Answer Sets for Prioritized Logic Programs

Yan Zhang
Department of Computing
University of Western Sydney, Nepean
Kingswood, NSW 2747, Australia
E-mail: yan@st.nepean.uws.edu.au

Norman Y. Foo
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW 2052, Australia
E-mail: norman@cse.unsw.edu.au

Abstract

Conflict resolution is an important issue in knowledge representation and reasoning. A common idea of solving conflicts in reasoning is to add preferences in the underlying reasoning mechanism. This paper describes extensions of Gelfond and Lifschitz's extended logic programs [5] by adding preference information. We first propose prioritized logic programs (PLPs) in which the preference is expressed *statically*. An extended answer set semantics is provided for PLPs. We then extend PLPs to dynamic PLPs (DPLPs) in which the preference can be expressed *dynamically*. The semantics of DPLPs is defined in terms of answer sets of the corresponding PLPs. By illustrating typical examples, we show how conflicts between rules are resolved in PLPs and DPLPs. We also investigate basic properties of PLPs and DPLPs in detail.

1 Introduction

Conflict resolution is an important issue in knowledge representation and reasoning. A common idea of solving conflicts in reasoning is to add preferences in the underlying reasoning mechanism. The goal of this paper is to investigate this problem in the framework of logic programs. In particular, we extend Gelfond and Lifschitz's extended logic programs by adding preference information. We first consider logic programs with *static* preferences which we call *prioritized logic programs* or PLPs, and then describe logic programs with *dynamic* preferences (dynamic PLPs or DPLPs). Formal semantics for PLPs and DPLPs is provided based on extensions of Gelfond and Lifschitz's answer set semantics for extended logic programs [5].

The paper is organized as follows. Next section introduces the syntax of

PLPs. Section 3 provides an answer set semantics of PLPs. By illustrating several typical examples, this section also shows how conflicts are resolved in PLPs. Section 4 defines syntax and semantics for dynamic PLPs (DPLPs), and presents simple applications of DPLPs, while section 5 investigates basic properties of PLPs and DPLPs. Finally, section 6 discusses some related work and concludes the paper.

2 Syntax of Prioritized Logic Programs (PLPs)

In this section we provide a formal description of prioritized logic programs (PLPs).

Our language $\mathcal{L}$ includes the following vocabulary:

- *Variables*: x, y, z, $\cdots$.
- *Constants*: C, C_1, C_2, $\cdots$, including logical constants *True* and *False*.
- *Predicates*: $P, Q, R, \cdots$.
- *Names*: $N, N_1, N_2, \cdots$.
- A strict partial ordering (i.e. antireflexive, antisymmetric and transitive) $<$ on names.
- A naming function $\mathcal{N}$, which maps a *rule* (see below) to a name.
- A symbol $\leftarrow$, which is used to represent a rule.
- Connectives $\neg$ and *not*, where $\neg$ represents the classical negation (strong negation), and *not* represents *negation as failure* (weak negation).

We also require that the sets of variables, constants, predicates and names be disjoint.

A *term* is either a constant or a variable. An *atom* is $P(t_1, \cdots, t_k)$, where P is a predicate of arity k and $t_1, \cdots, t_k$ are terms. A *literal* is either an atom P or a negation of an atom $\neg P$. A *rule* is a statement of the form

$$L_0 \leftarrow L_1, \cdots, L_m, not\ L_{m+1}, \cdots, not\ L_n,$$

where L_i $(0 \leq i \leq n)$ is a literal. L_0 is the *head* of the rule, while $L_1, \cdots, L_m, not\ L_{m+1}, \cdots, not\ L_n$ is the *body* of the rule. Obviously, the body of a rule could be empty.

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

For the naming function $\mathcal{N}$, we require that for any rules r and r' in a PLP (see the following definition), $\mathcal{N}(r) = \mathcal{N}(r')$ iff r and r' indicate the same rule.

An *extended logic program* Π is a collection of rules [5]. A *prioritized logic program* (PLP) $\mathcal{P}$ is a triplet $(\Pi, \mathcal{N}, <)$, where Π is an extended logic program, $\mathcal{N}$ is a naming function mapping each rule in Π to a name, and $<$ is a relation representing all strict partial orderings on names.

The following is an example of prioritized extended logic program.
$\mathcal{P}_1 = (\{P \leftarrow not\ Q,\ not\ R,\ Q \leftarrow not\ P,\ R \leftarrow not\ P\},\ \{\mathcal{N}(P \leftarrow not\ Q,\ not\ R) = N_1,\ \mathcal{N}(Q \leftarrow not\ P) = N_2,\ \mathcal{N}(R \leftarrow not\ P) = N_3\},\ \{N_1 < N_2, N_2 < N_3\})$.
To simplify our presentation, we usually represent $\mathcal{P}_1$ as the following form:

$\mathcal{P}_1$:
$N_1 : P \leftarrow not\ Q,\ not\ R,$
$N_2 : Q \leftarrow not\ P,$
$N_3 : R \leftarrow not\ P,$
$N_1 < N_2, N_2 < N_3.$

We also use notations $\mathcal{P}_1(\Pi)$, $\mathcal{P}_1(\mathcal{N})$, and $\mathcal{P}_1(<)$ to denote the sets of rules, naming function's values and $<$-relation of $\mathcal{P}_1$ respectively.

Consider the following program:

$\mathcal{P}_2$:
$N_1 : P \leftarrow not\ Q,\ not\ R,$
$N_2 : Q \leftarrow not\ P,$
$N_3 : R \leftarrow not\ P,$
$N_1 < N_2, N_2 < N_3, N_1 < N_3.$

Obviously, the only difference between $\mathcal{P}_1$ and $\mathcal{P}_2$ is that there is one more relation $N_1 < N_3$ in $\mathcal{P}_2$. As we mentioned earlier, since $<$ is a strict partial ordering (i.e., antireflexive, antisymmetric and transitive), we would expect that $\mathcal{P}_1$ and $\mathcal{P}_2$ are identical in some sense. Furthermore, if we rename rules in $\mathcal{P}_2$ as follows,

$\mathcal{P}'_2$:
$N'_1 : P \leftarrow not\ Q,\ not\ R,$
$N'_2 : Q \leftarrow not\ P,$
$N'_3 : R \leftarrow not\ P,$
$N'_1 < N'_2, N'_2 < N'_3, N'_1 < N'_3,$

$\mathcal{P}'_2$ would be also identical to $\mathcal{P}_2$ and hence to $\mathcal{P}_1$ too from our intuition. To make this precise, we first introduce $<$-*closure* as follows.

Definition 1 *Given a program $\mathcal{P} = (\Pi, \mathcal{N}, <)$. $\mathcal{P}(<^+)$ is the $<$-*closure *of $\mathcal{P}$ iff $\mathcal{P}(<^+)$ is the smallest set containing $\mathcal{P}(<)$ and closed under transitivity.*

We also need to define a renaming function as follows. A *renaming function* Rn maps a PLP $\mathcal{P} = (\Pi, \mathcal{N}, <)$ to another PLP $\mathcal{P}'$, i.e. $Rn(\mathcal{P}) = \mathcal{P}' = (\Pi', \mathcal{N}', <')$, such that (i) $\mathcal{P}(\Pi) = \mathcal{P}'(\Pi')$; (ii) for each rule $r \in \mathcal{P}(\Pi)$[1], $\mathcal{N}(r) = N \in \mathcal{P}(\mathcal{N})$ iff $\mathcal{N}'(r) = N' \in \mathcal{P}'(\mathcal{N}')$ (N and N' are not necessarily

[1]Of course, r is also in $\mathcal{P}'(\Pi')$.

different); (iii) for any rules r_1 and r_2 in $\mathcal{P}(\Pi)$, $\mathcal{N}(r_1) = N_1$, $\mathcal{N}(r_2) = N_2 \in \mathcal{P}(\mathcal{N})$, and $N_1 < N_2 \in \mathcal{P}(<)$ iff $\mathcal{N}'(r_1) = N_1'$, $\mathcal{N}'(r_2) = N_2' \in \mathcal{P}'(\mathcal{N}')$, and $N_1' < N_2' \in \mathcal{P}'(<')$. It is easy to see that applying a renaming function to a PLP will only change the names of rules in the PLP.

Two prioritized extended logic programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are *identical* iff there exists a renaming function Rn, mapping $\mathcal{P}_2$ to $\mathcal{P}'_2$ such that $\mathcal{P}_1(\Pi) = \mathcal{P}'_2(\Pi')$, $\mathcal{P}_1(\mathcal{N}) = \mathcal{P}'_2(\mathcal{N}')$, and $\mathcal{P}_1(<^+) = \mathcal{P}'_2(<'^+)$.

We have defined that a prioritized extended logic program is a Gelfond and Lifschitz's extended logic program [5] by associating with a partial ordering $<$ to it. Intuitively such ordering represents a preference of applying rules during the evaluation of the program. In particular, if in a program $\mathcal{P}$, relation $\mathcal{N}(r) < \mathcal{N}(r')$ holds, rule r would be preferred to apply over rule r' during the evaluation of $\mathcal{P}$ (i.e. rule r is more preferred than rule r'). Consider the following classical example represented in our formalism:

$\mathcal{P}_3$:
$N_1 : Fly(x) \leftarrow Bird(x),\ not\ \neg Fly(x),$
$N_2 : \neg Fly(x) \leftarrow Penguin(x),\ not\ Fly(x),$
$N_3 : Bird(Tweety) \leftarrow,$
$N_4 : Penguin(Tweety) \leftarrow,$
$N_2 < N_1.$

Obviously, rules N_1 and N_2 conflict with each other as their heads are complementary literals[2], and applying N_1 will defeat N_2[3] and *vice versa*. However, as $N_2 < N_1$, we would expect that rule N_2 is preferred to apply first and then defeat rule N_1 after applying N_2 so that the desired solution $\neg Fly(Tweety)$ could be derived.

3 Semantics of PLPs

In this section, we develop the semantics of PLPs. Our method is based on an extension of answer set semantics for extended logic programs [5]. Before we present our idea in detail, we need to introduce this answer set semantics first.

3.1 Answer Sets for Extended Logic Programs: A Review

Let Π be an extended logic program. For simplicity, we treat a rule r in Π with variables as the set of all ground instances of r formed from the set of ground literals of the language of Π. We will also adopt this assumption in our prioritized extended logic programs. In the rest of paper, we will

[2] Precisely, N_2 is the name of rule $\neg Fly(x) \leftarrow Penguin(x),\ not\ Fly(x)$. Whenever there is no confusion in the context, we just simply refer a rule by its name.

[3] Informally, when we say that a rule is defeated in a program, it simply means that this rule is unapplicable in the evaluation of the program. A precise definition of defeatness is given in the following section.

not explicitly declare this assumption whenever there is no ambiguity in our discussion.

Let Π be an extended logic program not containing *not* and Lit the set of all ground literals in the language of Π. The *answer set* of Π, denoted as $Ans(\Pi)$, is the smallest subset S of Lit such that

(i) for any rule $L_0 \leftarrow L_1, \cdots, L_m$ from Π, if $L_1, \cdots, L_m \in S$, then $L_0 \in S$;

(ii) if S contains a pair of complementary literals, then $S = Lit$.

Now consider Π be an extended logic program. For any subset S of Lit, let Π^S be the logic program obtained from Π by deleting

(i) each rule that has a formula *not* L in its body with $L \in S$, and

(ii) all formulas of the form *not* L in the bodies of the remaining rules.

We define that S is an *answer set* of Π, denoted $Ans(\Pi)$, iff S is an answer set of Π^S, i.e. $S = Ans(\Pi^S)$.

Consider $\mathcal{P}_3$ presented in last section. It is not difficult to see that extended logic program $\mathcal{P}_3(\Pi)$ has two answer sets: $\{Bird(Tweety)$, $Penguin(Tweety), \neg Fly(Tweety)\}$ and $\{Bird(Tweety), Penguin(Tweety)$, $Fly(Tweety)\}$.

3.2 Answer Sets for PLPs

In program $\mathcal{P}_3$, we have seen that rules N_1 and N_2 conflict with each other. Since $N_2 < N_1$, we try to solve the conflict by applying N_2 first and defeating N_1. However, in some programs, even if one rule is more preferred than the other, these two rules may not affect each other at all during the evaluation of the program. In this case, the preference relation between these two rules does not play any role in the evaluation and should be simply ignored. This is illustrated by the following program:

$\mathcal{P}_4$:
$N_1 : P \leftarrow not\ Q_1$,
$N_2 : \neg P \leftarrow not\ Q_2$,
$N_1 < N_2$.

Although heads of N_1 and N_2 are complementary literals, applying N_1 will not affect the applicability of N_2 and *vice versa*. Hence $N_1 < N_2$ should not be taken into account during the evaluation of $\mathcal{P}_4$. The following definition provides a formal description for this intuition.

Definition 2 *Let Π be an extended logic program and r a rule with the form $L_0 \leftarrow L_1, \cdots, L_m,\ not\ L_{m+1}, \cdots,\ not\ L_n$ (r does not necessarily belong to Π). Rule r is* defeated *by Π iff for any answer set $Ans(\Pi)$ of Π, there exists some $L_i \in Ans(\Pi)$, where $m + 1 \leq i \leq n$.*

Now our idea of evaluating a PLP is as follows. Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$. If there are two rules r and r' in $\mathcal{P}(\Pi)$ and $\mathcal{N}(r) < \mathcal{N}(r')$, r' will be ignored in the evaluation of $\mathcal{P}$, *only if* keeping r in $\mathcal{P}(\Pi)$ and deleting r' from $\mathcal{P}(\Pi)$ will result in a defeat of r', i.e. r' is defeated by $\mathcal{P}(\Pi) - \{r'\}$. By eliminating all such potential rules from $\mathcal{P}(\Pi)$, $\mathcal{P}$ is eventually reduced to an extended logic program in which the partial ordering $<$ has been removed. Our evaluation for $\mathcal{P}$ is then based on this extended logic program.

Let us consider program $\mathcal{P}_3$ once again. Since $N_2 < N_1$ and N_1 is defeated by $\mathcal{P}_3 - \{N_1\}$ (i.e. the unique answer set of $\mathcal{P}_3 - \{N_1\}$ is $\{Bird(Tweety), Penguin(Tweety), \neg Fly(Tweety)\}$), rule N_1 should be ignored during the evaluation of $\mathcal{P}_3$. For program $\mathcal{P}_4$, on the other hand, although $N_1 < N_2$, relation $N_1 < N_2$ will not affect the solution of evaluating $\mathcal{P}_4$ as $\mathcal{P}_4(\Pi) - \{N_2\}$ does not defeat N_2 (i.e. the unique answer set of $\mathcal{P}_4(\Pi) - \{N_2\}$ is $\{P\}$).

Definition 3 *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a prioritized extended logic program. $\mathcal{P}^<$ is a* reduct *of $\mathcal{P}$ with respect to $<$ if and only if there exists a sequence of sets Π_i $(i = 0, 1, \cdots)$ such that:*

(i) $\Pi_0 = \Pi$;

(ii) $\Pi_i = \Pi_{i-1} - \{r_1, \cdots, r_k \mid$ *(a) there exists $r \in \Pi_{i-1}$ such that for every j $(j = 1, \cdots, k)$, $\mathcal{N}(r) < \mathcal{N}(r_j) \in \mathcal{P}(<^+)$ and $r_1, \cdots, r_k$ are defeated by $\Pi_{i-1} - \{r_1, \cdots, r_k\}$, and (b) there does not exist a rule $r' \in \Pi_{i-1}$ such that $N(r_j) < N(r')$ for some j $(j = 1, \cdots, k)$ and r' is defeated by $\Pi_{i-1} - \{r'\}\}$;*

(iii) $\mathcal{P}^< = \bigcap_{i=0}^{\infty} \Pi_i$.

In Definition 3, clearly $\mathcal{P}^<$ is an extended logic program obtained from Π by eliminating some rules from Π. In particular, if $\mathcal{N}(r) < \mathcal{N}(r')$ and $\Pi - \{r'\}$ defeats r', then rule r' will be eliminated from Π if no *less preferred rule* can be eliminated (i.e. conditions (a) and (b)). This procedure is continued until a fixed point is reached. Note that due to the transitivity of $<$, we need to consider each $\mathcal{N}(r) < \mathcal{N}(r')$ in the $<$-closure of $\mathcal{P}$.

Example 1 Using Definition 1 and 3, it is not difficult to conclude that $\mathcal{P}_1$, $\mathcal{P}_3$ and $\mathcal{P}_4$ have unique reducts as follows respectively:

$$\begin{aligned}
\mathcal{P}_1^< &= \{P \leftarrow not\ Q,\ not\ R\},\\
\mathcal{P}_3^< &= \{\neg Fly(x) \leftarrow Penguin(x),\ not\ Fly(x),\\
&\qquad Bird(Tweety) \leftarrow, Penguin(Tweety) \leftarrow\},\\
\mathcal{P}_4^< &= \mathcal{P}_4(\Pi).
\end{aligned}$$

■

It is quite obvious to note that the reduct of a PLP may not be unique as the following example shows.

Example 2 Consider a PLP $\mathcal{P}_5$:

$\mathcal{P}_5$:
$N_1 : P \leftarrow,$
$N_2 : Q \leftarrow not\ R,$
$N_3 : T \leftarrow,$
$N_4 : R \leftarrow not\ Q,$
$N_1 < N_2, N_3 < N_4.$

According to Definition 3, it is easy to see that $\mathcal{P}_5$ has two reducts: $\{P \leftarrow, T \leftarrow, R \leftarrow not\ Q\}$ and $\{P \leftarrow, Q \leftarrow not\ R, T \leftarrow\}$. ■

We should also mention that the condition (b) in the construction of Π_i in Definition 3 is necessary. Without ths condition, some unintuitive results may be derived. For instance, if we have additional preference information $N_3 < N_2$ in program $\mathcal{P}_3$, then using a modified version of Definition 3 without condition (b) in the construction of Π_i, we will conclude that $\{Fly(x) \leftarrow Bird(x),\ not\ \neg Fly(x),\ Bird(Tweety) \leftarrow, Penguin(Tweety) \leftarrow\}$ is also a reduct of $\mathcal{P}_3$, which, as will be followed by Definition 4 next, leads to an unintuitive result saying that Tweety can fly.

Now it is quite straightforward to define the answer set(s) for a prioritized extended logic program.

Definition 4 *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a PLP and Lit the set of all ground literals in the language of $\mathcal{P}$. For any subset S of Lit, S is an* answer set *of $\mathcal{P}$, denoted as $Ans^P(\mathcal{P})$, iff $S = Ans(\mathcal{P}^<)$ for some reduct $\mathcal{P}^<$ of $\mathcal{P}$.*

Example 3 Immediately from Definition 4 and Examples 1 and 2, we have the following solutions:

$Ans^P(\mathcal{P}_1) = \{P\},$
$Ans^P(\mathcal{P}_3) = \{Bird(Tweety),$
$\qquad Penguin(Tweety), \neg Fly(Tweety)\},$
$Ans^P(\mathcal{P}_4) = Lit,$
and two answer sets for $\mathcal{P}_5$:
$Ans^P(\mathcal{P}_5) = \{P, R, T\},$
$Ans^P(\mathcal{P}_5) = \{P, Q, T\},$

which, respectively, are also consistent with our intuitions. ■

3.3 More Examples

Now let us examine more examples to illustrate some features of PLPs.

Example 4 Let $\mathcal{P}_6$ be:

$\mathcal{P}_6$:
$N_1 : P \leftarrow not\ Q,$
$N_2 : \neg P \leftarrow not\ P,$
$N_2 < N_1.$

This program was originally presented in [2]. The reduct of $\mathcal{P}_6$ is $\mathcal{P}_6(\Pi)$. So the unique answer set of $\mathcal{P}_6$ is $\{P\}$. Note that even if the heads of N_1 and N_2 are complementary literals and $N_2 < N_1$, rule N_1 can not be deleted from $\mathcal{P}_6(\Pi)$ as $\mathcal{P}_6(\Pi) - \{N_1\}$ does not defeat N_1. ■

Example 5 Let $\mathcal{P}_7$ be:

$$\begin{array}{l} \mathcal{P}_7: \\ \quad N_1 : P \leftarrow \mathit{not}\ Q, \\ \quad N_2 : Q \leftarrow \mathit{not}\ P, \\ \quad N_3 : R \leftarrow \mathit{not}\ Q,\ \mathit{not}\ S, \\ \quad N_4 : S \leftarrow \mathit{not}\ R, \\ \quad N_1 < N_2, N_3 < N_4. \end{array}$$

$\mathcal{P}_7(\Pi)$ has three answer sets: $\{P, R\}$, $\{Q, S\}$, and $\{P, S\}$. The unique reduct of $\mathcal{P}_7$ is $\{P \leftarrow \mathit{not}\ Q,\ R \leftarrow \mathit{not}\ Q,\ \mathit{not}\ S\}$. Therefore, the unique answer set of $\mathcal{P}_7$ is $Ans^P(\mathcal{P}_7) = \{P, R\}$. Note that this solution is consistent with our intuition since $N_1 < N_2$ and $N_3 < N_4$, and applying N_1 and N_3 causes N_2 and N_4 inapplicable respectively. ■

Example 6 Let $\mathcal{P}_8$ be:

$$\begin{array}{l} \mathcal{P}_8: \\ \quad N_1 : P \leftarrow \mathit{not}\ Q,\ \mathit{not}\ R, \\ \quad N_2 : Q \leftarrow \mathit{not}\ P, \\ \quad N_3 : R \leftarrow \mathit{not}\ P, \\ \quad N_1 < N_2. \end{array}$$

$\mathcal{P}_8(\Pi)$ has two answer sets $\{P\}$ and $\{Q, R\}$. Obviously, N_2 is not defeated by $\mathcal{P}_8(\Pi) - \{N_2\}$ as P only belongs to one of two answer sets $\{P\}$ and $\{R\}$ of $\mathcal{P}_8(\Pi) - \{N_2\}$. Therefore, the unique reduct $\mathcal{P}_8^{<}$ of $\mathcal{P}_8$ is the same as $\mathcal{P}_8(\Pi)$. So $\mathcal{P}_8$ has two answer sets $\{P\}$ and $\{Q, R\}$.

It is worth observing that if we add $N_1 < N_3$ to $\mathcal{P}_8$, $\mathcal{P}_8$ will then has a unique answer set $\{P\}$. On the other hand, if we add $N_3 < N_1$ instead of $N_1 < N_3$, the unique answer set of $\mathcal{P}_8$ will then have $\{Q, R\}$. ■

4 Logic Programs with Dynamic Preferences

So far, preferences specified in our prioritized logic programs are *static* in the sense that the partial ordering $<$ among rules is pre-defined from *outside*. Using PLPs to represent knowledge of a domain, the user must explicitly specify his/her preference information about the domain. However, as observed by Brewka recently [2], in many situations, preferences are context-dependent, and there may not exist a feasible way to specify such preferences explicitly. Consider the following extended logic program:

$\mathcal{P}_9$:

$$\begin{aligned}
N_1 : \neg Employed(x) &\leftarrow Student(x),\\
&\quad not\ Employed(x),\\
N_2 : Employed(x) &\leftarrow Age(x, > 25),\\
&\quad not\ \neg Employed(x),\\
N_3 : Student(x) &\leftarrow FT\text{-}Student(x),\\
N_4 : Student(x) &\leftarrow PT\text{-}Student(x),\\
N_5 : Student(Peter) &\leftarrow,\\
N_6 : Age(Peter, > 25) &\leftarrow.
\end{aligned}$$

Obviously, $\mathcal{P}_9$ has two answer sets, from one we conclude $\neg Employed(Peter)$ and from the other we conclude $Employed(Peter)$. By specifying $N_1 < N_2$ or $N_2 < N_1$, we can retain one answer set and exclude the other, and hence resolve the conflict as it may occur. However, the reason for specifying $N_1 < N_2$ (or $N_2 < N_1$) rather than $N_2 < N_1$ (or $N_1 < N_2$) is completely motivated by the user. There is no way in our PLPs to express the preference about preference. For instance, with a more natural way, we may hope to express that "if x is a full-time student, then N_1 is more preferred than N_2, while if x is a part-time student, then N_2 is more preferred than N_1". Therefore, if we obtain further information knowing that Peter is a full-time student, we would expect to conclude that Peter is unemployed, otherwise Peter's employment status will remain indefinite.

Hence, to make our system more flexible, we need to reason about preferences. In other words, we hope to specify the preference information *dynamically* in our prioritized logic programs. In this section, we will discuss the dynamic PLPs (or DPLPs for short). We provide an answer set semantics for DPLPs based on the answer set semantics of PLPs.

4.1 Syntax of DPLPs

A language $\mathcal{L}^D$ of DPLPs is a language $\mathcal{L}$ of PLPs except the following modifications:

- *Variables* consist of variables $x, y, z, \cdots$ of $\mathcal{L}$ and name variables n, n_1, n_2, $\cdots$, where $\{x, y, z, \cdots\}$ and $\{n, n_1, n_2, \cdots\}$ are disjoint.

- *Constants* consist of constants $C, C_1, C_2, \cdots$ of $\mathcal{L}$ and name constants N, N_1, N_2, $\cdots$, where $\{C, C_1, C_2, \cdots\}$ and $\{N, N_1, N_2, \cdots\}$ are disjoint.

- *Names* consist of name variables and name constants. The naming function $\mathcal{N}$ maps each rule to a name constant.

- A special predicate $<$ takes two names as arguments, where $<$ is used to represent a strict partial ordering among rules.

Terms, atoms, literals and rules are defined as the same in PLPs but under the language $\mathcal{L}^D$. Since $<$ is a special predicate in $\mathcal{L}^D$, $<$ can occur in any rules of $\mathcal{L}^D$. For instance, $N_1 < N_2 \leftarrow not\ N_2 < N_1$ and $n_1 < n_2 \leftarrow P(n_1), Q(n_2), not\ n_2 < n_1$ could be valid rules of $\mathcal{L}^D$.

A *dynamic* PLP (DPLP) is a pair $\mathcal{P} = (\Pi, \mathcal{N})$, where Π is a collection of rules of $\mathcal{L}^D$ and $\mathcal{N}$ is a naming function that maps each rule of Π to a name constant. Given a DPLP $\mathcal{P}$, Lit^D denotes the set of all ground literals of the language $\mathcal{L}^D$ of $\mathcal{P}$.

To keep the partial ordering $<$ consistent, we assume that any DPLP includes the following two rules[4]:

$$n_1 < n_3 \leftarrow n_1 < n_2, n_2 < n_3, \tag{1}$$

$$\neg n_2 < n_1 \leftarrow n_1 < n_2{}^5. \tag{2}$$

Example 7 Consider a DPLP as follows.

$\mathcal{P}_{10}$:
$N_1 : \neg Employed(x) \leftarrow Student(x),$
$\qquad not\ Employed(x),$
$N_2 : Employed(x) \leftarrow Age(x, > 25),$
$\qquad not\ \neg Employed(x),$
$N_3 : Student(x) \leftarrow FT\text{-}Student(x),$
$N_4 : Student(x) \leftarrow PT\text{-}Student(x),$
$N_5 : FT\text{-}Student(Peter) \leftarrow,$
$N_6 : Age(Peter, > 25) \leftarrow,$
$N_7 : N_1 < N_2 \leftarrow FT\text{-}Student(x), not\ N_2 < N_1,$
$N_8 : N_2 < N_1 \leftarrow PT\text{-}Student(x), not\ N_1 < N_2.$

$\mathcal{P}_{10}$ is similar to $\mathcal{P}_9$ except that there are two rules N_7 and N_8 in $\mathcal{P}_{10}$ which express conditional preferences and we know Peter is a full-time student (i.e. rule N_5)[6]. Intuitively, rule N_7 is interpreted as "if x is a full-time student and there is no explicit knowledge to conclude that N_2 is more preferred than N_1, then rule N_1 is more preferred than rule N_2". A similar interpretation can be stated for N_8. It should be also noted that in a DPLP, preference relations on rules are not explicitly represented like PLPs. They have been encoded into the rules of the DPLP. ■

4.2 Answer Set Semantics for DPLPs

Now we try to provide a formal semantics for DPLPs. Our method of evaluating a DPLP is based on a transformation of each DPLP into a PLP in language $\mathcal{L}^D$ under a sequence of reductions with respect to the partial ordering $<$[7].

[4]To simplify our presentation, we will not explicitly represent these two rules in our DPLPs.

[6]Note that implicitly other two rules (1) and (2) should be also included in $\mathcal{P}_{10}$.

[7]In the rest of the paper, whenever there is no confusion, we will simply refer $<$ as a partial ordering rather than a special predicate in $\mathcal{L}^D$ used to present a partial ordering.

Before to present our method formally, we first introduce some useful notations. Firstly, a DPLP $\mathcal{P} = (\Pi, \mathcal{N})$ can be treated as a special PLP *in* language $\mathcal{L}^D$ with the form $(\Pi, \mathcal{N}, <_0)$, where $<_0 = \emptyset$. Generally, we say that a logic program $\mathcal{P}$ is a PLP in language $\mathcal{L}^D$ if the program is specified as the form $(\Pi, \mathcal{N}, <)$, where Π is a set of rules of $\mathcal{L}^D$, $\mathcal{N}$ is a naming function mapping each rule in Π to a name constant and $<$ is a set of ground atoms of the form $N < N'$. In this case, we can compute each of $\mathcal{P}$'s *PLP answer sets*, denoted as $Ans^{P,D}(\mathcal{P})$, by using the approach proposed in section 3. Consider the following program.

$\mathcal{P}_{11}$:
$N_1 : P \leftarrow not\ Q,$
$N_2 : Q \leftarrow not\ P,$
$N_3 : N_1 < N_2 \leftarrow not\ N_2 < N_1,$
$N_4 : N_2 < N_1 \leftarrow not\ N_1 < N_2,$
$N_3 < N_4.$

Note that $\mathcal{P}_{11}$ is not a DPLP but a PLP in language $\mathcal{L}^D$ as $N_3 < N_4$ is specified from *outside* of the rules of $\mathcal{P}_{11}$. Clearly, under the answer set semantics of PLPs, $\mathcal{P}_{11}$ has two answer sets: $\{P, N_1 < N_2\}$ and $\{Q, N_1 < N_2\}$.

Now we give the formal descriptions of the semantics of DPLPs.

Definition 5 *Let $\mathcal{P} = (\Pi, \mathcal{N})$ be a DPLP. A PLP $\mathcal{P}^*$ in $\mathcal{L}^D$ is a* transformation *of $\mathcal{P}$ to PLP if and only if there exists a sequence of sets Ψ_i $(i = 0, 1, \cdots)$ such that*

(i) $\Psi_0 = (\Pi_0, \mathcal{N}, <_0)$, *where* $\Pi_0 = \Pi$ *and* $<_0 = \emptyset$;

(ii) $\Psi_i = (\Pi_i, \mathcal{N}, <_i)$, *where* $\Pi_i = \Psi_{i-1}^{<_{i-1}}$ *is a reduct*[8] *of* Ψ_{i-1} *with respect to* $<_{i-1}$, *and* $<_i = \{N < N' \mid N < N'$ *belongs to all PLP answer sets of* $\Psi_{i-1}\}$;

(iii) $\mathcal{P}^* = (\Pi_\infty, \mathcal{N}, <_\infty)$.

Definition 6 *Let $\mathcal{P} = (\Pi, \mathcal{N})$ be a DPLP. For any subset S of Lit^D, S is an* answer set *of $\mathcal{P}$, denoted as $Ans^D(\mathcal{P})$, iff $S = Ans^{P,D}(\mathcal{P}^*)$ for some transformation $\mathcal{P}^*$ of $\mathcal{P}$ to PLP as defined in Definition 5.*

Let us examine the above definitions more closely. The basic idea of evaluating a DPLP $\mathcal{P}$ is to transform $\mathcal{P}$ to a corresponding PLP $\mathcal{P}^*$ in $\mathcal{L}^D$, and then use the PLP answer set semantics to evaluate $\mathcal{P}^*$ (eg. Definition 6). From Definition 5, we can see that during this transformation, $\mathcal{P}$ is first treated as a special PLP in $\mathcal{L}^D$, i.e. Ψ_0. Then for each i $(i = 1, 2, \cdots)$, $\Psi_i = (\Pi_i, \mathcal{N}, <_i)$ is a PLP in $\mathcal{L}^D$ and generated from $\Psi_{i-1} = (\Pi_{i-1}, \mathcal{N}, <_{i-1}$

[8] See Definition 3.

). Intuitively, $<_i$ presents all ground instances of the partial ordering $<$ derived from Ψ_{i-1} under the semantics of PLP, and Π_i is a reduct of Ψ_{i-1} by eliminating defeated rules from Π_{i-1} with respect to $<_{i-1}$ (eg. $\Pi_i = \Psi_{i-1}^{<_{i-1}}$). This transformation procedure is continued until a fixed point is reached. To show how the answer set(s) of a DPLP can be computed, let us consider the following examples.

Example 8 Consider a DPLP $\mathcal{P}_{12}$ as follows.

$\mathcal{P}_{12}$:
$N_1 : P \leftarrow not\ Q,$
$N_2 : Q \leftarrow not\ P,$
$N_3 : N_1 < N_2 \leftarrow not\ N_2 < N_1,$
$N_4 : N_2 < N_1 \leftarrow not\ N_1 < N_2,$
$N_5 : N_3 < N_4 \leftarrow.$

According to Definition 5, we get the following sequence of Ψ_i $(i = 1, 2, 3)$[9]:

Ψ_1:
$N_1 : P \leftarrow not\ Q,$
$N_2 : Q \leftarrow not\ P,$
$N_3 : N_1 < N_2 \leftarrow not\ N_2 < N_1,$
$N_4 : N_2 < N_1 \leftarrow not\ N_1 < N_2,$
$N_5 : N_3 < N_4 \leftarrow,$
$N_3 < N_4.$
Ψ_2:
$N_1 : P \leftarrow not\ Q,$
$N_2 : Q \leftarrow not\ P,$
$N_3 : N_1 < N_2 \leftarrow not\ N_2 < N_1,$
$N_5 : N_3 < N_4 \leftarrow,$
$N_1 < N_2, N_3 < N_4.$
Ψ_3:
$N_1 : P \leftarrow not\ Q,$
$N_3 : N_1 < N_2 \leftarrow not\ N_2 < N_1,$
$N_5 : N_3 < N_4 \leftarrow,$
$N_1 < N_2, N_3 < N_4.$

Then it is easy to verify that $\Psi_4 = \Psi_3$. So the transformation of $\mathcal{P}_{12}$ to PLP is $\mathcal{P}_{12}^* = \Psi_3$[10]. Therefore, from Definition 6, $\mathcal{P}_{12}$ has a unique answer set $Ans^D(\mathcal{P}_{12}) = \{P, N_1 < N_2,\ N_3 < N_4, \neg N_2 < N_1, \neg N_4 < N_3\}$[11]. ■

Example 9 Example 7 continued. Ignoring the detail, it is not difficult to see that $\mathcal{P}_{10}$ has a unique answer set $Ans^D(\mathcal{P}_{10}) = \{Age(Peter, > 25)$, $FT\text{-}Student(Peter)$, $Student(Peter)$, $\neg Employed(Peter)$, $N_1 < N_2$, $\neg N_2 < N_1\}$, which presents the desired result for $\mathcal{P}_{10}$. ■

[9] Ψ_0 is trivial.
[10] In this example, the transformation is unique.
[11] Recall that any DPLP must include rules (1) and (2).

Example 10 Consider a DPLP $\mathcal{P}_{13}$ as follows.

$\mathcal{P}_{13}$:
$N_1 : n_1 < n_2 \leftarrow P(n_1), Q(n_2),\ not\ n_2 < n_1,$
$N_2 : R(C) \leftarrow not\ R(C'),$
$N_3 : R(C') \leftarrow not\ R(C),$
$N_4 : P(N_2) \leftarrow,$
$N_5 : Q(N_3) \leftarrow.$

This program is a bit different from those DPLPs discussed above. Intuitively, N_1 can be viewed as a general rule about the preference of the domain - "for any two rules n_1 and n_2, if n_1 and n_2 satisfy properties P and Q respectively, and there is no explicit knowledge to conclude that n_2 is more preferred than n_1, then n_1 is more preferred than n_2", while N_2 - N_5 present explicit knowledge of the domain[12]. Clearly, using the approach described above, $\mathcal{P}_{13}$ has a unique answer set $\{R(C), P(N_2), Q(N_3), N_2 < N_3, \neg N_3 < N_2\}$. ∎

5 Properties of Prioritized Logic Programs

In this section we discuss some properties of PLPs and DPLPs in detail. We first discuss the property of PLPs. To simplify our presentation, let us introduce some useful notations.

Let Π be an extended logic program. We use $ANS(\Pi)$ to denote the class of answer sets of Π. Suppose $\mathcal{P} = (\Pi, \mathcal{N}, <)$ is a PLP. From Definition 3, we can see that a reduct $\mathcal{P}^<$ of $\mathcal{P}$ is generated from a sequence of extended logic programs: $\Pi = \Pi_0, \Pi_1, \Pi_2, \cdots$. We use notation $\{\Pi_i\}$ $(i = 0, 1, 2, \cdots)$ to denote this sequence and call it a *reduct chain* of $\mathcal{P}$. Then we can prove the following useful result[13].

Theorem 1 *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a PLP, and $\{\Pi_i\}$ $(i = 0, 1, 2, \cdots)$ a reduct chain of $\mathcal{P}$. Suppose each Π_i has answer set(s). Then for any i and j where $i < j$, $ANS(\Pi_j) \subseteq ANS(\Pi_i)$.*

Theorem 1 shows an important property of the reduct chain of $\mathcal{P}$: each Π_i is consistent with Π_{i-1} but becomes more *specific* than Π_{i-1} in the sense that all answer sets of Π_i are answer sets of Π_{i-1} but some answer sets of Π_{i-1} are filtered out if they conflict with the preference partial ordering $<$. The following theorem shows the answer set relation between a PLP and its corresponding extended logic programs.

Theorem 2 *Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a PLP. Then a subset S of Lit is an answer set of $\mathcal{P}$ iff S is an answer set of each Π_i for some reduct chain $\{\Pi_i\}$ $(i = 0, 1, 2, \cdots)$ of $\mathcal{P}$, where each Π_i has answer set(s).*

[12] An application of this kind of program in legal reasoning was addressed in [2].
[13] All proofs of theorems presented in this paper were given in our full paper [9].

Now we investigate properties of DPLPs. Let $\mathcal{P}$ and $\mathcal{P}'$ be a DPLP and a PLP in $\mathcal{L}^D$ respectively. We use $ANS^{P,D}(\mathcal{P}')$ to denote the class of PLP answer sets of $\mathcal{P}'$. From Definition 5, we can see that a transformation $\mathcal{P}^*$ of $\mathcal{P}$ is generated from a sequence of PLPs in $\mathcal{L}^D$: Ψ_0, Ψ_1, Ψ_2, $\cdots$. We use notation $\{\Psi_i\}$ $(i = 0, 1, 2, \cdots)$ to denote this sequence and call it a *PLP-reduct chain* of $\mathcal{P}$. Then, similarly to the case of PLPs described earlier, we have the following results for DPLPs.

Theorem 3 *Let $\mathcal{P} = (\Pi, \mathcal{N})$ be a DPLP, and $\{\Psi_i\}$ $(i = 0, 1, 2, \cdots)$ a PLP-reduct chain of $\mathcal{P}$. Suppose each Ψ_i has PLP answer set(s). Then for any i and j where $i < j$, $ANS^{P,D}(\Psi_j) \subseteq ANS^{P,D}(\Psi_i)$.*

Theorem 4 *Let $\mathcal{P} = (\Pi, \mathcal{N})$ be a DPLP. Then a subset S of Lit^D is an answer set of $\mathcal{P}$ iff S is a PLP answer set of each Ψ_i for some PLP-reduct chain $\{\Psi_i\}$ $(i = 0, 1, 2, \cdots)$ of $\mathcal{P}$, where each Ψ_i has PLP answer set(s).*

6 Related Work and Conclusions

The issue of logic programs with preferences has been explored recently also by other researchers [4, 6, 7]. However, most of these proposals are not completely satisfactory. One of the major limitations of their work, as pointed by Brewka [2], is that the priority can only be expressed statically. Another restriction of some previous proposals is that only one type of negation was considered in their logic programs (eg. [4, 7]).

Our work described in this paper is most related to Brewka's recent work on prioritized logic programs [2], while Brewka proposed a well-founded semantics for logic programs with dynamic preferences.

Due to the space limitation, we will not compare these two semantics in detail. A thorough investigation of the relationship between these two approaches was presented in our technical report. In brief, our method inherits some advantages and drawbacks from answer set semantics, while Brewka's approach inherits some advantages and drawbacks from well-founded semantics as well. For instance, our answer set semantics can derive reasonable conclusions in most cases, but some reasonable solutions can not be obtained from Brewka's well-founded semantics (see page 35 in [2]). On the other hand, reasoning under our semantics is intractable in the general case while it can be done in polynomial time under Brewka's semantics. However, based on recent results of computations of stable models, eg. [3], it is possible to locate a reasonably broad tractable subclass of our prioritized logic programs so that the applicable range of our method can be identified. Detailed work concerning the computational analysis about our PLPs and DPLPs was presented in [9].

The prioritized logic programs proposed in this paper can be used to solve some important problems in reasoning about change. In [8] and [9], we also investigated the applications of PLPs to deal with generalized rule-based updates and actions in domains including defeasible and causal constraints. These results have enhanced our expectation of using prioritized logic programs as a general tool to formalize and implement dynamic knowledge systems in the real world.

Acknowledgements

We thank Chitta Baral for valuable comments on an earlier draft of this paper. Thanks are also due to anonymous referees for their criticisms and useful comments, especially the one of suggestion in revising Definition 3 to avoid some unintuitive results. This research is supported in part by a research grant from University of Western Sydney, Nepean and a grant from the Australian Research Council.

References

[1] C. Baral and M. Gelfond, Logic programming and knowledge representation. *Journal of Logic Programming*, **19,20** (1994) 73–148.

[2] G. Brewka, Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, **4** (1996) 19–36.

[3] R. Ben-Eliyahu, A hierarchy of tractable subsets for computing stable models. *Journal of Artificial Intelligence Research*, **6** 1996 27–52.

[4] F. Buccafurri, N. Leone, and P. Rullo, Stable models and their computation for logic programming with inheritance and true negation. *Journal of Logic Programming*, **27** 1996 5–43.

[5] M. Gelfond and V. Lifschitz, Classical negation in logic programs and disjunctive databases. *New Generation Computing*, **9** (1991) 365–386.

[6] R. Kowalski and F. Sadri, Logic programs with exceptions. *New Generation Computing*, **9** (1991) 387–400.

[7] X. Wang, J.H. You, and L.Y. Yuan, Circumscription by inference rules with priority. In *Proceedings of ECAI'96*, pp 111–114, 1996.

[8] Y. Zhang and N.Y. Foo, Towards generalized rule-based updates. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97).* Morgan Kaufmann Publishers Inc., 1997.

[9] Y. Zhang and N.Y. Foo, Prioritized logic programs and their applications. *Manuscript.* July 1997.