Alternating Fixpoint Theory for Logic Programs with Priority

Kewen Wang¹, Lizhu Zhou¹, and Fangzhen Lin²

¹ Department of Computer Science and Technology
Tsinghua University
Beijing, 100084, China
{kewen,dcxzlk}@tsinghua.edu.cn
² Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
flin@cs.ust.hk

Abstract. van Gelder’s alternating fixpoint theory has proven to be a very useful tool for unifying and characterizing various semantics for logic programs without priority. In this paper we propose an extension of van Gelder’s alternating fixpoint theory and show that it can be used as a general semantic framework for logic programs with priority. Specifically, we define three declarative and model-theoretic semantics in this framework for prioritized logic programs: prioritized answer sets, prioritized regular extensions and prioritized well-founded model. We show that all of these semantics are natural generalizations of the corresponding semantics for logic programs without priority. We also show that these semantics have some other desirable properties. In particular, they can handle conflicts caused indirectly by the priorities.

Keywords: logic programs; alternating fixpoints, priority, answer sets, well-founded model.

1 Introduction

Priorities play an important role in logic programming, and they arise in various applications for various purposes. For example, in inheritance hierarchies, it is generally assumed that more specific rules has precedence over less specific ones, and the exact axiomatization of this intuition has been attempted and investigated extensively by researchers in default reasoning. Other application domains include reasoning about actions and causality, where causal effect rules are considered to be preferred over inertia rules [4], and legal reasoning and diagnosis.

There have been some proposals for axiomatizing prioritized logic programs (for example, [1, 5, 6, 8, 12, 14, 15, 19, 28]), but as pointed out in [6], they are far
from satisfactory. In particular, they cannot handle indirect conflicts. Consider the following program $P$:

$$
R_1: \ p \leftarrow q_1 \\
R_2: \ \neg p \leftarrow q_2 \\
R_3: \ q_1 \leftarrow w_1 \\
R_4: \ q_2 \leftarrow w_2
$$

If a priority between $R_1$ and $R_2$ is specified, the situation is simple enough to be dealt with by many existing approaches. However, in many cases, we are only informed a priority between $R_3$ and $R_4$, say $R_3 \prec R_4$. Intuitively, $R_1$ has precedence over $R_2$, implicitly, and $p$ should be derived from $P$ rather than $\neg p$. Unfortunately, most of existing semantics for prioritized logic programs are unable to represent such domains.

In this paper, we shall propose an extension to van Gelder’s alternative fixed point theory and use it as a uniform semantic framework to give three different semantics to logic programs with priorities. These three semantics have their correspondences for logic programs without priorities, and capture different intuitions about logic programs and are tailored for different application needs.

This paper is organized as follows. Motivated by van Gelder’s alternating fixpoint theory [23] and the semi-constructive definition of extensions in default logic [3, 21], in section 3 we develop a semantic framework for prioritized logic programs, in which various semantics can be defined. In particular, we defined three semantics prioritized answer sets, prioritized regular extensions and prioritized well-founded model in section 4 and some demonstrating examples are given. In section 5, we prove some important semantic properties to justify the suitability of our semantics. We prove that these three semantics generalize the traditional well-known answer set semantics, regular extension semantics and well-founded semantics, respectively. The relation of our approach to some other semantics is compared in section 6. Section 7 is our conclusion.

## 2 Alternating Fixpoints without Priority

In this section, we briefly review the alternating fixpoint theory in [23] and related definitions.

An extended logic program $P$ is a finite set of rules of the form

$$
R: \ l \leftarrow l_1, \ldots, l_r, \sim l_{r+1}, \ldots, \sim l_m
$$

where $l$s with or without subscripts are literals, the symbol $\sim$ is default negation and the symbol $\neg$ is explicit negation. A literal is either an atom $a$ or its explicit negation $\neg a$. The set of literals of $P$ is denoted $\text{Lit}$.

A rule $R$ of an extended logic program is also expressed as $\text{head}(R) \leftarrow \text{pos}(R), \sim \text{neg}(R)$, where $\text{head}(R) = l$, $\text{pos}(R) = \{l_1, \ldots, l_r\}$ and $\text{neg}(R) = \{l_{r+1}, \ldots, l_m\}$.

We assume all logic programs are propositional and each rule $R$ is automatically interpreted into its “semi-normal” rule $\text{head}(R) \leftarrow \text{pos}(R), \sim \text{neg}(R), \sim \neg \text{head}(R)$. The reason will be explained later on.
The alternating fixpoint theory, introduced by van Gelder [23], is proven to be a very useful tool to unify and characterize different semantic intuitions for logic programs (without priority). This theory is based on an immediate consequence mapping [9].

Let $P$ be an extended logic program and $S$ a set of literals. $S$ is logically closed if it is consistent or is Lit.

The GL-transformation $P^S$ of $P$ with respect to $S$ is the logic program (without default negation) $P^S = \{\text{head}(R) \leftarrow \text{pos}(R) \mid R \in P, \neg \text{neg}(R) \cap S = \emptyset\}$. The set $C_P(S)$ of consequences of $P^S$ is the smallest set of literals which is both logically closed and closed under the rules of $P^S$.

For any set $S$ of literals, $GR(P, S) = \{R \in P \mid \text{pos}(R) \subseteq S, \neg \text{neg}(R) \cap S = \emptyset\}$ is said to be the generating set of $S$ in $P$.

Notice that, if $C_P(S)$ is consistent, then $C_P(S) = T_P S \uparrow \omega$, where $T_P$ is the immediate consequence operator of $P^S$ by considering each negative literal as a new atom. $C_P(S)$ is anti-monotonic, i.e., $C_P(S_1) \subseteq C_P(S_2)$ whenever $S_2 \subseteq S_1$.

The alternating operator $A_P(S) = C_P^2(S)$ is defined through the immediate consequences $C_P(S)$. It is known that $A_P$ is a monotonic operator. A fixpoint of $A_P$ is said to be an alternating fixpoint of $P$. An alternating fixpoint $S$ is normal if $S \subseteq C_P(S)$.

By the alternating fixpoint theory, many semantics for logic programs can be defined including the following three ones [23, 26]:

1. The well-founded model: the least alternating fixpoint.
2. The regular extensions: the maximal normal alternating fixpoints.
3. The answer sets: a special kind of the maximal normal alternating fixpoints (namely, $C_P(S) = S$).

3 Alternating Fixpoints with Priority

As noted in section 2, the existing alternating fixpoint approach considers only logic programs without priority. In this section, we will first define an intuitive generalization of the immediate consequence mapping for extended logic programs with priority, and then establish the corresponding alternating fixpoint theory.

Let $P$ be an extended logic program and $\prec$ an irreflexive and transitive binary relation on rules of $P$. Then the pair $P = (P, \prec)$ is said to be a prioritized logic program. By $R_1 \prec R_2$ we mean that $R_1$ has precedence over $R_2$.

Let $S_1$ and $S_2$ be two sets of literals in a prioritized logic program $P$. A rule $R$ in $P$ is active with respect to the pair $(S_1, S_2)$ if $\text{pos}(R) \subseteq S_1$ and $\neg \text{neg}(R) \cap S_2 = \emptyset$. In particular, if $S_1 = S_2 = S$, then $R$ is active with respect to $(S, S)$ if and only if the body of $R$ is satisfied by $S$ (in the usual sense).

For two rules $R_1$ and $R_2$ such that $R_1 \prec R_2$ (i.e., $R_1$ has precedence over $R_2$), there are often two kinds of existing approaches to represent this preference relation. One is to reflect that $R_2$ will not be applied provided that $R_1$ is applicable. The other is to reflect that the rule that has higher priority is first
applied: (1) if both $R_1$ and $R_2$ are applicable, $R_1$ is first applied. (2) if $R_1$ has been applied and $R_2$ is applicable, then $R_2$ can still be applied.

As argued by Delgrande and Schaub in [7], the second kind of approaches may be more general and thus can be used in wider application domains. The following definition is just designed to reflect the second intuition.

**Definition 1.** Let $\mathcal{P} = (P, \prec)$ be a prioritized logic program and $S$ be a set of literals. Set

$$S_0 = \emptyset,$$

$$S_{i+1} = S_i \cup \{l \mid \text{there exists a rule } R \text{ of } P \text{ such that (1) } \text{head}(R) = l \text{ and } R \text{ is active with respect to } (S_i, S) \text{ and, (2) no rule } R' \prec R \text{ is active with respect to } (S, S_i) \text{ and } \text{head}(R') \notin S_i\}.$$ 

Then the reduct of $\mathcal{P}$ with respect to $S$ is defined as the set of literals $C_\mathcal{P}(S) = \cup_{i \geq 0} S_i$ if $\cup_{i \geq 0} S_i$ is consistent. Otherwise, $C_\mathcal{P}(S) = \text{Lit}$.

The rule $R$ is accepted by stage $i + 1$ with respect to $S$ if $R$ satisfies the above two conditions in the definition of $S_{i+1}$. The sequence $S_0, S_1, \ldots, S_i, \ldots$ will be called the $\prec$-sequence of $S$ in $\mathcal{P}$.

We may note that $C_\mathcal{P}$ and $C_\mathcal{P}$ are two quite different operators. This abusing of notions should cause no confusion in understanding this paper. The main difference of our definition from van Gelder’s approach is that we obtain the consequences from $\mathcal{P}$ and $S$ directly. The traditional approaches (i.e. without considering priority) first obtain a positive program from $\mathcal{P}$ with respect to $S$ and then the consequences are derived from this positive program.

Another point on the above definition that should be addressed is why we do not replace the pair $(S, S_i)$ by the pair $(S_i, S)$ in the condition (2) of the definition of $S_{i+1}$. The reason for this can be explained by the following example.

**Example 1.** Let $\mathcal{P}$ be a prioritized logic program as follows:

$$R_1 \quad p \leftarrow q$$
$$R_2 \quad \neg p \leftarrow q'$$
$$R_3 \quad q' \leftarrow q'$$
$$R_4 \quad q \leftarrow$$
$$R_5 \quad q' \leftarrow$$

Here $R_2 \prec R_1 \prec R_3$. We should not infer $p$ since $R_2 \prec R_1$. Though $R_2 \prec R_1$, we can not infer $p$ because $R_2$ depends on the rule $R_3$ and $R_1 \prec R_3$.

This means that we should assign both $p$ and $\neg p$ the truth value ‘undefined’.

However, if we replace $(S, S_i)$ by $(S_i, S)$ in Definition 1, then $p$ is inferred. As a semantics for general purpose, we do not expect such a conclusion, though there also exist some domains that need a more credulous interpretation. Thus, we adopt a skeptical approach in our Definition 1 to treat the conflicts among rules.

The transformation $C_\mathcal{P}$ is not a monotonic operator in general, but we can prove its anti-monotonicity.
Lemma 1. \( C_P \) is an anti-monotonic operator. That is, for any two sets \( S \) and \( S' \) of literals such that \( S \subseteq S' \), \( C_P(S') \subseteq C_P(S) \).

Proof. If \( S \subseteq S' \), it suffices to prove that, for any number \( i \geq 0 \),
\[
S_i' \subseteq S_i, \tag{\star}
\]
where \( S_i \) and \( S_i' \) are defined as Definition 1.

The proof of (\star) is a direct induction on \( i \) and thus, we omit it here.

Definition 2. Let \( \mathcal{P} = (P, \prec) \) be a prioritized logic program. The alternating transformation of \( \mathcal{P} \) is defined as, for any set \( S \) of literals,
\[
A_{\mathcal{P}}(S) = C_P(C_P(S)).
\]

Proposition 1. The operator \( A_{\mathcal{P}} \) is monotonic and thus possesses the least fixpoint.

Proof. It follows from Lemma 1 and Tarski’s Theorem.

- A fixpoint of \( A_{\mathcal{P}} \) is called an alternating fixpoint of \( \mathcal{P} \).

By now, we establish the basic semantic framework for prioritized logic programs by using alternating fixpoint approach, in which a semantics of prioritized logic programs can be defined as a subset of the alternating fixpoints.

4 Semantics for Prioritized Logic Programs

In this section we will define three semantics in the framework established in section 3: prioritized answer sets, prioritized regular extensions and prioritized well-founded model. Like the traditional answer set semantics and well-founded model without priority, our new semantics are also to represent two intuitions in AI, i.e. maximalism and minimalism. It should be noted that we will omit the adjective ‘prioritized’ or add an adjective ‘unprioritized’ when we mention a semantics for logic programs without priority.

To characterize credulous reasoning, it is natural to choose all the maximal alternating fixpoints as the intended models of a prioritized logic program. If \( S \) is a maximal alternating fixpoint of prioritized program \( \mathcal{P} = (P, \prec) \), it may be the case that \( S \not\subseteq C_P(S) \). This case is not what we want since every literal in \( S \) should be derived from \( \mathcal{P} \) with respect to \( S \) for an intended model \( S \).

Definition 3. Let \( \mathcal{P} = (P, \prec) \) be a prioritized logic program. \( S \) is a prioritized regular extension of \( \mathcal{P} \) if it is a maximal normal alternating fixpoint of \( \mathcal{P} \): for any normal alternating fixpoint \( S' \) of \( \mathcal{P} \) such that \( S \subseteq S' \), \( S' = S \).

The prioritized semantics \( \text{PRE} \) (i.e. prioritized regular extension semantics) for \( \mathcal{P} \) is defined as the set of its prioritized regular extensions.

This definition has the same form as the characterization of the regular extensions without priority in [26].
Example 2. Consider the following prioritized logic program:

\[
\begin{align*}
R_1 &: \quad q \leftarrow \sim p \\
R_2 &: \quad w \leftarrow \sim w
\end{align*}
\]

The rule \(R_1\) has precedence over \(R_2\): \(R_1 \prec R_2\). This prioritized program has the unique prioritized regular extension \(S = \{q\}\). Notice that \(S\) is not a fixpoint of \(C_P\) since \(C_P(S) = \{w, q\}\).

Example 3. [3] If we have a knowledge base as follows:

\[
\begin{align*}
R'_1 &: \quad \neg Fly(x) \leftarrow Peguin(x), \sim Fly(x) \\
R'_2 &: \quad Winged(x) \leftarrow Bird(x), \sim Winged(x) \\
R'_3 &: \quad Fly(x) \leftarrow Winged(x), \sim \neg Fly(x) \\
R'_4 &: \quad Bird(x) \leftarrow Peguin(x) \\
R'_5 &: \quad Peguin(Tweety) \leftarrow
\end{align*}
\]

By the principle of specificity, \(R'_1 \prec R'_2\).

The difference of this example from the classical Bird-Fly example is that there is no explicit priority between \(R'_1\) and \(R'_3\) but there is an implicitly specified priority between \(R'_1\) and \(R'_5\): \(R'_1\) has precedence over \(R'_3\) since \(R'_3\) depends on \(R'_2\) and \(R'_1 \prec R'_2\).

Intuitively, a suitable semantics for this knowledge base should infer \(\neg Fly(Tweety)\) and \(Winged(Tweety)\). In particular, any semantics suitable should not contain the set \(\{Peguin(Tweety), Bird(Tweety), Fly(Tweety), Winged(Tweety)\}\) as a model.

For simplicity, we rewrite the above rules as the following prioritized logic program \(P = (P, \prec)\):

\[
\begin{align*}
R_1 &: \quad \neg Fly \leftarrow Peguin, \sim Fly \\
R_2 &: \quad Winged \leftarrow Bird, \sim \neg Winged \\
R_3 &: \quad Fly \leftarrow Winged, \sim \neg Fly \\
R_4 &: \quad Bird \leftarrow Peguin \\
R_5 &: \quad Peguin \leftarrow \\
\end{align*}
\]

Let \(S = \{Peguin, Bird, \neg Fly, Winged\}\) and \(S' = \{Peguin, Bird, Fly, Winged\}\). It can be verified that \(S\) is the unique prioritized regular extension but \(S'\) is not. However, the prioritized semantics in [6] and [28] admit \(S'\) as an intended model.

Another interesting credulous semantics for \(P\) is defined by the set of all fixpoints of \(C_P\) (this set is also a set of alternating fixpoints).

Definition 4. \(S\) is said to be a prioritized answer set of \(P\) if \(S\) is a fixpoint of \(C_P: C_P(S) = S\).

The semantics PAS (i.e., prioritized answer set semantics) of \(P\) is defined by the set of all prioritized answer sets of \(P\).
In Example 3, \( S = \{\text{Penguin, Bird, } \neg \text{Fly, Winged}\} \) is also a prioritized answer set of \( \mathcal{P} \). In general, each prioritized answer set is a prioritized regular extension. But a prioritized regular extension may not be a prioritized answer set as Example 2 has shown.

**Proposition 2.** If \( S \) is a prioritized answer set of \( \mathcal{P} \), then \( S \) is also a prioritized regular extension of \( \mathcal{P} \).

However, it should be noted that our definition of prioritized answer sets cannot ‘tolerate’ too unreasonable ordering. For instance, if \( \mathcal{P} \) has three rules \( R_1 : q \leftarrow w \), \( R_2 : p \leftarrow v \) and \( R_3 : w \leftarrow \). Suppose \( R_1 \prec R_2 \prec R_3 \), then, intuitively, \( R_1 \) should not have precedence over \( R_3 \) since \( R_1 \) depends on \( R_3 \). It is not hard to verify that \( \mathcal{P} = (\mathcal{P}, \prec) \) has no prioritized answer set though \( \mathcal{P} \) has the unique answer set \( \{ w, p, q \} \).

The third semantics that will be defined in our semantic framework is named the prioritized well-founded model, which is to characterize skeptical reasoning in artificial intelligence as the traditional well-founded semantics does.

**Definition 5.** The prioritized well-founded model of \( \mathcal{P} = (\mathcal{P}, \prec) \) is defined as the least alternating fixpoint of \( \mathcal{P} = (\mathcal{P}, \prec) \).

The semantics \( \text{PWF} \) (i.e. prioritized well-founded semantics) of \( \mathcal{P} \) is defined by the prioritized well-founded model of \( \mathcal{P} \).

**Proposition 3.** Every prioritized logic program has the unique prioritized well-founded model.

**Proof.** It follows directly from Proposition 1.

This proposition shows that our prioritized well-founded semantics also possesses an important semantic property: *completeness*.

One often criticized deficiency of the well-founded model without priority is that it is too skeptical that, in many cases, nothing useful can be derived from programs under the well-founded semantics. In certain degree, our prioritized well-founded semantics may overcome this problem as the following two examples demonstrate. This will also be shown theoretically in the next section.

**Example 4.**

\[
\begin{align*}
R_1 & : p \leftarrow \sim q \\
R_2 & : q \leftarrow \sim p
\end{align*}
\]

\( R_1 \prec R_2 \). Then the prioritized well-founded model of \( \mathcal{P} = (\mathcal{P}, \prec) \) is \( \{ p \} \). Notice that the well-founded model of \( \mathcal{P} \) is \( \emptyset \).

The next example further shows how to resolve conflicts with the prioritized well-founded model.
Example 5. Imagine such a scenario: when a train is approaching a bridge, the robot driver is told that a bomb may be put under the bridge and thus, he stops the train. He is also told that he cannot pass the bridge if this reported bomb is not found. Afterwards, he is told again that enough evidence proves that there is no such a bomb under the bridge. According to our commonsense, at this moment, the train can pass the bridge. This knowledge can be represented as an extended logic program $P$ as follows:

$R_1 : \neg \text{pass} \leftarrow \neg \text{bombFound}, \neg \text{pass}$

$R_2 : \text{pass} \leftarrow \neg \text{bomb}, \neg \text{pass}$

$R_3 : \neg \text{bombFound} \leftarrow \neg \text{bomb}$

$R_4 : \neg \text{bomb} \leftarrow$

$R_2 \prec R_1$ because the rule $R_2$ is newer than $R_1$.

It can be verified that the prioritized well-founded model of $P = (P, \prec)$ is $\{\neg \text{bomb}, \neg \text{bombFound}, \text{pass}\}$, which is just our intuition on this program. But the ordinary well-founded semantics can not say ‘yes’ or ‘no’ about ‘pass’. Notice that $\neg \text{pass}$ cannot be added to $S_1$ since $R_2 \prec R_1$ and $R_2$ is active with respect to $(S, S_0)$ (though $R_1$ is active with respect to $(S, S_0)$).

As mentioned before, if both rules $R_1$ and $R_2$ appear in $P$ and their heads are complementary literals, then we understand these rules as their corresponding semi-normal counterparts. For example, if $P$ consists of two rules $R_1 : p \leftarrow \neg q$ and $R_2 : \neg p \leftarrow \neg q$, and let $R_1 \prec R_2$. Then $P$ is actually understood as $\{p \leftarrow q, \neg q; \neg p \leftarrow q, p\}$. Under this assumption, the prioritized program $P$ has the unique intended model $\{p\}$. Otherwise, $P$ will be inconsistent.

5 Properties of Prioritized Semantics

Currently there are many different proposals about the semantics of logic programs with priorities ([5, 6, 8, 14, 15, 19, 28]). It is too early to say which one will eventually prevail. In the meantime, it may be useful to look at reasonable “postulates” that a sound semantics should satisfy. Recently, Brewka and Eiter [6] proposed two such postulates:

$P1$. Let $B_1$ and $B_2$ be two belief sets of a prioritized theory $(T, <)$ generated by the (ground) rules $R \cup \{d_1\}$ and $R \cup \{d_2\}$, where $d_1, d_2 \notin R$, respectively. If $d_1$ is preferred over $d_2$, then $B_2$ is not an intended belief set of $T$.

$P2$. Let $B$ be an intended belief set of a prioritized theory $(T, <)$ and $r$ a (ground) rule such that at least one prerequisite of $r$ is not in $B$. Then $B$ is an intended belief set of $(T \cup \{r\}, <')$ whenever $<'$ agrees with $<$ on priorities among rules in $T$.

Unfortunately, many of the existing prioritized semantics do not satisfy their postulates as pointed out by Brewka and Eiter [6], and it is not clear whether the fault is with most of the current semantics or that their postulates are too strict. However, at least the following example shows that $P1$ is not so intuitive.
Let \((P, \prec)\) be the following logic programs with \(r_1 < d_1 < d_2:\)

\[
\begin{align*}
r_1 & \quad \text{bird} \leftarrow \\
d_1 & \quad \neg \text{fly} \leftarrow \text{peguin} \\
d_2 & \quad \text{fly} \leftarrow \neg \text{peguin}, \text{bird}
\end{align*}
\]

We observe that: (1) \(\neg \text{peguin}\) and \(\text{bird}\) should be included in the intended belief set; and (2) the rule \(d_1\) is defeated by \(r_1\); and thus (3) \(d_2\) could be used to derive further beliefs. Accordingly, the intended belief set should be \(B = \{\text{bird}, \neg \text{peguin}, \text{fly}\}\). But, if we take \(R = \{r_1\}\), \(P1\) does not allow \(d_2\) is used in the case that \(d_1\) is defeated.

In this paper, we are going to play on the safe side and consider some relationships between semantics for logic programs with priorities and those without. In this direction, we notice that for logic programs without priorities, there are some well understood semantics such as the well-founded model and answer set semantics. It seems to us reasonable then that proposed new semantics for logic programs with priorities should be an extension of one of the semantics for logic programs without priorities. Since, as is well-known, all major semantics for logic programs without priorities agree on logic programs that are stratified, this means that for any stratified logic program, if the priorities associated with the rules are consistent with the stratification, then any semantics for this prioritized logic program should agree as well. In particular, they should agree with the perfect model semantics, which is indeed the case for our three semantics, as we shall show in this section. It should be noted that many semantics for prioritized logic programs are only defined for total orderings and thus, these semantics do not possess the above mentioned properties.

We start with the following lemma which will be very crucial in proving the results of this section. In the following, we assume that all programs are finite propositional programs.

**Lemma 2.** Let \(S\) be a set of literals in \(P = (P, \prec)\). Then

1. \(C_P(S) \subseteq C_P(S)\).
2. if \(S \subseteq C_P(S)\), then \(C_P(S) = C_P(S)\).
3. if \(\prec\) is empty, then \(C_P(S) = C_P(S)\).

**Proposition 4.** For any prioritized logic program \(P = (P, \prec)\), each prioritized regular extension of \(P\) is contained in a regular extension of \(P\). In particular, if the relation \(\prec\) is empty, then \(S\) is a prioritized regular extension of \(P\) if and only if \(S\) is a regular extension of \(P\).

This proposition reveals two connections between prioritized regular extension and unprioritized regular extension: (1) the adding of some priority makes the information represented by the logic program without priority more concrete (the number of ‘models’ and the number of elements in each ‘model’ are all reduced in general), (2) when the priority relation is empty (i.e. there is no preferences among rules), the prioritized regular extension semantics is the same
as the unprioritized regular extension semantics. These two properties seem to be natural.

The next proposition convinces that our prioritized answer sets generalize Gelfond and Lifschitz’s answer sets [10].

**Proposition 5.** A prioritized answer set of \( P = (P, \prec) \) is also an answer set of \( P \). In particular, if \( \prec \) is empty, \( S \) is a prioritized answer set of \( P = (P, \prec) \) iff \( S \) is an answer set of \( P \).

Like the well-founded model without priority, the prioritized well-founded model also possesses the property of completeness.

**Proposition 6.** Every prioritized logic program has a prioritized well-founded model.

*Proof.* It follows directly from Proposition 1.

The relationship between the well-founded model and the prioritized well-founded model can be stated as follows.

**Proposition 7.** Let \( M_p \) be the prioritized well-founded model of \( P = (P, \prec) \) and \( M \) the well-founded model of \( P \). Then \( M \subseteq M_p \). In particular, if \( \prec = \emptyset \), then \( M = M_p \).

*Proof.* If \( M_p = \text{Lit} \), then \( M = \text{Lit} \), the conclusion is obvious. We need only consider the case when \( M_p \) is consistent. Since \( M_p \) is a normal alternating fixpoint of \( P \), it follows that \( M_p \) is a normal alternating fixpoint of \( P \) by Lemma 2. Since the well-founded model \( M \) is the least alternating fixpoint of \( P \), we have \( M \subseteq M_p \).

The last part is obvious from Lemma 2.

Proposition 7 and Example 4 show that the prioritized well-founded semantics is less skeptical than the traditional well-founded semantics in general. This proposition also makes it possible to overcome the drawback (i.e. too skeptical) of the traditional well-founded model by adding preferences to rules of logic programs.

In the rest of this section, we study the relation of our three prioritized semantics to the perfect model [2, 20]. The perfect model semantics of stratified logic programs has already been well-accepted. Moreover, a stratification of a stratified program \( P \) actually determines a priority on rules of the program and thus a prioritized logic program \( P \) is obtained. Intuitively, a suitable semantics for priority should be consistent with the perfect model. Namely, \( P \) should have the unique `model’ \( M \) and \( M \) is exactly the perfect model of \( P \) for any stratified logic program.

The following proposition will convince that the prioritized regular extensions, prioritized answer sets and prioritized well-founded model exactly reflect the semantic intuition above.
Definition 6. A logic program \( P \) (without explicit negation) is said to be stratified if \( P \) has a partition (i.e., stratification) \( P = P_1 \cup \cdots \cup P_t \) such that the following conditions are satisfied:

1. \( P_i \cap P_j = \emptyset \) for \( i \neq j \).
2. If a rule \( R \) is in \( P_i \), then the atoms in \( \text{pos}(R) \) can appear only in \( \cup_{j=1}^{i-1} P_j \)
   and the atoms in \( \text{neg}(R) \) can appear only in \( \cup_{j=i+1}^{t} P_j \).

We recall that \( \text{pos}(R) \) is the set of atoms that appear positively in the body of \( R \); \( \text{neg}(R) \) is the set of atoms that appear negatively in the body of \( R \).

The perfect model of the stratified logic program \( P \) is recursively defined as follows.

\[
\begin{align*}
\forall i & \quad P_1 := P_1 \text{ and } M_1 := T_{P_1} \uparrow \omega. \\
\forall i & \quad P_{i+1} := \{p \leftarrow | p \in M_i \} \cup \{\text{head}(R) \leftarrow \text{pos}(R) | R \in P_{i+1}, \text{neg}(R) \cap M_i = \emptyset\}, \\
& \quad \text{and } M_{i+1} := T_{P'_{i+1}} \uparrow \omega.
\end{align*}
\]

Let \( P \) be a stratified logic program and \( P = P_1 \cup \cdots \cup P_t \) a stratification of \( P \). A natural priority relation \( \prec_s \) on \( P \) can be defined as:

- for any \( R_1 \) and \( R_2 \) in \( P \), \( R_1 \prec_s R_2 \) if and only if \( R_1 \in P_i \) and \( R_2 \in P_j \) such that \( i < j \).

Thus, we obtain a prioritized logic program \( \mathcal{P} = (P, \prec_s) \) for any given stratified logic program \( P \) and a stratification.

Proposition 8. Let the prioritized logic program \( \mathcal{P} = (P, \prec_s) \) be defined as above and the perfect model of \( P \) is \( M_t \). Then

1. \( \mathcal{P} \) has the unique prioritized answer set \( M_t \).
2. \( \mathcal{P} \) has the unique prioritized regular extension \( M_t \).
3. \( \mathcal{P} \) has the unique prioritized well-founded model \( M_t \).

Proposition 5 states that each prioritized answer set of a prioritized logic program \( \mathcal{P} = (P, \prec) \) is also an answer set of \( P \). In turn, we will show that, for each answer set \( S \) of a logic program \( P \), there is an ordering \( \prec \) on rules of \( P \) such that \( S \) is the unique prioritized answer set of \( \mathcal{P} = (P, \prec) \).

Proposition 9. Let \( P \) be a logic program and \( S \) an answer set of \( P \). Then there is a well-order \( \prec \) such that \( S \) is the unique prioritized answer set of the prioritized program \( \mathcal{P} = (P, \prec) \).

As noted previously, a totally ordered \( \mathcal{P} \) may have no prioritized answer set. But we will prove that, when the rules in \( P \) are totally ordered, \( \mathcal{P} = (P, \prec) \) has at most one prioritized answer set.

Proposition 10. Let \( \prec \) be a total ordering on the rules of logic program \( P \). If \( \mathcal{P} = (P, \prec) \) has prioritized answer set, then it has the unique one.

The two results above illustrate that, for any prioritized program \( \mathcal{P} = (P, \prec) \) such that \( \prec \) is a total ordering, its prioritized answer set semantics \( \text{PAS}(P) \) always outputs either nothing or the unique prioritized answer set (i.e., an answer set of \( P \)).
6 Comparison to Related Approaches

Several approaches treating priorities in the setting of logic programming have been described in the literature. In this section, we summarize their relationships to our approach.

Both preferred answer sets in [6] and prioritized answer sets in [28] extend Gelfond and Lifschitz's answer sets to handle rules with priority. Like our approach, they assume a priority among rules of logic program. The basic idea behind their approaches (though very different) is to transform a prioritized logic program $P$ into a unprioritized program and the answer sets of $P$ are defined through the GL-answer sets of the obtained program. As shown in Example 3, these semantics cannot correctly resolve conflicts caused by indirectly or implicitly specified priorities.

Another interesting definition of preferred answer sets was defined in [22]. This approach assumes a preference among the set of atoms of program and bears a similar idea as the perfect model of stratified programs [2, 20].

Brewka's preferred well-founded model in [5] extends the well-founded model to treat logic programs with priority. His approach is different from our prioritized well-founded model in at least two aspects: (1) he represents priority in object level; (2) given a set of literals $S$, he first gradually extends the empty set to a set of rules and then obtains a set of literals as the transformation of $S$. But we directly extend the empty set and obtain the transformation of $S$. Though the precise relationship between Brewka's and ours is not clear, we still believe these two semantics have some close connection.

Two argumentation-based semantics for prioritized logic programs are defined in [27, 19]. These approaches are also semantic frameworks for logic programs with priority. Clearly, their intuition is quite different from ours and they also obtain different semantics from ours. The approach in [27] is to characterize the first kind of semantic intuition mentioned in section 1. For example, let $P$ consists of the following rules: $R1 : \text{bird} \leftarrow \text{peguin}$ and $R2 : \text{peguin} \leftarrow$. If $R1$ has precedence over $R2$, our semantics allows to derive $\text{peguin}$ and $\text{bird}$. But theirs allows to derive only $\text{peguin}$. The difference between the semantics in [19] and our approach can be illustrated by Example 2. According to our and Brewka's approaches, $P$ has the unique model $\{p\}$, but under Prakken and Sartor's semantics, the priority between $R1 : p \leftarrow q$ and $R2 : q \leftarrow p$ has not effect on the reasoning in $P$. Namely, this prioritized program has two models $\{p\}$ and $\{q\}$ under their semantics.

Both our approach and BH-prioritized default logic [3] employ the semi-constructive definition of default extensions [21] and our prioritized answer sets correspond to the BH-extensions. But they are different semantics as shown in the following example.

Example 6. Let $P$ consist of the following rules:

$$R1 : \quad p \leftarrow w, \sim q, \sim \neg p$$
$$R2 : \quad \neg p \leftarrow \sim q, \sim p$$
$$R3 : \quad w \leftarrow$$
Here, $R_1 \prec R_2$.

It can be verified that $P = (P, \prec)$ has the unique prioritized answer set $S = \{w, p\}$. This set should be the intended semantics. But if we regard $P$ as a set of defaults and take $W = \emptyset$, then the prioritized default theory $(P, W, \prec)$ has the unique BH-extension $S'$ and $S'$ contains $\neg p$ rather than $p$.

Marek, Nerode and Remmel [15] propose a bottom-up procedure for computing the answer sets of a logic program $P$ when the rules of $P$ are totally ordered. For each stable model $S$ of $P$, there is a total ordering on $P$ such that their procedure outputs $S$. This procedure is not sound with respect to our prioritized answer sets.

Example 7. Let $P$ consist of the following rules:

\begin{align*}
R_1 & : q \leftarrow \sim p \\
R_2 & : w \leftarrow v \\
R_3 & : p \leftarrow w
\end{align*}

Here, $R_1 \prec R_2 \prec R_3$. Then the procedure proposed in [14] outputs the set $\{q\}$ which is not a stable model of $P$. Under our semantics, $P$ has the unique prioritized answer set $\{w, p\}$.

There are also some other approaches about treating priority in default logic and logic programming, such as [7, 8, 11], we will not discuss them here.

7 Conclusion

We have proposed a framework for studying logic programs with priorities based on van Gelder’s alternating fixpoint theory for logic programs without priorities. To illustrate the usefulness of this framework, we have proposed three different semantics: PAS (the prioritized answer sets), PWF (the prioritized well-founded model) and PRE (prioritized regular extensions), all of which have counterparts for logic programs without priorities, and have some additional properties. We believe that this framework is simple and intuitive. In the full version of this paper (see [25]), we shall show that the semantics defined by our prioritized answer sets can also be used to represent defeasible causal theories. Hopefully, this theory might provide a unifying framework for different semantics with priority. Currently, we are also working on extending the augmentation-theoretic framework in [24] to disjunctive logic programs with priority.

Recently, Marek, Truszczynski [16] and Niemelä [18] discussed the stable model semantics (answer set semantics) as the foundation of a computational logic programming system (i.e. SLP). This system differs from standard logic programming systems in several aspects including the following two important points:

- In the SLP, each program is assigned a collection of intended models rather than a single model.
In the SLP, the rules of a program are interpreted as constraints on objects to be computed.

Similar to the methods of solving constraint satisfaction problems, there are two steps to develop an SLP program for a given application domain: (1) to specify an SLP program whose set of answer sets encodes the general domain of candidate objects. (2) to add to this program more rules representing constraints that must be enforced.

As shown in [6, 18], many constraint satisfaction problems can be solved in SLP. However, if the preferences among rules are also enforced as constraints to the program obtained in the first step, the task of representing and solving application domains will most probably become simpler and more powerful. Thus, the semantics with priority may provide a suitable framework for SLP.

Acknowledgments

We would like to thank the anonymous referees for their useful comments. This work was supported in part by the Natural Science Foundation of China (No. 69883008, 69773027), the National Foundation Research Programme of China (No. G1999032704), and the IT School of Tsinghua University.

References