Preferred Answer Sets for Ordered Logic Programs

Davy Van Nieuwenborgh[⋆] and Dirk Vermeir

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
{dvnieuwe,dvermeir}@vub.ac.be

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 as well as disjunction. We illustrate an application of the approach by considering database repairs, where minimal repairs are shown to correspond to preferred answer sets.

1 Introduction

The intuition behind the stable model semantics, and, more generally, behind answer set semantics 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 $P_M^\star$ 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 $P_M^\star = M$, 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 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., [7]), but the reduct is restricted to rules that are not defeated. We show that the 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

⋆ Supported by the FWO

the rules induces a natural partial order on the reducts of the program, and hence on its candidate answer sets. Minimal elements in this induced partial order are called preferred answer sets.

The resulting semantics for ordered programs has applications in several areas. E.g. we show that the minimal repairs of a database D [1] w.r.t. a set of constraints C correspond with the preferred answer sets of an ordered program $P(D, C)$ that can be trivially constructed out of D and C.

The remainder of this paper is organized as follows: Section 2 extends the usual answer set semantics to cover also inconsistent programs. In Section 3, 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 4 proposes an algorithm to compute such preferred answer sets and shows that the complexity of deciding whether there exists a proper preferred answer set containing a given atom is Σ_2^P-complete. Section 5 illustrates the use of preferred answer set semantics in solving database repair problems. We believe that our approach may have further natural applications, e.g. in diagnostic systems [8, 3] where the order can be used to differentiate between the normal and fault models of the system. Due to space restrictions, all proofs and a full account of relationships with other approaches had to be omitted.

2 Extended Answer Sets for Simple Programs

In this section, we consider simple programs, e.g. logic programs with only classical negation (and no disjunction in the head of a rule).

We use the following basic definitions and notation. A *literal* is an *atom* a or a negated atom $\neg a$. For a set of 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 $I \cap \neg I = \emptyset$.

The following definitions (see, e.g. [7]) will also be useful. A *disjunctive logic program* (DLP) is a countable set of rules of the form $\alpha \leftarrow \beta, \textit{not}(\gamma)$ where α and $\alpha \cup \beta \cup \gamma$ are nonempty finite sets of literals (we use *not* to denote negation as failure, $\textit{not}(\gamma)$ is an abbreviation for $\{\textit{not}(c) \mid c \in \gamma\}$). If α is always a singleton, we drop the "disjunctive" qualification. If, for all rules, all literals in $\alpha \cup \beta \cup \gamma$ are atoms, the program is called *seminegative* and if, furthermore, $\gamma = \emptyset$, the program is said to be *positive*. An *interpretation* for a DLP is any consistent set of literals (for seminegative programs, we can restrict to a set of atoms).

For a DLP P without negation as failure ($\gamma = \emptyset$), an *answer set* is a minimal interpretation I that is *closed* under the rules of P (i.e. for a rule $\alpha \leftarrow \beta$, $\alpha \cap I \neq \emptyset$ whenever $\beta \subseteq I$). For a DLP P containing negation as failure and an interpretation I, the Gelfond-Lifschitz transformation [4] yields the *reduct* program P^I that consists of those rules $\alpha \leftarrow \beta$ where $\alpha \leftarrow \beta, \textit{not}(\gamma)$ is in P and $\gamma \cap I = \emptyset$ (note that all rules in

P^I 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.

Simple logic programs are logic programs without negation as failure.

Definition 1. *A* ***simple logic program*** *(SLP) is a countable set P of* ***rules*** *of the form $a \leftarrow \alpha$ where $\{a\} \cup \alpha$ is a finite set of literals*[1].

The ***Herbrand base*** *$\mathcal{B}_P$ of P contains all atoms appearing in P. An* ***interpretation*** *I of P is any consistent subset of $\mathcal{B}_P \cup \neg\mathcal{B}_P$. An interpretation I is* ***total*** *if $\mathcal{B}_P \subseteq I \cup \neg I$.*

A rule $r = a \leftarrow \alpha$ is ***satisfied*** *by I, denoted $I \models r$, if $\{a\} \subseteq I$ whenever $\alpha \subseteq I$, i.e. if r is* ***applicable*** *($\alpha \subseteq I$) then it must be* ***applied*** *($\alpha \cup \{a\} \subseteq I$). The rule r is* ***defeated*** *w.r.t. I iff there exists an applied* ***competing rule*** *$\neg a \leftarrow \alpha'$ in P, such a rule is said to* ***defeat*** *r.*

Thus, a rule $r = a \leftarrow \alpha$ cannot be left unsatisfied unless one accepts the opposite conclusion $\neg a$ which is motivated by a competing applied rule $\neg a \leftarrow \alpha'$ that *defeats* r.

Example 1. Consider the SLP P_1 containing the rules $\neg a \leftarrow$, $\neg b \leftarrow$, $a \leftarrow \neg b$, and $b \leftarrow \neg a$. For the interpretation $I = \{\neg a, b\}$ we have that I satisfies all rules in P_1 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[10] model of the positive logic program consisting of the rules in R where negative literals $\neg a$ are considered as fresh atoms. This operator is monotonic, i.e. if $R \subseteq Q$ then $R^\star \subseteq Q^\star$.

For the program of Example 1, we have that $P_1^\star = \{\neg a, \neg b, a, b\}$ is inconsistent. The following definition allows us to not apply certain rules, when computing a consistent interpretation for programs such as P_1.

Definition 2. *The* ***reduct*** *$P_I \subseteq P$ of P w.r.t. I contains just the rules satisfied by I, i.e. $P_I = \{r \in P \mid I \models r\}$. An interpretation I is* ***founded*** *if $P_I^\star = I$.*

A founded interpretation I is an ***extended answer set*** *of P if all rules in P are satisfied or defeated.*

Thus, extended answer set semantics deals with inconsistency in a simple yet intuitive way: when faced with contradictory applicable rules, just select one for application 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_1 from Example 1 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 ${P_1}_{M_1} = P_1 \setminus \{\neg b \leftarrow\}$ while ${P_1}_{M_2} = P_1 \setminus \{\neg a \leftarrow\}$, and ${P_1}_{M_3} = P_1 \setminus \{a \leftarrow \neg b, b \leftarrow \neg a\}$, 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.

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

Theorem 1. *Let P be a simple logic program and let M be an answer set of P Then M is the unique extended answer set of P.*

While allowing for P_M, with M an extended answer set, to be a strict subset of P, Definition 2 still maximizes the set of satisfied rules w.r.t. an extended answer set.

Theorem 2. *Let P be a simple logic program and let M be an extended answer set for P. Then P_M is maximal among the reducts of founded interpretations of P.*

The reverse of Theorem 2 does not hold in general, as can be seen from the following example.

Example 2. Consider the program P_2 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 $P_{2N} = \{b \leftarrow, \; \neg b \leftarrow \neg a\}$ which is obviously maximal since $P_2^\star$ is inconsistent. Still, N is not an extended answer set because $\neg a \leftarrow$ is not defeated.

However, for total interpretations, founded interpretations with maximal reducts are extended answer sets.

Theorem 3. *Let P be a simple logic program 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.*

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 [5] for logic programs with exceptions.

Definition 3. *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 $a \leftarrow \alpha$ by its extended version $a \leftarrow \alpha, \; not(\neg a)$.*

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

Theorem 4. *Let P be an SLP. The extended answer sets of P coincide with the answer sets of $E(P)$.*

Let P be a simple program. Consider the function $\delta_P : 2^{\mathcal{B}_P} \to 2^{\mathcal{B}_P}$ where $\delta_P(I) = \{a \notin I \mid \neg a \notin I \wedge (a \leftarrow \alpha) \in P \wedge \alpha \subseteq I\}$. 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 seen to be an extended answer set. Hence the extended answer set semantics is universal.

Theorem 5. *Each simple logic program has extended answer sets.*

3 Ordered Programs and Preferred Answer Sets

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_1 from Example 1 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 4. *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]*. 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 $R_1 \neq R_2$.*

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

Intuitively, $r_1 < r_2$ indicates that r_1 is more preferred than r_2 while the definition of $\sqsubset$ shows how this preference carries over to reducts: 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. Note that, unlike other approaches, e.g. [6], we do not require that r_1 is applied and neither does r_1 need to be a competitor of r_2, as illustrated in the following example.

Example 3. Consider $P_4 = \langle R, < \rangle$, 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 (and uncomfortable) suspicion that not studying leads to

[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.

[4] Note that the rule $pass \leftarrow \neg pass$ can only be satisfied by an interpretation containing *pass*, *without* providing a justification for it.

failure.

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

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 $r_2 \in R_{M_2} \backslash R_{M_1}$ is countered by $r_1 \in R_{M_1} \backslash R_{M_2}$ 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 6. *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.*

Preferred answer sets for ordered programs correspond to minimal reducts in the $\sqsubseteq$-partial order that is induced by the rule preference order.

Definition 5. *Let $P = \langle R, < \rangle$ be an ordered logic program. A* ***preferred answer set*** *for P is any extended answer set M of R such that R_M is minimal w.r.t $\sqsubseteq$ among the reducts of all extended answer sets of R. An extended answer set is* ***proper*** *if it satisfies all minimal (according to $<$) rules in R.*

Proper extended answer sets respect the strongest (minimal) rules of the program. The following theorem confirms that taking the minimal (according to the $\sqsubseteq$ order on the corresponding reducts) elements among the proper extended answer sets is equivalent to selecting the proper elements among the preferred answer sets.

Theorem 7. *Let P be an ordered program. The set of minimal proper extended answer sets of P coincides with the set of proper preferred answer sets of P.*

In Example 3, $M_1 = \{\mathit{pass}, \mathit{study}\}$ is the only proper preferred answer set.

Example 4. Consider the ordered program $\langle P_1, < \rangle$ where P_1 is as in Example 1 and $<$ is as shown below.

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

The reducts of the extended answer sets of P_1 are $P_{1M_1} = P_1 \backslash \{\neg b \leftarrow\}$, $P_{1M_2} = P_1 \backslash \{\neg a \leftarrow\}$, and $P_{1M_3} = P_1 \backslash \{a \leftarrow \neg b, b \leftarrow \neg a\}$ which are ordered by $P_{1M_1} \sqsubset P_{1M_3}$ and $P_{1M_2} \sqsubset P_{1M_3}$. Thus $\langle P_1, < \rangle$ 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 $\{a \leftarrow \mathit{not}(b), b \leftarrow \mathit{not}(a)\}$, i.e. the stronger rules of $\langle P_1, < \rangle$ where negation as failure (*not*) replaces classical negation ($\neg$). In fact, the ordering of P_1, 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 8. *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 $P' < P_\neg$ (note that $P' \cap P_\neg = \emptyset$). Then M is a stable model of P iff $M \cup \neg(\mathcal{B}_P \setminus M)$ is a proper preferred answer set of $N(P)$.*

Interestingly, preference can also simulate disjunction.

Definition 6. *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 $P_+ = \{a \leftarrow \mid a \in \mathcal{B}_P\}$, $P_- = \{\neg a \leftarrow \mid a \in \mathcal{B}_P\}$, $P_p = \{a \leftarrow \beta \cup \neg(\alpha \setminus \{a\}) \mid (\alpha \leftarrow \beta) \in P \wedge a \in \alpha\}$, 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 5. Consider the disjunctive program $P_5 = \{a \vee b \leftarrow,\ a \leftarrow b,\ b \leftarrow a\}$. This program illustrates that the shifted version[5] of a disjunctive program need not have the same models, see e.g.[11]. The program $D(P_5)$ is represented below.

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

$D(P_5)$ has a single proper preferred answer set $\{a, b\}$ which is also the unique minimal model of P_5. 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 9. *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 8 and Theorem 9, it is natural to try to simulate programs that combine negation as failure and disjunction.

Definition 7. *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 $P_c = \{a \leftarrow \beta' \mid (\alpha \leftarrow \beta) \in P \wedge a \in \alpha\}$, $P_- = \{\neg a \leftarrow \mid a \in \mathcal{B}_P\}$, $P_p = \{a \leftarrow \beta' \cup \neg(\alpha \setminus \{a\}) \mid (\alpha \leftarrow \beta) \in P \wedge a \in \alpha\}$, and $P_p < P_- < P_c$. Here, β' is obtained from β by replacing all occurrences of not$(a) \in \beta$ by $\neg a \in \beta'$.*

[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 $a \leftarrow \beta \cup \mathit{not}((\alpha \setminus \{a\}))$ for each $a \in \alpha$.

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.

Theorem 10. *Let P be a seminegative disjunctive logic program. If M is an answer set of P then $M \cup \neg(\mathcal{B}_P \backslash M)$ is a proper preferred answer set of $D_n(P)$.*

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

Example 6. Consider the seminegative disjunctive program $P_6 = \{a \vee b \leftarrow, b \leftarrow a, a \leftarrow \textit{not}(a)\}$. 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 $P_6^M = \{a \vee b \leftarrow,\ b \leftarrow a\}$ has only one minimal answer set $\{b\} \neq M$.

However, $\{a, b\}$ is the unique minimal preferred answer set of $D_n(P_6)$ 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 [9].

Theorem 11. *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.*

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

4 Computing Preferred Answer Sets

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

In the algorithm below, we use *extended literals*, i.e. literals of one of the forms a, $\neg a$, $\textit{not}(a)$ or $\textit{not}(\neg a)$. A set of extended literals is consistent if for any ordinary literal l in I, neither $\neg l$ nor $\textit{not}(l)$ appear in I. A set of extended literals I can be *normalized* by removing $\textit{not}(l)$ whenever $\neg l$ is in I. We use $\phi(I)$ to denote the normalized version of I and $\psi(I)$ to denote $I \backslash \{\textit{not}(l) \mid l \in (\mathcal{B}_P \cup \neg\mathcal{B}_P\}$.

The *not*-forms are needed in the procedure because, for an interpretation I, a rule $a \leftarrow \alpha$ may be satisfied without being applied or blocked, e.g. when α contains literals that are neither true nor false in I, i.e. $\mathcal{B}_\alpha \backslash \mathcal{B}_I \neq \emptyset$.

The following auxiliary procedure returns a, possibly empty, set of extensions of I that *decide* a given rule r, where $a \leftarrow \alpha$ is decided by I if either $a \in I$, in which case the rule is certainly satisfied (and possibly applied), or $\exists b \in \alpha \cdot I \cap \{\neg b, \textit{not}(b)\} \neq \emptyset$, i.e. the rule is not applicable.

```
set<set<extended literal>>
extend(set<extended literal> $I$, rule $r \equiv a \leftarrow \alpha$) {
if ($a \in I$) /* i.e. $a \in \psi(I)$ */
  return $\{I\}$;
if ($\exists b \in \alpha \cdot I \cap \{\neg b, not(b)\} \neq \emptyset$) /* $r$ ''blocked'' */
  return $\{I\}$;
if ($\alpha \subseteq I$) /* $r$ applicable */
  if ($I \cap \{\neg a, not(a)\} = \emptyset$) /* apply $r$ */
    return $\{I \cup \{a\}\}$;
  else /* $r$ cannot be applied */
    return $\emptyset$;
/* $r$ could become applied or not applicable */
result = $\{I \cup \alpha \cup \{a\}\}$ ; /* $r$ will be applied */
for each ($b \in \alpha \backslash I$)
  result = result $\cup \{I \cup \{not(b)\}\}$ ; /* $r$ will not be applicable */
return result;
}
```

Hence, when extending I to satisfy a rule r, one may demand that a certain literal l will not occur in the answer set by adding $not(l)$ to I. If, later on, $\neg l$ is added as well, normalization will remove $not(l)$.

The following algorithm can be used to compute preferred answer sets.

```
bool
aset(set<extended literal> $I$, set<rule> $P$, out set<literal> M) {
$R = \{r \in P \mid \psi(I) \not\models r\}$
if ($R = \emptyset$)
  if ($P^* = \psi(I)$)
    $M \leftarrow \psi(I)$ /* $\psi(I)$ is a preferred answer set of $P$ */
    return true;
  else
    return false;
/* $R \neq \emptyset$ */
choose minimal $r \in R$
for each $J \in$ extend($I, r$) { /* $J$ is a minimal
        consistent extension of $I$ that decides $r$ */
  if aset($\phi(J)$, $P$, $M$)
    return true;
  }
/* $\forall J \supset I \cdot \psi(J) \not\models r$, $r$ will be defeated */
return aset($I, P \backslash \{r\}$, $M$)
}
```

It is straightforward to show that the aset procedure satisfies the following proposition.

Theorem 12. *For P an OLP, I a set of extended literals such that $\psi(I)$ is founded, $aset(I, P, M)$ returns* **true** *iff there exists an extension $M \supseteq I$ such that $\psi(M)$ is a preferred answer set of P.*

From Theorem 12, it follows that $aset(\emptyset, P, M)$ will compute a preferred answer set M of P.

Of course, the above algorithm can be improved in many ways: e.g. the test whether $P^\star = \psi(I)$ can be performed incrementally by keeping two sets of (extended) literals: one for literals that already have a motivation and one for literals that still must be confirmed using a proper rule application. Also, rule satisfaction may look ahead to only consider feasible extensions (that may be supported by weaker rules).

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

It is straightforward to show that checking whether M is not a (proper) preferred answer set of an OLP P is in NP. Finding a (proper) 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 (proper) preferred answer set. Hence the following theorem.

Theorem 13. *The problem of deciding whether, given an arbitrary ordered program P and a literal a, whether a occurs in any proper preferred answer set of P is in Σ_2^P.*

To show that the problem of Theorem 13 is Σ_2^P-hard, we use 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 [2] for disjunctive logic programs.

The program P corresponding to ϕ is shown below where we have introduced new atoms w, u, $Y' = \{y' \mid y \in Y\}$ and for each $c \in C$, c' is obtained by replacing each occurrence of $y \in Y$ with the corresponding $y' \in Y'$.

$$\begin{array}{c}
\{\neg b \leftarrow \;\; b \leftarrow \mid b \in Y \cup Y'\} \cup \{\neg w \leftarrow\} \\
\hline
\{w \leftarrow c \mid c \in C\} \cup \{\neg u \leftarrow c' \mid c \in C\} \\
\hline
\{u \leftarrow \neg u\} \cup \{\neg a \leftarrow \;\; a \leftarrow \mid a \in X\} \\
\hline
w \leftarrow \neg w
\end{array}$$

It can then be shown that $\neg u \in M$ for some proper preferred answer set M iff ϕ is valid. Intuitively, if one succeeds in falsifying ϕ for some assignment to Y', $\neg u$ will not be in M and M will satisfy $u \leftarrow \neg u$, making it preferred over answer sets corresponding to other assignments of Y'.

Theorem 14. *The problem of deciding whether, 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-hard.*

Corollary 1. *The problem of deciding whether, 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.*

5 Repairing Databases Using Ordered Programs

We review some definitions from [1], using a simplified notation.

Definition 8. *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 9. *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*[6] *$\leq_D$ 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 10. *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 [1] where $\leq_D$ was defined based on the symmetric difference $\Delta(D, D') = (D^+ \setminus D'^+) \cup (D'^+ \setminus D^+)$ rather than our $\Delta_D(D')$. However, it is straightforward to show that both definitions lead to the same $\leq_D$, i.e. $\Delta_D(D_1) \subseteq \Delta_D(D_2)$ iff $\Delta(D, D_1) \subseteq \Delta(D, D_2)$ and thus Definition 10 is equivalent to the one in [1].

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 answer sets. Using ordered logic instead of logic programs with exceptions [5] greatly simplifies (w.r.t. [1]) 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 [1] is limited to constraints containing up to two literals.

Definition 11. *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.*

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

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.

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

Theorem 15. *Let D be a database and let C be a consistent set of constraints with $\mathcal{B}_C \subseteq \mathcal{B}_D$. A database R is a repair of D w.r.t. C iff R is a proper preferred answer set of $P(D, C)$.*

Example 7. Consider the propositional version of the example from [1] 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.

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

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 $P(D, C)$.

References

1. Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. Specifying and querying database repairs using logic programs with exceptions. In *Proceedings of the 4th International Conference on Flexible Query Answering Systems*, pages 27–41, Warsaw, Octobre 2000. Springer-Verlag.
2. T. Eiter and G. Gottlob. Complexity results for disjunctive logic programming and application to nonmonotnic logics. In *Proceedings of the 1983 International Logic Programming Symposium*, pages 266–279, Vancouver, October 1993. MIT Press.
3. Thomas Eiter, Georg Gottlob, and Nicola Leone. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1-2):129–177, 1997.
4. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, Seattle, Washington, August 1988. The MIT Press.
5. Robert A. Kowalski and Fariba Sadri. Logic programs with exceptions. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 598–613, Jerusalem, 1990. The MIT Press.
6. Els Laenens and Dirk Vermeir. Assumption-free semantics for ordered logic programs: On the relationship between well-founded and stable partial models. *Journal of Logic and Computation*, 2(2):133–172, 1992.
7. Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artifi cial Intelligence*, to appear.
8. Raymond Reiter. A theory of diagnosis from first principles. *Artifi cial Intelligence*, 32(1):57–95, 1987.
9. Chiaki Sakama and Katsumi Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13(1):145–172, 1994.
10. M.H. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery*, 23(4):733–742, 1976.
11. M. De Vos and D. Vermeir. Semantic forcing in disjunctive logic programs. *Computational Intelligence*, 17(4):651–684, 2001.