A Monotonic Declarative Semantics for Normal Logic Programs

Paqui Lucio
Dept. de L.S.I. ,Universidad del Pais Vasco
San Sebastian, SPAIN

Fernando Orejas, Elvira Pino
Dept. de L.S.I. , Universitat Politècnica de Catalunya
Barcelona, SPAIN

Abstract

In this paper, we propose a declarative semantics for normal logic programs in terms of model classes that is monotonic in the sense that $Mod(P \cup P') \subseteq Mod(P)$, for any programs P and P'. In addition, we 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, and such that this least model is "typical" for the class of models of the Clark-Kunen's completion of the program. This means that our semantics is equivalent to Clark-Kunen's completion. The final aim of our work is the definition of compositional semantics for modular units over the class of normal logic programs. In this sense, following the results of a previous paper, it has been shown that our semantics constitutes a "specification frame" equipped with the adequate categorical constructions to guarantee the existence of such semantics.

1 Introduction

Despite the amount of papers on the semantics of negation (see e.g. [2]) , there are several semantic issues that are yet insufficiently explored. One such basic issue is modularity. The reason is that a proper semantics for any kind of modular unit must be shown to be compositional with respect to the kind of module operations considered, but the non-monotonic nature of negation in Logic Programming does not seem to fit too well with compositionality. In particular, for different reasons, none of the various operational semantics [7,11,26], neither the different model-theoretic approaches (see e.g. [2]) nor the completion semantics [6,17] seem to be adequate to be the basis for defining a compositional semantics for normal logic program units. To our knowledge, only [8, 14, 19] provide some compositional semantic constructs for normal logic programs. Below we compare the results presented in this paper and these approaches.

In [21], it is presented a methodology for the semantic definition of modular logic programs ensuring compositionality and full abstraction, and we applied it to study several kinds of program units for the class of definite logic programs. The approach is based on the fact that most modular constructions can be defined and studied independently of the underlying formalism used "inside" the modules, as far as this formalism is an "institution" [16] or a "specification frame" [12] (or some similar notion) equipped with some categorical constructions. In particular, the proposed methodology for defining the semantics of a certain kind of modular unit consists, essentially, of three steps. Firstly, one has to study the given

unit, and the associated composition operations, at the general level. This means defining the meaning of the construction in terms of the categorical constructions that the specification frames will be assumed to provide. Secondly, one has to define the given class of logic programs as an institution or specification frame with the needed constructions. At this point one may obtain, already, a compositional and fully-abstract semantic definition for the given unit. However, such definition could be considered too abstract, since it would be stated in terms of the categorical constructions of the institution. Finally, the third step consists in showing that this categorical semantics is equivalent to some specific concrete semantics. Applying this methodology not only may save some work (since some results must be proved just once, independently of the classes of logic programs considered) but, what is more important, it provides clear guidelines about how the concrete semantics for the various constructions must be defined. In particular, these guidelines were extremely valuable for the work reported in this paper.

In principle, the main problem found in order to apply this methodology to study modularity and compositionality issues for the class of normal logic programs is (the lack of) monotonicity. Institutions and specification frames can be seen as characterizations of monotonic formalisms. This seems to be in contradiction with the non-monotonic nature of negation as failure and of constructive negation. However, if we look at the simpler case of the class of definite logic programs with negative queries, one may see one of the basic ideas of our proposal: the class of definite logic programs (Horn Clause Logic) is, obviously, a monotonic logic; the non-monotonicity of the negative queries is related to the selection of an arbitrary model (the least one) to define what is assumed to be "false". Similarly, in this paper, we propose a declarative semantics for normal logic programs in terms of model classes that is monotonic in the sense that $Mod(P \cup P') \subseteq Mod(P)$, for any programs P and P'. This is enough for defining an specification frame of normal logic programs. Nevertheless, in addition, we 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 and that this least model is "typical" for the class of models of the Clark-Kunen's completion of the program. This means that our semantics is equivalent to Clark-Kunen's completion. Moreover, we provide a continuous immediate consequence operator and show that this least model can be built "upwards" as the least fixpoint of that operator. Finally, we (very briefly) sketch the results that allow us to show that the specification frame associated to our semantic definition is equipped with the categorical constructions needed to apply the results in [21] to the class of normal logic programs, obtaining compositional and fully abstract (categorical) semantics for a number of program units [4,15].

The kind of compositionality results that can be achieved, using our semantics as the basis for defining the corresponding specification frame, are quite more powerful than the results presented in [8, 14, 19]. In these papers different semantic definitions are provided for certain kinds of modular units which are shown to be compositional. However, they all impose (at least) the restriction that when putting together (through the corresponding composition operation) two units then the sets of predicates defined in each unit must be disjoint. This means that, there can not be clauses defining the same predicate p (i.e. having p in the head of a clause) in both units. This restriction hinders the application of those results to approaches where the given system of modules supports the incremental definition of predicates through some form of inheritance (e.g. [3]). This kind of restriction is not needed in our work.

We have not directly related our approach with other kinds of semantics, although the relation established with completion implies, by transitivity, that our semantics can be considered equivalent to constructive negation approaches as [7,26]. Actually, the relation to [7] is quite more direct, in the sense that the construction of our least models is closely related to ranked

resolution as defined there. There is also a certain relation between the construction of our least models and Fitting's fixpoint semantics [9], or rather with the variation defined in [13], although not as close as it may seem: one may notice that in each "layer" of our least models we add not just the immediate consequences of the previous layer, but all logical consequences.

We are convinced that, not only with respect to compositionality issues, our semantics is just the "right" kind of model-theoretic semantics for normal logic programs. There are several reasons for this. On the one hand, if model-theoretic semantics are usually the most adequate tool for meta-logical reasoning, the structure of our classes of models, together with the closeness to ranked resolution, makes our semantics highly adequate for the proof of such kind of properties, especially for the case of properties of operational approaches (e.g. completeness). On the other hand, one may think that our semantics is more complicated than needed, in the sense of dealing with complex ranked structures instead of just three-valued ones. We disagree with that: ranked structures are just a special case of Beth structures, used to provide semantics to intuitionistic logic. In this sense, our semantics establishes an explicit link (already mentioned by other authors) between constructive negation and intuitionistic logic that may be worth to study. In particular, it could serve as a basis for extending with negation those approaches to modularity based on the use of an intuitionistic implication (e.g. see [27]).

The paper is organized as follows: in the next section we introduce some basic notions and notation; in section 3, we sketch the propositional case to provide some intuition about the proposed solutions. In sections 4, 5 and 6, we present the semantics for the first-order case, including a fixpoint construction of least models, and show its connection with Clark-Kunen's completion. Finally, in section 7, we discuss the results presented with respect to compositionality issues.

2 Preliminaries

A countable signature Σ consists of a pair of sets (FS_Σ, PS_Σ) of function and predicate symbols, respectively, with some arity associated. Σ-terms and Σ-atoms are built using functions and predicates from Σ and, also, variables from a prefixed countable set X of variable symbols. Terms will be denoted by t,s,..., and var(t) will denote the variables appearing in t. In what follows, we will consider that all programs are built over a fixed and predetermined countable signature. The main reason for this is technical, and is related to the definition of a *specification frame* based on the semantics introduced in this paper. Actually, the results presented in this paper are independent of this condition. One can argue that this is highly inconvenient with respect to modularity issues, however we do not think that this is important as far as visibility is treated completely at the static semantics level. On the other hand, we believe that this negative situation is in some sense related with the nature of negation-as-failure where one can always expect to obtain (negative) answers to queries over predicates which are not in the signature of the given program.

Normal programs of a signature Σ (or Σ-programs) are sets of Σ-clauses a:- $l_1,... l_n$, where a is a Σ-atom, $k \geq 0$, and each l_i is a Σ-literal, that is b or not(b) where b is an atom.

For simplifying some technical constructions, we consider that any Σ-program is written as its equivalent constraint normal program with flat head. That is, any clause $p(t_1,..., t_n)$:- $l_1,... l_k$ is written as the constraint clause

$$p(x_1,..,x_n) :- l_1,..,l_k \square x_1=t_1,..,x_n=t_n.$$

Moreover, we suppose identical tuples $x_1,..., x_n$ of *fresh variables* for all clauses (in a program) with predicate p in its head.

Constraints appearing in programs are a special kind of simple constraints. In general, we consider that Σ-constraints are arbitrary first order Σ-formulas over equality atoms. That is, formulas composing equality atoms with the connectives: $\neg,\vee,\wedge,\rightarrow$, and the quantifiers $\forall$, $\exists$. We denote constraints by c,d,... (possibly with sub- or superscripts). By c(X) we mean that free(c)$\subseteq$X, where free(c) is the set of free variables in c. Formulas $\varphi^{\forall},\varphi^{\exists}$ stand for the universal and existential closures of φ, respectively. The atomic formulas naming the two classical truth values are **T** and **F**.

We will handle constraints in a logical way, using logical consequence of the *free equality theory*. In particular, the free equality theory $\mathcal{FET}\Sigma$ for a signature Σ consists of the following set of formulas:

$$\begin{array}{ll}
\forall\, x\,(x = x) & \\
\forall\, \bar{x}\,\forall\, \bar{y}\;(\bar{x} = \bar{y} \leftrightarrow f(\bar{x}) = f(\bar{y})) & \text{for each } f \in FS\Sigma. \\
\forall\, \bar{x}\,\forall\, \bar{y}\;(\bar{x} = \bar{y} \rightarrow (p(\bar{x}) \leftrightarrow p(\bar{y}))) & \text{for each } p \in PS\Sigma \cup \{=\}. \\
\forall\, \bar{x}\,\forall\, \bar{y}\;(f(\bar{x}) \neq g(\bar{y})) & \text{for each pair } f,g \in FS\Sigma \text{ such that } f \not\equiv g. \\
\forall\, x\,(x \neq t) & \text{for each } \Sigma\text{-term } t \text{ and variable } x \text{ such} \\
 & \text{that } x \not\equiv t \text{ and } x \text{ occurs in } t.
\end{array}$$

We say that a constraint c is satisfiable (resp. unsatisfiable) iff

$$\mathcal{FET}\Sigma \models c^{\exists} \quad (\text{resp. } \mathcal{FET}\Sigma \models \neg\,(c^{\exists})).$$

A ground substitution $\bar{x} = \bar{t}$ (where t_i are closed) is called a solution for a constraint $c(\bar{x})$ iff

$$\mathcal{FET}\Sigma \models (\bar{x} = \bar{t} \rightarrow c)^{\forall}.$$

From a logical point of view, programs are sets of formulas. There are, mainly, two logical ways of interpreting a normal program P. The first one, denoted by $P^{\forall}$, interprets every clause as the universal closure of the formula which results from substituting "commas" and ▫ (in the clause body) by logical conjunction, and the symbol ":-" by logical implication (right-to-left). The second one is Clark's program completion, denoted by Comp(P), consisting of the free equality theory $\mathcal{FET}\Sigma$ together with, for each predicate symbol $p \in \Sigma$, a formula:

$$\forall\, x\,(p(\bar{x}) \leftrightarrow \bigvee_{i=1}^{k} \exists\, \bar{y}^{i}\,(\bar{x} = \bar{t}^{i} \wedge \bar{l}^{i}))$$

where $\bar{y}^{i}$ are the variables appearing in $\bar{t}^{i}$ and $\bar{l}^{i}$ and $\{p(\bar{x}) :- \bar{l}^{i} \square \bar{x}^{i} = \bar{t}^{i} \mid i=1,..,k\}$ is the set of all clauses with head p appearing in P. In both interpretations, conjunction (resp. disjunction) of an empty set is simplified to the atomic formula **T** (resp. **F**).

However, clauses like (p:-¬ p) are inconsistent when considering completion. To avoid this problem, Kunen [17] proposed to interpret Clark's completion in a three-valued logic. In particular, in this logic the three truth values are true (**t**), false (**f**) and undefined (**u**), the connectives $\neg,\vee,\wedge,\rightarrow$ are interpreted in Kleene's partial logic, $\leftrightarrow$ is interpreted as the identity of truth values, so it is two-valued; finally, existential quantification can be seen as infinite disjunction, and universal quantification is treated as infinite conjunction. Currently, this interpretation of Clark's completion (from now on Clark-Kunen completion) is considered the standard declarative meaning of normal logic programs. Anyhow, it must be noted that, in the context of completion, any three valued extension of classical implication can be considered. The reason is that implication does not appear in predicate completion formulas, i.e. the choice of a three-valued semantics for implication becomes an important matter when the program itself is treated as a logical theory. In this sense, we will use Przymusinski's implication, whose intuitive meaning is "$\varphi\rightarrow\psi$ is true if and only if whenever φ is true ψ is also true and whenever ψ is false φ is also false". Then $\varphi\leftrightarrow\psi$ is equivalent to $(\varphi\rightarrow\psi)\wedge(\psi\rightarrow\varphi)$ and, in particular, we have that $Comp(P) \models P^{\forall}$. However, the classical equivalence $(\varphi\rightarrow\psi) \equiv (\neg\varphi\vee\psi)$ is only satisfied, in general, if φ is two-valued (e.g. an equality formula).

A three-valued Σ-structure $\mathcal{A}$ consists of a universe of values A and an interpretation of every function symbol by a (total) function from A^n to A (of the adequate arity) and of every predicate symbol by a partial relation, which can be seen as a (total) function from A^n into $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$. A Herbrand three-valued structure $\mathcal{H}$ is a three-valued Σ-structure whose universe H is the Herbrand base for Σ (the set of all closed Σ-terms), function symbols are trivially interpreted and the predicate interpretation is given by a pair of disjoint sets: H^+ of true ground atoms and H^- of false ground atoms, so that any other ground atom is undefined. The value of any first order sentence φ in a three-valued structure $\mathcal{A}$ will be denoted by $\mathcal{A}(\varphi)$. A three-valued structure $\mathcal{A}$ is a model of a set of formulas Φ, denoted by $\mathcal{A} \models \Phi$, iff $\mathcal{A}(\varphi) = \mathbf{t}$ for any formula $\varphi \in \Phi$. Three-valued logical consequence $\Phi \models \varphi$ means that for every three-valued structure $\mathcal{A}$ if $\mathcal{A} \models \Phi$ then $\mathcal{A} \models \varphi$.

3 A first approach: The propositional case

As said in the introduction, our aim is to define a model-theoretic semantics for normal programs (i.e. the meaning of a program P is the set Mod(P) of all models of P, for a given notion of model), such that the following monotonicity property holds

$$\text{Mod}(P) \supseteq \text{Mod}(P \cup P') \text{ for all P and P'.}$$

In addition, we require the existence of some ordering $\preceq$ over the class Mod(P) such that the least model of P, with respect to $\preceq$, represents the meaning of P. We will denote this model by $\mathcal{M}P$. It may be noted that the monotonicity property requires that: $\mathcal{M}P \preceq \mathcal{M}P \cup P'$. Unfortunately, if models are three valued structures and we want our semantics to be equivalent, in some sense, to Clark-Kunen's completion, then it is impossible to satisfy these requirements, as the following counter-example shows:

Let us consider the normal program $P1 = \{a:\neg b\}$, its least model $\mathcal{M}P1$ should be the pair $(\{a\},\{b\})$, and consider $P1' = \{b:-\}$, then $\mathcal{M}P1 \cup P1' = (\{b\},\{a\})$. Then, we must have $(\{a\},\{b\}) \preceq (\{b\},\{a\})$. Now, by considering the program $P2 = \{b:\neg a\}$ and extending it with the clause $\{a:-\}$ we have that $(\{a\},\{b\}) \preceq (\{b\},\{a\})$ should hold.

From our point of view, the problem in this counter-example is that $\mathcal{M}P1 \cup P1'$ and $\mathcal{M}P2$ should not be identical and should reflect, in some sense, the "dependencies from negative information" that make a given atom to be in the model. For instance, $\mathcal{M}P2$ includes b as a consequence of the negative information provided by a, while $\mathcal{M}P1 \cup P1'$ includes b without any dependency on negative information. This consideration has led us to consider models having "layers" that reflect these dependencies. We call these models *ranked structures* because of their relation with ranked resolution. For instance, if we consider again the above example. The "intended" model for P1 has a first layer given by $(\emptyset,\{b\})$ and a second layer $(\{a\},\{b\})$. However for $P1 \cup P1'$ the first layer is $(\{b\},\emptyset)$, and the second is $(\{b\},\{a\})$. Similarly, for P2 the first layer is $(\emptyset,\{a\})$ and a second layer $(\{b\},\{a\})$. Now the intended models associated to $P1 \cup P1'$ and to P2 are different, since their first layer differs.

Therefore, our semantics is defined in terms of *ranked three-valued Σ-structures*. In the propositional case, it is enough to consider sequences of Herbrand three-valued Σ-structures: $\mathcal{A} = \langle (A_i^+, A_i^-) \rangle_{i \in \mathbb{N}}$ such that for any $i \in \mathbb{N}$:

$$A_i^+ \subseteq A_{i+1}^+ \text{ and } A_i^- \subseteq A_{i+1}^-, \text{ and } A_i^+ \cap A_i^- = \emptyset$$

Now, we define when a ranked structure is a model of a program:

3.1 Definition. A ranked three-valued Σ-structure $\mathcal{A}$ is a model of a normal program P, denoted by $\mathcal{A} \models P$, iff the following four conditions are satisfied:

(a) If $P^{\forall} \models a$ then $a \in A_0^+$ (In particular if $a:- \in P$)

(b) If $a \in A_0^-$ then there is not any clause $a:- \overline{l}$ in P

(c) If $P^{\forall} \cup A_i^+ \cup \neg A_i^- \models a$ then $a \in A_{i+1}^+$, where $\neg A_i^-$ means $\{\neg a \mid a \in A_i^-\}$

(d) If $a \in A_{i+1}^-$ then for all $a:- \overline{l} \in P$ it holds one of the following two facts:

- there exists $b \in \overline{l}$ such that $b \in A_i^-$
- there exists $\neg b \in \overline{l}$ such that $b \in A_i^+$.

Notice that, for the above program P consisting of a unique clause $a:-\neg b$, the following are some of its models (if we just write n layers, it must be assumed that the following layers are equal to the n-th layer):

$\mathcal{M}1 = \langle(\emptyset,\{b\}), (\{a\},\{b\})\rangle$ $\mathcal{M}2 = \langle(\{a\}, \{b\})\rangle$ $\mathcal{M}3 = \langle(\{a,b\}, \emptyset)\rangle$
$\mathcal{M}4 = \langle(\emptyset,\emptyset)\rangle$ $\mathcal{M}5 = \langle(\{b\},\emptyset), (\{b\},\{a\})\rangle$
but the following is not a model of P: $\mathcal{M}6 = \langle(\{b\},\{a\})\rangle$.

Our model notion allows one to include (in any layer) more positive information than what is supported as logical consequence of the previous layers, but the negative information of each layer must be supported (in that sense). Thus, if we want to define an ordering, $\preceq$, on ranked structures such that the least model is the one having, at each layer, the least amount of positive information and the greatest amount of negative information supported by the previous layer, it suffices to consider that $\preceq$ is the lexicographic extension, over sequences $\langle(A_i^+,A_i^-)\rangle_{i\in\mathbb{N}}$, of the standard ordering over three-valued structures:

$$(A^+,A^-) \preceq (B^+,B^-) \quad \text{iff} \quad A^+ \subseteq B^+ \text{ and } A^- \supseteq B^-.$$

It is easy to see that, for the above program P, $\mathcal{M}1$ is the $\preceq$-least model in Mod(P).

Now, consider the case where we add the clause b:-b to P, then $\mathcal{M}1$ and $\mathcal{M}2$ are not models of the new program. In this case, the least model is $\mathcal{M}4$. Furthermore, by adding a third clause b:-, $\mathcal{M}4$ is not a model of the new program {a:-¬b; b:-b; b:-} and the least model would now be $\mathcal{M}5$.

4 A model-theoretic semantics for constructive negation

In this section we extend the model-theoretic semantics, described in the previous section for the propositional case, to the general case of normal programs with variables. Firstly, one must note that this extension can not just consist in seeing programs with variables as just abbreviations for programs including all the possible ground instantiations of the given clauses. For instance, the programs

P1 = {nat(0):-, nat(s(X)):- nat(X)} and P2 = { nat(X):- }

have exactly the same instances, if we consider a signature including only the constant 0 and the unary function symbol s, but they have a completely different behaviour. For instance the query

?- not(nat(X)).

would be undefined for P1 and false for P2. The solution proposed is rather to handle the first-order case in a similar manner to the propositional case, but considering ranked structures including not just ground atoms but constrained atoms with variables:

4.1 Definition A ranked Σ-structure is a sequence: $\mathcal{A}=\langle (A_i^+,A_i^-) \rangle_{i\in\mathbb{N}}$ such that for any $i\in\mathbb{N}$:

- A_i^+ and A_i^- are sets of pairs $p(\overline{x}) \square c(\overline{x})$ where $p \in PS_\Sigma$ and $c(\overline{x})$ is a satisfiable Σ-constraint,
- A^+ and A^- are closed under renaming of variables,
- $A_i^+ \subseteq A_{i+1}^+$ and $A_i^- \subseteq A_{i+1}^-$, and

- $A_i^+ \cap A_i^- = \emptyset$

We will not make explicit the free variables of a constrained atom whenever they are not relevant. A pair $p \square c \in A_i^+$ is logically interpreted as the formula $(c \rightarrow p)^\forall$, and $p \square c \in A_i^-$ has the logical meaning of $(c \rightarrow \neg p)^\forall$. Consequently, we define the sets:

$(A_i^+)^\forall = \{(c \rightarrow p)^\forall \mid p \square c \in A_i^+\}$, $(A_i^-)^\forall = \{(c \rightarrow \neg p)^\forall \mid p \square c \in A_i^-\}$ and $A_i^\forall = (A_i^+)^\forall \cup (A_i^-)^\forall$

4.2 Definition Given a ranked three-valued Σ-structure $\mathcal{A}$ and a program P, $\mathcal{A} \models_\Sigma P$ iff the following four conditions are satisfied:

(a) If $\mathcal{FET}_\Sigma \cup P^\forall \cup (A_0^+)^\forall \models (c \rightarrow p)^\forall$ and c is satisfiable then $p \square c \in A_0^+$

(b) If $p(\bar{x}) \square c(\bar{x}) \in A_0^-$ then $c \wedge c'$ is unsatisfiable for all (properly renamed) clauses $p(\bar{x}) :- \bar{l} \square c' \in P$

(c) If $\mathcal{FET}_\Sigma \cup P^\forall \cup (A_{i+1}^+)^\forall \cup (A_i^-)^\forall \models (c \rightarrow p)^\forall$ and c is satisfiable then $p \square c \in A^+_{i+1}$

(d) If $p(\bar{x}) \square c(\bar{x}) \in A_{i+1}^-$ then $\mathcal{FET}_\Sigma \cup A_i^\forall \models ((c \wedge c') \rightarrow \neg \bar{l})^\forall$ for all (properly renamed) clauses $p(\bar{x}) :- \bar{l} \square c' \in P$.

Remark Conditions (a) and (c) could be slightly simplified into:

(a') If $\mathcal{FET}_\Sigma \cup P^\forall \models (c \rightarrow p)^\forall$ and c is satisfiable then $p \square c \in A_0^+$
(c') If $\mathcal{FET}_\Sigma \cup P^\forall \cup (A_i^-)^\forall \models (c \rightarrow p)^\forall$ and c is satisfiable then $p \square c \in A^+_{i+1}$

if we would not have the aim of proving that this semantics defines a specification frame. Unfortunately, properties (a) and (c) are needed for proving the so-called amalgamation property of specification frames.

Now, we can define a model-theoretic semantics for normal programs, in terms of the class of models of every program P:

$$Mod(P) = \{ \mathcal{A} \mid \mathcal{A} \models P \}.$$

This semantics is monotonic with respect to program extension:

4.3 Theorem For all programs P, P': $Mod(P) \supseteq Mod(P \cup P')$.

Sketch of the proof: Suppose $\mathcal{A} \models P \cup P'$, for proving that $\mathcal{A} \models P$ conditions b) and d) are trivial. In order to prove conditions a) and c) it is enough to consider that $(P \cup P')^\forall = P^\forall \cup P'^\forall$, which means that for any formula φ, it holds: if $\mathcal{FET}_\Sigma \cup P^\forall \models \varphi$ then $\mathcal{FET}_\Sigma \cup (P \cup P')^\forall \models \varphi$. ■

Likewise in the propositional case, the ordering considered over Mod(P) is the lexicographic extension $\preceq$ over sequences $\langle (A_i^+, A_i) \rangle_{i \in \mathbb{N}}$ of the standard ordering

$$(A^+, A^-) \preceq (B^+, B^-) \quad \text{iff} \quad A^+ \subseteq B^+ \text{ and } A^- \supseteq B^-$$

As in the propositional case, we have:

4.4 Theorem For any Σ-program P there exists a $\preceq$-least Σ-model $\mathcal{M}_P$ in the class Mod(P).

Sketch of the proof Given P, it is enough to define $\mathcal{M}_P$ as the ranked Σ-structure $\langle (M_i^+, M_i) \rangle_{i \in \mathbb{N}}$ such that
- M_0^+ is the $\subseteq$-least set satisfying condition (a)
- M_0^- is the $\subseteq$-greatest set satisfying condition (b)
- M_{i+1}^+ is the $\subseteq$-least set satisfying condition (c)
- M_{i+1}^- is the $\subseteq$-greatest set satisfying condition (d) of definition 4.2. ■

5 The least model

In this section we study some interesting properties of least models. In particular, different characterizations by logical consequence closure and its constructive definition through an immediate consequence operator. From now on, $\langle (M_i^+, M_i^-)\rangle_{i\in\mathbb{N}}$ will denote the least model $\mathcal{M}P$ of a prefixed program P.

For each layer $i\in\mathbb{N}$, the positive part of $\mathcal{M}P$ can be obtained as the intersection of the positive parts, at layer i, of all models of P, and the negative part of $\mathcal{M}P$ is the union of i layered negative parts of all models of P. However, for two arbitrary models of a program P, the ranked structure resulting to intersect positive parts and to joint negative parts, layer by layer, could not be a model of P, as the following counterexample shows:

5.1 Counterexample Let P be the program {a:-b, c:-¬d, b:-}, then two models of P are: $\langle$({a,b,c},{d}), ({a,b,c},{d})$\rangle$ and $\langle$({a,b},Ø), ({a,b},Ø)$\rangle$, but $\langle$({a,b},{d}), ({a,b},{d})$\rangle$ is not.

5.2 Theorem Let P be any program, then: $\mathcal{M}P = \langle (\bigcap_{\mathcal{A}\in Mod(P)} A_i^+, \bigcup_{\mathcal{A}\in Mod(P)} A_i^-)\rangle_{i\in\mathbb{N}}$.

Proof Let $i\in\mathbb{N}$ and take any constrained atom $a\square c\in M_i^+$, since $M_i^+\subseteq A_i^+$ for all $\mathcal{A}\in Mod(P)$, then $a\square c\in\bigcap_{\mathcal{A}\in Mod(P)} A_i^+$. Conversely, suppose that $a\square c$ belongs to the intersection, since $\mathcal{M}P\in Mod(P)$, then it belongs to M_i^+. For negative parts, take $i\in\mathbb{N}$ and any constrained atom $a\square c\in M_i^-$, since $\mathcal{M}P\in Mod(P)$, then $a\square c\in\bigcup_{\mathcal{A}\in Mod(P)} A_i^-$. In the opposite direction it is enough to use that $M_i^-\supseteq A_i^-$ for all $\mathcal{A}\in Mod(P)$ ■

A second characterization is made in terms of the logical consequence closure of the equality theory and the standard logical interpretation of the program.

5.3 Theorem For any program P:

(i) $a\square c \in M_0^+ \Leftrightarrow \mathcal{FET}_\Sigma \cup P^\forall \models (c\rightarrow a)^\forall$

(ii) $a\square c \in M_{i+1}^+ \Leftrightarrow \mathcal{FET}_\Sigma \cup P^\forall \cup M_i^\forall \models (c\rightarrow a)^\forall$

(iii) $a\square c \in M_i^- \Leftrightarrow \mathcal{FET}_\Sigma \cup P^\forall \cup M_i^\forall \models (c\rightarrow \neg a)^\forall$

Sketch of the proof The right-to-left implications of (i) and (ii) as well as the left-to-right implication of (iii) are trivial. For the converse implication of (i), it is enough to notice that the set $B= \{b\square d/\ \mathcal{FET}_\Sigma \cup P^\forall \models (b\rightarrow d)^\forall\}$ satisfies condition a) of 4.2 since for every set of formulas Φ: $\Phi\cup\{\varphi/\ \Phi\models\varphi\}\models\psi \Rightarrow \Phi\models\psi$.

The proof for the converse implication of (ii) is similar, but taking as B the set

$$B = \{b\square d/\ \mathcal{FET}_\Sigma \cup P^\forall \cup M_i^\forall \models (b\rightarrow d)^\forall\}$$

For the right-to-left implication of (iii), the key idea is that the program P cannot "add" new negative logical consequences. In particular, if we assume that $a\square c$ is not in M_i^-, then we can build a model of $\mathcal{FET}_\Sigma \cup P^\forall \cup M_i^\forall$ which is not a model of $(c\rightarrow\neg a)^\forall$: it is enough to consider the Herbrand structure (A^+,A^-) where A^- consists of all atoms $b\sigma$ such that $b\square d \in M_i^-$ and σ is a substitution satisfying d, and A^+ includes the rest of the atoms. ■

A trivial consequence of the previous lemma is that $\mathcal{M}P$ is closed with respect to less general constraints.

5.4 Corollary Let P be a program and let $\langle (M_i^+, M_i^-)\rangle_{i\in\mathbb{N}}$ be the least model of P, for every $i\in\mathbb{N}$:

(i) $a\square c \in M_i^+$ and $\mathcal{FET}_\Sigma \models (d\rightarrow c)^\forall$ and d is satisfiable then $a\square d \in M_i^+$

(ii) $a\square c \in M_i^-$ and $\mathcal{FET}_\Sigma \models (d\rightarrow c)^\forall$ and d is satisfiable then $a\square d \in M_i^-$

Proof It is enough to notice that for any set of formulas $\Phi \cup \{\varphi\}$, if $\mathcal{FET}_\Sigma \cup \Phi \models (c \rightarrow \varphi)^\forall$ and $\mathcal{FET}_\Sigma \models (d \rightarrow c)^\forall$, then $\mathcal{FET}_\Sigma \cup \Phi \models (d \rightarrow \varphi)^\forall$ ■

Now, we are going to characterize the least model in the usual constructive way as a least fixpoint of a monotonic and continuous immediate consequence operator. For that purpose we order ranked structures by the trivial extension of Fitting's ordering:

$$\mathcal{A} \preceq_F \mathcal{B} \text{ iff } A_i^+ \subseteq B_i^+ \text{and } A_i^- \subseteq B_i^- \text{ for all } i \in \mathbb{N}$$

It is easy to see that ranked structures are a cpo with respect to $\preceq_F$, where bottom is the sequence of infinite pairs of empty sets, considering as least upper bound for an infinite increasing chain the level-by level union of positive and negative parts of all of them. We define an immediate consequence operator in the following way:

5.5 Definition Let P be a program and $\mathcal{A}$ a ranked structure, $T_P(\mathcal{A}) = \mathcal{B}$ is the ranked structure defined for each $n \in \mathbb{N}$ by means of two operators $\langle T_P^+, T_P^- \rangle$:

$$B_n^+ = T_P^+(A_n^{+\forall} \cup A_{n-1}^{-\forall}) \qquad B_n^- = T_P^-(A_{n-1}^{\forall})$$

where we assume $A_{-1}^+ = A_{-1}^- = \emptyset$ by convention, and $\langle T_P^+, T_P^- \rangle$ are defined over sets of formulae of the form $(c \rightarrow l)^\forall$ in the following way:

$T_P^+(C) = \{a \square c \setminus \exists \{a :- \overline{l}^k \square d^k\}_{1 \le k \le m} \in P$ (properly renamed) and there exist constraints c^k such that c is satisfiable and $\mathcal{FET}_\Sigma \models (c \rightarrow \bigvee_{1 \le k \le m}(d^k \wedge c^k))^\forall \wedge$
$\forall 1 \le k \le m$: $\mathcal{FET}_\Sigma \cup C \models (c^k \rightarrow \overline{l}^k)^\forall\}$

$T_P^-(C) = \{a \square c \setminus c$ is satisfiable and $\forall\, a :- \overline{l} \square d \in P$ (properly renamed)
$\mathcal{FET}_\Sigma \cup A_i^\forall \models ((c \wedge d \rightarrow \neg \overline{l})^\forall\}$

The monotonicity of logical consequence trivially implies that T_P^+ and T_P^- are monotonic with respect to $\subseteq$, hence T_P is monotonic with respect to $\preceq_F$. Moreover, as a consequence of the compactness of the logic, it can be easily shown that T_P is continuous. Therefore, $T_P \uparrow \omega(\perp)$ is its least fixpoint, where $\perp$ is the ranked structure $\langle (\emptyset, \emptyset), (\emptyset, \emptyset) \ldots \rangle$. Now, it can be shown that $\mathcal{M}_P$ and $T_P \uparrow \omega(\perp)$ coincide:

5.6 Theorem $\mathcal{M}_P = T_P \uparrow \omega(\perp)$

Sketch of the proof The inclusion $T_P \uparrow \omega(\perp) \subseteq \mathcal{M}_P$ is trivial since $\mathcal{M}_P$ includes $T_P \uparrow i(\perp)$ for every i. For the opposite inclusion two key results were used. On the one hand, we used the fact that the set of positive two-valued logical consequences of a program and the set of (positive and negative) constrained facts coincides with the corresponding set of three-valued consequences. On the other hand, we used that resolution is a refutationally complete proof method for (general, two-valued) clausal logic with (equality) constraints. It must be noted that at layers of $\mathcal{M}_P$ greater than 0 we deal with negative facts which are not Horn clauses. ■

6 Equivalence with Clark-Kunen semantics

In this section we are going to prove that our model-theoretic semantics is equivalent to the Clark-Kunen semantics. In particular, what we will prove is that, on the one hand, for every program P its least model $\mathcal{M}_P$ is also a model of Comp(P) and, on the other that $\mathcal{M}_P$ is typical in the class of models of Comp(P), in the sense that if $\mathcal{M}_P$ satisfies any constrained literal $l \square c$ then any model of Comp(P) also satisfy $l \square c$.

Now, in order to show that $\mathcal{M}_P$ is a model of Comp(P) we should first define the truth-values of any formula in any ranked three-valued structure $\mathcal{A}$, so that $\mathcal{A}$ can also be considered a "standard" three-valued structure. The key (and obvious) definition is:

$$\mathcal{A}(p\square c) = \begin{cases} \underline{t} & \text{if } \exists i\ p\square c \in A_i^+ \\ \underline{f} & \text{if } \exists i\ p\square c \in A_i^- \\ \underline{u} & \text{otherwise.} \end{cases}$$

This definition can be extended, in a direct way to any arbitrary (constrained) formula. Here, we omit this definition due to lack of space; however a similar extension, to interpret goals of CLP-programs, is made in [26].

6.1 Theorem For all programs P, $\mathcal{M}P \models \mathrm{Comp}(P)$.

Sketch of the proof One has to prove that every axiom in Comp(P) holds in $\mathcal{M}P$. For the axioms in $\mathcal{FET}$ this is trivial. For the axioms of the form:

$$\forall x(\, p(\bar{x}) \leftrightarrow \bigvee_{i=1}^{k} \exists \bar{y}^i (\, \bar{x} = \bar{t}^i \wedge \bar{l}^i \,))$$

we proceed by case-analysis of the three possible values of the right-hand side of the if-and-only-if. In each case, one has to make use of the definition of the truth-value corresponding to formulas with the connectives involved. ■

6.2 Lemma For all $n \in \mathbb{N}$:

(i) $p\square c \in M_n^+ \Rightarrow \mathrm{Comp}(P) \models (c \rightarrow p)^{\forall}$

(ii) $p\square c \in M_n^- \Rightarrow \mathrm{Comp}(P) \models (c \rightarrow \neg p)^{\forall}$

Sketch of the proof The proof, by simultaneous induction, is quite straightforward and makes use of the fact that $\mathrm{Comp}(P) \models P^{\forall}$ (see section 2). ■

A trivial consequence of this lemma is:

6.3. Theorem For any program P and any constrained literal $p\square c$:

$$\mathcal{M}P \models p\square c \Rightarrow \mathrm{Comp}(P) \models p\square c.$$

7 Compositionality

The original aim of the approach introduced in this paper was to serve as a basis for providing compositional semantic definitions for different kinds of modular program units (e.g. [4, 15]) following the approach studied in [21]. As said in the introduction, this approach consists of three steps. In the first step, one has to define and study the given units and the associated composition operations at an abstract level, making use of notions like *institutions* or *specification frames*, in terms of some categorical constructions. These notions were introduced [16, 12] to axiomatize formalisms with certain basic compositionality properties to study the structuring and modularization of specifications independently of any specific logic formalism. In particular, in [21] the kind of units introduced in [4, 15] have been defined and studied in detail at this abstract level.

In the second step, one has to define the given class of logic programs as an institution or specification frame with the needed constructions. In particular, we have been able to show that the semantics of normal logic programs introduced in this paper defines, indeed, a specification frame with all the needed categorical constructs. This implies, together with the results presented in [21], that we have (categorical) semantic definitions of various normal logic program units: open logic programs, modules and arbitrary program fragments, which are compositional and fully abstract with respect to the operations of Ω-union, module union and (standard) program union, respectively. Nevertheless, in this paper, due to lack of space to present and motivate our results adequately, we will omit their presentation. Instead, we will just present two key results for proving that our semantics defines a specification frame with the adequate constructs. Full details can be found in the long version of this paper. Finally, the

third step, still ongoing work, consists in showing that these categorical semantic definitions are equivalent to some specific concrete semantics.

If we consider as the meaning of a program its class of models, and not just the least model, then the first result is, already, a compositionality result. It is just a direct consequence of the definition of satisfaction (cf. 4.2) and, in this case, coincides essentially with the construction of *amalgamation* in this specification frame.

7.1 Fact Let P1 and P2 be two logic programs then $Mod(P1 \cup P2) = Mod(P1) \cap Mod(P2)$.

The second result, is the key for showing that the associated specification frame has a number of constructions, in particular *free constructions*, *free extensions* and *generalized free extensions*.

7.2 Theorem For any program P, Mod(P) is an upper semilattice.

Sketch of the proof The join of two ranked structures $C = \mathcal{A} \sqcup \mathcal{B}$ can be defined as follows:

If $\mathcal{A} \preceq \mathcal{B}$ then $C = \mathcal{B}$. If $\mathcal{B} \preceq \mathcal{A}$ then $C = \mathcal{A}$. Otherwise:
For all $i \leq k$:
$C_i^+ = \{a \sqcap c \,/\, \mathcal{FET} \cup P \cup (A_i^+ \cup B_i^+)^{\forall} \models (c \rightarrow a)^{\forall}\}$
$C_i^- = (A_i^- \cap B_i^-) - C_i^+$
where $k \in \mathbb{N}$ is the lowest layer such that $A_i^+ \neq B_i^+$ or $A_i^- \neq B_i^-$.
For all $i > k$:
$C_i^+ = \{a \sqcap c \,/\, \mathcal{FET} \cup P \cup (C_{i-1}^+)^{\forall} \cup (C_{i-1}^-)^{\forall} \models (c \rightarrow a)^{\forall}\}$
$C_i^- = \{a \sqcap c \,/\, \forall\ a{:-}\ \overline{1} \sqcap d \in P\colon \mathcal{FET} \cup (C_{i-1}^+)^{\forall} \cup (C_{i-1}^-)^{\forall} \models (c \wedge d \rightarrow \neg \overline{1})^{\forall}\}$ ■

We can actually prove a stronger result, namely that Mod(P) is a complete lattice. However, we have not stated and sketched a proof of the additional properties due to lack of space and because they are not especially useful in showing the existence of the above mentioned constructions.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish CICYT project COSMOS (ref. TIC95-1016-C02-01) and the UPV-project 141-226-EA209/94 TIC.

References

[1] K. R. Apt, Logic Programming, in *Handbook of Theoretical Computer Science. Vol B: Formal Models and Semantics*, chapter 10. 1990.

[2] K. R. Apt, R. N. Bol, Logic Programming and Negation: A Survey, *Journal of Logic Programming*, 19 (20), 1994, pp. 9-71.

[3] A. Bossi, M. Bugliesi, M. Gabbrielli, G. Levi, M. C. Meo, Differential Logic Programming, in *Proc. 20th POPL* (1993) pp. 359-370.

[4] A. Bossi, M. Gabbrielli, G. Levi, M. C. Meo, Contributions to the Semantics of Open Logics Programs, in *Proc. 5th Gen. Comp. Syst.* (1992) pp. 570-580.

[5] M. Barr, C. Wells, *Category Theory for Computing Science*. Prentice Hall. 1990.

[6] K. L. Clark, Negation as failure, in H. Gallaire, J. Minker (eds), *Logic and Databases*, Plenum, 1978, pp. 293-322.

[7] W. Drabent, What is failure? An approach to constructive negation, *Acta Informatica*, 32, 1995, pp. 27-59.

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

[9] M. Fitting, A Kripke-Kleene Semantics for Logic Programs, *Journal of Logic Programming*, 4, 1985, pp. 295-312.

[10] M. Fitting, Partial Models and Logic Programming, *Theoretical Computer Science*, 48, 1986, pp. 229-255.

[11] D. Chan, Constructive Negation Based on the Completed Database, in *Proc. 5th Int. Conf. and Symp. on Logic Programming*, Seattle, pp. 111-125. 1988.

[12] H. Ehrig, P. Pepper, F. Orejas, On Recent Trends in Algebraic Specification, in *Proc. ICALP'89*, Springer LNCS 372 (1989), pp. 263-288

[13] F. Fages, Constructive negation by pruning, *Journal of Logic Programming*, to appear.

[14] G. Ferrand, A. Lallouet, A compositional proof method of partial correctness for normal logic programs, in *Proc. Int. Logic Programming Symp.*, J. Lloyd, ed. (1995), 209-223.

[15] H. Gaifman, and E. Shapiro, Fully abstract compositional semantics for logics programs, in *Proc. Sixteenth Annual ACM Symp. on Principles of Programming Languages*, (1989) pp. 134-142.

[16] J. A. Goguen, R.M. Burstall, Institutions: Abstract model theory for specification and programming, *Journ. of the ACM* 39, 1 (1992) 95-146.

[17] K. Kunen, Negation in Logic Programming, *Journal of Logic Programming*, 4, 1987, pp. 289-308.

[18] J. W. Lloyd, *Foundations of Logic Programming*. Springer-Verlag. 1987.

[19] M. Maher, A logic programming view of CLP, in *Proc. 10th Int. Conf. on Logic Programing*, D.S. Warren, ed. (1993), 737-753

[20] D. Miller, A Logical Analysis of Modules in Logic Programing, in *Journal of Logic Programming*, pp. 79-108. 1989.

[21] F. Orejas, E. Pino, H. Ehrig, Institutions for Logic Programming, in *Theoretical Computer Science* 173 (1997) pp. 485-511.

[22] T. C. Przymusinski, On the declarative semantics of stratified deductive databases and logic programs, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programs*, Morgan Kaufmann, Los Altos, CA, 1988, pp. 193-216.

[23] J. C. Shepherdson, Negation as Failure II, *Journal of Logic Programming*, 3, 1985, pp. 185-202.

[24] J. C. Shepherdson, Negation in Logic Programming, in J. Minker (ed), *Foundations of Deductive Databases and Logic Programs*, Morgan Kaufmann, Los Altos, CA, 1988, pp. 19-88.

[25] J. C. Shepherdson, A sound and complete semantics for a version of negation as failure, *Theoretical Computer Science*, 65(3), 1989, pp. 343-371.

[26] P. J. Stuckey, Negation and Constraint Logic Programming, in *Information and Computation* (1995), pp. 12-23.

[27] D. van Dalen, Intuitionistic Logic, in *Handbook of Philosophical Logic*, D. Gabbay, F. Guenthner (eds), Vol III, 1986, pp. 225-339.

[28] J.-H. You, L. Y, Yuan, A Three-Valued Semantics for Deductive Database and Logic Programs, *Journal of Computer and Systems Sciences*, 49, 1994, pp. 334-361.

[29] J.-H. You, L. Y, Yuan, On the equivalence of semantics for normal logic programs, *Journal of Logic Programming*, 1995, pp. 211-222.