Semantics for Reasoning with Contradictory Extended Logic Programs

Anastasia Analyti & Sakti Pramanik
Department of Computer Science, Michigan State University, E. Lansing, MI 48824
E-mail: analyti@cps.msu.edu, pramanik@cpswh.msu.edu

Abstract

Extended programs are normal programs extended with classical negation. Because of the classical negation in the head of the rules, an extended logic program can be contradictory. Human reasoning is often based on conflicting evidence and on assumptions which are not always valid. Our goal is to derive useful conclusions from programs that may be contradictory. We consider rules to be defaults. Rule prioritization can be viewed as a tool to specify confidence information about these defaults. We present a new semantics for extended programs with rule prioritization, called *contradiction-free semantics* (*CFS*). *CFS* is defined as the least fixpoint of a monotonic operator. Every extended program with rule prioritization has at least one *stable c-model*. We show that the *CFS* of a program *P* coincides with the least stable *c*-model of *P*. A sound and complete proof procedure to answer queries based on *CFS* is described.

1 Introduction

Extended programs provide negative information both implicitly (negation by default ~) and explicitly (classical negation ¬). Classical negation is needed: (i) in case of incomplete information, since it may not be justified for a particular information to be considered false because of absence of further information (closed world reasoning), (ii) when negative information should be inferred if some conditions are satisfied, for example, ¬light_off ← light_on, and (iii) to represent default reasoning and exceptions, for example, some of the exceptions of the general rule fly(X)← bird(X) are: ¬fly(X)← ostrich(X) and ¬fly(X)← penguin(X).

Several semantics for extended programs have been proposed in the literature [19, 10, 5, 16, 17, 18, 22, 6, 23]. Yet, these semantics are not defined for all extended programs. In [19], the *well-founded model* [20] of an extended program *P* is computed as that of a normal program after replacing every literal ¬*L* of *P* with a new atom ¬_*L*. However, the well-founded model of an extended program can be *contradictory*. For example, the well-founded model of *P*={¬*p*←¬*a*. *p*←. *b*←.} is {~*a*, ¬*p*, *p*, *b*} and because of the contradiction, *P* is not given any semantics in [19]. However, intuitively, the rule *b*← is not "suspect" for the derivation of literals *p*, ¬*p* in *P* and thus *b* should be true.

The *contradiction removal semantics* (*CRS*), defined in [16, 17], extends the *well-founded semantics* [20] and avoids contradictions brought about by closed-world assumptions (*CWAs*). For example, the *CRS* of *P*={¬*p*←¬*a*. *p*←. *b*←.} is {*p*, *b*} which is non-contradictory. Yet, the problem is not totally solved since no semantics is given to *P'* ={¬*p*←. *p*←. *b*←.} even though *b* should be true. The same arguments hold for the *argumentation*

semantics [6]. Rule prioritization can be used to specify the relative reliability of the rules. Rule prioritization is investigated in [12, 8, 1, 2, 13, 14]. Yet, negation by default is not considered in these works. In [12, 8, 13, 14], alternative semantics for *ordered logic* programs are presented. A default in an ordered logic program is a unidirectional rule. In [1,2], a default is a clause, that is, there is no distinction between the head and the body of a default rule.

A *prioritized extended program* (*PEP*) consists of a set of partially ordered rules. Every rule *r* has a corresponding set $C_r \subseteq Body_r$[1] which is called the *contrapositive set* of *r*. Intuitively, an extended program *P* is contradictory when there is a literal *L* such that both *L* and ¬*L* are derived from *P*. In this case, there is a rule *r* such that both $Body_r$ and $\neg Head_r$ are derived from *P*. The value of C_r indicates which rules are "suspect" for the contradiction. When C_r={}, only the rule *r* is "suspect". When $C_r \neq \{\}$, the rules used in the derivation of the literals in C_r are also "suspect". To facilitate this reasoning, *P* is expanded with the contrapositives *r'* of every rule *r* such that $Head_{r'} \in \{\neg L \mid L \in C_r\}$. When contradiction occurs, rule prioritization indicates the relative reliability of the "suspect" rules. We define the *contradiction-free semantics* (*CFS*) of a *PEP*. *CFS* is always defined and non-contradictory. Every *PEP* has at least one *stable c-model*. The *CFS* of a program *P* is the least[2] fixpoint of a monotonic operator and the least stable *c*-model of *P*. When the Herbrand base is finite, the complexity of computing *CFS* is polynomial w.r.t. the size of the program.

CFS extends the well-founded semantics for normal programs [20] to *PEPs*. The use of contrapositives for resolving contradictions in *CFS* has been supported by [11, 21]. Yet, in these works, rule prioritization is not considered and $C_r = Body_r$ ∀ rule *r*. Let *P* be an extended program with $C_r = Body_r$ ∀ rule *r*. Then, the *CFS* of *P* is a subset of the *generalized stable model semantics* of *P* [11], if the latter is defined. Our semantics is also related to ordered logic [8, 13]. An ordered logic program can be seen as a *PEP* which is free of default literals and C_r={} ∀ rule *r*. If *P* is an ordered logic program then the *CFS* of *P* coincides with the *skeptical c-partial model* of *P* [8] and is a subset of the *well-founded partial model* of *P* [13].

The rest of the paper is organized as follows. In Section 2, we define the *c*-models of a *PEP*. In Section 3, we define the *CFS* and stable *c*-models of a *PEP*. We show that the *CFS* of a *PEP*, *P*, coincides with the least stable *c*-model of *P*. In Section 4, we compare *CFS* with other semantics. In Section 5, we present the procedural semantics for *CFS*. Section 6 contains the concluding remarks.

2 *c*-models for Prioritized Extended Programs

Our alphabet contains a finite set of constant, predicate and variable symbols from which terms and atoms are constructed in the usual way. A *classical literal* is either an atom *A* or its

---

[1] $Body_r$ denotes the set of literals in the body of rule *r* and $Head_r$ denotes the head of rule *r*.

[2] A set *I* is the *least* element of a set **I** iff $I \in \mathbf{I}$ and $I \subseteq J$, for all $J \in \mathbf{I}$.

classical negation $\neg A$. The classical negation of a literal L is denoted by $\neg L$ and $\neg(\neg L)=L$. The symbol $\sim$ stands for negation by default and $\sim(\sim L)=L$, $\neg(\sim L)=L$. A *default literal* is denoted by $\sim L$, where L is a classical literal.

A *prioritized extended program* (*PEP*) is a tuple $P=<R_P,<_R>$. R_P is a set of rules r: $L_0 \leftarrow L_1,...,L_m,\sim L_{m+1},...,\sim L_n$, where r is a label and L_i are classical literals. Every rule r has a corresponding set $C_r \subseteq Body_r$, called the *contrapositive set* of r. The precise meaning of C_r will be given in the definitions. Intuitively, when there is a rule r such that both $Body_r$ and $\neg Head_r$ are derived from P, the value of C_r indicates the "suspects" for the contradiction. When $C_r=\{\}$, the rule r is considered incomplete[3]. When $C_r \neq\{\}$, the contradiction is considered as evidence that one of the literals in C_r was wrongly derived. Thus, the *CWAs* and/or rules used in some step of the derivation of literals in C_r are considered unreliable. To facilitate this reasoning, P is expanded with the contrapositives r' of every rule r such that $Head_{r'} \in \{\neg L|\ L \in C_r\}$.

For example, consider the program $P=\{r_1: a.\ \ r_2: b.\ \ r_3: p \leftarrow a.\ \ r_4: \neg p \leftarrow b.$ with $C_{r3}=\{\}$ and $C_{r4}=\{\}\}$. Because both a, $\neg p$ are derived in P and $C_{r3}=\{\}$, the rule r_3 is considered incomplete, i.e., r_3 should be $p \leftarrow a, \sim\neg p$. A similar argument applies to rule r_4. Thus, literals a, b can be reliably evaluated as true but the truth value of p is unknown. In contrast, consider the program $P'=\{r_1: a.\ \ r_2: b.\ \ r_3: p \leftarrow a.\ \ r_4: \neg p \leftarrow b.$ with $C_{r3}=\{a\}$ and $C_{r4}=\{b\}\}$. Since $C_{r3}=\{a\}$, the rule r_1 used for the derivation of a is considered unreliable. Similarly, the rule r_2 used for the derivation of b is considered unreliable. By expanding P' with the contrapositives r'_3: $\neg a \leftarrow \neg p$ and r'_4: $\neg b \leftarrow p$, the derivation of a, b from rules r_1, r_2 is blocked and the literals a, b, p are evaluated as unknown. The view $C_r=\{\}$ for every rule r is implicit in ordered logic [8, 13] and vivid logic [23]. The view $C_r = Body_r$ for every rule r is adopted in [11, 21]. Yet, other views such as $C_r \neq\{\}$ and $C_r \neq Body_r$ for a rule r are also possible.

The relation $<_R \subseteq R_P \times R_P$ is a strict partial order (irreflexive, asymmetric and transitive), denoting the relative reliability of the rules. Let r and r' be two rules. The notation $r<r'$ means that r is less reliable than r', that is, $r<r'$ iff $(r,r') \in <_R$. The notation $r \not< r'$ means that r is not less reliable that r'. Note that, $r \not< r$ since $<_R$ is irreflexive. Intuitively, when $Body_r$ is true, $Head_r$ is evaluated as true iff $\neg Head_r$ cannot be derived from rules with priority no lower than r. Thus, deciding if $Head_r$ is true depends only on the rules $r' \not< r$. Note that a *PEP* with $<_R=\{\}$ is an extended logic program.

The set of instantiated classical literals of P is called the *Herbrand Base* (HB_P) of P. The instantiation of a *PEP*, P, is defined as follows: The instantiation of R_P is defined the usual

[3] We say that a rule is *incomplete* if not all possible exceptions are enumerated in its body.

way. Let r_{inst} and r'_{inst} be instances of rules r and r' in P then $r_{inst}<r'_{inst}$ iff $r<r'$. In all sections but Section 5, we assume that programs have been instantiated and thus all rules are propositional. If S is a set of literals then $\sim S =_{def} \{\sim L|\ L \in S\}$ and $\neg S =_{def} \{\neg L|\ L \in S\}$.

Definition 2.1 (program expansion): Let P be a *PEP*. The *expansion* $exp(P)$ of P is also a *PEP*, defined as follows:

- For every rule r: $H \leftarrow L_1,...,L_n$ of P, $exp(P)$ contain rules $\{r'_i: \neg L_i \leftarrow L_1,...,L_{i-1},L_{i+1},...,L_n,\neg H|\ L_i \in C_r$ and $C_{r'_i}=(C_r-\{L_i\}) \cup \{\neg H\}\}$ (called *contrapositives* of r) and the rule r.
- The partial ordering of the rules of P is extended to the rules of $exp(P)$ as follows: If r and r' are two rules of P with $r<r'$ (resp. $r \not< r'$) then r and any contrapositive of r has less (resp. neither less nor more) priority than r' and any contrapositive of r'. If r and r' are contrapositives then $r \not< r'$.

Note that $exp(exp(P))=exp(P)$.

Definition 2.2 (interpretation): Let P be a *PEP*. A set $I=T \cup \sim F$ is an interpretation of P iff T and F are disjoint subsets of HB_P. An interpretation I is *consistent* iff there is no L such that both $L \in T$ and $\neg L \in T$. An interpretation I is *coherent* iff it satisfies the *coherence property*: if $L \in T$ then $\neg L \in F$.

In interpretation $I=T \cup \sim F$, T contains the *classically true* literals, $\neg T$ contains the *classically false* literals and F contains the literals *false by default*. The *coherence property* first appeared in [18] and it expresses that if a literal is classically false then it is also false by default.

Definition 2.3 (truth valuation of a literal): A literal L is true (resp. false) w.r.t. an interpretation I iff $L \in I$ (resp. $\sim L \in I$). A literal that is neither true nor false w.r.t. I, it is undefined w.r.t. I.

An interpretation I can be seen equivalently as a function from the set of ground classical literals to $\{0,1/2,1\}$, where $I(L)=1$ when L is true w.r.t. I, $I(L)=0$ when L is false w.r.t. I and $I(L)=1/2$ when L is undefined w.r.t. I. Both views of an interpretation, as a set and as a function, will be used in the paper. Note that, $I(\sim L)=1-I(L)$, for any literal L. If I is a coherent interpretation then $I(L)=1$ implies $I(\neg L)=0$. We define $I(\emptyset) =_{def} 1$ and $I(S) =_{def} \min\{I(L)|\ L \in S\}$ where S is a non-empty set of literals. The *coherence operator* (*coh*) [18] transforms an interpretation to a coherent one. Let $I=T \cup \sim F$ be an interpretation of a *PEP*. Then, $coh(I)$ is the coherent interpretation $T \cup \sim F'$, where $F'=F \cup \{L|\ \neg L \in T\}$.

In Definition 2.4, the concept of *c*-unfounded classical literal w.r.t. a rule r and interpretation I is defined. I represents the set of literals known to be true. This concept is used in the fixpoint computation of *CFS*. In particular, a rule r is used for the derivation of $Head_r$ only if $\neg Head_r$ is *c*-unfounded w.r.t. r and *CFS*. Intuitively, a literal L is *c*-unfounded w.r.t. r and I if L cannot be derived from the rules $r' \not< r$ when literals are assumed to be false as indicated in I.

Definition 2.4 (*c*-unfounded literal w.r.t. r and I): Let P be a *PEP*, r a rule and I an interpretation. A classical literal L is called *c*-unfounded w.r.t. r and I iff there is a classical literal set S s.t. $L \in S$ and $\forall H \in S$, if r' is a rule in $exp(P)$ s.t. $Head_{r'} = H$ and $r' \not< r$ then (i) $I(Body_{r'})=0$ or (ii) there is a classical literal $L' \in C_{r'}$ s.t. $L' \in S$ or (iii) there is a default literal $\sim L' \in C_{r'}$ s.t. $\neg L' \in S$.

Note that if a literal L is *c*-unfounded w.r.t. r and I then L is *c*-unfounded w.r.t. r and any interpretation $I' \supseteq I$. If a rule r is unidirectional ($C_r=\{\}$) and $I(Body_r)\neq 0$ then $Head_r$ is not *c*-unfounded w.r.t. any rule r' s.t. $r \not< r'$. Intuitively, when $C_r=Body_r$ $\forall$rule r, every rule is given higher priority than the *CWAs*. In Example 2.1, we show that this is not true when there is a literal $L \in Body_r - C_r$ for a rule r. An algorithm that decides if a literal L is *c*-unfounded w.r.t. r and I is given in the Appendix. The time-complexity of the algorithm is linear w.r.t. the size of $exp(P)$.

Example 2.1: Let P be the expanded (with contrapositives) *PEP*:
$R_P = \{r_1: fly.\quad r_2: \neg fly \leftarrow \sim bird.\quad r'_2: bird \leftarrow fly.$ with $C_{r_2}=\{\sim bird\}$, $C_{r'_2}=\{fly\}$ and $<_R=\{\}$.
Then, the literal $\neg fly$ is *c*-unfounded w.r.t. r_1 and $\emptyset$ (in Def. 2.4 take $S=\{\neg fly, \sim bird\}$). This implies that fly can be reliably derived from rule r_1. Intuitively, in this case, rule r_1 is given higher priority than the *CWA*, $\sim bird$. However, this is not the case if $C_{r_2}=\{\}$. Consider the program P': $R_{P'} = \{r_1: fly.\quad r_2: \neg fly \leftarrow \sim bird.$ with $C_{r_2}=\{\}$ and $<_R=\{\}$.
Then, the literal $\neg fly$ is not *c*-unfounded w.r.t. r_1 and $\emptyset$ since there is no S to satisfy conditions in Def. 2.4. This, intuitively, implies that r_1 is blocked and fly is evaluated as unknown.

Example 2.2: (credit confusion problem) Consider the following expanded *PEP*, P:
$R_P = \{$ /* If Ann is a foreign student (resp. teaching assistant) then she needs 12 (resp. 6) credits */

r_1: need_credits(ann,12)←foreign_stud(ann). r_2: need_credits(ann,6)←TA(ann).
r_3: TA(ann). r_4: foreign_stud(ann).
r_5: ¬need_credits(ann,6)←need_credits(ann,12).
r'_5: ¬need_credits(ann,12)←need_credits(ann,6).
with $C_{r_i}=\{\}$ for i=1,2,3,4, $C_{r_5}=\{$need_credits(ann,12)$\}$ and $C_{r'_5}=\{$need_credits(ann,6)$\}\}$
and $r_1<r_5$, $r_2<r_5$, $r_3<r_5$, $r_4<r_5$, $r_1<r'_5$, $r_2<r'_5$, $r_3<r'_5$, $r_4<r'_5$ and $r_1<r_2$.
/* Rules r_5 and r'_5 have higher priority than the other rules */

The literal ¬TA(ann) is *c*-unfounded w.r.t. r_3 and $\emptyset$ (in Def. 2.4 take $S=\{\neg$TA(ann)$\}$). So, TA(ann) can be reliably derived from rule r_3. Similarly, foreign_stud(ann) can be reliably derived from rule r_4. Since $r_1<r_2$, the literal ¬need_credits(ann,6) is *c*-unfounded w.r.t. r_2 and $\emptyset$ (in Def. 2.5 take $S=\{\neg$need_credits(ann,6), need_credits(ann,12)$\}$). So, need_credits(ann,6) can be reliably derived from rule r_2. In contrast, ¬need_credits(ann,12) is not *c*-unfounded w.r.t. r_1 and $\emptyset$. However, if P' is as P with $<_R=\{\}$ then ¬need_credits(ann,6) is not *c*-unfounded w.r.t. r_2 and $\emptyset$ in P'.

Definition 2.5 (truth valuation of a rule): Let P be a *PEP*. A rule r in $exp(P)$ is *c-true* w.r.t. an interpretation I iff: (i) $I(Head_r) \geq I(Body_r)$ or (ii) $I(Body_r)=1/2$ and $I(\neg Head_r)=1$ or (iii) $I(Body_r)=1$ and ($I(Head_r)=1/2$ or $I(\neg Head_r)=1$) and $\neg Head_r$ is not *c*-unfounded w.r.t. r and I.

Definition 2.6 (*c*-model): Let P be a *PEP*. A consistent, coherent interpretation I of P is a *c-model* of P iff every rule in $exp(P)$ is *c*-true w.r.t. I.

Example 2.3: Let P be as in Example 2.1 and M be a *c*-model of P. Then, $fly \in M$ because r_1 should be *c*-true w.r.t. M and $\neg fly$ is *c*-unfounded w.r.t. r_1 and $M \supseteq \emptyset$. In contrast, fly is not true in all models of P' of Example 2.1. The *c*-models of P' are $M_1=\{\sim bird\}$, $M_2=coh(\{fly, \sim bird\})$ and $M_3=coh(\{\neg fly, \sim bird\})$.

Example 2.4: Let P be as in Example 2.2. Then, $M=coh(\{$TA(ann), foreign_stud(ann), need_credits(ann,6), ¬need_credits(ann,12)$\})$ is a *c*-model of P. We will show that M is the unique *c*-model of P. Let M' be a *c*-model of P. Then, ¬TA(ann), ¬foreign_stud(ann), ¬need_credits(ann,6), need_credits(ann,12) are *c*-unfounded w.r.t. $M \supseteq \emptyset$ and rules r_3, r_4, r_2 and r'_5, respectively. Thus, $M \subseteq M'$. The literal need_credits(ann,12)$\notin M'$ because otherwise ¬need_credits(ann,6)$\in M'$ (need_credits(ann,6) is *c*-unfounded w.r.t. r_5 and $M \supseteq \emptyset$) and thus, M' is contradictory.

Let P be a normal program and I an interpretation as defined in [19]. In [19], a rule r is true w.r.t. I iff $I(Head_r) \geq I(Body_r)$. Since P is a normal program, the heads of the rules are atoms. Consequently, the bodies of all contrapositives of a rule r in P contain the classically negative literal $\neg Head_r$. This implies that all classically negative literals are *c*-unfounded w.r.t. any rule and I. If $I' = I \cup \{\sim\neg A \mid A$ is an atom of $P\}$ then conditions (ii) and (iii) in Def. 2.5 are not satisfied by I', for all rules in P. This implies that a rule r in P is *c*-true w.r.t. I' iff r is true w.r.t. I.

Proposition 2.1: Let P be a normal program. M is a model of P iff $M \cup \{\sim\neg A \mid A$ is an atom of $P\}$ is a *c*-model of P.

3 Contradiction-Free Semantics

In this Section, we define the *contradiction-free model*, *stable c-models* and *contradiction-free semantics* of a *PEP*, P. We define the contradiction-free model of P as the least fixpoint of a monotonic operator and we show that it is the least stable *c*-model of P.

Definition 3.1 (W_P operator): Let P be a *PEP* and J a set of literals. We define:

- $T_J(T)=\{L \mid \exists r: L \leftarrow L_1,\ldots,L_n$ in $exp(P)$ s.t. $L_i \in T \cup J$, $\forall i \leq n$ and $\neg L$ is *c*-unfounded w.r.t. r and $J\}$.
- $T(J)= \cup\{T_J \uparrow^{\alpha}(\emptyset) \mid \alpha < \omega\}$, where ω is the first limit ordinal.

- $\mathbf{F}(J)$ is the greatest set of classical literals S s.t. $\forall L \in S$, if r is a rule in $exp(P)$ with $Head_r = L$ then $J(Body_r)=0$ or $\exists L' \in Body_r$ s.t. $L' \in S$.
- $\mathbf{W}_P(J) = coh(\mathbf{T}(J) \cup \sim\mathbf{F}(J))$.

When $\neg Head_r$ is not c-unfounded w.r.t. r and J, we say that r is blocked w.r.t. J. Note that the sequence $\{\mathbf{T}_J \uparrow a\}$ is monotonically increasing (w.r.t. $\subseteq$). So, $\mathbf{T}(J)$ is the least fixpoint of $\mathbf{T}_J$. We define the transfinite sequence: $I_0=\{\}$, $I_{a+1}=\mathbf{W}_P(I_a)$ and $I_a = \cup\{I_b \mid b<a\}$ if a is a limit ordinal.

Proposition 3.1: Let P be a *PEP*. $\{I_a\}$ is a monotonically increasing (w.r.t. $\subseteq$) sequence of consistent, coherent interpretations of P.

Since $\{I_a\}$ is monotonically increasing (w.r.t. $\subseteq$), there is a smallest countable ordinal d s.t. $I_d = I_{d+1}$.

Proposition 3.2: Let P be a *PEP*. Then, I_d is a c-model of P.

Definition 3.2 (contradiction-free semantics): Let P be a *PEP*. The *contradiction-free model* of P (CFM_P) is the c-model I_d. The *contradiction-free semantics* (*CFS*) of P is the "meaning" represented by CFM_P.

In Example 3.1, we show that contrapositives are necessary in order to avoid the derivation of complementary literals in $\{I_a\}$.

Example 3.1: Consider the *PEP*, $P=\langle R_P, <_R \rangle$:

$R_P=\{r_1$: *OK_M.* r_2: $\neg rings.$ r_3: $rings \leftarrow OK\_M.$ with $C_{r_3}=Body_{r_3}\}$ and $r_1<r_2<r_3$.

Rule r_3 expresses that if machine M is OK then it rings. Rule r_2 expresses the observation that machine M does not ring. Rule r_1 expresses the assumption that machine M is OK.

The program $exp(P)$ is as follows:

$R_{exp(P)}=\{r_1$: *OK_M.* r_2: $\neg rings.$ r_3: $rings \leftarrow OK\_M.$ r'_3: $\neg OK\_M \leftarrow \neg rings.$

with $C_{r_3}=Body_{r_3}$ and $C_{r'_3}=Body_{r'_3}\}$ and $r_1<r_2<r_3$, $r_1<r_2<r'_3$.

Since *rings* is c-unfounded w.r.t. r_2 and $\emptyset$, $\neg rings \in \mathbf{T}(\emptyset)$. Since the literal $\neg OK\_M$ is not c-unfounded w.r.t. r_1 and $\emptyset$, rule r_1 is blocked w.r.t. $\emptyset$. Since OK_M is c-unfounded w.r.t. r'_3 and $\emptyset$, $\neg OK\_M \in \mathbf{T}(\emptyset)$ (derived from r'_3). So, $\mathbf{T}(\emptyset)=\{\neg rings, \neg OK\_M\}$, $\mathbf{F}(\emptyset)=\{\}$ and $\mathbf{W}_P(\emptyset)=coh(\{\neg rings, \neg OK\_M\})$. Because $\mathbf{W}_P \uparrow 2(\emptyset)=\mathbf{W}_P(\emptyset)$, it follows that $CFM_P = coh(\{\neg rings, \neg OK\_M\})$.

We will show that contrapositives are necessary in order to avoid the derivation of complementary literals. Assume that $exp(P)=P$. Then, $\neg OK\_M$ is c-unfounded w.r.t. r_1 and $\emptyset$

and thus $OK\_M \in \mathbf{W}_P(\emptyset)$. Consequently, *rings* is derived from r_3, since $\neg rings$ is c-unfounded w.r.t. r_3 and $\emptyset$. However, $\neg rings \in \mathbf{W}_P(\emptyset)$ (derived from r_2), since *rings* is c-unfounded w.r.t. r_2 and $\emptyset$. So, $\mathbf{W}_P(\emptyset)=coh(\{OK\_M, rings, \neg rings\})$ which is inconsistent.

Let P' be as P with the additional rule r_4: *rings*. Let $r_3<r_4$, expressing that r_4 is a more reliable observation than r_3. Then, $\mathbf{W}_{P'}(\emptyset)=coh(\{rings\})$ where *rings* is derived from rule r_4. Note that $\neg OK\_M$ is not c-unfounded w.r.t. r_1 and $\emptyset$ in P' and thus r_1 is blocked w.r.t. $\emptyset$. In contrast, $\neg OK\_M$ is c-unfounded w.r.t. r_1 and $\mathbf{W}_{P'}(\emptyset)$ because $Body_{r'_3}=\{\neg rings\}$ is false w.r.t. $\mathbf{W}_{P'}(\emptyset)$. Thus, $CFM_{P'} = \mathbf{W}_{P'} \uparrow 2(\emptyset)=coh(\{OK\_M, rings\})$.

Example 3.2: Let P be the program of Example 2.2. Then, $CFM_P = coh(\{$TA(ann), foreign_stud(ann), need_credits(ann,6), $\neg$need_credits(12)$\})$. If P' is as P with $<_R=\{\}$ then $CFM_{P'}=coh(\{$TA(ann), foreign_stud(ann)$\})$ which corresponds to the skeptical meaning of P'. If P' is as P with $C_r = Body_r$ $\forall r$ then $\neg$TA(ann) (resp. $\neg$foreign_stud(ann)) is not c-unfounded w.r.t. r_3 (resp. r_4) and $\emptyset$ because of the contrapositive of r_2 (resp. r_1) in $exp(P)$ So, $CFM_P=\{\}$.

Proposition 3.3: Let P be a *PEP*. The complexity of computing CFM_P is $O(|HB_P|^*|exp(P)|^2)$.

An algorithm for computing CFM_P is given in the Appendix. The contradiction-free model of a *PEP* corresponds to the skeptical meaning of the program. Other meanings can be obtained using the transformation $P/_c I$, where I is an interpretation of P. The transformation P/I is defined in [9, 19] for a normal program P. $P/_c I$ extends P/I to *PEPs*.

Definition 3.3 (transformation $P/_c I$): Let P be an expanded *PEP* and I be an interpretation of it. The program $P/_c I$ is obtained as follows:

(i) Remove from P all rules that contain in their body a default literal $\sim L$ s.t. $I(L)=1$.
(ii) Remove from P any rule r with $I(\neg Head_r)=1$.
(iii) If r is a rule in P s.t. $I(Body_r)=1$ and $I(Head_r)=1/2$ then replace r with $Head_r \leftarrow u$.
(iv) Remove from the body of the remaining rules of P any default literal $\sim L$ s.t. $I(L)=0$.
(v) Replace all remaining default literals $\sim L$ with u.
(vi) Replace every classically negative literal $\neg A$ with a new atom $\neg\_A$.

Example 3.5: Let P be as in Example 2.2 and M be as in Example 2.4. Then

$P/_c I = \{$need_credits(6)$\leftarrow$TA(mary). TA(mary). foreign_stud(mary).
$\neg$need_credits(12)$\leftarrow$need_credits(6). $\}$

The program $P/_c I$ is a non-negative program with a special proposition u. For any interpretation J, $J(u)=1/2$. When P is a normal program and M is a model of P [19], $P/_c M \equiv P/M$ since Steps (ii), (iii), and (vi) do not have any effect on $P/_c M$. We say that a model M of

P is the $least_v$ model of P iff $M(L) \leq M'(L)$ for any model M' and classical literal L of P. The $least_v$ model of P is the least fixed point of a monotonic operator given in [19].

Definition 3.4 (stable c-model): Let P be a *PEP* and M a *c*-model of P. M is a *stable c-model* of P iff $least_v(exp(P)/_c M)=M$.

Example 3.6: Let P' be as P in Example 2.2 with $<_R=\{\}$. Let

$M_1=coh(\{TA(ann), foreign\_stud(ann), need\_credits(ann,6), \neg need\_credits(12)\})$ and

$M_2=coh(\{TA(ann), foreign\_stud(ann), need\_credits(ann,12), \neg need\_credits(6)\})$.

Then, M_1 and M_2 are stable *c*-models of P'.

The program P of Example 2.2 has a unique stable *c*-model equal to CFM_P.

Let M be a stable *c*-model of P. Changing the ordering of the rules in P, the condition $least_v(exp(P)/_c M)=M$ will still be satisfied but M may not be a *c*-model of the new program. For example, let P be as in Example 2.2 and M be as in Example 2.4. Then, M is a stable *c*-model of P. If we replace $r_1<r_2$ in P with $r_2<r_1$, the condition $least_v(\text{exp}(P)/_c M)=M$ is still satisfied. However, M is not a *c*-model of the new program because $\neg$need_credits(12) becomes *c*-unfounded w.r.t. r_1 and M and thus, r_1 is not *c*-true w.r.t. M.

Proposition 3.4: Let P be a *PEP*. Then, CFM_P is a stable *c*-model of P.

Proposition 3.5: Let P be a *PEP*. Then, CFM_P is the least stable *c*-model of P.

4. Related Work

The contradiction-free semantics for *PEPs* is a generalization of the 3-*valued stable model semantics* which is defined for normal programs [19].

Proposition 4.1: Let P be a normal program. Then, M is a 3-valued stable model of P iff $M \cup \{\sim\neg A \mid A$ is an atom of $P\}$ is a stable *c*-model of P, independently of the values of C_r in P.

Proposition 4.1 implies that the contradiction-free model of a normal program P coincides with the *well-founded model* (*WFM*) of P [20].

In [10], the *answer-set semantics* of an extended program is defined as the intersection of its answer-sets. However, the answer set semantics is not defined for all extended programs and can be contradictory. Moreover, the problem of finding whether an extended program has an answer set is NP-complete [7]. The following relationship between *CFS* and answer set semantics can be shown.

Proposition 4.2: Let P be an extended program with $C_r=\{\}$ $\forall$ rule r. If $M \neq HB_P$ is an answer-set of P then $M \cup \{\sim A \mid A \notin M\}$ is a stable *c*-model of P.

Prioritization of rules is investigated in [8, 13]. An *ordered logic program* is a partially-ordered set of rules without negation by default. Even though the *c-assumption-free semantics* [8] and *assumption-free semantics* [13] are defined for all ordered logic programs, negation by default is not supported. The *skeptical c-partial model* of the program P in Example 3.1 is $\{OK\_M, rings\}$. According this model, machine M works OK and it rings even though rule

r_2:$\neg rings\leftarrow$ has higher priority than rule r_1:$OK\_M\leftarrow$. This unintuitive result is derived because in [8], the rule ordering $r' < r$ represents that rule r is an *exception* of rule r'. This corresponds in our framework with the case that $C_r=\{\}$ $\forall$ rule r. Indeed, let P' be as program P in Example 3.1 with $C_r=\{\}$ $\forall$ rule r. Then, the contradiction-free model of P' equals $coh(\{OK\_M, rings\})$. The skeptical *c*-partial model of $exp(P)$ (P expanded with contrapositives) is $\{\}$.

In [13], the *well-founded partial model* of an ordered logic program P is defined. Similarly to [8], rule ordering in [13] represents *exceptions* and not *reliability*. Another difference between the contradiction-free model and the well-founded partial model is demonstrated by the following example. The well-founded partial model of $P=\{p.\ \neg p\leftarrow q.\ \neg q.\ q.\}$ is $\{p\}$. According to this model, p is *true* even though $\neg p$ can also be derived from P. The reasoning in [13] is that the derivation of q is blocked (because of the rule $\neg p\leftarrow$) and thus, p can be reliably derived. In other words, [13] takes an ambiguity blocking approach. In our approach, ambiguities are propagated and thus, rule p is evaluated as *unknown*. Note that $CFM_P=\{\}$, independently of the values of C_r. Proposition 4.3 gives the relationship of *CFS* with ordered logic.

Proposition 4.3: Let $P = <R_P, <_R>$ be a *PEP* which is free from default literal and $C_r=\{\}$ $\forall$ rule r. Then, the set of classical literals in CFM_P is a subset of the well-founded partial.model of P [13] and coincides with the skeptical *c*-partial model of P [8].

The use of contrapositives for resolving contradictions appears in [11, 21]. Yet, in these works, rule prioritization is not considered and $C_r=Body_r$ $\forall$ rule r (all contrapositives of a rule are considered). Let P be an extended program. Giordano and Martelli [11] define a *generalized stable model* of P as a 2-valued stable model [9, 10] of the expansion of P with (i) the contrapositives of the rules and (ii) the constraints $\{\bot \leftarrow L, \neg L \mid L \in HB_P\}$. Not all programs have a generalized stable model. If $P=<R_P,<_R>$ with $<_R=\{\}$ and $C_r=Body_r$ $\forall$ rule r then every *generalized stable model* of P is a stable *c*-model of P. However, the opposite is not valid.

Let P be a normal program with constraints. Witteveen [21] defines the *strong belief revision model* (*SBRM*) of P as the *WFM* of the expansion of P with (i) the contrapositives of the rules, (ii) the rules $\{L\leftarrow \sim\neg L \mid L \in HB_P\}$, and (iii) the constraints $\{\bot \leftarrow L, \neg L \mid L \in HB_P\}$. Let $P=<R_P,<_R>$ with $<_R=\{\}$ and $C_r=Body_r$ $\forall$ rule r. We can show that if the rules $\{L\leftarrow \sim\neg L \mid L \in HB_P\}$ are removed from the expansion of P in [21] and *coherence* is enforced in the computation of the *SBRM* then when the *SBRM* of P exists, it coincides with *CFS*.

5 Procedural Semantics

In this Section, we present *SLCF*-resolution (*CF* for contradiction-free), a proof procedure to answer queries on *PEPs* based on the contradiction-free semantics. *SLCF*-resolution is inspired by the approach to constructive negation taken in *SLDFA*-resolution [3,4].

Substitutions in goals are replaced by constraints which are represented by Greek letters. The idempotent substitution $\{x_1/t_1,..., x_n/t_n\}$ corresponds to the constraint $x_1=t_1,...,x_n=t_n$. A constraint θ is *satisfiable* iff $CET \models \theta$, where CET stands for the Clark equality theory [15]. We use the letter Q to represent a list of literals. A *goal* is a formula $\leftarrow\theta,Q$, where θ is a satisfiable constraint. An *SLCF-refutation* of a goal is defined in Def. 5.1. Let $\leftarrow\theta,Q$ be a goal. $\mathbf{W}_P{}^{\uparrow a}(\emptyset) \models \exists\theta,Q$ iff there exists an *SLCF*-refutation of rank a for the goal $\leftarrow\theta,Q$. A selection function selects a literal from a goal. The selected literal of a goal is underlined, e.g., the selected literal of $\leftarrow\theta,Q,\underline{L},Q'$ is L.

Definition 5.1: Let P be a *PEP* and a be a countable ordinal. An *SLCF-refutation of rank* $a\geq 1$ for a goal G is a sequence of goals $G_1,...,G_n$ s.t. $G_1=G$, $G_n=\leftarrow\theta''$ and $\forall G_i$ one of the following is true:

1. (i) $G_i=\leftarrow\theta,Q,\underline{L}(x_1,...,x_n),Q'$ and
   (ii) $\exists$ a variant r: $L(t_1,...,t_n)\leftarrow L_1,...,L_m$ of a rule in $exp(P)$ and
   (iii) $\exists\theta'$ s.t. $\leftarrow\theta,\theta',c(\neg L)$ r-fails at rank $\leq a$ (Def. 5.3) and
   (iv) $G_{i+1}=\leftarrow\theta,\theta',(x_1=t_1,...,x_n=t_n),Q, L_1,...,L_m,Q'$.
2. $G_i=\leftarrow\theta,Q,\neg\underline{L},Q'$, where L is a classical literal and one of the following is true:
   (i) there is θ' s.t. $\leftarrow\theta,\theta',L$ fails at rank $< a$ (Def. 5.2) and $G_{i+1}=\leftarrow\theta,\theta',Q,Q'$ or
   (ii) there is an *SLCF*-computed answer θ' of rank $\leq a$ for $\leftarrow\theta,\neg L$ and $G_{i+1}=\leftarrow\theta,\theta', Q,Q'$.

The constraint θ'' restricted to the free variables of G is an *SLCF-computed answer of rank a* for G.

A goal $\leftarrow\theta,\theta',c(L)$ r-fails at rank $a+1$ iff every ground instance of θ,θ',L is c-unfounded w.r.t. r and $\mathbf{W}_P{}^{\uparrow a}(\emptyset)$. Thus, condition 1 expresses that the head L of a rule r is true if all literals in the body of the rule are true and $\neg L$ is c-unfounded w.r.t. r and *CFS*. Note that an *SLCF*-computed answer of rank $a+2$ for $\leftarrow\theta,\neg L$ is θ,θ' iff (i) $\exists\theta'$ s.t. the goal $\leftarrow\theta,\theta',L$ fails at rank $a+1$ which means that every ground instance of $\theta,\theta',L\in \mathbf{F}(\mathbf{W}_P{}^{\uparrow a}(\emptyset))$ or (ii) $\exists\theta'$ s.t. every ground instance of $\theta,\theta',\neg L\in \mathbf{W}_P{}^{\uparrow a+2}(\emptyset)$ (*coherence*). The constraint θ' in conditions 1(iii) and 2(i) of Def. 5.1 can be computed similarly to the way failed answers are computed in [3].

Definition 5.2: Let P be a *PEP* and $a\geq 1$ a countable ordinal. A goal G *fails at rank a* iff there exists a tree T s.t. (i) T has root G, (ii) no node of T has the form $\leftarrow\theta$, and (iii) $\forall$ node N, N is a goal and

1. If $N=\leftarrow\theta,Q,\underline{L}(x_1,..., x_n),Q'$ s.t. L is a classical literal then one of the following is true:
   (i) N is not the root, there is an *SLCF*-computed answer θ' of rank $<a$ for $\leftarrow\theta,\neg L(x_1,...,x_n)$, and N has children: $\leftarrow\theta,\theta_1,Q,c(L(x_1,...,x_n)),Q'$, ..., $\leftarrow\theta,\theta_m,Q,c(L(x_1,..., x_n)),Q'$, where $CET \models \theta\rightarrow\theta'\vee\theta_1\vee...\vee\theta_m$.

   (ii) For every variant r: $L(t_1,...,t_n)\leftarrow L_1,...,L_m$ of a rule in $exp(P)$ s.t. $\theta,(x_1=t_1,..., x_n=t_n)$ is satisfiable, N has a child: $\leftarrow\theta,(x_1=t_1,...,x_n=t_n), Q, L_1,..., L_m, Q'$.
2. If $N=\leftarrow\theta,Q,\neg\underline{L}(x_1,..., x_n),Q'$ s.t. L is a classical literal then one of the following is true:
   (i) There is an *SLCF*-computed answer θ' of rank $<a$ for $\leftarrow\theta, L(x_1,...,x_n)$ and N has children: $\leftarrow\theta,\theta_1,Q,\neg L(x_1,...,x_n),Q'$, ..., $\leftarrow\theta,\theta_m,Q,\neg L(x_1,...,x_n),Q'$, where $CET \models \theta\rightarrow\theta'\vee\theta_1\vee...\vee\theta_m$.
   (ii) N has a child: $\leftarrow\theta, Q, Q'$.

Definition 5.3: Let P be a *PEP*, $a\geq 1$ a countable ordinal and $G=\leftarrow\theta'',c(K)$ a goal where K is a classical literal. G *r-fails at rank a* iff there exists a tree T s.t. (i) T has root G (ii) no node of T has the form $\leftarrow\theta$, and (iii) $\forall$ node N, N is a goal and

1. If $N=\leftarrow\theta,Q,c(\underline{L}(x_1,..., x_n)),Q'$ s.t. L is a classical literal then one of the following is true:
   (i) N is not the root, there is an *SLCF*-computed answer θ' of rank $< a$ for $\leftarrow\theta,\neg L(x_1,..., x_n)$, and N has children: $\leftarrow\theta,\theta_1,Q,c(L(x_1,..., x_n)),Q'$, ..., $\leftarrow\theta,\theta_m,Q,c(L(x_1,..., x_n)),Q'$, where $CET \models \theta\rightarrow\theta'\vee\theta_1\vee...\vee\theta_m$.
   (ii) For every variant r': $L(t_1,..., t_n)\leftarrow L_1,...,L_m$ of a rule in $exp(P)$ s.t. $\theta,(x_1=t_1,..., x_n=t_n)$ is satisfiable and $r'\not\leq r$, N has a child: $\leftarrow\theta,(x_1=t_1,..., x_n=t_n),Q,L'_1,...,L'_m,Q'$, where:
      - if $L_i\in C_r$ and L_i is a classical literal then $L'_i=c(L_i)$,
      - if $L_i\in C_r$ and L_i is the default literal $\sim L'$ then $L'_i=c(\neg L')$,
      - if $L_i\notin C_r$ and L_i is a classical literal then $L'_i=L_i$,
      - if $L_i\notin C_r$ and L_i is the default literal $\sim L'$ then $L'_i=\neg L'$.
2. If $N=\leftarrow\theta,Q,\underline{L}(x_1,..., x_n),Q'$ s.t. L is a classical literal then one of the following is true:
   (i) There is an *SLCF*-computed answer θ' of rank $< a$ for $\leftarrow\theta,\neg L(x_1,..., x_n)$ and N has children: $\leftarrow\theta,\theta_1,Q,L(x_1,..., x_n),Q'$, ..., $\leftarrow\theta,\theta_m,Q,L(x_1,..., x_n),Q'$, where $CET \models \theta\rightarrow\theta'\vee\theta_1\vee...\vee\theta_m$.
   (ii) N has a child $\leftarrow\theta, Q,Q'$.

Proposition 5.1 (soundness, completeness): Let P be a *PEP*, R a selection rule, Q a list of classical and default literals and $G\leftarrow\theta,Q$ a goal. If θ' is an *SLCF*-computed answer for G then every ground instance of θ',Q is true w.r.t. CFM_P. If $Q\tau$ is ground and true w.r.t. CFM_P, and τ,θ is satisfiable then there is an *SLCF*-computed answer θ' for G s.t. $Q\tau$ is a ground instance of θ',Q. Every ground instance of θ,Q is false w.r.t. CFM_P iff the goal $\leftarrow\theta,Q$ fails.

Example 5.1: Consider the *PEP*, $P=<R_P, <_R>$:

$R_P=\{r_1: \neg q(0).\quad r_2: \neg q(1).\quad r_3: p(X).\quad r_4: \neg q(X)\leftarrow\sim s(X).\quad r_5: q(X)\leftarrow p(X).$

th C_r=$Body_r$ $\forall$ rule r} and $r_1 < r_3$. Then, the expanded program $exp(P)$ is as follows:

$exp(P)$={r_1: $\neg q(0)$. r_2: $\neg q(1)$. r_3: $p(X)$. r_4: $\neg q(X) \leftarrow \sim s(X)$. r'_4: $s(X) \leftarrow q(X)$.
r_5: $q(X) \leftarrow p(X)$. r'_5: $\neg p(X) \leftarrow \neg q(X)$. with C_r=$Body_r$ $\forall$ rule r} and $r_1 < r_3$.

An *SLCF*-refutation for $\leftarrow q(X)$ is: $G_1 = \leftarrow \underline{q}(X)$, $G_2 = \leftarrow X \neq 1, \underline{p}(X)$ (using rule r_5 in ndition 1 of Def. 5.1), $G_3 = \leftarrow X \neq 1$ (using rule r_3 in condition 1 of Def 5.1).

ie goal $\leftarrow X \neq 1, c(\neg q(X))$ r_5-fails since there is a tree satisfying the conditions of Def. 5.3.

$\leftarrow X \neq 1, c(\neg \underline{q}(X))$
| (using rule r_4 in condition 1(ii) of Def. 5.3)
$\leftarrow X \neq 1, c(\neg \underline{s}(X))$ (it is a leaf)

he goal $\leftarrow X \neq 1, c(\neg p(X))$ r_3-fails since there is a tree satisfying the conditions of Def. 5.3.

$\leftarrow X \neq 1, c(\neg \underline{p}(X))$
| (using rule r'_5 in condition 1(ii) of Def. 5.3)
$\leftarrow X \neq 1, c(\neg \underline{q}(X))$
Note that, since $r_1 < r_3$, rule r_1 does not satisfy the condition 1(ii) of Def 5.3.
Since $X \neq 1$, $X=1$ is unsatisfiable, r_2 does not satisfy the condition 1(ii) of Def 5.3.
| (using rule r_4 in condition 1(ii) of Def. 5.3)
$\leftarrow X \neq 1, c(\neg \underline{s}(X))$ (it is a leaf)

o, an *SLCF*-computed answer for $\leftarrow q(X)$ is $X \neq 1$.

Conclusions

Ve have presented a new semantics for *prioritized extended programs* (*PEP*). The semantics, alled *contradiction-free semantics* (*CFS*), is a generalization of the well-founded semantics or normal programs [20] and defined for all *PEPs*. We describe fixpoint, model theoretic and rocedural characterizations of *CFS*. The *contradiction-free model* of a program P is the least table c-model of P and it represents the skeptical "meaning" of P. Stable c-models of P epresent possible "meanings" of P. The degree of "skepticism" in *CFS* depends on the ontrapositive sets of the program rules.

Appendix

Let P be a propositional *PEP*. ***CFM***(program P) returns the contradiction-free model of P. To ompute $\mathbf{F}(I)$ and the set of c-unfounded literals w.r.t. a rule r and I, the complement set is onstructed first, as in [20]. The complexity of the algorithm is $O(|HB_P|^*|exp(P)|^2)$.

```
CFM(program P)
{ new_I={};
  repeat
    I=new_I;
```



```
    repeat          /* compute T(I) */
      for each rule r of exp(P) do
        if Body_r ⊆new_I and c-unfounded(¬Head_r, r, I) then add Head_r to new_I;
      end /* for */
    until no change in new_I;

    compl_F={};
    repeat          /* compute F(I) */
      for each rule r of exp(P) do
        if I(Body_r)≠0 and all body classical literals of r are true w.r.t I
        then add Head_r to compl_F;
      end /* for */
    until no change in compl_F;
    for each L∈HB_P do
      if L∉compl_F then add ~L to new_I;
    end /* for */
    new_I = coh(new_I);
  until I=new_I;
  return(I);
}

c-unfounded(literal L, rule r, interpretation I) /* returns TRUE if L is c-unfounded w.r.t. r, I */
{ compl_U={};
  repeat
    for each rule r' of exp(P) s.t. r' ≮r do
      if I(Body_r')≠0 and for each L∈Body_r' either L is true w.r.t. compl_U or L∉C_r'
      then add Head_r' and ~¬Head_r' to compl_U;
    end /* for */
  until no change in compl_U;
  if L∉compl_U then return(TRUE); else return(FALSE);
}
```

References

[1] S. Brass, U.W. Lipeck, *Semantics of Inheritance in Logical Object Specifications*, 2nd Intern. Conference on Deductive and Object-Oriented Databases, 1991, pp.411-430.

[2] S. Brass, U.W. Lipeck, *Bottom-Up Query Evaluation with Partially Ordered Defaults*, 3rd Intern. Conference on Deductive and Object-Oriented Databases (DOOD'93), 1993.

[3] W. Drabent, "What is failure? An approach to constructive negation", Technical Report LiTH-IDA-R-91-23, Linkoping University, August 91, Submitted to *Acta Informatica*.

[4] W. Drabent, *SLS-resolution without floundering*, 2nd Intern. Workshop on Logic programming and Non-monotonic Reasoning, 1993, pp.82-98.

[5] P.M. Dung, P. Ruamviboonsuk, *Well-Founded Reasoning with Classical Negation*, First Int. Workshop on Logic Programming and Non-monotonic Reasoning, 1991, 120-132.

[6] P.M. Dung, *An Argumentation Semantics for Logic Programs with Explicit Negation*, 10th Intern. Conference on Logic programming (ICLP'93), 1993, pp.616-630.

[7] C. Elkan, *A Rational Reconstruction on Nonmonotonic Truth Maintenance System*, Artificial Intelligence, Elsevier Science Publishers, 43(2), 1990, pp. 219-234.

[8] D. Gabbay, E. Laenens, D. Vermeir, *Credulous vs. Sceptical Semantics for Ordered Logic Programs*, Proc. of the 2nd Intern. Conference on Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann, 1991, pp. 208-217.

[9] M. Gelfond, V. Lifschitz, *The Stable Model Semantics for Logic Programming*, 5th Symposium on Logic Programming, MIT Press, 1988, pp. 1070-1080.

[10] M. Gelfond, V. Lifschitz, *Classical Negation in Logic programs and Disjunctive Databases*, New Generation Computing, 9, OHMSHA, 1991, pp. 365-385.

[11] L. Giordano, A. Martelli, *General Stable Models, Truth Maintenance and Conflict Resolution*, 7th Intern. Conference on Logic Programming, 1990, pp. 427-441.

[12] E. Laenens, D. Vermeir, *A Fixpoint Semantics for Ordered Logic*, Journal of Logic and Computation, 1(2), Oxford University Press, 1990, pp. 159-185.

[13] E. Laenens, D. Vermeir, *Assumption-free Semantics for Ordered Logic Programs: On the Relationship Between Well-founded and Stable Partial Models*, Journal of Logic and Computation, 2(2), Oxford University Press, 1992, pp. 133-172.

[14] N. Leone, P. Rullo, *Stable Model Semantics and its Computation for Ordered Logic Programs*, 10th European Conf. on Artificial Intelligence (ECAI'92), 1992, pp. 92-96.

[15] J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 2nd edition, 1987.

[16] L.M. Pereira, J.J. Alferes, J.N. Aparicio, *Contradiction Removal within Well Founded Semantics*, A. Nerode, W. Marek, V.S. Subrahmanian (eds.), First Intern. Workshop on Logic programming and Non-monotonic Reasoning, 1991, pp.106-119.

[17] L.M. Pereira, J.J. Alferes, J.N. Aparicio, *Contradiction Removal with Explicit Negation*, Applied Logic Conference, ILLC, 1992.

[18] L.M. Pereira, J.J. Alferes, *Well Founded Semantics for Logic Programs with Explicit Negation*, 10th European Conference on Artificial Intelligence, 1992, pp. 102-106.

[19] T. Przymusinski, *Well-Founded Semantics Coincides with the Three-Valued Stable Semantics*, Fundamenta Informaticae XIII, IOS Press, 1990, pp. 445-463.

[20] A. van Gelder, K.A. Ross, J.S. Schlipf, *The Well-Founded Semantics for General Logic Program*, Journal of the Association for Computing Machinery", 38(3), 1991, 620-650.

[21] C. Witteveen, *Skeptical Reason Maintenance is Tractable*, 2nd Intern. Conference on Knowledge Representation and Reasoning (KR91), 1991, pp.570-581.

[22] C. Witteveen, *Expanding Logic Programs*, Logics in AI, LNCS 633, 1992, pp. 372-390.

[23] G. Wagner, *Reasoning with Inconsistency in Extended Deductive Databases*, 2nd Intern. Workshop on Logic programming and Non-monotonic Reasoning, 1993, pp.300-315.

A Non-Deterministic Semantics for Ordered Logic Programs

F.Buccafurri
D.E.I.S
Univ.dellaCalabria, Italy

N.Leone
ISI – CNR
Rende, Italy

P.Rullo
Dip.diMatematica
Univ.dellaCalabria, Italy

Abstract

Previous researchers have proposed extensions of logic programming to support nonmonotonic reasoning. *Ordered logic programming*, first proposed in [5], achieves such a goal by allowing rules with negated heads (*true* negation) in the context of an inheritance hierarchy.
Another area of interest in logic programming is that dealing with the semantics of negation. Recent research focuses on both declarative and constructive characterizations of *stable models* [12]. A peculiarity of this semantics is that a program may have several alternative models (even none), each corresponding to a possible view of the world. This endows logic programming with the power to express don't-care non-determinism in a purely declarative framework.
In this paper we define a stable model semantics for ordered logic programs. First, we define stable models and show that all such models are minimal. Then, we give a constructive semantics and, based on that, an effective algorithm that computes one (non-deterministically chosen) stable model (if any).

1 Introduction

A current research direction in the area of logic programming is that devoted to the definition of suitable formalisms for non-monotonic reasoning [5, 6, 11, 4, 3, 7] In particular, ordered logic, first proposed in [5], is an elegant yet powerful logic that models defeasible reasoning by allowing rules with negated heads along with explicit preference criteria among such rules.

For an instance, consider the ordered logic program (simply "program" whenever no ambiguity arises) shown in Figure 1. Here, we have three *components*, each consisting of an *identifier* and a set of rules. The identifiers are *person*, *unemployed* and *tom*. Components are organized into a hierarchical structure, where *tom* is "lower" than *unemployed* which, in turn, is "lower" than *person*. Unlike traditional logic programming, rules may have negated heads. Thus, negation is considered as *true* negation, as a negative fact is true only if explicitly derived from a rule of the program (in other terms, a negative information is considered as much valuable as a positive

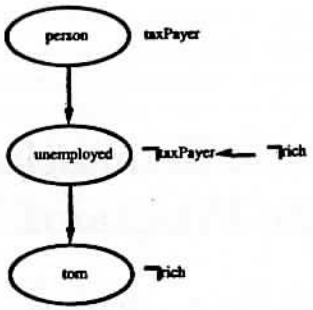

Figure 1: the program P_{tom}

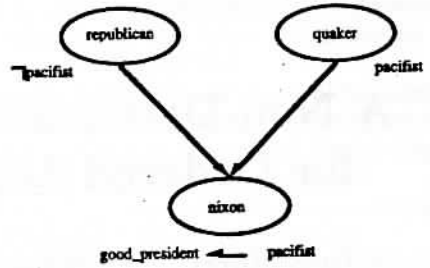

Figure 2: the program P_{nixon}

Like in the object-oriented approach, properties defined for the "higher" objects
e hierarchy flow down to the "lower" ones.
ence, the contradicting conclusion *tom both pays and does not pay taxes* should
in our case. However, this is not true. Indeed, by considering the "lower" rule
Payer $\leftarrow \neg rich$ as a sort of refinement to the first general rule, the meaning of the
am is rather clear: *tom does not pay taxes*, as he is both not rich and unemployed.
is, $\neg taxPayer \leftarrow \neg rich$ is preferred to the default rule $taxPayer \leftarrow$ as the
rchy explicitly states the specificity of the former (with respect to the object

e, ordered logic provides support to choose among several contradicting rules
xplicitly defining a preference order on them. In particular, rules specified at
lower" levels in the hierarchy are considered preferable to the more general but
adictory rules.
lowever, there are cases where conflicting pieces of information cannot be avoided.
because they arise from contradicting rules none of them is preferable to the oth-
As an example, consider the program reported in Figure 2 which is a reformulation
e well-known "Nixon Diamond". Here, we intuitively may expect two kinds of
ble behaviors. In a *pessimistic* approach, we expect the empty set to be the
el for Nixon. It is because there is no way to decide which either of *pacifist* or
ifist is to be trusted more. Contrary to this, in an *optimistic* approach, the sets
nclusions $\{pacifist, goodpresident\}$ and $\{\neg pacifist\}$ are proposed as alternative
nings to be assigned to the program.
The idea that a program may have several alternative models, each corresponding
particular view of the real world, is not new in logic programming. Indeed, the
le model semantics, introduced in [2], provides any non-positive (traditional) logic
ram with a number (possibly zero) of intended models. This actually represents
nceptual novelty w.r.t. the previous approaches for dealing with negation, such
ne *stratified* [1, 8, 10, 9] and the *well-founded* [13] semantics, both relying on the
ons of existence and uniqueness of the intended model. The lack of this two prop-
s in the stable model semantics actually increases the expressive power of logic
rams, as it endows logic languages with the power of don't-care non-determinism
purely declarative fashion. Another advantage of the SMS w.r.t. WFS is *minimal*

undefinedness, i.e., the intended model of a program has not to assign undefined value to literals that can actually be classified as true or false.
Intuitively speaking, there is a correspondence between what we called the pessimistic approach to ordered logic programs, and the well-founded semantics of logic programming. In fact, the well-founded model of a logic program is in the intersection of all partial stable models, so that a given fact is true only if it is true in every stable model. For this reason this semantics is often called the *certainty* semantics, as it provides a (possibly incomplete) view of the world where possible inconsistencies are resolved simply by assigning them an "undefined" classification. On the other side, there is a strict relationship between the so-called optimistic approach and the stable model semantics of (traditional) logic programs, as the existence of more different intended models is in both cases supported. This semantics is in fact called the *possibility* semantics, as it only requires a fact to be in any stable model in order to be true.

In this paper we provide a formalization of the "optimistic" approach in terms of a stable model semantics for ordered logic programs. The contribution of our work is twofold. First, the semantics of ordered logic programs is given as an extension of the stable model semantics as well as of other related concepts as they are defined for traditional logic programs. Second, a constructive definition of stable models is provided along with a procedure, based on a backtracking strategy, to compute (in a nondeterministic fashion) any stable model of an ordered logic program.

2 Syntax of Ordered Logic Programs

This section provides a formal description of ordered logic programs.

A *term* is either a constant or a variable. An *atom* is $p(t_1, ..., t_n)$, where p is a *predicate* of arity n and $t_1, ..., t_n$ are terms.

A *literal* is either a *positive literal* Q or a *negative literal* $\neg Q$, where Q is an atom. Two literals are *complementary* if they are of the form P and $\neg P$, for some atom P. Given a literal Q, $\neg.Q$ denotes its complementary literal. Accordingly, given a set A of literals, $\neg.A$ denotes the set $\{\neg.Q \mid Q \in A\}$.

A *rule* r is a statement of the form $H \leftarrow B$, where H is a literal (*head* of the rule) and B is a set of literals (*body* of the rule) (note that the head of a rule may be a negative literal). Given a rule r, we shall denote by $H(r)$ and $B(r)$ the head and the body of r, respectively.

A term, an atom, a literal or a rule is *ground* if no variable appears in it.

Let $(C, \leq)$ be a finite partially ordered set of symbols, called *identifiers*.

A *component* is a pair $< c, D(c) >$, where $c \in C$ and $D(c)$, the *definition of* c, is a finite set of rules.

A *knowledge base* on C is a set of components, one for each element of C.

In the following we shall denote by $<$ the reflexive-reduction of $\leq$.

Definition 1 Given a knowledge base $\mathcal{K}$ and an identifier $c \in C$, the *ordered logic program for* c is the set of components $\mathcal{P} = \{< c', D(c') > \in \mathcal{K} | c \leq c'\}$.

For simplicity, in the sequel of this paper, we shall often call an ordered logic ogram simply "program".

xample 1 Let $C = \{c, c_1, c_2, c_3, c_4\}$ be the set of object identifiers and $\{c < c_1, c < c_3 < c_2, c < c_2, c_4 < c_1\}$ the relation $<$ defined on it. A possible knowledge base C is

$$\mathcal{K} = \{< c, \{t \leftarrow \neg p\} >, < c_1, \{\neg p \leftarrow\} >, < c_2, \{\neg p \leftarrow, q \leftarrow\} >,$$

$$< c_3, \{p \leftarrow q\} >, < c_4, \{s \leftarrow\} >\}.$$

ie program $\mathcal{P}$ for the identifier c is the set consisting of the components in $\mathcal{K}$ with ntifiers c, c_1, c_2 and c_3.

Interpretations and Models

this section we assume that a knowledge base $\mathcal{K}$ is given and a component $c \in C$ s been fixed. Let $\mathcal{P}$ be the program for c.

The *Universe* $H_\mathcal{P}$ of $\mathcal{P}$ is the set of all constants appearing in the rules of $\mathcal{P}$. The se $B_\mathcal{P}$ of $\mathcal{P}$ is the set of all possible ground (i.e., variable free) literals constructible m the predicates appearing in the rules of $\mathcal{P}$ and the constants occurring in $H_\mathcal{P}$. ain, we note that $B_\mathcal{P}$ is finite (as no function symbols are allowed in the language). tice that, unlike traditional logic programming, the Base of an ordered logic pro- ms contains also negative literals; this is because both negative and positive literals treated in a uniform way in ordered logic.

A *ground instance* of r is a rule obtained from r by replacing every variable X in y $\sigma(X)$, where σ is a mapping from the variables occurring in r to the constants $H_\mathcal{P}$. We denote by $ground(\mathcal{P})$ the set of all possible ground instances of the rules $\mathcal{P}$. However, there is problem to deal with: since one rule may appear is several ferent components of $\mathcal{P}$, we require that the respective ground instances are distinct. at is, $ground(\mathcal{P})$ is a multiset. Hence, we can define a function $comp - of$ from $ound(\mathcal{P})$ onto the set C of the identifiers associating with a ground instance $\overline{r}$ of r (unique) component of r.

Given a program $\mathcal{P}$, an *interpretation* for $\mathcal{P}$ is a set I of ground literals such t $I \cap \neg.I = \emptyset$ (i.e., it is a *consistent* set of literals, in the sense that no two nplementary literals belong to I). A ground literal Q is *true* (resp., *false*) w.r.t. interpretation I if $Q \in I$ (resp., $Q \in \neg.I$). The set of literals which are neither e nor false w.r.t. I is denoted $\overline{I}$ (i.e., $\overline{I} = (B_\mathcal{P}) - (I \cup \neg.I)$). The elements of $\overline{I}$ are *defined* literals.

Let I be an interpretation for $\mathcal{P}$ and r be a rule in $ground(\mathcal{P})$. A rule $r \in ound(\mathcal{P})$ is *satisfied* in I if either the head of r is true w.r.t. I or some literal in body of r is not true w.r.t. I.

Next we introduce the concept of *model* for an ordered logic program. Unlike tra- ional logic programming, the notion of satisfiability of a rule is not sufficient for our

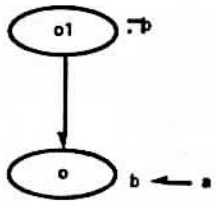

Figure 3.a.: the program P_1

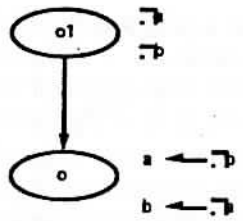

Figure 3.b.: the program P_2

purposes, in that it does not take into account the presence of explicit contradictions. Hence, we present first some preliminary definitions.

Definition 2 Let I be an interpretation for a program $\mathcal{P}$. A rule $r \in ground(\mathcal{P})$ is *overruled* in I if there exists a rule $r' \in ground(\mathcal{P})$ such that the following conditions hold: 1) $comp - of(r') < comp - of(r)$, and (2) the heads of r and r' are complementary literals, i.e., $H(r) = \neg.H(r')$ and (3) $B(r') \subseteq I$. □

Example 2 Consider the program $\mathcal{P}_{tom}$ of Figure 1, and let $I = \{\neg rich\}$ be given. It is clear that the rule r : $taxPayer \leftarrow$ is overruled by r' : $\neg taxPayer \leftarrow \neg rich$, for the given interpretation. Informally, this corresponds to the intuition that r' is "preferable" to r in that belonging to a more specific component. In other words, rule r' is considered as a sort of refinement of the more general rule r. □

Definition 3 . Let $\mathcal{P}$ be a program and I an interpretation for $\mathcal{P}$. We say that a rule $r \in ground(\mathcal{P})$ is *defeated* in I if there exists a rule r' in $ground(\mathcal{P})$ such that: (1) neither $comp - of(r) < comp - of(r')$ nor $comp - of(r') < comp - of(r)$, (2) the heads of r and r' are complementary literals, i.e., $H(r') = \neg.H(r)$, (3) $B(r') \subseteq I$ and $H(r') \in I$, and (4) r' is not overruled (in I). □

Example 3 Consider the program $\mathcal{P}_{nixon}$ of Figure 2. The rule $pacifist \leftarrow$ is defeated in $I = \{\neg pacifist\}$, while $\neg pacifist \leftarrow$ is defeated in $I = \{pacifist\}$. □

We are now ready to provide the definition of model.

Definition 4 Let M be an interpretation for a program $\mathcal{P}$. We say that M is a *model* for $\mathcal{P}$ if every rule in $ground(\mathcal{P})$ is either satisfied or overruled or defeated in M. □

Definition 5 . A *minimal model* M for $\mathcal{P}$ is a model for which there exists no other model N such that N is a proper subset of M. □

Example 4 The program $\mathcal{P}_{tom}$ has the unique minimal model $\{\neg rich, \neg taxPayer\}$.
The program $\mathcal{P}_{nixon}$ has two minimal models, namely, $\{pacifist\}$ and $\{\neg pacifist\}$.
The program $\mathcal{P}_1$ of Figure 3.a has the minimal models $\{\neg b\}$ and $\{a, b\}$.
The program $\mathcal{P}_2$ of Figure 3.b has the minimal models $\{a, \neg b\}$ and $\{b, \neg a\}$. □

Next we extend to ordered logic programs the notion of *unfounded set* defined for logic programs in [13].

Definition 6 Let a program $\mathcal{P}$ and the sets of ground literals I and X be given. X is an *unfounded set w.r.t. I* if the following holds. For each $p \in X$, for each rule $r \in ground(\mathcal{P})$ such that $H(r) = p$, either (1) $B(r)$ is false in I, or (2) $B(r) \cap X \neq \emptyset$, or (3) r is overruled in I, or (4) r is defeated in I. □

Clearly, the union of two unfounded sets of $\mathcal{P}$ w.r.t. I is an unfounded set of $\mathcal{P}$ w.r.t. I. Hence, there exists the *Greatest Unfounded Set* of $\mathcal{P}$ w.r.t. I, denoted $GUS_{\mathcal{P}}(I)$. It is immediate to recognize that $GUS_{\mathcal{P}}(I)$ is a monotonic transformation.

Proposition 1 *Let M be an interpretation for a program $\mathcal{P}$. M is a model for P iff $B_{\mathcal{P}} - M$ is an unfounded set for $\mathcal{P}$ w.r.t. M.*

Proof *if part.* We proceed by negation, assuming that M is not a model for $\mathcal{P}$ and then showing that $B_{\mathcal{P}} - M$ is not an unfounded set for $\mathcal{P}$ w.r.t. M. Indeed, if M is not a model, there exists a rule $r \in ground(\mathcal{P})$, neither overruled nor defeated in M, such that $H(r) \notin M$ and $B(r) \subseteq M$. Therefore, since $H(r)$ is in $B_{\mathcal{P}} - M$, $B_{\mathcal{P}} - M$ is not an unfounded set.

only if part. For any $p \in B_{\mathcal{P}} - M$ and rule $r \in ground(\mathcal{P})$ such that $H(r) = p$, since M is a model, we have that r is either satisfied or overruled or defeated in M. If r is satisfied then $B(r)$ is not a subset of M (as $p \notin M$), thus $B(r) \cap (B_{\mathcal{P}} - M) \neq \emptyset$. Hence, $B_{\mathcal{P}} - M$ is an unfounded set. □

4 Stable Models

We have seen in the previous section that programs admit, in general, a number of minimal models. This does not actually represent a novelty in logic programming, as it is well known that non-positive logic programs are characterized by a multiplicity of minimal models (). Various proposals on how to select those that can be considered as representative of the intended meaning of a logic program can be found in literature. In particular, the *stratified* [1, 8, 10, 9] and the *well-founded* [13] semantics provide a way to choose *one* minimal model as its intended model, while the *stable model* semantics assigns each program with *several* alternative models (possibly none).

In this section we provide a stable model semantics for ordered logic programs. We next provide a formulation of the notion of stable model which is a natural generalization of the (2-valued) stable model semantics given for traditional logic programs [2]. Before giving this new definition, we recall the *immediate consequence operator* $T_{\mathcal{P}} : 2^{B_{\mathcal{P}}} \rightarrow 2^{B_{\mathcal{P}}}$ defined as: $T_{\mathcal{P}}(I) = \{Q |\ \exists r \in ground(\mathcal{P}) \wedge H(r) = Q \wedge B(r) \subseteq I\}$. It is easy to see that $T_{\mathcal{P}}$ is a monotonic transformation in the complete lattice $< 2^{B_{\mathcal{P}}}, \subseteq >$ and, therefore, it has the least fixpoint. Now, consider the sequence $\{T^n\}$, where T^n is inductively defined as follows: $T^0 = \emptyset$, $T^n = T_{\mathcal{P}}(T^{n-1})$. Clearly, $\{T^n\}$ is monotonically increasing. Since the Base of $\mathcal{P}$ is finite, $\{T^n\}$ is upper bounded, so that there exists a natural i such that, for each $j \geq i$, $T^j = T^i$. Clearly, T^i coincides with the least fixpoint of $T_{\mathcal{P}}$, which we denote by $T^{\infty}_{\mathcal{P}}$. It is worth noting that $T_{\mathcal{P}}(I)$ contains a (possibly negative) literal Q if and only if it is the head of a rule such that every literal in its body, even if negative, is in I. That is, negation is considered as classical negation.

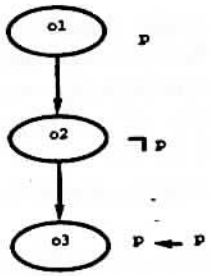

Figure 4.

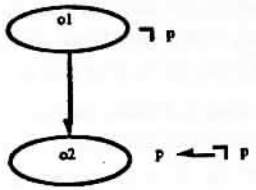

Figure 5: the program P_3

Definition 7 Let M be an interpretation for $\mathcal{P}$. A rule $r \in ground(\mathcal{P})$ is *effective* w.r.t. M if it is neither overruled nor defeated in M and, further, $B(r) \subseteq M$.

Definition 8 Let a program $\mathcal{P}$ and an interpretation M be given. The *reduction of $\mathcal{P}$ w.r.t. M* is $\mathcal{P}^M = \{r \in ground(\mathcal{P}) |\ r \text{ is effective w.r.t. } M\}$. □

Example 5 Given the program $\mathcal{P}_{nixon}$ and the set $M = \{pacifist, goodpresident\}$, $\mathcal{P}^M_{nixon}$ is the set consisting of the following rules: $pacifist \leftarrow$ and $goodpresident \leftarrow pacifist$.
For another example, consider again the program of Figure 3.a. For $M_1 = \{\neg b\}$, $\mathcal{P}^{M_1} = \{\neg b \leftarrow\}$, and for $M_2 = \{a, b\}$, $\mathcal{P}^{M_2} = \{b \leftarrow a\}$. □

In analogy with traditional logic programming, we now give the following definition of stable model.

Definition 9 Let a program $\mathcal{P}$ and a model M be given. M is a *stable model* for $\mathcal{P}$ if $M = T^{\infty}_{\mathcal{P}^M}(\emptyset)$. □

We point out that the above definition of stable model is very close to the the classical one given for logic programs. Intuitively, we define a stable model as a model "regenerating" itself - it can be computed as the least fixpoint of a program obtained from the original one simply by removing those rules that cannot contribute to draw conclusions.

A first important characterization of a stable model M for $\mathcal{P}$ is that the literals in it are precisely those that can be derived by the effective rules (w.r.t. M) of $\mathcal{P}$.

Proposition 2 *Let M be a stable model for $\mathcal{P}$. Then, $p \in M$ iff $\exists r \in ground(\mathcal{P})$ such that $H(r) = p$ and r is effective w.r.t. M.*

Proof. By definition 9, M is a fixpoint of $T_{\mathcal{P}^M}$, i.e., $M = T_{\mathcal{P}^M}(M)$. On the other hand, from definition 8, a rule r is in $\mathcal{P}^M$ iff it is an effective rule of $ground(\mathcal{P})$. From which the statement follows. □

Thus, informally, stable models support the notion of justifiability, i. e., every literal in a stable model is inferable by an effective rule.

Example 6 Consider the program $\mathcal{P}$ in Figure 4. The reduction of the program w.r.t. $M = \{p\}$ is $\{p \leftarrow p\}$. Indeed, $p \leftarrow p$ is the only effective rule w.r.t. M. It is easy to recognize that the least fixpoint $T^{\infty}_{\mathcal{P}^M}(\emptyset)$ is the empty set. We point

t that this corresponds to our intuition of stable model (based on the informal tion of justifiability). Indeed, in the program $\mathcal{P}$ the fact p is overruled by the more ecific fact $\neg p$. Thus, no support to infer p from the lowest rule $p \leftarrow p$ is provided. n the other hand, the reduction of the program w.r.t. $M = \{\neg p\}$ is $\neg p \leftarrow$ and $^{\infty}_{\mathcal{P}^M}(\emptyset) = \{\neg p\}$. Thus, $M = \{\neg p\}$ is a stable model for $\mathcal{P}$. □

Propositions 3 and 4 below prove that the class of stable models coincides with at of the unfounded-free models.

roposition 3 *Let M be a stable model for $\mathcal{P}$. Then $M \cap GUS_{\mathcal{P}}(M) = \emptyset$.*

Proof. Let X denote $M \cap GUS_{\mathcal{P}}(M)$. By way of contradiction, assume that X not empty. Then, we show that the set $N = M - X$ is a fixpoint of $T_{\mathcal{P}^M}$, thus ntradicting the assumption that M is a stable model (that is, the least fixpoint of $_{\mathcal{P}^M}$, by definition 9). Indeed, $p \in N$ iff there exists $r \in ground(\mathcal{P}^M)$ such that $(r) \subseteq M$ (as $N \subseteq M$ and M is a fixpoint of $T_{\mathcal{P}^M}$). On the other hand, it is mediate to recognize that $B(r) \cap X = \emptyset$ holds (otherwise p would belong to X). ence, $p \in N$ iff there exists $r \in ground(\mathcal{P}^M)$ such that $B(r) \subseteq N$. Thus, N is a xpoint of $T_{\mathcal{P}^M}$. □

roposition 4 *Let M be a model for $\mathcal{P}$ such that $M \cap GUS_{\mathcal{P}}(M) = \emptyset$. Then, M is stable model for $\mathcal{P}$.*

Proof. $M \cap GUS_{\mathcal{P}}(M) = \emptyset$ implies that, for each $p \in M$, there is a rule $r \in round(\mathcal{P})$ with head p whose body $B(r)$ is not false in M (i.e., $B(r) \subseteq M \cup \overline{M}$), $(r) \cap GUS_{\mathcal{P}}(M) = \emptyset$ and, further, it is neither defeated nor overruled in M. However, nce M is a model, by proposition 1, $\overline{M} \subseteq GUS_{\mathcal{P}}(M)$ follows, so that $B(r) \subseteq M$ holds otherwise p would be in $GUS_{\mathcal{P}}(M)$). Thus, r is effective w.r.t. M. We note that, by efinition 8, $r \in ground(T_{\mathcal{P}^M})$. On the other hand, since M is a model, cannot exist n effective rule $r \in ground(\mathcal{P})$ such that $H(r) \notin M$. Therefore, since r belongs to $round(\mathcal{P}^M)$, we can easily recognize that M is a fixpoint of $T_{\mathcal{P}^M}$. Now, we conclude y showing that so subset N of M is a fixpoint of $T_{\mathcal{P}^M}$. By way of contradiction, ssume $N = T_{\mathcal{P}^M}(N)$. Let $p \in M - N$, then there exists $r \in ground(\mathcal{P}^M)$ such hat $B(r) \subseteq M$ (as M is a fixpoint of $T_{\mathcal{P}^M}$) and $B(r) \cap (M - N) \neq \emptyset$ (otherwise p ould be in N). Hence, $M - N$ is an unfounded set w.r.t. M (thus contradicting the ypothesis $M \cap GUS_{\mathcal{P}}(M) = \emptyset$). □

From the above propositions, it turns out that stable models are minimal models.

Corollary 1 *Let M be a stable model for $\mathcal{P}$. Then, M is a minimal model of $\mathcal{P}$.*

Proof Straightforward. □

Example 7 The unique minimal model $\{\neg rich, \neg taxPayer\}$ of $\mathcal{P}_{tom}$ is also a stable odel.

The interpretations $\{pacifist, goodpresident\}$ and $\{\neg pacifist\}$ are both stable models or $\mathcal{P}_{nixon}$.

The program of Figure 3.a has the unique stable model $\{\neg b\}$. Notice that the other inimal model $\{a, b\}$ (see Example 4) is not stable in that neither a nor b can be erived from the program.

Both the minimal models $\{a, \neg b\}$ and $\{b, \neg a\}$ of $\mathcal{P}_2$ (see Figure 3.b) are stable models.

Finally, no stable model exists for the program $\mathcal{P}_3$ of Figure 5. □

5 A Fixpoint Semantics

Now that we have the appropriate declarative concepts, we provide a fixpoint semantics of stable models. We next define the transformation $W_{\mathcal{P}}$. As it will be shown next, $W_{\mathcal{P}}$ provides the ground for the computation of stable models.

Definition 10 Let $\mathcal{P}$ be a program. Define the transformation $W_{\mathcal{P}} : 2^{B_{\mathcal{P}}} \rightarrow 2^{B_{\mathcal{P}}}$ as follows:

$$W_{\mathcal{P}}(I) = T_{\mathcal{P}}(I) \cap \neg.GUS_{\mathcal{P}}(I)$$

□

Clearly, $W_{\mathcal{P}}$ is monotonic in the complete lattice $< 2^{B_{\mathcal{P}}}, \subseteq >$. Therefore it has the least fixpoint. Consider now the sequence $\{W^n\}$ inductively defined as follows: $W^0 = \emptyset$, $W^n = W_{\mathcal{P}}(W^{n-1})$. Since the Base of $\mathcal{P}$ is finite, $\{W^n\}$ is upper bounded, so that there exists a natural i such that, for each $j \geq i$, $W^j = W^i$. Clearly, W^i coincides with the least fixpoint of $W_{\mathcal{P}}$, which we denote by $W^{\infty}_{\mathcal{P}}(\emptyset)$.

Having defined the transformation $W_{\mathcal{P}}$, we next provide a number of results stating the relationship between the $W_{\mathcal{P}}$ operator and the stable models of $\mathcal{P}$.

Proposition 5 *Let M be a stable model for $\mathcal{P}$. Then it is a fixpoint of $W_{\mathcal{P}}$.*

Proof. By proposition 2, M stable implies that $p \in M$ iff there exists an effective rule $r \in ground(\mathcal{P})$. Further, by proposition 1, $\neg.M \subseteq GUS_{\mathcal{P}}(M)$ (thus, $\neg.p \in GUS_{\mathcal{P}}(M)$). Hence, M is a fixpoint of $W_{\mathcal{P}}$ □

Proposition 5 states that any stable model can be computed as a fixpoint of the $W_{\mathcal{P}}$ operator. Unfortunately, not every fixpoint of $W_{\mathcal{P}}$ is a stable model.

Example 8 Consider the program of Figure 4. It is easy to see that $\{p\}$ is a fixpoint of $W_{\mathcal{P}}$ but it is not a stable model (see Example 6). □

Thus, we need to state conditions allowing us to recognize, from among the fixpoints of $W_{\mathcal{P}}$, those that are stable models. To this end, we provide the following preliminary definition.

Definition 11 Given an interpretation I and a program $\mathcal{P}$, an undefined ground literal $Q \in \overline{I}$ is *possibly-true w.r.t. I* if $\exists r \in ground(\mathcal{P})$ with head Q such that: (1) $B(r) \subseteq I$, and (2) r is not overruled w.r.t. I. We denote $PT_{\mathcal{P}}(I)$ the set of possibly-true facts of $\mathcal{P}$ w.r.t. I. □

Example 9 For the program of the example 8, the set of possibly-true facts, w.r.t. the interpretation $\{\neg a\}$, is $\{\neg b\}$. Indeed, the rule $\neg b \leftarrow$ verifies all of the conditions of definition 11.
As another example, for the program of Figure 4.b, the set of possibly-true facts w.r.t. the empty set is $\{\neg a, \neg b\}$. □

Proposition 6 *Let M be an interpretation such that $M = W_{\mathcal{P}}(M)$, $M \cap GUS_{\mathcal{P}}(M) = \emptyset$ and $PT_{\mathcal{P}}(M) = \emptyset$. Then M is a stable model for $\mathcal{P}$.*

Proof. By definition of $W_\mathcal{P}$, $M = W_\mathcal{P}(M)$ implies that, for each $p \in M$, $\neg p \in$ $US_\mathcal{P}(M)$, i.e., $\neg.M \subseteq GUS_\mathcal{P}(M)$.

Now, from the condition $PT_\mathcal{P}(M) = \emptyset$, we have that for each $p \in \overline{M}$ and every ule r with $H(r) = p$, either $B(r) \not\subseteq M$ (i.e., $B(r) \subseteq \neg.M \cup \overline{M}$) or r is overruled in M. Hence, $\overline{M}$ is an unfounded set (w.r.t. M).

Since, by hypothesis, $M \cap GUS_\mathcal{P}(M) = \emptyset$, it immediately follows that $B_\mathcal{P} - M = \overline{M} \cup \neg.M$ is an unfounded set for $\mathcal{P}$ w.r.t. M (in particular, the greatest unfounded set). Thus, the statement follows from propositions 1 and 4. □

Example 10 Consider again the program $\mathcal{P}_1$ of Figure 4.a. Note that $M = \{a, b\}$ is a fixpoint of $W_\mathcal{P}$ and that $PT_{\mathcal{P}_1}(M) = \emptyset$. Indeed, M is a model for $\mathcal{P}_1$ (see example 4). However, M is not stable as it is not unfounded-free. Conversely, the set $M_1 = \{\neg b\}$ is a unfounded-free fixpoint of $W_\mathcal{P}$ for which $PT_\mathcal{P}(M_1) = \emptyset$ holds. Indeed, M is a stable model of $\mathcal{P}$. □

Thus, we have shown that *all* fixpoints of $W_\mathcal{P}$ that are unfounded-free and for which the condition $PT_\mathcal{P}(M) = \emptyset$ holds, are stable models for $\mathcal{P}$. Next we show that such fixpoints are the *only* stable models of $\mathcal{P}$.

Lemma 1 *Let M be a model for $\mathcal{P}$. Then $PT_\mathcal{P}(M) = \emptyset$.*

Proof. By way of contradiction, assume that there exists $p \in PT_\mathcal{P}(M)$. Thus, there is a rule $r \in ground(\mathcal{P})$ such that $H(r) = p$, $B(r) \subseteq M$ and r is not overruled in M. But, if so, by the hypothesis that M is a model, $H(r) \in M$ follows (a contradiction, as $PT_\mathcal{P}(M) \subseteq \overline{M}$). □

Proposition 7 *Let M be a stable model for $\mathcal{P}$. Then, $M = W_\mathcal{P}(M)$, $M \cap GUS_\mathcal{P}(M) = \emptyset$ and $PT_\mathcal{P}(M) = \emptyset$.*

Proof. Immediate from propositions 5 and 3 and lemma 1. □

6 Computing Stable Models

In the previous section we have seen that, to determine a stable model, we have to "navigate" through the unfounded-free fixpoints of the $W_\mathcal{P}$ operator and then checking whether the associated set of possibly-true facts is empty. In this section we show *how* to "navigate", that is, we provide a constructive way to determine stable models. We start by showing that the least fixoint of $W_\mathcal{P}$ is contained in any stable model (if any) of $\mathcal{P}$.

Proposition 8 *Let M be a stable model for $\mathcal{P}$. Then, (1) $W^\infty_\mathcal{P}(\emptyset) \subseteq M$ and (2) $W^\infty_\mathcal{P}(\emptyset) \cap GUS_\mathcal{P}(W^\infty_\mathcal{P}(\emptyset)) = \emptyset$*

Proof. 1) By proposition 5, any stable model is a fixpoint of $W_\mathcal{P}$. Since $W^\infty_\mathcal{P}(\emptyset)$ is contained in any fixpoint of $W_\mathcal{P}$, it is clearly contained in any stable model of $\mathcal{P}$.

2) *Basis*($i = 0$). Trivial, as $V^0 = \emptyset$.

Induction($i > 0$). Let p be a literal in $W^i - W^{i-1}$. Thus, there exists $r \in ground(\mathcal{P})$ such that $H(r) = p$, $B(r) \subseteq W^{i-1}$ and $\neg p \in GUS_\mathcal{P}(W^{i-1})$. Note that, since $W^{i-1} \cap GUS_\mathcal{P}(W^\infty_\mathcal{P}(\emptyset)) = \emptyset$ (inductive hypothesis), r is neither defeated nor overruled in W^{i-1} (otherwise $\neg p \notin GUS_\mathcal{P}(W^{i-1})$). We proceed by contradiction, assuming that $p \in GUS_\mathcal{P}(W^\infty_\mathcal{P}(\emptyset))$. Now, $W^{i-1} \cap GUS_\mathcal{P}(W^\infty_\mathcal{P}(\emptyset)) = \emptyset$ and $B(r) \subseteq W^{i-1}$ imply

$B(r) \cap GUS_\mathcal{P}(W^\infty_\mathcal{P}(\emptyset)) = \emptyset$. Further, $B(r) \subseteq W^\infty_\mathcal{P}(\emptyset)$ holds (as $W^{i-1} \subseteq W^\infty_\mathcal{P}(\emptyset)$). Hence, in order $p \in GUS_\mathcal{P}(W^\infty_\mathcal{P}(\emptyset))$, r must be either defeated or overruled in $W^\infty_\mathcal{P}(\emptyset)$. To this end, there must exist a conflicting rule r' (i.e., r' s.t. $H(r') = \neg p$ and either $comp-of(r') < comp-of(r)$ or r and r' are uncomparable (w.r.t. $<$)) such that $B(r') \subseteq W^\infty_\mathcal{P}(\emptyset)$. But then, $\neg p$ is also in $W^\infty_\mathcal{P}(\emptyset)$ (as $W^\infty_\mathcal{P}(\emptyset)$ is a fixpoint of $V_{\mathcal{P},C}$) thus contradicting point 1 above. Indeed, being $W^\infty_\mathcal{P}(\emptyset)$ contained in any stable model, it is an interpretation (i.e., it is contradiction-free). □

Thus, in general, $W^\infty_\mathcal{P}(\emptyset)$ is a subset of all stable models of $\mathcal{P}$ (if any). Hence, to determine a stable model we have to move beyond $W^\infty_\mathcal{P}(\emptyset)$, towards new unfounded-free fixpoints of $W_\mathcal{P}$. Let us show how this can be done using first an example.

Example 11 It is easy to see that for the program $\mathcal{P}_2$ of Figure 4.b both $W^\infty_{\mathcal{P}_2}(\emptyset) = \emptyset$ and $PT_\mathcal{P}(\emptyset) = \{\neg a, \neg b\}$ hold. Hence, by proposition 7, the empty set is not a model for $\mathcal{P}_2$. Now, let us assume that the possibly-true literal $\neg a$ is true. By using the $W_\mathcal{P}$ operator, we compute $W^\infty_{\mathcal{P}_2}(\{\neg a\}) = \{\neg a, b\}$, which we know to be a stable model. Likewise, by assuming the truth of $\neg b$, the stable model $\{a, \neg b\}$ is generated. We note that $PT_\mathcal{P}(\{\neg a, b\}) = PT_\mathcal{P}(\{a, \neg b\}) = \emptyset$.
Let us now take the program $\mathcal{P}_3$ of Figure 5. It is easy to recognize that $W^\infty_{\mathcal{P}_3}(\emptyset) = \emptyset$ and $PT_\mathcal{P}(\emptyset) = \{\neg p\}$. Notice that the assumption $\neg p$ leads to a contradiction (as $\neg p$ is derived). Thus, no stable model can be generated. □

The above example shows us how possibly-true facts play a central role in the computation of stable models. In fact, they not only provide a means to check whether a justified fixpoint of $W_\mathcal{P}$ is a stable model, but they also provide the basis from which selecting literals to "force" the computation of $W_\mathcal{P}$ in the "right" direction, i.e., according to proposition 7, towards new unfounded-free fixpoints of $W_\mathcal{P}$. Next we formalize the intuition given by the above example.

Definition 12 Define the family of functions *Choice* from $2^{B_\mathcal{P}}$ to $2^{B_\mathcal{P}}$:

$$Choice(A) = \begin{cases} \emptyset & \text{if } A = \emptyset \\ \{a\} & \text{for any } a \in A \end{cases}$$

□

Notice that each function in *Choice* is a non-deterministic function which differs from the others in the way an element is chosen from A.

Definition 13 Let $\mathcal{P}$ be a program. Define the transformation $V_{\mathcal{P},C}$ as

$$V_{\mathcal{P},C}(I) : 2^{B_\mathcal{P}} \leftarrow 2^{B_\mathcal{P}}$$

$$V_{\mathcal{P},C}(I) = \begin{cases} W_\mathcal{P}(I) \cup I & \text{if } W_\mathcal{P}(I) \neq I \\ I \cup C(PT_\mathcal{P}(I)) & \text{otherwise} \end{cases}$$

where $C \in Choice$. □

Note that there are as many transformations $V_{\mathcal{P},C}$ as the number of different functions C. Now, for a function $C \in Choice$, $V_{\mathcal{P},C}$ coincides with the transformation $W_\mathcal{P}$, but in its fixpoints. In particular, whenever a fixpoint of $W_\mathcal{P}$ is reached, a

literal is chosen from the associated set of the possibly-true literals. We point out that, in general, for an interpretation I, the application of $V_{\mathcal{P},C}$ may not yield an interpretation (recall that an interpretation is a consistent set of literals). Consider now the sequence $\{V^n_{\mathcal{P},C}\}$ inductively defined as follows: $V^0 = \emptyset$, $V^n = V_{\mathcal{P},C}(V^{n-1})$, where, for each natural j, $1 \leq j \leq n$, $V^{j-1}_{\mathcal{P},C}$ is an interpretation. It can be easily recognized that $\{V^n_{\mathcal{P},C}\}$ is monotonically increasing. Since the Base of $\mathcal{P}$ is finite, $\{V^n_{\mathcal{P},C}\}$ is upper bounded, so that there exists a natural i such that $V^j_{\mathcal{P},C} = V^i_{\mathcal{P},C}$, for each $j \geq i$. We denote $V^i_{\mathcal{P},C}$ by $V^\infty_{\mathcal{P},C}(\emptyset)$.

We next provide some useful properties about the $V_{\mathcal{P},C}$ transformation.

Lemma 2 *For any $C \in Choice$, M is a fixpoint of $V_{\mathcal{P},C}$ iff $W_{\mathcal{P}}(M) = M$ and $PT_{\mathcal{P}}(M) = \emptyset$.*

Proof By definition, M is a fixpoint of $V_{\mathcal{P},C}$, for any $C \in Choice$, iff $W_{\mathcal{P}}(M) = M$ and $M \cup C(PT_{\mathcal{P}}(M)) = M$. The latter condition holds iff $C(PT_{\mathcal{P}}(M)) \subseteq M$. Since $p \in PT_{\mathcal{P}}(M)$ implies $p \in \overline{M}$, $C(PT_{\mathcal{P}}(M)) \subseteq M$ holds iff $C(PT_{\mathcal{P}}(M)) = \emptyset$ iff $PT_{\mathcal{P}}(M) = \emptyset$. □

Proposition 9 *Let $V^\infty_{\mathcal{P},C}(\emptyset)$ be an interpretation. Then it is a stable model for $\mathcal{P}$.*

Proof. By lemma 2 and proposition 6, we just need to show that $V^\infty_{\mathcal{P},C}(\emptyset) \cap GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset)) = \emptyset$. This statement is proved by induction on the number of applications of $V_{\mathcal{P},C}$. Let $V^0 = \emptyset$, $V^1 = V_{\mathcal{P},C}(V^0), ..., V^i = V_{\mathcal{P},C}(V^{i-1}), ...$ In particular, we prove that, for each natural i, $V^i \cap GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset)) = \emptyset$.

Basis($i = 0$). Trivial, as $V^0 = \emptyset$.

Induction($i > 0$). Let p be a literal in $V^i - V^{i-1}$. We proceed by contradiction, assuming that $p \in GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset))$. Now, by definition of $V_{\mathcal{P},C}$, either (1) $p \in W_{\mathcal{P}}(V^{i-1})$ or (2) $p \in PT_{\mathcal{P}}(V^{i-1})$.

1. $p \in W_{\mathcal{P}}(V^{i-1})$. Thus, there exists $r \in ground(\mathcal{P})$ such that $H(r) = p$, $B(r) \subseteq V^{i-1}$ and $\neg p \in GUS_{\mathcal{P}}(V^{i-1})$. Note that, since $V^{i-1} \cap GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset)) = \emptyset$ (inductive hypothesis), r is neither defeated nor overruled in V^{i-1} (otherwise $\neg p \notin GUS_{\mathcal{P}}(V^{i-1})$).

2. $p \in PT_{\mathcal{P}}(V^{i-1})$ (i.e, p is a possibly-true fact). Hence, by definition 11, there exists a ground rule r with head p such that $B(r) \in V^{i-1}$ and, further, r is not overruled in V^{i-1}.

Now, $V^{i-1} \cap GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset)) = \emptyset$ and $B(r) \subseteq V^{i-1}$ imply $B(r) \cap GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset)) = \emptyset$. Further, $B(r) \subseteq V^\infty_{\mathcal{P},C}(\emptyset)$ holds (as $V^{i-1} \subseteq V^\infty_{\mathcal{P},C}(\emptyset)$). Hence, in order $p \in GUS_{\mathcal{P}}(V^\infty_{\mathcal{P},C}(\emptyset))$, r must be either defeated or overruled in $V^\infty_{\mathcal{P},C}(\emptyset)$. To this end, there must exist a conflicting rule r' (i.e., r' s.t. $H(r') = \neg p$ and either $comp - of(r') < comp - of(r)$ or r and r' are uncomparable (w.r.t. $<$)) such that $B(r') \subseteq V^\infty_{\mathcal{P},C}(\emptyset)$. But then, $\neg p$ is also in $V^\infty_{\mathcal{P},C}(\emptyset)$ (as $V^\infty_{\mathcal{P},C}(\emptyset)$ is a fixpoint of $V_{\mathcal{P},C}$) thus contradicting the hypothesis that $V^\infty_{\mathcal{P},C}(\emptyset)$ is an interpretation (i.e., it is contradiction-free). □

Corollary 2 *Let $V^\infty_{\mathcal{P},C}(\emptyset)$ be an interpretation. If $V^\infty_{\mathcal{P},C}(\emptyset) = W^\infty_{\mathcal{P}}(\emptyset)$ then it is the unique stable model for $\mathcal{P}$.*

Proof. Trivial. □

So, we have shown that $V^\infty_{\mathcal{P},C}(\emptyset)$, for any $C \in Choice$, is a stable model provided that it is an interpretation. Next we prove that if M is a stable model then there exists $C \in Choice$ such that $M = V^\infty_{\mathcal{P},C}(\emptyset)$. In other terms, any unfounded-free fixpoint M of $W_{\mathcal{P}}$, for which $PT_{\mathcal{P}}(M) = \emptyset$, can be computed as the least fixpoint of $V_{\mathcal{P},C}$, for some $C \in Choice$.

Proposition 10 *Let M be a stable model for $\mathcal{P}$. Then, $M = V^\infty_{\mathcal{P},C}(\emptyset)$, for some $C \in Choice$.*

Proof. By proposition 7, M stable implies (i) $M = W_{\mathcal{P}}(M)$, (ii) $PT_{\mathcal{P}}(M) = \emptyset$ and (iii) $M \cap GUS_{\mathcal{P}}(M) = \emptyset$. Conditions (i) and (ii) above imply, by lemma 2, that M is a fixpoint of $V_{\mathcal{P},C}$, for some $C \in Choice$. Thus, it remains to show that M is the least fixpoint $V^\infty_{\mathcal{P},C}(\emptyset)$. To this end, assume by way of contradiction that there exists an interpretation $N \subset M$ such that $N = V_{\mathcal{P},C}(N)$. Thus, by lemma 2, $N = W_{\mathcal{P}}(N)$ and $PT_{\mathcal{P}}(N) = \emptyset$. Further, by the monotonicity of $GUS_{\mathcal{P}}$, we have that $GUS_{\mathcal{P}}(N) \subseteq GUS_{\mathcal{P}}(M)$ and hence, from point (iii) above, $N \cap GUS_{\mathcal{P}}(N) = \emptyset$. Thus, by proposition 6, N is a stable model for $\mathcal{P}$. But this contradicts the minimality of M. □

7 A Backtracking Algorithm

We are now in a position to give an effective algorithm for the computation of a stable model which uses non-determinism and backtracking. According to proposition 7, the algorithm computes any stable model of a program $\mathcal{P}$ as the least fixpoint of the $V_{\mathcal{P},C}$ operator, for some function $C \in Choice$.

The algorithm implementing the $V_{\mathcal{P},C}$ transformation is shown in Figure 6. According to definition 13, the least fixpoint of $V_{\mathcal{P},C}$ is computed as a sequence of fixpoints of the $W_{\mathcal{P}}$ operator. In particular, the least fixpoint $W^\infty_{\mathcal{P}}(\emptyset)$ is first computed. If $PT_{\mathcal{P}}(W^\infty_{\mathcal{P}}(\emptyset))$ is empty then the computation stops, as the unique stable model of $\mathcal{P}$ has been found (see corollary 2). Otherwise, a choice of an element of $PT_{\mathcal{P}}(W^\infty_{\mathcal{P}}(\emptyset))$ is performed and then added to $W^\infty_{\mathcal{P}}(\emptyset)$. Thus, new derivations are enforced by again applying the $W_{\mathcal{P}}$ operator and a new fixpoint of $W_{\mathcal{P}}$ is computed. This process is reiterated until either a contradiction is found or the set of possibly-true facts is empty. In the latter case, according to proposition 9, a stable model has been determined. On the contrary, if a contradiction arose, a backtracking to the last *choice point* is performed. Thus, the process continues until a stable model has been found (if any) or no more choices are available. We notice that the backtracking strategy (from which the name *Backtracking Fixpoint*) is used to implement the $V_{\mathcal{P},C}$ transformation for every possible function in $C \in Choice$. Backtracking is controlled by means of a stack whose elements are pairs of the form $< I, D_I >$ where: I is an interpretation, and D_I is the set of the possibly-true literals that have been chosen and then discarded (as a contradiction arose). $Backtrack(AnotherChoice, J, D_J)$ is a primitive that backtracks to the first choice point on the stack, i.e., the first element to which a not empty set of further possible choices is still associated (formally, the first element such that $PT_{\mathcal{P}}(J) - D_J \neq \emptyset$. This primitive returns $AnotherChoice = false$ if no

Backtracking Algorithm

```
rocedure Backtrack(AnotherChoice, I, D_I);
egin
        AnotherChoice := false;
        while Stack is not empty and ¬AnotherChoice do
                let < J, D_J > be the element at the top of Stack
                if PT_P(J) − D_J ≠ ∅ then
                        AnotherChoice := true;
                        I = J; D_I = D_J
                else decrement the top;
        endwhile
nd.

egin
. Stack := ∅; I := ∅;
0. Stable := false; AnotherChoice := true;
1. for each interpretation J set D_J := ∅;
2. repeat
3.      repeat
4.              J := I;
5.              I := W_P(J);
6.      until I = J;
7.      if I ∩ ¬.I = ∅ then /* no contradiction
8.              stable := (PT_P(I) = ∅)
9.              ifstable then AnotherChoice := false
0.      else Backtrack(AnotherChoice, I, D_I)
1.      ifAnotherChoice then
2.                      A_I := choice(PT_P(I) − D_I);
3.                      push Stack(I, A_I ∪ D_I);
4.                      I := I ∪ {A_I};
5. until stable or ¬(AnotherChoice)
6. if stable then output "I is a stable model"
7. else output "there is no stable model"
nd
```

Figure 6

such an element exists. The command *choice*, in turn, represents a nondeterministic construct that selects one element from a set.

To conclude, we remark that the problem of computing stable models for ordered logic programs is NP-complete. This result can be proved by showing that this problem is in NP and, further, the stable model semantics for ordered logic programs generalizes tha Datalog stable model semantics, which is known to be NP-complete.

Acknowledgements.
The authors Francesco Buccafurri and Nicola Leone are partially supported by the Italian National Research Council under grant 24.07.12 and 24.07.11, respectively.

References

[1] Apt, K., Bair, H., Walker, A., Towards a theory of declarative knowledge, In *Found. of Ded. Database and Logic Prog.* Minker, J. (ed.), Morgan Kaufman, Los Altos, 1987, pp. 89-148.

[2] Gelfond, M. and Lifschitz, V. "Logic Programs with Classical Negation", *Proc.of 7th ICLP*, Jerusalem, 1990, pp 579-597.

[3] Geerts, P., Vermier, D., "Credulous and Autoepistemic Reasoning Using Ordered Logic", *Proc. of the First Int. Work. on Logic Progr. and Nonmonot. Reas.*, Washington, July, 1991.

[4] Laenes, E., "Foundation of Ordered Logic", Ph. D. thesis, Univ. of Antwerp, 1990.

[5] Laenens, E., Saccá, D., and Vermeir, D., "Extending Logic Programming", *Proc. of ACM SIGMOD*, May 1990.

[6] Laenens, E., and Vermeir, D., "A Fixpoint Semantics for Ordered Logic", *Journal of Logic and Computation*, vol.1, n.2, december, 1990, pp. 159-185.

[7] Leone, N., Rullo, P., "Stable Model Semantics and its Computation for Ordered Logic Programs", in *10th Eur. Conf. on Art. Int.*, August, Vienna, Austria, 1992, pp. 92-96.

[8] Przymusinski, T. C., "On the declarative semantics of stratified deductive databases and logic programs", in *Found. of Ded. Databases and Logic Progr.*, Minker, J. ed., Morgan Kaufman, Los Altos, 1987, pp. 193-216

[9] Przymusinski, T. C., "Every logic program has a natural stratification and an iterated fixed point model", *Proc. of Symp. on Princ. of Datab. Syst.*, pp 11-21, *ACM SIGACT-SIGMOD* 1989.

[10] Przymusinska, H., Przymusinsky, T. C., "Weakly perfect model semantics for logic programs", *Proc. Fifth Int. Conf. and Symp. on Logic Programming*, 1988.

[11] Reiter, R., "A Logic for Default Reasoning", *Artificial Intelligence*, vol.13, pp. 81-132, 1980.

[12] Saccá, D., Zaniolo, C., "Stable Models and Nondeterminism in Logic Programs with Negation", *Proc. ACM Symp. on Principles of Database Systems* 1990.

[13] Van Gelder, A., Ross, K. A., Schlipf, J. S., "The well-founded semantics for general logic programs", *Journal of the ACM*, 1990.

Finite failure is and-compositional

Roberta Gori and Giorgio Levi

Dipartimento di Informatica,
Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy
e-mail: levi@di.unipi.it

Abstract

We study some properties of SLD-trees related to finite failure. The main result is a theorem stating that finite failures are AND-compositional, i.e. that the finite failure behavior of conjunctive goals can be derived from the finite failure behavior of atomic goals. The proof is based on two new lemmata which generalize to infinite derivations theorems which are valid for successful and finitely failed derivations. A side result is the proof that the non-ground finite failure set is correct and fully abstract wrt finite failures.

1 Introduction

The operational semantics of (positive) logic programs is usually based on SLD-trees. Several operational properties, useful for reasoning about programs, can be extracted from an SLD-tree. Examples are SLD-derivations, resultants, partial answers, computed answers, finite failures. All these properties, that we call *observables*, can be obtained as abstractions of the SLD-tree. The study of the observables may provide a new insight into the operational semantics and results that can be helpful in the theory of logic programs.

Sometimes it is useful to associate an observable with a denotation, which is a mathematical object which characterizes the behavior of a program with respect to that observable. This is the approch chosen in the s-semantics for the computed answers observable [3, 4] and in [5, 6, 2] for other less abstract observables.

In both the above mentioned examples it turns out that a lot of interesting theorems about the observable can more naturally be stated and proved if expressed in terms of the denotation. The relation between the observable and the denotation can be understood in terms of the concepts of *correctness* and *full abstraction*.

Let o be the observable, i.e. an abstraction of the SLD-trees. o induces an observational equivalence $\approx_o$ on programs. Two programs are equivalent according to $\approx_o$,

if they are observationally indistinguishable wrt o. Now, assume $D(P)$ is a denotation of program P. Then D is correct with respect to o, if $D(P_1) = D(P_2)$ implies $P_1 \approx_o P_2$. D is also fully abstract if $P_1 \approx_o P_2$ implies $D(P_1) = D(P_2)$. A denotation useful to discuss an observable must be correct with respect to it. In addition, if it is fully abstract, it enjoys a kind of minimality property.

In this paper we consider the properties of finite failures, following the above mentioned approach. After showing that the standard finite failure set is too weak as a denotation for characterizing finite failures, we define a denotation which is correct and fully abstract with respect to finite failures. The main technical result is and-compositionality. And-compositionality is one of the most important properties already proved for the above mentioned observables. It is related to the fact that the observable behavior of conjunctive goals can be derived from the observable behaviors of atomic goals. And-compositionality allows us to build a denotation by considering atomic goals only. It is straightforward to show that if a denotation is and-compositional it is also correct. Our main theorem will then be the and-compositionality theorem. The result is technically quite complex and relies on a set of properties of (fair) SLD-trees, which are of some interest by themselves.

The paper is organized as follows. In section 3 we show that the Finite Failure set is not correct and we define the denotation $NGFF$ introduced in [8]. Section 4 contains the proof of the and-compositionality theorem and all the related lemmata. Finally section 5 shows that $NGFF$ is also fully abstract.

2 Preliminaries

The reader is assumed to be familiar with the terminology of and the basic results in the semantics of logic programs [9, 1]. Let the signature S of a program P consist of the set $\sum_P$ of *function simbols*, a finite set Π_P of *predicate symbols*, a denumerable set V of *variable symbols*. All the definitions in the following will assume a given signature S. Let T be the set of terms built on $\sum_P$ and V. Variable-free terms are called *ground*. A substitution is a mapping ϑ: $V \rightarrow T$ such that the set $Dom(\vartheta) = \{X | \vartheta(X) \neq X\}$ (*domain of* ϑ) is finite. If $W \subset V$, we denote by $\vartheta_W(Y)$ the *restriction* of ϑ to the variables in W, i.e. $\vartheta_{|W}(Y)$ for $Y \notin W$. ϵ denotes the empty substitution. The *composition* $\vartheta \cdot \sigma$ of the substitutions ϑ and σ is defined as functional composition. If $Dom(\vartheta) \cap Dom(\sigma) = \emptyset$ we can define the *union* of the substitutions ϑ and σ as $(\vartheta \cup \sigma)(X) = X\vartheta$ if $X \in Dom(\vartheta)$, $X\sigma$ otherwise. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \epsilon$. The pre-ordering $\leq$ (more general than) on substitutions is such that $\vartheta \leq \sigma$ iff there exists ϑ' such that $\vartheta \cdot \vartheta' = \sigma$. The result of the application of the substitution ϑ to a term t is an *instance* of t denoted by $t\vartheta$. We define $t \leq t'$ (t is more general than t') iff there exists ϑ such that $t\vartheta = t'$. A substitution ϑ is *grounding* for t if $t\vartheta$ is ground. The relation $\leq$ is a preorder. $\approx$ denotes the associated equivalence relation (*variance*). A substitution ϑ is a *unifier* of terms t and t' if $t\vartheta \equiv t'\vartheta$. The *most general unifier* of t_1 and t_2 is denoted by $mgu(t_1, t_2)$. All the above definitions can be extended to other syntactic expressions in the obvious way. An atom is an object of the form $p(t_1, \ldots, t_n)$, where $p \in \Pi_P$ and $t_1, \ldots, t_n$ belong to T. A *clause* is a formula of the

form $H : -L_1, \ldots, L_n$, with $n \geq 0$, where H (the *head*) and $L_1, \ldots, L_n$ (the *body*) are atoms. ":-" and "," denote logic implication and conjunction respectively, and all the variables are universally quantified. If the body is empty the clause is a *unit clause*. A *program* is a finite set of clauses. A *goal* is a formula $\leftarrow L_1, \ldots, L_n$, where each L_i is an atom. By $Var(E)$ we denote the set of variables occurring in the expression E. The *Herbrand base* B_P for P is the set of all the ground atoms belonging to the signature of the program P. A *Herbrand interpretation* is any subset of B_P. The *extended Herbrand base* B_V for P is any subset of all the atoms built with the predicate symbols Π_P applied to the set of function symbols $\sum_P$ and to the set of variables V. If G is a goal, $G \overset{\vartheta}{\leadsto}_{P,R} B_1, \ldots, B_n$ denotes an SLD derivation with the selection rule R of $B_1, \ldots, B_n$ in the program P, where ϑ is the composition of the mgu's used in the derivation. $G \overset{\vartheta}{\leadsto}_{P,R} \square$ denotes the refutation of G in the program P with computed answer substitution ϑ. A computed answer substitution is always restricted to the variables occurring in G. We will denote by $\tilde{X}$ and $\tilde{t}$ a tuple of distinct variables and a tuple of terms respectively, while $\tilde{B}$ will denote a (possible empty) conjuction of atoms. We extend now the preorder on atoms by defining a preorder on conjunctions of atoms.

Definition 2.1 $A_1, \ldots, A_k \leq \overline{A}_1, \ldots, \overline{A}_k$ *if for* $i = 1, \ldots, k \quad A_i \leq \overline{A}_i$.

Definition 2.2 $A_1, \ldots, A_k < \overline{A}_1, \ldots, \overline{A}_k$ *if* $\exists\ J = \{j_1, \ldots, j_f\} \subseteq \{1, \ldots, k\}$ *and* $J \neq \emptyset$ *such that* $A_i < \overline{A}_i\ \forall i \in J$ *and* $A_s \leq \overline{A}_s\ \forall s \in \{1, \ldots, k\} \backslash J$.

3 Which semantics for finite failure

We first define the observational equivalence relation $\approx_{FF}$, induced on programs by the finite failures observable.

Definition 3.1 *Let* P_1 *and* P_2 *be positive programs,* G *be a positive goal and* T_1 *and* T_2 *be SLD-trees (defined by a fair selection rule) for* G *in* P_1 *and* P_2 *respectively. Then* $P_1 \approx_{FF} P_2$ *if, for every goal* G, T_1 *is finitely failed if and only if* T_2 *is finitely failed.*

As we will show now, the standard semantics, i.e. the ground finite failure set, is not able to model the behavior of finite failure. Namely, the ground finite failure set cannot distinguish programs which have different sets of goals having a fair finitely failed SLD-tree.

The finite failure set FF_P is the set of all the ground atoms which finitely fail in P.

$$FF_P = \{\ A \mid A \text{ is a ground atom and } \leftarrow A \text{ has a finitely failed SLD-tree }\}.$$

It is easy to note that FF_P is not correct with respect to $\approx_{FF}$. Here is a counterexample.

Example 3.1

P_1 :
```
p(f(X)):-p(X)
s(a)
```

P_2 :
```
p(f(X)):-p(X),p(a)
s(a)
```

P_1 and P_2 have the same finite failure set.

$$FF_{P_1} = FF_{P_2} = \{\ p(a), p(f(a)), p(f(f(a))), \cdots \\ s(f(a)), s(f(f(a))), s(f(f(f(a)))), \ldots\}$$

However the goal $\leftarrow p(X)$ has a fair finitely failed SLD-tree in P_2 while $\leftarrow p(X)$ has an infinite fair SLD-tree in P_1.

The set FF_P is not adequate for modeling the finite failure of non-ground goals.

Our aim is to find a semantics which can distinguish programs which are not observationally equivalent wrt the finite failure behavior. In other words, we want to find a semantics σ which is correct wrt $\approx_{FF}$, i.e. the following relation must hold:

$$\sigma(P_1) = \sigma(P_2) \Rightarrow P_1 \approx_{FF} P_2.$$

The semantics we consider is *the non-ground finite failure set*. It is the set of atomic goals which have at least a finitely failed SLD-tree. Namely,
$NGFF_P = \{A \mid \leftarrow A$ has a finitely failed SLD-tree $\}$.
$NGFF_P$ was first introduced in [8], where an equivalent bottom-up definition of $NGFF$ is also given.

We will show that $NGFF_P$ is correct wrt the equivalence $\approx_{FF}$, that is, given two programs P and Q, if $NGFF_P = NGFF_Q$ then if G finitely fails in P it also finitely fails in Q and vice versa.

A direct proof of the correctness of $NGFF_P$ can be found in [7]. Alternatively one can prove a stronger property, i.e. that $NGFF_P$ is and-compositional. This is the approach taken in this paper . The fact that $NGFF$ is and-compositional implies that it also correct wrt the equivalence relation $\approx_{FF}$. In fact, since $NGFF_P$ is the set of atomic goals finitely failed in P, if we can, on the basis of this information, decide whenever a compound goal finitely fails in P, then it is not possible to find a goal which finitely fails in P and not in Q or vice versa, if $NGFF_P = NGFF_Q$.

We will often use the complement of $NGFF_P$, called $\overline{NGFF}_P$ and defined as $B_V / NGFF_P$, where B_V is the extended Herbrand base. An equivalent characterization can be given as
$\overline{NGFF}_P = \{\ A \mid \exists\ \vartheta'$ such that $\leftarrow A\vartheta'$ has a successful SLD-tree or an infinite fair SLD-tree$\}$.

4 $NGFF_P$ is and-compositional

And-compositionality is relevant because it implies correctness. However it has a great relevance by itself. In fact, it is very useful to find a way to simulate the execution of a goal in the "semantics of a program P", in order to obtain the same result which one would obtain by computing the same goal in the program P.

The problem of determining whether a compound goal G finitely fails, by knowing whether $\leftarrow A$ finitely fails, for each A belonging to G, is a hard problem, as shown by the following example.

Example 4.1

P_1 :
```
p(a)
q(b):-q(b)
s(a)
```

$\overline{NGFF}_{P_1} = \{p(a), p(X), q(b), q(X), s(a), s(X)\}$ *The goal* $G = \leftarrow p(X), q(X)$ *finitely fails in* P_1 *and both* $p(X)$ *and* $q(X)$ *belong to* $\overline{NGFF}_{P_1}$.

Example 4.2

P_2 :
```
p(X):-p(X)
q(b)
s(a)
```

$\overline{NGFF}_{P_2} = \{p(a), p(b), p(X), q(b), q(X), s(a), s(X)\}$ *The goal* G *has an infinite fair SLD-tree in* P_2 *and both* $p(X)$ *and* $q(X)$ *belong to* $\overline{NGFF}_{P_2}$.

The problem shown by the above examples is the following. Consider first the program P_1. Both $p(X)$ and $q(X)$ belong to $\overline{NGFF}_P$ because they have a successful SLD-tree and an infinite fair SLD-tree respectively. However, the SLD-trees instantiate the variables occurring in the goal computing incompatible substitutions for X. This is shown by the presence of $p(a)$ and $q(b)$ in $\overline{NGFF}_{P_1}$. Hence, for $p(X)$ both substitutions $\{X/a\}$ and $\{X/b\}$ are possible. In program P_2, on the contrary, $p(X)$ has an infinite SLD-tree, which does not instantiate the variables occurring in the goal. $q(X)$ has a successful tree which computes the substitution $\{X/b\}$. This is shown by the presence of $p(b)$ in $\overline{NGFF}_{P_2}$. One could try to solve this problem, by considering ground atoms only in $\overline{NGFF}$. This is not correct, as shown by the following example.

Example 4.3

P_3 :
```
q(f(X)):-q(X)
p(X)
```

$$\overline{NGFF}_{P_3} = \{ \begin{array}{l} q(X), q(f(X)), q(f(f(X))), \ldots \\ p(X), p(f(X)), p(f(f(X))), \ldots \\ p(a), p(f(a)), p(f(f(a))), \ldots \end{array} \}.$$

The goal $\leftarrow p(X), q(X)$ *does not finitely fail in* P_3. *Both* $p(X)$ *and* $q(X)$ *belong to* $\overline{NGFF}_{P_3}$ *but there exists no pair of ground atoms belonging to* $\overline{NGFF}_{P_3}$ *showing a compatible substitution for* $p(X)$ *and* $q(X)$.

Since we need to characterize also infinite objects, we extend the usual notion of extended Herbrand Universe, Base and Interpretation to continuous ones, thus allowing the characterization of infinite objects, computed by infinite derivations. We need to embed the extended Herbrand Universe into a complete domain. In order to achieve our aim we follow the approach presented in [8]. In the following a continuous Herbrand Universe will be considered.

We will now formalize what we have understood from the previous examples. Let us first introduce the $Min(I)$ operator, where I is a subset of the continuos Herbrand Base.

Definition 4.1 $Min(I) = \{t \mid t \in I$ *and* $\nexists\, t'$, $t' \neq t$ *such that* $t' \geq t$ $t' \in I\}$

Min captures the essential component of I, i.e. the component that was involved in the previous examples.

The proof of the and-compositionality theorem is based on properties of infinite derivations, stated by lemmata 4.3 and 4.4. These lemmata generalize to two classes of infinite derivations the following two well known results about successful and finitely failed derivations.

Lemma 4.1 (Successful derivations) *[9] If the goal* G *has a successful SLD-tree in* P, *with computed answer substitution* ϑ', *then* $G\vartheta$ *has a successful SLD-tree in* P, *for every substitution* ϑ *such that* $\vartheta \geq \vartheta'$.

Lemma 4.2 (Finitely failed derivations) *[1] If the goal* G *has a finitely failed SLD-tree in* P *via* R *(where* R *is a fair selection rule) then also* $G\vartheta$ *has a finitely failed SLD-tree for every substitution* ϑ.

We want to find a similar property for infinite SLD-trees computed by fair selection rules. In other words, we want to be able to say something concerning the behavior of the goal $G\vartheta$, once we know that the goal G has an infinite SLD-tree.

In order to achieve our aim we need to distinguish two kinds of infinite derivations for a given goal G.

- The substitutions computed during the infinite derivation of G keep instantiating the variables occurring in the goal G. This derivation is called *perpetual infinite derivation.*
- There exists k, such that all the substitutions computed at steps whose depth is $\geq k$, do not instantiate the variables occurring in the goal G and instantiate only variables introduced in the derivation. Such a derivation is called *non-perpetual infinite derivation.*

Let us give now the formal definition of perpetual derivation.

Definition 4.2 *Let* d *be an infinite derivation in the fair SLD-tree for the goal* G. *Let* ϑ_i *be the substitution computed at the* i*-th resolution step. Then* d *is a perpetual infinite derivation if* $\forall i\ \exists n$

$$G(\vartheta_0 \cdot \vartheta_1 \cdot \ldots \cdot \vartheta_i) < G(\vartheta_0 \cdot \vartheta_1 \cdot \ldots \cdot \vartheta_i \cdot \ldots \cdot \vartheta_{i+n}).$$

The following example is meant to help understanding the concept of perpetual infinite derivation.

Example 4.4

```
P :  p(f(X)):-p(X)
     q(X):-t(X)
     t(b):-t(X)
```

The goal $\leftarrow p(X)$ has an infinite perpetual derivation, while the goal $\leftarrow q(X)$ has an infinite SLD-tree with an infinite derivation which, on the contrary, is not perpetual.

It is worth noting that if d is a non-perpetual derivation for the goal G in P, then $\exists k$ such that $\forall i,\ i \geq k$

$$G(\vartheta_0 \cdot \vartheta_1 \cdot \ldots \cdot \vartheta_k) = G(\vartheta_0 \cdot \vartheta_1 \cdot \ldots \cdot \vartheta_k \cdot \ldots \cdot \vartheta_i).$$

$\vartheta = (\vartheta_0 \cdot \vartheta_1 \cdot \ldots \cdot \vartheta_k)_{|Vars(G)}$ is called the *definite answer* for the non-perpetual derivation d.

Now we have all the notions which are necessary to state the following lemma, whose proof can be found in [7].

Lemma 4.3 (Non-perpetual infinite derivations) *Let G be a goal and R be a fair selection rule. Assume that G has an infinite SLD-tree via R, with at least one non-perpetual infinite derivation d, with definite answer ϑ'. Consider now the SLD-tree of $G\vartheta$, $\vartheta \geq \vartheta'$, via the selection rule R' which selects, at every resolution step, that atom in $G\vartheta$, which is in the same position of the one selected by R in G. Then $G\vartheta$ has an infinite SLD-tree via R' with a non-perpetual derivation which has ϵ as definite answer.*

Lemma 4.3. gives a property for fair SLD-trees of $G\vartheta$, $\vartheta \geq \vartheta'$, once we know that the goal G has an infinite fair SLD-tree with a definite answer ϑ' for an infinite non-perpetual derivation.

We want to find a similar result for a goal $G\vartheta$, in the case where G has an infinite fair SLD-tree with a perpetual derivation. To this aim, it is useful to distinguish which are the variables belonging to the goal G which are going to be instantiated infinitely many times in a given derivation and which are the ones that will not be instantiated anymore after a suitable resolution depth.

Definition 4.3 (Perpetual variables) *Let P be a program, G be a goal having at least one infinite perpetual derivation d. Let*

$$\vartheta_1, \vartheta_2, \ldots,$$

be the substitutions computed at each resolution step in the derivation d, restricted to the variables in $Vars(G)$. The set of perpetual variables of G, $Per(G)$ is defined as follows. $Per(G)=\{X_1, \ldots, X_s\}$, where $\{X_1, \ldots, X_s\} \subseteq Vars(G)$ and $\forall X \in \{X_1, \ldots, X_s\}$ the term bound to X is instantiated infinitely many times by the substitutions $\vartheta_1, \vartheta_2, \ldots$. Namely, for every X there exists an infinite set of indexes of substitutions $\{s_1, s_2, \ldots\}$ such that

$$\vartheta_{s_1|X_i} < \vartheta_{s_2|X_i} < \ldots\ldots$$

The perpetual variables of a goal G, in a given derivation, are then the variables which are instantiated infinitely many times during an infinite perpetual derivation.

In the following we will sometimes use the complement of the set $Per(G)$ wrt to the set of all the variables occurring in G.

Definition 4.4 $\overline{Per}(G) = \{X_1, \ldots, X_n\}$ *such that for every* $X_i \in \{X_1, \ldots, X_n\}$, $X_i \in Vars(G)$ *and* $X_i \notin Per(G)$.

From the definitions of $\overline{Per}(G)$ and of perpetual derivation, we can easily prove that there exists a resolution depth k, such that the variables in $\overline{Per}(G)$ are not instantiated anymore by the substitutions computed by resolution steps of depth $> k$. Namely there exists k such that $\forall r,\ r > k$

$$\vartheta_{k_{|\overline{Per}(G)}} = \vartheta_{r_{|\overline{Per}(G)}},$$ where ϑ_r is the substitution computed at the r-th resolution step. $\vartheta_{k_{|\overline{Per}(G)}}$ is called the *partial perpetual answer* for the goal G and the perpetual infinite derivation we are analyzing. The following lemma, whose proof is in [7], characterizes the behavior of $G\vartheta'$, if $\vartheta' \geq \overline{\vartheta}$ and $\overline{\vartheta}$ is the partial perpetual answer for the goal G, where G has at least one perpetual derivation.

Lemma 4.4 (Perpetual infinite derivations) *Let G be a goal and R be a fair selection rule. Assume that G has an infinite SLD-tree via R, with at least one perpetual infinite derivation d, with partial perpetual answer ϑ'. Consider now the SLD-tree of $G\vartheta$, $\vartheta \geq \vartheta'$ and $Dom(\vartheta) = Dom(\vartheta')$, via the selection rule R' which selects, at every resolution step, that atom in $G\vartheta$, which is in the same position of the one selected by R in G. Then $G\vartheta$ has an infinite SLD-tree via R' with at least one perpetual derivation.*

We are now ready to state the main and-compositionality theorem. The characterization of a compound goal will be obtained by "computing" the goal in the finite failure semantics $NGFF$. To this aim we apply the Min operator to the interpretation $\overline{NGFF}_P$.

The following theorem states that a goal $G = \leftarrow A_1, \ldots, A_k$ has a finitely failed SLD-tree in P iff there does not exist a substitution ϑ such that for every atom A_i in G, $A_i\vartheta$ belongs to the set $Min(\overline{NGFF}_P)$.

Theorem 4.1 (And-compositionality) *The goal $G = \leftarrow A_1, \ldots, A_k$ has a finitely failed SLD-tree in P if and only if*
$\nexists \vartheta$: $A_i\vartheta \in Min(\overline{NGFF}_P)$ *for i=1...k.*

Proof
In the following the set $\overline{NGFF}_P$ will be called $\sigma(P)$.

$\Longrightarrow$

Assume that $\leftarrow A_1, \ldots, A_k$ has a finitely failed SLD-tree in P and that $\exists \vartheta$ $A_i\vartheta \in Min(\sigma(P))$ *for* $i = 1, \ldots, k$.
Since $Min(\sigma(P))$ is a subset of $\sigma(P)$, then no atom $A_i\vartheta$ has a finitely failed SLD-tree.
Since $\forall i \ A_i\vartheta \in \sigma(P)$, we can have two cases related to the behavior of the compound goal G.

1. Every goal $\leftarrow A_i\vartheta$ has a successful or infinite fair SLD-tree, which, in the first case, has an empty answer substitution and, in the second case, has an infinite derivation, such that the substitution computed at each resolution step, does not instantiate variables occurring in G. Then the compound goal $\leftarrow A_1\vartheta, \ldots, A_k\vartheta$ has a successful fair SLD-tree or an infinite SLD-tree.
   Together with the assumption that the goal $\leftarrow (A_1, \ldots, A_k)\vartheta$ had a finitely failed fair SLD-tree, this gives the contradiction.

2. There exists an A_h for which the above case 1 does not apply. $A_h\vartheta \in \sigma(P)$ and $A_h\vartheta$ has not a successful or infinite fair SLD-tree. Then there exists a substitution ϑ' such that
   $$A_h\vartheta \leq A_h\vartheta'$$
   and $\leftarrow A_h\vartheta'$ has a successful or an infinite fair SLD-tree which, in the first case, has an empty answer substitution and, in the second case, has an infinite derivation, such that the substitution computed at each resolution step does not instantiate variables occurring in G.
   If this were the case, $A_h\vartheta$ would not belong to $Min(\sigma(P))$, by definition of the Min operator.

$\Longleftarrow$

Assume that
$\nexists \vartheta \ A_i\vartheta \in Min(\sigma(P))$, for $i = 1, \ldots, k$, and
the goal $\leftarrow A_1, \ldots, A_k$ does not finitely fail in P.
We have the following cases.

1. $\leftarrow A_1, \ldots, A_k$ has a successful fair SLD-tree. Then there exists at least a computed answer ϑ such that $\leftarrow (A_1, \ldots, A_k)\vartheta$ has a successful fair SLD-tree. By lemma 4.1,
   $$\forall \ \vartheta_a \text{ grounding substitution for } (A_1, \ldots, A_k)\vartheta$$
   $$\leftarrow (A_1, \ldots, A_k)(\vartheta \cdot \vartheta_a) \text{ has a successful fair SLD-tree} \Rightarrow$$
   $$\forall i \quad A_i(\vartheta \cdot \vartheta_a) \text{ has a successful fair SLD-tree} \Rightarrow$$
   $$\forall i \ A_i(\vartheta \cdot \vartheta_a) \in \sigma(P).$$
   Since $\forall i \ A_i(\vartheta \cdot \vartheta_a)$ is ground,
   $$\forall i \ A_i(\vartheta \cdot \vartheta_a) \in Min(\sigma(P)).$$
   By hypothesis such a substitution did not exist.

2. $\leftarrow A_1, \ldots, A_k$ has an infinite fair SLD-tree with at least one non-perpetual derivation. Let ϑ_a be a grounding substitution for $(A_1, \ldots, A_k)\overline{\vartheta}$, where $\overline{\vartheta}$ is the definite answer substitution. By lemma 4.3
   $$(A_1, \ldots, A_k)(\overline{\vartheta} \cdot \vartheta_a) \text{ has an infinite fair SLD-tree with a non-perpetual}$$
   infinite derivation, which has an empty definite answer. Hence
   $$\forall i \ A_i(\overline{\vartheta} \cdot \vartheta_a) \in \sigma(P).$$
   Since $\forall i \ A_i(\overline{\vartheta} \cdot \vartheta_a)$ is ground
   $$\forall i \ A_i(\overline{\vartheta} \cdot \vartheta_a) \in Min(\sigma(P)).$$
   By hypothesis such a substitution did not exist.

3. $\leftarrow A_1, \ldots, A_k$ has an infinite SLD-tree with at least one perpetual derivation. Then there exists a partial perpetual answer ϑ such that $Dom(\vartheta) \subseteq \overline{Per}(G)$ and, by lemma 4.4, $(A_1, \ldots, A_k)\vartheta$ has an infinite fair SLD-tree with a perpetual infinite derivation.
   Let ϑ_a be a grounding substitution for all the variables belonging to $Per(G)$. Namely
   $$Vars((A_1, \ldots, A_k)(\vartheta \cdot \vartheta_a)) \cap \overline{Per}(G) = \emptyset \text{ and } Dom(\vartheta_a) \cap Per(G) = \emptyset.$$
   Since $\leftarrow (A_1, \ldots, A_k)(\vartheta \cdot \vartheta_a)$ has a fair SLD-tree with at least one perpetual derivation, by lemma 4.4
   $$\forall i \ A_i(\vartheta \cdot \vartheta_a) \in \sigma(P).$$
   If, for some i, $Vars(A_i(\vartheta \cdot \vartheta_a)) = \emptyset$ then $A_i(\vartheta \cdot \vartheta_a)$ is ground and $A_i(\vartheta \cdot \vartheta_a) \in Min(\sigma(P))$.
   There exists at least one j such that $A_j(\vartheta \cdot \vartheta_a)$ is not ground, since $Dom(\vartheta \cdot \vartheta_a) \cap \overline{Per}(G) = \emptyset$.
   This is due to our choice of ϑ_a and to the definition of partial perpetual answer.
   Since we are considering an infinite perpetual derivation of the SLD-tree for the goal $\leftarrow A_1, \ldots, A_k$ then $Per(A_1, \ldots, A_k) \neq \emptyset$.
   Let us consider A_j such that $Vars(A_j) \neq \emptyset$.
   Since $\leftarrow (A_1, \ldots, A_k)(\vartheta \cdot \vartheta_a)$ has a perpetual infinite derivation
   there exist infinitely many substitutions ϑ_r, where ϑ_r is the substitution computed at the r-th resolution step, restricted to $Vars(G)$
   $$\forall r \ (A_1, \ldots, A_k)(\vartheta \cdot \vartheta_a \cdot \vartheta_r) \text{ does not fail} \Rightarrow$$
   $$\forall r \ A_j(\vartheta \cdot \vartheta_a \cdot \vartheta_r) \in \sigma(P) \text{ so}$$
   $$A_j(\vartheta \cdot \vartheta_a \cdot \vartheta_r) \in \sigma(P)$$
   $$A_j(\vartheta \cdot \vartheta_a \cdot \vartheta_{r+1}) \in \sigma(P)$$

$$A_j(\vartheta \cdot \vartheta_a \cdot \vartheta_{r+2}) \in \sigma(P)$$

$$\vdots$$

Recalling that $Vars((A_j(\vartheta \cdot \vartheta_a)) \subseteq Per(G)$, we can always find indexes of substitutions, computed during the infinite perpetual derivation, such that

$$A_j(\vartheta \cdot \vartheta_a)\vartheta_{s_1} < A_j(\vartheta \cdot \vartheta_a)\vartheta_{s_2} < \ldots\ldots \text{ where } s_j < s_i \text{ if } j < i.$$

There exists an infinite descending chain of instances of the atom $A_j(\vartheta \cdot \vartheta_a)$, which belong to $\sigma(P)$,which are $A_j(\vartheta \cdot \vartheta_a)\vartheta_{s_1}, A_j(\vartheta \cdot \vartheta_a)\vartheta_{s_2}, \ldots$.
Then there exists a representative of limit of the Chauchy sequence $A_j(\vartheta \cdot \vartheta_a)\vartheta_{s_i}$ computed with the metric in [8], which still belongs to $\sigma(P)$.
This representative of the limit element belongs also to the set $Min(\sigma(P))$. This is because the chain is infinite and there exists no other atom, not belonging to the chain, which is more instantiated than all the atoms in an infinite descending chain.
It is worth noting that our argument for A_j holds for every atom belonging to the goal such that $Vars(A_j(\vartheta \cdot \vartheta_a)) \neq \emptyset$.
Since all the computed substitutions are the same for every A_j, then there exists a substitution $(\vartheta \cdot \vartheta_a \cdot \vartheta_{lim})$ such that

$$\forall\, i \;\; A_i(\vartheta \cdot \vartheta_a \cdot \vartheta_{lim}) \in Min(\sigma(P)).$$

By hypothesis, such a substitution does not exist. ◇

We consider now examples 4.2, 4.3 and 4.4 to show how the above result can be applied. For the sake of simplicity, when it will be easy to check, we will write $p(\tilde{x})\vartheta$ instead of $p(\tilde{x})\Theta$, where Θ is an ideal of substitutions, if ϑ is the substitution represented by the ideal Θ.

Example 4.5

```
P1 :  p(a)
      q(b):-q(b)
      s(a)
```

$\overline{NGFF}_{P_1} = \{p(a), p(X), q(b), q(X), s(a), s(X)\}$
$Min(\overline{NGFF}_{P_1}) = \{p(a), q(b), s(a)\}$.
The goal $\leftarrow p(X), q(X)$ *has a finitely failed fair SLD-tree in* P_1 *since there exists no* ϑ *such that* $(p(X), q(X))\vartheta \in Min(\overline{NGFF}_{P_1})$.

Example 4.6

```
P2 :  p(X):-p(X)
      q(b)
      s(a)
```

$\overline{NGFF}_{P_2} = \{p(a), p(b), p(X), q(b), q(X), s(a), s(X)\}$
$Min(\overline{NGFF}_{P_1}) = \{p(a), p(b), q(a), s(a)\}$
The goal $\leftarrow p(X), q(X)$ *does not have a finitely failed fair SLD-tree in* P_1 *since there exists* $\vartheta = \{X/b\}$ *such that* $(p(X), q(X))\vartheta \in Min(\overline{NGFF}_{P_1})$.

Example 4.7

```
P3 :  q(f(X)):-q(X)
      p(X)
```

$$\overline{NGFF}_{P_3} = \{\; q(X), q(f(X)), q(f(f(X))), \ldots \quad p(X), p(f(X)), p(f(f(X))), \ldots \quad p(a), p(f(a)), p(f(f(a))), \ldots\}.$$

$Min(\overline{NGFF}_{P_1}) = \{p(X)\Theta, q(X)\Theta, q(a), q(f(a)), q(f(f(a))), \ldots\}$
$\Theta = \{\epsilon, \{x = f(x)\}, \{x = f(f(x))\}, \ldots, \{x = f^j(x)\}, \ldots\}$
and $\vartheta = f^\omega(x)$ *is the substitution represented by the ideal of substitutions* Θ.
The goal $\leftarrow p(X), q(X)$ *does not finitely fail in* P_3 *since there exists the substitution* ϑ, *which binds* X *to the limit element* $f^\omega(X)$ *such that* $(p(X), q(X))\vartheta \in Min(\overline{NGFF}_{P_1})$.

It is worth noting that the set $Min(\overline{NGFF})$ does not in general contain ground atoms only, as shown by the last example. In general $Min(\overline{NGFF})$ contains ground atoms and non-ground atoms which are representatives of the limits of infinite chains.

As we have already explained at the begining of this section the and–compositionality of $NGFF_P$ implies that $NGFF_P$ is also correct wrt $\approx_{FF}$.

Corollary 4.1 (Correctness) *Let* P *and* Q *be programs.*
If $NGFF_P = NGFF_Q$ *then* $P \approx_{FF} Q$

5 $NGFF(P)$ is fully abstract

We have just proved that $NGFF_P$ is correct wrt finite failures and it is also and-compositional. The correctness wrt the observational behavior we want to model is a necessary property for any sensible semantics.

There exists also another property we can be interested in, i.e. full abstraction. Full abstraction means that if two programs have the same behavior, then they must have the same semantics, i.e.
$P_1 \approx_{FF} P_2 \Longrightarrow \sigma(P_1) = \sigma(P_2)$
In the case of finite failures the full abstraction property guarantees that the semantics is "minimal" and there exists no better semantics.

The following theorem states that $NGFF$ is indeed fully abstract wrt $\approx_{FF}$.

Theorem 5.1 (Full abstraction) *Let* P *and* Q *be two programs.*
If $P \approx_{FF} Q$ *then* $NGFF_P = NGFF_Q$.

Proof
Assume $P \approx_{FF} Q$ and $NGFF_P \neq NGFF_Q$. Then we are in one of the following cases

- $\exists A : A \in NGFF_P$ and $A \notin NGFF_Q$. Then if we consider the goal $G = \leftarrow A$, G has a finitely failed fair SLD-tree in P but G has no finitely failed SLD-tree in Q. This gives a contradiction.
- $\exists A : A \in NGFF_Q$ and $A \notin NGFF_P$. Then if we consider the goal $G = \leftarrow A$, G has no finitely failed fair SLD-tree in Q but G has not a finitely failed SLD-tree in P. This gives a contradiction. ◇

6 Conclusion

We have shown that $NGFF$ is correct, fully abstract and and-compositional wrt finite failures.

The problem of defining a denotation correct (and fully abstract) with respect to finite failures was an open problem in the theory of logic programs, interesting in the framework of program trasformations. In fact, if we want a program trasformation technique to preserve the semantics, we should require it to preserve computed answers and finite failures, i.e. the s-semantics (which models computed answers) and our finite failures semantics $NGFF$.

The and-compositionality theorem might be useful to simplify the proof of the usual theorems concerned with finite failures and the related theory of Negation as Failure.

We have not considered in this paper another property one can be interested in, i.e. compositionality wrt union of clauses, sometimes called or-compositionality. Or-compositionality is important if one wants to be able to reason about programs in a modular way. It is easy to realize that $NGFF$ is not or-compositional. An or-compositional semantics for finite failures can be found in [7]. It consists of more concrete abstractions of SLD-trees, namely sequences of resultants. The semantics is also and-compositional but not fully abstract.

References

[1] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier, Amsterdam and The MIT Press, Cambridge, 1990.

[2] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 1994. to appear.

[3] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A new Declarative Semantics for Logic Languages. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 993–1005. The MIT Press, Cambridge, Mass., 1988.

[4] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989.

[5] M. Gabbrielli. *The Semantics of Logic Programming as a Programming Language.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 1993.

[6] M. Gabbrielli, G. Levi, and M. C. Meo. Observational Equivalences for Logic Programs. In K. Apt, editor, *Proc. Joint Int'l Conf. and Symposium on Logic Programming*, pages 131–145. The MIT Press, Cambridge, Mass., 1992.

[7] R. Gori. Una semantica corretta e composizionale per l'osservabile fallimento finito. Master's thesis, Dipartimento di Informatica, Università di Pisa, 1994. in italian.

[8] G. Levi, M. Martelli, and C. Palamidessi. Failure and success made symmetric. In S. K. Debray and M. Hermenegildo, editors, *Proc. North American Conf. on Logic Programming'90*, pages 3–22. The MIT Press, Cambridge, Mass., 1990.

[9] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.

Author Index

C. Aguilera, II-438
E. Alba, II-60
J.F. Aldana, II-60
M. Alpuente, I-62
V. Ambriola, II-132
A. Analyti, I-434
D. Aquilino, II-440
F. Arcelli, II-253
P. Arenas, II-117
P. Asirelli, II-440
R. Bagnara, I-312
M. Baldoni, II-324
S. Bertarello, II-16
E. Bertino, II-206
K. Bohlmann, I-234
A. Brogi, II-408
P. Bruscoli, II-221
F. Buccafurri, I-449
F. Bueno, I-281
F. Buffoli, I-155
M. Bugliesi, II-161
J.C. Casamayor, II-32
B. Catania, II-206
M. Celma, II-75
I. Cervesato, II-336
C. Chiopris, II-102
L. Chittaro, II-336
G.A. Cignoni, II-132
S. Clérici, II-423
M. Codish, I-327
M. Comini, I-170
A. Cortesi, I-372
S. Costantini, II-16
J. Cuevas, II-191,II-446
V. Dahl, II-268
H. Decker, II-32
B. Demoen, I-327
J. Devesa, II-191,II-446
G. Dore, II-366
A. Dovier, I-403
F.J. Durán, II-147
M. Enciso, II-309
G. Escalada, II-1
S. Etalle, I-218
M. Fabris, II-102
M. Falaschi, I-62,I-140
F. Ferrucci, I-388,II-379
G. Filé, I-357,I-372
F. Formato, II-253
J.L. Freire, II-351
M. Gabbrielli, I-140,I-218
J.L. Galán, II-438
L.A. Galán, II-393
J.E. Gallardo, I-266
M.M. Gallardo, I-342
C. García, II-75
R. Giacobazzi, I-77
A. Gil, II-117
L. Giordano, II-324
H.-J. Goltz, I-31
A. González, II-452
R. Gori, I-464
B.A. Grima, II-176
P. Guerrero, I-266
G. Guerrini, II-206
A. Guglielmi, II-221
V.M. Gulías, II-351
W. Hans, II-238
Y.-N. Huang, II-268
G. Iannello, II-253
P. Inverardi, II-440
H.M. Jamil, II-161
J.-P. Jouannaud, I-202
G.A. Lanzarone, II-16
B. Le Charlier, I-92
N. Leone, I-449
G. Levi, I-170,I-464
N.-W. Lin, I-297
L. Liquori, II-296
J.W. Lloyd, I-3
V. Loia, II-379
R. Loogen, I-234
S. Lucas, I-125
U. Manfredi, II-366,II-442

K. Marriott, I-140
A. Martelli, II-324
M. Martelli, I-187,II-206
P. Mascellani, I-46
A. Messora, I-187
J.M. Molinelli, II-351
A. Montanari, II-336
D. Montesi, II-206
L. Moreno, II-283
J.J. Moreno, I-2
N. Mylonakis, II-444
R. Nieuwenhuis, I-1
M. Nitsche, I-18
M. Núñez, II-393
M. Ojeda, II-438
J. Oliver, I-125
E.G. Omodeo, I-403
Y. Ortega, I-234
G. Pacini, I-388,II-379
C. Palamidessi, I-140,I-187
M. Palomar, II-283
C. Pareja, II-393
M.A. Pastor, II-75
O. Pastor, II-446
D. Pedreschi, I-46,I-418
R. Peña, II-393,II-423
I. Pérez, II-309,II-438
J. Pérez, II-444
A. Pettorossi, I-203
E. Pimentel, II-147
G. Plagenza, II-47
A. Policriti, I-403
S. Pramanik, I-434
M. Proietti, I-203
M.J. Ramis, I-62
G. Ramos, II-448
I. Ramos, II-191,II-446
F. Ranzato, I-357
C. Renso, II-408
C. Rossi, II-309
G. Rossi, I-403
S. Rossi, I-92
A. Rubio, I-1
C. Ruggieri, II-88
S. Ruggieri, I-418
B.C. Ruiz, I-266
P. Rullo, I-449
J.J. Ruz, II-238
F. Sáenz, II-238
M. Sancassani, II-88,II-366,II-442
M.L. Sapino, II-296
L. Semini, II-132
M.I. Sessa, I-388,II-379
P. Tarau, II-268
A. Tirabosco, II-102
M. Toro, II-450
J. Torres, II-450
A. Toval, II-176
J.M. Troya, I-342,II-60,II-147
J.A. Troyano, II-450
J. Tubella, II-452
F. Turini, II-408
P. Van Hentenryck, I-92
P. Van Roy, I-296
R. Varela, I-251
G. Vidal, I-62
L. Vila, II-1
P. Volpe, I-107
S. Winkler, II-238
W. Winsborough, I-372