Extending Logic Programming

Els Laenens
Philips Intl 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*

$$P_1 = \langle \{C_1, C_2\}, \{C_1 < C_2\} \rangle,$$

where

$$\begin{aligned} C_2 = \{ & \\ & bird(penguin), \\ & bird(pigeon), \\ & fly(X) \leftarrow bird(X), \\ & \neg ground\_animal(X) \leftarrow bird(X) \} \\ C_1 = \{ & \\ & ground\_animal(penguin), \\ & \neg fly(X) \leftarrow ground\_animal(X) \} \end{aligned}$$

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, contradicting information could flow around if not blocked A rule of a component C_j gets *overruled* in the component C_i 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-

$P_2 = <\{C_1, C_2, C_3\}, \{C_1 < C_2, C_1 < C_3\}>$,

where

C_3 = {
rich(mummo)
$\neg poor(X)$ – *rich*(X)}
C_2 = {
poor(mummo)
$\neg rich(X)$ – *poor*(X)}
C_1 = {
free_ticket(X) – *poor*(X) }

Fig 2 *Ordered program* P_2 with defeating

ponents C_j and C_k In this case, both pieces of information are defeated (see the ordered program P_2 of Figure 2 - we cannot establish whether *mummo* 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

$P_2 = <\{C_1, C_2, C_3, C_4\}, \{C_1 < C_2, C_1 < C_3, C_3 < C_4\}>$,

where

C_2 = {
take_loan ← *inflation*(X), $X>11$ }
C_4 = {
$\neg$*take_loan* ← *loan_rate*(X), $X>14$ }
C_3 = {
take_loan ← *inflation*(X), *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 *myself* (component C_1 that is empty for now) has taken some knowledge on loan procedures from three experts, where the knowledge of *Expert*2 (component C_2) is independent from those of the other two experts On the other side, *Expert*3 (component C_3) has refined the knowledge of *Expert*4 (component C_4) Obviously, as no rule can be actually fired, no inference is possible at *myself* level Suppose now that the rule

inflation(12)

is stated at *myself* level Then it is possible to infer from *Expert*2 that *take_loan* is true Suppose now that, at *myself* level, the following two rules are instead defined

inflation(12)

loan_rate(16)

Then, as the conflicting information *take_loan* and $\neg$*take_loan* should be inferred, both pieces of information are defeated and nothing can be said about taking loans at *myself* level Suppose finally that the two rules defined at *myself* level are the following

inflation(19)

loan_rate(16)

Then the rule of *Expert*4 is overruled by the rule of *Expert*3, as there is no conflicting information coming from *Expert*2 and *Expert*3, *take_loan* is inferred at *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 $p(\mathbf{t})$, where p is a predicate symbol with arity n ($n \geq 0$) and $\mathbf{t}$ 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 $\{\neg B \mid B \in X\}$ 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 [L] as follows

$$Q_0 \leftarrow Q_1, \quad , Q_m$$

where Q_0, , 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 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 ϕ is a mapping from the set of all variables occurring in P to H_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 $B_P \cup \neg B_P$

Let I be an interpretation for a negative program P, then $\bar{I}$ 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, $\bar{I}$ contains all the elements of the Herbrand base for which no value has been assigned in the interpretation (*undefined elements*) [FB, P3, SZ] An interpretation is *total* if $\bar{I}$ 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 "$\leq$" is the partial order

(b) An *interpretation for* an ordered program *in* a component C_i is any interpretation of C_i^*, where C_i^* denotes the negative program $\{r \mid r \in C_j$ and $C_i \leq C_j\}$ □

Let P be an ordered program The restriction of "$\leq$" to all pairs of distinct components is denoted by "$<$", moreover, given two distinct components C_i and C_j, $C_i <> C_j$ means that neither $C_i < C_j$ nor $C_j < C_i$ 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 "$<$" Finally, given a rule r in *ground*(C_i^*), $C(r)$ denotes the component C_j 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 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 $C_i < C_j$), where local rules may hide (overrule) global rules The "$<$" 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_1 of Figure 1 To the best of the knowledge of C_2^*, the *penguin* is not a *ground animal* and flies whereas these facts are contradicted in C_1^* On the other side, C_1 can inherit a rule from C_2 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 $C(\hat{r}) < C(r)$, and $H(\hat{r}) = \neg H(r)$,
- *defeated* if there exists a non-blocked rule $\hat{r}$ in *ground*(C_i^*) such that (i) $C(\hat{r}) <> C(r)$ or $C(\hat{r}) = C(r)$, and (ii) $H(\hat{r}) = \neg H(r)$ □

Example 2 Take the ordered program P_1 of Example 1 The set $I_1 =$ {*bird*(*pigeon*), *bird*(*penguin*), *ground_animal*(*penguin*), $\neg$*ground_animal*(*pigeon*), *fly*(*pigeon*), $\neg$*fly*(*penguin*)} is a total interpretation for P_1 in C_1 The ground rule

$$fly(penguin) \leftarrow bird(penguin)$$

is applicable but it is overruled by the applied ground rule

$$\neg fly(penguin) \leftarrow ground\_animal(penguin)$$

The ground rule

$$\neg fly(pigeon) \leftarrow ground\_animal(pigeon)$$

is both blocked and non-applicable

Consider now the ordered program $\hat{P}_1 = <\{C\}, \emptyset>$, where C contains all the rules that are either in the component C_1

or C_2 of the ordered program P_1 In this case, considering I_1 as an interpretation for $\hat{P}_1$ in C, the applicable rule

$$fly(penguin) \leftarrow bird(penguin)$$

is defeated by the applied rule

$$\neg fly(penguin) \leftarrow ground\_animal(penguin)$$

Also the applied rule

$$ground\_animal(penguin)$$

is defeated by the applicable rule

$$\neg ground\_animal(penguin) \leftarrow bird(penguin)$$

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

$$rich(mimmo)$$

and

$$\neg rich(mimmo) \leftarrow poor(mimmo)$$

defeat each other This means that more information is needed on *mimmo* 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 $\overline{M}$, 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 is reconfirmed 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 3 Consider the ordered programs P_1, $\hat{P}_1$ and P_2 of Example 2 The interpretation I_1 is a model for P_1 in C_1 On the other side, I_1 is not a model for the ordered program $\hat{P}_1$ in C A model for $\hat{P}_1$ in C is $\hat{I}_1 = \{bird(pigeon), bird(penguin), fly(pigeon), \neg ground\_animal(pigeon)\}$, 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

$$a \leftarrow b$$

$$\neg a \leftarrow b$$

We have that $\{b\}$, $\{\neg b\}$, $\{a, \neg b\}$, $\{\neg a, \neg b\}$ 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 component 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 (sketch) It is routine to prove that $V_{P|C}$ is monotone But $V_{P|C}$ is monotone in the complete lattice $\langle I, \subseteq \rangle$ Hence its least fixpoint exists [T] □

From now on, we shall denote the least fixpoint of $V_{P|C}$ by $V^{\omega}_{P|C}(\emptyset)$

PROPOSITION 1 *Given an ordered program P and a component C, $V^{\omega}_{P|C}(\emptyset)$ is a model for P in C*

PROOF (sketch) It is easy to see that the two conditions of Definition 3 are satisfied □

Thus we have proved that a model exists for each 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 a component of it A model M for P in C is

(a) *total* if $\overline{M}$ is empty,

(b) *exhaustive* if there exists no other model $\hat{M}$ for P in C such that M is a proper subset of $\hat{M}$ □

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 [LV] 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 no subset of M is an assumption set w r t M □

Example 4 Consider the ordered programs P_1, $\hat{P}_1$ and P_2 of Examples 2 and 3 The model I_1 for P_1 in C_1 as well as $\hat{I}_1$ for $\hat{P}_1$ in C 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

$$a \leftarrow b$$

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 model $\{\neg a, \neg b\}$ is not assumption free in our definition Actually, it is not assumption-free even for the traditional program C_1 for it is based on some particular implicit assumption for asserting negative ground literals (e g , negation by failure [Cl], stratified negation [ABW, CH, N, VG], perfect model [P1, P2], well-founded model [VRS], stable model [GL1], etc) The model $\{\neg a, \neg b\}$ becomes the only assumption-free model of P in C_1 if we add a second component C_2 with the following two rules

$$\neg a$$

$$\neg b$$

and $C_1 < C_2$ □

As it has been sketched in the previous example and as it will be formally shown in the next section, most of recent semantics for negation are subsumed by ordered program semantics by just explicating that every negative literal is true unless it is overruled The advantage of our approach even for the case of programs with only one component is that any assumption for deriving negative literals must be explicitly declared, and, besides, it is possible to use more assumptions at the same time For instance, when defining a predicate, three different situations may arise (i) every negative literal is true unless it is overruled (classical situation), (ii) every positive literal is true unless it is overruled (in this case, the "sign" of the predicate could be changed to reduce to the previous case), and (iii) every literal is undefined unless its value is explicitly derived All the three cases can be captured by ordered programs

We now formalize the intuition that an assumption-free model only contains ground literals which can be inferred from the rules of the program

Definition 8 Let P be an ordered program, C be a component of P and M be a model for P in C The *enabled version* of $ground(C^*)$, denoted by C^e_M, is the program containing all applied rules of $ground(C^*)$ □

Let us now apply the *immediate consequence transformation* T as defined for seminegative and positive programs to C^e_M Hence, given any interpretation I for P in C, $T_{C^e_M}(I) = \{A \mid$ there exists a rule r in C^e_M such that $A = H(r)$ and $B(r) \subseteq I\}$

LEMMA 2 *Let P be an ordered program, C be a component of P and M be a model for P in C Then $T_{C^e_M}$ is monotone and has the least fixpoint, denoted by $T^{\infty}_{C^e_M}(\emptyset)$ Moreover, $T^{\infty}_{C^a_M}(\emptyset) \subseteq M$*

PROOF (sketch) It is easy to recognize that T_{C^e} is defined in the complete lattice $\langle I, \subseteq \rangle$, where I is the family of all interpretations for C^e, since no contradictions can arise in C^e, moreover, T_{C^e} is monotone It follows then T_{C^e} has the least fixpoint Finally, since for each r in C^e, $H(r)$ is in M by construction, $T^{\infty}_{C^e}(\emptyset) \subseteq M$ □

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 $T^{\infty}_{C^e}(\emptyset) = M$*

(b) *$V^{\infty}_{P|C}(\emptyset)$ 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 that $T^{\infty}_{C^e}(\emptyset) = M$ We prove by contradiction that M is assumption free Let us assume that $X \subseteq M$ is an assumption set w r t M It is easy to see that no rule in C^e 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^e} Hence M is assumption-free Suppose now that M is an assumption free model for P in C We again proceed by contradiction to prove that $T^{\infty}_{C^e}(\emptyset) = M$ Suppose then that $T^{\infty}_{C^e}(\emptyset) \neq M$ By Lemma 2, $T^{\infty}_{C^e}(\emptyset) \subset M$ Let $X = M - T^{\infty}_{C^e}(\emptyset)$ By definition of C^e, for each A in X, there exists no rule r of C^e such that $H(r) = A$ Hence X is an assumption set and we get a contradiction Hence, $T^{\infty}_{C^e}(\emptyset) = M$

(b) The proof that $V^{\infty}_{P|C}(\emptyset)$ 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 $V_{P|C}$ Therefore,

the least fixpoint of $V_{P|C}$ is contained in N □

It follows that $V^{\omega}_{P|C}(\emptyset)$ 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 it 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 $<\{C_1, C_2\}, \{C_1 < C_2\}>$ where C_2 has the following rules

$$a$$

$$b$$

$$c$$

and C_1 consists of

$$\neg a \ - \ b, c$$

$$\neg b \ - \ a$$

$$\neg b \ - \ \neg b$$

$\{a, \neg b, c\}$ 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 it 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 $\overline{M} \cup \neg\overline{M}$ 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 maximal 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 program 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 $<\{\neg B_C, C\}, \{C < \neg B_C\}>$ (note that the Herbrand Base $\neg B_C$ is here considered as a set of negative rules with empty body) As it will be shown next, the component $\neg B_C$ 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 B_C it is sufficient to write a rule

$$\neg p(X_1, \quad , X_n)$$

where $X_1, \quad , X_n$ 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

$$anc(X,X)$$

$$anc(X,Y) \ - \ parent(X,Z), \ anc(Z,Y)$$

where *parent* is defined through a database relation [U] Then $OV(C) = <\{C, \hat{C}\}, \{C < \hat{C}\}>$, where $\hat{C}$ is

$$\neg parent(X,Y)$$

$$\neg anc(X,Y)$$

Note that we have used a reduced form to write down 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 it Given a ground literal A, $value(A)$ is equal to $T(rue)$ if A is in I, $F(alse)$ if $\neg A$ is in I and $U(ndefined)$ otherwise (i e, either A or $\neg A$ is in $\overline{I}$) 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, $value(J) = \min_{A \text{ in } J}(value(A))$ If J is empty then we assume that $value(J) = T$ Then I is a *3-valued model* for C if for each rule r in $ground(C)$, $value(H(r)) \geq value(B(r))$ 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 $value(H(r)) \geq value(B(r))$ Suppose that $value(H(r)) \leq U$ otherwise the proof would be trivial If $value(H(r)) = U$, 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 $value(B(r)) \leq U$ If $value(H(r)) = F$, then $\neg H(r)$ is in M Since r cannot be overruled, by definition of model for ordered programs r is blocked, i e, $value(B(r)) = F$ □

Note that the converse of Proposition 3 does not hold

Example 7 Consider the program C consisting of the following rule

$$p \leftarrow \neg p$$

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 □

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 w r t M, denoted C_M, is the positive program obtained from *ground*(C) by deleting (a) each not applied rule and (c) all negative literals from the remaining rules Then M is *founded* if $T_{C_M}^{\infty}(\emptyset) = M^+$ 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 $T_{C_M}^{\infty}(\emptyset) \subseteq M^+$ Moreover, it can be shown that $T_{C_M}^{\infty}(\emptyset) = M^+$ since otherwise $M - T_{C_M}^{\infty}(\emptyset)$ would be an assumption set w r t 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 □

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 □

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 $A \leftarrow A$, 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 dramaticly 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

$$A \leftarrow A$$

Hence, the condition (a) of Definition 3 is satisfied Let us now consider any element A in $\overline{M}$ By definition of 3-valued model, for each rule r in ground(C) with $H(r) = \neg A$, *value*$(H(r)) \leq U$ 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 w r t $OV(C)$ in C, so, by Definition 6, there exists an applied rule r in *ground*(C^*) w r t $OV(C)$ 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*$(C^*$ w r t $EV(C)$ Moreover, r cannot be overruled or defeated by any reflexive rule Hence, X is not an assumption set w r t $EV(C)$ in C It follows that every subset of M is not an assumption set w r t $EV(C)$ in C, i e , M is an 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 by contradiction Let us then assume that $\hat{M}$ is an assumption set for $EV(C)$ in C such that $M \subset \hat{M}$ By Proposition 5 (part *c*), there exists an assumption-free model N for $OV(C)$ in C such that $\hat{M} \subseteq N$ Hence, $M \subset N$ (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 by Proposition 5 (part b) 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 no play any constructive role

Example 8 Consider now the negative program C consisting of the following two rules

$$fly(X) \leftarrow bird(X)$$

$$\neg fly(X) \leftarrow ground\_animal(X)$$

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 program

Let C be a negative program The *3-level version* of C, denoted by $3V(C)$, is the ordered program $<\{\neg B_C, C^+, C^-\}, \{C^- < C^+, C^+ < \neg B_C, C^- < \neg B_C\}>$, where C^+ contains both 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 a *stable* model 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 $3V(C) = <\{C_0, C^+, C^-\}, C_- < C^+, C_- < C^0, C_+ < C^0\}>$, where C_0, in the reduced form, consists of the following rules

$$\neg fly(X)$$

$$\neg bird(X)$$

$$\neg ground\_animal(X)$$

C^+ consists of all facts defining *bird* and *ground_animal* and of the following rule

$$fly(X) \leftarrow bird(X)$$

C^- only contains the rule

$$\neg fly(X) \leftarrow ground\_animal(X)$$

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

Consider now the negative program C consisting of the following two rules

$$colored(X) \leftarrow color(X), \neg colored(Y), X \neq Y$$

$$\neg colored(X) \leftarrow ugly\_color(X)$$

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., $H(r)$ is 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 seminegative 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 presently 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

[ABW] Apt, K, Bair, H, and Walker, A, *"Towards a Theory of Declarative Knowledge,"* Minker, J (ed), Morgan Kaufman, Los Altos, 1987, pp 89-148

[AK] Abiteboul, S, Kanellakis, P C, "Object Identity as a Query Language Primitive", *ACM SIGMOD Conf on the Management of Data,* SIGMOD Record, Vol 18, No 2, June 1989, pp 159-173

[CH] Chandra, A, Harel, D, "Horn Clauses and Generalization", *Journal of Logic Programming 2,* 1, 1985, pp 320-340

[Cl] Clark, K L, "Negation as Failure", in *Logic and Data Base, (Gallaire and Minker, eds),* Plenum Press, New York, 1978, pp 293-322

[FB] Fitting, M, Ben-Jacob, M, "Stratified and Three-valued Logic Programming Semantics", *Proc 5th Int Conf and Symp on Logic Programming,* MIT Press, Cambridge, Ma, 1988, pp 1054-1068

[GL1] Gelfond, M, Lifschitz, V, "The Stable Model Semantics for Logic Programming", *Proc 5th Int Conf and Symp on Logic Programming,* MIT Press, Cambridge, Ma, 1988, pp 1070-1080

[GL1] Gelfond, M, Lifschitz, V, "Logic Programs with Classical Negation", unpublished manuscript, September 1989

[K] KIWIs Team, "The specifications of BQM", ESPRIT Technical Report, February 1990

[KL] Kifer, M, Lausen, G, "F-Logic A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme" *ACM SIGMOD Conf on the Management of Data,* SIGMOD Record, Vol 18, No 2, June 1989, pp 134-146

[KS] Kowalski, R A, Sadri, F, "Logic Programs with Exceptions", unpublished manuscript, November 1989

[L] Lloyd, J W, *Foundations of Logic Programming,* Springer Verlag, Berlin, 1987

[LV] Laenens, E, Vermeir, D, "A Fixpoint Semantics for Ordered Logic", Journal of Logic and Computation, to appear

[N] Naqvi, S A, "A Logic for Negation in Database Systems," in *Foundations of Deductive Databases and Logic Programming, (Minker, J ed),* Morgan Kaufman, Los Altos, 1987

[P1] Przymusinski, T C, "On the Semantics of Stratified Deductive Databases and Logic Programs", *Journal of Automated Reasoning,* to appear

[P2] Przymusinski, T C, "On the Declarative and Procedural Semantics of Deductive Databases and Logic Programs", in *Foundations of Deductive Databases and Logic Programming, (Minker, J ed),* Morgan Kaufman, Los Altos, 1987, pp 193-216

[P3] Przymusinski, T C, "Well-founded models are intersections of three-valued stable models", unpublished manuscript, 1989

[R] Reiter, R, "On Closed World Databases", in *Logic and Data Base, (Gallaire and Minker, eds)*, Plenum Press, New York, 1978, pp 55-76

[SZ] Saccà, D, Zaniolo, C, "Stable models and Non-determinism for logic programs with negation", to appear in *Proc ACM Symp on Principles of Database Systems*, 1990

[T] Tarski, A "A Lattice Theoretical Fixpoint Theorem and its Application," *Pacific Journal of Mathematics 5*, 1955, pp 285-309

[U] Ullman, J D, *Principles of Database and Knowledge-Base Systems, Vol 1 and 2*, Computer Science Press, Rockville, Md, 1988

[VG] Van Gelder, A, "Negation as Failure Using Tight Derivations for Logic Programs," *Proc 3rd IEEE Symp on Logic Programming*, Springer-Verlag, 1986, pp 127-138

[VRS] Van Gelder, A, Ross, K, Schlipf, J S, "Unfounded Sets and Well-Founded Semantics for General Logic Programs", *ACM SIGMOD-SIGACT Symp on Principles of Database Systems*, March 1988, pp 221-230