Preferred Answer Sets for Ordered Logic Programs

DAVY VAN NIEUWENBORGH[∗] and DIRK VERMEIR[†]

Vrije Universiteit Brussel
Dept. of Computer Science
Pleinlaan 2, B-1050 Brussel
Belgium
(*e-mail:* dvnieuwe@vub.ac.be)
(*e-mail:* dvermeir@vub.ac.be)

submitted November 27, 2003; revised June 6, 2004; accepted July 19, 2004

Abstract

We extend answer set semantics to deal with inconsistent programs (containing classical negation), by finding a “best” answer set. Within the context of inconsistent programs, it is natural to have a partial order on rules, representing a preference for satisfying certain rules, possibly at the cost of violating less important ones. We show that such a rule order induces a natural order on extended answer sets, the minimal elements of which we call preferred answer sets. We characterize the expressiveness of the resulting semantics and show that it can simulate negation as failure, disjunction and some other formalisms such as logic programs with ordered disjunction. The approach is shown to be useful in several application areas, e.g. repairing database, where minimal repairs correspond to preferred answer sets.

To appear in Theory and Practice of Logic Programming (TPLP).

KEYWORDS: nonmonotonic reasoning, knowledge representation,answer set programming, preference

1 Introduction

The intuition behind the stable model semantics (Gelfond and Lifschitz 1988), and, more generally, behind answer set semantics (Gelfond and Lifschitz 1991) for (extended) logic programs is both intuitive and elegant. Given a program P and a candidate answer set M, one computes a reduct program P_M of a simpler type for which a semantics $P^\star_M$ is known. The reduct P_M is obtained from P by taking into account the consequences of accepting the proposed truth values of the literals in M. The candidate set M is then an answer set just when $P^\star_M = M$, i.e. M is “self-supporting”.

In this paper, we apply this reduction technique to deal with inconsistent programs, e.g. programs with (only) classical negation (denoted as $\neg$) where the immediate consequence operator would yield inconsistent interpretations. For example, computing the least fixpoint of the program $\{a \leftarrow\ ,\ b \leftarrow\ , \neg a \leftarrow b\}$, where negative literals $\neg a$ are considered as

∗ Supported by the FWO
† This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project

fresh atoms, yields the inconsistent $\{a, b, \neg a\}$. To prevent this, we will allow for a rule to be defeated by an opposing rule w.r.t. an interpretation. In the example, $\{a, b\}$ will be accepted because the rule $\neg a \leftarrow b$ is defeated by the rule $a \leftarrow$. The definition of answer set remains the same (see, e.g., (Lifschitz 2002)), but the reduct is restricted to rules that are not defeated. We show that the *extended answer set semantics* thus obtained can be simulated by an extended logic program $E(P)$ that is trivially constructed from the original program P.

The above technique can be generalized to *ordered programs* where a partial order, representing preference or specificity, is defined on the rules of a program. E.g. one may prefer certain “constraint” rules to be satisfied, possibly at the expense of defeating less important “optional” or “default” rules. We show that such a preference structure on the rules induces a natural partial order on the reducts of the program, and hence on its candidate (extended) answer sets. Minimal elements in this induced partial order are called *preferred answer sets*.

Intuitively, an answer set M_1 is preferred over an answer set M_2 if any rule r_2 that is satisfied by M_2 but not by M_1, is “countered” by a more preferred (than r_2) rule r_1 that is satisfied by M_1 but not by M_2. In other words, with preferred answer sets, one tries to maximize rule satisfaction, taking into account the relative “priority” of the rules. The approach has some immediate applications in e.g. diagnostic systems, as illustrated in the example below.

Example 1
Consider the problem of diagnosing a simple system where the light fails to come on. The normal operation of the system is described using the rules r_1, r_2 and r_3.

$$\begin{array}{lrcl} r_1 & \mathit{light} & \leftarrow & \mathit{power}, \mathit{bulb} \\ r_2 & \mathit{power} & \leftarrow & \\ r_3 & \mathit{bulb} & \leftarrow & \end{array}$$

which, by themselves, yield light. The fault model is given by the following rules (r_4 and r_5), which indicate that the power may fail and the bulb may be broken.

$$\begin{array}{lrc} r_4 & \neg\mathit{power} & \leftarrow \\ r_5 & \neg\mathit{bulb} & \leftarrow \end{array}$$

Finally, the observation that something is wrong is encoded by the constraint-like rule r_6, i.e. a rule that can only be satisfied when there is no light.

$$r_6 \quad \neg\mathit{light} \quad \leftarrow \quad \mathit{light}$$

Obviously, the program $\{r_1, r_2, r_3, r_4, r_5, r_6\}$ is inconsistent. On the other hand, it is natural to structure the rules in a preference hierarchy where r_1 (the “law” governing the system) and r_6 (the observation) are most preferred. Slightly less preferred (than r_1 and r_6) are the assumptions r_2 and r_3 representing normal system operation. Finally, the fault rules r_4 and r_5 are least preferred, indicating that, if the program is inconsistent, such rules will be the first to be considered for defeat.

Without the observation r_6, the program is still inconsistent and, following the preference relation on the rules, $M_1 = \{\mathit{bulb}, \mathit{power}, \mathit{light}\}$ will be a preferred answer set satisfying all but the least preferred (r_4 and r_5) rules. If we take into account the observation r_6, M_1, which does not satisfy r_6, will turn out to be less preferred than either of $M_2 = \{\neg bulb, power\}$ or $M_3 = \{bulb, \neg power\}$. E.g. M_2 is preferred over M_1 because, unlike M_1, M_2 satisfies r_6 which counters the non-satisfaction of r_3 by M_2. It can be verified that both M_2 and M_3 are preferred answer sets, with each corresponding to a minimal explanation of the observation.

Besides diagnostic systems, ordered logic programs may be useful in other application areas. E.g. we show that the minimal repairs of a database D (Arenas et al. 2000) w.r.t. a set of constraints C correspond with the preferred answer sets of an ordered program where the constraints C are preferred over D, which is itself preferred over $\neg D$, the latter representing the negation of the facts in D.

Although simple, ordered programs turn out to be rather expressive under the preferred answer set semantics. E.g., it is possible to simulate both negation as failure and disjunction in classical non-ordered programs.

Negation as failure has a long history, starting from the Clark completion (Clark 1978), over stable model semantics (Gelfond and Lifschitz 1988) and well-founded semantics (van Gelder et al. 1988), to answer set programming (Gelfond and Lifschitz 1991; Lifschitz 2002). It is well-known that adding negation as failure to programs results in a more expressive formalism. However, in the context of disjunctive logic programming (Przymusinski 1991; Leone et al. 1997), (Inoue and Sakama 1998) demonstrated that adding negation as failure positively in a program, i.e. in the head of the rules, yields no extra computational power to the formalism. One of the more interesting features of negation as failure in the head is that answers no longer have to be minimal w.r.t. subset inclusion (e.g. the program $\{a \vee \textit{not}\ a \leftarrow\}$ has both $\{a\}$ and $\emptyset$ as answer sets). Indeed, such minimality turns out to be too demanding to express certain problems, e.g. in the areas of abductive logic programming (Kakas et al. 1992; Inoue and Sakama 1996) or logic programming with ordered disjunction (Brewka 2002; Brewka et al. 2002).

In light of the above, it is natural to consider *extended ordered programs* where negation as failure is allowed in both the head and the body of a clause. Just as for disjunctive logic programs, adding negation as failure positively results in a formalism where answer sets are not anymore guaranteed to be subset minimal. Nevertheless, we will present a construction that translates an extended ordered program into a semantically equivalent ordered program without negation as failure, thus demonstrating that negation as failure does not increase the expressiveness of ordered programs.

Although extended ordered programs do not improve on ordered programs w.r.t. computational power, they can be used profitably to express certain problems in a more natural way. They also support the simulation of certain extensions of answer set programming, which we illustrate by two intuitive transformations that translate more complex concepts, i.e. ordered disjunction (Brewka 2002; Brewka et al. 2002) and consistency-restoring rules (Balduccini and Gelfond 2003; Balduccini and Mellarkod 2003), into equivalent extended ordered programs. This demonstrates that ordered programs can be used successfully as an implementation vehicle for such high level extensions of answer set programming, where the translation is processed by an ordered logic program solver such as OLPS (Section 3.2).

The remainder of this paper is organized as follows. After some preliminary notions and notations, Section 2 presents an extension of the usual answer set semantics to cover also inconsistent simple programs (without disjunction or negation as failure).

In Section 3.1,we introduce ordered programs where rules are partially ordered according to preference. It is shown that the rule-order induces a partial order on extended answer sets. The minimal elements in the latter order are called preferred answer sets. We characterize the expressiveness of the resulting semantics and show that it can simulate negation as failure as well as disjunction. Section 3.2 proposes an algorithm to compute such preferred answer sets and shows that the complexity is the same as for disjunctive logic programming. In Section 3.3, we show that adding negation as failure to ordered programs does not yield any extra expressive power.

The relation of preferred answer set semantics with similar formalisms from the literature is discussed in Section 4. We consider Brewka's preferred answer sets (Brewka and Eiter 1999) in Section 4.1, together with D- and W-preferred answer sets (Delgrande et al. 2000; Wang et al. 2000). It turns out that these semantics are not related to our framework as they yield, in general, different preferred answer sets, that are sometimes less intuitive than the ones resulting from the semantics in Section 3.1. Section 4.2 shows that logic programs with ordered disjunction (Brewka 2002) have a natural simulation using ordered programs with preferred answer sets. In Section 4.3 we elaborate on the simulation of consistency-restoring rules (Balduccini and Gelfond 2003) using the preferred answer set semantics. In Section 4.4, we compare our semantics with $\mathcal{DOL}$ (Buccafurri et al. 1998), $\mathcal{DLP}^<$ (Buccafurri et al. 1999) and ordered logic (Laenens and Vermeir 1990). Section 5 illustrates another application of the preferred answer set semantics: the minimal repairs of a database D w.r.t. a set of constraints C can be obtained as the preferred answer sets of an ordered program $P(C, D)$.

In Section 6 we conclude and give some directions for further research.

To increase readability, several proofs and lemmas have been moved to the appendix.

2 Extended Answer Sets for Simple Programs

2.1 Preliminaries and Notation

We use the following basic definitions and notation.

Literals

A *literal* is an *atom* a or a negated atom $\neg a$. An *extended literal* is a literal or of the form *not* l where l is a literal. The latter form is called a *naf-literal* and denotes negation as failure: *not* l is interpreted as "l is not true". We use $\hat{l}$ to denote the ordinary literal underlying an extended literal, i.e. $\hat{l} = a$ if $l =$ *not* a while $\hat{a} = a$ if a is an ordinary literal. Both notations are extended to sets so $\hat{X} = \{\hat{e} \mid e \in X\}$, with X a set of extended literals, while *not* $Y = \{$*not* $l \mid l \in Y\}$ for any set of (ordinary) literals Y.

For a set of (ordinary) literals X we use $\neg X$ to denote $\{\neg p \mid p \in X\}$ where $\neg(\neg a) \equiv a$. Also, X^+ denotes the positive part of X, i.e. $X^+ = \{a \in X \mid a \text{ is an atom}\}$. The *Herbrand base* of X, denoted $\mathcal{B}_X$, contains all atoms appearing in X, i.e. $\mathcal{B}_X = (X \cup \neg X)^+$. A set I of literals is *consistent* if $I \cap \neg I = \emptyset$.

For a set of extended literals X, we use X^- to denote the literals underlying elements of X that are not ordinary literals, i.e. $X^- = \{l \mid \textit{not}\, l \in X\}$. We say that X is consistent iff the set of ordinary literals $\neg X^- \cup (X \backslash \textit{not}\, X^-)$ is consistent.

Programs

An *extended disjunctive logic program* (EDLP, see e.g. (Lifschitz 2002)) is a countable set of rules of the form $\alpha \leftarrow \beta$ where $\alpha \cup \beta$ is a finite set of extended literals. In a *disjunctive logic program* (DLP), the head α of each rule $\alpha \leftarrow \beta$ must contain only ordinary literals.

If always $|\alpha| \leq 1$, i.e. α is a singleton or empty, we drop the "disjunctive" qualification. If, for all rules, all literals in $\alpha \cup \hat{\beta}$ are atoms, the program is called *seminegative* and if, furthermore, each rule satisfies $\beta^- = \emptyset$, the program is said to be *positive*.

Answer sets

The *Herbrand base* $\mathcal{B}_P$ of and EDLP P contains all atoms appearing in P. An *interpretation* I of P is any consistent subset of $\mathcal{B}_P \cup \neg\mathcal{B}_P$ (for seminegative programs, we can restrict to a set of atoms). An interpretation I is *total* if $\mathcal{B}_P \subseteq I \cup \neg I$.

An extended literal l is true w.r.t. an interpretation I, denoted $I \models l$ if $l \in I$ in case l is ordinary, or $I \not\models a$ if $l = \textit{not}\, a$ for some ordinary literal a. As usual, $I \models X$ for some set of (extended) literals l iff $\forall l \in X \cdot I \models l$.

A rule $r = \alpha \leftarrow \beta$ is *satisfied* by I, denoted $I \models r$, if $I \models l$ for some $l \in \alpha$ and $\alpha \neq \emptyset$, whenever $I \models \beta$, i.e. if r is *applicable* ($I \models \beta$), then it must be *applied* ($\exists l \in \alpha \cdot I \models \beta \cup \{l\}$). As a consequence, a *constraint*, i.e. a rule with empty head ($\alpha = \emptyset$), can only be satisfied if it is not applicable ($I \not\models \beta$).

For a DLP P without negation as failure ($\beta^- = \emptyset$), an *answer set* is a minimal (w.r.t. set inclusion) interpretation I that is *closed* under the rules of P (i.e. $\forall r \in P \cdot I \models r$).

For an EDLP P containing negation as failure and an interpretation I, the Gelfond-Lifschitz transformation (Gelfond and Lifschitz 1988) yields the *GL-reduct* program P^I that consists of those rules $(\alpha \backslash \textit{not}\, \alpha^-) \leftarrow (\beta \backslash \textit{not}\, \beta^-)$ where $\alpha \leftarrow \beta$ is in P, $I \models \textit{not}\, \beta^-$ and $I \models \alpha^-$.

Thus, P^I is obtained from P by (a) removing all true naf-literals *not* a, $a \notin I$, from the bodies of rules in P, (b) removing all false naf-literals *not* a, $a \in I$ from the heads of rules in P, and (c) keeping in P^I only the transformed rules that are free from negation as failure. An interpretation I is then an *answer set* of P iff I is an answer set of the reduct P^I.

Reducts

In this paper, we use the term "reduct" of a program, w.r.t. an interpretation, to denote the set of rules that are satisfied w.r.t. the interpretation.

Definition 1
Let P be an EDLP program. The **reduct** $P_I \subseteq P$ of P w.r.t. an interpretation I contains just the rules satisfied by I, i.e. $P_I = \{r \in P \mid I \models r\}$.

Naturally, $P_M = P$ for any answer set M of P.

2.2 Simple Programs and Extended Answer Sets

In this section, we consider simple logic programs which are logic programs with only classical negation and no disjunction in the head of a rule.

Definition 2
A **simple logic program** (SLP) is a countable set P of **rules** of the form $\alpha \leftarrow \beta$ where $\alpha \cup \beta$ is a finite set of literals[1] and $|\alpha| \leq 1$, i.e. α is a singleton or empty.

A rule $r = a \leftarrow \beta$ is **defeated** w.r.t. an interpretation I iff there exists an applied (w.r.t. I) **competing rule** $\neg a \leftarrow \beta'$ in P; such a rule is said to **defeat** r.

We will often confuse a singleton set with its sole element, writing rules as $a \leftarrow \beta$ or $\leftarrow \beta$. Thus, a rule $r = a \leftarrow \beta$ cannot be left unsatisfied unless one accepts the opposite conclusion $\neg a$ which is motivated by a competing applied rule $\neg a \leftarrow \beta'$ that *defeats* r. Obviously, it follows that a constraint can never be defeated.

Example 2
Consider the SLP P containing the following rules.

$$\begin{array}{rr} \neg a \leftarrow & \neg b \leftarrow \\ a \leftarrow \neg b & b \leftarrow \neg a \end{array}$$

For the interpretation $I = \{\neg a, b\}$ we have that I satisfies all rules in P but one: $\neg a \leftarrow$ and $b \leftarrow \neg a$ are applied while $a \leftarrow \neg b$ is not applicable. The unsatisfied rule $\neg b \leftarrow$ is defeated by $b \leftarrow \neg a$.

For a set of rules R, we use $R^\star$ to denote the unique minimal (van Emden and Kowalski 1976) model of the positive logic program consisting of the rules in R where (a) negative literals $\neg a$ are considered as fresh atoms and (b) constraint rules $\leftarrow \beta$ are replaced by rules of the form $\perp \leftarrow \beta$. Besides the normal notion of inconsistency, a set of literals containing $\perp$ is also considered inconsistent. Clearly, the * operator is monotonic.

For the program of Example 2, we have that $P^\star = \{\neg a, \neg b, a, b\}$ is inconsistent. The following definition allows us to not apply certain rules, when computing a consistent interpretation for programs such as P.

Definition 3
An interpretation I of a SLP P is **founded** iff $P_I^\star = I$. A founded interpretation I is an **extended answer set** of P if all rules in P are satisfied or defeated.

The following is a straightforward consequence of the above definition and the fact that a simple (reduct) program has at most one answer set.

Theorem 1
An interpretation M is an extended answer set of a SLP P iff M is the unique answer set (Section 2.1) of P_M and every rule in $P \backslash P_M$ is defeated w.r.t M.

Thus, the extended answer set semantics deals with inconsistency in a simple yet intuitive way: when faced with contradictory applicable rules, just select one for application

[1] As usual, we assume that programs have already been grounded.

and ignore (defeat) the other. In the absence of extra information (e.g. regarding a preference for satisfying certain rules at the expense of others), this seems a reasonable strategy for extracting a consistent semantics from inconsistent programs.

Using the above definition, it is easy to verify that the program P from Example 2 has three extended answer sets, namely $M_1 = \{\neg a, b\}$, $M_2 = \{a, \neg b\}$ and $M_3 = \{\neg a, \neg b\}$. Note that $P_{M_1} = P \backslash \{\neg b \leftarrow\ \}$ while $P_{M_2} = P \backslash \{\neg a \leftarrow\ \}$, and $P_{M_3} = P \backslash \{a \leftarrow \neg b, b \leftarrow \neg a\}$, i.e. $\neg b \leftarrow$ is defeated w.r.t. M_1, $\neg a \leftarrow$ is defeated w.r.t. M_2 and both $a \leftarrow \neg b$ and $b \leftarrow \neg a$ are defeated w.r.t. M_3.

The definition of extended answer set is rather similar to the definition of answer sets for (non-disjunctive) programs without negation as failure: the only non-technical difference being that, for extended answer sets, a rule may be left unsatisfied if it is defeated by a competing (i.e. a rule with opposite head) rule. This is confirmed by the following theorem.

Theorem 2

Let P be a SLP and let M be an answer set of P. Then, M is the unique extended answer set of P.

Proof

By definition, M is a minimal consistent interpretation that satisfies all rules in P. The latter implies that $P_M = P$. Because M is minimal, it follows that $M = P_M^\star$, making M founded. Obviously, M must be unique. $\square$

While allowing for P_M, with M an extended answer set, to be a strict subset of P, Definition 3 still maximizes the set of satisfied rules w.r.t. an extended answer set.

Theorem 3

Let P be a SLP and let M be an extended answer set for P. Then, P_M is maximal w.r.t. $\subseteq$ among the reducts of founded interpretations of P.

Proof

Assume that, on the contrary, P_M is not maximal, i.e. there exists a founded interpretation N such that $P_M \subset P_N$. From the monotonicity of the $\star$-operator, it follows that $M \subseteq N$. As M is an extended answer set, all constraints are included in P_M, thus, by $P_M \subset P_N$, also in P_N. So, $P_N \backslash P_M$ does not contain any constraint. Let $r = (a \leftarrow \beta) \in P_N \backslash P_M$. Since r is not satisfied w.r.t. M, it must be the case that $\beta \subseteq M$ while $a \notin M$. Because M is an extended answer set, r must have been defeated by an applied rule $r' = (\neg a \leftarrow \beta') \in P_M \subset P_N$ and, consequently, $\neg a \in M \subseteq N$. On the other hand, $\beta \subseteq N$ and thus, since $r \in P_N$, r must be applied w.r.t. N, yielding that $a \in N$. This makes N inconsistent, a contradiction. $\square$

The reverse of Theorem 3 does not hold in general, as can be seen from the following example.

Example 3
Consider the program P containing the following rules.

$$\begin{aligned} \neg a &\leftarrow \\ b &\leftarrow \\ \neg b &\leftarrow \neg a \end{aligned}$$

The interpretation $N = \{b\}$ is founded with $P_N = \{b \leftarrow\ , \neg b \leftarrow \neg a\}$ which is obviously maximal since $P^\star$ is inconsistent. Still, N is not an extended answer set because $\neg a \leftarrow$ is not defeated.

However, when considering simple programs without constraints, for total interpretations, founded interpretations with maximal reducts are extended answer sets.

Theorem 4
Let P be a SLP without constraints and let M be a total founded interpretation such that P_M is maximal among the reducts of founded interpretations of P. Then, M is an extended answer set.

Proof
It suffices to show that each unsatisfied rule is defeated w.r.t. M. Assume that, on the contrary, $r = (a \leftarrow \beta) \in P \backslash P_M$ is not defeated, i.e. $a \notin M$ while $\beta \subseteq M$ and there is no applied competitor $\neg a \leftarrow \beta'$. But then also $\neg a \notin M$, contradicting the fact that M is total. $\square$

The need for programs to be constraint free in the previous theorem is demonstrated by the following example.

Example 4
Consider the program P containing the following rules.

$$a \leftarrow \qquad \neg a \leftarrow \qquad \leftarrow a$$

The total interpretation $N = \{a\}$ is founded with $P_N = \{a \leftarrow\ \}$ which is obviously maximal. However, N is not an extended answer set as the constraint $\leftarrow a$ is neither satisfied nor defeated w.r.t. N.

The computation of extended answer sets reduces to the computation of answer sets for seminegative non-disjunctive logic programs, using the following transformation, which is similar to the one used in (Kowalski and Sadri 1990) for logic programs with exceptions.

Definition 4
Let P be a SLP. The **extended version** $E(P)$ of P is the (non-disjunctive) logic program obtained from P by replacing each rule $a \leftarrow \beta$ by its extended version $a \leftarrow \beta,$ *not* $\neg a$.

Note that the above definition captures our intuition about defeat: one can ignore an applicable rule $a \leftarrow \beta$ if it is defeated by evidence for the contrary $\neg a$, thus making *not* $\neg a$ false and the rule $a \leftarrow \beta,$ *not* $\neg a$ not applicable.

Theorem 5
Let P be a SLP. The extended answer sets of P coincide with the answer sets of $E(P)$.

When considering programs without constraints, the extended answer set semantics is universal.

Theorem 6
Each simple logic program without constraints has extended answer sets.

Proof
Let P be a simple logic program without constraints. Define $\delta_P : 2^{\mathcal{B}_P} \rightarrow 2^{\mathcal{B}_P}$ by

$$\delta_P(I) = \{a \notin I \mid \neg a \notin I \wedge \exists (a \leftarrow \beta) \in P \cdot \beta \subseteq I\}$$

Then, clearly, any sequence $I_0 = \emptyset, I_1, \ldots$ where, for $i \geq 0$, $I_{i+1} = I_i \cup \{a\}$ for some $a \in \delta_P(I_i)$ if $\delta_P(I_i) \neq \emptyset$, and $I_{i+1} = I_i$ otherwise, is monotonically increasing and thus reaches a fixpoint $I^\star$ which is easily verified to be an extended answer set.

Note that a similar result is well-known for normal default logic(Reiter 1980). □

3 Ordered Programs and Preferred Answer Sets

3.1 Definitions and Basic Results

When constructing extended answer sets for simple logic programs, one can defeat any rule for which there is an applied competing rule. In many cases, however, there is a clear preference among rules in the sense that one would rather defeat less preferred rules in order to keep the more preferred ones satisfied.

As an example, reconsider the program P from Example 2 and assume that we prefer not to defeat the rules with positive conclusion ($\{a \leftarrow \neg b,\ b \leftarrow \neg a\}$). Semantically, this should result in the rejection of $M_3 = \{\neg a, \neg b\}$ in favor of either $M_1 = \{\neg a, b\}$ or $M_2 = \{a, \neg b\}$ because the latter two sets are consistent with our preferences.

In ordered programs, such preferences are represented by a partial order on the rules of the program.

Definition 5
An **ordered logic program** (OLP) is a pair $\langle R, < \rangle$ where R is a a simple program and $<$ is a well-founded strict[2] partial order on the rules in R[3].

Intuitively, $r_1 < r_2$ indicates that r_1 is more preferred than r_2. In the examples we will often represent the order implicitly using the format

$$\begin{array}{c} \ldots \\ \hline R_2 \\ \hline R_1 \\ \hline R_0 \end{array}$$

where each $R_i, i \geq 0$, represents a set of rules, indicating that all rules below a line are more

[2] A strict partial order $<$ on a set X is a binary relation on X that is antisymmetric, anti-reflexive and transitive. The relation $<$ is well-founded if every nonempty subset of X has a $<$-minimal element.

[3] Strictly speaking, we should allow R to be a multiset or, equivalently, have labeled rules, so that the same rule can appear in several positions in the order. For the sake of simplicity of notation, we will ignore this issue in the present paper: all results also hold for the general multiset case.

preferred than any of the rules above the line, i.e. $\forall i \geq 0 \cdot \forall r_i \in R_i, r_{i+1} \in R_{i+1} \cdot r_i < r_{i+1}$ or $\forall i \geq 0 \cdot R_i < R_{i+1}$ for short.

Example 5
Consider the OLP $P = \langle R, < \rangle$ where $<$ is as shown below.

$$\begin{array}{c} f \leftarrow b \\ \hline \neg f \leftarrow p \\ \hline b \leftarrow p \\ p \leftarrow \end{array}$$

The program uses the preference order to indicate that the rule $f \leftarrow b$ ("birds fly") should be considered a "default", i.e. the rule $\neg f \leftarrow p$ ("penguins don't fly") is more preferred. The lowest rules, i.e. $b \leftarrow p$ ("penguins are birds") and $p \leftarrow$ ("the bird under consideration is a penguin"), are the "strongest" (minimal): an extended answer set for P that respects the preference order should satisfy these minimal rules, if at all possible.

For the interpretations $I_1 = \{p, b, f\}$ and $I_2 = \{p, b, \neg f\}$, the reducts are $R_{I_1} = \{f \leftarrow b,\ b \leftarrow p,\ p \leftarrow\}$ and $R_{I_2} = \{\neg f \leftarrow p,\ b \leftarrow p,\ p \leftarrow\}$, respectively. Both I_1 and I_2 are extended answer sets of P: for I_1, the unsatisfied rule $\neg f \leftarrow p$ is defeated by $f \leftarrow b$ while for I_2, the reverse holds: $f \leftarrow b$ is defeated by $\neg f \leftarrow p$.

Intuitively, if we take the preference order $<$ into account, I_2 is to be preferred over I_1 because I_2 defeats less preferred rules than does I_1. Specifically, I_2 compensates for defeating $f \leftarrow b$ by satisfying the stronger $\neg f \leftarrow p$ which is itself defeated w.r.t. I_1.

The following definition formalizes the above intuition by defining a preference relation between reducts.

Definition 6
Let $P = \langle R, < \rangle$ be an OLP. For subsets R_1 and R_2 of R we define $R_1 \sqsubseteq R_2$ iff $\forall r_2 \in R_2 \backslash R_1 \cdot \exists r_1 \in R_1 \backslash R_2 \cdot r_1 < r_2$. We write $R_1 \sqsubset R_2$ just when $R_1 \sqsubseteq R_2$ and not $R_2 \sqsubseteq R_1$.

Intuitively, a reduct R_1 is preferred over a reduct R_2 if every rule r_2 which is in R_2 but not in R_1 is "countered" by a stronger rule $r_1 < r_2$ from R_1 which is not in R_2.

According to the above definition, we obtain that, indeed, $R_{I_2} \sqsubset R_{I_1}$, for the program P from Example 5.

Note that, unlike other approaches, e.g. (Laenens and Vermeir 1992), we do not require that the stronger rule $r_1 \in R_1 \backslash R_2$ that counters a weaker rule $r_1 < r_2 \in R_2 \backslash R_1$, is applied and neither does r_1 need to be a competitor of r_2. Thus, unlike the other approaches, we do not consider rule application as somehow "stronger" than satisfaction. This is illustrated in the following example.

Example 6
Consider $P = \langle R, < \rangle$, were R is shown below and the interpretations $M_1 = \{study, pass\}$, $M_2 = \{\neg study, pass\}$, $M_3 = \{\neg study, \neg pass\}$, and $M_4 = \{study, \neg pass\}$. The program indicates a preference for not studying, a strong desire to pass[4] and an equally strong

[4] Note that, while the rule $pass \leftarrow \neg pass$ can only be satisfied by an interpretation containing $pass$, it does not

(and uncomfortable) suspicion that not studying leads to failure.

$$\begin{array}{lcl}
r_4 & : & \mathit{pass} \leftarrow \mathit{study} \\
r_3 & : & \mathit{study} \leftarrow \\
\hline
r_2 & : & \neg \mathit{study} \leftarrow \\
\hline
r_1 & : & \neg \mathit{pass} \leftarrow \neg \mathit{study} \\
r_0 & : & \mathit{pass} \leftarrow \neg \mathit{pass}
\end{array}$$

It is easily verified that $R_{M_1} \sqsubset R_{M_2}$, $R_{M_1} \sqsubset R_{M_3}$, $R_{M_1} \sqsubset R_{M_4}$ (vacuously) and $R_{M_3} \sqsubset R_{M_4}$. Here, e.g. $R_{M_1} = \{r_0, r_1, r_3, r_4\} \sqsubset R_{M_2} = \{r_0, r_2, r_4\}$ because $r_2 \in R_{M_2} \setminus R_{M_1}$ is countered by $r_1 \in R_{M_1} \setminus R_{M_2}$ which is neither applied nor a competitor of r_2.

The following theorem implies that the relation $\sqsubseteq$ is a partial order on reducts.

Theorem 7
Let $<$ be a well-founded strict partial order on a set X. The binary relation $\sqsubseteq$ on 2^X defined by $X_1 \sqsubseteq X_2$ iff $\forall x_2 \in X_2 \setminus X_1 \cdot \exists x_1 \in X_1 \setminus X_2 \cdot x_1 < x_2$ is a partial order.

Theorem 7 can be used to define a partial order on extended answer sets of R, where $\langle R, < \rangle$ is an ordered logic program.

Definition 7
Let $P = \langle R, < \rangle$ be an OLP. For M_1, M_2 extended answer sets of R, we define $M_1 \sqsubseteq M_2$ iff $R_{M_1} \sqsubseteq R_{M_2}$. As usual, $M_1 \sqsubset M_2$ iff $M_1 \sqsubseteq M_2$ and not $M_2 \sqsubseteq M_1$.

Preferred answer sets for ordered programs correspond to minimal (according to $\sqsubseteq$) extended answer sets.

Definition 8
Let $P = \langle R, < \rangle$ be an OLP. An **answer set** for P is any extended answer set of R. An answer set for P is called **preferred** if it is minimal w.r.t. $\sqsubseteq$. An answer set is called **proper** if it satisfies all minimal (according to $<$) rules in R.

Proper answer sets respect the strongest (minimal) rules of the program.

Lemma 1
Let M be a proper answer set of an OLP P. Then any more preferred answer set $N \sqsubset M$ is also proper.

Proof
Assume that, on the contrary, $N \sqsubset M$ for some answer set N which is not proper. It follows that there is some minimal rule $r \in P_M \setminus P_N$ which cannot be countered by N, contradicting that $N \sqsubset M$. $\square$

The following theorem confirms that taking the minimal (according to $\sqsubseteq$) elements among the proper answer sets is equivalent to selecting the proper elements among the preferred answer sets.

provide a justification for *pass*. Thus such rules act like constraints. However, note that, depending on where such a rule occurs, it may, unlike traditional constraints, be defeated.

Theorem 8
Let P be an OLP. The set of minimal proper answer sets of P coincides with the set of proper preferred answer sets of P.

Proof
Let M be a minimal proper answer set and suppose that, on the contrary, M is not a proper preferred answer set. Since M is proper, this would imply that M is not preferred, i.e. $N \sqsubset M$ for some answer set N. From Lemma 1, we obtain that N must also be proper, contradicting that M is a minimal proper answer set.

To show the reverse, let M be a proper preferred answer set of P. If M were not a minimal proper answer set, there would exist a proper answer set $N \sqsubset M$, contradicting that M is preferred. $\square$

The program from Example 5 has a single preferred answer set $\{p, b, \neg f\}$ which is also proper. In Example 6, $M_1 = \{pass, study\}$ is the only proper preferred answer set.

While all the previous examples have a linear ordering, the semantics also yields intuitively correct solutions in case of non-linear orderings, as witnessed by the following example.

Example 7
Consider a problem taken from (Balduccini and Mellarkod 2003). We need to take full-body exercise. Full-body exercise is achieved either by combining swimming and ball playing, or by combining weight lifting and running. We prefer running to swimming and ball playing to weight lifting, but we do not like to do more than necessary to achieve our full-body exercise. This last condition implies that we cannot have a solution containing our two most preferred sports as in that case we also need a third sport to have a full-body exercise. The ordered program P corresponding to this problem is shown below using a straightforward extension of the graphical representation defined before.

$$\mathit{lift\_weights} \leftarrow \qquad\qquad \mathit{run} \leftarrow$$
$$\mathit{play\_ball} \leftarrow \qquad\qquad \mathit{swim} \leftarrow$$
$$\neg \mathit{full\_body\_exercise} \leftarrow$$

$\neg \mathit{play\_ball} \leftarrow$	$\neg \mathit{run} \leftarrow$
$\neg \mathit{lift\_weights} \leftarrow$	$\neg \mathit{swim} \leftarrow$

$$\mathit{full\_body\_exercise} \leftarrow \mathit{lift\_weights}, \mathit{run}$$
$$\mathit{full\_body\_exercise} \leftarrow \mathit{play\_ball}, \mathit{swim}$$
$$\mathit{full\_body\_exercise} \leftarrow \neg \mathit{full\_body\_exercise}$$

The rules in the least preferred component indicate a reluctance to do any sport; they will be used only to satisfy more preferred rules. On the other hand, the rules in the most preferred component contain the conditions for a full body exercise, together with a constraint-like rule that demands such an exercise. The rules in the middle components represent our preferences for certain sports. Note that, in order to minimize the sports we need to do, preferences are expressed on the negated facts. Consequently, e.g. a preference for running over swimming is encoded as a preference for not swimming over not running.

Consider the following extended answer sets:

$$\begin{array}{rcl} M_1 & = & \{\mathit{full\_body\_exercise}, \mathit{lift\_weights}, \mathit{run}, \neg \mathit{swim}, \neg \mathit{play\_ball}\} \ , \\ M_2 & = & \{\mathit{full\_body\_exercise}, \mathit{swim}, \mathit{play\_ball}, \neg \mathit{lift\_weights}, \neg \mathit{run}\} \ , \\ M_3 & = & \{\mathit{full\_body\_exercise}, \mathit{lift\_weights}, \mathit{run}, \mathit{swim}, \neg \mathit{play\_ball}\} \ . \end{array}$$

Clearly, all extended answer sets satisfy the three most specific rules. Comparing the reducts P_{M_1} and P_{M_3} yields that $P_{M_3} \backslash P_{M_1} = \{\mathit{swim} \leftarrow\ \}$ and $P_{M_1} \backslash P_{M_3} = \{\neg \mathit{swim} \leftarrow\ \}$. From $\neg \mathit{swim} \leftarrow\ < \mathit{swim} \leftarrow$, it then follows that M_1 is preferred over M_3, fitting our desire that we do not like to do more than necessary.

As for M_1 and M_2, it appears that the rule $\neg \mathit{play\_ball} \leftarrow\ \in P_{M_1} \backslash P_{M_2}$ is countered by the rule $\neg \mathit{lift\_weights} \leftarrow\ \in P_{M_2} \backslash P_{M_1}$. However, there is no rule in in $P_{M_2} \backslash P_{M_1}$ to counter $\neg \mathit{swim} \leftarrow\ \in P_{M_1} \backslash P_{M_2}$, and thus $M_2 \not\sqsubseteq M_1$. On the other hand, M_1 cannot counter $\neg \mathit{lift\_weights} \leftarrow\ \in P_{M_2} \backslash P_{M_1}$, and thus $M_1 \not\sqsubseteq M_2$, making M_1 and M_2 incomparable. It can be verified that both M_1 and M_2 are minimal w.r.t. $\sqsubseteq$, making them preferred extended answer sets.

Example 8
Consider the ordered program $\langle P, < \rangle$ where P is as in Example 2 and $<$ is as shown below.

$$\begin{array}{c} \neg a \leftarrow \\ \neg b \leftarrow \\ \hline a \leftarrow \neg b \\ b \leftarrow \neg a \end{array}$$

The reducts of the extended answer sets of P are $P_{M_1} = P \backslash \{\neg b \leftarrow\ \}$, $P_{M_2} = P \backslash \{\neg a \leftarrow\ \}$, and $P_{M_3} = P \backslash \{a \leftarrow \neg b, b \leftarrow \neg a\}$ which are ordered by $P_{M_1} \sqsubset P_{M_3}$ and $P_{M_2} \sqsubset P_{M_3}$. Thus $\langle P, < \rangle$ has two (proper) preferred answer sets: $M_1 = \{\neg a, b\}$ and $M_2 = \{a, \neg b\}$.

Note that, in the above example, the preferred answer sets correspond to the stable models (answer sets) of the logic program $\{a \leftarrow \textit{not}\ b, b \leftarrow \textit{not}\ a\}$, i.e. the stronger rules of $\langle P, < \rangle$ where negation as failure (*not*) replaces classical negation ($\neg$). In fact, the ordering of P, which makes the rules $\neg a \leftarrow$ and $\neg b \leftarrow$ less preferred, causes $\neg$ to behave as negation as failure, under the preferred answer set semantics.

In general, we can easily simulate negation as failure using classical negation and a trivial ordering.

Theorem 9
Let P be an (non-disjunctive) seminegative logic program The ordered version of P, denoted $N(P)$ is defined by $N(P) = \langle P' \cup P_\neg, < \rangle$ with $P_\neg = \{\neg a \leftarrow\ \mid a \in \mathcal{B}_P\}$ and P' is obtained from P by replacing each negated literal *not* p by $\neg p$. The order is defined by $P' < P_\neg$, i.e. $\forall r \in P', r' \in P_\neg \cdot r < r'$ (note that $P' \cap P_\neg = \emptyset$). Then M is a stable model of P iff $M \cup \neg(\mathcal{B}_P \backslash M)$ is a proper preferred answer set of $N(P)$.

Note that 2-level programs as above can also be used to support an extension of simple programs with “strict” rules. Such a program has the form $\langle P_s \cup P_d, < \rangle$ where P_s contains strict rules that may not be defeated and P_d contains "default" rules. The order $<$ is defined by $P_s < P_d$, i.e. $r_s < r_d$ for all $r_s \in P_s$, $r_d \in P_d$. The proper preferred answer sets then provide an intuitive semantics for such programs.

Interestingly, preference can also simulate disjunction.

Definition 9
Let P be a positive disjunctive logic program. The ordered version of P, denoted $D(P)$, is defined by $D(P) = \langle P_+ \cup P_- \cup P_p, < \rangle$ where $P_+ = \{ a \leftarrow \ \mid a \in \mathcal{B}_P \}$, $P_- = \{ \neg a \leftarrow \ \mid a \in \mathcal{B}_P \}$, $P_p = \{ a \leftarrow \beta \cup \neg(\alpha \backslash \{a\}) \mid (\alpha \leftarrow \beta) \in P \wedge a \in \alpha \}$, and $P_p < P_- < P_+$.

Intuitively, the rules from $P_+ \cup P_-$ guess a total interpretation I of P while the rules in P_p ensure that I^+ is a model of P. Minimality is assured by the fact that negations are preferred.

Example 9
Consider the disjunctive program $P = \{ a \vee b \leftarrow \ , \ a \leftarrow b, \ b \leftarrow a \}$. This program illustrates that the shifted version[5] of a disjunctive program need not have the same models, see e.g. (Dix et al. 1996; De Vos and Vermeir 2001). The program $D(P)$ is represented below.

$$\begin{array}{cc}
a \leftarrow & b \leftarrow \\
\hline
\neg a \leftarrow & \neg b \leftarrow \\
\hline
b \leftarrow \neg a & a \leftarrow \neg b \\
a \leftarrow b & b \leftarrow a
\end{array}$$

$D(P)$ has a single proper preferred answer set $\{a, b\}$ which is also the unique minimal model of P, while the shifted version yields no models at all. Note that both $\neg a \leftarrow$ and $\neg b \leftarrow$ are defeated because minimization is overridden by satisfaction of more preferred non-disjunctive rules.

Theorem 10
Let P be a positive disjunctive logic program. M is a minimal model of P iff $M' = M \cup \neg(\mathcal{B}_P \setminus M)$ is a proper preferred answer set of $D(P)$.

In view of Theorem 9 and Theorem 10, it is natural to try to simulate programs that combine negation as failure and disjunction.

Definition 10
Let P be a seminegative disjunctive logic program. The ordered version of P, denoted $D_n(P)$, is defined by $D_n(P) = \langle P_c \cup P_- \cup P_p, < \rangle$ where $P_c = \{ a \leftarrow \beta' \ \mid (\alpha \leftarrow \beta) \in P \wedge a \in \alpha \}$, $P_- = \{ \neg a \leftarrow \ \mid a \in \mathcal{B}_P \}$, $P_p = \{ a \leftarrow \beta' \cup \neg(\alpha \backslash \{a\}) \mid (\alpha \leftarrow \beta) \in P \wedge a \in \alpha \}$, and $P_p < P_- < P_c$. Here, β' is obtained from β by replacing all occurrences of *not* $a \in \beta$ by $\neg a \in \beta'$.

Intuitively, the rules in P_c apply disjunctive rules by choosing a literal from the head of the original rule; rules in P_p ensure that any proper answer set is a model and the preference $P_- < P_c$ supports minimization.

[5] The shifted version of a disjunctive program is a seminegative program where each disjunctive rule $\alpha \leftarrow \beta$ is replaced by the set of rules containing $a \leftarrow \beta \cup$ *not* $(\alpha \backslash \{a\})$ for each $a \in \alpha$.

Theorem 11

Let P be a seminegative disjunctive logic program. If M is an answer set of P then $M \cup \neg(\mathcal{B}_P \setminus M)$ is a proper preferred answer set of $D_n(P)$.

Proof

The theorem immediately follows from Theorem 12 and Proposition 3.3 in (Sakama and Inoue 1994).
$\square$

Unfortunately, $D_n(P)$ may have too many proper preferred answer sets, as illustrated by the following example.

Example 10

Consider the seminegative disjunctive program $P = \{a \vee b \leftarrow\ , b \leftarrow a, a \leftarrow \textit{not}\ a\}$. This program does not have an answer set. Indeed, any answer set M would need to contain a and thus, by the rule $b \leftarrow a$, also b, thus $M = \{a, b\}$. But the reduct $P^M = \{a \vee b \leftarrow\ ,\ b \leftarrow a\}$ has only one minimal answer set $\{b\} \neq M$.

However, $\{a, b\}$ is the unique minimal preferred answer set of $D_n(P)$ which is shown below.

$$\begin{array}{cc}
a \leftarrow & b \leftarrow \\
\hline
\neg a \leftarrow & \neg b \leftarrow \\
\hline
b \leftarrow a & a \leftarrow \neg a \\
a \leftarrow \neg b & b \leftarrow \neg a
\end{array}$$

In fact, the preferred answer sets semantics of $D_n(P)$ corresponds to the possible model semantics of (Sakama and Inoue 1994).

Definition 11

For a seminegative disjunctive logic program P, we define a **split program** as the (non-disjunctive) seminegative program obtained from P by replacing each rule $\alpha \leftarrow \beta \in P$ with $a \leftarrow \beta$ for every $a \in S$, where S is some non-empty subset of α. Now, a **possible model** of P is any answer set of any split program of P.

Theorem 12

Let P be a seminegative disjunctive logic program. An interpretation M is a proper preferred answer set of $D_n(P)$ iff M^+ is a minimal possible model of P.

In the next subsection, we'll see that, nevertheless, the expressiveness of the preferred answer set semantics of OLP is similar to that of seminegative disjunctive programs.

3.2 Computing Preferred Answer Sets

In this subsection, we only consider finite programs (corresponding to datalog-like rules).

Definition 12
Let $\langle P, < \rangle$ be a partially ordered set (where $<$ is strict). The **downward closure** of a subset $X \subseteq P$ is defined by $down(X) = \{u \in P \mid \exists x \in X \cdot u < x\}$. A set $X \subseteq P$ is **downward closed** iff $down(X) \subseteq X$.

A *specification* for an ordered program poses restrictions on the sets of rules that should be satisfied, respectively defeated, by a conforming extended answer set.

Definition 13
Let $\langle P, < \rangle$ be an ordered program. A **specification** for P is pair $\langle R_i, R_o \rangle$ of disjoint subsets of P such that $R_i \cup R_o$ is downward closed.

A specification $\langle R'_i, R'_o \rangle$ **extends** another specification $\langle R_i, R_o \rangle$, denoted $\langle R_i, R_o \rangle \preceq \langle R'_i, R'_o \rangle$, iff $R_i \subseteq R'_i$ and $R_o \subseteq R'_o$.

A set of rules $R \subseteq P$ **satisfies** a specification $\langle R_i, R_o \rangle$, denoted $R \models \langle R_i, R_o \rangle$, iff $R^\star$ is an extended answer set for P, $R_i \subseteq R$, $R_o \cap R = \emptyset$ and, moreover, $\forall r \in R_o \cdot R^* \not\models r$.

Obviously, if R satisfies $\langle R_i, R_o \rangle$ then R satisfies any weaker specification $\langle R'_i, R'_o \rangle \preceq \langle R_i, R_o \rangle$.

In the remainder of this section, we will use the term "extended answer set" for both the interpretation I and the corresponding set of rules $\{r \in P \mid I \models r\}$ that it satisfies.

To force a conforming extended answer set to satisfy at least one out of a collection of sets of rules, we define a constraint. Such constraints will be used to ensure that conforming extended answer sets are not smaller (w.r.t. $\sqsubset$) than others.

Definition 14
Let $\langle P, < \rangle$ be an ordered program. A **constraint** is a set of sets of rules $C \subseteq 2^P$. A specification $\langle R_i, R_o \rangle$ is consistent with a constraint iff $\exists c \in C \cdot R_o \cap c = \emptyset$. A rule set R satisfies a constraint C, denoted $R \models C$, iff $\exists c \in C \cdot c \subseteq R$.

The **expansion** of a specification and a constraint, denoted $\mu(\langle R_i, R_o \rangle, C)$, is defined by

$$\mu(\langle R_i, R_o \rangle, C) = \{R \subseteq P \mid R \models \langle R_i, R_o \rangle \wedge R \models C\} \ .$$

We use $\min \mu(\langle R_i, R_o \rangle, C)$ to denote the minimal elements of $\mu(\langle R_i, R_o \rangle, C)$ w.r.t. the $\sqsubset$-order.

By definition, $\min \mu(\langle R_i, R_o \rangle, C)$ contains minimal (according to $\sqsubset$) answer sets that satisfy both the specification $\langle R_i, R_o \rangle$ and the constraint C.

Definition 15
Let $\langle P, < \rangle$ be an ordered program and $R \subseteq P$ and $T \subseteq P$ be sets of rules. A **witness** of R against T is any rule $r \in R \backslash T$ such that $\forall t \in T \backslash R \cdot t \not< r$.

We use $\omega(T)$ to denote the set $\{\{r\} \cup (down(\{r\}) \cap T) \mid r \in P \setminus T\}$.

In Lemma 4 from the Appendix, it is shown that $T \not\sqsubset R$ iff R has a witness against T, which is itself equivalent to $\exists X \in \omega(T) \cdot X \subseteq R$.

The basic algorithm to compute preferred answer sets is shown in Figure 1. Intuitively, $aset(\langle R_i, R_o \rangle, C)$ returns the minimal elements among the extended answer sets from P that satisfy both the specification $\langle R_i, R_o \rangle$ and the constraint C.

This is achieved as follows:

- If $\langle R_i, R_o\rangle$ is inconsistent with C then, obviously, there are no elements satisfying both.
- If $R_i \cup R_o = P$, R_i should be returned, if it is an extended answer set.
- Otherwise, we first compute the set M of minimal extended answer sets containing a minimal rule r from $P\backslash(R_i \cup R_o)$, using the call $aset(\langle R_i \cup \{r\}, R_o\rangle, C)$.
- Next, we compute the minimal extended answer sets not containing r. Such answer sets must contain a witness against each $m \in M$. This is ensured by appropriately extending the constraint C to C'. The missing solutions are then computed using $aset(\langle R_i, R_o \cup \{r\}\rangle, C')$.

Formally, we will have that $aset(\langle R_i, R_o\rangle, C) = \min \mu(\langle R_i, R_o\rangle, C)$ from which, since the preferred answer sets of P obviously correspond to $\min \mu(\langle \emptyset, \emptyset\rangle, \{\emptyset\})$, it follows that $aset(\langle \emptyset, \emptyset\rangle, \{\emptyset\})$ returns exactly the preferred answer sets of P.

```
set<RuleSet>
aset (⟨$R_i, R_o$⟩, C) {
// precondition: $C \neq \emptyset$

if ( ⟨$R_i, R_o$⟩ and $C$ are inconsistent )
  return $\emptyset$

if (($R_i \cup R_o$) = $P$ )
  if ( $R_i^\star$ is an extended answer set  of  $P$)
    return $\{R_i\}$
  else
    return $\emptyset$

choose $r$ minimal in $P\backslash(R_i \cup R_o)$

// compute preferred answer sets containing $r$
$M$ = $aset(\langle R_i \cup \{r\}, R_o\rangle, C)$;

// add constraints that guarantee that each element in
// $\mu(\langle R_i, R_o \cup \{r\}\rangle, C')$ has a witness against any $m \in M$.
$C' = C$
for each $m \in M$
  $C'$ = $\{c \cup x \mid c \in C' \wedge x \in \omega(m) \wedge r \notin x\}$
if ($C' = \emptyset$) // $\exists m \in M \cdot \forall x \in \omega(m) \cdot r \in x$
  return $M$
// compute preferred answer sets not containing $r$
return $M \cup aset(\langle R_i, R_o \cup \{r\}\rangle, C')$;
}
```

Fig. 1: Basic Algorithm

Theorem 13
Let $\langle P, <\rangle$ be an ordered program, $\langle R_i, R_o\rangle$ be a specification and C a constraint. Then $aset(\langle R_i, R_o\rangle, C) = \min \mu(\langle R_i, R_o\rangle, C)$.

Since a preferred answer set is minimal w.r.t. the extended answer sets that satisfy the "empty" specification and constraint, we obtain the following corollary.

Corollary 1
Let $\langle P, < \rangle$ be an ordered program. The preferred answer sets of P are computed by $aset(\langle \emptyset, \emptyset \rangle, \{\emptyset\})$.

Clearly, the algorithm of Figure 1 can be further optimized, e.g. by proactively computing certain conditions, such as the consistency of R_i, etc.

A first implementation of an ordered logic program solver (OLPS) is available under the GPL at `http://tinf2.vub.ac.be/olp/`. After grounding, OLPS computes (a selection of) the proper preferred answer sets of a finite ordered program which is described using a sequence of module definitions and order assertions. A module is specified using a module name followed by a set of rules, enclosed in braces while an order assertion is of the form $m_0 < m_1 < \ldots < m_n$, $n > 0$, where each m_i, $0 \leq i \leq n$ is a module name. Such an assertion expresses that each rule in m_i, $0 \leq i \leq n$ is more preferred than any rule in m_{i+1}. Figure 2 shows the OLPS version of the the diagnostic problem from Example 1.

```
FaultModel {
        −power.
        −bulb.
}
NormalOperation {
        power.
        bulb.
}
System {
        light :− power, bulb.
}
System < NormalOperation < FaultModel
Observations { − light :− light . }
```

Fig. 2: The OLPS version of Example 1.

The following results shed some light on the complexity of the preferred answer set semantics.

First we note that checking whether M is not a preferred answer set of an OLP P is in NP because

1. Checking that M is an extended answer set of P, i.e. verifying foundedness and verify that each non-satisfied rule is defeated, can be done in deterministic polynomial time. (E.g. foundedness can be verified using a marking algorithm that repeatedly scans all rules in P, marking elements of M that have a "motivation" based on already marked elements from M).
2. Guess a set $N \sqsubseteq M$, which can be done in polynomial time, and verify that N is an extended answer set.

Finding a preferred answer set M can then be performed by an NP algorithm that guesses M and uses an NP oracle to verify that it is not the case that M is not a preferred answer set. Hence the following theorem.

Theorem 14
The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in any preferred answer set of P is in Σ_2^P.

Theorem 15
The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in every preferred answer set of P is in Π_2^P.

Proof
Finding a preferred answer set M such that $a \notin M$ is in Σ_2^P due to Theorem 14. Thus, the complement is in Π_2^P. $\square$

Theorem 16
The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in any preferred answer set of P is Σ_2^P-hard.

Proof
The proof uses a reduction of the known Σ_2^P-hard problem of deciding whether a quantified boolean formula $\phi = \exists x_1, \ldots, x_n \cdot \forall y_1, \ldots, y_m \cdot F$ is valid, where we may assume that $F = \vee_{c \in C} c$ with each c a conjunction of literals over $X \cup Y$ with $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ ($n, m > 0$). The construction is inspired by a similar result in (Eiter and Gottlob 1993) for disjunctive logic programs.

The program P corresponding to ϕ is shown below using a straightforward extension of the graphical representation of Definition 5: the order in P is defined by $P_4 < P_3 < P_2$ (note that the rules in P_1 are not related to any other rules).

$$
\begin{array}{c|c}
 & P_2 = \{ y \leftarrow \ \ \neg y \leftarrow \ \mid y \in Y \} \\
\cline{2-2}
P_1 = \{ x \leftarrow \ \ \neg x \leftarrow \ \mid x \in X \} & P_3 = \neg sat \leftarrow sat \\
\cline{2-2}
 & P_4 = \{ sat \leftarrow c \mid c \in C \}
\end{array}
$$

Obviously, the construction of P can be done in polynomial time. Intuitively, the rules in P_1 and P_2 are used to guess a truth assignment for $X \cup Y$.

In the sequel, we will abuse notation by using x_M and y_M where M is an answer set for P, to denote subsets of M, e.g. $x_M = X \cap M$ and in expressions such as $F(x_M, y_M)$ which stands for $F(x_1, \ldots, x_n, y_1, \ldots y_m)$ with $x_i = \mathbf{true}$ iff $x_i \in x_M$ and, similarly, $y_j = \mathbf{true}$ iff $y_j \in y_M$. We will also sometimes abbreviate the arguments of F, writing e.g. $F(x, y)$ rather than $F(x_1, \ldots, x_n, y_1, \ldots y_m)$.

The following properties of P are straightforward to show:

1. If we have an extended answer set M containing $sat \in M$, then $F(x_M, y_M)$ must hold.
2. Any extended answer set satisfies all the rules in P_4.
3. For extended answer sets M_1 and M_2, with $M_1 \cap X \neq M_2 \cap X$, neither $M_1 \sqsubset M_2$ nor $M_2 \sqsubset M_1$ holds, as the rules in P_1 are unrelated to any other rules.
4. If $M_1 \sqsubset M_2$ for some extended answer sets M_1 and M_2, then $M_1 \cap X = M_2 \cap X$ and, moreover, $sat \in M_2 \setminus M_1$, i.e. $(\neg sat \leftarrow sat) \in P_{M_1} \setminus P_{M_2}$.

We show that ϕ is valid iff $sat \in M$ for some preferred answer set M of P.

To show the "if" part, assume that M is a preferred answer set with $sat \in M$. By (1) we have that $F(x_M, y_M)$ holds. To prove that ϕ is valid it remains to show that $\forall y \cdot F(x_M, y)$. Suppose that, on the contrary, $\exists y \cdot \neg F(x_M, y)$ and consider the extended answer set $M' = (M \cap (X \cup \neg X)) \cup y \cup \neg(Y \setminus y)$. It is easy to verify that $M' \sqsubset M$ since the rules in P_1 satisfied by M are the same as the rules in P_1 satisfied by M'; and all the rules in P_4 are satisfied by both M and M' (due to (2)); and the rule in P_3 is defeated by M and satisfied by M'. But $M' \sqsubset M$ contradicts the fact that M is a preferred answer set. Thus, ϕ is valid.

To show the reverse, assume that ϕ is valid, i.e. there exists some $x_M \subseteq X$ such that $\forall y \cdot F(x_M, y)$. Consider $M = x_M \cup \neg(X \setminus x_M) \cup y \cup \neg(Y \setminus y) \cup \{sat\}$ where $y \subseteq Y$ is arbitrary. Clearly, M is an extended answer set. To show that M is preferred, assume that, on the contrary, $M' \sqsubset M$ for some extended answer set M'. By (4), $M \cap X = M' \cap X$ and $sat \notin M'$. These imply that $\neg F(x_M, y_{M'})$, contradicting that $\forall y \cdot F(x_M, y)$. $\square$

Theorem 17
The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in every preferred answer set of P is Π_2^P-hard.

Proof
Reconsider the program P in the proof of Theorem 16. Let a be a fresh atom not occurring in P and define P' as P with two extra rules $a \leftarrow$ and $\neg a \leftarrow$ in the component P_2. Clearly, showing that a does not occur in every preferred answer set is the same as showing that $\neg a$ occurs in any preferred answer set of P. Deciding the latter is Σ_2^P-hard by Theorem 16; thus deciding the complement of the former is Π_2^P-hard.

In the appendix an alternative proof is provided using quantified boolean formulas. $\square$

The following is immediate from Theorem 14, Theorem 15, and Theorem 16.

Corollary 2
The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in any proper preferred answer set of P is Σ_2^P-complete. The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in every proper preferred answer set of P is Π_2^P-complete.

3.3 Adding Negation as Failure

In view of the results from Section 3.1 and Section 3.2, it is natural to wonder whether adding negation as failure to ordered programs leads to a more expressive formalism.

To study this question, we first extend simple logic programs to allow negation as failure in both the head and the body of rules. The definition closely mirrors Definition 2: we only generalize the notion of defeat to take into account the possible presence of negation as failure in the head of a rule.

Definition 16
An **extended logic program** (ELP) is a countable set P of **extended rules** of the form $\alpha \leftarrow \beta$ where $\alpha \cup \beta$ is a finite set of extended literals, and $|\alpha| \leq 1$, i.e. α is a singleton or empty.

An extended rule $r = a \leftarrow \beta$ is **defeated** w.r.t. P and I iff P contains an applied **competing rule** $r' = a' \leftarrow \beta'$ such that $\{a, a'\}$ is inconsistent.

An interpretation I is an **extended answer set** of P iff I is an answer set (see Section 2.1) of P_I and each unsatisfied rule from $P \setminus P_I$ is defeated w.r.t. I.

Example 11
Consider the extended program P containing the following rules.

$$\begin{array}{ccc} \neg a \leftarrow & \neg b \leftarrow & c \leftarrow \\ a \leftarrow \textit{not}\, b & b \leftarrow \textit{not}\, a & \textit{not}\, c \leftarrow a \end{array}$$

For the interpretation $I = \{a, \neg b\}$, P_I contains all rules but $\neg a \leftarrow$ and $c \leftarrow$ which are defeated (w.r.t. I) by the applied rules $a \leftarrow \textit{not}\, b$ and $\textit{not}\, c \leftarrow a$, respectively. I is then an extended answer set because $\{a, \neg b\}$ is an answer set of $(P_I)^I = \{\neg b \leftarrow\ ,\ a \leftarrow\ \}$.

P has three more extended answer sets, namely $J = \{\neg a, b, c\}$, $K = \{\neg a, \neg b, c\}$ and $L = \{a, \neg b, c\}$. Here, $P_J = P \setminus \{\neg b \leftarrow\ \}$, and $(P_J)^J$ contains $\neg a \leftarrow$, $b \leftarrow$, $c \leftarrow$ and $\leftarrow a$. For K, we have that $P_K = P \setminus \{a \leftarrow \textit{not}\, b,\ b \leftarrow \textit{not}\, a\}$ and $(P_K)^K$ contains $\neg a \leftarrow$, $\neg b \leftarrow$, $c \leftarrow$ and $\leftarrow a$. Finally, L yields that $P_L = P \setminus \{\neg a \leftarrow\ , \textit{not}\, c \leftarrow a\}$ and $(P_L)^L$ contains $a \leftarrow$, $\neg b \leftarrow$ and $c \leftarrow$.

Unlike for simple logic programs, extended answer sets for extended logic programs are not necessary minimal w.r.t. subset inclusion, as demonstrated by the previous example where $I \subset L$. The same holds for extended disjunctive logic programs as shown in (Inoue and Sakama 1994).

Furthermore, the extended answer set semantics for extended logic programs is not universal, even for programs without constraints, as witnessed by the following example.

Example 12
Consider the extended logic program P containing the rules $a \leftarrow \textit{not}\, b$ and $b \leftarrow a, \textit{not}\, c$. Clearly, no extended answer set can contain c or $\neg c$.

For $I = \emptyset$ we obtain $(P_I)^I = \{b \leftarrow a, \textit{not}\, c\}^I = \{b \leftarrow a\}$, which has a unique answer set $\emptyset = I$. However, $a \leftarrow \textit{not}\, b$ is neither satisfied nor defeated in I. For $I = \{a\}$, $(P_I)^I = \{a \leftarrow \textit{not}\, b\}^I = \{a \leftarrow\ \}$ which has a unique answer set $\{a\} = I$. However, $b \leftarrow a, \textit{not}\, c$ is neither satisfied nor defeated in I. For $I = \{b\}$, $(P_I)^I = P^I = \{b \leftarrow a\}$ which has a unique answer set $\emptyset \neq I$. Finally, for $I = \{a\}$ we obtain $(P_I)^I = P^I = \{b \leftarrow a\}$ which has $\emptyset \neq I$ as a unique answer set.

Thus, P has no extended answer sets.

Obviously, any traditional answer set of an ELP P is also an extended answer set. However, unlike for simple programs, a consistent ELP, i.e. a program that has answer sets, may also have additional extended answer sets, as in the following example.

Example 13
Consider the following program P.

$$\begin{array}{cc} \neg b \leftarrow a & b \leftarrow \textit{not}\, b \\ a \leftarrow \textit{not}\, b & b \leftarrow \textit{not}\, a \end{array}$$

Clearly, $I = \{b\}$ is an answer set with P^I containing $\neg b \leftarrow a$ and $b \leftarrow$ which has $\{b\}$ as a minimal answer set. Since $P_I = P$, $\{b\}$ is also an extended answer set.

However, also $J = \{a, \neg b\}$ is an extended answer set because:

- P_J contains all rules but $b \leftarrow$ *not* b. The latter rule is defeated (w.r.t. J) by the applied rule $\neg b \leftarrow a$.
- $(P_J)^J$ contains just $a \leftarrow$ and $\neg b \leftarrow a$ and thus J is an answer set of P_J.

Clearly, though, J is not an answer set of P.

Note that the program in the above example does not contain negation as failure in the head of a rule. In fact, negation as failure in the heads of rules can be removed by a construction that is similar to the one used in (Inoue and Sakama 1998) for reducing DLP's with negation as failure in the head to DLP's without.

Definition 17
For P an ELP, define $E(P)$ as the ELP, without negation as failure in the head, obtained from P by replacing each rule $a \leftarrow \beta$ by (for a an ordinary literal, *not*$_a$ is a new atom) by $a \leftarrow \beta,$ *not* $\neg a,$ *not not*$_a$ when a is an ordinary literal; or by *not*$_{\hat{a}} \leftarrow \beta,$ *not* $\hat{a}$ when a is a naf-literal.

Intuitively, one can ignore an applicable rule $a \leftarrow \beta$ if it is defeated by evidence for either $\neg a$ or *not* a, thus making either *not* $\neg a$ or *not not*$_a$ false and the rule $a \leftarrow \beta,$ *not* $\neg a,$ *not not*$_a$ not applicable.

The extended answer sets of P can then be retrieved from the traditional answer sets of $E(P)$.

Theorem 18
Let P be an ELP. Then, S is an extended answer set of P iff there is an answer set S' of $E(P)$ such that $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$.

For example, let $P = \{a \leftarrow\ ,$ *not* $a \leftarrow\ \}$, which has two extended answer sets $\{a\}$ and $\emptyset$. Then $E(P) = \{a \leftarrow$ *not* $\neg a,$ *not not*$_a,$ *not*$_a \leftarrow$ *not* $a\}$ which has two traditional answer sets $\{a\}$ and $\{$*not*$_a\}$ corresponding with $\{a\}$ and $\emptyset$.

The definitions from Section 3.1 can be reused to define extended ordered logic programs and their preferred answer set semantics. However, unlike simple programs, an extended program R can have extended answer sets $M_1 \neq M_2$ while $R_{M_1} = R_{M_2}$. E.g., the program $\{a \leftarrow$ *not* $b\ ,\ b \leftarrow$ *not* $a\}$ has two (extended) answer sets $\{a\}$ and $\{b\}$ that both satisfy all the rules. Intuitively, M_1 should be incomparable with M_2, hence the extra condition $R_{M_1} \neq R_{M_2}$ in the definition of $\sqsubseteq$ between extended answer sets.

Definition 18
An **extended ordered logic program** (EOLP) is a pair $\langle R, < \rangle$ where R is an extended program and $<$ is a well-founded strict partial order on the rules in R[6].

The partial order $\sqsubseteq$ between subsets of R is defined as in Definition 6. For M_1, M_2 extended answer sets of R, we define $M_1 \sqsubseteq M_2$ iff $R_{M_1} \neq R_{M_2}$ and $R_{M_1} \sqsubseteq R_{M_2}$. As usual, $M_1 \sqsubset M_2$ iff $M_1 \sqsubseteq M_2$ and $M_1 \neq M_2$.

An **answer set** for an EOLP P is any extended answer set of R. An answer set for P is called **preferred** if it is minimal w.r.t. $\sqsubseteq$.

[6] Strictly speaking, we should allow R to be a multiset or, equivalently, have labeled rules, so that the same rule can appear in several positions in the order. For the sake of simplicity of notation, we will ignore this issue: all results also hold for the general multiset case.

Example 14
Reconsider the program from Example 11 with the following preference relation, yielding an extended ordered program $\langle P, < \rangle$.

$$\frac{\neg a \leftarrow \qquad \neg b \leftarrow \qquad \textit{not}\ c \leftarrow a}{a \leftarrow \textit{not}\ b \qquad b \leftarrow \textit{not}\ a \qquad c \leftarrow}$$

The reducts of the answer sets of P are $P_I = P \setminus \{c \leftarrow\ ,\ \neg a \leftarrow\ \}$, $P_J = P \setminus \{\neg b \leftarrow\ \}$, $P_K = P \setminus \{a \leftarrow \textit{not}\ b,\ b \leftarrow \textit{not}\ a\}$ and $P_J = P \setminus \{\neg a \leftarrow\ , \textit{not}\ c \leftarrow a\}$, which are ordered by $P_J \sqsubseteq P_I$, $P_J \sqsubseteq P_K$, $P_L \sqsubseteq P_I$ and $P_L \sqsubseteq P_K$, making both $J = \{\neg a, b, c\}$ and $L = \{a, \neg b, c\}$ preferred over both $I = \{a, \neg b\}$ and $K = \{\neg a, \neg b, c\}$.

An interesting interaction between defeat and negation as failure can occur when default (minimally preferred) rules of the form $\textit{not}\ a \leftarrow$ are used. At first sight, such rules are useless because *not* a is true by default. However, if present, such rules can also be used to defeat others as in the following example.

Example 15
Consider the following EOLP.

$$\frac{\dfrac{\textit{not}\ a \leftarrow}{a \leftarrow}}{\leftarrow a}$$

This program has the empty set as its single preferred answer set, its reduct containing the rules $\textit{not}\ a \leftarrow$ and $\leftarrow a$. Without $\textit{not}\ a \leftarrow$, it would be impossible to defeat $a \leftarrow$, thus violating $\leftarrow a$ and thus the program would not have any answer sets.

Extended (unordered) programs can be regarded as EOLP's with an empty order relation.

Theorem 19
For an ELP P, the extended answer sets of P coincide with the preferred answer sets of the EOLP $\langle P, \emptyset \rangle$.

Proof
Trivial. If the order relation is empty, there are no rules to counter defeated rules, so every extended answer set is also preferred. □

Interestingly, negation as failure can be simulated using order alone. However, from Theorem 19 and Example 11 (where $I \subset L$ are both preferred answer sets), it follows that preferred answer sets for EOLP's are not necessarily subset-minimal, which is not the case for the preferred answer sets of the ordered programs from Section 3.1. Hence, simulating an EOLP with an OLP will necessarily involve the introduction of fresh atoms.

One might be tempted to employ a construction similar to the one used in Definition 9 for simulating negation as failure using a two-level order. This would involve replacing extended literals of the form $\textit{not}\ a$ by fresh atoms $\textit{not}_a$ and adding "default" rules to introduce $\textit{not}_a$.

E.g. the extended program $P = \{a \leftarrow \mathit{not}\ b,\ \ b \leftarrow \mathit{not}\ a\}$ would be simulated by the ordered program $N(P)$

$$\begin{array}{cc}
\mathit{not}_a \leftarrow & \mathit{not}_b \leftarrow \\
\hline
a \leftarrow \mathit{not}_b & b \leftarrow \mathit{not}_a \\
\hline
\neg a \leftarrow \mathit{not}_a & \neg b \leftarrow \mathit{not}_b \\
\neg \mathit{not}_a \leftarrow a & \neg \mathit{not}_b \leftarrow b
\end{array}$$

where the rules on the lowest level act as constraints, forcing one of the "default rules" in the top level to be defeated in any proper answer set of $N(P)$. The "constraint rules" also serve to indirectly introduce competition between formally unrelated atoms: in the example, we need e.g. a rule to defeat $\mathit{not}_a \leftarrow$, based on the acceptance of a.

This does not work, however, since it may introduce unwanted answer sets as for the program $\{a \leftarrow \mathit{not}\ a\}$ which would yield the OLP program

$$\begin{array}{c}
\mathit{not}_a \leftarrow \\
\hline
a \leftarrow \mathit{not}_a \\
\hline
\neg a \leftarrow \mathit{not}_a \\
\neg \mathit{not}_a \leftarrow a
\end{array}$$

which has a (proper) preferred answer set $\{\mathit{not}_a, \neg a\}$ while the original program has no extended answer sets.

The above examples seem to point to contradictory requirements for the corresponding OLP programs: for the first example, rules implying not_a should be (indirect) competitors for a-rules while for the second example, the not_a-rule should *not* compete with the a-rule, in order not to introduce spurious answer sets.

The solution is to add not only fresh atoms for extended literals of the form *not* a, but also for ordinary literals. Thus each extended literal l will be mapped to an independent new atom $\phi(l)$. A rule $l \leftarrow \beta$ will then be translated to $\phi(l) \leftarrow \phi(\beta)$, which does not compete with any other such rule. Defeat between such rules is however supported indirectly by adding extra rules that encode the consequences of applying such a rule: for an original rule of the form $a \leftarrow \beta$, a an ordinary literal, we ensure that its replacement $\phi(a) \leftarrow \phi(\beta)$ can, when applied, indirectly defeat $\phi(\neg a)$- and $\phi(\mathit{not}\ a)$-rules by adding both $\neg\phi(\neg a) \leftarrow \phi(\beta), \phi(a)$ and $\neg\phi(\mathit{not}\ a) \leftarrow \phi(\beta), \phi(a)$. Similarly, for an original rule of the form *not* $a \leftarrow \beta$, a an ordinary literal, a rule $\neg\phi(a) \leftarrow \phi(\beta), \phi(\mathit{not}\ a)$ will be added along with its replacement $\phi(\mathit{not}\ a) \leftarrow \phi(\beta)$. Consistency is assured by introducing a new most preferred component containing, besides translated constraints $\leftarrow \phi(\beta)$ for the original ones, rules of the form $\leftarrow \phi(a), \phi(\mathit{not}\ a)$ and $\leftarrow \phi(a), \phi(\neg a)$. In addition, this new component also contains translations $a \leftarrow \phi(a)$ of the new atoms, that correspond to ordinary literals, back to their original versions.

Negation as failure can then be simulated by introducing "default" rules of the form $\phi(\mathit{not}\ a) \leftarrow$ in a new least preferred component.

Spurious answer sets are prevented, as these new default rules introducing $\phi(\mathit{not}\ a)$, which do not have defeat-enabling accompanying rules as described above, cannot be used to defeat transformed rules of the original program, but only to make them applicable. E.g. the program $\{a \leftarrow \textit{not}\ a\}$ mentioned above would be translated as

$$\begin{array}{rl}
\phi(\textit{not}\ a) \leftarrow & \qquad \phi(\textit{not}\ \neg a) \leftarrow \\
\hline
\phi(a) \leftarrow \phi(\textit{not}\ a) & \\
\neg\phi(\textit{not}\ a) \leftarrow \phi(\textit{not}\ a), \phi(a) & \\
\neg\phi(\neg a) \leftarrow \phi(\textit{not}\ a), \phi(a) & \\
\hline
\leftarrow \phi(a), \phi(\textit{not}\ a) & \qquad a \leftarrow \phi(a) \\
\leftarrow \phi(a), \phi(\neg a) & \qquad \neg a \leftarrow \phi(\neg a) \\
\leftarrow \phi(\neg a), \phi(\textit{not}\ \neg a) &
\end{array}$$

which has no proper preferred answer sets.

Formally, for an EOLP $\langle R, < \rangle$, we define a mapping ϕ translating original extended literals by: $\phi(a) = a'$, $\phi(\neg a) = a'_{\neg}$, $\phi(\textit{not}\ a) = \textit{not}_a$ and $\phi(\textit{not}\ \neg a) = \textit{not}_{\neg a}$; where for each atom $a \in \mathcal{B}_R$, $a', a_{\neg}', \textit{not}_a$ and $\textit{not}_{\neg a}$ are fresh atoms. We use $\phi(X)$, X a set of extended literals, to denote $\{\phi(x) \mid x \in X\}$.

Definition 19

Let $P = \langle R, < \rangle$ be an extended ordered logic program. The OLP version of P, denoted $N_s(P)$, is defined by $N_s(P) = \langle R_n \cup R' \cup R_c, R_c < R'_< < R_n \rangle$, where

- $R_n = \{\phi(\textit{not}\ a) \leftarrow \ \mid a \in \mathcal{B}_R \cup \neg\mathcal{B}_R\}$,
- R' is obtained from R by replacing each rule
  - $a \leftarrow \beta$, where a is a literal, by the rules $\phi(a) \leftarrow \phi(\beta)$ and $\neg\phi(\neg a) \leftarrow \phi(\beta), \phi(a)$ and $\neg\phi(\textit{not}\ a) \leftarrow \phi(\beta), \phi(a)$;
  - $\textit{not}\ a \leftarrow \beta$ by the rules $\phi(\textit{not}\ a) \leftarrow \phi(\beta)$ and $\neg\phi(a) \leftarrow \phi(\beta), \phi(\textit{not}\ a)$;
- $R_c = \{ \leftarrow \phi(\beta) \mid \ \leftarrow \beta \in R\} \cup \{ \leftarrow \phi(a), \phi(\textit{not}\ a); \ \leftarrow \phi(a), \phi(\neg a); \ a \leftarrow \phi(a) \mid a \in \mathcal{B}_R \cup \neg\mathcal{B}_R\}$.

Furthermore, $R'_<$ stands for the original order on R but defined on the corresponding rules in R'.

Note that $N_s(P)$ is free from negation as failure.

Example 16

The OLP $N_s(P)$, corresponding to the EOLP of Example 14 is shown below.

$$
\begin{array}{rrr}
\mathit{not}_a \leftarrow & \mathit{not}_b \leftarrow & \mathit{not}_c \leftarrow \\
\mathit{not}_{\neg a} \leftarrow & \mathit{not}_{\neg b} \leftarrow & \mathit{not}_{\neg c} \leftarrow \\
\hline
a'_{\neg} \leftarrow & b'_{\neg} \leftarrow & \mathit{not}_c \leftarrow a' \\
\neg a' \leftarrow a'_{\neg} & \neg b' \leftarrow b'_{\neg} & \neg c' \leftarrow a', \mathit{not}_c \\
\neg \mathit{not}_{\neg a} \leftarrow a'_{\neg} & \neg \mathit{not}_{\neg b} \leftarrow b'_{\neg} & \\
\hline
a' \leftarrow \mathit{not}_b & b' \leftarrow \mathit{not}_a & c' \leftarrow \\
\neg a'_{\neg} \leftarrow \mathit{not}_b, a' & \neg b'_{\neg} \leftarrow \mathit{not}_a, b' & \neg c'_{\neg} \leftarrow c' \\
\neg \mathit{not}_a \leftarrow \mathit{not}_b, a' & \neg \mathit{not}_b \leftarrow \mathit{not}_a, b' & \neg \mathit{not}_c \leftarrow c' \\
\hline
a \leftarrow a' & b \leftarrow b' & c \leftarrow c' \\
\neg a \leftarrow a'_{\neg} & \neg b \leftarrow b'_{\neg} & \neg c \leftarrow c'_{\neg} \\
\leftarrow a', \mathit{not}_a & \leftarrow b', \mathit{not}_b & \leftarrow c', \mathit{not}_c \\
\leftarrow a'_{\neg}, \mathit{not}_{\neg a} & \leftarrow b'_{\neg}, \mathit{not}_{\neg b} & \leftarrow c'_{\neg}, \mathit{not}_{\neg c} \\
\leftarrow a', a'_{\neg} & \leftarrow b', b'_{\neg} & \leftarrow c', c'_{\neg}
\end{array}
$$

The OLP has two proper preferred answer sets $J' = \{\neg a, b, c, a'_{\neg}, b', c', \neg a', \mathit{not}_a, \neg \mathit{not}_b, \neg b'_{\neg}, \neg \mathit{not}_c, \neg c'_{\neg}, \neg \mathit{not}_{\neg a}, \mathit{not}_{\neg b}, \mathit{not}_{\neg c}\}$ and $L' = \{a, \neg b, c, a', b'_{\neg}, c', \neg b', \neg \mathit{not}_a, \neg a'_{\neg}, \mathit{not}_b, \neg \mathit{not}_c, \neg c'_{\neg}, \mathit{not}_{\neg a}, \neg \mathit{not}_{\neg b}, \mathit{not}_{\neg c}\}$, corresponding to the preferred answer sets J and L of P.

In the above example, the preferred answer set of P can be recovered from the proper preferred answer set of $N_s(P)$ by selecting the literals from $\mathcal{B}_P \cup \neg \mathcal{B}_P$. The following theorem shows that this is a general property of $N_s(P)$.

Theorem 20
Let $P = \langle R, < \rangle$ be an extended ordered logic program. Then, M is a preferred answer set of P iff there exists a proper preferred answer set M' of $N_s(P)$, such that $M = M' \cap (\mathcal{B}_R \cup \neg \mathcal{B}_R)$.

Since the construction of $N_s(P)$ is polynomial, the above result, together with Theorems 9 and 11, suggests that order is at least as expressive as negation as failure, even if the latter is used in combination with the former.

4 Relationship to Other Approaches

4.1 Brewka's Preferred Answer Sets

Preferred answer sets have been introduced in the setting of extended logic programs. In (Brewka and Eiter 1999) a strict partial order on the rules in a program is used to prefer certain traditional answer sets above others. Intuitively, such preferred answer sets, which we call B-preferred answer sets in what follows to avoid confusion, are traditional answer sets that can be reconstructed by applying the rules in order of their priorities, i.e. starting with a most specific rule and ending with the least specific ones.

First, we note that the semantics defined in (Brewka and Eiter 1999) resides at the first level of the polynomial hierarchy, i.e. Σ_1^P, while the semantics from Section 3.1 is at the second level, i.e. Σ_2^P. The extra expressiveness of the latter semantics is useful for diagnostic and abductive reasoning applications: e.g. finding a subset minimal explanation is known to be Σ_2^P-complete (Eiter et al. 1997). In addition, we illustrate with some simple examples the differences between both approaches.

Using the ordered programs from Section 2 directly in the setting of the B-preferred answer set semantics is not very useful since the latter applies only to consistent programs that do have traditional answer sets. However, if we first apply the translation from Definition 4 to our programs, thus transforming the extended answer set semantics into the traditional one, we have a means to compare both approaches.

We start with a brief, formal description of B-preferred answer sets. The programs under consideration in (Brewka and Eiter 1999) are prioritized extended logic programs.

Definition 20

A **prioritized extended logic program** is a pair $P = (R, <)$, where R is an extended logic program (Definition 16) and $<$ is a strict partial order on the rules[7] in R.

The **prerequisites** of a rule $r = a \leftarrow \beta$ are the literals from $(\beta \backslash \mathit{not}\beta^-)$, i.e. the ordinary literals in its body. If β contains no ordinary literals, r is said to be **prerequisite-free**.

The B-preferred answer set semantics is defined on programs having a well-ordering[8] relation on the rules. Therefore another definition is needed to go from an ordinary prioritized program to one with a well-ordering.

Definition 21

A **full prioritization** of a prioritized program $P = (R, <)$ is any pair $P^* = (R, <^*)$ where $<^*$ is a well-ordering on R compatible with $<$, i.e. $r_1 < r_2$ implies $r_1 <^* r_2$, for all $r_1, r_2 \in R$. By $\mathcal{FP}(P)$ we denote the collection of all full prioritizations of P. We say that P is fully prioritized, if $\mathcal{FP}(P) = \{P\}$, i.e. P coincides with its unique full prioritization.

Like the traditional answer set semantics, B-preferred answer sets are defined in two steps. In the first step, B-preferred answer sets are defined for prerequisite-free programs, i.e. programs containing only prerequisite-free rules. A rule r is *blocked*[9] by a literal l iff $l \in B_r^-$. On the other hand, r is blocked by a set of literals X, iff X contains a literal that blocks r.

In the following construction, the rules in a full prioritization are applied in the order of their priorities.

Definition 22

Let $P = (R, <)$ be a full prioritization of a prerequisite-free prioritized program; let S be

[7] Again we only consider grounded programs, thus avoiding the complex definitions in (Brewka and Eiter 1999) dealing with the ground instantiations of prioritized rule bases.

[8] A (strict) partial order $<$ on S is called a (strict) total order iff $\forall x, y \in S \cdot x \neq y \Rightarrow x < y \ \lor \ y < x$. A (strict) total order $S, <$ is called a well-ordering iff each nonempty subset of S has a minimal element, i.e. $\forall X \subseteq S, X \neq \emptyset \cdot \exists x \in X \cdot \forall y \in X \cdot (x = y \lor x < y)$.

[9] We use the term "blocked" instead of the original "defeat" (Brewka and Eiter 1999) to avoid confusion with the notion of defeat from Definition 2.

a set of literals and let $(R, <) = \{r_\alpha\}_<$. We define the sequence $S_\alpha, 0 \leq \alpha < ord(<)$, of sets $S_\alpha \subseteq \mathcal{B}_R \cup \neg\mathcal{B}_R$ as follows:

$$S_\alpha = \begin{cases} \bigcup_{\beta<\alpha} S_\beta, & \textit{if } r_\alpha \textit{ is blocked by } \bigcup_{\beta<\alpha} S_\beta \textit{ or} \\ & H_{r_\alpha} \in S \textit{ and } r_\alpha \textit{ is blocked by } S, \\ \bigcup_{\beta<\alpha} S_\beta \cup \{H_{r_\alpha}\} & \textit{otherwise.} \end{cases}$$

The set $C_P(S)$ is the smallest set of ground literals, i.e. $C_P(S) \subseteq \mathcal{B}_R \cup \neg\mathcal{B}_R$, such that

1. $\bigcup_{\alpha<ord(<)} S_\alpha \subseteq C_P(S)$, and
2. $C_P(S)$ is logically closed.

The condition "$H_{r_\alpha} \in S$ and r_α is blocked by S" in the above definition is necessary to avoid situations in which a literal in S is derived by two rules r_1 and r_2 such that $r_1 < r_2$, but r_2 is applicable in S and r_1 is not. This would result in the conclusion H_{r_1} at a priority higher than effectively sanctioned by the rules, as in the following example.

Example 17
Consider the following program, taken from (Brewka and Eiter 1999).

$$\begin{array}{c}
r_4 \ : \ p \leftarrow \textit{not } \neg p \\
\hline
r_3 \ : \ \neg p \leftarrow \textit{not } p \\
\hline
r_2 \ : \ q \leftarrow \textit{not } \neg q \\
\hline
r_1 \ : \ p \leftarrow \textit{not } q
\end{array}$$

This program has two classical answer sets, i.e. $S_1 = \{p, q\}$ and $S_2 = \{\neg p, q\}$. Without the condition "$H_{r_\alpha} \in S$ and r_α is blocked by S" in Definition 22, the sequence for S_1 would be $\{p\}, \{p, q\}, \{p, q\}, \{p, q\}$ because r_1 would be applied, even if it is blocked w.r.t. the final set S_1. This in turn blocks r_3. On the other hand, using Definition 22 correctly, S_1 yields the sequence $\{\}, \{q\}, \{\neg p, q\}, \{\neg p, q\}$ where r_1 is not applied and r_3 becomes applicable. With Definition 23, this implies that S_1 is not B-preferred.

In general, C_P does not necessarily return the consequences of R, i.e. an applied rule r_α may later be blocked by some less preferred rule r_β where $\alpha < \beta$. However, if a normal answer set A of R is a fixpoint of C_P, then all preferences are taken into account, i.e. a rule whose head is not in A is blocked by a more preferred rule applied in A. Such answer sets will be preferred.

Definition 23
Let $P = (R, <)$ be a full prioritization of a prerequisite-free prioritized program; and let A be a normal answer set of R. Then A is a **B-preferred answer set** of P iff $C_P(A) = A$.

The B-preferred answer sets for programs with prerequisites are obtained using a reduction to prerequisite-free programs.

Definition 24
Let $P = (R, <)$ be a full prioritization of a prioritized program; and let $X \subseteq \mathcal{B}_R \cup \neg\mathcal{B}_R$. Then, ${}^XP = ({}^XR, {}^X<)$ is the fully prioritized program obtained from P, where ${}^XR$ is the set of rules obtained from R by

1. deleting every rule having a prerequisite l such that $l \notin X$, and
2. removing from each remaining rule all prerequisites,

and where ${}^{X}\!<$ is inherited from $<$ by the mapping $f : {}^{X}R \longrightarrow R$, i.e. $r'_1 \; {}^{X}\!<\! r'_2$ iff $f(r'_1) < f(r'_2)$, where $f(r')$ is the first rule in R w.r.t. $<$ such that r' results from r by step 2.

In the above reduction, a rule $a \leftarrow \beta$ is removed from the program w.r.t. a set of literals X iff the prerequisite part of the rule is not applicable w.r.t. X, i.e. $(\beta \setminus \mathit{not}\,\beta^{-}) \not\subseteq X$. The prerequisites of the remaining rules can then be safely removed.

Definition 25
A set of ground literals $A \subseteq \mathcal{B}_R \cup \neg\mathcal{B}_R$ is a B-preferred answer set of a full prioritization $P = (R, <)$ of a prioritized program, if A is a B-preferred answer set of ${}^{A}P$. A is a B-preferred answer set of a prioritized program Q, if A is a B-preferred answer set for some $P \in \mathcal{FP}(Q)$. We use $\mathcal{AS}_B(Q)$ to denote the set of all B-preferred answer sets of Q.

The following examples suggest that there is no clear relationship between the preferred (Section 3) and B-preferred (Brewka and Eiter 1999) semantics.

Example 18
Consider the OLP P on the left side below.

$$\begin{array}{cccccc} \neg a & \leftarrow & & \neg b & \leftarrow & \\ \hline a & \leftarrow & & b & \leftarrow & \\ \hline \neg b & \leftarrow & \neg a & \neg a & \leftarrow & \neg b \\ a & \leftarrow & b & b & \leftarrow & a \end{array} \qquad \begin{array}{cccccc} \neg a & \leftarrow & \mathit{not}\ a & \neg b & \leftarrow & \mathit{not}\ b \\ \hline a & \leftarrow & \mathit{not}\ \neg a & b & \leftarrow & \mathit{not}\ \neg b \\ \hline \neg b & \leftarrow & \neg a, \mathit{not}\ b & \neg a & \leftarrow & \neg b, \mathit{not}\ a \\ a & \leftarrow & b, \mathit{not}\ \neg a & b & \leftarrow & a, \mathit{not}\ \neg b \end{array}$$

This program has only one preferred answer set, i.e. $I = \{a, b\}$. On the other hand, the transformed program $E(P)$ (see Definition 4) with the same ordering among the rules as in P (see the program on the right side above), has two B-preferred answer sets, namely I and $J = \{\neg a, \neg b\}$.

While the above example illustrates that the B-preferred answer set semantics may yield too many answers w.r.t. the preferred answer set semantics, the next example shows that both approaches can yield different answers.

Example 19
Consider the OLP P on the left side below, and its consistent version $E(P)$, with the same order among the rules, on the right.

$$\begin{array}{ccc} \neg b & \leftarrow & \\ \hline b & \leftarrow & \\ \hline a & \leftarrow & b \\ \hline \neg a & \leftarrow & \end{array} \qquad \begin{array}{ccc} \neg b & \leftarrow & \mathit{not}\ b \\ \hline b & \leftarrow & \mathit{not}\ \neg b \\ \hline a & \leftarrow & b, \mathit{not}\ \neg a \\ \hline \neg a & \leftarrow & \mathit{not}\ a \end{array}$$

Let us interpret a as "we have light" and b as "the power is on". The fact that we do not have any light ($\neg a$) has the highest priority, followed by the rule that, if there is power, we should have light ($a \leftarrow b$). The weakest rules assert that probably the power is on (b), but on the other hand, it may not be ($\neg b$).

Clearly, the only preferred answer set $\{\neg a, \neg b\}$ of P fits well with our intuition: we have no light because somehow, the power is cut (defeating the $b \leftarrow$ rule). On the other hand, the single B-preferred answer set is $\{\neg a, b\}$ which questions the rule $a \leftarrow b$, although this rule is more preferred than $b \leftarrow$. This example illustrates some form of contrapositive reasoning: since it is preferable to satisfy $a \leftarrow b$, and $\neg a$ holds, any answer set that provides (a motivation for) $\neg b$ will be preferred over one that does not. It is this capability that is of interest in e.g. diagnostic applications (Van Nieuwenborgh and Vermeir 2003a; Van Nieuwenborgh and Vermeir 2003b).

In the literature two stricter version of the B-preferred answer set semantics have been considered: the D-preferred answer set semantics (Delgrande et al. 2000) and the W-preferred answer set semantics (Wang et al. 2000). A nice overview of and comparison between the B-, W- and D- preferred answer sets can be found in (Schaub and Wang 2001). It turns out (Theorem 8 in (Schaub and Wang 2001)) that $\mathcal{AS}_D(R,<) \subseteq \mathcal{AS}_W(R,<) \subseteq \mathcal{AS}_B(R,<) \subseteq \mathcal{AS}(R)$, i.e. B-preferredness is more strict than traditional answer sets, W-preferredness is more strict than B-preferredness and D-preferredness is the most strict preference relation. This is illustrated by Example 18 where neither I or J is W- or D-preferred. Nevertheless, in Example 19 the B-preferred answer set $\{\neg a, b\}$ is also W- and D-preferred, suggesting that there is no relationship between those approaches and the preferred answer set semantics from Section 3.

(Brewka and Eiter 1999) also introduces two principles, which we rephrase below using the present framework, that every system based on prioritized defeasible rules should satisfy.

Principle 1

Let M_1 and M_2 be two different extended answer sets of an ordered logic program $P = \langle R, < \rangle$, generated by the rules $R' \cup \{d_1\}$ and $R' \cup \{d_2\}$, where $d_1, d_2 \not\in R'$, respectively. If d_1 is preferred over d_2, then M_2 is not a (maximally) preferred answer set of P.

Principle 2

Let M be a preferred answer set of an ordered logic program $P = \langle R, < \rangle$ and let r be an inapplicable rule w.r.t. M. Then M is a preferred answer set of $\langle R \cup \{r\}, <' \rangle$ whenever $<'$ agrees with $<$ on the preference among the rules in R.

In this context, a rule r is said to be generating w.r.t. an extended answer set M, if it is applied w.r.t. M. It turns out that the preferred answer set semantics from Section 3 violates Principle 1, as illustrated by the following example.

Example 20

Let P be the OLP

$$\begin{array}{c} a \leftarrow \\ b \leftarrow \\ \hline \neg a \leftarrow \\ \hline \neg b \leftarrow \neg a \\ \hline b \leftarrow \neg b \end{array}$$

and consider the interpretations $I = \{a, b\}$ and $J = \{\neg a, b\}$. The generating rules for I are $\{a \leftarrow\ , b \leftarrow\ \}$, while those for J are $\{\neg a \leftarrow\ , b \leftarrow\ \}$. Furthermore, we have that $\neg a \leftarrow\ < a \leftarrow$. Since I is a preferred answer set of P, this violates Principle 1.

The problem with Principle 1 comes from the fact that it only considers "generating", i.e. applied, rules while the semantics of Section 3 also takes into account unapplied but satisfied rules, which have an equal influence on the shape of an answer set. E.g. in the above example, the rule $\neg b \leftarrow \neg a$ is violated by J but not by I, which should lead one to prefer I over J.

These considerations lead to a modified version of Principle 1 where "generating rules" is replaced by "reducts" (Definition 1) of the extended answer sets.

Principle 1' *Let M_1 and M_2 be two different extended answer sets of an ordered logic program $P = \langle R, < \rangle$, with reducts $R_{M_1} = R' \cup \{d_1\}$ and $R_{M_2} = R' \cup \{d_2\}$, where $d_1, d_2 \notin R'$, respectively. If d_1 is preferred over d_2, then M_2 is not a (maximally) preferred answer set of P.*

Clearly, the semantics of Section 3 obeys this principle (obviously, $R_{M_1} \sqsubset R_{M_2}$). The new principle is also vacuously satisfied for approaches, such as the B-preferred semantics, that start from consistent programs. However, if we apply e.g. the B-preferred semantics "indirectly" on inconsistent programs using the construction of Definition 4, it appears from Example 19 that this semantics does satisfy Principle 1'.

Obviously, the second principle holds for the preferred answer set semantics.

Theorem 21
The preferred answer set semantics satisfies Principle 2.

Proof
Trivial. Adding inapplicable rules w.r.t. a preferred answer set to an ordered logic program does not change anything to its preferredness. $\square$

4.2 Logic Programming with Ordered Disjunction

Logic programming with ordered disjunction (LPOD) (Brewka 2002; Brewka et al. 2002) combines qualitative choice logic (Brewka et al. 2002) with answer set programming by adding a new connective, called ordered disjunction, to logic programs. Using this connective, conclusions in the head of a rule are ordered according to preference. Intuitively, one tries to satisfy an applicable rule by using its most preferred, i.e. best ranked, conclusion.

Definition 26
A **logic program with ordered disjunction** (LPOD) is a set of rules of the form $a_1 \times \ldots \times a_n \leftarrow \beta$, $n \geq 1$, where the a_i's are (ordinary) literals and β is a finite set of extended literals.

An ordered disjunctive rule $a_1 \times \ldots \times a_n \leftarrow \beta$ can intuitively be read as: if β is true, then accept a_1, if possible; if not, then accept a_2, if possible; ...; if none of $a_1, \ldots, a_{n-1}$ are possible, then a_n must be accepted.

Similarly to ordered programs, the semantics for LPOD's is defined in two steps. First, answer sets for LPOD's are defined, which are then ordered by a relation that takes into account to what degree rules are satisfied. The minimal elements in this ordering are also called preferred answer sets. To avoid confusion, we will use the term "preferred LPOD answer sets" for the latter.

Answer sets for LPOD's are defined using *split programs*, a mechanism first used in (Sakama and Inoue 1994) to define the possible model semantics for disjunctive logic programs.

Definition 27
Let $r = a_1 \times \ldots \times a_n \leftarrow \beta$ be a LPOD rule. For $k \leq n$ we define the k^{th} option of r as $r^k = a_k \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{k-1}\}$.

An extended logic program P' is called a **split program** of a LPOD P if it is obtained by replacing each rule in P by one of its options. An interpretation S is then an (LPOD) **answer set** of P if it is an answer set of a split program P' of P.

Note that the split programs defined above do not contain negation as failure in the head of rules.

Example 21
The LPOD

$$\begin{aligned} b \times c \times d &\leftarrow \\ c \times a \times d &\leftarrow \\ \neg c &\leftarrow b \end{aligned}$$

has five answer sets, namely $S_1 = \{a, b, \neg c\}$, $S_2 = \{b, \neg c, d\}$, $S_3 = \{c\}$, $S_4 = \{a, d\}$ and $S_5 = \{d\}$.

Note that $S_5 \subset S_4$, illustrating that answer sets for LPOD programs need not be subset-minimal. Intuitively, only S_1 and S_3 are optimal in that they correspond with a best combination of options for each of the rules.

The above intuition is formalized in the following definition of a preference relation on LPOD answer sets.

Definition 28
Let S be an answer set of a LPOD P. Then S satisfies the rule $a_1 \times \ldots \times a_n \leftarrow \beta$

- to degree 1 if $S \not\models \beta$.
- to degree j $(1 \leq j \leq n)$ if $S \models \beta$ and $j = min\{i \mid a_i \in S\}$.

For a set of literals S, we define $S^i(P) = \{r \in P \mid deg_S(r) = i\}$, where $deg_S(r)$ is used to denote the degree to which r is satisfied w.r.t. S.

Let S_1 and S_2 be answer sets of P. Then S_1 is preferred over S_2, denoted $S_1 \sqsubset_b S_2$, iff there is a k such that $S_2^k(P) \subset S_1^k(P)$, and for all $j < k$, $S_1^j(P) = S_2^j(P)$. A minimal (according to $\sqsubset_b$) answer set is called a (LPOD) **preferred answer set** of P.

Example 22
In the LPOD from Example 21, both S_1 and S_3 satisfy the third rule to degree 1. In addition, S_1 satisfies the first rule to degree 1 and the second rule to degree 2, while S_3 satisfies the second rule to degree 1 and the first rule to degree 2. Thus S_1 and S_3 are incomparable w.r.t. $\sqsubset_b$. All other answer sets are less preferred than either S_1 or S_3: e.g. S_5 satisfies the first and second rule only to degree 3, and the third rule to degree 1, from which $S_1 \sqsubset_b S_5$ and $S_3 \sqsubset_b S_5$. It follows that S_1 and S_3 are both preferred.

Next, we show that the preference relation which is implicit in ordered disjunctive rules can be intuitively simulated using preference between non-disjunctive rules in one single ordered program. In (Brewka et al. 2002) an algorithm is presented to compute LPOD preferred answer sets using traditional answer set programming. The algorithm needs two different programs, i.e. a generator that computes answer sets and a tester for checking preferredness, which are executed in an interleaved fashion. On the other hand, using the following translation together with the OLPS solver (see Section 3.2) does the job in a single run.

Definition 29
The EOLP version of a LPOD P, denoted $L(P)$, is defined by $L(P) = \langle P_r \cup P_1 \cup \ldots \cup P_n \cup P_d, P_r < P_1 < \ldots < P_n < P_d \rangle$, where

- n is the size of the greatest ordered disjunction in P;
- P_r contains every non-disjunctive rule $a \leftarrow \beta \in P$. In addition, for every ordered disjunctive rule $r = a_1 \times \ldots \times a_n \leftarrow \beta \in P$, P_r contains a rule $a_i \leftarrow \beta, \textit{not}\, \{a_1, \ldots, a_n\} \setminus \{a_i\}$ for every $1 \leq i \leq n$, a rule $nap_r \leftarrow \textit{not}\, l$ for every literal $l \in \beta$ and a rule $nap_r \leftarrow l$ for every $\textit{not}\, l \in \beta$.
- P_d contains for every ordered disjunctive rule $r = a_1 \times \ldots \times a_n \leftarrow \beta \in P$ a rule $\textit{not}\, nap_r \leftarrow$ and rules $\textit{not}\, a_i \leftarrow \beta, \textit{not}\, \{a_1, \ldots, a_{i-1}\}$ for every $1 \leq i \leq n$.
- for $1 \leq k \leq n$, P_k is defined by $P_k = \{a_k \leftarrow \beta, \textit{not}\, \{a_1, \ldots, a_{k-1}\} \mid a_1 \times \ldots \times a_m \leftarrow \beta \in P \textit{ with } k \leq m \leq n\}$.

Intuitively, the rules in P_r ensure that all rules in P are satisfied, while the rules in P_d allow to defeat a rule in one of the $P_1, \ldots, P_{n-1}$ in favor of a rule in a less preferred component (see also Example 15). Finally, the rules in $P_1, \ldots, P_n$ encode the intuition behind LPOD, i.e. better ranked literals in an ordered disjunction are preferred.

The nap_r-literals (and their corresponding rules) serve to prevent LPOD answer sets that block disjunctive rules to be preferred over others in $L(P)$. E.g. the program

$$\begin{aligned} c \times d &\leftarrow a \\ a &\leftarrow \textit{not}\, b \\ b &\leftarrow \textit{not}\, a \end{aligned}$$

has both $\{a, c\}$ and $\{b\}$ as preferred LPOD answer sets, with the latter blocking the disjunctive rule. Applying Definition 29 yields the EOLP

$$\begin{array}{r}
\textit{not}\ nap_{c \times d \leftarrow a} \leftarrow \\
\textit{not}\ c \leftarrow a \\
\textit{not}\ d \leftarrow a, \textit{not}\ c \\
\hline
d \leftarrow a, \textit{not}\ c \\
\hline
c \leftarrow a \\
\hline
c \leftarrow a, \textit{not}\ d \\
d \leftarrow a, \textit{not}\ c \\
nap_{c \times d \leftarrow a} \leftarrow \textit{not}\ a \\
a \leftarrow \textit{not}\ b \\
b \leftarrow \textit{not}\ a
\end{array}$$

which has two preferred answer sets $\{a, c\}$ and $\{b, nap_{c \times d \leftarrow a}\}$. Without the $nap_{c \times d \leftarrow a}$ construct, only $\{b\}$ would be preferred as $\{b\}$ would satisfy all rules, while $\{a, c\}$ would defeat *not* $c \leftarrow a$.

Example 23
The result of the transformation of the LPOD from Example 21 is shown below, where $r = b \times c \times d \leftarrow$ and $s = c \times a \times d \leftarrow$.

$$\begin{array}{rr}
\textit{not}\ b \leftarrow & \textit{not}\ c \leftarrow \\
\textit{not}\ c \leftarrow \textit{not}\ b & \textit{not}\ a \leftarrow \textit{not}\ c \\
\textit{not}\ d \leftarrow \textit{not}\ b, \textit{not}\ c & \textit{not}\ d \leftarrow \textit{not}\ c, \textit{not}\ a \\
\textit{not}\ nap_r \leftarrow & \textit{not}\ nap_s \leftarrow \\
\hline
d \leftarrow \textit{not}\ b, \textit{not}\ c & d \leftarrow \textit{not}\ a, \textit{not}\ c \\
\hline
c \leftarrow \textit{not}\ b & a \leftarrow \textit{not}\ c \\
\hline
b \leftarrow & c \leftarrow \\
\hline
b \leftarrow \textit{not}\ c, \textit{not}\ d & c \leftarrow \textit{not}\ a, \textit{not}\ d \\
c \leftarrow \textit{not}\ b, \textit{not}\ d & a \leftarrow \textit{not}\ c, \textit{not}\ d \\
d \leftarrow \textit{not}\ b, \textit{not}\ c & d \leftarrow \textit{not}\ a, \textit{not}\ c \\
\neg c \leftarrow b &
\end{array}$$

This EOLP program has two proper preferred answer sets, i.e. $S_1 = \{a, b, \neg c\}$ and $S_3 = \{c\}$.

In general, the preferred LPOD answer sets for a LPOD program P coincide with the proper preferred answer sets of $L(P)$.

Theorem 22
An interpretation S is a preferred LPOD answer set of a LPOD P iff there exists a proper preferred answer set S' of $L(P)$ such that $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$.

4.3 Answer Set Programming with Consistency-Restoring Rules

In (Balduccini and Gelfond 2003; Balduccini and Mellarkod 2003) an extension of answer set programming with consistency-restoring rules (cr-rules) and preferences[10] is presented. The approach allows problems to be described in a concise, and easy to read, manner. While (Balduccini and Gelfond 2003) introduces cr-rules, (Balduccini and Mellarkod 2003) combines them with ordered disjunction (Section 4.2). A possible application is presented in (Balduccini 2004) where cr-rules are used to improve the quality of solutions in the planning domain.

Intuitively, cr-rules in a program are like normal rules, but they can only be used as a last resort to obtain solutions, i.e. only when the program without cr-rules is inconsistent we can apply some of the cr-rules to obtain a solution for the problem, taking into account the preferences among the cr-rules. Consider for example the following program where $\leftarrow_{cr}$ is used to denote cr-rules:

$$\begin{array}{rccl} r_1: & p & \leftarrow_{cr} & \textit{not}\ t \\ r_2: & q & \leftarrow_{cr} & \textit{not}\ t \\ r_3: & s & \leftarrow & \end{array}$$

and r_2 is preferred over r_1. As the single rule r_3 is consistent, the above program has only one answer set, i.e. $\{s\}$. Adding the constraint

$$r_4 : \leftarrow \textit{not}\ p, \textit{not}\ q$$

causes the combination of r_3 and r_4 to be inconsistent. In this case, one of the cr-rules r_1 or r_2 is selected for application. Either of them yields, when combined with r_3 and r_4, a consistent program, making both $\{p, s\}$ and $\{q, s\}$ candidate solutions for the program. However, since r_2's application is preferred over r_1's, only the latter candidate solution should be sanctioned as an answer set of the program.

It turns out that the intuition behind consistency restoring rules can be captured by the preferred answer set semantics using a translation similar to the one used for diagnostic and abductive reasoning with preferences (Van Nieuwenborgh and Vermeir 2003a; Van Nieuwenborgh and Vermeir 2003b). E.g., the above program can be captured by the following ordered program:

$$\begin{array}{rcl} p & \leftarrow & \textit{not}\ t, inconsistent \\ q & \leftarrow & \textit{not}\ t, inconsistent \\ \hline \textit{not}\ q & \leftarrow & inconsistent \\ \hline \textit{not}\ p & \leftarrow & inconsistent \\ \hline s & \leftarrow & \\ inconsistent & \leftarrow & \textit{not}\ s \\ & \leftarrow & \textit{not}\ p, \textit{not}\ q \\ inconsistent & \leftarrow & \textit{not}\ p, \textit{not}\ q \end{array}$$

[10] (Balduccini and Gelfond 2003; Balduccini and Mellarkod 2003) allow for both static and dynamic preferences; here we only consider the static case.

Intuitively, the non cr-rules, which must always be satisfied, are placed in the most specific component, together with rules that check when the normal program is inconsistent, so allowing consistency-restoring rules to be applied. The cr-rules are placed at the least preferred level because they will only be applied as a last resort. In the middle levels we introduce rules that allow for the defeat of the cr-rules, i.e. allowing an applicable cr-rule not to be applied. The ordering relation on these rules is the opposite of the preference relation between cr-rules as preferring (the application of) r_2 over r_1 corresponds to preferring the satisfaction of *not* $p \leftarrow$ upon *not* $q \leftarrow$. It can be verified that the above ordered program has only one preferred answer set $\{q, s, inconsistent\}$ corresponding with the application of the most preferred cr-rule, while removing r_4 from the ordered program will result in the single preferred answer set $\{s\}$ implying that no cr-rules are used.

Example 7 (Section 3.1) provides another illustration of the above translation[11]: the ordered program there is obtained by applying the construction on the following program with cr-rules:

$$\begin{array}{rcl}
lift\_weights & \leftarrow_{cr} & \\
play\_ball & \leftarrow_{cr} & \\
run & \leftarrow_{cr} & \\
swim & \leftarrow_{cr} & \\
full\_body\_exercise & \leftarrow & lift\_weights, run \\
full\_body\_exercise & \leftarrow & swim, play\_ball \\
 & \leftarrow & \textit{not}\ full\_body\_exercise
\end{array}$$

with, additionally, $run \leftarrow_{cr}\ < swim \leftarrow_{cr}$ and $ball\_play \leftarrow_{cr}\ < lift\_weights \leftarrow_{cr}$ [12].

As with the translation of LPOD (Section 4.2), this illustrates how OLP can be used to encode other extensions of answer set programming. Together with an OLP solver such as OLPS (Section 3.2), it also provides a convenient translation-based implementation of such higher level formalisms.

4.4 Ordered Logic, $\mathcal{DOL}$ and $\mathcal{DLP}^<$

Ordered logic programming has a long history: in (Laenens and Vermeir 1990; Gabbay et al. 1991; Laenens and Vermeir 1992) semantics are given for ordered programs containing non-disjunctive rules, while (Buccafurri et al. 1998; Buccafurri et al. 1999) apply the same ideas to the disjunctive case.

In all these approaches, the partial order on rules is used to decide conflicts between contradictory rules, i.e. applicable rules that cannot both be applied in a consistent interpretation. Specifically, an applicable rule r may be left unapplied, i.e. defeated, iff there exists an applied competitor r' that is more preferred, i.e. $r < r'$[13]. Clearly, these classical semantics for ordered logic use the order relation on a local basis, i.e. to resolve conflicts

[11] As the program, except the constraint, is positive we can use classical negation ($\neg$) and dispense the $inconsistent$ literals and rules.

[12] We use $r_1 < r_2$ to indicate that (the application of) the cr-rule r_1 is preferred over r_2.

[13] In some approaches one demands that the competitor r' cannot be less preferred then r, i.e. $r' \not> r$, thus allowing rules on the same level or on unrelated levels to defeat each other. However, this does not change anything to the conclusions made in this section.

between individual contradicting rules, while the semantics from Section 3 is based on a global comparison of the reducts corresponding to the candidate answer sets.

This difference in using the order relation has important consequences on the complexity of the resulting formalisms: while the semantics that use the order in a ”local” way all stay in the same complexity class of the underlying non-ordered language, i.e. Σ_1^P-complete for (Laenens and Vermeir 1990; Gabbay et al. 1991; Laenens and Vermeir 1992) and Σ_2^P-complete for (Buccafurri et al. 1998; Buccafurri et al. 1999), the “global order” semantics from Section 3 increments the complexity in the polynomial hierarchy, i.e. Σ_2^P-complete instead of Σ_1^P-complete (see Section 3.2).

The following example illustrates how the classical semantics leads to different, and, in our opinion less intuitive, results from the preferred answer set semantics.

Example 24
Consider the following ordered program.

$$\begin{array}{c} a \leftarrow \\ \hline \neg a \leftarrow \\ \hline b \leftarrow \\ \hline \neg b \leftarrow \neg a \end{array}$$

Using the semantics from Section 3 yields one preferred answer set, i.e. $I = \{a, b\}$. However, in the setting of the classical semantics I cannot be a model as the rule $\neg a \leftarrow$ is applicable, not applied and no better rule with opposite head exists. On the contrary, those classical semantics will yield $J = \{\neg a, \neg b\}$ as a model, which is certainly not a preferred answer set as the rule $b \leftarrow$ is defeated w.r.t. J, while it is satisfied w.r.t. I.

5 Application: Repairing Databases

The notion of database repair was first introduced in (Arenas et al. 1999) to address the problem of consistent query answering in inconsistent databases, which was first mentioned in (Bry 1997). (Arenas et al. 1999) describes an algorithm to compute such query answers using database repairs, while (Arenas et al. 2000) provides an algorithm to compute such repairs themselves, using a complex translation to disjunctive logic programs with exceptions (Kowalski and Sadri 1990).

Here we show that database repairs can be obtained as the preferred answer sets of a simple and intuitive ordered program corresponding to the original database and constraints. We review some definitions from (Arenas et al. 2000), using a simplified notation.

Definition 30
A **database** is a consistent set of literals. A **constraint** is a set A of literals, to be interpreted as a disjunction, $c = \vee_{a \in A} a$. The Herbrand base $\mathcal{B}_C$ of a set of constraints C is defined by $\mathcal{B}_C = \cup_{c \in C} \mathcal{B}_c$. A database D is **consistent** with a set of constraints C, where $\mathcal{B}_C \subseteq \mathcal{B}_D$, just when $D \models \wedge_{c \in C} c$, i.e. D is a classical model of C. A set of constraints C is consistent iff there exists a database D such that D is consistent with C.

Definition 31
Let D and D' be databases with $\mathcal{B}_D = \mathcal{B}_{D'}$. We use $\Delta_D(D')$ to denote the difference $D' \setminus D$. A database D induces a partial order relation[14] $\leq_D$ defined by

$$D_1 \leq_D D_2 \text{ iff } \Delta_D(D_1) \subseteq \Delta_D(D_2) \ .$$

Intuitively, $\Delta_D(D')$ contains the update operations that must be performed on D to obtain D'. A negative literal $\neg a$ in $\Delta_D(D')$ means that the fact a must be removed from D while $a \in \Delta_D(D')$ suggests adding a to D.

The $\leq_D$ relation represents the closeness to D: $D_1 \leq_D D_2$ means that D_1 is a better approximation of D than D_2 (note that $D \leq_D D$).

Definition 32
Let D be a database and let C be a set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. A database D' is a C**-repair** of D iff $D' \models C$ and D' is minimal in the $\leq_D$ partial order; i.e. $D'' \leq_D D'$ implies that $D'' = D'$.

This definition differs from the one in (Arenas et al. 2000) where $\leq_D$ was defined based on the symmetric difference $\Delta(D, D') = (D^+ \setminus D'^+) \cup (D'^+ \setminus D^+)$ rather than our $\Delta_D(D')$.

The following lemma shows that $\leq_D$ is the same using both relations.

Lemma 2
Let D , D_1 and D_2 be databases over the same Herbrand base. $\Delta_D(D_1) \subseteq \Delta_D(D_2)$ iff $\Delta(D, D_1) \subseteq \Delta(D, D_2)$.

From Lemma 2, it follows that Definition 32 is equivalent to the one in (Arenas et al. 2000).

Next we provide a construction that maps a database D and a set of constraints C to an ordered logic program $P(D, C)$ which, as shall be shown further on, has the C-repairs of D as preferred answer sets. Using ordered logic instead of logic programs with exceptions (Kowalski and Sadri 1990) greatly simplifies (w.r.t. (Arenas et al. 2000)) constructing repairs: we can dispense with the shadow versions of each predicate and we do not need disjunction. Moreover, our approach handles constraints of arbitrary size, while (Arenas et al. 2000) is limited to constraints containing up to two literals.

Definition 33
Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. The **ordered version** of D w.r.t. C, denoted $P(D, C)$, is shown below.

$$\begin{array}{cc}
\{\neg a \leftarrow \ \mid a \in D\} & (n) \\
\hline
\{a \leftarrow \ \mid a \in D\} & (d) \\
\hline
\{a \leftarrow \neg(A \setminus \{a\}) \mid \vee_{a \in A} a \in C\} & (c)
\end{array}$$

Intuitively, the c-rules enforce the constraints (they are also the strongest rules according to the partial order). The d-rules simply input the database as “default” facts while the n-rules will be used to provide a justification for literals needed to satisfy certain c-rules, thus defeating d-rules that would cause constraints to be violated.

[14] The proof that $\leq_D$ is a partial order is straightforward.

Theorem 23

Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. Each repair of D w.r.t. C is a preferred answer set of $P(D, C)$.

The reverse of Theorem 23 also holds.

Theorem 24

Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. Each preferred answer set of $P(D, C)$ is a C-repair of D.

Example 25

Consider the propositional version of the example from (Arenas et al. 2000) where the database $D = \{p, q, r\}$ and the set of constraints $C = \{\neg p \vee q, \neg p \vee \neg q, \neg q \vee r, \neg q \vee \neg r, \neg r \vee p, \neg r \vee \neg p, \}$. The program $P(D, C)$ is shown below.

$$\begin{array}{ccc}
\neg p \leftarrow & \neg q \leftarrow & \neg r \leftarrow \\
\hline
p \leftarrow & q \leftarrow & r \leftarrow \\
\hline
\neg p \leftarrow \neg q & q \leftarrow p & \neg p \leftarrow q \\
\neg q \leftarrow p & \neg q \leftarrow \neg r & r \leftarrow p \\
\neg r \leftarrow q & \neg q \leftarrow r & p \leftarrow r \\
\neg r \leftarrow \neg p & \neg p \leftarrow r & \neg r \leftarrow p
\end{array}$$

It is easily verified that $R = \{\neg p, \neg q, \neg r\}$ is the only repair of D w.r.t. C and the only proper preferred answer set of $P(D, C)$.

The example below illustrates that the simple translation of Definition 33 for database repairs does not work with other ordered formalisms (see Section 4), where a rule may be left unsatisfied if there exists an applicable (or applied) better rule with an opposite head.

Example 26

Consider the database $D = \{\neg a, \neg b\}$ and the following set of constraints $C = \{a \vee \neg b, \neg a \vee b\}$. Obviously, D itself is the only repair w.r.t. C. Now, consider $P(D, C)$ which is shown below.

$$\begin{array}{cc}
a \leftarrow & b \leftarrow \\
\hline
\neg a \leftarrow & \neg b \leftarrow \\
\hline
a \leftarrow b & b \leftarrow a \\
\neg a \leftarrow \neg b & \neg b \leftarrow \neg a
\end{array}$$

This program has only one proper preferred answer set, namely D itself. But, when we consider the same program in most other ordered formalisms, e.g. (Laenens and Vermeir 1992), we get two solutions, i.e. D and $\{a, b\}$.

6 Conclusions and Directions for Further Research

The preferred answer set semantics for ordered programs is based on a few simple intuitions: ignore rules that are defeated by other applied rules and order answer sets according to the natural order between the sets of rules that they satisfy. The resulting system is surprisingly powerful and versatile and turns out to be useful for several application areas such as database repair (Section 5) or diagnostic systems (Van Nieuwenborgh and Vermeir 2003a; Van Nieuwenborgh and Vermeir 2003b). It may also serve as a "common base language" for the encoding and implementation (through an OLP solver) of other higher level extensions of answer set programming such as ordered disjunctions (Brewka 2002; Brewka et al. 2002) or programs with consistency-restoring rules (Balduccini and Gelfond 2003; Balduccini and Mellarkod 2003).

A first implementation of an ordered logic program solver (OLPS) is available under the GPL at `http://tinf2.vub.ac.be/olp/`. After grounding, OLPS computes (a selection of) the proper preferred answer sets of a finite ordered program.

Besides research topics relating to the theory and implementation of the preferred answer set semantics, there are also some other application areas that still need to be explored. One such area concerns updates of logic programs, see e.g. (Eiter et al. 2000; Alferes and Pereira 2000). Here, it is natural to consider a new update as an addition of a new most specific level to the ordered program representing the previous version. The preferred answer set semantics will then return solutions that favor compatibility with more recent knowledge, possibly at the expense of earlier rules.

The formalism can also be extended to consider hierarchies of preference orders. Such a system could be used to model hierarchical decision making where each agent has her own preferences, and agents participate in a hierarchy representing relative authority or confidence. The semantics of such a system should reflect both individual preferences and the decision hierarchy.

Finally, it would be interesting to apply the intuition of Definition 6 to a system where preferences are expressed on literals rather than on rules. Intuitively, in such a formalism, preference would depend on the content of the candidate answer sets, rather than how well they satisfy the underlying rules[15]. In fact, a system that combines both rule- and literal-preferences, perhaps in a complex hierarchy as mentioned above, could be useful for certain applications.

Appendix: Proofs

Theorem 5
Let P be a SLP. The extended answer sets of P coincide with the answer sets of $E(P)$.

Proof
$\boxed{\Longrightarrow}$ Let M be an extended answer set of P. We must show that M is an answer set of $E(P)$, i.e. M is an answer set of the simple program $E(P)^M$ which is obtained from $E(P)$ by keeping (a) the rules $a \leftarrow \beta$ from P where $\neg a \notin M$ and (b) the constraints $\leftarrow \beta$ from P. Thus it suffices to show that $P^\star_M = E(P)^{M\star}$.

[15] A system using preferences on literals has been proposed in (Sakama and Inoue 1996) but the way these preferences are used to obtain a ranking of answer sets is different from our proposal.

Consider a rule $r = a \leftarrow \beta$ in $E(P)^M$. Since $\neg a \notin M$, this rule cannot be defeated and hence $r \in P_M$. Further, constraints $s = \leftarrow \beta$ in $E(P)^M$ can never be defeated and as M is an extended answer set they are satisfied w.r.t. M, so $s \in P_M$. As a result $E(P)^M \subseteq P_M$ and, by the monotonicity of the $\star$-operator,

$$E(P)^{M\star} \subseteq P_M^\star \quad . \tag{1}$$

Next, note that $P_M^\star = \{r \mid r \text{ applied w.r.t. } M\}^\star$, i.e. only the applied rules in P_M suffice to generate M. Each applied rule $r = a \leftarrow \beta$ in P_M must belong to $E(P)^M$ since $a \in M$, and thus $\neg a \notin M$, because r is applied. Thus $P_M^\star \subseteq E(P)^{M\star}$ and, by (1), $P_M^\star = E(P)^{M\star}$.

$\boxed{\Longleftarrow}$ Let M be an answer set of $E(P)$, i.e. $M = E(P)^{M\star}$ where $E(P)^M$ is as above. We show that M is an extended answer set of P. Obviously, all rules in $E(P)^M$ are satisfied by M and thus $E(P)^M \subseteq P_M$ and, by the monotonicity of the $\star$-operator, $M \subseteq P_M^\star$.

The constraints in $E(P)$ do not contain negation as failure and, obviously, they are satisfied by M. So, all rules r not kept in $E(P)^M$ have the form $a \leftarrow \beta$.

For a rule $r = a \leftarrow \beta$ not in $E(P)^M$ we know that $\neg a \in M$. If $\beta \not\subseteq M$, r is not applicable and thus $r \in P_M$. If, on the other hand, $\beta \subseteq M$, r is defeated because $\neg a \in M \subseteq P_M^\star$, which implies the existence of an applied competitor in P_M. Thus each unsatisfied rule in P is defeated w.r.t. M. It remains to be shown that $P_M^\star = M$. This follows from $E(P)^M \subseteq P_M$ and the fact that only rules that are not applicable w.r.t. M are in $P_M \backslash E(P)^M$. $\square$

Theorem 7
Let $<$ be a well-founded strict partial order on a set X. The binary relation $\sqsubseteq$ on 2^X defined by $X_1 \sqsubseteq X_2$ iff $\forall x_2 \in X_2 \backslash X_1 \cdot \exists x_1 \in X_1 \backslash X_2 \cdot x_1 < x_2$ is a partial order.

Proof
The relation $\sqsubseteq$ is clearly reflexive. To show that it is antisymmetric, assume that both $X_1 \sqsubseteq X_2$ and $X_2 \sqsubseteq X_1$. We show that $X_1 \backslash X_2 = X_2 \backslash X_1 = \emptyset$, from which $X_1 = X_2$. Assume that, on the contrary, $X_2 \backslash X_1 \neq \emptyset$ and let x_0 be a minimal element in $X_2 \backslash X_1$ (such an element exists because $<$ is well-founded). Because $X_1 \sqsubseteq X_2$, there exists $x_1 \in X_1 \backslash X_2$ for which $x_1 < x_0$. Since $X_2 \sqsubseteq X_1$, $x_1 \in X_1 \backslash X_2$ implies the existence of $x_2 \in X_2 \backslash X_1$ such that $x_2 < x_1 < x_0$, contradicting the fact that x_0 is minimal in $X_2 \backslash X_1$.

To show that $\sqsubseteq$ is transitive, let $X_1 \sqsubseteq X_2 \sqsubseteq X_3$ and let $x_0 \in X_3 \backslash X_1$. We consider two possibilities.

- If $x_0 \in X_2$ then, because $X_1 \sqsubseteq X_2$, there exists $x_1 \in X_1 \backslash X_2$ with $x_1 < x_0$, where we choose x_1 to be a minimal element satisfying these conditions. Again there are two cases:
  - — If $x_1 \notin X_3$ then $x_1 \in X_1 \backslash X_3$, showing that $X_1 \sqsubseteq X_3$.
  - — If $x_1 \in X_3$ then $x_1 \in X_3 \backslash X_2$ and thus, since $X_2 \sqsubseteq X_3$, there exists $x_2 \in X_2 \backslash X_3$ such that $x_2 < x_1 < x_0$. If $x_2 \in X_1$ we are done. Otherwise $x_2 \in X_2 \backslash X_1$ and thus, since $X_1 \sqsubseteq X_2$, there exists $x_3 \in X_1 \backslash X_2$ where $x_3 < x_2 < x_1 < x_0$, contradicting the minimality assumption on x_1.
- Otherwise, $x_0 \notin X_2$ and thus, because $X_2 \sqsubseteq X_3$, there exists an element $x_1 \in X_2 \backslash X_3$ such that $x_1 < x_0$ where we choose x_1 to be a minimal element satisfying these conditions. Again there are two cases:

— If $x_1 \in X_1$ then we are done as $x_1 < x_0$.
— Otherwise, since $X_1 \sqsubseteq X_2$, there exists an element $x_2 \in X_1 \backslash X_2$ such that $x_2 < x_1$. If $x_2 \notin X_3$ then we are done as $x_2 < x_0$. If, on the other hand, $x_2 \in X_3$, $X_2 \sqsubseteq X_3$ implies the existence of an element $x_3 \in X_2 \backslash X_3$ with $x_3 < x_2$, contradicting our earlier assumption of the minimality of x_1.

Hence, in call cases, there exists an element $x \in X_1 \backslash X_3$ such that $x < x_0$ and thus $X_1 \sqsubseteq X_3$. $\square$

Theorem 9

Let P be an (non-disjunctive) seminegative logic program The ordered version of P, denoted $N(P)$ is defined by $N(P) = \langle P' \cup P_\neg, < \rangle$ with $P_\neg = \{ \neg a \leftarrow \ \mid a \in \mathcal{B}_P \}$ and P' is obtained from P by replacing each negated literal *not* p by $\neg p$. The order is defined by $P' < P_\neg$, i.e. $\forall r \in P', r' \in P_\neg \cdot r < r'$ (note that $P' \cap P_\neg = \emptyset$). Then M is a stable model of P iff $M \cup \neg(\mathcal{B}_P \backslash M)$ is a proper preferred answer set of $N(P)$.

Proof

$\boxed{\Longrightarrow}$ Let $M \subseteq \mathcal{B}_P$ be a stable model of P. Thus, by definition, $P^{M^*} = M$ where P^M is the result of the Gelfond-Lifschitz transformation. We show first that $M' = M \cup \neg(\mathcal{B}_P \backslash M)$ is an extended answer set of $R = P' \cup P_\neg$. Besides $\{\neg a \mid \neg a \in M'\}$ the reduct $R_{M'}$ also contains all rules from P': if not, there would be a rule $a \leftarrow \beta$ in P^M where $\beta \subseteq M$ while $a \notin M$, contradicting that $P^{M^\star} = M$. Also, it is is easy to see that applying the $R_{M'}$-rules from $\{\neg a \leftarrow \ \mid \neg a \in M'\}$ blocks[16] all rules corresponding to rules from $P \backslash P^M$. Consequently, $R^\star_{M'} = (\{\neg a \leftarrow \ \mid \neg a \in M'\} \cup P^M)^\star = M'$.

To show that $R_{M'}$ is minimal (and M' is preferred), note for any R-extended answer set X with $R_X \sqsubset R_{M'}$, it must be the case that $R_X \supset R_{M'}$ and, moreover, $(R_X \backslash R_{M'}) \subseteq P_\neg$ since the only rules defeated by M' are in $P_\neg$. Let $\neg a \leftarrow \ \in R_X \backslash R_{M'}$. Since * is monotonic and $P' \subseteq R_X$, it follows that $R^\star_X \supseteq \{a, \neg a\}$, contradicting that X is an extended answer set. From the fact that $P' \subseteq R_{M'}$, it follows that M' is proper.

$\boxed{\Longleftarrow}$ Let M' be a proper preferred answer set of $N(P) = \langle R = P' \cup P_\neg, < \rangle$. As M' is proper, $P' \subseteq R_{M'}$. Clearly $M' \cap \neg \mathcal{B}_P = \{\neg a \mid \neg a \leftarrow \ \in (P_\neg \cap R_{M'}\}$ and $M'^+ = (R^\star_{M'})^+ = Q^\star$, where Q is obtained from $P' \subseteq R_{M'}$ by removing all rules that contain a literal $\neg a$ in the body such that $a \in M'$ and, moreover, removing $\neg a \in M'$ from the bodies of the remaining rules. Then $Q = P^{M^+}$, yielding that M^+ is a stable model of P. $\square$

Theorem 10

Let P be a positive disjunctive logic program. M is a minimal model of P iff $M' = M \cup \neg(\mathcal{B}_P \setminus M)$ is a proper preferred answer set of $D(P)$.

[16] A rule $a \leftarrow \alpha$ is blocked w.r.t. and interpretation I if $\alpha \cap \neg I \neq \emptyset$.

Proof

$\boxed{\Longrightarrow}$ Let M be a minimal model of P. Clearly, M' is total for $D(P)$ and, by construction, each rule in $D(P)$ is either satisfied or defeated. On the other hand, the rules from $D(P)_{M'} \cap (P_+ \cup P_-)$ suffice to generate M' which is thus founded. Hence, by Definition 3, M' is an extended answer set of $D(P)$. Since all rules in P_p are satisfied by any model of P, M' is also proper.

To show that M' is preferred, assume it is not, i.e. $N \sqsubset M'$ for some proper extended answer set N of $D(P)$, for which, obviously, N^+ is also a model of P. From the construction of $D(P)$ and Definition 6, we obtain that $D(P)_{M'} \setminus D(P)_N \subseteq P_+$ from which $N^+ \subset M$, contradicting the fact that M is a minimal model of P.

$\boxed{\Longleftarrow}$ Let $M' = M \cup \neg(\mathcal{B}_P \setminus M)$ be a proper preferred answer set of $D(P)$. Clearly, since $M' \models P_p$, M must be a model of P by construction of the rules in P_p. To show that M is minimal, assume that it is not, i.e. $N \subset M$ for some model N of P. A similar reasoning as above yields that $N' = N \cup \neg(\mathcal{B}_P \setminus N)$ is a proper extended answer set of $D(P)$. Moreover, from $N \subset M$, it is straightforward to show that $N' \sqsubset M'$, contradicting that M' is preferred. □

Lemma 3

Let P be a seminegative DLP and let M_1 and M_2 be proper extended answer sets of $D_n(P)$. Then $M_1^+ \subset M_2^+$ iff $M_1 \sqsubset M_2$.

Proof

$\boxed{\Longrightarrow}$ Assume that $M_1^+ \subset M_2^+$ and let $r \in D_n(P)_{M_2} \setminus D_n(P)_{M_1}$ and let $s \in D_n(P)_{M_1} \setminus D_n(P)_{M_2}$. Since both M_1 and M_2 are proper, it must be the case that $r, s \in P_- \cup P_c$. Obviously, $D_n(P)_{M_2} \cap P_- = \{\neg a \leftarrow \ \in P_- \mid \neg a \in M_2 \setminus M_2^+\}$. As both M_1 and M_2 are total and $M_1^+ \subset M_2^+$, we know that $\forall \neg a \in M_2 \setminus M_2^+ \cdot \neg a \in M_1 \setminus M_1^+$ and $\forall a \in M_2^+ \setminus M_1^+ \cdot \neg a \in M_1^+$. Combining the above yields that $D_n(P)_{M_1} \cap P_- = (D_n(P)_{M_2} \cap P_-) \cup \{\neg a \leftarrow \ \in P_- \mid a \in M_2^+ \setminus M_1^+\}$. Thus, $(D_n(P)_{M_2} \cap P_-) \subset (D_n(P)_{M_1} \cap P_-)$, making M_1 preferred upon M_2, i.e. $M_1 \sqsubset M_2$.

$\boxed{\Longleftarrow}$ Assume that $M_1 \sqsubset M_2$ (from which, $M_1 \neq M_2$) and let $a \in M_1^+ \setminus M_2^+$. Since $a \notin M_2^+$, the P_- rule $r_{\neg a} = \neg a \leftarrow$ cannot be defeated and thus $\neg a \in M_2$ and $r_{\neg a} \in D_n(P)_{M_2}$. On the other hand, $r_{\neg a} \notin D_n(P)_{M_1}$ because the rule is defeated by a rule introducing a into M_1. Thus $r_{\neg a} \in D_n(P)_{M_2} \setminus D_n(P)_{M_1}$ but, since M_1 and M_2 are proper, $r_{\neg a}$ is not countered by $D_n(P)_{M_1}$, contradicting that $M_1 \sqsubset M_2$. □

Theorem 12

Let P be a seminegative DLP. An interpretation M is a proper preferred answer set of $D_n(P)$ iff M^+ is a minimal possible model of P.

Proof

We show that the proper extended answer sets of $D_n(P)$ coincide with the possible models of P, from which, by Lemma 3, the theorem readily follows.

$\boxed{\Longrightarrow}$ Let M be a proper extended answer set of $D_n(P)$ and consider $S_M(P) = (D_n(P)_M) \cap P_c$ where all occurrences of negative literals $\neg a$ have been replaced by their negation-as-failure counterparts *not* a. We claim that $S_M(P)$ is a split program of P because $S_M(P)$ contains at least one split clause for each rule $\alpha \leftarrow \beta$ from P. Indeed, if this were not the case, there would exist a rule $\alpha \leftarrow \beta$ such that $\beta \subseteq M$ while $\alpha \cap M = \emptyset$, violating the corresponding rules in P_p and thus contradicting the assumption that M is proper.

To show that M is a possible model, it then suffices to show that $P'^{\star} = M^+$, where $P' = S_M(P)^{M^+}$ is the result of applying the Gelfond-Lifschitz transformation to $S_M(P)$ and M^+. However, we already know that $P''^{\star} = M$ where $P'' = (D_n(P)_M) \cap (P_c \cup P_-)$ because rules from $D_n(P)_M \cap P_p$ are not needed to "produce" M (indeed, for any applicable (and thus applied) rule $a \leftarrow \beta \cup \neg(\alpha \setminus \{a\})$ in $D_n(P)_M \cap P_p$, there exists an "equivalent" applied rule $a \leftarrow \beta$ in $D_n(P)_M \cap P_c$). It is then not difficult to see that P' can also be obtained from P'' by (a) removing all negative literals $\neg a$, where $\neg a \leftarrow$ is in P'', from all rule bodies, and (b) removing any rules that still contain negative literals, thus also the rules $\neg a \leftarrow$ which were not affected by (a). Since this this operation can be viewed as part of the computation of $(P''^{\star})^+ = M^+$, it follows that $P'^{\star} = M^+$.

$\boxed{\Longleftarrow}$ Let $N \subseteq \mathcal{B}_P$ be a possible model corresponding to a split program $S_N(P)$ of P and let $M = N \cup \neg(\mathcal{B}_P \setminus N)$. We use P' to denote the program obtained from $S_N(P)$ by replacing *not* β by $\neg\beta$ in all its rules. It follows that $P'' = D_n(P)_M = X \cup \{\neg a \leftarrow \mid a \in (\mathcal{B}_P \setminus N\} \cup P_p$ where $P' \subseteq X \subseteq P_c$. To show that M is a proper extended answer set we must show that (1) $M \models r$ for any $r \in P_p$, (2) $P''^{\star} = M$, and (3) each rule in $D_n(P) \setminus P''$ is defeated. Clearly, (1) follows from the fact that M is a model of P (Proposition 3.1 in (Sakama and Inoue 1994)). To establish (2), it suffices to note that $(P' \cup (P'' \cap P_-))^{\star} = M$ because M^+ is a possible model, from which $P''^{\star} = M$ because $(P' \cup (P'' \cap P_-) \subseteq P''$ (and, by definition of P'', $P''^{\star} \subseteq M$). (3) then follows immediately from the fact that M is total and (2). $\square$

Lemma 4

Let $\langle P, < \rangle$ be an ordered program and $R \subseteq P$ and $T \subseteq P$ be sets of rules. Then $T \not\sqsubset R$ iff R has a witness against T, which is equivalent to $\exists X \in \omega(T) \cdot X \subseteq R$.

Proof

From the definition of $\sqsubset$, it immediately follows that $T \not\sqsubset R$ iff R has a witness against T.

To show the second part of the lemma, suppose $X \subseteq R$ for some $X \in \omega(T)$. Then $X = \{r\} \cup (down(\{r\}) \cap T)$ for some $r \notin T$. It is then straightforward to verify that r is a witness for R against T and thus $T \not\sqsubset R$.

Conversely, if $T \not\sqsubset R$ then, according to the first part of the lemma, there exists a witness $r \in R \setminus T$ for which it is easy to verify that $r \notin T$ while $down(\{r\}) \cap T \subseteq R$, and thus $r \in \omega(T)$. $\square$

Lemma 5

Let $\langle P, < \rangle$ be an ordered program, $\langle R_i, R_o \rangle$ be a specification, C a constraint and r a minimal (according to $<$) element from $P \setminus (R_i \cup R_o)$. Let R, with $r \in R$, be a set of rules. Then $R \in \min \mu(\langle R_i, R_o \rangle, C)$ iff $R \in \min \mu(\langle R_i \cup \{r\}, R_o \rangle, C)$.

Proof
$\boxed{\Longrightarrow}$ Let $R \in \min \mu(\langle R_i, R_o\rangle, C)$. Then, by definition, $R \in \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$ because $r \in R$. Assume that, on the contrary, $R \not\in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. Hence $\exists T \in \mu(\langle R_i \cup \{r\}, R_o\rangle, C) \cdot T \sqsubset R$. But then also $T \in \mu(\langle R_i, R_o\rangle, C)$ because $\mu(\langle R_i \cup \{r\}, R_o\rangle, C) \subseteq \mu(\langle R_i, R_o\rangle, C)$. Consequently, R would not be minimal in $\mu(\langle R_i, R_o\rangle, C)$, a contradiction.

$\boxed{\Longleftarrow}$ Let $R \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$ and assume that, on the contrary, $\exists T \in \mu(\langle R_i, R_o\rangle, C) \cdot T \sqsubset R$. Surely, $r \not\in T$, since otherwise $T \in \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$ and R would not be minimal in $\mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. Because r is minimal in $P \backslash (R_i \cup R_o)$, $down(\{r\}) \cap T = down(\{r\}) \cap R$, and thus $r \in R$ is a witness for R against T. It follows that $T \not\sqsubset R$, contradicting our assumption. $\square$

Lemma 6
Let $\langle P, <\rangle$ be an ordered program, $\langle R_i, R_o\rangle$ be a specification, C a constraint and r a minimal (according to $<$) element from $P \backslash (R_i \cup R_o)$. Let R, with $r \not\in R$ be a set of rules. Then, $R \in \min \mu(\langle R_i, R_o\rangle, C)$ iff $R \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$ and $\forall T \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C) \cdot T \not\sqsubset R$.

Moreover, if C' is such that

$$\mu(\langle R_i, R_o \cup \{r\}\rangle, C') = \{T \mid T \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C) \wedge \forall m \in M \cdot \exists x \in \omega(m) \cdot x \subseteq T\} \quad (2)$$

with $M = \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$, then

$$\min \mu(\langle R_i, R_o \cup \{r\}\rangle, C') = \{T \mid T \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C) \wedge \forall m \in M \cdot \exists x \in \omega(m) \cdot x \subseteq T\} \quad (3)$$

Proof
$\boxed{\Longrightarrow}$ Suppose that $R \in \min \mu(\langle R_i, R_o\rangle, C)$. Since $\mu(\langle R_i, R_o \cup \{r\}\rangle, C) \subseteq \mu(\langle R_i, R_o\rangle, C)$ and $R \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$, $R \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$.

Let $T \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. By Lemma 5, $T \in \min \mu(\langle R_i, R_o\rangle, C)$. Consequently, R can only be minimal in $\mu(\langle R_i, R_o\rangle, C)$ if $T \not\sqsubset R$.

$\boxed{\Longleftarrow}$ Let $R \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$ such that $\forall T \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C) \cdot T \not\sqsubset R$ and assume that, on the contrary, $S \sqsubset R$ for some $S \in \mu(\langle R_i, R_o\rangle, C)$. Without loss of generality, we can assume that S is minimal in $\mu(\langle R_i, R_o\rangle, C)$. Note that $r \not\in S$ is impossible since that would put S in $\mu(\langle R_i, R_o \cup \{r\}\rangle, C)$, contradicting that R is minimal in $\mu(\langle R_i, R_o \cup \{r\}\rangle, C)$. It follows that $r \in S$ and thus, by our assumption that S is minimal in $\mu(\langle R_i, R_o\rangle, C)$ and by Lemma 5, that S is minimal in $\mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. But then our assumption guarantees $S \not\sqsubset R$, a contradiction.

$\boxed{\supseteq \text{ of (3)}}$ Let T be any set in $\min \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$ that satisfies $\forall m \in M \cdot \exists x \in \omega(m) \cdot T \cap \omega(m) \neq \emptyset$. By (2), $T \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C')$. If T would not be minimal in $\mu(\langle R_i, R_o \cup \{r\}\rangle, C')$, there would exist a $S \sqsubset T$ in $\mu(\langle R_i, R_o \cup \{r\}\rangle, C')$. But, by (2), $S \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$, contradicting that T is minimal in $\mu(\langle R_i, R_o \cup \{r\}\rangle, C)$.

$\boxed{\subseteq \text{ of (3)}}$ Assume, on the contrary, that $S \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C')$ while $S \not\in \{T \mid T \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C) \wedge \forall m \in M \cdot \exists x \in \omega(m) \cdot x \subseteq T\}$. Obviously, $S \in \mu(\langle R_i,$

$R_o \cup \{r\}\rangle, C')$ which, combined with (2), yields that $\forall m \in M \cdot \exists x \in \omega(m) \cdot x \subseteq S$ holds. So, it must be the case that $S \notin \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$, but from (2) we have that $S \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$, thus $\exists U \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C) \cdot U \sqsubset S$. Since $S \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C')$ we have that $U \notin \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C')$, but also $U \notin \mu(\langle R_i, R_o \cup \{r\}\rangle, C')$ and thus $\exists m \in M \cdot \forall x \in \omega(m) \cdot x \not\subseteq U$. As a result, U has no witness against some $m \in M$, from which, by Lemma 4, $m \sqsubset U$. Since $\sqsubset$ is a partial order, transitivity with $U \sqsubset S$ yields $m \sqsubset S$, contradicting that S has a witness against any $m \in M$. $\Box$

Lemma 7
Let $\langle P, <\rangle$ be an ordered program. If R is minimal w.r.t. $\sqsubset$ in $\mu(\langle R_i, R_o\rangle, C)$ then $R \in aset(\langle R_i, R_o\rangle, C)$.

Proof
Let $R \in \min \mu(\langle R_i, R_o\rangle, C)$. We show the result by induction on the cardinality of $P \setminus (R_i \cup R_o)$.

For the base case, we have that $R_i \cup R_o = P$ and, consequently, $\mu(\langle R_i, R_o\rangle, C) = \{R\}$ which is also returned by $aset(\langle R_i, R_o\rangle, C)$.

For the induction step, take a minimal rule $r \in P \setminus (R_i \cup R_o)$, such that $aset(\langle R_i, R_o\rangle, C)$ calls $aset(\langle R_i \cup \{r\}, R_o\rangle, C)$ and $aset(\langle R_i, R_o \cup \{r\}\rangle, C')$.

We consider two cases.

- If $r \in R$ then, by Lemma 5, $R \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. From the induction hypothesis, it follows that $R \in aset(\langle R_i \cup \{r\}, R_o\rangle, C)$. It is clear that $aset(\langle R_i \cup \{r\}, R_o\rangle, C) \subseteq aset(\langle R_i, R_o\rangle, C)$ and thus $R \in aset(\langle R_i, R_o\rangle, C)$.
- If $r \notin R$ then, by Lemma 6, $R \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C)$ and, moreover, $S \not\sqsubset R$ for any $S \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. The latter condition is ensured by the definition of C' in the code of Figure 1: it is straightforward to show that $R \models C'$ iff $R \models C$ and R has a witness against every $m \in M$. Hence

$$\mu(\langle R_i, R_o \cup \{r\}\rangle, C') = \\ \{T \mid T \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C) \wedge \forall m \in M \cdot \exists x \in \omega(m) \cdot x \subseteq T\}$$

From (3) in Lemma 6 and the induction hypothesis, it then follows that $R \in aset(\langle R_i, R_o \cup \{r\}\rangle, C')$, from which $R \in aset(\langle R_i, R_o\rangle, C)$ by the last line in Figure 1.

$\Box$

Lemma 8
Let $\langle P, <\rangle$ be an ordered program, $\langle R_i, R_o\rangle$ be a specification and C a constraint. Then $aset(\langle R_i, R_o\rangle, C) \subseteq \min \mu(\langle R_i, R_o\rangle, C)$.

Proof
Clearly, the function terminates since, within a call $aset(\langle R_i, R_o\rangle, C)$, any recursive call $aset(\langle R_i', R_o'\rangle, C')$ satisfies $R_i \cup R_o \subset R_i' \cup R_o'$ while $R_i \cup R_o$ is bounded from above by the finite set P.

Since the function *aset* is recursive, we can show the result by induction on the depth of the recursion.

For the base case, we consider two possibilities:

- If $\langle R_i, R_o\rangle$ is inconsistent with C then $\mu(\langle R_i, R_o\rangle, C) = \emptyset$ and the lemma holds vacuously.
- If $R_i \cup R_o = P$, $R_i^\star$ is an extended answer set and $\langle R_i, R_o\rangle$ is consistent with C then $\mu(\langle R_i, R_o\rangle, C) = \{R_i\}$ and, again, the lemma holds vacuously.

For the induction step, let $R \in aset(\langle R_i, R_o\rangle, C)$. There are two possibilities.

- $R \in aset(\langle R_i \cup \{r\}, R_o\rangle, C)$ and thus, by the induction hypothesis, $R \in \min \mu(\langle R_i \cup \{r\}, R_o\rangle, C)$. Lemma 5 then implies $R \in \min \mu(\langle R_i, R_o\rangle, C)$.
- If $R \in aset(\langle R_i, R_o \cup \{r\}\rangle, C')$ where C' satisfies

$$\mu(\langle R_i, R_o \cup \{r\}\rangle, C') = \{T \mid T \in \mu(\langle R_i, R_o \cup \{r\}\rangle, C) \wedge \forall m \in M \cdot \exists x \in \omega(m) \cdot x \subseteq T\} \quad (4)$$

The induction hypothesis implies $R \in \min \mu(\langle R_i, R_o \cup \{r\}\rangle, C')$. Together with (4) and Lemma 6, this implies that $R \in \min \mu(\langle R_i, R_o\rangle, C)$.

□

Theorem 13
Let $\langle P, <\rangle$ be an ordered program, $\langle R_i, R_o\rangle$ be a specification and C a constraint. Then $aset(\langle R_i, R_o\rangle, C) = \min \mu(\langle R_i, R_o\rangle, C)$.

Proof
Immediate from Lemma 7 and Lemma 8 □

Theorem 17
The problem of deciding, given an arbitrary ordered program P and a literal a, whether a occurs in every preferred answer set of P is Π_2^P-hard.

Proof
The proof uses a reduction of the known Π_2^P-hard problem of deciding whether a quantified boolean formula $\phi = \forall x_1, \ldots, x_n \cdot \exists y_1, \ldots, y_m \cdot F$ is valid, where we may assume that $F = \wedge_{c \in C} c$ with each c a disjunction of literals over $X \cup Y$ with $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ $(n, m > 0)$.

The program P corresponding to ϕ is shown below.

$$P_1 = \{x \leftarrow \quad \neg x \leftarrow \ \mid x \in X\} \ \left| \ \begin{array}{c} P_2 = sat \leftarrow \\ \hline P_3 = \{y \leftarrow \quad \neg y \leftarrow \ \mid y \in Y\} \\ \hline P_4 = sat \leftarrow \neg sat \\ \hline P_5 = \{\neg sat \leftarrow c' \mid c \in C\} \end{array} \right.$$

where c' is obtained from c by taking the negation, i.e. if $c = l_1 \vee \ldots \vee l_n$, then c' denotes $\neg l_1 \wedge \ldots \wedge \neg l_n$.

Obviously, the construction of P can be done in polynomial time. Intuitively, the rules in P_1 and P_3 are used to guess a truth assignment for $X \cup Y$.

In the sequel, we will abuse notation by using x_M and y_M where M is an answer set for P, to denote subsets of M, e.g. $x_M = X \cap M$ and in expressions such as $F(x_M, y_M)$ which stands for $F(x_1, \ldots, x_n, y_1, \ldots y_m)$ with $x_i = \mathbf{true}$ iff $x_i \in x_M$ and, similarly, $y_j = \mathbf{true}$ iff $y_j \in y_M$. We will also sometimes abbreviate the arguments of F, writing e.g. $F(x, y)$ rather than $F(x_1, \ldots, x_n, y_1, \ldots y_m)$.

The following properties of P are straightforward to show:

1. A rule in P_5 is only applicable if F does not hold. If we have an extended answer set M containing $\neg sat \in M$, then $F(x_M, y_M)$ does not hold.
2. Any preferred answer set M of P always satisfies all the rules in P_5, otherwise $M' = (M \setminus \{sat\}) \cup \{\neg sat\}$ would be better then M, a contradiction.
3. For each combination X_i of X we have at least one preferred answer set containing X_i. For extended answer sets M_1 and M_2, with $M_1 \cap X \neq M_2 \cap X$, neither $M_1 \sqsubset M_2$ nor $M_2 \sqsubset M_1$ holds, as the rules in P_1 are unrelated to any other rules.

We show that ϕ is valid iff $sat \in M$ for every preferred answer set M of P.

To show the "if" part, assume that every preferred answer set M contains $sat \in M$. Suppose ϕ is not valid, then there exists a combination X_i of X such that $\forall y \cdot \neg F(X_i, y)$. By (3) we know that there must exist at least one preferred answer set M' with $X_i \subseteq M'$. Combining (1) and (2) with the fact that $\forall y \cdot \neg F(X_i, y)$ yields $\neg sat \in M'$, contradicting that every preferred answer set contains sat. Thus, ϕ is valid.

To show the reverse, assume that ϕ is valid. Suppose there exists a preferred answer set M of P not containing sat, i.e. $sat \notin M$. Then, by construction of P, $\neg sat \in M$. By (1), this yields that $F(x_M, y_M)$ does not hold. As M is preferred any other preferred answer set M' containing x_M (we will have one for each combination of Y due to the rules in P_3) must also make F not valid, otherwise M could not be preferred, as the one making F valid would satisfy the rule P_4 which is defeated w.r.t. M, making it obviously preferred upon M. This yields that for the combination x_M of X no combination of Y exists making F true, i.e. $\exists x \cdot \forall y \cdot \neg F(x, y)$, a contradiction. Thus, every preferred answer set contains sat. $\square$

Theorem 18
Let P be an ELP. Then, S is an extended answer sets of P iff there is an answer set S' of $E(P)$ such that $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$.

Proof
$\boxed{\Longrightarrow}$ Let S be an extended answer set of P and consider the interpretation $S' = S \cup \{\textit{not}_a \mid \textit{not}\ a \leftarrow \beta \in P \wedge S \models \beta \cup \{\textit{not}\ a\}\}$. Then, S' is an answer set of $E(P)$.

First we show that every rule in $E(P)$ is satisfied w.r.t. S'. Constraint rules are clearly satisfied w.r.t. S'. Suppose there is a rule $a \leftarrow \beta, \textit{not}\ \neg a, \textit{not}\ (\textit{not}_a) \in E(P)$ such that $S' \models \beta \cup \{\textit{not}\ \neg a, \textit{not}\ (\textit{not}_a)\}$ and $S' \not\models a$. By construction of the rule, the corresponding rule in P is also applicable and not applied w.r.t. S. As S is an extended answer set, this means that there must be an applied rule $X \leftarrow \beta$ with $X = \neg a$ or $X = \textit{not}\ a$ in P defeating the rule $a \leftarrow \beta$. By construction of the rules in $E(P)$ and the construction of S', the corresponding rule in $E(P)$ is also applied w.r.t. S', i.e. either $S' \models \neg a$ or $S' \models \textit{not}_a$, contradicting $S' \models \{\textit{not}\ \neg a, \textit{not}\ (\textit{not}_a)\}$. The same reasoning can be done for the rules $\textit{not}_a \leftarrow \beta, \textit{not}\ a$ in $E(P)$.

From S an extended answer set of P we know that $(P_S)^{S^\star} = S$. Consider the rules $a \leftarrow \beta \in P$ with a an ordinary literal, such that $S \models \beta \cup \{a\}$ and thus $a \leftarrow \beta \setminus \textit{not}\,(\beta^-) \in (P_S)^S$. This rule is represented in $E(P)$ as $a \leftarrow \beta, \textit{not}\,\neg a, \textit{not}\,(\textit{not}_a)$ and by construction of S' we have that $a \leftarrow \beta \setminus \textit{not}\,(\beta^-) \in E(P)^{S'}$. Thus, $S \subseteq E(P)^{S'^\star}$. Clearly, if $S \models \beta \cup \{\textit{not}\, a\}$ for a rule $\textit{not}\ a \leftarrow \beta \in P$, then the corresponding rule $\textit{not}_a \leftarrow \beta, \textit{not}_a \in E(P)$ is also applicable w.r.t. S' by construction of S', thus $\textit{not}_a \leftarrow \beta \setminus \textit{not}\,(\beta^-) \in E(P)^{S'}$, yielding that $S' \setminus S \subseteq E(P)^{S'^\star}$. Finally, $S' \subseteq E(P)^{S'^\star}$ and because all rules are satisfied w.r.t. S', $E(P)^{S'^\star} = S'$.

$\boxed{\Longleftarrow}$ Let S' be an answer set of $E(P)$ and let $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. We show that S is an extended answer set of P.

First we show that every rule in P is either satisfied or defeated w.r.t. S. Again, constraints are clearly satisfied w.r.t. S. Suppose there is a rule $a \leftarrow \beta \in P$ such that $S \models \beta$ and $S \not\models a$. The corresponding rule in $E(P)$, i.e. either $a \leftarrow \beta, \textit{not}\,\neg a, \textit{not}\,(\textit{not}_a)$ or $\textit{not}_a \leftarrow \beta, \textit{not}\ a$, is satisfied w.r.t. S', yielding that either $S' \not\models \textit{not}\,\neg a$, $S' \not\models \textit{not}\,(\textit{not}_a)$ or $S' \not\models \textit{not}\, a$. Thus, there is an applied rule $X \leftarrow Y \in E(P)$ with $X = \neg a$, $X = \textit{not}_a$ or $X = a$. By construction of the rules in $E(P)$ this means that the corresponding rule in P is also applied w.r.t. S, defeating the rule $a \leftarrow \beta$.

Finally we have to show that $(P_S)^{S^\star} = S$, which is quite obvious as the generating, i.e. applicable and applied, rules $a \leftarrow \beta \setminus \textit{not}\,(\beta^-) \in E(P)^{S'}$ for $E(P)^{S'^\star}$ are also in $(P_S)^S$ and they clearly do not depend on any literal of the form $\textit{not}_a$. Thus, $(P_S)^{S^\star} = S$. $\square$

Lemma 9
Let $P = \langle R, < \rangle$ be an extended ordered logic program. Every proper extended answer set S' of $N_s(P)$ is of the form $S' = S \cup \{\phi(a), \neg\phi(\textit{not}\, a), \neg\phi(\neg a) \mid a \in S\} \cup \{\phi(\textit{not}\, a) \mid a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S\} \cup \{\neg\phi(a) \mid \textit{not}\ a \leftarrow \beta \in R \wedge S \models \beta \cup \{\textit{not}\, a\}\}$, where $S \subseteq \mathcal{B}_R \cup \neg\mathcal{B}_R$.

Proof
Take S' a proper extended answer set of $N_s(P)$. Let $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$.

First consider $a \in S$. For a to be in S', we must have that $\phi(a) \in S'$, by construction of the rules $a \leftarrow \phi(a)$ in R_c. Combining $\phi(a) \in S'$, with the constraint $\leftarrow \phi(a), \phi(\textit{not}\ a)$ and the rule $\phi(\textit{not}\ a) \leftarrow$, yields that $\neg\phi(\textit{not}\, a) \in S'$ as otherwise S' cannot be proper, i.e. $\phi(\textit{not}\ a) \leftarrow$ cannot be defeated.

As $\phi(a) \in S'$, there must be an applied rule $\phi(a) \leftarrow \phi(\beta) \in N_s(P)$, i.e. $\phi(\beta) \cup \{\phi(a)\} \subseteq S'$. Then, also $\neg\phi(\neg a) \leftarrow \phi(\beta), \phi(a)$ is applicable and yields, combined with the constraint $\leftarrow \phi(a), \phi(\neg a)$ and the fact that S' is proper, $\neg\phi(\neg a) \in S'$. Thus, $X = \{\phi(a), \neg\phi(\textit{not}\, a), \neg\phi(\neg a) \mid a \in S\} \subseteq S'$.

Let $Y = (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S$. By construction of the rules in $N_s(P)$, we can never defeat the rules $\{\phi(\textit{not}\ a) \leftarrow\ \mid a \in Y\}$, yielding that $Z = \{\phi(\textit{not}\, a) \mid a \in Y\} \subseteq S'$.

Finally, one can easily see that the only rules left which can derive something new into S', i.e. something in $T = S' \setminus S \setminus X \setminus Z$, are of the form $\neg\phi(a) \leftarrow \phi(\beta), \phi(\textit{not}\ a)$, where $a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S$. Thus, $T \subseteq \{\neg\phi(a) \mid a \in Y\}$.

To end, we show that $T = \{\neg\phi(a) \mid \textit{not}\ a \leftarrow \beta \in R \wedge S \models \beta \cup \{\textit{not}\ a\}\}$. The $\supseteq$ direction is obvious, as the corresponding rules $\neg\phi(a) \leftarrow \phi(\beta), \phi(\textit{not}\ a)$ in $N_s(P)$ will be applicable, thus applied by construction of the rules, by construction of S' and the fact that S' is a proper extended answer set. The $\subseteq$ direction is also quite obvious. Take $\neg\phi(a) \in T$

with $a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S$ and suppose there is no rule *not* $a \leftarrow \beta$ in P that is applied w.r.t. S. Then, by construction of the rules in $N_s(P)$, there is no applied rule in $N_s(P)$ with $\neg\phi(a)$ in the head, contradicting that S' is a proper extended answer set, as $T \subseteq S'$. $\square$

Lemma 10
Let $P = \langle R, < \rangle$ be an extended ordered logic program. Then, S is an extended answer set of P iff $S' = S \cup \{\phi(a), \neg\phi(\textit{not}\ a), \neg\phi(\neg a) \mid a \in S\} \cup \{\phi(\textit{not}\ a) \mid a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S\} \cup \{\neg\phi(a) \mid \textit{not}\ a \leftarrow \beta \in R \wedge S \models \beta \cup \{\textit{not}\ a\}\}$ is a proper extended answer set of $N_s(P)$.

Proof
$\boxed{\Longrightarrow}$ Take S an extended answer set of P and take S' as defined. First we show that every rule r of $N_s(P)$ is either satisfied or defeated w.r.t. S'.

- Every rule $\phi(\textit{not}\ a) \leftarrow\ \in R_n$ is either satisfied or defeated w.r.t. S'. Suppose $\phi(\textit{not}\ a) \notin S'$. Then, by construction of S', $a \in S$. As S is an extended answer set of P, there is a rule $a \leftarrow \beta \in P$ such that $S \models \{a\} \cup \beta$. For this rule we have three corresponding rules in $N_s(P)$, i.e. $\phi(a) \leftarrow \phi(\beta)$, $\neg\phi(\neg a) \leftarrow \phi(\beta), \phi(a)$ and $\neg\phi(\textit{not}\ a) \leftarrow \phi(\beta), \phi(a)$. By construction of S' and $\phi(\beta)$ we have that $\phi(\beta) \cup \{\phi(a), \neg\phi(\textit{not}\ a)\} \subseteq S'$, which yields that $\neg\phi(\textit{not}\ a) \leftarrow \phi(\beta), \phi(a)$ defeats $\phi(\textit{not}\ a) \leftarrow$.
- The rules of type $\phi(a) \leftarrow \phi(\beta)$ are either satisfied or defeated w.r.t. S'. Suppose that $\phi(\beta) \subseteq S'$ and $\phi(a) \notin S'$, implying that $a \notin S'$, which yields that the corresponding rule $a \leftarrow \beta$ in P is applicable w.r.t. S by construction of the rules in $N_s(P)$ and the construction of S'. As $a \notin S'$ we have $a \notin S$, implying that the rule $a \leftarrow \beta$ in P must be defeated w.r.t. S by an applied rule $\neg a \leftarrow \beta'$ or *not* $a \leftarrow \beta'$, as S is an extended answer set of P. Again by construction of S' and the rules in $N_s(P)$, the rule $\phi(\neg a) \leftarrow \phi(\beta')$ or $\phi(\textit{not}\ a) \leftarrow \phi(\beta')$ in $N_s(P)$ is applied w.r.t. S', making either the rule $\neg\phi(a) \leftarrow \phi(\beta'), \phi(\neg a)$ or the rule $\neg\phi(a) \leftarrow \phi(\beta'), \phi(\textit{not}\ a)$ applied, defeating $\phi(a) \leftarrow \phi(\beta)$.
- The rules of type $\phi(\textit{not}\ a) \leftarrow \phi(\beta)$ are either satisfied or defeated w.r.t. S'. Suppose that $\phi(\beta) \subseteq S'$ and $\phi(\textit{not}\ a) \notin S'$. This implies that $a \in S$, by construction of S'. As S is an extended answer set, there must be a rule $a \leftarrow \beta' \in P$ such that $S \models \beta' \cup \{a\}$. Then, by construction of S', the rule $\phi(a) \leftarrow \phi(\beta') \in R'$ is applied w.r.t. S', but also the rule $\neg\phi(\textit{not}\ a) \leftarrow \phi(\beta'), \phi(a) \in R'$ is, defeating $\phi(\textit{not}\ a) \leftarrow \phi(\beta)$.
- The rules left in R', i.e. rules of type $\neg\phi(\neg a) \leftarrow \phi(\beta), \phi(a);\ \neg\phi(\textit{not}\ a) \leftarrow \phi(\beta), \phi(a)$ and $\neg\phi(a) \leftarrow \phi(\beta), \phi(\textit{not}\ a)$, are always satisfied w.r.t. S' by construction of S'.
- All the rules in R_c are always satisfied w.r.t. S'. By construction of S', the constraints $\leftarrow \phi(a), \phi(\textit{not}\ a)$ and $\leftarrow \phi(a), \phi(\neg a)$ are always satisfied w.r.t. S', i.e. inapplicable. Clearly, rules of the form $a \leftarrow \phi(a)$ will be either applied, or inapplicable. Finally, the constraints in P are satisfied w.r.t. S, thus the constraints $\leftarrow \phi(\beta) \in R_c$ are satisfied w.r.t. S', yielding that S' is proper.

The only thing left to proof is that $N_s(P)^\star_{S'} = S'$. First of all, let $T = \{\phi(\textit{not}\ a) \mid \phi(\textit{not} a) \in S'\}$. Obviously, $T \subseteq N_s(P)^\star_{S'}$ as $\phi(\textit{not}\ a) \leftarrow\ \in N_s(P)_{S'}$ for every $\phi(\textit{not} a) \in S'$.

We know that $(R_S)^{S^\star} = S$ as S is an extended answer set of P. A rule $a \leftarrow \beta \in R$ with

a an ordinary literal that is kept in $(R_S)^S$ is of the form $a \leftarrow \beta \setminus \textit{not}\,\beta^-$[17]. For all those rules in $(R_S)^S$ we have that $\{\phi(\textit{not}\,a) \mid a \in \beta^-\} \subseteq T$, and by construction of the rules in $N_s(P)$, the construction of S', the fact that T can be derived immediately in $N_s(P)^\star_{S'}$ and the fact that $(R_S)^{S\star} = S$, this yields that $\{\phi(a) \mid a \in S\} \subseteq N_s(P)^\star_{S'}$, implying that also $S \subseteq N_s(P)^\star_{S'}$ by the rules $a \leftarrow \phi(a)$ in R_c.

For every applied rule $\phi(a) \leftarrow \phi(\beta) \in N_s(P)$ with $a \in S$, we have that $\neg\phi(\neg a) \leftarrow \phi(\beta), \phi(a)$ and $\neg\phi(\textit{not}\,a) \leftarrow \phi(\beta), \phi(a)$ are satisfied w.r.t. S' and thus are in $N_s(P)_{S'}$, yielding that $\{\neg\phi(\neg a), \neg\phi(\textit{not}\,a) \mid a \in S\} \subseteq N_s(P)^\star_{S'}$.

Finally, consider the applied rules *not* $a \leftarrow \beta \in P$. By construction of S' and the rules $\phi(\textit{not}\,a) \leftarrow \phi(\beta)$ and $\neg\phi(a) \leftarrow \phi(\beta), \phi(\textit{not}\,a)$ in $N_s(P)$, both rules are satisfied w.r.t. S', thus in $N_s(P)_{S'}$. From *not* $a \leftarrow \beta$ applied w.r.t. S we have that $S \models \beta \cup \{\textit{not} a\}$ and by construction of S', both rules in $N_s(P)_{S'}$ are applicable w.r.t. $S \cup \{\phi(a), \neg\phi(\textit{not}\,a), \neg\phi(\neg a) \mid a \in S\} \cup \{\phi(\textit{not}\,a) \mid a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S\}$, which is already shown to be in $N_s(P)^\star_{S'}$. Thus, $\{\neg\phi(a) \mid \textit{not}\,a \leftarrow \beta \in R \wedge S \models \beta \cup \{\textit{not}\,a\}\} \subseteq N_s(P)^\star_{S'}$. As a conclusion, $S' \subseteq N_s(P)^\star_{S'}$ and $S' = N_s(P)^\star_{S'}$ by the fact that each rule is satisfied or defeated w.r.t. S'.

$\boxed{\Longleftarrow}$ Take S' a proper extended answer set of $N_s(P)$. From Lemma 9 we know that S' is of the form $S' = S \cup \{\phi(a), \neg\phi(\textit{not}\,a), \neg\phi(\neg a) \mid a \in S\} \cup \{\phi(\textit{not}\,a) \mid a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S\} \cup \{\neg\phi(a) \mid \textit{not}\,a \leftarrow \beta \in R \wedge S \models \beta \cup \{\textit{not} a\}\}$, where $S \subseteq \mathcal{B}_R \cup \neg\mathcal{B}_R$. The only thing we have to show is that S is an extended answer set of P.

By construction, all constraints in $\leftarrow \beta \in P$ are satisfied w.r.t. S as the constraints $\leftarrow \phi(\beta) \in R_c$ are satisfied w.r.t. S'. Every rule $a \leftarrow \beta$ in P, with a an extended literal, is either satisfied or defeated w.r.t. S. Suppose not, i.e. $S \models \beta \wedge S \not\models a$ and there is no defeater w.r.t. S in P for $a \leftarrow \beta$. By construction of the rules in $N_s(P)$, the corresponding rule $\phi(a) \leftarrow \phi(\beta) \in N_s(P)$ is applicable w.r.t. S', but not applied by construction of S'. As S' is a proper extended answer set, the rule $\phi(a) \leftarrow \phi(\beta)$ must be defeated w.r.t. S', i.e. there must be an applied rule $\neg\phi(a) \leftarrow \beta', \phi(X)$ in $N_s(P)$ with $X = \neg a$ or $X = \textit{not}\, a$ when a is an ordinary literal and $X = \hat{a}$ when a is an extended literal. This yields that the corresponding rule in P is also applied w.r.t. S, defeating $a \leftarrow \beta \in P$, a contradiction.

Finally, we show that $(R_S)^{S\star} = S$. For every applied rule $\phi(a) \leftarrow \phi(\beta) \in N_s(P)$ w.r.t. S' and a an ordinary literal, we have that $\{a \mid \phi(\textit{not}\,a) \in \phi(\beta)\} \cap S = \emptyset$ by construction of S' and thus of S. This implies that $a \leftarrow \beta \setminus \textit{not}\,\beta^- \in (R_S)^S$ and $\beta \setminus \textit{not}\,\beta^- \subseteq S$. We know that $N_s(P)^\star_{S'} = S'$ and that $\{\phi(a) \mid a \in S\}$ are only produced by the applied rules of the form $\phi(a) \leftarrow \phi(\beta)$ with a an ordinary literal, starting from $\{\phi(\textit{not}\,a) \mid a \in (\mathcal{B}_R \cup \neg\mathcal{B}_R) \setminus S\}$.

Now, combining the above with the construction of the rules in $N_s(P)$ yields that $S \subseteq (P_S)^{S\star}$. As every rule in P is either applied or defeated w.r.t. S we have $S = (P_S)^{S\star}$. $\square$

Theorem 20
Let $P = \langle R, < \rangle$ be an extended ordered logic program. Then, M is a preferred answer set of P iff there exists a proper preferred answer set M' of $N_s(P)$, such that $M = M' \cap (\mathcal{B}_R \cup \neg\mathcal{B}_R)$.

[17] Constraints, i.e. $\leftarrow \beta$, in $(R_S)^S$ are not important in here, as they do not play a role in $(R_S)^{S\star}$ because S is an extended answer set.

Proof

$\boxed{\Longrightarrow}$ Take S a preferred answer set of P and take S' as defined in Lemma 10. By that same lemma we get that S' is a proper extended answer set of $N_s(P)$. Suppose S' is not preferred, then there exists a proper extended answer set $S'' \neq S'$ of $N_s(P)$ such that $S'' \sqsubseteq S'$. Let $T = S'' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. From Lemma 10 we know that T is an extended answer set of P. As both S' and S'' are proper, we have that $N_s(P)_{S'} \setminus N_s(P)_{S''}$ and $N_s(P)_{S''} \setminus N_s(P)_{S'}$ only contain rules from R' and R_n.

Suppose $N_s(P)_{S'} \setminus N_s(P)_{S''}$ contains a rule r from R'. Then, $S'' \sqsubseteq S'$ implies that there exists a rule $r' \in N_s(P)_{S''} \setminus N_s(P)_{s'}$ such that $r' < r$. From the proof of Lemma 10 we know that only rules of the type $\phi(a) \leftarrow \phi(\beta) \in R'$, with a an extended literal, can be defeated and we also know from the same proof that the rules in P corresponding with the defeated rules in R' w.r.t. S' (S'') are also defeated w.r.t. S (T), yielding that $T \sqsubseteq S$, a contradiction.

Suppose $N_s(P)_{S'} \setminus N_s(P)_{S''}$ contains only rules from R_n, which implies that $P_S \setminus P_T = \emptyset$. Then $S'' \sqsubseteq S'$ yields that $N_s(P)_{S''} \setminus N_s(P)_{S'}$ must contain at least a rule from R' to counter the R_n rules in $N_s(P)_{S'} \setminus N_s(P)_{S''}$. As a result, $P_S \neq P_T$, implying that, combined with $P_S \setminus P_T = \emptyset$, we have $T \sqsubseteq S$, a contradiction.

Thus, S' is a proper preferred answer set of $N_s(P)$ and $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$.

$\boxed{\Longleftarrow}$ Take S' a proper preferred answer set of $N_s(P)$. Let $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. Again from Lemma 10 we know that S is an extended answer set of P. Suppose S is not preferred. Then, there exists an extended answer set T of P such that $T \sqsubseteq S$, which implies $P_S \neq P_T$. Take T' as described by Lemma 10 and we get that T' is a proper extended answer of $N_s(P)$ with the same defeated rules in R' as the corresponding defeated rules in P. The same holds for defeated rules w.r.t. S and S'. Then $T \sqsubseteq S$ implies $T' \sqsubseteq S'$, a contradiction. $\square$

Lemma 11

Let P be an LPOD. S is answer set of P iff $S' = S \cup \{nap_{a_1 \times \ldots \times a_n \leftarrow \beta} \mid a_1 \times \ldots \times a_n \leftarrow \beta \in P \wedge S \not\models \beta\}$ is a proper extended answer set of $L(P)$.

Proof

$\boxed{\Longrightarrow}$ Take S an answer set of P, yielding that there exists a split program P' of P such that S is an answer set of P'. Take S' as defined. We first show that every rule in $L(P)$ is either satisfied or defeated w.r.t. S'.

Obviously, all rules in P_r corresponding with normal rules in P are satisfied as these rules are contained in every split program. The rules $a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_n\} \setminus a_i$ for every $1 \leq i \leq n$ for each ordered disjunctive rule are clearly satisfied w.r.t. S' as every split program contains an option that is satisfied w.r.t. S, thus also w.r.t. S'. Furthermore, the rules $nap_r \leftarrow \textit{not}\, l$ and $nap_r \leftarrow l$ in P_r are also satisfied w.r.t. S' by construction of S', i.e. when an ordered disjunctive rule r is not applicable w.r.t. S, we have $nap_r \in S'$ making the rules clearly satisfied, while in the case r is applicable, none of the rules are applicable.

The rules $\textit{not}\, nap_r \leftarrow \ \in P_d$ for which $nap_r \notin S'$ are clearly satisfied, while the ones for which $nap_r \in S'$ are clearly defeated by an applied rule $nap_r \leftarrow l$ or $nap_r \leftarrow \textit{not}\, l$, this by construction of S'. Further, the rules $\textit{not}\, a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{i-1}\} \in P_d$ for which $S' \models \beta \cup \textit{not}\,\{a_1, \ldots, a_{i-1}\}$ and $a_i \in S'$ are defeated by the corresponding applied rule $a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{i-1}\} \in P_i$.

Finally, if a rule $a \leftarrow \beta \in P_1 \cup \ldots \cup P_n$ is applicable but not applied, it is defeated by a rule *not* $a \leftarrow \beta \in P_d$, by construction of P_d.

The only thing left to show is $(L(P)_{S'})^{S'^\star} = S'$. By construction of $L(P)$ we get that $P' \subseteq L(P)$. This yields, as all rules in P' are satisfied w.r.t. S, that $P'^S \subseteq (L(P)_{S'})^{S'}$. As a result, $S \subseteq (L(P)_{S'})^{S'^\star}$. Clearly, for each unapplicable ordered disjunctive rule $r = a_1 \times \ldots \times a_n \leftarrow \beta$, i.e. $\exists l \in \beta \cdot S \not\models l$, we have that $nap_r \leftarrow \textit{not}\, l$ (when l is a literal) or $nap_r \leftarrow l$ (when l is a naf-literal) is in $L(P)_{S'}$, yielding that $S' \subseteq (L(P)_{S'})^{S'^\star}$. As all rules are satisfied or defeated w.r.t. S', this results in $S' = (L(P)_{S'})^{S'^\star}$.

$\boxed{\Longleftarrow}$ Take S' a proper extended answer set of $L(P)$ and let $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. Consider $L(P)_{S'}$, i.e. all satisfied rules in $L(P)$ w.r.t. S'. Take $P' \subseteq L(P)_{S'}$ in the following way:

1. Take all rules in P_r corresponding with normal rules in P. As S' is proper and the satisfaction of the selected rules does not change w.r.t. S, every normal rule in P is also in P'.
2. Take every applicable and applied rule r from the P_i's. By construction of $L(P)$ at most one option for every ordered disjunctive rule is taken, as options in P_j with $j > i$ are not applicable, because they contain *not* $H(r)$ in the body; and options r' in P_j with $j < i$ must be defeated, otherwise r would not be applicable because it contains *not* $H(r')$ in the body.
3. For every ordered disjunctive rule that is not applicable w.r.t. S' (or S) take an arbitrary option to be in P'.

Clearly, P' as constructed above, is a split program of P.

The rules $c_i \leftarrow \beta, \textit{not}\,(\{c_1, \ldots, c_n\} \setminus \{c_i\}) \in P_r$ that are applicable and applied are not needed to produce the S part in $(L(P)_{S'})^{S'^\star}$, as also the rule $c_i \leftarrow \beta, \textit{not}\,\{c_1, \ldots, c_{i-1}\} \in P_i$ is applicable and applied, thus in $(L(P)_{S'})^{S'}$. Furthermore, the rules with nap_r in the head are also not needed to derive S. Thus, all rules needed to produce S in $L(P)$ are also in P', yielding that $P'^{S^\star} = S$, making S an answer set of P. $\square$

Lemma 12
Let P be a LPOD and let $S \neq T$ be proper extended answer sets for $L(P)$. Then, the following is equivalent:

1. $S \sqsubseteq T$,
2. $L(P)_S \neq L(P)_T$ and $\forall r \in L(P)_T \setminus L(P)_S, \exists r' \in L(P)_S \setminus L(P)_T \cdot r' < r$,
3. $L(P)_S \neq L(P)_T$ and $\exists k \cdot Sat^k_{L(P)}(T) \subset Sat^k_{L(P)}(S) \wedge \forall j < k \cdot Sat^j_{L(P)}(T) = Sat^j_{L(P)}(S)$.

where $Sat^k_{L(P)}(S)$ denotes the set of rules in $P_k \subseteq L(P)$ that are satisfied w.r.t. S.

Proof
By definition (1) $\Leftrightarrow$ (2) holds.

$\boxed{(2) \Rightarrow (3)}$ First, it can never happen that $L(P)_T \setminus L(P)_S = \emptyset$. Suppose it is, than together with $L(P)_T \neq L(P)_S$, it implies $L(P)_T \subset L(P)_S$. We have three cases:

- $r = a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{i-1}\} \in P_i$ with $r \in L(P)_S \setminus L(P)_T$ yields $T \models \beta \cup \textit{not}\,\{a_1, \ldots, a_{i-1}\}$ and $T \not\models a_i$. By virtue of Lemma 11 we have that $T \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$ is an LPOD answer set, thus we must have a satisfied, w.r.t. T, option $r' = a_j \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{j-1}\} \in P_j$ with $j > i$. For this rule, the corresponding rule $r'' = \textit{not}\,a_j \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{j-1}\} \in P_d$ is defeated w.r.t. T. Further, all rules $\textit{not}\,a_k \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{k-1}\} \in P_d$ with $k \leq j$ are satisfied w.r.t. T, thus also w.r.t. S. Now consider S which satisfies the rule r. We get two cases:
  - — $\exists k \leq i \cdot a_k \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{k-1}\} \in P_k$ is applied w.r.t. S. However, this implies the rule $\textit{not}\,a_k \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{k-1}\} \in P_d$ is defeated w.r.t. S, a contradiction.
  - — $S \not\models \beta$, i.e. the ordered disjunctive rule s corresponding with the rules r, r' and r'' is not applicable w.r.t. S. Then, Lemma 11 yields that $nap_s \notin T$, while $nap_s \in S$. The former implies that the rule $\textit{not}\,nap_s \leftarrow\ \in P_d$ is satisfied w.r.t. T and while it should also be satisfied by S, the latter clearly shows its defeated, a contradiction.

- $r = \textit{not}\,a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{i-1}\} \in P_d$ with $r \in L(P)_S \setminus L(P)_T$ yields that there also must be a rule $r' = a_j \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{j-1}\} \in P_j$ with $r' \in L(P)_S \setminus L(P)_T$, which is handled in the previous case.
- $r = \textit{not}\,nap_s \leftarrow\ \in P_d$ with $r \in L(P)_S \setminus L(P)_T$ yields that s is applicable w.r.t. S and not applicable w.r.t. T. However, s applicable w.r.t. S implies at least one option $a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{i-1}\} \in P_i$ of s will be applied w.r.t. S, defeating the rule $t = \textit{not}\,a_i \leftarrow \beta, \textit{not}\,\{a_1, \ldots, a_{i-1}\} \in P_d$. However, s not applicable w.r.t. T implies t is satisfied w.r.t. T, yielding $t \in L(P)_T \setminus L(P)_S$, a contradiction.

As $L(P)_T \setminus L(P)_S \neq \emptyset$, $S \sqsubseteq T$ implies that $L(P)_S \setminus L(P)_T \neq \emptyset$. Take the most specific rule $r' \in L(P)_S \setminus L(P)_T$. Clearly, $r' < r$ for every $r \in L(P)_T \setminus L(P)_S$. As S and T are proper we must have $r' \in P_k$ for a certain $k \in [1 \ldots n]$. As r' is the most specific rule in $L(P)_S \setminus L(P)_T$, we have $\forall j < k \cdot (L(P)_S \setminus L(P)_T) \cap P_j = (L(P)_T \setminus L(P)_S) \cap P_j = \emptyset$, which is equivalent with $\forall j < k \cdot Sat^j_{L(P)}(T) = Sat^j_{L(P)}(S)$. As r' is the most specific rule in $L(P)_S \setminus L(P)_T$, every rule $r \in L(P)_T \setminus L(P)_S$ must belong to a P_i with $i > k$ (or to P_d), which yields that $(L(P)_T \setminus L(P)_S) \cap P_k \subset (L(P)_S \setminus L(P)_T) \cap P_k$; and as a result $Sat^k_{L(P)}(T) \subset Sat^k_{L(P)}(S)$.

$\boxed{(3) \Rightarrow (2)}$ Take a k so that (3) holds. From $\forall j < k \cdot Sat^j_{L(P)}(T) = Sat^j_{L(P)}(S)$ it follows that $\forall j < k \cdot (L(P)_S \setminus L(P)_T) \cap P_j = (L(P)_T \setminus L(P)_S) \cap P_j = \emptyset$. From $Sat^k_{L(P)}(T) \subset Sat^k_{L(P)}(S)$, we get $(Sat^k_{L(P)}(S) \setminus Sat^k_{L(P)}(T)) \subset (L(P)_S \setminus L(P)_T)$ and $(L(P)_T \setminus L(P)_S) \cap P_k = \emptyset$, which directly yields that $r' < r$ for every $r' \in Sat^k_{L(P)}(S) \setminus Sat^k_{L(P)}(T)$ and every $r \in L(P)_T \setminus L(P)_S$. □

Theorem 22
An interpretation S is a preferred LPOD answer set of a LPOD P iff there exists a proper preferred answer set S' of $L(P)$ such that $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$.

Proof
$\boxed{\Longrightarrow}$ Take S a preferred LPOD answer set of P. From Lemma 11 we have that there exists a proper extended answer set S' of $L(P)$ such that $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. Suppose S' is not preferred, i.e. there exists a proper extended answer set $T' \neq S'$ of $L(P)$ such that

$T' \sqsubseteq S'$. Again, by virtue of Lemma 11, we have that $T = T' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$ is an LPOD answer set of P.

Applying Lemma 12 to $T' \sqsubseteq S'$ w.r.t. $L(P)$ yields

$$\exists k \cdot Sat^k_{L(P)}(S') \subset Sat^k_{L(P)}(T') \wedge \forall j < k \cdot Sat^j_{L(P)}(S') = Sat^j_{L(P)}(T') \ . \tag{5}$$

By construction of $L(P)$ we have that $Sat^1_{L(P)}(S') = S^1(P) \setminus \{a \leftarrow \beta \in P\}$ and $Sat^j_{L(P)}(S') \setminus Sat^{j-1}_{L(P)}(S') = S^j(P)$[18] (and the same for T and T'). Combined with (5) this yields

$$\exists k \cdot S^k(P) \subset T^k(P) \wedge \forall j < k \cdot S^j(P) = T^j(P) \ . \tag{6}$$

But, (6) implies that T is LPOD-preferred upon S for P, a contradiction.

$\boxed{\Longleftarrow}$ Take S' a proper preferred answer set of $L(P)$. By Lemma 11, $S = S' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$ is an LPOD answer set of P. Suppose S is not LPOD preferred, i.e. there exists an LPOD answer set T such that $T \sqsubset_b S$. This yields

$$\exists k \cdot S^k(P) \subset T^k(P) \wedge \forall j < k \cdot S^j(P) = T^j(P) \ . \tag{7}$$

Again from Lemma 11 we have that there must exist a proper extended answer set T' of $L(P)$ such that $T = T' \cap (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. By construction of $L(P)$, for every ordered disjunctive rule r that is satisfied to degree k, the corresponding options in $P_1, \ldots, P_{k-1}$ are defeated, while the corresponding options in $P_k, \ldots, P_n$ are satisfied, yielding that (it also hold for T and T')

$$Sat^1_{L(P)}(S') = S^1(P) \setminus \{a \leftarrow \beta \in P\} \ , \tag{8}$$

$$Sat^k_{L(P)}(S') = Sat^{k-1}_{L(P)}(S') + S^k(P) \ . \tag{9}$$

Combining (8), (9) and (7) results in

$$\exists k \cdot Sat^k_{L(P)}(S') \subset Sat^k_{L(P)}(T') \wedge \forall j < k \cdot Sat^j_{L(P)}(S') = Sat^j_{L(P)}(T') \ . \tag{10}$$

Using (10) with Lemma 12 yields $T' \sqsubseteq S'$, a contradiction. □

Lemma 2
Let D , D_1 and D_2 be databases over the same Herbrand base. $\Delta_D(D_1) \subseteq \Delta_D(D_2)$ iff $\Delta(D, D_1) \subseteq \Delta(D, D_2)$.

Proof
To show the "only if" part, assume that $p \in \Delta(D, D_1) = (D^+ \backslash {D_1}^+) \cup ({D_1}^+ \backslash D^+)$. We consider two possibilities:

1. If $p \in D^+ \backslash {D_1}^+$ then $p \in D$ and $\neg p \in D_1$. Hence, by definition, $\neg p \in \Delta_D(D_1)$ Since $\Delta_D(D_1) \subseteq \Delta_D(D_2)$, $\neg p \in D_2 \backslash D$ and thus $p \in D^+ \backslash {D_2}^+ \subseteq \Delta(D, D_2)$.
2. If $p \in {D_1}^+ \backslash D^+$ then $p \in \Delta_D(D_1) \subseteq \Delta_D(D_2)$. Since p is an atom, this implies that $p \in {D_2}^+ \backslash D^+ \subseteq \Delta(D, D_2)$.

[18] This is not completely correct as rules in $Sat^j_{L(P)}(S') \setminus Sat^{j-1}_{L(P)}(S')$ are options from ordered disjunctive rules, while $S^j(P)$ contains ordered disjunctive rules. However, as only one option for such a rule can be in the former, there is a one-to-one mapping between the elements in the former and the elements in the latter.

To show the "if" part, assume that $p \in \Delta_D(D_1) = D_1 \setminus D$. We consider two cases.

1. If p is an atom, then $p \in {D_1}^+ \setminus D^+ \subseteq \Delta(D, D_1) \subseteq \Delta(D, D_2)$. Thus $p \in D^+ \setminus {D_2}^+$ or $p \in {D_2}^+ \setminus D^+$. The former is impossible because $p \not\in D^+$. The latter implies that $p \in (D_2 \setminus D) = \Delta_D(D_2)$.
2. If p is a negative literal, then $\neg p \in (D^+ \setminus {D_1}^+) \subseteq \Delta(D, D_1)$. Thus $\neg p \in \Delta(D, D_2)$ and, because $\neg p \in D$, $\neg p \in D^+ \setminus {D_2}^+$. Consequently, $p \in ({D_2}^+ \setminus D^+) \subseteq \Delta_D(D_2)$.

$\square$

Lemma 13
Let D be a database and let C be a consistent set of constraints with L_C the set of literals occurring in C. For any C-repair R of D, we have that $R \setminus L_C = D \setminus L_C$, i.e. D and R agree on $\mathcal{B}_D \setminus \mathcal{B}_{L_C}$.

Proof
Straightforward. Suppose e.g. that D and R do not agree on $l \not\in L_C$, i.e. $l \in R$ and $\neg l \in D$. Since l does not occur in any constraint, $R' = (R \setminus \{l\}) \cup \neg l \models C$ and, moreover $R' \leq_D R$, contradicting that R is a repair. $\square$

Theorem 23
Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. Each repair of D w.r.t. C is a preferred answer set of $P(D, C)$.

Proof
Let R be a repair of D w.r.t. C. By definition, $\mathcal{B}_R = \mathcal{B}_D$ and R is consistent. On the other hand, $P(D, C)_R$ obviously contains a rule $d\colon a \leftarrow$ for each $a \in R \cap D$ and a rule $n\colon \neg a \leftarrow$ for each $a \in R \setminus D$. Thus $P(D, C)^\star_R = R$.

We next show that R satisfies or defeats each rule in $P(D, C)$. By definition, and the construction of $P(D, C)$, all c-rules are satisfied. On the other hand, any n-rule $n\colon a \leftarrow$ which is not satisfied is defeated by an applied d-rule $d\colon \neg a \leftarrow$. As to the d-rules, Lemma 13 implies that each rule $d\colon a \leftarrow$ where a does not occur in C is applied.

The remaining case concerns d-rules $d\colon a \leftarrow$ where a occurs in C and $a \not\in R$, i.e. the rule is not satisfied. We consider two possibilities: either $\neg a$ occurs in C and thus, by the construction of $P(D, C)$, there exists a satisfied c-rule $c\colon \neg a \leftarrow \alpha$, defeating $d\colon a \leftarrow$, or $\neg a$ does not occur in C. The latter case is impossible since then, by Lemma 13, R should agree with D on $\neg a$, contradicting our assumption that $\neg a \in R \setminus D$.

Hence, R is an extended answer set.

To show that R is minimal w.r.t $\sqsubseteq$, assume that, on the contrary, there exists an extended answer set $M \sqsubseteq R$ of $P(D, C)$ such that $M \neq R$. Thus, by Definition 6,

$$\forall r \in P(D, C)_R \setminus P(D, C)_M \cdot \exists r' \in P(D, C)_M \setminus P(D, C)_R \cdot r' < r \ . \tag{11}$$

Note that $P(D, C)_R \setminus P(D, C)_M \neq \emptyset$ (otherwise, $M = R$ would follow). Since c-rules cannot be defeated, any r as in (11) must have a label d or n. By definition, R satisfies all c-rules and thus any r' satisfying (11) must also be a d or n rule. Combining these observations yields that any r and r' satisfying (11) must be an n-rule and a d-rule, respectively. But this implies that $\Delta_D(M) \subset \Delta_D(R)$, contradicting the fact that R is a C-repair of D.

$\square$

Lemma 14
Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. Each preferred answer set M of $P(D, C)$ satisfies C, i.e. $M \models C$.

Proof
Clearly, as C is consistent, there must exist a proper extended answer set I, i.e. $I \models C$. By Lemma 1, each preferred answer set must be proper, from which this lemma follows. $\square$

Theorem 24
Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. Each preferred answer set of $P(D, C)$ is a C-repair of D.

Proof
Let M be a preferred answer set of $P(D, C)$. From Lemma 14 it follows that $M \models C$. Assume that, on the contrary, M is not a C-repair of D, i.e. there exists a C-repair R such that $\Delta_D(R) \subset \Delta_D(M)$ (Definition 32). Let l be a literal in $\Delta_D(M) \backslash \Delta_D(R)$. Thus $l \in M$ while $\neg l \in D \cap R$. By construction, $P(D, C)$ contains an d-rule $r = d\colon \neg l \leftarrow$ in $P_R \backslash P_M$. Since M is preferred and, by Theorem 23, R is an extended answer set, $M \sqsubseteq R$ implies the existence of a rule $r' < r$ with $r' \in P_M \setminus P_R$. But any such rule r' must be a c-rule which, by definition, is satisfied by R, a contradiction. $\square$

References

ALFERES, J. J. AND PEREIRA, L. M. 2000. Updates plus preferences. See Ojeda-Aciego et al. (2000), 345–360.

ARENAS, M., BERTOSSI, L., AND CHOMICKI, J. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, Philadelphia, 68–79.

ARENAS, M., BERTOSSI, L., AND CHOMICKI, J. 2000. Specifying and querying database repairs using logic programs with exceptions. In *Proceedings of the 4th International Conference on Flexible Query Answering Systems*. Springer-Verlag, Warsaw, 27–41.

BALDUCCINI, M. 2004. Usa-smart: Improving the quality of plans in answer set planning. In *Proceedings of the 6th International Symposium on Practical Aspects of Declarative Languages (PADL2004)*. Lecture Notes in Artificial Intelligence. Springer Verlag, Dallas, Texas, USA.

BALDUCCINI, M. AND GELFOND, M. 2003. Logic programs with consistency-restoring rules. In *Proceedings of the International Symposium on Logical Formalization of Commonsense Reasoning*, P. Doherty, J. McCarthy, and M.-A. Williams, Eds. AAAI 2003 Spring Symposium Series.

BALDUCCINI, M. AND MELLARKOD, V. 2003. Cr-prolog2: Cr-prolog with ordered disjunction. In *ASP03 Answer Set Programming: Advances in Theory and Implementation, volume 78 of CEUR Workshop proceedings*.

BREWKA, G. 2002. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, Edmonton, Canada, 100–105.

BREWKA, G., BENFERHAT, S., AND LE BERRE, D. 2002. Qualitative choice logic. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning*. Morgan Kaufmann, Toulouse, France, 158–169.

BREWKA, G. AND EITER, T. 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence 109,* 1-2 (April), 297–356.

BREWKA, G., NIEMELA, I., AND SYRJANEN, T. 2002. Implementing ordered disjunction using answer set solvers for normal programs. In *European Workshop, JELIA 2002*, S. Flesca, S. Greco, N. Leone, and G. Ianni, Eds. Lecture Notes in Artificial Intelligence, vol. 2424. Springer Verlag, Cosenza, Italy, 444–455.

BRY, F. 1997. Query answering in information systems with integrity constraints. In *Integrity and Internal Controls in Information Systems I - Increasing the confidence in information systems*. Chapman & Hall.

BUCCAFURRI, F., FABER, W., AND LEONE, N. 1999. Disjunctive logic programs with inheritance. In *Logic Programming: The 1999 International Conference*, D. De Schreye, Ed. MIT Press, Las Cruces, New Mexico, 79–93.

BUCCAFURRI, F., LEONE, N., AND RULLO, P. 1998. Disjunctive ordered logic: Semantics and expressiveness. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, A. G. Cohn, L. K. Schubert, and S. C. Shapiro, Eds. Morgan Kaufmann, Trento, 418–431.

CLARK, K. L. 1978. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plemum Press, New York, 293–322.

DE VOS, M. AND VERMEIR, D. 2001. Semantic forcing in disjunctive logic programs. *Computational Intelligence 17,* 4, 651–684.

DELGRANDE, J., SCHAUB, T., AND TOMPITS, H. 2000. Logic programs with compiled preferences. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI'2000)*, W. Horn, Ed. IOS Press, Amsterdam, Netherlands, 392–298.

DIX, J., GOTTLOB, G., AND MAREK, V. W. 1996. Reducing disjunctive to non-disjunctive semantics by shift-operations. *Fundamenta Informaticae 28,* 1-2, 87–100.

EITER, T., FINK, M., SABBATINI, G., AND TOMPITS, H. 2000. Considerations on updates of logic programs. See Ojeda-Aciego et al. (2000), 2–20.

EITER, T. AND GOTTLOB, G. 1993. Complexity results for disjunctive logic programming and application to nonmonotonic logics. In *Proceedings of the 1983 International Logic Programming Symposium*. MIT Press, Vancouver, 266–279.

EITER, T., GOTTLOB, G., AND LEONE, N. 1997. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science 189,* 1-2, 129–177.

GABBAY, D., LAENENS, E., AND VERMEIR, D. 1991. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan Kaufmann, Cambridge, Mass, 208–217.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, R. A. Kowalski and K. A. Bowen, Eds. The MIT Press, Seattle, Washington, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3-4, 365–386.

INOUE, K. AND SAKAMA, C. 1994. On positive occurrences of negation as failure. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, J. Doyle, E. Sandewall, and P. Torasso, Eds. Morgan Kaufmann, Bonn, Germany, 293–304.

INOUE, K. AND SAKAMA, C. 1996. A fixpoint characterization of abductive logic programs. *Journal of Logic Programming 27,* 2 (May), 107.136.

INOUE, K. AND SAKAMA, C. 1998. Negation as failure in the head. *Journal of Logic Programming 35,* 1 (April), 39–78.

KAKAS, A. C., KOWALSKI, R. A., AND TONI, F. 1992. Abductive logic programming. *Journal of Logic and Computation 2,* 6, 719–770.

KOWALSKI, R. A. AND SADRI, F. 1990. Logic programs with exceptions. In *Proceedings of the 7th International Conference on Logic Programming*, D. H. D. Warren and P. Szeredi, Eds. The MIT Press, Jerusalem, 598–613.

LAENENS, E. AND VERMEIR, D. 1990. A logical basis for object oriented programming. In *European Workshop, JELIA 90*, J. van Eijck, Ed. Lecture Notes in Artificial Intelligence, vol. 478. Springer Verlag, Amsterdam, The Netherlands, 317–332.

LAENENS, E. AND VERMEIR, D. 1992. Assumption-free semantics for ordered logic programs: On the relationship between well-founded and stable partial models. *Journal of Logic and Computation 2,* 2, 133–172.

LEONE, N., RULLO, P., AND SCARCELLO, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation 135,* 2 (June), 69–112.

LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Journal of Artificial Intelligence 138,* 1-2, 39–54.

OJEDA-ACIEGO, M., DE GUZMÁN, I. P., BREWKA, G., AND PEREIRA, L. M., Eds. 2000. *Logic in Artificial Intelligence*. Lecture Notes in Artificial Intelligence, vol. 1919. Springer Verlag, Malaga, Spain.

PRZYMUSINSKI, T. C. 1991. Stable semantics for disjunctive programs. *New Generation Computing 9,* 3-4, 401–424.

REITER, R. 1980. A logic for default reasoning. *Artificial Intelligence 13*, 81–132.

SAKAMA, C. AND INOUE, K. 1994. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning 13,* 1, 145–172.

SAKAMA, C. AND INOUE, K. 1996. Representing priorities in logic programs. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, M. J. Maher, Ed. MIT Press, Bonn, 82–96.

SCHAUB, T. AND WANG, K. 2001. A comparative study of logic programs with preference. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI) 2001*, B. Nebel, Ed. Morgan Kaufmann, Seatlle, Washington, USA, 597–602.

VAN EMDEN, M. H. AND KOWALSKI, R. A. 1976. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery 23,* 4, 733–742.

VAN GELDER, A., ROSS, K. A., AND SCHLIPF, J. S. 1988. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, Austin, Texas, 221–230.

VAN NIEUWENBORGH, D. AND VERMEIR, D. 2003a. Ordered diagnosis. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR2003)*. Lecture Notes in Artificial Intelligence, vol. 2850. Springer Verlag, Almaty, Kazachstan, 244–258.

VAN NIEUWENBORGH, D. AND VERMEIR, D. 2003b. Ordered programs as abductive systems. In *Proceedings of the APPIA-GULP-PRODE Conference on Declarative Programming (AGP2003)*. Regio di Calabria, Italy, 374–385.

WANG, K., ZHOU, L., AND LIN, F. 2000. Alternating fixpoint theory for logic programs with priority. In *CL*, J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, Eds. Lecture Notes in Computer Science, vol. 1861. Springer, London, UK, 164–178.