (c' \rightarrow c)^{\forall}$ *such that* $(P_v \cup P, \emptyset), \mathcal{B} \Vdash \neg G \square c'$
7. $v, \mathcal{B} \Vdash p(\overline{x}) :- \overline{G} \square d$ *iff* $\forall w \succeq v$ *if* $w, \mathcal{B} \Vdash \overline{G} \square d$ *then* $w, \mathcal{B} \Vdash p(\overline{x}) \square d$.

A program P can be seen as an intuitionistic theory. As a consequence, one could just define the class of models defined by P as the subclass of all the structures such that P is forced by the world $(\emptyset, \emptyset)$ (or, perhaps, by the world $(P, \emptyset)$). However, this is not satisfactory for our purposes. Many models in that class would not agree with the computational interpretation of our models discussed above. According to the definition below, a structure is a model of a program P if two conditions are satisfied. The first one is that the structure associated to a world (P', L) should satisfy all the consequences that could be computed from the clauses in P and in P' and the negative information in L. The second condition states that the negative information, L, in a world (P', L) must be supported by the clauses in P and in P' and the information included in previous worlds.

Now, in order to formalize these intuitions we will define a notion of *local forcing*, which can be seen as a kind of local satisfaction on a given world. There are two

key ideas in this definition. The first one is to consider that a positive literal ℓ is locally forced in a world w if ℓ is in the interpretation of w, and a negative literal ℓ is locally forced in $w = (P, L)$ if ℓ is in L. The second idea is to consider that a formula $P' \supset \ell$ is locally forced in a world $w = (P, L)$ if ℓ is locally forced in a bar for the world $(P \cup P', \emptyset)$ consisting of worlds $(P \cup P', L')$ where L' is included in L. This means that in L we have enough negative information to compute ℓ. The extension to other kind of formulas is the obvious one.

Definition 7 (Local forcing) *The local forcing relation, $\Vdash_l$, on a Σ-structure $\mathcal{B} = (W, \preceq, I)$ is inductively defined for every world as follows. Let $v \in W$, then:*

1. $v, \mathcal{B} \Vdash_l \ell \Box c$ *iff* $\ell \Box c \in (I(v), L_v)$.
2. $v, \mathcal{B} \Vdash_l P^1 \supset \ldots \supset P^n \supset \ell \Box c$ *iff there exists a satisfiable constraint* c', $FET_\Sigma \models (c' \rightarrow c)^\forall$ *and there exists a bar* B *w.r.t.* $(P_v \cup P^1 \cup \ldots \cup P^n, \emptyset)$ *such that* $\forall v' \in B,\ L_{v'} \subseteq L_v$ *and* $\ell \Box c' \in (I(v'), L_{v'})$.
3. $v, \mathcal{B} \Vdash_l \neg(P^1 \supset \ldots \supset P^n \supset \ell) \Box c$ *iff* $v, \mathcal{B} \Vdash_l P^1 \supset \ldots \supset P^n \supset \neg\ell \Box c$.
4. $v, \mathcal{B} \Vdash_l \overline{G}_1, \overline{G}_2 \Box c$ *iff* $v, \mathcal{B} \Vdash_l \overline{G}_1 \Box c$, $v, \mathcal{B} \Vdash_l \overline{G}_2 \Box c$.
5. $v, \mathcal{B} \Vdash_l p(\overline{x}) :- \overline{G} \Box d$ *iff* $\forall w \succeq v$ *if* $w, \mathcal{B} \Vdash_l \overline{G} \Box d$ *then* $w, \mathcal{B} \Vdash_l p(\overline{x}) \Box d$.

The relation $\Vdash_l$ is included in $\Vdash$, i.e. if $v, \mathcal{B} \Vdash_l \overline{G} \Box c$ then $v, \mathcal{B} \Vdash \overline{G} \Box c$.

Definition 8 (Models) *$\mathcal{B} \in Struct(\Sigma)$ is a model of P, written $\mathcal{B} \models_d P$, if, and only if, $\forall w \in W_\mathcal{B}$, the following two conditions hold:*

1. $\forall p(\overline{x}) :- \overline{G} \Box d \in P \cup P_w : w, \mathcal{B} \Vdash_l p(\overline{x}) :- \overline{G} \Box d$
2. *Supported worlds: if* $p \Box c \in L_w$ *then* $\forall p(\overline{x}) :- G_1, \ldots, G_m \Box d \in Def(P \cup P_w, p)$, *there exist satisfiable constraints* $\{d_j\}_{j \in J}$, $J \subseteq \{1, \ldots, m\}$ *such that* $\forall j \in J$ $\exists v \in W_\mathcal{B}$, $v \prec w$ *with* $v, \mathcal{B} \Vdash_l \neg G_j \Box d_j$ *and* $FET_\Sigma \models (c \rightarrow \neg d \vee \bigvee_{j \in J} d_j)^\forall$.

For every program P, we define $Mod(P)$ as the class of all its models.

Example 9 *Consider the program $P = \{r :- \{p :- \neg q\} \supset s,\ s :- p\}$. A model of P could include any of the structures $\mathcal{B}_1$ or $\mathcal{B}_2$ described below, where the sets at the right hand side of the worlds in $\mathcal{B}_1$ or $\mathcal{B}_2$ denote their interpretation.*

$$\mathcal{B}_1 = \left\{ \begin{array}{c} (\emptyset, \{p, q, s\})\{r\} \\ | \\ (\emptyset, \{p, q\})\{r\} \\ | \\ (\emptyset, \emptyset)\emptyset \end{array} \qquad \begin{array}{c} (\{p :- \neg q\}, \{q\})\{p, r, s\} \\ | \\ (\{p :- \neg q\}, \emptyset)\emptyset \end{array} \right\} \quad \mathcal{B}_2 = \left\{ \begin{array}{c} (\emptyset, \{q\})\{p, s, r\} \\ | \\ (\emptyset, \emptyset)\{p, s, r\} \end{array} \qquad (\{p :- \neg q\}, \emptyset)\{p, q, r, s\} \right\}$$

5 Least fixpoint semantics

In this section, we define an immediate consequence operator T_P that can be used, as usual, to build (bottom-up) a least fixpoint of the operator, which is shown to be a model of the given program P. This fixpoint will be shown to be the least model in $Mod(P)$, with respect to an ordering that will be defined in the following section. Moreover, the operational semantics defined above will be shown to be sound and complete with respect to that model. However, T_P is not defined on all Σ-structures as one would expect, since the class $Struct(\Sigma)$ includes some structures that are not constructive. Instead, T_P is defined on the subclass of *Noetherian* Σ-structures, which is enough for our purposes. In particular, we show that our operator is monotonic and continuous for this subclass.

Although the definition of T_P may look complex, the intuition is quite simple. Given a Σ-structure $\mathcal{A}$, $T_P(\mathcal{A})$, on one hand, for every world (P', L) in $\mathcal{A}$, T_P builds a new world, and the corresponding interpretation, where all the immediate negative consequences (w.r.t. the information we have at this point and the clauses in P and P') are added to L and all the immediate positive consequences are added to its interpretation. On the other hand, additional worlds can be added including the negative information that is supported by the existing worlds in $\mathcal{A}$. The interpretation of these additional worlds just includes the positive information that we have at this point.

Definition 10 *For all $\mathcal{A} \in Struct(\Sigma)$, $T_P(\mathcal{A}) = \langle W_{T_P(\mathcal{A})}, \preceq, I_{T_P(\mathcal{A})} \rangle$ is the structure in $Struct(\Sigma)$ defined as follows for every Σ-program P':*

1. *$W_{T_P(\mathcal{A})}(P') = \{t_{P'}(v) \mid v \in W_{\mathcal{A}}(P')\} \cup \{Succ_{P'}(v) \mid v$ is maximal in $W_{\mathcal{A}}(P')\}$ where $t_{P'}(v) = (P', L_{t_{P'}(v)})$ and $Succ_{P'}(v) = (P', L_{Succ_{P'}(v)})$ are defined as follows for every $v \in W_{\mathcal{A}}$,*

$$L_{t_{P'}(v)} = \{p \square c \mid \text{for all } p(\overline{x}) :- G_1, \ldots, G_m \square d \in P \cup P', \text{ there exist satisfiable constraints } \{d_j\}_{j \in J}, \text{ and for every } j \in J \subseteq \{1, \ldots, m\}, \exists v' \in W_{\mathcal{A}}(P') \text{ with } v' \prec v \text{ such that } v', \mathcal{A} \Vdash_l \neg G_j \square d_j, \text{ and } FET_\Sigma \models (c \rightarrow \neg d \vee \bigvee_{j \in J} d_j)^\forall\}$$

$$L_{Succ_{P'}(v)} = \{p \square c \mid \text{for all } p(\overline{x}) :- G_1, \ldots, G_m \square d \in P \cup P', \text{ there exist satisfiable constraints } \{d_j\}_{j \in J}, \text{ and for every } j \in J \subseteq \{1, \ldots, m\}, v, \mathcal{A} \Vdash_l \neg G_j \square d_j, \text{ and } FET_\Sigma \models (c \rightarrow \neg d \vee \bigvee_{j \in J} d_j)^\forall\}$$

2. *$I_{T_P(\mathcal{A})}$ is defined as follows. If $w \in t_{P'}(W_{\mathcal{A}}(P'))$:*

$$I_{T_P(\mathcal{A})}(w) = \{p \square c \mid \text{there is } \{p(\overline{x}) :- \overline{G}^k \square d^k \mid 1 \leq k \leq n\} \subseteq P \cup P', \text{ and } \forall k, 1 \leq k \leq n, \exists v \in W_{\mathcal{A}}(P'), w = t_{P'}(v) \text{ such that } v, \mathcal{A} \Vdash_l \overline{G}^k \square d^k \text{ and } FET_\Sigma \models (c \rightarrow \bigvee_{k=1}^n d^k)^\forall\}$$

If $w \in Succ_{P'}(W_{\mathcal{A}}(P'))$, $I_{T_P(\mathcal{A})}(w) = \{p \square c \in I_{T_P(\mathcal{A})}(w') \mid w' \preceq w\}$.

Example 11 *Let us see the construction of the least fixpoint of the program P given in example 9. The bottom* Σ*-structure is just a structure where, for every program* P'*, we just have a world* $(P', \emptyset)$ *whose interpretation is the empty set of atoms. Then, the least fixpoint is* $T_P^4(\bot)$*:*

$$T_P(\bot) = \left\{ \begin{array}{c} (\emptyset,\{p,q\})\emptyset \\ | \\ (\emptyset,\emptyset)\emptyset \end{array} \quad \begin{array}{c} (\{p:-\neg q\},\{q\})\emptyset \\ | \\ (\{p:-\neg q\},\emptyset)\emptyset \end{array} \right\} \qquad T_P^2(\bot) = \left\{ \begin{array}{c} (\emptyset,\{p,q,s\})\emptyset \\ | \\ (\emptyset,\{p,q\})\emptyset \\ | \\ (\emptyset,\emptyset)\emptyset \end{array} \quad \begin{array}{c} (\{p:-\neg q\},\{q\})\{p\} \\ | \\ (\{p:-\neg q\},\emptyset)\emptyset \end{array} \right\}$$

$$T_P^3(\bot) = \left\{ \begin{array}{c} (\emptyset,\{p,q,s\})\{r\} \\ | \\ (\emptyset,\{p,q\})\{r\} \\ | \\ (\emptyset,\emptyset)\emptyset \end{array} \quad \begin{array}{c} (\{p:-\neg q\},\{q\})\{p,s\} \\ | \\ (\{p:-\neg q\},\emptyset)\emptyset \end{array} \right\} \qquad T_P^4(\bot) = \left\{ \begin{array}{c} (\emptyset,\{p,q,s\})\{r\} \\ | \\ (\emptyset,\{p,q\})\{r\} \\ | \\ (\emptyset,\emptyset)\emptyset \end{array} \quad \begin{array}{c} (\{p:-\neg q\},\{q\})\{p,r,s\} \\ | \\ (\{p:-\neg q\},\emptyset)\emptyset \end{array} \right\}$$

Σ-structures can be ordered according to the amount of information they contain. In particular, given two structures $\mathcal{A}$ and $\mathcal{B}$, we consider that $\mathcal{A} \preceq_F \mathcal{B}$ if there is some mapping from the worlds in $\mathcal{A}$ to the worlds in $\mathcal{B}$ such that the positive and negative information in a world w in $\mathcal{A}$ is included in the positive and negative information of the corresponding worlds in $\mathcal{B}$. In general, this mapping may associate to each w in $\mathcal{A}$, not just a world in $\mathcal{B}$, but a set of worlds. In addition, we ask this mapping to be downward surjective, which means that the worlds in $\mathcal{A}$ are surjectively mapped into a prefix of the set of worlds in $\mathcal{B}$.

Definition 12 *For all* $\mathcal{A}$ *and* $\mathcal{B} \in Struct(\Sigma)$, $\mathcal{A} \preceq_F \mathcal{B}$ *if, and only if, for every* Σ*-program* P*, there exists a map* $f_P : W_{\mathcal{A}}(P) \to 2^{W_{\mathcal{B}}(P)}$ *which is:*

i) Monotonic: $\forall w \in W_{\mathcal{A}}(P) \forall v \in f_P(w), w \preceq v$ *and* $I_{\mathcal{A}}(w) \subseteq I_{\mathcal{B}}(v)$*, and*

ii) Downward surjective: $\forall w \in W_{\mathcal{A}}(P) \forall w' \in W_{\mathcal{B}}(P)$ *such that* $\forall v \in f_P(w), w' \preceq v$ *then* $\exists w'' \in W_{\mathcal{A}}(P)$ *with* $w'' \preceq w$ *and* $w' \in f_P(w'')$*.*

Remark 13 *For every* Σ*-program* P*, if* T_P *is applied to a* Σ*-structure* $\mathcal{A}$ *such that for every* P'*,* $W_{\mathcal{A}}(P')$ *is a totally ordered structure, then* $t_{P'} : W_{\mathcal{A}}(P') \to W_{T_P(\mathcal{A})}(P')$ *in Definition 10, is a downward surjective embedding. In addition, all* $t_{P'}$ *defined by the powers of* T_P *on* $\bot$*,* $\{T_P^k(\bot) \mid 0 \preceq k\}$*, are monotonic.*

The previous relation is not an ordering when defined over the class of all Σ-structures. The problem is that antisymmetry may fail when considering non-constructive structures where there are infinite descending sequences of worlds. However, the following theorem shows that this relation is indeed an ordering when restricted to the subclass of *Noetherian* Σ-structures i.e. structures that do not include infinite descending sequences of worlds.

Theorem 14 $\preceq_F$ *is a complete partial order on Noetherian* Σ*-structures.*

Proof $\preceq_F$ is trivially reflexive. It is transitive because $i)$ and $ii)$ are preserved under composition. To show that it is antisymmetric we use parallel induction over the Noetherian order, $\preceq$, of worlds of Σ-structures, to prove that, if there exist monotonic and downward surjective maps $f_P : W_{\mathcal{A}}(P) \to 2^{W_{\mathcal{B}}(P)}$ and $g_P : W_{\mathcal{B}}(P) \to 2^{W_{\mathcal{A}}(P)}$ then their compositions are the corresponding identities.

The bottom is $\perp = \langle W_\perp, \preceq_\perp, I_\perp \rangle$, where $W_\perp = \{(P, \emptyset) \mid P \; is \; a \; \Sigma - program\}$, $\preceq_\perp = \{(w, w) \mid w \in W_\perp\}$ and $\forall w \in W_\perp$, $I_\perp(w) = \{\underline{t}\square c \mid FET_\Sigma \models c^\exists\}$. Note that this bottom is trivially Noetherian.

Finally, the least upper bound for every increasing chain of structures $\mathcal{A}_1 \preceq_F \mathcal{A}_2 \preceq_F \ldots$, can be described as a "world-by-world union" where the correspondence between them is established by the maps $W_{\mathcal{A}_1}(P) \to W_{\mathcal{A}_2}(P) \to \ldots$. ■

Theorem 15 *T_P, when restricted to the class of Noetherian Σ-structures, is monotonic and continuous with respect to $\preceq_F$ so, it has a least fixpoint $T_P \uparrow \omega$.*

Proof First of all, monotonicity is proved by showing that $\forall \mathcal{A}, \mathcal{B} \in Struct(\Sigma)$ such that $\mathcal{A} \preceq_F \mathcal{B}$ and, $\forall f_{P'} : W_{\mathcal{A}}(P') \to 2^{W_{\mathcal{B}}(P')}$ satisfying Definition 12, the map $g_{P'} : W_{T_P(\mathcal{A})}(P') \to 2^{W_{T_P(\mathcal{B})}(P')}$ defined as follows, also satisfies definition 12:

- $\forall w \in W_{T_P(\mathcal{A})}(P')$, if $w = t_{P'}(v)$ then $g_{P'}(t_{P'}(v)) = t_{P'}(f_{P'}(v))$
- $\forall w \in W_{T_P(\mathcal{A})}(P')$, if $w = Succ_{P'}(v)$ then $g_{P'}(Succ_{P'}(v)) = Succ_{P'}(f_{P'}(v))$

Then, since T_P is monotonic, to prove continuity it is enough to prove that T_P is finitary, that is for any infinite chain of Σ-structures $\mathcal{A}_1 \preceq_F \mathcal{A}_2 \preceq_F \ldots$, we have that $T_P(\sqcup \mathcal{A}_i) \preceq_F \sqcup T_P(\mathcal{A}_i)$. The proof proceeds in the standard way. First, one can show that any immediate consequence in $T_P(\sqcup \mathcal{A}_i)$ is obtained using a finite set of literals $\ell_j \square d_j$ from a finite set of worlds in $\sqcup \mathcal{A}_i$. Then, there will be a least upper bound $\mathcal{A}_n$, in the chain $\{\mathcal{A}_i\}$, including all these literals. ■

6 Least model semantics

In this section we will prove that the least fixpoint of the immediate consequence operator $T_P \uparrow \omega$ is the least model in $Mod(P)$ with respect to a proper notion of ordering. The key issue here is to define an ordering relation in $Mod(P)$, which we will denote by $\sqsubseteq$, such that it adequately captures the intuition that the "best model" is the least one. The definition of this ordering is based, first, on the definition of an ordering between ordered structures associated to a given program P. Then this ordering is extended to compare Σ-structures by comparing the ordered structures included.

One may notice that, in an ordered structure associated to P, (if this structure is part of a model of a program P') the negative information associated to a given world will contain, at most, the negative information supported by the worlds below. Similarly, the positive information associated to a given world will contain, at least, all the consequences that can be computed from the clauses in P and in P' and the negative information in the world. In this sense, one may consider that

the "best ordered structure" is one in which the negative and positive information associated to each world is, respectively, the maximum and the minimum amount of possible information. This means that the ordering between ordered structures should be based on an extension of the, so-called, standard ordering of 3-valued structures. However, given two ordered structures $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$, we should not try to compare pairwise all the worlds in one structure with the associated worlds in the other. For instance, let us suppose that P consists of the clause $r :- \neg q$ and P' is empty. If q is not included in the interpretation of the world $(P, \emptyset)$ in $\mathcal{B}_1(P)$ then the world above may be $(P, \{q\})$ and its interpretation would include r. However, if q is included in the interpretation of $(P, \emptyset)$ in $\mathcal{B}_2(P)$ then the world above can be $(P, \{r\})$. $\mathcal{B}_1(P)$ should be considered better than $\mathcal{B}_2(P)$. This can be done by defining this ordering among ordered structures as some kind of lexicographic extension of the standard ordering.

However, the fact that ordered structures may be not linear poses some small additional difficulty: two worlds may be incomparable but, at the same time, be defined over the same set of worlds. Nevertheless, with the intuition discussed above, to compare $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$ we proceed as follows. First, we look for two bars $B1$ and $B2$ in $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$, respectively, in each structure, such that all the worlds and interpretations below coincide and such that all the worlds and/or interpretations in both bars are different. Then, if all the worlds and interpretations in $B1$ are smaller (w.r.t. the standard ordering) than all the worlds and interpretations in $B2$, then we consider $\mathcal{B}_1(P)$ smaller than $\mathcal{B}_2(P)$. The following definitions capture these intuitions:

Definition 16 *Given a Σ-structure $\mathcal{B} = (W, \preceq, I)$, a Σ-program P and a strict bar $B \subseteq W$ w.r.t. $(P, \emptyset)$, we define $B{\downarrow} = \langle W_{B\downarrow}, \preceq, I_{B\downarrow}\rangle$ such that $W_{B\downarrow} = \{v \in W(P) | \exists w \in B$ and $v \prec w\}$ and $I_{B\downarrow} = \{I(w) \mid w \in W_{B\downarrow}\}$.*

Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be Σ-structures, P a Σ-program, $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$ the ordered structures associated to P in $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. Let $B_i \subseteq W_i(P)$, $i \in \{1, 2\}$ be strict bars w.r.t. $(P, \emptyset)$. Then, B_1 and B_2 are separator bars w.r.t. $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$ if, and only if, $B_1{\downarrow} = B_2{\downarrow}$ and for any other strict bars $B_i' \subseteq W_i(P)$ w.r.t. $(P, \emptyset)$ such that $\forall v_i \in B_i\ \exists v_i' \in B_i'$: $v_i \preceq v_i'$, $i \in \{1, 2\}$, $B_1'{\downarrow} \neq B_2'{\downarrow}$.

Separator bars are the bars, discussed above, that define where the differences in two ordered structures start. Separator bars are uniquely determined:

Lemma 17 *Let $\mathcal{B}_1 = (W_1, \preceq, I_1)$ and $\mathcal{B}_2 = (W_2, \preceq, I_2)$ be Σ-structures, P a Σ-program and $\mathcal{B}_1(P)$, $\mathcal{B}_2(P)$ the ordered structures associated to P in $\mathcal{B}_1$ and $\mathcal{B}_2$, and B_1, B_2 separator bars w.r.t. $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$. Then B_1 and B_2 are unique.*

Notice that given a Σ-structure $\mathcal{B}$, each bar B w.r.t. $(P, \emptyset)$ in $\mathcal{B}(P)$ induces a substructure $\langle W_{B\downarrow} \cup B, \preceq, I_{B\downarrow} \cup \{I(w) \mid w \in B\}\rangle$ which corresponds to an initial segment of the ordered structure $\mathcal{B}(P)$. For simplicity, in what follows, we will refer to this substructure as $B{\downarrow} \cup B$.

Definition 18 *Let $\mathcal{B}_1$ and $\mathcal{B}_2$ be Σ-structures, P a Σ-program and $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$ the ordered structures associated to P in $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively.*

1. *Let $B_i \subseteq W_i(P)$ be strict bars w.r.t $(P,\emptyset)$, $i \in \{1,2\}$, $B_1 \equiv B_2$ if, and only if, $B_1 = B_2$ and $\forall w \in B_1 : I_1(w) = I_2(w)$*

2. *Given two strict bars $B_i \subseteq W_i(P)$ w.r.t $(P,\emptyset)$, $i \in \{1,2\}$, $B_1 \sqsubseteq_b B_2$ if, and only if, $B_1 \equiv B_2$ or one of the following conditions holds:*

   (a) $\forall w_1 \in B_1 \forall w_2 \in B_2 : L_{w_2} \subset L_{w_1}$

   (b) $B_1 = B_2$ *and* $\forall w \in B_1 : I_1(w) \subset I_2(w)$.

   The strict order associated to this definition is denoted $\sqsubset_b$.

3. *$\mathcal{B}_1(P) \sqsubseteq_s \mathcal{B}_2(P)$ if, and only if, there exist separator bars $B_i \subseteq W_i(P)$, $i \in \{1,2\}$ w.r.t. $\mathcal{B}_1(P)$ and $\mathcal{B}_2(P)$ such that (a) or (b) holds:*

   (a) $B_1 \equiv B_2$ *and* $\mathcal{B}_2(P) = B_1{\downarrow} \cup B_1$

   (b) $B_1 \sqsubset_b B_2$

   The strict order associated to this definition is denoted $\sqsubset_s$.

Theorem 19 *The relation $\sqsubseteq_b$ over strict bars in ordered structures associated to a Σ-program in a Σ-structure is a partial order.*

Proof Reflexivity is straightforward and transitivity is a quite direct consequence of transitivity of $\subseteq$. To prove antisymmetry assume that $B_i \subseteq W_i(P)$, $i \in \{1,2\}$ are strict bars w.r.t $(P,\emptyset)$, such that $B_1 \sqsubseteq_b B_2$ and $B_2 \sqsubseteq_b B_1$ but they do not satisfy $B_2 \equiv B_1$. Then, B_1 and B_2 only can be comparable by 18.2a. Then, $\forall w_1 \in B_1 \forall w_2 \in B_2 : L_{w_2} \subset L_{w_1}$ and $L_{w_1} \subset L_{w_2}$. This is a contradiction. ■

Theorem 20 *The relation $\sqsubseteq_s$ over ordered structures associated to a Σ-program P in a Σ-structures is a partial order.*

Proof Reflexivity holds trivially. To prove antisymmetry suppose $\mathcal{B}_1(P) \sqsubseteq_s \mathcal{B}_2(P)$ and $\mathcal{B}_2(P) \sqsubseteq_s \mathcal{B}_1(P)$. By lemma 17, both relationships are established by the same separator bars B_1 and B_2 and $B_1 \sqsubseteq_b B_2$ and $B_2 \sqsubseteq_b B_1$. So, $B_1 \equiv B_2$ and, $\mathcal{B}_1(P) = \mathcal{B}_2(P)$ by 18.3a ($\mathcal{B}_1(P) = B_1{\downarrow} \cup B_1$ and $\mathcal{B}_2(P) = B_2{\downarrow} \cup B_2$). Transitivity is a consequence of lemma 17 and of transitivity of $\sqsubseteq_b$. ■

Example 21 *In Example 9, we can see that $\mathcal{B}_1(\emptyset) \sqsubseteq_s \mathcal{B}_2(\emptyset)$. In this case, the separator bars are $B_1 = \{(\emptyset, \{p,q\})\}$ and $B_2 = \{(\emptyset, \{q\})\}$, and $B_1 \sqsubseteq_b B_2$ because $\{q\} \subset \{p,q\}$ Also, $\mathcal{B}_1(\{p :- \neg q\}) \sqsubseteq \mathcal{B}_2(\{p :- \neg q\})$. Here, the separator bars are $B_1' = B_2' = \{(\{p :- \neg q\}, \emptyset)\}$ and $B_1' \sqsubseteq_b B_2'$ because $I_1((\{p :- \neg q\}, \emptyset)) = \emptyset \subset I_2((\{p :- \neg q\}, \emptyset)) = \{p,q,r,s\}$.*

Now, we may define the order relation between Σ-structures. One obvious possible definition for such an ordering would consist in saying that $\mathcal{B}_1$ is smaller than $\mathcal{B}_2$ if for every program P, $\mathcal{B}_1(P)$ is smaller than $\mathcal{B}_2(P)$. However, this definition would not be adequate. The problem is that, to decide what there should be in a given world for a program P we may need to look what information is included in the

worlds associated to a different program P'. The reason is that a certain clause in P may include an intuitionistic implication. This means that before comparing the ordered structures associated to P one should compare the ordered structures associated to P'. Again, this means that the ordering over Σ-structures should be a kind of lexicographic extension of the ordering on the structures associated to programs.

Definition 22 *Given $\mathcal{B}_i \in Struct(\Sigma)$, $i \in \{1,2\}$, $\mathcal{B}_1 \sqsubseteq \mathcal{B}_2$ if, and only if, for every Σ-program P one of the following conditions holds:*

1. *$\mathcal{B}_1(P) \sqsubseteq_s \mathcal{B}_2(P)$*
2. *There exists a program P', $P \subseteq P' \subseteq flat(P)$ such that $\mathcal{B}_1(P') \sqsubseteq_s \mathcal{B}_2(P')$*

where $flat(P)$ denotes the program consisting of all the clauses in P and in all the programs $flat(Q)$, where Q occurs in the left-hand-side of a clause in P.

Theorem 23 *The relation $\sqsubseteq$ is a partial order in $Struct(\Sigma)$.*

Proof Reflexivity is a consequence of reflexivity of $\sqsubseteq_s$. To prove antisymmetry we can consider the four cases resulting of combining 22.1 and 22.2. Combining just 22.1 directly leads to the equality of structures, and, it is not difficult to see that any other case leads to a contradiction. Again we can consider the four cases to prove transitivity. Now, combining just 22.2 let to a contradiction, and, the other cases hold by transitivity of $\sqsubseteq_s$. ■

Theorem 24 *For any Σ-program P, $T_P{\uparrow}\omega$ is the $\sqsubseteq$-least model in $Mod(P)$.*

Proof The proof uses double induction on the iterations of T_P to prove that for every $\mathcal{B} \in Mod(P)$ and for every $k \in \mathbb{N}$, $T_P^k(\bot) \sqsubseteq \mathcal{B}$; and, then, on the $\subseteq$-chain of programs (ordered structures) in the Σ-structures $T_P^k(\bot)$ and $\mathcal{B}$ to prove that conditions 22.1 and 22.2 hold. The base case is quite simple. The proof of $T_P^{k+1}(\bot) \sqsubseteq \mathcal{B}$ from $T_P^k(\bot) \sqsubseteq \mathcal{B}$ is a little long due to technicalities but the essential idea is quite simple and direct: It is enough to consider that the definition of T_P for worlds and interpretations are, respectively, if-and-only-if versions of the notions of supported models and local forcing of clauses in Definition 7. ■

7 Soundness and completeness

In this section we will prove the equivalence between the operational and the model-theoretic semantics defined above. In particular this means showing the soundness and completeness of the operational semantics of a program P with respect to the least fixpoint of the immediate consequence operator T_P.

Theorem 25 (Soundness) *If $P \vdash \overline{G}{\Box}c$ can be proved with computed answer c', then $(\emptyset,\emptyset)$, $T_P{\uparrow}\omega \Vdash \overline{G}{\Box}c'$.*

Proof We prove that for every Σ-program Q, if $P \cup Q \vdash \overline{G}\Box c$ can be proved with computed answer c', then $(Q, \emptyset), T_P\uparrow\omega \Vdash \overline{G}\Box c'$. The proof proceeds by induction on the number of derivation (plus subderivations) steps. The base step, when the Σ-expression in the goal is of the form $\Box c$, is trivial. To prove the inductive step there are four cases to consider depending if the goal is negative and if it has embbeded implication. The proofs for each of these cases are very similar and standard. Using the corresponding operational rule, the inductive hypothesis and the definition of T_P we conclude that $(Q, \emptyset), T_P\uparrow\omega \Vdash \overline{G}\Box c'$. ■

Theorem 26 (Completeness) *If* $(\emptyset, \emptyset), T_P\uparrow\omega \Vdash \overline{G}\Box c$, *then* $P \vdash \overline{G}\Box c$ *can be proved with computed answers* $c_1, \ldots, c_n$ *such that* $FET_\Sigma \models (c \rightarrow \bigvee_{i=1}^{n} c_i)^\forall$.

Proof As in the previous theorem, we prove that for every Σ-program Q if $(Q, \emptyset), T_P\uparrow \omega \Vdash \overline{G}\Box c$, then $P \cup Q \vdash \overline{G}\Box c$ can be proved with computed answers $c_1, \ldots, c_n$ such that $FET_\Sigma \models (c \rightarrow \bigvee_{i=1}^{n} c_i)^\forall$. Again the proof is very standard and uses induction on the iterations of T_P. The base step, $k = 0$, is trivial. For proving the inductive step (in the same four cases) it is enough to use the corresponding operational rule, the inductive hypothesis and the definition of T_P. ■

Acknowledgements This work has been partially supported by the CICYT project HEMOSS (ref. TIC98-0949-C02-01) and CIRIT GRC 1999SGR-150.

References

[1] A. J. Bonner and L. T. McCarty. Adding negation-as-failure to intuitionistic logic programming. In *Proc. of the North American Conference on Logic Programming (NACLP'90)*, pages 681–703, 1990.

[2] M. Bugliesi, E. Lamma, and Mello Paola. Modularity in logic programming. *Journal of Logic Programming*, 19,20:443–502, 1994.

[3] W. Drabent. What is a failure? An approach to constructive negation. *Acta Informática*, 32:27–59, 1995.

[4] S. Etalle and F. Teusink. A compositional semantics for normal open programs. In *Proc. Int. Conf. and Symp. on Logic Programming'96*. The MIT Press, 1996.

[5] F. Fages. Constructive negation by pruning. *Journal of Logic Programming*, 32:85–118, 1997.

[6] G. Ferrand and A. Lallouet. A compositional proof method of partial correctness for normal logic programs. In *Int. Logic Programming Symp.*, pages 209–223. J. Lloyd, ed., 1995.

[7] L. Giordano and N. Olivetti. Combining negation as failure and embedded implication in logic programs. *The Journal of Logic Programming*, (36):91–147, 1998.

[8] P. Lucio, F. Orejas, and E. Pino. An algebraic framework for the definition of compositional semantics of normal logic programs. *Journal of Logic Programming*, 40:89–123, 1999.

[9] D. Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 6:79–108, 1989.

[10] F. Orejas, E. Pino, and H. Ehrig. Institutions for logic programming. *Theoretical Computer Science*, 173:485–511, 1997.

[11] E. Pasarella, E. Pino, and F. Orejas. Constructive negation without subsidiary trees. In Alpuente M., editor, *Proceedings of the 9th International Workshop on Functional and Logic Programming, WFLP'2000.*, pages 195–209. Universidad Politécnica de Valencia, España, 2000.

[12] J.C. Shepherson. Negation in logic programming. In J. Minker, editor, *Foundations on Deductive Databases and Logic Programs*, pages 19–88. Morgan Kaufmann, 1988.

[13] P. J. Stuckey. Negation and constraint logic programmming. *Information and Computation*, 118:12–23, 1995.

[14] D. van Dalen. Intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic – Vol III*, 1986.