Extending Logic Programming

Els Laenens
Philips Int B V, PASS/AIT
Eindhoven, The Netherlands

Domenico Saccà
Dipartimento di Sistemi,
Università della Calabria, Rende, Italy

Dirk Vermeir
Department of Mathematics and Computer Science,
University of Antwerp UIA, Wilrijk, Belgium

Abstract
An extension of logic programming, called "ordered logic programming", which includes some abstractions of the object-oriented paradigm, is presented. An ordered program consists of a number of modules (objects), where each module is composed by a number of rules possibly with negated head predicates. A sort of "isa" hierarchy can be defined among the modules in order to allow for rule inheritance. Therefore, every module sees its own rules as local rules and the rules of the other modules to which it is connected by the "isa" hierarchy as global rules. In this way, as local rules may hide global rules, it is possible to deal with default properties and exceptions. This new approach represents a novel attempt to combine the logic paradigm with the object-oriented one in knowledge base systems. Moreover, this approach provides a new ground for explaining some recent proposals of semantics for classical logic programs with negation in the rule bodies and gives an interesting semantics to logic programs with negated rule heads.

1. Introduction
In this paper we present an extension of logic programming, called ordered logic programming, that includes, besides to classical inference mechanisms, object-oriented abstractions and amenities for non-monotonic reasoning. Ordered logic programs are partially-ordered sets of "traditional" logic programs (called components) where negation may also occur in the rule heads and were first introduced in [LV].

An ordered program is a multiple possibly contradictory representation of the knowledge, one for each component of it, that can be thought of as a module or an object. It follows that there are as many meanings as components. Given a component $C_i$, the meaning of $C_i$ is given by the rules in $C_i$ as well as the rules in $C_j$ with $C_i \leq C_j$. In other words a program inherits rules from other programs, possible contradiction is removed in two different ways by overruling and by defeating.

$$P_1 = \langle C_1, C_2 \rangle, \ [C_1 < C_2],$$

where

$$C_2 = \{$$

bird(penguin),

bird(pigeon),

fly(X) - bird(X),

\neg ground\_animal(X) - bird(X) \}$$

$$C_1 = \{$$

ground\_animal(penguin),

\neg fly(X) - ground\_animal(X) \}$$

Fig 1. Ordered program $P_1$ with overruling

The process of overruling is strongly related to the fact that, as we do allow negation also in the head of predicates, contradictory information could flow around if not blocked. A rule of a component $C_i$ gets overruled in the component $C_j$ with $C_i < C_j$, if it introduces some contradiction in $C_i$ (see the ordered program $P_1$ of Figure 1 - the penguin does not fly since some rules in $C_2$ are overruled in $C_1$).

The second way to avoid contradicting information flow is to defeat the whole information. This happens when the component $C_i$ inherits contradicting rules from two other com-
\[ P_2 = \langle \{C_1, C_2, C_3\}, \{C_1 < C_2, C_1 < C_3\} \rangle, \]

where

\[ C_3 = \{
\text{rich (nnmno)} \rightarrow \neg \text{poor}(X) - \text{rich}(X)\}
\]

\[ C_2 = \{
\text{poor (nnmno)} \rightarrow \neg \text{rich}(X) - \neg \text{poor}(X)\}
\]

\[ C_1 = \{
\text{free_ticket}(X) - \text{poor}(X)\}
\]

Fig 2 Ordered program \(P_2\) with defeating components \(C_2\) and \(C_4\). In this case, both pieces of information are defeated (see the ordered program \(P_2\) of Figure 2 - we cannot establish whether \text{nnmno} is to receive a free ticket as from the point of view of \(C_1\), \(C_3\) cannot be trusted better than \(C_2\) or vice versa). It turns out that the meaning of a program may be partial,

\[ P_2 = \langle \{C_1, C_2, C_3, C_4\}, \{C_1 < C_2, C_1 < C_3, C_3 < C_4\} \rangle, \]

where

\[ C_2 = \{
\text{take_loan} \leftarrow \text{inflation}(X), X > 11\}
\]

\[ C_4 = \{
\neg \text{take_loan} \leftarrow \text{loan_rate}(X), X > 14\}
\]

\[ C_3 = \{
\text{take_loan} \leftarrow \text{inflation}(X), \text{loan_rate}(Y), X > Y + 2\}
\]

\[ C_1 = \{
\}
\]

Fig 3 Loan program

The flavor of the language, as given so far, confirms that it is a powerful attempt to include object-oriented mechanisms, notably, inheritance and default values, into logic programming. Our claim is that application domains of logic programming are extended by the proposed approach for it is possible to represent uncertain knowledge as required in advanced knowledge base applications. Consider the program in Figure 3, it models a situation where \text{myself} (component \(C_1\), that is empty for now) has taken some knowledge on loan procedures from three experts, where the knowledge of \text{Expert2} (component \(C_2\)) is independent from those of the other two experts. On the other side, \text{Expert3} (component \(C_3\)) has refined the knowledge of \text{Expert4} (component \(C_4\)). Obviously, as no rule can be actually fired, no inference is possible at \text{myself} level. Suppose now that the rule

\[ \text{inflation} \ (12) \]

is stated at \text{myself} level. Then it is possible to infer from \text{Expert2} that \text{take_loan} is true. Suppose now that, at \text{myself} level, the following two rules are instead defined

\[ \text{inflation} \ (12) \]

then, as the conflicting information \text{take_loan} and \(\neg \text{take_loan}\) should be inferred, both pieces of information are defeated and nothing can be said about taking loans at \text{myself} level. Suppose finally that the two rules defined at \text{myself} level are the following

\[ \text{inflation} \ (19) \]

\[ \text{loan_rate} \ (16) \]

Then the rule of \text{Expert4} is overruled by the rule of \text{Expert3}, as there is no conflicting information coming from \text{Expert2} and \text{Expert3}, \text{takeloan} is inferred at \text{myself} level.

In this paper we elaborate the declarative model-theoretic semantics of ordered programs. A nice feature of this semantics is that it is able to capture the stable model semantics for classical logic programming with negation on rule bodies [GL1, SZ]. This confirms that our extension of logic programming is well founded.

The paper is organized as follows. The semantics of ordered programs is described in Section 2. In Section 3 we show that the semantics of ordered programs provides a new framework to explain the semantics of classical logic programs with negation. In Section 4, we show that, as a particular case of ordered program, a logic program with negated head rules can be equipped with an interesting semantics where the rules with negated heads play the role of exceptions to general rules. We present the conclusion and discuss further work in Section 5.

2 Ordered Programs

Let us first introduce the basic concepts and notations of our language.

We suppose that a (possibly infinite) number of constants, variables and symbols are available. The basic tokens of the language are terms, predicates and literals. A term is recursively defined as a variable, a constant or \(f(t_1, \ldots, t_n)\), where \(t_1, \ldots, t_n\) are terms and \(f\) is a function symbol. A predicate is a formula of the language that is of the form \(p(t)\), where \(p\) is a predicate symbol witharity \(n\) (\(n \geq 0\)) and \(t\) is a sequence of \(n\) terms (arguments of the predicate). A literal is either a predicate (positive literal) or its negation (negative literal). A term, predicate or literal is ground if it is variable free.

Two literals are complementary if they are of the form \(A\) and \(\neg A\), for some predicate \(A\). In general, given a literal \(A\) and a set of ground literals \(X\), \(\neg A\) denotes the complement of \(A\) and \(\neg X\) denotes the set of literals \(\langle \neg B \mid B \in X \rangle\). Moreover, \(X^+\) (resp. \(X^-\)) denotes the set of all positive (resp. negative) literals in \(X\). Finally, we say that \(X\) is consistent if there are not two literals \(A\) and \(B\) in \(X\) such that \(A = \neg B\).

A negative rule (or, simply, a rule) is a formula of the language represented with the usual Prolog's notation [LJ] as follows.
where \( Q_0 \), \( Q_1 \), \( Q_m \) are literals, \( Q_0 \) is the head of the rule, and \( Q_1 \), \( Q_m \) is the body of the rule. If \( Q_0 \) is positive then the rule is a seminegative rule; moreover, if also \( Q_1 \), \( Q_m \) are all positive then the rule is a positive rule (or Horn clause). Given a rule \( r \), \( H(r) \) denotes the head of \( r \) and \( B(r) \) denotes the set of all literals in the body of \( r \). A rule is a fact if it has an empty body and is ground if it is variable free.

A negative program is a set of rules. If all rules are seminegative (resp., positive) then the program is called a seminegative program (resp., positive program).

Let \( P \) be a negative program. The Herbrand's Universe of \( P \) (denoted by \( H_P \)) is the set of all possible ground terms recursively constructed by using constants and function symbols occurring in \( P \). The Herbrand's Base of \( P \) (denoted by \( B_P \)) is the set of all possible ground predicates whose predicate symbols occur in \( P \) and whose arguments are elements of \( H_P \). A ground instance of a rule \( r \) in \( P \) is a rule obtained from \( r \) by replacing every variable \( X \) in \( r \) by \( \phi(X) \), where \( \phi \) is a mapping from the set of all variables occurring in \( P \) to \( B_P \). The set of all ground instances of all rules in \( P \) is denoted by \( ground(LP) \). An interpretation for \( P \) is any consistent subset of \( B_P \) \( \cup \) \( B_P \).

Let \( I \) be an interpretation for a negative program \( P \), then \( T \) denotes the set of predicates \( \{ A \mid A \in B_P \} \), and neither \( A \) nor \( \neg A \) is in \( I \). Note that, according to the interpretation \( I \), a ground literal is true if and only if it is member of \( I \). Therefore, \( T \) contains all the elements of the Herbrand base for which no value has been assigned in the interpretation (undefined elements) [PB, P3, SZ]. An interpretation is total if \( T \) is empty.

For a positive or seminegative program \( P \), those total interpretations for \( P \) that make true all rules in \( ground(P) \) are called total models and a total model \( M \) for \( P \) is minimal if there exists no other total model \( N \) for \( P \) such that \( M \) is a proper subset of \( N \). It is known that a total model exists for every positive or seminegative program, moreover, the minimal total model of a positive program is unique and represents the meaning of it [L, U]. On the other hand, the notion of model cannot be easily extended to negative programs as inconsistency can now arise because of negative head predicates and there are many alternative ways to cope with inconsistency. In this paper, the semantics of negative programs will be eventually explained as a particular case of a more general class of logic programs (ordered programs).

**Definition 1**

(a) An ordered program is a finite partially-ordered set of negative programs (called components) where "less than" is the partial order.

(b) An interpretation for an ordered program in a component \( C \) is any interpretation of \( C \), where \( C \) denotes the negative program \( \{ r \mid r \in C \} \) and \( r \leq C \).

Let \( P \) be an ordered program. The restriction of "less than" to all pairs of distinct components is denoted by "less", moreover, given two distinct components \( C \) and \( C_j \), \( C_i \leq C_j \) means that neither \( C_i \leq C_j \) nor \( C_j \leq C_i \). Throughout all examples of this paper, an ordered program \( P \) is represented by a pair \( (C, L) \) where \( C \) is the set of components and \( L \) is the relation "less". Finally, given a rule \( r \) in \( ground(C_i) \), \( C_i(r) \) denotes the component \( C_i \) containing the rule of which \( r \) is the ground instance. If a rule occurs in more than one component then we assume that it has distinct ground instances so that \( C_i \) is actually a function from ground instances to components.

Every component \( C_i \) of an ordered program can be thought of as a module or object with local rules (thus those defined in \( C_i \)) and with global rules (thus those defined in all other components \( C_j \) such that \( C_i \leq C_j \)), where local rules may override (overrule) global rules. The "less" relation is a sort of isa hierarchy for the components and provide the ground for inheritance. It turns out that a predicate can be defined in different ways in the various components, thus the program \( P \) has several meanings, one for each of its components.

**Example 1** Consider the ordered program \( P \) of Figure 1. To the best of the knowledge of \( C \), the penguin is not a ground animal and flies whereas these facts are contradicted in \( C \). On the other side, \( C \) can inherit a rule from \( C \) to infer that the pigeon flies.

**Definition 2** Given an interpretation \( I \) for \( P \) in \( C_i \), a rule \( r \) in \( ground(C_i) \) is
- applicable if \( B(r) \subseteq I \).
- applied if it is applicable and \( H(r) \in I \).
- blocked if there exists \( A \in B(r) \) such that \( \neg A \in I \).
- overruled if there exists a non-blocked rule \( \hat{r} \) in \( ground(C_i^*) \) such that \( C(\hat{r}) \subseteq C(r) \), and \( H(\hat{r}) = \neg H(r) \).
- defeated if there exists a non-blocked rule \( \hat{r} \) in \( ground(C_i^*) \) such that (i) \( C(\hat{r}) \subseteq C(r) \) or \( C(\hat{r}) = C(r) \), and (ii) \( H(\hat{r}) = \neg H(r) \).

**Example 2** Take the ordered program \( P \) of Example 1. The set \( L = \{ \text{bird(pigeon)}, \text{bird(penguin)}, \text{ground_animal(penguin)}, \neg\text{ground_animal(pigeon)}, \neg\text{fly(pigeon)}, \neg\text{fly(penguin)} \} \) is a total interpretation for \( P \) in \( C \). The ground rule

\[ \text{fly(penguin)} \rightarrow \text{bird(penguin)} \]

is applicable but it is overruled by the applied ground rule

\[ \neg\text{fly(penguin)} \rightarrow \text{ground_animal(penguin)} \]

The ground rule

\[ \neg\text{fly(pigeon)} \rightarrow \text{ground_animal(pigeon)} \]

is both blocked and non-applicable.

Consider now the ordered program \( P \) of Figure 1. The set \( L = \{ \text{bird(pigeon)}, \text{bird(penguin)}, \text{ground_animal(penguin)}, \neg\text{ground_animal(pigeon)}, \neg\text{fly(pigeon)}, \neg\text{fly(penguin)} \} \) is a total interpretation for \( P \) in \( C \). The ground rule

\[ \text{fly(penguin)} \rightarrow \text{bird(penguin)} \]

is applicable but it is overruled by the applied ground rule

\[ \neg\text{fly(penguin)} \rightarrow \text{ground_animal(penguin)} \]

The ground rule

\[ \neg\text{fly(pigeon)} \rightarrow \text{ground_animal(pigeon)} \]

is both blocked and non-applicable.

186
or $C_3$ of the ordered program $P_1$. In this case, considering $I_1$ as an interpretation for $\bar{P}_1$ in $C$, the applicable rule

$$fly(penguin) \rightarrow bird(penguin)$$

is defeated by the applied rule

$$\neg fly(penguin) \rightarrow ground_animal(penguin)$$

Also the applied rule

$$ground_animal(penguin)$$

is defeated by the applicable rule

$$\neg ground_animal(penguin) \rightarrow bird(penguin)$$

Let $P_2$ be the ordered program in Figure 2. Then the set $I_2 = \{ rich\ (mummo), poor\ (mummo) \}$ is a (non-total) interpretation of $P_3$ in $C_1$. The two ground rules

$$rich\ (mummo)$$

and

$$\neg rich\ (mummo) \rightarrow poor\ (mummo)$$

defeat each other. This means that more information is needed on mummo in order to certify whether he is rich or not.

**Definition 3** Let $P$ be an ordered program and $C$ be a component of it. An interpretation $M$ is a model for $P$ in $C$ if

- (a) for each $A \in M$, every rule $r$ with $H(r) = \neg A$ is either blocked or overruled by an applied rule and (b) for each $A \in M$, every applicable rule $r$ with $H(r) = A$ or $H(r) = \neg A$ is either overruled or defeated.

Condition (a) guarantees that either the value of a literal $A$ in a model cannot be contradicted by any rule (no matter value is assigned to any undefined element) or if it is contradicted then it is confirmed by a most specific rule (i.e., a rule in a lower component). Condition (b) says that the value of $A$ that could be inferred by some applicable rule can remain undefined only if this rule is overruled or defeated, possibly by assigning suitable values to some undefined elements.

**Example 2** Consider the ordered programs $P_1$, $\bar{P}_1$ and $P_2$ of Example 2. The interpretation $I_1$ is a model for $P_1$ in $C$. On the other side, $I_1$ is not a model for the ordered program $\bar{P}_1$ in $C$. A model for $\bar{P}_1$ in $C$ is $I_1 = \{ bird(pigeon), bird(penguin), fly(pigeon), \neg ground\_animal(pigeon) \}$, note that $fly(penguin)$ and $ground\_animal(penguin)$ are undefined. The interpretation $I_2$ is not a model for $P_2$ in $C_1$.

We are now given the ordered program $P_3$ composed by only one component, $C$, consisting of the following two rules

$$a \rightarrow b$$

$$\neg a \rightarrow b$$

We have that $\{ b \}$, $\{ \neg b \}$, $\{ a, \neg b \}$ and $\{ \}$ are models, whereas all other interpretations (including the Herbrand Base) are not.

The previous example has shown that the Herbrand Base is not necessarily a model as for traditional logic programs. Then one could suspect that not all ordered programs have a model. However, we next prove that a model always exists. To this end, we need to introduce some preliminary definitions and results.

**Definition 4** Let $P$ be an ordered program and $C$ be one of its components. Let $I$ be the family of all interpretations of $P$ in $C$. The ordered immediate transformation for $P$ in $C$ is the function $V_{P,C}(I) \rightarrow I$ defined as follows: given an interpretation $I$, $V_{P,C}(I) = \{ A \mid$ there exists a rule $r$ in $C'$ such that $H(r) = A$, $B(r) \subseteq I$ and $r$ is neither overruled nor defeated w.r.t. $I$ by any rule $\}$.\$

**Lemma 1** The transformation $V_{P,C}$ is monotone and has the least fixpoint.

**Proof** It is routine to prove that $V_{P,C}$ is monotone. But $V_{P,C}$ is monotone in the complete lattice $<I, \subseteq>$. Hence its least fixpoint exists.\$

From now on, we shall denote the least fixpoint of $V_{P,C}$ by $V_{P,C}(\emptyset)$.

**Proposition 1** Given an ordered program $P$ and a component $C$, $V_{P,C}(\emptyset)$ is a model for $P$ in $C$.

**Proof** It is easy to see that the two conditions of Definition 3 are satisfied.

Thus we have proved that a model exists for every ordered program in every of its components. A subsequent question is whether a total model always exists.

**Definition 5** Let $P$ be an ordered program and $C$ be a component of it. A model $M$ for $P$ in $C$ is

- (a) total if $M$ is empty,
- (b) exhaustive if there exists no other model $\bar{M}$ for $P$ in $C$ such that $\bar{M}$ is a proper subset of $M$.

Obviously every total model is also exhaustive but the converse is not true. As a model always exists, an exhaustive model must exist as well. On the other side, the existence of a total model is not guaranteed. For instance, no total model exists for the program $P_2$ of Example 2 in $C$. We also pointed out that it may happen that there exists a non-total exhaustive model even when there is a total one.

Finding a total model is hard even for seminegative programs. Besides such a model does not necessarily capture the "intended" semantics of the program, i.e., it may contain literals that are not derivable from the program (informally, such literals are "assumptions" as their value is, in a sense, arbitrary). For instance well-founded semantics [VRS] does not guarantee the existence of a total well-founded model.

As a consequence, we prefer to find a partial model without any assumption rather than a larger (even total) model with assumptions. Let us elaborate this issue next.

**Definition 6** Let $P$ be an ordered program, $C$ be a component of it, and $I$ be an interpretation for $P$ in $C$. A non-empty subset $X$ of $I$ is an assumption set w.r.t. $I$ if for each
A in X, every rule r in ground(C*) with H(r) = A satisfies one of the following conditions

(a) \( r \) is non-applicable, or
(b) \( r \) is overruled, or
(c) \( r \) is defeated, or
(d) \( B(r) \cap X \neq \emptyset \)

Note that the notion of assumption set was first introduced in [1, V] and is an extension of the definition of assumption set used in [SZ] that, in turn, is an extension of the notion of unfounded set given in [VRS]. The models we are going to analyze are those which do not include any assumption set.

Definition 7 Let \( P \) be an ordered program, \( C \) be a component of \( P \) and \( M \) be a model for \( P \) in \( C \). Then \( M \) is an assumption-free model for \( P \) in \( C \) if \( C \) is not an assumption set w.r.t. \( M \).

Example 4 Consider the ordered programs \( P_1, P_2 \), and \( P_3 \) of Examples 2 and 3. The model \( I \) for \( P_1 \) in \( C_1 \) as well as \( I \) for \( P_2 \) in \( C_2 \) is assumption-free. The empty set is an assumption-free model for \( P_2 \) in \( C_1 \). Take now the ordered program \( P_3 \) of Example 3. The empty set is the only assumption-free model for \( P_3 \) in \( C \).

Consider now the ordered program \( P_4 \) with a unique component \( C_1 \) consisting of the following rule:

\[ a \rightarrow b \]

The only assumption-free model of \( P_4 \) in \( C_1 \) is the empty set, this means that no ground literal is true without making some assumption. Moreover, the rule \( \neg a \rightarrow \neg b \) is not assumption-free.

Now, let \( \{
eg a, \neg b\} \) be an assumption-free program for \( P_2 \) in \( C_1 \). The model \( I_2 \) for \( P_2 \) in \( C_2 \) is as well an assumption-free model for \( P_2 \) in \( C_1 \). Take now the ordered program \( P_3 \) of Example 3. The empty set is the only assumption-free model for \( P_3 \) in \( C \).

Consider now the ordered program \( P_4 \) with a unique component \( C_1 \) consisting of the following rule:

\[ a \rightarrow b \]

The only assumption-free model of \( P_4 \) in \( C_1 \) is the empty set, this means that no ground literal is true without making some assumption. Note that the rule \( \neg a \rightarrow \neg b \) is not assumption-free.

Now, let \( \{
eg a, \neg b\} \) be an assumption-free program for \( P_2 \) in \( C_1 \). The model \( I_2 \) for \( P_2 \) in \( C_2 \) is as well an assumption-free model for \( P_2 \) in \( C_1 \). Take now the ordered program \( P_3 \) of Example 3. The empty set is the only assumption-free model for \( P_3 \) in \( C \).

Theorem 1 Let \( P \) be a program and \( C \) be a component of it,

(a) A model \( M \) for \( P \) in \( C \) is assumption-free if and only if \( T^*_{C^*}(\emptyset) = M \)

(b) \( V^*_{IC}(\emptyset) \) is an assumption-free model for \( P \) in \( C \) and is the intersection of all models for \( P \) in \( C \).

Proof (sketch) (a) Suppose that \( M \) is a model for \( P \) in \( C \) and \( T^*_{C^*}(\emptyset) = M \). We prove by contradiction that \( M \) is assumption-free. Let us assume that \( X \subseteq M \) is an assumption set with \( M \). It is easy to see that no rule in \( C^* \) is non-applicable, overruled or defeated. Hence, for each \( A \) in \( X \), every rule with \( H(r) = A \) contains at least an element of \( X \) in its body. Hence, \( M - X \) is a model (contradiction with the fact that \( M \) is the least fixpoint of \( T^*_{C^*} \).

(b) The proof that \( V^*_{IC}(\emptyset) \) is assumption-free is rather straightforward. On the other side, it can be shown that every model \( N \) for \( P \) in \( C \) is a fixpoint of \( V^*_{IC} \). Therefore,
the least fixpoint of \( V_{\mathcal{C}} \) is contained in \( N \).

It follows that \( V_{\mathcal{C}}(\emptyset) \) is the least model for \( P \) in \( C \). Let us now turn our attention to maximal assumption-free models for \( P \) in \( C \), i.e., any assumption-free model that is a proper subset of no other assumption-free model for \( P \) in \( C \).

**Definition 9** Let \( P \) be an ordered program and \( C \) a component of \( P \). A model \( M \) for \( P \) in \( C \) is **stable** if \( M \) is a maximal assumption-free model.

We point out that the uniqueness of stable models is not guaranteed.

**Example 5** Let \( P_5 \) be the ordered program \( \langle C_1, C_2 \rangle \), \( \{ C_1 < C_2 \} \rangle \) where \( C_2 \) has the following rules

\[
\begin{align*}
  a & \\
b & \\
c & \\
\end{align*}
\]

and \( C_1 \) consists of

\[
\begin{align*}
  \neg a & \rightarrow b, c \\
  \neg b & \rightarrow a \\
  \neg b & \rightarrow \neg b
\end{align*}
\]

\( \{ a, \neg b, c \} \) and \( \{ \neg a, b, c \} \) are two stable models for \( P_5 \) in \( C_1 \), whereas \( \{ c \} \) is an assumption-free (but not stable) model for \( P_5 \) in \( C_1 \).

The next result shows that every assumption-free model is an "approximation" of a total or exhaustive model in the sense that, by assigning value to some of the undefined elements and without changing the value for those literals that are already in the model, a total or exhaustive model can be achieved.

**Proposition 2** Let \( P \) be a program and \( C \) be a component of \( P \). Then every model for \( P \) in \( C \) is a (not necessarily proper) subset of an exhaustive model for \( P \) in \( C \).

**Proof (sketch)** Let \( M \) be an assumption-free model for \( P \) in \( C \). Let \( L \) be the family of all consistent subsets of \( \overline{M} \cup \overline{-M} \) such that \( M \cup L \) is a model for \( P \) in \( C \). If \( L \) is empty then \( M \) is obviously exhaustive and, then, the proposition holds. Otherwise, we select a consistent element \( L \) in \( L \), i.e., \( L \) is in \( L \) and is not a proper subset of any other element in \( L \). It is easy to see that \( M \cup L \) is an exhaustive model for \( P \) in \( C \).

It turns out that a stable model is the maximal subset of some exhaustive model that can be inferred from the rules.

3. Seminegative Programs

In this section we show that the semantics for ordered programs is able to capture the semantics for classical seminegative programs.

Let \( C \) be a seminegative program. We say that the ordered version of \( C \), denoted by \( OV(C) \), is the ordered program \( \langle \neg \mathcal{B}_C, C \rangle \rangle \) (note that the Herbrand Base \( \mathcal{B}_C \) is here considered as a set of negative rules with empty body). As it will be shown next, the component \( \neg \mathcal{B}_C \) corresponds to an explicit closed world assumption declaration \( [R] \) "every element of the Herbrand Base is false unless its truth is proved." Note that, instead of writing down all the elements in \( \mathcal{B}_C \), it is sufficient to write a rule

\[
\neg \mathcal{P}(X_1, \ldots, X_n)
\]

where \( X_1, \ldots, X_n \) are distinct variables, for each \( n \)-ary predicate symbol \( p \) occurring in \( C \). In this case the size of \( OV(C) \) is polynomially bounded in the size of \( C \) (the size of a program is the total number of symbols that occur in it).

We now show that there are close relationships between the models of \( C \) (as in classical logic programming) and the models for \( OV(C) \) in \( C \).

**Example 6** Consider the ancestor program \( C \)

\[
\mathcal{A}(X, X)
\]

\[
\mathcal{A}(X, Y) \rightarrow \mathcal{P}(X, Z), \mathcal{A}(Y, Z)
\]

where \( \mathcal{A} \) is defined through a database relation \( [U] \).

Then \( OV(C) = \langle \mathcal{C}, \mathcal{C} \rangle, \{ C < \mathcal{C} \} \rangle \), where \( \mathcal{C} \) is

\[
\neg \mathcal{P}(X, Y)
\]

\[
\neg \mathcal{A}(X, Y)
\]

Note that we have used a reduced form to write down \( \mathcal{B}_C \).

Let us now introduce the notion of 3-valued model for seminegative programs, as given in [P3]. Let \( C \) be a seminegative program and \( I \) be an interpretation of \( C \). Given a ground literal \( A \), \( \text{value}(A) \) is equal to \( T(\text{true}) \) if \( A \) is in \( I \), \( F(\text{false}) \) if \( \neg A \) is in \( I \) and \( U(\text{undefined}) \) otherwise (i.e., either \( A \) or \( \neg A \) is in \( I \)). Moreover, we assume that \( F < U < T \) and that the value of a conjunction \( J \) of ground literals is the minimal value of these literals, i.e., \( \text{value}(J) = \min_{A \in J}(\text{value}(A)) \).

If \( J \) is empty then we assume that \( \text{value}(J) = T \). Then \( I \) is a 3-valued model for \( C \) if for each rule \( r \) in \( ground(C) \), \( \text{value}(H(r)) \geq \text{value}(B(r)) \).

Note that if a 3-valued model is total then it makes true all the rules in \( ground(C) \) and every exhaustive model for \( C \) is total.

**Proposition 3** Let \( C \) be a seminegative program. Then every model for \( OV(C) \) in \( C \) is a 3-valued model for \( C \).

**Proof (sketch)** Let \( M \) be a model for \( OV(C) \) in \( C \). Obviously \( M \) is an interpretation for \( C \). Let \( r \) be any rule in \( ground(C) \). In order to prove that \( M \) is a 3-valued model for \( C \) it is sufficient to show that \( \text{value}(H(r)) \geq \text{value}(B(r)) \). Suppose that \( \text{value}(H(r)) < \text{value}(B(r)) \) otherwise the proof would be trivial. If \( \text{value}(H(r)) = U \), then \( H(r) \) is in \( \overline{M} \). Since \( r \) cannot be overruled or defeated, by definition of model for ordered programs \( r \) is not applicable, i.e., \( \text{value}(B(r)) \leq U \).

If \( \text{value}(H(r)) = F \), then \( \neg H(r) \) is in \( \overline{M} \). Since \( r \) cannot be overruled, by definition of model for ordered programs \( r \) is blocked, i.e., \( \text{value}(B(r)) = F \).

Note that the converse of Proposition 3 does not hold.
Example 7 Consider the program \( C \) consisting of the following rule

\[ p \rightarrow \neg p \]

We have that \( \{ p \} \) is a 3-valued model for \( C \) but not for \( OV(C) \) in \( C \) since the implicit rule \( \neg p \) is not overruled by a non-blocked rule. In fact, the rule of \( C \) is not applicable \( \square \)

Let us now introduce the concept of founded model as given in \([SZ]\). To this end, we apply the stability transformation condition for a seminegative program given in \([GL1]\) to 3-valued models. Given a seminegative program \( C \) and a 3-valued model \( M \) for \( C \), the positive version of \( C \) with respect to \( M \), denoted \( C_M^+ \), is the positive program obtained from \( ground(C) \) by deleting \( (a) \) each not applied rule and \( (b) \) all negative literals from the remaining rules. Then \( M \) is founded if \( T_{C_M}^+(\emptyset) = M^+ \). Moreover, \( M \) is stable when it is maximally founded (i.e., it is founded and is a proper subset of no other founded model). Note that if \( M \) is total then \( M \) is stable also according to the definition of \([GL1]\). We recall that the latter definition only refers to total models.

**Proposition 4** Let \( C \) be a seminegative program. Then every interpretation \( M \) for \( C \) is a 3-valued, founded model for \( C \) if and only if \( M \) is an assumption-free model for \( OV(C) \) in \( C \).

**Proof** (sketch) Suppose that \( M \) is an assumption-free model for \( OV(C) \) in \( C \). Then, by Proposition 3, \( M \) is a 3-valued model for \( C \). It is easy to see that \( T_{C_M}^+(\emptyset) \subseteq M^+ \). Moreover, it can be shown that \( T_{C_M}^+(\emptyset) = M^+ \) since otherwise \( M = T_{C_M}^+(\emptyset) \) would be an assumption set with respect to \( M \). Hence \( M \) is also founded. Suppose now that \( M \) is a 3-valued, founded model for \( C \). It is easy to see that the condition \( (a) \) of Definition 3 is satisfied because \( M \) is a founded model and the condition \( (b) \) of Definition 3 is satisfied because \( M \) is a 3-valued model. Hence \( M \) is a model for \( OV(C) \) in \( C \). Finally, the fact that \( M \) is assumption free derives from the fact that \( M \) is founded \( \square \)

**Corollary 1** Let \( C \) be a seminegative program. Then an interpretation \( M \) for \( C \) is a stable model for \( C \) if and only if \( M \) is a stable model for \( OV(C) \) in \( C \).

**Proof** It follows from Proposition 4 and the definitions of stable models \( \square \)

We have then shown that recent semantics for negation such as stable model semantics can be also explained in the framework of ordered programs. Nevertheless, Example 7 has point out that not all 3-valued models are captured by an ordered program. To remove this limitation, we introduce a different ordered version of a seminegative program \( C \), called the extended version of \( C \) and denoted by \( EV(C) \), that is obtained from \( OV(C) \) by adding the rule \( A \rightarrow A \), for every \( A \) in \( B_C \), to the component \( C \). The above rules are called reflexive rules. Also in this case the number of such additional rules can be dramatically reduced by writing them in a non-ground form.

**Proposition 5** Let \( C \) be a seminegative program and \( M \) be an interpretation for \( C \).

(a) \( M \) is a 3-valued model for \( C \) if and only if \( M \) is a model for \( EV(C) \) in \( C \).

(b) Every assumption-free model for \( OV(C) \) in \( C \) is an assumption-free model for \( EV(C) \) in \( C \).

(c) Every assumption-free model for \( EV(C) \) in \( C \) is a (not necessarily proper) subset of an assumption-free model for \( OV(C) \) in \( C \).

(d) \( M \) is a stable model for \( OV(C) \) in \( C \) if and only if \( M \) is a stable model for \( EV(C) \) in \( C \).

**Proof** (sketch) (a) The proof that if \( M \) is a model for \( EV(C) \) in \( C \) then \( M \) is a 3-valued model for \( C \) is similar to the proof of Proposition 3. Let us now assume that \( M \) is a 3-valued model for \( C \). We have to prove that \( M \) is a model for \( EV(C) \) in \( C \). Let \( A \) be any element in \( M \). If \( A \) is negative then for each rule \( r \) in \( ground(C) \) with \( H(r) = \neg A \), value \( (H(r)) = F \) by definition of 3-valued model. Hence \( r \) is blocked if \( A \) is positive then the fact \( \neg A \in B_C \) is overruled by the reflexive rule.

\[ A = A \]

Hence, the condition \( (a) \) of Definition 3 is satisfied. Let us now consider any element \( A \in M \). By definition of 3-valued model, for each rule \( r \) in \( ground(C) \) with \( H(r) = \neg A \), value \( (H(r)) = U \). Hence, \( r \) is not applicable and also the condition \( (b) \) of Definition 3 is satisfied. Therefore, \( M \) is a model for \( EV(C) \) in \( C \).

(b) Let \( M \) be any assumption-free model for \( OV(C) \) in \( C \). By Proposition 3, \( M \) is a 3-valued model for \( C \). By Proposition 5 (part \( a) \), \( M \) is a model for \( EV(C) \). Let \( X \) be any non-empty subset of \( M \) and \( A \) be any element in \( X \). By hypothesis, \( X \) is not an assumption set with respect to \( OV(C) \) in \( C \), so, by Definition 6, there exists an applied rule \( r \) in \( ground(C^+) \) with \( OV(C) \) such that \( H(r) = A \), \( B(r) \cup X = \emptyset \) and \( r \) is neither overruled or defeated. By definition of extended version, \( r \) is also in \( ground(C^+) \) with respect to \( EV(C) \). Moreover, \( r \) cannot be overruled or defeated by any reflexive rule. Hence, \( X \) is not an assumption set with respect to \( EV(C) \) in \( C \). It follows that every subset of \( M \) is not an assumption set with respect to \( EV(C) \) in \( C \). Since \( M \) is a stable model for \( EV(C) \) in \( C \), \( M \) is assumption-free model for \( EV(C) \) in \( C \). (c) It follows from the fact that reflexive rule cannot play any role for assumption sets.

(d) Let \( M \) be a stable model for \( OV(C) \) in \( C \). Since \( M \) is also an assumption-free model for \( OV(C) \) in \( C \) by definition of stable model, \( M \) is an assumption-free model for \( EV(C) \) in \( C \) by Proposition 5 (part \( b) \). We show that \( M \) is actually a stable model for \( EV(C) \) in \( C \). Let us then assume that \( \bar{M} \) is an assumption set for \( EV(C) \) in \( C \) such that \( M \subseteq \bar{M} \). By Proposition 5 (part \( c) \), there exists an assumption-free model \( N \) for \( OV(C) \) in \( C \) such that \( M \subseteq N \). Hence, \( M \subseteq N \) (contradiction with the fact that \( M \) is a stable model for \( OV(C) \) in \( C \)). Therefore, \( M \) is also a stable model.
for \(EV(C)\) in \(C\). Suppose now that \(M\) is a stable model for \(EV(C)\) in \(C\). We have to prove that \(M\) is also a stable model for \(OV(C)\) in \(C\). By Proposition 5 (part c), there exists an assumption-free model \(N\) for \(OV(C)\) in \(C\) such that \(M \subseteq N\). But if \(M\) were a proper subset of \(N\) then \(M\) would not be a stable model for \(EV(C)\) in \(C\) since \(N\) would be a larger assumption-free model for \(EV(C)\) in \(C\). Hence, \(M = N\) and, then, \(M\) is a stable model for \(OV(C)\) in \(C\). 

It turns out that ordered programs subsume 3-valued semantics for seminegative programs.

4 Negative Programs.

Let us now consider a negative program \(C\). A straightforward way to provide a semantics for \(C\) is to follow the approach used in the previous section and, therefore, to state that the models of \(C\) are those of \(OV(C)\) or, better, those of \(EV(C)\). The following example shows that, in this framework, negative rules do not play any constructive role.

**Example 8** Consider now the negative program \(C\) consisting of the following two rules

\[
fly(X) \quad bird(X)
\]

\[
\neg fly(X) \quad ground\_animal(X)
\]

and of a number of database facts defining \(bird\) and \(ground\_animal\). In this case, according to the two-level semantics, we cannot state anything about the flying capabilities of any ground bird.

The previous example has pointed out that two-level semantics for negative programs is rather poor since negative rules either defeat the derivation of a positive literal or just confirm the truthness of a negative literal, already asserted in the first level. We then propose a different semantics for negative programs.

Let \(C\) be a negative program. The 3-level version of \(C\), denoted by \(3V(C)\), is the ordered program \(<\neg B_C, C^+, C^->\), \(C^+ \subseteq C\), \(C^- \equiv \neg B_C\), \(C^- \equiv \neg B_C\), where \(C^+\) consists of all seminegative rules of \(C\) and all reflexive rules, and \(C^-\) contains all negative rules. Note that \(C^-\) can be thought of as a set of exceptions to the general rules of \(C^+\). We can now define the semantics of a negative program \(C\) by referring to its 3-level version \(3V(C)\). To this end, we observe that a set of literals is an interpretation of \(C\) if and only if it is an interpretation for \(3V(C)\) in \(C^-\).

**Definition 10** Let \(C\) be a negative program and \(I\) be an interpretation for \(C\)

(a) \(I\) is a model for \(C\) if \(I\) is a model for \(3V(C)\) in \(C^-\)

(b) \(I\) is an assumption-free model for \(C\) if \(I\) is an assumption-free model for \(3V(C)\) in \(C^-\)

(c) \(I\) is stable for \(C\) if \(I\) is a stable model for \(3V(C)\) in \(C^-\).

**Example 9** Consider the program \(C\) of Example 8. We have that \(3V(C) = \langle C_{\delta}, C^+, C^-\rangle\), \(C^- \equiv C^+\), \(C^- \equiv C^+\), \(C^+ \equiv C^+\), \(C^- \equiv C^-\).

where \(C_{\delta}\) in the reduced form, consists of the following rules:

\[
fly(X) \\
\neg bird(X) \\
\neg ground\_animal(X)
\]

\(C^-\) consists of all facts defining \(bird\) and \(ground\_animal\) and of the following rule:

\[
fly(X) \quad bird(X)
\]

\(C^-\) only contains the rule:

\[
\neg fly(X) \quad ground\_animal(X)
\]

According to the three-level semantics, every ground animal which is also a bird does not fly.

Consider now the negation program \(C\) consisting of the following two rules:

\[
\neg colored(X) \quad \neg colored(Y), X \neq Y
\]

\[
\neg colored(X) \quad ugly\_color(X)
\]

and of a number of database facts defining \(color\) and \(ugly\_color\). Obviously every \(ugly\_color\) is also a color. The meaning of the program is rather clear: “select exactly one of the available non-ugly colors.”

As promised in Section 2, we have provided a semantics for negative programs, where negative rules play the interesting role of exceptions to general rules. We now propose a direct semantics for a negative program which does not make any reference to ordered programs.

Let \(C\) be a negative program. A subset \(X\) of \(I^+\) is an assumption set w.r.t. \(I\) if for each \(A\) in \(X\), every rule \(r\) in \(ground(C)\) with \(H(r) = A\) satisfies one of the following conditions: either \(value(B(r)) \leq U\) or \(B(r) \cap X \neq \emptyset\). This is the definition of assumption set given in [SZ] and coincides with the definition of unfounded set [VRS] if the condition \(value(B(r)) \leq U\) is changed into \(value(B(r)) = F\).

**Definition 11** Let \(C\) be a negative program and \(I\) be an interpretation of \(C\)

(a) \(I\) is a model for \(C\) if for each rule \(r\) in \(ground(C)\), either (i) \(value(H(r)) \geq value(B(r))\) or (ii) there is an exception, i.e., \(H(r) \in I^+\) and there exists a negative rule \(\hat{r}\) in \(ground(C)\) for which both \(H(\hat{r}) = \neg H(r)\) and \(value(B(\hat{r})) = T\).

(b) \(I\) is an assumption-free model for \(C\) if no non-empty subset of \(I^+\) is an assumption set w.r.t. \(I\).

(c) \(I\) is a stable model for \(C\) if \(I\) is a maximal assumption-free model for \(C\), i.e., it is a proper subset of no other assumption-free model for \(C\).

Note that Definition 11 only uses concepts introduced in classical logic programming. For reasons of space, we state the next result without any proof.
THEOREM 2 Definitions 10 and 11 are equivalent

Thus Definition 11 can be seen as a simple, direct extension of seminominal program semantics to negative programs

5. Conclusion

We have introduced an extension of logic programming, called ordered logic programming, which includes some abstractions of the object-oriented paradigm. In fact, an ordered logic program consists of a number of modules, each module being composed by a set of rules possibly with negated head predicate. A sort of "isa" hierarchy can be defined among modules in order to provide the ground for rule inheritance. In fact, every module sees its own rules as local rules and the rules of the other modules to which it is connected in hierarchy as global rules. Therefore, as local rules may hide global rules, it is possible to deal with default properties and exceptions. Moreover, since a most specific module can be also thought of as the new version of a more general module, also versioning can be dealt with by our approach. It turns out that modules correspond to objects since they already include such concepts as methods, defaults, inheritance and version while an effective support for object identity can be easily provided (see, for instance, [K]) On the other side, because of its capability to deal with default properties and exceptions, ordered logic programming can be seen as a powerful language for non-monotonic reasoning as well as a new formalism for explaining some recent proposals of semantics for classical logic programs with negation in the rule bodies. In sum, ordered logic programs appear to be a step toward the construction of knowledge base systems of great flexibility for they include two relevant features (a) combination of logic programming and object-oriented paradigms, and (b) treatment of logic programs with negated rule heads. As for the issue (a), we note that two interesting proposals have been recently proposed in [AK] and [KL], on the other side, recent work is dealing with the issue (b) (see, for instance, [KS, GL2]) The advantage of our approach is that the two issues are treated in a unified framework.

We conclude by mentioning that further work is currently devoted to better support object identity and to extend well-founded semantics [VRS] to ordered logic programs. In this context, a proof procedure for inferring conclusions from some classes of ordered logic programs has been devised in [LV]

ACKNOWLEDGMENT This work has been carried out within the project KIWiS that is partially supported by the Commission of the European Communities in the framework of the ESPRIT program

6 References


