Answer Sets for Prioritized Logic Programs 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.	Các Tập Trả Lời cho Các Chương Trình Logic Ưu Tiên Tóm tắt 20/6 8 h 30 Giải quyết xung đột là một vấn đề quan trọng trong biểu diễn tri thức và lập luận. Ý tưởng phổ biến để giải quyết xung đột trong lập luận là thêm vào các ưu tiên trong cơ chế lập luận cơ bản. Bài báo này mô tả việc khái quát hóa các chương trình logic mở rộng của Gelfond và Lifschitz bằng cách thêm thông tin ưu tiên [5]. Trước tiên chúng tôi đề xuất các chương trình logic ưu tiên (các PLP), trong những chương trình này ưu tiên được biểu diễn dưới dạng tĩnh. Ngữ nghĩa tập trả lời mở rộng được cung cấp cho các PLP. Sau đó chúng tôi mở rộng các PLP thành các PLP động (các DPLP), trong đó ưu tiên được biểu diễn theo kiểu động. Ngữ nghĩa của các DPLP được xác định theo các tập trả lời của các PLP tương ứng. Thông qua các ví dụ minh họa điển hình, chúng tôi sẽ cho bạn thấy cách thức giải quyết xung đột giữa các quy tắc trong các PLP và DPLP. Chúng tôi cũng xét các tính chất cơ bản của các PLP và DPLP một cách chi tiết.
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	[redacted]

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 se-mantics for PLPs and DPLPs is provided based on extensions of Gelfond and Lifschitz's answer set semantics for extended logic programs [5].	[redacted]
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.	[redacted]
2 Syntax of Prioritized Logic Programs (PLPs)	[redacted]
In this section we provide a formal description of prioritized logic programs (PLPs).	[redacted]
Our language L includes the following vocabulary:	[redacted]
- Variables: x, y, z, ■ ■ •. - Constants: C, Ci, C2, • • •, including logical constants True and False.	[redacted]
- Predicates: P,Q, R, ■ ■ ■. - Names: N, N\, N2, ■ ■ ■■ - A strict partial ordering (i.e.	[redacted]

antireflexive, antisymmetric and transitive) < on names.	[redacted]
- A naming function A/", which maps a rule (see below) to a name.	[redacted]
- A symbol , which is used to represent a rule.	[redacted]
- Connectives -1 and not, where -1 represents the classical negation (strong negation), and not represents negation as failure (phủ định ngầm, phủ định như thất bại) (weak negation).	[redacted]
We also require that the sets of variables, constants, predicates and names be disjoint.	[redacted]
A term is either a constant or a variable. An atom (nguyên tố, nguyên tử) is P{t\, • • •, tk), where P is a predicate of arity k and t\, ■ ■ - ,tk are terms. A literal (trực kiện) is either an atom P or a negation of an atom —>P. A rule is a statement of the form Lq i Pi}"' 1 Lm,not• • *, not Ln,	[redacted] $L_0 \leftarrow L_1, \cdots, L_m, not\ L_{m+1}, \cdots, not\ L_n,$
where Li (0 < i < n) is a literal (trực kiện). LQ is the head of the rule, while Li, ■ ■ ■, Lm,not Lm.)_i, • • not Ln is the body of the rule. Obviously, the body of a rule could be empty.	[redacted]
A term, an atom, a literal, or a rule is ground if no variable occurs in it.	[redacted]
For the naming function A/", we require that for any rules r and r' in a PLP (see the following definition), Af{r) = Af(r') iff r and r' indicate the same rule.	[redacted]
An extended logic program II is a collection of rules [5]. A prioritized logic program (PLP) V	[redacted]

is a triplet (II, A/", <), where II is an extended logic program, Af is a naming function mapping each rule in II to a name, and < is a relation representing all strict partial orderings on names.	[redacted]
The following is an example of prioritized extended logic program. V\ = ({P <— not Q, not R, Q <— not P, R <— not P}, {Af(P <— not Q, not R) = Ni, N{Q <r- not P) = N2, Af(R P) = A3}, {Nx < N2,N2 < Ns}).	[redacted] $\mathcal{P}_1 = (\{P \leftarrow not\, Q,\, not\, R,\, Q \leftarrow not\, P,\, R \leftarrow not\, P\}, \{\mathcal{N}(P \leftarrow not\, Q,\, not\, R) = N_1, \mathcal{N}(Q \leftarrow not\, P) = N_2, \mathcal{N}(R \leftarrow not\, P) = N_3\}, \{N_1 < N_2, N_2 < N_3\})$.
To simplify our presentation, we usually represent V\ as the following form: Vn Ni : P <— not Q, not R, A2 : Q <— not P, A3 : R <— not P, A1 < A2, N 2 < A3.	[redacted] $\mathcal{P}_1$: $N_1 : P \leftarrow not\, Q,\, not\, R,$ $N_2 : Q \leftarrow not\, P,$ $N_3 : R \leftarrow not\, P,$ $N_1 < N_2, N_2 < N_3.$
We also use notations V\ (II), V\(A/"), and 7?i(<) to denote the sets of rules, naming function's values and <-relation of V\ respectively.	[redacted]
Consider the following program: v2-. N\ : P <— not Q, not R, A2 : Q <— P, A3 : i? <— no£ P, N1 < A2, A2 < A3, Ai < A3.	[redacted] $\mathcal{P}_2$: $N_1 : P \leftarrow not\, Q,\, not\, R,$ $N_2 : Q \leftarrow not\, P,$ $N_3 : R \leftarrow not\, P,$ $N_1 < N_2, N_2 < N_3, N_1 < N_3.$
Obviously, the only difference between V\ and V2 is that there is one more relation Ai < A3 in V2. As we mentioned earlier, since < is a strict partial ordering (i.e., antirellexive, antisymmetric and	[redacted]

transitive), we would expect that V\ and V2 are identical in some sense. Furthermore, if we rename rules in V2 as follows,
v'2-
N[: P <— not Q, not R,
N2 : Q <— not P,
A3 : R <— not P,
A' < A', A' < A', A' < A',
V'2 would be also identical to V2 and hence to V\ too from our intuition. To make this precise, we first introduce <-closure as follows.

Definition 1 Given a program V = (II, A", <). V(< +) is the <-closure ofV iffV{< +) is the smallest set containing V(<) and closed under transitivity.
We also need to define a renaming function as follows. A renaming function Rn maps a PLP V = (II, A”, <) to another PLP V , i.e. Rn(V) = V' = (IT, A/7, <'), such that (i) V{Ii) = ^'(n'); (ii) for each rule r £ P(II)1, J\f (r) = A G V(AI’) iff Af(r) = A' G V'(Afr) (A and A' are not necessarily different); (iii) for any rules r\ and r2 in P(II), M{r\) = N\, Nir?) = N2 G P(A0, and JVj < JV2 € V{<) iff N'in) = N[, Af'fa) = G P'(Af'), and< ^2 ^ 7,/(</)- ^ 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 V\ and V2 are identical iff there exists a renaming function Rn, mapping V2 to V'2 such that Pi (II) = P'2(n'), Pi(A0 = and Vl{<+) = P'2(</+).
We have defined that a prioritized

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

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

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 V, relation Af{r) < Af(r') holds, rule r would be preferred to apply over rule r' during the evaluation of V (i.e. rule r is more preferred than rule r'). Consider the following classical example represented in our formalism:

V3:

Nx : Fly(x) <- - Bird(x), not ^Fly{x),

n2 : ^Fly(x) Penguin(x) not Fly{x

n3 : Bird(Fweety) ,

n4 : Penguin (Fweety) ,

n2 < Nx.

Obviously, rules N\ and N2 conflict with each other as their heads are complementary literals , and applying N\ will defeat A^ and vice versa. However, as N2 < N1, we would expect that rule N2 is preferred to apply first and then defeat rule N1 after applying N2 so that the desired solution ~^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

[redacted]

$\mathcal{P}_3$:

$N_1 : Fly(x) \leftarrow Bird(x),\ not\ \neg Fly(x),$

$N_2 : \neg Fly(x) \leftarrow Penguin(x),\ not\ Fly(x),$

$N_3 : Bird(Tweety) \leftarrow,$

$N_4 : Penguin(Tweety) \leftarrow,$

$N_2 < N_1.$

[redacted]

[redacted]

[redacted]

Logic Programs: A Review

Let II be an extended logic program. For simplicity, we treat a rule r in II with variables as the set of all ground instances (thực thể, thể hiện, thể nghiệm) of r formed from the set of ground literals of the language of II. We will also adopt this assumption in our prioritized extended logic programs. In the rest of paper, we will not explicitly declare this assumption whenever there is no ambiguity in our discussion.

[redacted]

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

[redacted]

(i) for any rule LQ <— L1, • • •, Lm from II, if L1, • • •, Lm G S, then LQ G S]

[redacted]

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

[redacted]

Now consider II be an extended logic program. For any subset S of Lit, let II5 be the logic program obtained from II by deleting

[redacted]

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

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

[redacted]

We define that S is an answer set of II, denoted Ans(II), iff S is an answer set of II5, i.e. S = Aras(II5).

[redacted]

Consider V3 presented in last section. It is not difficult to see that extended logic program ^(II) has two answer sets: {Bird(Tweety), Penguin(Tweety),

[redacted]

-1Fly(Tweety)} and {Bird(Tweety), Penguin(Tweety), Fly(T weety)}. 3.2 Answer Sets for PLPs In program V3, we have seen that rules N1 and N2 conflict with each other. Since N2 < N\, we try to solve the conflict by applying N2 first and defeating N1. 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: N\ : P <— not Qi, N2 : -i-P not Q2, Ni < N2.	[redacted] $\mathcal{P}_4$: $N_1 : P \leftarrow not\ Q_1,$ $N_2 : \neg P \leftarrow not\ Q_2,$ $N_1 < N_2.$
Although heads of N1 and N2 are complementary literals, applying N1 will not affect the applicability of N2 and vice versa. Hence N1 < N2 should not be taken into account during the evaluation of V4. The following definition provides a formal description for this intuition.	[redacted]
Definition 2 Let n be an extended logic program and r a rule with the form Lq <— L1, • • •, Lm, not Lm+1, • • •, not Ln (r does not necessarily belong to II). Rule r is defeated by II iff for any answer set Ans(II) of II, there exists some Li G Ans(II), where m + I < i < n.	[redacted]

Now our idea of evaluating a PLP is as follows. Let V = (II, A/", <). If there are two rules r and r' in P(II) and Af(r) < Af(r'), r' will be ignored in the evaluation of V, only if keeping r in P(II) and deleting r' from P(II) will result in a defeat of r', i.e. r' is defeated by P(II) — {r'}. By eliminating all such potential rules from P(II), V is eventually reduced to an extended logic program in which the partial ordering < has been removed. Our evaluation for V is then based on this extended logic program.	[redacted]
Let us consider program V3 once again. Since N2 < N1 and A^i is defeated by V3 — {A^i} (i.e. the unique answer set of V3 — {A^i} is {Bird(Tweety), Penguin(Tweety), —>Fly(Tweety)}), rule A^i should be ignored during the evaluation of V3. For program V4, on the other hand, although A^i < N2, relation A^i < N2 will not affect the solution of evaluating V4 as ^(II) —{A^} does not defeat N2 (i.e. the unique answer set of ^(II) — {A^} is {-P}).	[redacted]
…………………………………… Definition 3 Let V = (II, A/", <) be a prioritized extended logic program. V< is a reduct of V with respect to < if and only if there exists a sequence of sets IIj- (i = 0,1, • • ■) such that: (i) n0 = II; (it) — n8—1 (fl') there exists r (z II%—1 such that for every j (j = 1, • • •, k), Af{r) < M{rj) G V(< +) and ri, • • •, rfc	[redacted] [redacted] [redacted]

are defeated by IIj_i — {ri, • • •, r^}, and (b) there does not exist a rule r' G n8_i such that N(ri) < N(r') for some j (j = 1, • • •, k) and rJ is defeated by IIj_i — {r'}}; (in) p< = a=0n8-	[redacted] [redacted]
In Definition 3, clearly V< is an extended logic program obtained from II by eliminating some rules from II. In particular, if Af(r) < Af (rr) and II — {r'} defeats r', then rule r' will be eliminated from II 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 Af(r) < Af (rr) in the <-closure of V.	[redacted]
Example 1 Using Definition 1 and 3, it is not difficult to conclude that V\, V3 and V4 have unique reducts as follows respectively: Vi = {P <— not Q, not R}, = {-iFly(x) <— Penguin(x), not Fly(x), Bird(Tweety) , Penguin(Tweety) <—}, v< = p4(n).	[redacted] $\mathcal{P}_1^< = \{P \leftarrow not\, Q,\, not\, R\},$ $\mathcal{P}_3^< = \{\neg Fly(x) \leftarrow Penguin(x),\, not\, Fly(x),$ $Bird(Tweety) \leftarrow, Penguin(Tweety) \leftarrow\},$ $\mathcal{P}_4^< = \mathcal{P}_4(\Pi).$
It is quite obvious to note that the reduct of a PLP may not be unique as the following example shows.	[redacted]
Example 2 Consider a PLP V5: V5: Ni : P A2 ■ Q not R, A3 : T A4 : R <— not Q, A1 < A^2, A3 < A4.	[redacted] $\mathcal{P}_5$: $N_1 : P \leftarrow,$ $N_2 : Q \leftarrow not\, R,$ $N_3 : T \leftarrow,$ $N_4 : R \leftarrow not\, Q,$ $N_1 < N_2, N_3 < N_4.$
According to Definition 3, it is easy to see that V5 has two reducts: {P , T R <— not Q} and	[redacted]

{P , Q n°t R, T <—}. ■

We should also mention that the condition (b) in the construction of 11; in Definition 3 is necessary. Without ths condition, some unintuitive results may be derived. For instance, if we have additional preference information A3 < A2 in program V3, then using a modified version of Definition 3 without condition (b) in the construction of 11;, we will conclude that {Fly(x) <— Bird(x), not -1 Fly(x), Bird(Tweety) , Penguin(Tweety) <—} is also a reduct of V3, 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 V = (II, A/", <) be a PLP and Lit the set of all ground literals in the language of V. For any subset S of Lit, S is an answer set of V, denoted as Ansp(V), iff S = Ans(V<) for some reduct P< of V.

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

Ansp(V1) = {P},

Ansp (V3) = {Bird{F weety), Penguin(Fweety), ~^Fly(Fweety)},

Ansp (V4) = Lit,

and two answer sets for V5:

Ansp(V5) = {P,R,F},

Ansp{V5) = {P,Q,F},

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

3.3 More Examples

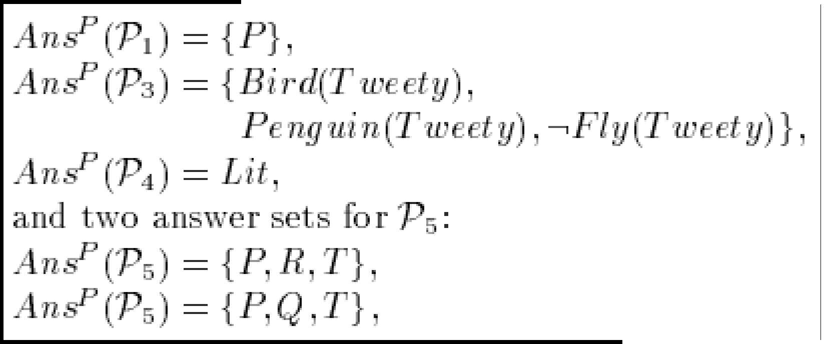
Now let us examine more examples to illustrate some

[redacted]

[redacted]

[redacted]

[redacted]


$Ans^P(\mathcal{P}_1) = \{P\}$,

$Ans^P(\mathcal{P}_3) = \{Bird(Tweety),$
$Penguin(Tweety), \neg Fly(Tweety)\}$,

$Ans^P(\mathcal{P}_4) = Lit$,

and two answer sets for $\mathcal{P}_5$:

$Ans^P(\mathcal{P}_5) = \{P, R, T\}$,

$Ans^P(\mathcal{P}_5) = \{P, Q, T\}$,

[redacted]

[redacted]

features of PLPs. Example 4 Let VQ be: V6: N\ : P <— not Q, N2 : -i-P <— not P, N2 < Ai.	[redacted] $\mathcal{P}_6$: $N_1 : P \leftarrow not\, Q,$ $N_2 : \neg P \leftarrow not\, P,$ $N_2 < N_1.$
This program was originally presented in [2]. The reduct of Vq is Vq{T1). So the unique answer set of Vq is {P}. Note that even if the heads of N\ and N2 are complementary literals and N2 < N\, rule Ni can not be deleted from ^(n) as ^(n) — {A^i} does not defeat N\. ■	[redacted]
Example 5 Let V7 be: vr. N\ : P <— not Q, N2 : Q <— not P, : R <— not Q, not S, N4 : S <— not R, N1 < N2, N3 < N4.	$\mathcal{P}_7$: $N_1 : P \leftarrow not\, Q,$ $N_2 : Q \leftarrow not\, P,$ $N_3 : R \leftarrow not\, Q,\ not\, S,$ $N_4 : S \leftarrow not\, R,$ $N_1 < N_2, N_3 < N_4.$
^(II) has three answer sets: {P, R}, {Q, S}, and {P, S}. The unique reduct of V7 is {P <— not Q, R <— not Q, not S}. Therefore, the unique answer set of V7 is Ansp (V7) = {P,R}. Note that this solution is consistent with our intuition since N1 < N2 and < N4, and applying N1 and causes N2 and N4 inapplicable respectively. ■	[redacted]
Example 6 Let Vs be: Vs: N\ : P <— not Q, not R, N2 : Q <— not P, : R <— not P, Ni < N2.	[redacted] $\mathcal{P}_8$: $N_1 : P \leftarrow not\, Q,\ not\, R,$ $N_2 : Q \leftarrow not\, P,$ $N_3 : R \leftarrow not\, P,$ $N_1 < N_2.$
^(n) has two answer sets {P} and {Q,R}. Obviously, N2 is not defeated by Ps(n) — {^2} as P only belongs to one of two answer sets {P} and {i?} of Ps(n) — {^2}.	[redacted]

Therefore, the unique reduct Vg of Vs is the same as Ps(n). So Vs has two answer sets {P} and {Q, R}.

It is worth observing that if we add N1 < to Vs, Vs will then has a unique answer set {P}. On the other hand, if we add < N1 instead of N\ < N%, the unique answer set of Vs 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:

V9:
Ni : -iEmployed(x) <— Student(x),
no£ Employed(x),
A^2 : Employed(x) <— Age(x, > 25),
no£ -iEmployed(x),
: Student(x) <— FT-Student(x),
W4 : Student(x) <— PT-Student(x),
N5 : Student(Peter) ,
A6 : Age(Peter1 > 25) .

Obviously, Pg has two answer sets, from one we conclude -iEmployed(Peter) and from the other we conclude Employed(Peter). By specifying

[redacted]

[redacted]

[redacted]

$\mathcal{P}_9$:

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

[redacted]

Ni < A2 or N2 < N\, we can retain one answer set and exclude the other, and hence resolve the conflict as it may occur. However, the reason for specifying Ni < N2 (or N2 < Ni) rather than N2 < (or A^i < N2) 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 1 is a full-time student, then N1 is more preferred than N2, while if 2; is a part-time student, then N2 is more preferred than N\”. 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.

[redacted]

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.

[redacted]

4.1 Syntax of DPLPs

[redacted]

A language CD of DPLPs is a language £ of PLPs except the following modifications:

- Variables consist of variables x, y, z, ■ ■ ■ of C and name variables n, n\, n2, ■ ■ •, where {x, y, z, ■ ■ •} and {n, n\, n2, ■ ■ •} are disjoint.

[redacted]

- Constants consist of constants C, Ci, C2, • • • of C and

[redacted]

name constants N, N\, N2, ■ ■ •, where {C, Ci, C2, ■ ■ ■} and {TV, N\, N2, ■ ■ •} are disjoint.

- Names consist of name variables and name constants. The naming function Af 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 CP. Since < is a special predicate in CP, < can occur in any rules of CP. For instance, N\ < N2 <— not N2 < N\ and n\ < n2 <— P(rai), Q(n2), not n2 < n\ could be valid rules of CD.

A dynamic PLP (DPLP) is a pair V = (II, A/"), where II is a collection of rules of CD and Af is a naming function that maps each rule of II to a name constant. Given a DPLP V, LitD denotes the set of all ground literals of the language CD of V.

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

< n3 <r- nt < n2, n2 < n3, (1)

-w2 < ni ni < n2 . (2)

Example 7 Consider a DPLP as follows.

'Pio-

N\ : -iEmployed(x) <— Student (x),

no£ Employed(x),

A^2 : Employed(x) <— Age(x, > 25),

no£ -iEmployed(x),

N3 : Student (x) <— FT-Student(x),

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

$$n_1 < n_3 \leftarrow n_1 < n_2, n_2 < n_3, \qquad (1)$$

$\neg n_2 < n_1 \leftarrow n_1 < n_2$[5]. (2)

[redacted]

$\mathcal{P}_{10}$:

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

W4 : Student (x) <— PT-Student(x),
: FT-Student [Peter) ,
7V6 : Age{Peter1 > 25) ,
W7 : TVi < A^2 <— FT-Student(x),not N2 < ./Vi,
A's : A^2 < <— PT-Student(x),not N\ < A^.

Pio is similar to Pg except that there are two rules A7 and TVs in ^10 which express conditional preferences and we know Peter is a full-time student (i.e. rule N5) . Intuitively, rule N7 is interpreted as “if a; is a full-time student and there is no explicit knowledge to conclude that N2 is more preferred than Ni, then rule N1 is more preferred than rule N2”. A similar interpretation can be stated for Ng. 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 CP under a sequence of reductions with respect to the partial ordering < .

Before to present our method formally, we first introduce some useful notations. Firstly, a DPLP V = (II, Af) can be treated as a special PLP in language CP with the form (II, Af, <o), where <0= 0- Generally, we say that a logic program V is a PLP in language CD if the program is specified as

[redacted]

[redacted]

[redacted]

[redacted]

the form (II, Af, <), where II is a set of rules of CP, Af is a naming function mapping each rule in II to a name constant and < is a set of ground atoms of the form N < N'. In this case, we can compute each of P's PLP answer sets, denoted as Ansp,D(V), by using the approach proposed in section 3. Consider the following program.	[redacted]
Vn. iVi : P not Q, n2 : Q <— not P, n3 :NX<N2^- not N2 < A^i n4 :N2<Nt^ not N\ < N2 n3 < N4.	$\mathcal{P}_{11}$: $N_1 : P \leftarrow not\, Q,$ $N_2 : Q \leftarrow not\, P,$ $N_3 : N_1 < N_2 \leftarrow not\, N_2 < N_1,$ $N_4 : N_2 < N_1 \leftarrow not\, N_1 < N_2,$ $N_3 < N_4.$
Note that Vn is not a DPLP but a PLP in language CD as N3 < N4 is specified from outside of the rules of V\\. Clearly, under the answer set semantics of PLPs, Vu has two answer sets: {P, N\ < A^} and {Q, N\ < A2}.	[redacted]
Now we give the formal descriptions of the semantics of DPLPs.	[redacted]
Definition 5 Let V = (II, Af) be a DPLP. A PLP V* in CP is a transformation of V to PLP if and only if there exists a sequence of sets \P4- (i = 0,1, • • ■) such that	[redacted]
(i) = (n0, A/", <o), where n0 = II and <0= 0;	[redacted]
(ii) vpj = (11;, Af, <i), where 11; = is a reducfe with respect to <;_i, and <i= {A < N' | N < N' belongs to all PLP answer sets of^i-i};	[redacted]
(in) V* = (IIoo, Af, <oo).	$(iii)\ \mathcal{P}^* = (\Pi_\infty, \mathcal{N}, <_\infty).$

Definition 6 Let V = (II, Af) be a DPLP. For any subset S of LitD, S is an answer set of V, denoted as AnsD(fP), iff S = Ansp,D (fP*) for some transformation V* of V to PLP as defined in Definition 5.

Let us examine the above definitions more closely. The basic idea of evaluating a DPLP V is to transform V to a corresponding PLP V* in CP, and then use the PLP answer set semantics to evaluate V* (eg. Definition 6). From Definition 5, we can see that during this transformation, V is first treated as a special PLP in £D, i.e. \Po- Then for each i (i = 1,2, •••), \J/,- = (11;, Af, <i) is a PLP in CD and generated from lI,;_i = (II;_i, Af, <;_ 1). Intuitively, <4- presents all ground instances of the partial ordering < derived from \Pi_i under the semantics of PLP, and II; is a reduct of \Pi_i by eliminating defeated rules from IIj_i with respect to <;_i (eg. II; =

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 V12 as follows.

Pi2'-

N\ : P <— not Q,

N2 '■ Q not P,

N3 : N\ <C N2 i— not N2 <C N\,

N4 : N2 ^ -^1 ^— not Ni <C N2 >

N5:N3< N4

According to Definition 5, we get the following sequence of \Pj- (i = 1,2, 3) : tfi:

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

$\mathcal{P}_{12}$:

$N_1 : P \leftarrow not\, Q,$

$N_2 : Q \leftarrow not\, P,$

$N_3 : N_1 < N_2 \leftarrow not\, N_2 < N_1,$

$N_4 : N_2 < N_1 \leftarrow not\, N_1 < N_2,$

$N_5 : N_3 < N_4 \leftarrow .$

[redacted]

N\ : P <— not Q,
N2 : Q <— not P,
N3 : N\ <C N2 i— not N2 <C N\,
N4 : N2 ^ i— not A^l <C N2 >
N5:N3< N4 N3 < N4.
®2:
: P <— no£ Q,
A^2 : Q <— P,
A3 : AT^ <C A^2 ^A^2 <C AT^ ,
N5:N3< N4 A^i < A^2, A3 < W4.
®3:
Ni : P <r- not Q,
A3 : A^l <C A^2 ^— ?io£ A^2 <C AT^,
N5:N3< N4 A^i < A^2, A3 < W4.

Then it is easy to verify that ^4 = ^3. So the transformation of V12 to PLP is VI2 = 'I’s - Therefore, from Definition 6, V12 has a unique answer set AnsD{V12) = {P, Ni < N2, N3 < AT4, -.AT2 < ATX, -.AT4 < N3}u. m

Example 9 Example 7 continued. Ignoring the detail, it is not difficult to see that V\o has a unique answer set AnsD (V10) = {Age(Peter, > 25), FT-Student(Peter), Student(Peter), -1 Employed(Peter), Ni < A^2, -iA^2 < Ari}, which presents the desired result for Viq. ■

Example 10 Consider a DPLP V\s as follows.
V13:
Nx ni < n2 P(ni), Q(n2), not n2 <
n2 R(C)<- not R(C),
n3 R(C') <r- not R(C),
n4 P(N2) <r ~ 1
n5 Q(n3) *-

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

[redacted]

$Ans^D(\mathcal{P}_{12}) = \{P, N_1 < N_2, N_3 < N_4, \neg N_2 < N_1, \neg N_4 < N_3\}^{11}$. ■

[redacted]

[redacted]

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

This program is a bit different from those DPLPs discussed above. Intuitively, Ni can be viewed as a general rule about the preference of the domain - “for any two rules n\ and n2, if n\ and n2 satisfy properties P and Q respectively, and there is no explicit knowledge to conclude that n2 is more preferred than ni, then n\ is more preferred than n2”, while N2 - N$ present explicit knowledge of the domain . Clearly, using the approach described above, V\s has a unique answer set {R(C), P(N2), Q(N3), N2 < N%, ~^N3 < 7V2}. ■

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 II be an extended logic program. We use AArS'(II) to denote the class of answer sets of II. Suppose V = (II, Af, <) is a PLP. From Definition 3, we can see that a reduct V< of V is generated from a sequence of extended logic programs: II = IIo, IIi, II2, • • •• We use notation {11;} (i = 0,1, 2, • • •) to denote this sequence and call it a reduct chain of V. Then we can prove the following useful result .

Theorem 1 Let V = (II, Af, <) be a PLP, and {II;} (i = 0,1, 2, • • ■) a reduct chain ofV. Suppose each II; has answer setfs). Then for any i and j where i < j, ANSiUj) C

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

ANS(Ui).	
Theorem 1 shows an important property of the reduct chain of V: each II; is consistent with II;_i but becomes more specific than II;_i in the sense that all answer sets of II; are answer sets of II;_i but some answer sets of II; 1 are filtered out if they conflict with the preference partial ordering <.	[redacted]
The following theorem shows the answer set relation between a PLP and its corresponding extended logic programs.	[redacted]
Theorem 2 Let V = (II, Af, <) be a PLP. Then a subset S of Lit is an answer set ofV iff S is an answer set of each II; for some reduct chain {n.-} (i = 0,1, 2, • • ■) ofV, where each II; has answer set(s).	[redacted]
Now we investigate properties of DPLPs. Let V and V' be a DPLP and a PLP in CP respectively. We use ANSP,D {V') to denote the class of PLP answer sets of V'. From Definition 5, we can see that a transformation V* of V is generated from a sequence of PLPs in CD: \Po, ^i> ^2> • • •• We use notation {'I';} (i = 0,1,2, •••) to denote this sequence and call it a PLP- reduct chain of V. Then, similarly to the case of PLPs described earlier, we have the following results for DPLPs.	[redacted]
Theorem 3 Let V = (II, M) be a DPLP, and {'I';} (i = 0,1, 2, • • ■) a PLP- reduct chain of V. Suppose each has PLP answer set(s). Then for any i and j where i < j, ANSP,D(\Pj) C ANS^H^i).	[redacted]
Theorem 4 Let V = (II, M) be a DPLP. Then a subset S of LitD is	[redacted]

an answer set of V iff S is a PLP answer set of each for some PLP-reduct chain {'I';} (i = 0,1, 2, • • ■) ofV, where each has PLP answer set(s).

[redacted]

6 Related Work and Conclusions

[redacted]

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]).

[redacted]

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.

[redacted]

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

[redacted]

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.

[redacted]

[redacted]