Towards Generalized Rule-based Updates

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
NSW 2052, Australia
E-mail: norman@cse.unsw.edu.au

Abstract

Recent work on rule-based updates provided new frameworks for updates in more general knowledge domains [Marek and Truszczyński, 1994; Baral, 1994; Przymusinski and Turner, 1995]. In this paper, we consider a simple generalization of rule-based updates where incomplete knowledge bases are allowed and update rules may contain two types of negations. It turns out that previous methods cannot deal with this generalized rule-based update properly. To overcome the difficulty, we argue that necessary preferences between update rules and inertia rules must be taken into account in update specifications. From this motivation, we propose prioritized logic programs (PLPs) by adding preferences into extended logic programs [Gelfond and Lifschitz, 1991]. Formal semantics of PLPs is provided in terms of the answer set semantics of extended logic programs. We then show that the procedure of generalized rule-based update can be formalized in the framework of PLPs. The minimal change property of the update is also investigated.

1 Introduction

Marek and Truszczyński's recent work on rule-based updates [Marek and Truszczyński, 1994] provided a new framework for updates in more general knowledge domains. Generally, they addressed the following problem: given an initial *knowledge base* $\mathcal{B}$, i.e. a set of ground atoms, and a set of *update rules* $\mathcal{P}$ with the forms[1]:

$$in(A) \leftarrow in(B_1), \cdots, in(B_m), out(C_1), \cdots, out(C_n), \tag{1}$$

$$out(A) \leftarrow in(B_1), \cdots, in(B_m), out(C_1), \cdots, out(C_n), \tag{2}$$

where $A, B_1, \cdots, B_m, C_1, \cdots, C_n$ are ground atoms, what is the resulting knowledge base $\mathcal{B}'$ after updating $\mathcal{B}$ by $\mathcal{P}$? The intuitive meaning of (1) (or (2)) is that if $B_1, \cdots, B_m$ are *in* the knowledge base, and $C_1, \cdots, C_n$ are *not in* the knowledge base, then A should be (or not be) in the knowledge base.

For example, given a knowledge base $\mathcal{B} = \{A, D\}$ and a set of update rules $\mathcal{P} = \{in(C) \leftarrow in(A), out(B), out(D) \leftarrow in(C), out(B)\}$, where A, B, C and D are ground atoms, then after updating $\mathcal{B}$ by $\mathcal{P}$, according to Marek and Truszczyński's approach, we would expect to have a resulting knowledge base $\mathcal{B}' = \{A, C\}$.

Relationships between rule-based updates and logic programming have been studied by Baral [Baral, 1994] and Przymusinski and Turner [Przymusinski and Turner, 1995]. In particular, they showed that Marek and Truszczyński's formal procedure of specifying $\mathcal{B}'$ can be reduced to a computation of the answer set of a corresponding extended logic program (we will review this procedure in next section). However, there are two limitations with Marek and Truszczyński's rule-based update: the initial knowledge base should be complete, i.e. any ground atom not in the initial knowledge base is treated as its negation; and update rules only contain classical negations. For instance, following ideas of [Baral, 1994] and [Przymusinski and Turner, 1995], rules (1) and (2) are translated into the following inference rules respectively in the corresponding extended logic program:

$$A \leftarrow B_1, \cdots, B_m, \neg C_1, \cdots, C_n,$$
$$\neg A \leftarrow B_1, \cdots, B_m, \neg C_1, \cdots, C_n.$$

In this paper, we consider a simple generalization of rule-based updates where a knowledge base can be *incomplete*, eg. a set of ground literals, and update rules have the following form:

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

where each L_i $(0 \leq i \leq n)$ is a literal, and *not* represents *negation as failure* (or called weak negation). As a literal can be a negative atom, the above rule actually may contain two types of negations, i.e. classical and weak

[1] $\mathcal{P}$ was called a *revision program* in [Marek and Truszczyński, 1994].

negations. The intuitive semantics of this rule can be interpreted as follows: if facts $L_1, \cdots, L_n$ are true in the knowledge base, and there are no explicit representations saying that facts $L_{m+1}, \cdots, L_n$ are true in the knowledge base, then fact L_0 should be true in the knowledge base.

Such generalized rule-based update is important in many applications. For example, in a secure computer system, a formal specification of users' access rights is usually required, and the access policy of the system can be represented by a knowledge base. An update of this knowledge base must be performed whenever new access control rules are applied to the system. Generally, two types of negations are needed to specify access control rules. Let $\mathcal{B} = \{Member(A, G),\ Member(B, G),\ Access(A, F),\ \neg Access(B, F)\}$ represent the current access policy of the system, where $Member(A, G)$ and $Member(B, G)$ mean that users A and B are members of group G, and $Access(A, F)$ and $\neg Access(B, F)$ indicate that user A can access file F and user B cannot access file F respectively. Suppose that new users C and D are added into group G and a global access control rule is now applied to each member of group G:

$$Access(x, F) \leftarrow Member(x, G), not\ \neg Access(x, F).$$

This rule actually says that any user belonging to group G can access file F unless it is explicitly stated that the user is not allowed to access F. Updating $\mathcal{B}$ by $\mathcal{P} = \{Member(C, G) \leftarrow,\ Member(D, G) \leftarrow,\ Access(x, F) \leftarrow Member(x, G), not\ \neg Access(x, F)\}$, from our intuition, we would expect that $Access(C, F)$ and $Access(D, F)$ are obtained, while facts $Access(A, F)$ and $\neg Access(B, F)$ remain persistent.

As we will see next, previous rule-based update approaches are not suitable to deal with this generalized rule-based update properly. To overcome the difficulty, we argue that necessary preferences between update rules and inertia rules with respect to the update must be taken into account. From this motivation, we propose prioritized logic programs (PLPs) by adding preferences into extended logic programs. Formal semantics of PLPs is provided in terms of the answer set semantics of extended logic programs. We then show that the procedure of generalized rule-based update can be formalized in the framework of PLPs. The minimal change property of the update is also investigated.

2 A Motivating Example

In this section we first review the concept of extended logic programs proposed by Gelfond and Lifschitz [Gelfond and Lifschitz, 1991] and then discuss an example of generalized rule-based update in the framework of extended logic programs.

2.1 Preliminaries

A language $\mathcal{L}$ of extended logic programs is determined by its object constants, function constants and predicates constants. *Terms* are built as in the corresponding first order language; *atoms* have the form $P(t_1, \cdots, t_n)$, where t_i $(1 \leq i \leq n)$ is a term and P is a predicate symbol of arity n; a *literal* is either an atom $P(t_1, \cdots, t_n)$ or a negative atom $\neg P(t_1, \cdots, t_n)$. A *rule* is an expression of the form:

$$L_0 \leftarrow L_1, \cdots, L_m, notL_{m+1}, \cdots, notL_n, \tag{3}$$

where each Li $(0 \leq i \leq n)$ is a literal. L_0 is called the *head* of the rule, while $L_1, \cdots, L_m, not\ L_{m+1}, \cdots, not\ L_n$ is called the *body* of the rule. Obviously, the body of a rule could be empty. A term, atom, literal, or rule is *ground* if no variable occurs in it. An *extended logic program* Π is a collection of rules. The following is an example of extended logic program Π_0:

$$\neg Employed(x) \leftarrow Student(x),\ not\ Employed(x),$$
$$Employed(x) \leftarrow Age(x, > 25), \neg Student(x),\ not\ \neg Employed(x).$$

To evaluate a extended logic program, Gelfond and Lifschitz proposed the answer set semantics for extended logic programs. 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. In the rest of paper, we will 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 let 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 an extended logic program Π_1 obtained from Π_0 by adding other two rules in Π_0: $\Pi_1 = \Pi_0 \cup \{\neg Student(Peter) \leftarrow,\ Age(Peter, > 25) \leftarrow\}$.

It is not difficult to see that Π_1 has a unique answer set set: $\{\neg Student(Peter),\ Age(Peter, > 25),\ Employed(Peter)\}$.

2.2 An Example

As we mentioned earlier, in previous formulations a specification of rule-based update can be represented by an extended logic program [Baral, 1994; Przymusinski and Turner, 1995]. By illustrating a simple example here, we will show that these methods are not suitable for specifying generalized rule-based updates.

Example 1 Suppose $\mathcal{B} = \{\neg A, B, C\}$ is a knowledge base, and $\mathcal{P} = \{\neg B \leftarrow not\ B, A \leftarrow C\}$ is a set of update rules. Consider an update of $\mathcal{B}$ by $\mathcal{P}$. Obviously, fact $\neg A$ should change to A by applying the second rule of $\mathcal{P}$. Fact B, on the other hand, seems persistent because B is true in the initial knowledge base, and $\neg B$ can only be derived from the first rule of $\mathcal{P}$ if fact B is absent from the current knowledge base. Therefore, from our intuition, the resulting knowledge base should be $\{A, B, C\}$.

Now we follow the principle of Baral and Przymusinski and Turner's methods [Baral, 1994; Przymusinski and Turner, 1995] to specify the above update procedure within an extended logic program[2]. Firstly, we need to extend the language of our domain by adding new propositional letters with the form $New\text{-}L$ if L is a propositional letter in the original language[3]. In this example, the extended language will include propositional letters $A, B, C, New\text{-}A$, $New\text{-}B$ and $New\text{-}C$. Then an extended logic program $\Pi(\mathcal{B}, \mathcal{P})$ is formed by the following rules:

Initial knowledge rules:
$\neg A \leftarrow$,
$B \leftarrow$,
$C \leftarrow$,
Inertia rules:
$New\text{-}A \leftarrow A, not\ \neg New\text{-}A$,
$New\text{-}B \leftarrow B, not\ \neg New\text{-}B$,
$New\text{-}C \leftarrow C, not\ \neg New\text{-}C$,
$\neg New\text{-}A \leftarrow \neg A, not\ New\text{-}A$,
$\neg New\text{-}B \leftarrow \neg B, not\ New\text{-}B$,
$\neg New\text{-}C \leftarrow \neg C, not\ New\text{-}C$,
Update rules:
$\neg New\text{-}B \leftarrow not\ New\text{-}B$,
$New\text{-}A \leftarrow New\text{-}C$.

Generally speaking, an answer set of program $\Pi(\mathcal{B}, \mathcal{P})$ represents a possible resulting knowledge base after updating $\mathcal{B}$ by $\mathcal{P}$, where literal $New\text{-}L$ in the answer set denotes the persistence of literal L if $L \in \mathcal{B}$, or a change of L if $\neg L \in \mathcal{B}$ or $L \notin \mathcal{B}$ with respect to this update.

It is easy to see that the above $\Pi(\mathcal{B}, \mathcal{P})$ has two answer sets: in one $New\text{-}B$ is true where in the other $New\text{-}B$ is false. Obviously this solution is not consistent with our previous observation. ■

Observing program $\Pi(\mathcal{B}, \mathcal{P})$, it is not difficult to see that a conflict occurs between inertia rule $New\text{-}B \leftarrow B$, $not\ \neg New\text{-}B$ and update rule $\neg New\text{-}B \leftarrow not\ New\text{-}B$, that is, applying $New\text{-}B \leftarrow B$, $not\ \neg New\text{-}B$ will defeat $\neg New\text{-}B \leftarrow not\ New\text{-}B$, and *vice versa*. This conflict leads $\Pi(\mathcal{B}, \mathcal{P})$ to have two different answer sets with an indefiniteness of $New\text{-}B$.

On the other hand, it seems that the inertia rule $New\text{-}B \leftarrow B$, $not\ \neg New\text{-}B$ should *override* the update rule $\neg New\text{-}B \leftarrow not\ New\text{-}B$ during the evaluation of $\Pi(\mathcal{B}, \mathcal{P})$ in order to obtain the desired solution. But this preference information cannot be expressed in Gelfond and Lifschitz's extended logic programs.

From the above discussion, we argue that to represent such generalized rule-based update properly, necessary preferences between inertia rules and update rules have to be taken into account during the evaluation of the update. We approach this problem from a general ground: we will first propose prioritized logic programs where preferences between rules can be explicitly expressed, and then formalize the generalized rule-based update in the framework of prioritized logic programs.

3 Prioritized Logic Programs (PLPs)

In this section we propose prioritized logic programs (PLPs) which extend Gelfond and Lifschitz's extended logic programs [Gelfond and Lifschitz, 1991] by adding preference information into programs. We first describe the syntax of PLPs and then provide an answer set semantics for PLPs.

3.1 Syntax

The language $\mathcal{L}^P$ of PLPs is a language $\mathcal{L}$ of extended logic programs just with the following augments:

- *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 to a name.

Terms, atoms, literals and rules in PLPs are defined as the same in extended logic programs. 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.

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

[2] The formalism used here, of course, is different from theirs.

[3] For simplicity, here we restrict the language to be propositional.

$<$ 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, $<$ 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 $r \in \mathcal{P}(\Pi)$[4], $\mathcal{N}(r) = N \in \mathcal{P}(\mathcal{N})$ iff $\mathcal{N}'(r) = N' \in \mathcal{P}'(\mathcal{N}')$ (N and N' are not necessarily 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.

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

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 an extended logic program by associating with a partial ordering $<$ to it. Intuitively such ordering represents a preference of applying rules during the evaluation of a query 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[5], and applying N_1 will defeat N_2 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.

3.2 Answer Sets for PLPs

Now we are ready to provide the semantics of PLPs. The semantics of PLPs is defined in terms of the answer set semantics of extended logic programs described earlier.

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.

[5]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.

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 described 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. We define a* reduct *of $\mathcal{P}$ with respect to $<$, denoted as $\mathcal{P}^<$, as follows.*

(i) $\Pi_0 = \Pi$;

(ii) $\Pi_i = \Pi_{i-1} - \{r_1, \cdots, r_k \mid$ *there exists* $r \in \Pi_{i-1}$ *such that* $\mathcal{N}(r) < \mathcal{N}(r_i) \in \mathcal{P}(<^+)$ $(i = 1, \cdots, k)$ *and* $r_1, \cdots, r_k$ *are defeated by* $\Pi_{i-1} - \{r_1, \cdots, r_k\}\}$;

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

In above definition, 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', rule r' is eliminated from Π. 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 $\mathcal{P}$. It is also not difficult to note that the reduct of a PLP may not be unique generally.

Example 2 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:

$$\begin{aligned}
&\mathcal{P}_0^< = \{P \leftarrow not\ Q\},\\
&\mathcal{P}_1^< = \{P \leftarrow not\ Q,\ not\ R\},\\
&\mathcal{P}_3^< = \{\neg Fly(x) \leftarrow Penguin(x),\ not\ Fly(x),\\
&\qquad Bird(Tweety) \leftarrow, Penguin(Tweety) \leftarrow\},\\
&\mathcal{P}_4^< = \mathcal{P}_4(\Pi).
\end{aligned}$$

■

Now it is quite straightforward to define the answer set 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 $S = Ans(\mathcal{P}^<)$.*

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

$$\begin{aligned}
&Ans^P(\mathcal{P}_0) = \{P\},\\
&Ans^P(\mathcal{P}_1) = \{P\},\\
&Ans^P(\mathcal{P}_3) = \{Bird(Tweety),\\
&\qquad Penguin(Tweety), \neg Fly(Tweety)\},\\
&Ans^P(\mathcal{P}_4) = Lit,
\end{aligned}$$

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

3.3 Basic Properties of PLPs

We now discuss some properties of PLPs. To simplify our presentation, let us introduce some useful notations. Let Π and $\mathcal{P}$ be an extended logic program and a PLP respectively. We use $ANS(\Pi)$ to denote the classes 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 solutions[6].

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 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).*

4 Generalized Rule-based Update

Consider a language $\mathcal{L}$ of extended logic programs as described in section 2.1. We specify that a *knowledge base* $\mathcal{B}$ is a set of ground literals of $\mathcal{L}$ and $\mathcal{P}$ is a set of rules of $\mathcal{L}$ with form (3) that are called *update rules*. Note that we allow a knowledge base to be *incomplete*. That is, a literal not in a knowledge base is treated as *unknown*.

We will use a prioritized logic program to specify an update of $\mathcal{B}$ by $\mathcal{P}$. For this purpose, we first need to extend language $\mathcal{L}$ by the following way. We specify $\mathcal{L}^P_{new}$ to be a language of PLPs based on $\mathcal{L}$ as described in section 3.1 with one more augment: For each predicate symbol P in $\mathcal{L}$, there is a corresponding predicate symbol New-P in $\mathcal{L}^P_{new}$ with the same arity of P.

[6] All proofs of theorems presented in this paper were given in our manuscript [Zhang and Foo, 1997].

To simplifying our presentation, in $\mathcal{L}^P_{new}$ we use notation New-L to denote the corresponding literal L in $\mathcal{L}$. For instance, if a literal L in $\mathcal{L}$ is $\neg P(x)$, then notation New-L simply means $\neg New$-$P(x)$. We use Lit_{new} to denote the set of all ground literals of $\mathcal{L}^P_{new}$. Clearly, $Lit_{new} = Lit \cup \{New\text{-}L \mid L \in Lit\}$. Now we are ready to formalize our generalized rule-based update.

Definition 5 *Let $\mathcal{B}$, $\mathcal{P}$, $\mathcal{L}$ and $\mathcal{L}^P_{new}$ be defined as above. The* specification of updating $\mathcal{B}$ by $\mathcal{P}$ *is defined as a PLP of $\mathcal{L}^P_{new}$, denoted as $Update(\mathcal{B},\mathcal{P}) = (\Pi(\mathcal{B},\mathcal{P}), \mathcal{N}, <)$, as follows:*

1. *$\Pi(\mathcal{B},\mathcal{P})$ consists of following rules:*
   Initial knowledge rules*: for each L in $\mathcal{B}$, there is a rule $L \leftarrow$;*
   Inertia rules*: for each predicate symbol P in $\mathcal{L}$, there are two rules:*
   New-$P(x) \leftarrow P(x)$, *not* $\neg New$-$P(x)$[7], *and*
   $\neg New$-$P(x) \leftarrow \neg P(x)$, *not* New-$P(x)$,
   Update rules*: for each rule*
   $L_0 \leftarrow L_1, \cdots, L_m$, *not* $L_{m+1}, \cdots$, *not* L_n *in* $\mathcal{P}$, *there is a rule*
   New-$L_0 \leftarrow New$-$L_1, \cdots, New$-L_m,
   not New-$L_{m+1}, \cdots$, *not* New-L_n;
2. *Naming function $\mathcal{N}$ assigns a unique name N for each rule in $\Pi(\mathcal{B},\mathcal{P})$;*
3. *For any inertia rule with name N and update rule with name N', there is a partial ordering between N and N': $N < N'$.*

Comparing Definition 5 with the update specification described in Example 1, we can see that the difference between these two approaches is that in our formulation preference relations between inertia and update rules are explicitly expressed. We specify inertia rules to be more preferred than update rules in $Update(\mathcal{B},\mathcal{P})$.

The intuitive idea behind this is that a preference ordering between an inertia rule and an update rule in $Update(\mathcal{B},\mathcal{P})$ will affect the evaluation of $Update(\mathcal{B},\mathcal{P})$ *only if* these two rules conflict with each other, eg. applying one rule causes the other inapplicable. On the other hand, a fact in the initial knowledge based $\mathcal{B}$ is always preferred to persist during an update whenever there is no violation of update rules[8]. Therefore, when conflicts occur between inertia and update rules, inertia rules should override the corresponding update rules. Otherwise, the preference ordering does not play any role in the evaluation of $Update(\mathcal{B},\mathcal{P})$. Also note that there will be at most $2k \cdot l$ instances of relation $<$ in $Update(\mathcal{B},\mathcal{P})$, where k is the number of predicate symbols of $\mathcal{L}$ and l is the number of update rules in $\mathcal{P}$.

[7] x might be a tuple of variables.

[8] Note that an update rule in $Update(\mathcal{B},\mathcal{P})$ is defeasible if it contains a weak negation *not* in the body.

Finally, on the basis of Definition 5, we can formally define a knowledge base $\mathcal{B}'$ resulting from updating $\mathcal{B}$ by $\mathcal{P}$ in a straightforward way.

Definition 6 *Let $\mathcal{B}, \mathcal{P}, \mathcal{L}$ and $\mathcal{L}^P_{new}$ be specified as before, and $Update(\mathcal{B},\mathcal{P})$ the specification of updating $\mathcal{B}$ by $\mathcal{P}$ as defined in Definition 5. A set of ground literals of $\mathcal{L}$, $\mathcal{B}'$, is called a* possible resulting knowledge base *with respect to the update specification $Update(\mathcal{B},\mathcal{P})$, iff $\mathcal{B}'$ satisfies the following conditions:*

1. *if $Update(\mathcal{B},\mathcal{P})$ does not have an answer set, then $\mathcal{B}' = \mathcal{B}$;*
2. *if $Update(\mathcal{B},\mathcal{P})$ has a consistent answer set, say $Ans^P(Update(\mathcal{B},\mathcal{P}))$, then*
   $$\mathcal{B}' = \{L \mid New\text{-}L \in Ans^P(Update(\mathcal{B},\mathcal{P}))\};$$
3. *$\mathcal{B}' = Lit$ if $Ans^P(Update(\mathcal{B},\mathcal{P})) = Lit_{new}$.*

Example 4 Example 1 continued. Let $\mathcal{B} = \{\neg A, B, C\}$ and $\mathcal{P} = \{\neg B \leftarrow not\ B,\ A \leftarrow C\}$. From Definition 5, the specification of updating $\mathcal{B}$ by $\mathcal{P}$, $Update(\mathcal{B},\mathcal{P})$, is as follows:

Initial knowledge rules:
$N_1 : \neg A \leftarrow$,
$N_2 : B \leftarrow$,
$N_3 : C \leftarrow$,
Inertia rules:
$N_4 : New\text{-}A \leftarrow A, not\ \neg New\text{-}A$,
$N_5 : New\text{-}B \leftarrow B, not\ \neg New\text{-}B$,
$N_6 : New\text{-}C \leftarrow C, not\ \neg New\text{-}C$,
$N_7 : \neg New\text{-}A \leftarrow \neg A, not\ New\text{-}A$,
$N_8 : \neg New\text{-}B \leftarrow \neg B, not\ New\text{-}B$,
$N_9 : \neg New\text{-}C \leftarrow \neg C, not\ New\text{-}C$,
Update rules:
$N_{10} : \neg New\text{-}B \leftarrow not\ New\text{-}B$,
$N_{11} : New\text{-}A \leftarrow New\text{-}C$,
$<$:
$N_4 < N_{10}, N_5 < N_{10}, N_6 < N_{10}$,
$N_7 < N_{10}, N_8 < N_{10}, N_9 < N_{10}$,
$N_4 < N_{11}, N_5 < N_{11}, N_6 < N_{11}$,
$N_7 < N_{11}, N_8 < N_{11}, N_9 < N_{11}$.

Now from Definitions 3 and 4, it is not difficult to see that $Update(\mathcal{B},\mathcal{P})$ has a unique answer set: $\{\neg A, B, C, New\text{-}A, New\text{-}B, New\text{-}C\}$. Note that in $Update(\mathcal{B},\mathcal{P})$, only ordering $N_5 < N_{10}$ is used in $Update(\mathcal{B},\mathcal{P})$'s evaluation, while other orderings are useless (see Definition 3). Hence, from Definition 6, the only resulting knowledge base $\mathcal{B}'$ after updating $\mathcal{B}$ by $\mathcal{P}$ is: $\{A, B, C\}$ ■

Example 5 Let us consider the secure computer system domain described in section 1 again. Let

$\mathcal{B} = \{Member(A,G), Member(B,G),$
$Access(A,F), \neg Access(B,F)\}$ and
$\mathcal{P} = \{Member(C,G) \leftarrow, Member(D,G) \leftarrow,$

$$Access(x,F) \leftarrow Member(x,G), \\ not\ \neg Access(x,F)\}.$$

Consider the update of $\mathcal{B}$ by $\mathcal{P}$. Ignoring the detail, using the approach presented above, we get a unique resulting knowledge base

$$\mathcal{B}' = \{Member(A,G),\ Member(B,G), \\ Member(C,G),\ Member(D,G), \\ Access(A,F),\ \neg Access(B,F), \\ Access(C,F), Access(D,F)\}.$$

■

5 Update and Minimal Change

In this section we investigate the minimal change property for the generalized rule-based update described previously. Let $\mathcal{B}$ be a consistent knowledge base and r a rule with the form (3). $\mathcal{B}$ *satisfies* r iff if facts $L_1, \cdots, L_m$ are in $\mathcal{B}$ and fact $L_{m+1}, \cdots, L_n$ are not in $\mathcal{B}$, then fact L_0 is in $\mathcal{B}$. Let $\mathcal{P}$ be a set of rules with the form (3). $\mathcal{B}$ satisfies $\mathcal{P}$ if $\mathcal{B}$ satisfies each rule in $\mathcal{P}$.

Let $\mathcal{B}$ and $\mathcal{B}'$ be two knowledge bases. We use $Diff(\mathcal{B},\mathcal{B}')$ to denote the set of different ground atoms between $\mathcal{B}$ and $\mathcal{B}'$, i.e.

$$Diff(\mathcal{B},\mathcal{B}') = \{|L| \mid L \in (\mathcal{B}-\mathcal{B}') \cup (\mathcal{B}'-\mathcal{B})\},$$

where notation $|L|$ indicates the corresponding ground atom of ground literal L, and $Min(\mathcal{B},\mathcal{P})$ to denote the set of all consistent knowledge bases satisfying $\mathcal{P}$ but with minimal differences from $\mathcal{B}$, i.e.

$$Min(\mathcal{B},\mathcal{P}) = \{\mathcal{B}' \mid \mathcal{B}' \text{ satisfies } \mathcal{P} \text{ and } \\ Diff(\mathcal{B},\mathcal{B}') \text{ is minimal with} \\ \text{respect to set inclusion}\}.$$

Then we have the following result.

Theorem 3 *Let $\mathcal{B}$ be a knowledge base, $\mathcal{P}$ a set of update rules, and $Update(\mathcal{B},\mathcal{P})$ the specification of updating $\mathcal{B}$ by $\mathcal{P}$ as defined in Definition 5. If $\mathcal{B}'$ is a consistent resulting knowledge base with respect to the update specification $Update(\mathcal{B},\mathcal{P})$, then $\mathcal{B}' \in Min(\mathcal{B},\mathcal{P})$.*

The above theorem guarantees that our generalized rule-based update satisfies the principle of minimal change. However, it should be noted that not every element of $Min(\mathcal{B},\mathcal{P})$ could be a resulting knowledge base of updating $\mathcal{B}$ by $\mathcal{P}$. The following example illustrates the case.

Example 6 Let $\mathcal{B} = \{A, B\}$ and $\mathcal{P} = \{\neg A \leftarrow B, \neg B \leftarrow not\ B\}$. Using our approach, updating $\mathcal{B}$ by $\mathcal{P}$ will give us a unique resulting knowledge base $\mathcal{B}' = \{\neg A, B\}$. However, it is not difficult to see that $Min(\mathcal{B},\mathcal{P})$ also contains another element: $\mathcal{B}'' = \{A, \neg B\}$. Although $\mathcal{B}''$ satisfies $\mathcal{P}$ and has minimal difference from $\mathcal{B}$, it does not represent an intuitive solution of updating $\mathcal{B}$ by $\mathcal{P}$ according to our previous discussion. ■

6 Concluding Remarks

In this paper we considered generalized rule-based updates in the framework of prioritized logic programs. We should mention that Marek and Truszczński's rule-based update is embeddedable into our framework. For example, if our update rules only contain classical negations, our formulation is reduced to Przymusinski and Turner's described in [Przymusinski and Turner, 1995].

Finally, we have noticed that the issue of logic programs with preferences has also been explored recently by some other researchers (eg. [Brewka, 1996]). In fact, we can further extend our prioritized logic programs by associating with *dynamic preference* so that our prioritized logic programs can be used as a more general tool in broad areas of knowledge representation and reasoning. A detailed comparison between our PLPs and others' work and other applications of PLPs in reasoning about change is beyond the scope of this paper and was represented in our full version manuscript [Zhang and Foo, 1997].

Acknowledgements

This research is supported in part by a grant from the Australian Research Council. We thank to Chitta Baral for many valuable comments on an earlier draft of this paper.

References

[Baral, 1994] C. Baral. Rule based updates on simple knowledge bases. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'94)*, pages 136–141. AAAI Press, 1994.

[Brewka, 1996] G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 14:19–36, 1996.

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

[Marek and Truszczński, 1994] M. Marek and M. Truszczński. Update by means of inference rules. In *Proceedings of JELIA'94, Lecture Notes in Artificial Intelligence*, 1994.

[Przymusinski and Turner, 1995] T.C. Przymusinski and H. Turner. Update by means of inference rules. In *Proceedings of LPNMR'95, Lecture Notes in Artificial Intelligence*, pages 156–174, 1995.

[Zhang and Foo, 1997] Y. Zhang and N.Y. Foo. Prioritized logic programming and reasoning about change. Technical report, University of Western Sydney, Nepean, 1997.