Logic Program Based Updates

YAN ZHANG
University of Western Sydney

In logic program-based updates, contradictory information elimination, conflict resolution, and syntactic representation are three major issues that interfere with each other and significantly influence the update result. We observe that existing approaches of logic program-based updates, in one way or another, are problematic to deal with these issues. In this paper, we address all these problems in a systematic manner. Our approach to the logic program-based update has the following features: (1) a prioritized logic programming language is employed for providing a formal basis of formalizing logic program-based updates, so that information conflict and its related problems in updates can be handled properly; (2) our approach presents both semantic characterization and syntactic representation for the underlying update procedure, and hence is consistent with the nature of updates within the logic program extent - declarative semantics and syntactic sensitivity; and (3) our approach also provides nontrivial solutions to simplify various update evaluation procedures under certain conditions.

Categories and Subject Descriptors: I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving —*logic programming*; *nonmonotonic reasoning and belief revision*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*representations (procedural and rule-based)*

General Terms: Languages, Theory

Additional Key Words and Phrases: Computational complexity, conflict resolution, logic program-based update, prioritized logic programming

1. INTRODUCTION

Logic programming has been proved to be one of the most promising logic based formulations for problem solving, knowledge representation and reasoning, and reasoning about changes. As one growing method, the logic program-based update provides a feasible framework for modeling agent's activities in dynamic environments [Eiter et al. 2002]. Comparing with other logic based revision/update formulations, e.g. [Boutilier 1996; Herzig and Rifi 1999], it inherits great advantages from logic programming for both declarative semantics (e.g. stable mode/answer set semantics [Gelfond and Lifschitz 1991]) and efficient proving procedures (e.g. `smodels`, `DLV` and `XSB` [Nemela and Simons 1996; Eiter et al. 1997; Rao et al. 1997]), which make this method be more applicable in the real world problem domains, e.g. [Crescini and Zhang 2004].

Author's address: Y. Zhang, School of Computing and Mathematics, University of Western Sydney, Penrith South DC, NSW 1797, Australia; email: yan@scm.uws.edu.au.

1.1 Two Types of Logic Program Based Updates

Logic program-based updates can be viewed in two kinds: one is *simple fact updates* and the other is *program updates*. In simple fact updates, a knowledge base is represented as a finite set of ground atoms/literals. Such knowledge base may be updated in order to integrate with new information/knowledge, which is represented as a finite set of rules (called *update rules*).

One of the earliest formal work on simple fact updates (also called *rule based updates*) was due to Marek and Truszczyński [Marek and Truszczyński 1994; 1998]. Basically, Marek and Truszczyński addressed the following problem: given an initial *knowledge base* $\mathcal{B}$ which is a finite a set of ground atoms, and a set Π of update rules of 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 $\mathcal{B}$ with Π? The intuitive meaning of (1) is that if $B_1, \cdots, B_m$ are *in* the *current* knowledge base, and $C_1, \cdots, C_n$ are *not*, then A should be added into the knowledge base (if it is not there already). A similar interpretation can be given for (2) as well. For example, given a knowledge base $\mathcal{B} = \{A, D\}$ and a set of update rules

Π:
$in(C) \leftarrow in(A), out(B)$,
$out(D) \leftarrow in(C), out(B)$,

where A, B, C and D are ground atoms, then after updating $\mathcal{B}$ with Π, Marek and Truszczyński's approach will give a resulting knowledge base $\mathcal{B}' = \{A, C\}$. Obviously, this result is correct from the general principle of minimal change in knowledge base updates, i.e. [Katsuno and Mendelzon 1991; Winslett 1988]

Marek and Truszczyński's simple fact update procedure can be extended by allowing a knowledge base to be incomplete or/and update rules to be generalized as logic inference rules having both classical and weak negations (negation as failure), i.e. [Baral 1994; Przymusinski and Turner 1997]. However, it is observed that the requirement on minimal change turns out to be a difficult goal when the simple fact update has such generalized forms (will be addressed in section 3).

In program updates, on the other hand, a knowledge base is usually represented as a logic program, and this knowledge base may be updated in terms of a set of update rules, which is also a logic program. Usually, we use *extended logic programs* [Gelfond and Lifschitz 1991] to represent both the knowledge base and the set of update rules. Generally speaking, an extended logic program is a finite set of rules:

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

where $L_0, L_1, \cdots, L_n$ are literals. In the body of rule (3), both classical negation $\neg$ and weak negation *not* are allowed to be presented. The meaning of (3) is as follows: if $L_1, \cdots, L_m$ hold, and there is no explicit evidence to show that $L_{m+1}, \cdots, L_n$ hold,

[1]Π was also called a *revision program* in [Marek and Truszczyński 1994; 1998].

then L_0 holds. Typically, the problem of a program update is as follows: given two extended logic programs Π_0 and Π_1, where Π_0 represents the agent's initial knowledge base and Π_1 represents the agent's new knowledge, how to update Π_0 in terms of Π_1? Sometimes, the simple fact update may be viewed as a special case of the program update because each fact (i.e. ground atom/literal) can be viewed as a rule without body, i.e. $L_0 \leftarrow$. But, as we will show in this paper, the simple fact update plays a critical role in the formalization of program updates.

To illustrate key issues associated with program updates, we consider the following example. Suppose that an access control policy base for a computer system is represented by the following extended logic program Π_0:

$$\begin{array}{l} Member(A,G) \leftarrow, \\ Member(B,G) \leftarrow, \\ Access(A,F_2) \leftarrow, \\ Access(x,F_1) \leftarrow Member(x,G), \\ \neg Access(x,F_2) \leftarrow Member(x,G), not\ Access(x,F_2). \end{array}$$

Under Gelfond and Lifschitz's answer set semantics, it is clear that implicit facts $Access(A,F_1)$, $Access(B,F_1)$ and $\neg Access(B,F_2)$ are entailed from Π_0. Now suppose we want to update this policy base Π_0 with the following extended logic program Π_1:

$$\begin{array}{l} Member(C,G) \leftarrow, \\ \neg Access(x,F_1) \leftarrow Member(x,G), \\ Access(x,F_2) \leftarrow Member(x,G), not\ \neg Access(x,F_2). \end{array}$$

Here the question is: what is the result after updating Π_0 with Π_1?

Intuitively, since program Π_1 represents the agent's new knowledge about the domain, information represented by Π_1 should be retained in the final result (in some form) after the update. On the other hand, it appears that rule

$$r_1 : Access(x,F_1) \leftarrow Member(x,G)$$

in Π_0 represents a contradictory meaning compared to the rule

$$r_2 : \neg Access(x,F_1) \leftarrow Member(x,G)$$

in Π_1, and if both rules are retained, an inconsistency may be derived (together with other information) under the answer set semantics. In this sense, rule r_1 contradicts with rule r_2 and should *not* be effective in the final result of the update.

Also, although rule

$$r_3 : \neg Access(x,F_2) \leftarrow Member(x,G), not\ Access(x,F_2)$$

in Π_0 does not contradict any rules in Π_1, it indeed *conflicts* with rule

$$r_4 : Access(x,F_2) \leftarrow Member(x,G), not\ \neg Access(x,F_2)$$

in Π_1 because retaining r_3 and r_4 together will generate indefinite results on facts $Access(x,F_2)$ (and $\neg Access(x,F_2)$, resp., where $x = A,B,C$). In this sense, we would like to *override* r_3 by using r_4.

From the above discussion, we may conclude, in an intuitive way, that after the update, the following set of facts holds:

$$S = \{\neg Access(A, F_1), \neg Access(B, F_1), \neg Access(C, F_1),\ Access(A, F_2),\\ Access(B, F_2), Access(C, F_2)\}.$$

However, this semantic characterization for an update result does not capture all relevant information conducted in an update process. For instance, fact $Access(B, F_2)$ in S is derived from a generic rule r_4 in Π_1, which is not reflected in S. This is so-called *information loss* - one of the most distinguishing features between traditional knowledge base updates and logic program-based updates. Hence, a proper syntactic representation for logic program updates is important. In this particular example, we would prefer to generate a resulting program Π' after updating Π_0 with Π_1 as follows:

Π':
$Member(A, G) \leftarrow,$
$Member(B, G) \leftarrow,$
$Member(C, G) \leftarrow,$
$Access(A, F_2) \leftarrow,$
$\neg Access(x, F_1) \leftarrow Member(x, G),$
$Access(x, F_2) \leftarrow Member(x, G), not\ \neg Access(x, F_2),$

which has a unique answer set S.

1.2 Contributions of This Paper

From the discussions in section 1.1, we can see that traditional minimal change criterion is no longer a single issue in logic program-based updates. Instead, three major issues must be considered in logic program-based updates:

(1) *Contradictory information elimination.* Clearly, this is also a basic requirement for classical knowledge base updates. However, in logic program-based updates, this requirement should be achieved with proper justifications *and* without violating the underlying minimal change semantics.

(2) *Conflict resolution.* As in the general form of logic program-based updates, both classical and weak negations are allowable in update rules (e.g. the simple fact update) or in both knowledge base and update rules (e.g. the program update), information conflict will significantly influence the update result. Conflict resolution is essential not only for achieving a desirable update result but also to guarantee a minimal elimination of contradictory information.

(3) *Syntactic representation.* It has been observed that semantic characterizations are not enough to capture all information embedded in an update procedure. Instead, a proper syntactic form is equally important to represent an update result. Intuitively, it is expected that the syntactic representation of an update result should contain as many as possible consistent rules from the initial knowledge base and update program under the conditions of contradiction elimination and conflict resolution.

Although logic program-based updates have been studied by many researchers recently, e.g. [Alferes et al. 1998; Alferes et al. 2002; Alves et al. 1995; Guessoum and Lloyd 1991], most current approaches only focused on the first issue - contradictory information elimination and with different restrictions on other issues.

These approaches were developed either on a semantic basis or a syntactic basis, but without considering the influence on an update procedure from both aspects[2].

This paper addresses all these three major issues of logic program-based updates in a systematic manner. We observed that contradictory information elimination, conflict resolution and syntactic representation are closely related, and they should be studied within a unified formalism. For this purpose, we employ a prioritized logic programming language to provide a formal basis of formulating both the simple fact and program updates. Our approach integrates both semantic characterizations and syntactic representations and hence meets the essential requirements of logic program-based updates where none of existing approaches does at the same time. This paper also investigates important theoretical properties of logic program-based updates, provides detailed complexity analysis and useful computational strategies for simplifying the underlying update procedure.

Some results presented in this paper have been previously published in IJCAI-97, ECAI-98, ICLP-99 and AI-01 [Zhang and Foo 2001; 1998; Zhang 1999; 2001].

The paper is organized as follows. Section 2 gives an overview of the author's prioritized logic programming language which provides a formal basis for our formulation of logic program-based updates. Section 3 develops an approach for the simple fact update which overcomes major difficulties of Marek and Truszczyński's approach. Section 4 focuses on the study of the restricted monotonicity - one of the most important issues in nonmonotonic system changes. We show how restricted monotonicity properties can be used to simplify the evaluation procedure in a simple fact update. Section 5 addresses the problem of program updates and propose an approach which integrates both semantic and syntactic considerations in program updates. Section 6 studies important properties of program updates. Specifically, this section explores how the inference problem related to a program update may be simplified. Section 7 analyzes the computational complexity of both simple fact and program updates. Section 8 presents detailed comparisons between our approach and other related work. In particular, we provide formal characterizations on the relations between our approach and three typical existing logic program update approaches respectively. Finally, section 9 concludes this paper with some further discussions.

2. PRIORITIZED LOGIC PROGRAMS: AN OVERVIEW

In this section, we present a brief overview on prioritized logic programs proposed initially by Zhang and Foo [Zhang and Foo 1997] and then refined by Zhang [Zhang 2003b]. Basically, a prioritized logic program is an extended logic program associating to a preference ordering on rules in the program. The semantics of prioritized logic programs is defined based on Gelfond and Lifschitz's answer set semantics for extended logic programs [Gelfond and Lifschitz 1991].

A language $\mathcal{L}$ of extended logic programs is determined by its object constants, function constants and predicate 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 constant 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 (3):

[2]Detailed comparisons between our approach and others are referred to section 8.

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

where each L_i $(0 \leq i \leq n)$ is a literal. L_0 is called the *head* of the rule, while $\{L_1, \cdots, L_m,\ notL_{m+1}, \cdots,\ notL_n\}$ is called the *body* of the rule. Obviously, the body of a rule could be empty. We also allow the head of a rule to be empty. In this case, the rule with an empty head is called *constraint.* A term, atom, literal, or rule is *ground* if no variable occurs in it. An *extended logic program* Π is a collection of rules. Π is *ground* if each rule in Π is ground.

Let r be a ground rule of the form (3), we use $pos(r)$ to denote the set of literals in the body of r without weak negation $\{L_1, \cdots, L_m\}$, and $neg(r)$ the set of literals in the body of r with weak negation in front $\{L_{m+1}, \cdots, L_n\}$. We specify $body(r)$ to be $pos(r) \cup neg(r)$. We also use $head(r)$ to denote the head of r: $\{L_0\}$. Then we use $lit(r)$ to denote $head(r) \cup body(r)$. By extending these notations, we use $pos(\Pi)$, $neg(\Pi)$, $body(\Pi)$, $head(\Pi)$, and $lit(\Pi)$ to denote the unions of corresponding components of all rules in the ground program Π, e.g. $body(\Pi) = \bigcup_{r \in \Pi} body(r)$. If Π is a non-ground program, then notions $pos(\Pi)$, $neg(\Pi)$, $body(\Pi)$, $head(\Pi)$, and $lit(\Pi)$ are defined based on the ground instantiation of Π (see below definition).

Let Π be a ground extended logic program not containing *not* and *Lit* the set of all ground literals in the language of Π. An *answer set* of Π 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 $L_0 \in S$; and (ii) if S contains a pair of complementary literals, then $S = Lit$. Now let Π be a ground arbitrary 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[3]. We define that S is an *answer set* of Π iff S is an answer set of Π^S.

For a non-ground extended logic program Π, we usually view a rule in Π containing variables to be the set of all ground instances of this rule formed from the set of ground literals in the language. The collection of all these ground rules forms the *ground instantiation* Π' of Π. Then a set of ground literals is an answer set of Π if and only if it is an answer set of Π'. It is easy to see that an extended logic program may have one, more than one, or no answer set at all.

A *prioritized logic program* (PLP) $\mathcal{P}$ is a triple $(\Pi, \mathcal{N}, <)$, where Π is an extended logic program, $\mathcal{N}$ is a naming function mapping each rule in Π to a name, and $<$ is a strict partial ordering on names. The partial ordering $<$ in $\mathcal{P}$ plays an essential role in the evaluation of $\mathcal{P}$. We also use $\mathcal{P}(<)$ to denote the set of $<$-relations of $\mathcal{P}$. Intuitively $<$ represents a preference of applying rules during the evaluation of the program. In particular, if $\mathcal{N}(r) < \mathcal{N}(r')$ holds in $\mathcal{P}$, 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').

Definition 2.1. [Zhang 2003b] Let Π be a ground extended logic program and r a ground rule of the form (3) (r does not necessarily belong to Π). Rule r is *defeated* by Π iff Π has an answer set and for any answer set S of Π, there exists some $L_i \in S$, where $m + 1 \leq i \leq n$.

[3]We also call Π^S the Gelfond-Lifschitz transformation of Π in terms of S.

Similarly to the case of extended logic programs, the evaluation of a PLP will be based on its ground form. We say that a PLP $\mathcal{P}' = (\Pi', \mathcal{N}', <')$ is the *ground instantiation* of $\mathcal{P} = (\Pi, \mathcal{N}, <)$ if (1) Π' is the ground instantiation of Π; and (2) $<'$ is a strict partial ordering and $\mathcal{N}'(r'_1) <' \mathcal{N}'(r'_2) \in \mathcal{P}'(<')$ if and only if there exist rules r_1 and r_2 in Π such that r'_1 and r'_2 are ground instances of r_1 and r_2 respectively and $\mathcal{N}(r_1) < \mathcal{N}(r_2) \in \mathcal{P}(<)$ (see [Zhang 2003b] for more details about ground instantiation requirement).

In the rest of the paper, whenever there is no confusion, we will only consider ground prioritized (extended) logic programs without explicit declaration.

Definition 2.2. [Zhang 2003b] Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$ be a prioritized 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:

(1) $\Pi_0 = \Pi$;

(2) $\Pi_i = \Pi_{i-1} - \{r_1, r_2, \cdots \mid$ (a) there exists $r \in \Pi_{i-1}$ such that
for every j $(j = 1, 2, \cdots)$, $\mathcal{N}(r) < \mathcal{N}(r_j) \in \mathcal{P}(<)$ and
$r_1, r_2, \cdots$ are defeated by $\Pi_{i-1} - \{r_1, r_2, \cdots\}$, and (b) there
are no rules $r', r'', \cdots \in \Pi_{i-1}$ such that $N(r_j) < N(r')$,
$N(r_j) < N(r''), \cdots$ for some j $(j = 1, 2, \cdots)$ and $r', r'', \cdots$
are defeated by $\Pi_{i-1} - \{r', r'', \cdots\}\}$;

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

Definition 2.3. [Zhang 2003b] 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 $\mathcal{P}$ iff S is an answer set for some reduct $\mathcal{P}^{<}$ of $\mathcal{P}$.

As the same for extended logic programs, a PLP may have one, more than one, and no answer set at all. We call a PLP *well defined* if it has a consistent answer set. Now we consider a program $\mathcal{P}$:

$$\begin{aligned}
&N_1 : A \leftarrow,\\
&N_2 : B \leftarrow not\ C,\\
&N_3 : D \leftarrow,\\
&N_4 : C \leftarrow not\ B,\\
&N_1 < N_2, N_3 < N_4.
\end{aligned}$$

According to Definition 2.2, it is easy to see that $\mathcal{P}_2$ has two reducts:

$$\begin{aligned}
&\{A \leftarrow, \quad D \leftarrow, \quad C \leftarrow not\ B\}, \text{ and}\\
&\{A \leftarrow, \quad B \leftarrow not\ C, \quad D \leftarrow\}.
\end{aligned}$$

From Definition 2.3, it follows that $\mathcal{P}_2$ has two answer sets: $\{A, C, D\}$ and $\{A, B, D\}$.

Now if we remove rule N_3 from the above program $\mathcal{P}$, and add preference relation $N_2 < N_4$ into $\mathcal{P}$, then the resulting program will have one answer set $\{A, B\}$, where rule N_4 is defeated.

The implementation detail of our prioritized logic programming language is referred to [Zhang et al. 2001].

3. SIMPLE FACT UPDATE

In this section, we first discuss limitations of existing simple fact update approaches, specifically, the limitation of Marek and Truszczyński's approach. Then we propose a framework of simple fact update which generalizes previous approaches and overcomes the major limitation in these approaches.

3.1 Limitation of Marek and Truszczyński's Approach

To simplify our presentation, we adopt the extended logic program based formulation as shown in [Baral 1994; Przymusinski and Turner 1997] to discuss Marek and Truszczyński's simple fact update. In particular, by illustrating a simple example here, we will see that Marek and Truszczyński's approach is not suitable for generalized simple fact updates as we mentioned in section 1.1.

EXAMPLE 3.1. Suppose $\mathcal{B} = \{\neg A, B, C\}$ is a knowledge base, and Π consists of the following update rules:

$\neg B \leftarrow not\ B$,
$A \leftarrow C$.

Consider an update of $\mathcal{B}$ with Π. Obviously, fact $\neg A$ should change to A by applying the second rule of Π. 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 Π 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 formalism defined by Baral and Przymusinski and Turner [Baral 1994; Przymusinski and Turner 1997], which is logically equivalent to Marek and Truszczyński's, to specify the above update procedure within an extended logic program[4]. Firstly, we need to extend the language of our domain by adding new propositional letters of the form $New\text{-}L$ if L is a propositional letter in the original language. 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 Π^* 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:

[4]The formalism used here, nevertheless, is slightly different from theirs for the purpose of simplicity.

$\neg New\text{-}B \leftarrow not\ New\text{-}B$,
$New\text{-}A \leftarrow New\text{-}C$.

Generally speaking, an answer set of program Π^* represents a possible resulting knowledge base after updating $\mathcal{B}$ with Π, 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 Π^* has two answer sets: in one $New\text{-}B$ is true where in the other $New\text{-}B$ is false. This means that the truth value of B is indefinite with respect to this update. This solution, however, seems not quite consistent with our previous observation.

Observing program Π^*, 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 Π^* to have two different answer sets with an indefinite truth value of $New\text{-}B$. This result also violates the requirement of minimal change.

Taking a closer look, we know that the inertia rule $New\text{-}B \leftarrow B,\ not\ \neg New\text{-}B$ should be preferred over the update rule $\neg New\text{-}B \leftarrow not\ New\text{-}B$ in order to meet the minimal change principle. In this case, the inertia rule will defeat the corresponding update rule during the evaluation of Π^*. But such preference information cannot be expressed in Gelfond and Lifschitz's extended logic programs.

While prioritized ligic programming has been recently studied by many researchers, e.g. [Delgrande et al. 2000; Schaub and Wang 2001; Zhang 2003b], it seems a natural way to employ a kind of prioritized logic programming language to formalize the generalized simple fact update where the information conflict can be explicitly handled. This motivates our following formalization on generalized simple fact update within the framework of prioritized logic programs proposed by the author [Zhang and Foo 1997; Zhang 2003b].

3.2 Generalized Simple Fact Update

Consider a language $\mathcal{L}$ of prioritized logic programs. We specify that a *knowledge base* $\mathcal{B}$ is a *consistent* set of ground literals of $\mathcal{L}$ and *update program* Π is an extended logic program where each rule of the form (3) in Π is called an *update rule*. 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 $\mathcal{B}$ with Π. For this purpose, we first extend language $\mathcal{L}$ by the following way. We specify $\mathcal{L}_{new}$ to be a language of PLPs based on $\mathcal{L}$ with one more augment: for each predicate symbol P in $\mathcal{L}$, there is a corresponding predicate symbol $New\text{-}P$ in $\mathcal{L}_{new}$ with the same arity of P.

To simplify our presentation, in $\mathcal{L}_{new}$ we use notation $New\text{-}L$ to denote the corresponding literal L in $\mathcal{L}$. For instance, if a literal L in $\mathcal{L}$ is $\neg P(x)$, then notation $New\text{-}L$ simply means $\neg New\text{-}P(x)$. We use Lit_{new} to denote the set of all ground literals of $\mathcal{L}_{new}$, i.e. $Lit_{new} = Lit \cup \{New\text{-}L \mid L \in Lit\}$.

Definition 3.1. Let $\mathcal{B}$, Π, $\mathcal{L}$, and $\mathcal{L}_{new}$ be specified as above. The *specification* *of updating* $\mathcal{B}$ *with* Π is a PLP of $\mathcal{L}_{new}$, denoted as $Update(\mathcal{B}, \Pi) = (\Pi^*, \mathcal{N}, <)$, as follows:

(1) Π^* 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:
$New\text{-}P(x) \leftarrow P(x), not \neg New\text{-}P(x)$[5], and
$\neg New\text{-}P(x) \leftarrow \neg P(x), not New\text{-}P(x)$,
Update rules: for each rule
$L_0 \leftarrow L_1, \cdots, L_m, not L_{m+1}, \cdots, not L_n$ in Π, there is a rule
$New\text{-}L_0 \leftarrow New\text{-}L_1, \cdots, New\text{-}L_m,\ not New\text{-}L_{m+1}, \cdots, not New\text{-}L_n;$

(2) Naming function $\mathcal{N}$ assigns a unique name N for each rule in Π^*;

(3) For any inertia rule r and update rule r', $\mathcal{N}(r) < \mathcal{N}(r')$.

Comparing Definition 3.1 with the update specification described in Example 3.1, 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 have higher preferences than update rules in $Update(\mathcal{B}, \Pi)$. The intuitive idea behind this is that a preference ordering between an inertia rule and an update rule in $Update(\mathcal{B}, \Pi)$ will affect the evaluation of $Update(\mathcal{B}, \Pi)$ *only if* these two rules conflict with each other, e.g. applying one rule causes the other inapplicable. On the other hand, from the minimal change principle, a fact in the initial knowledge base $\mathcal{B}$ is always preferred to persist during an update whenever there is no violation of update rules[6]. Therefore, when conflicts occur between inertia and update rules, inertia rules should defeat the corresponding update rules. Otherwise, the preference ordering does not play any role in the evaluation of $Update(\mathcal{B}, \Pi)$. Note that in the case that $\mathcal{B}$ is finite and Π is a propositional finite program, there will be at most $2k \cdot l$ $<$-relations in $Update(\mathcal{B}, \Pi)$, where k is the number of predicate symbols of $\mathcal{L}$ and l is the number of update rules in Π.

Definition 3.2. Let $Update(\mathcal{B}, \Pi)$ be specified as in Definition 3.1. A set $\mathcal{B}'$ of ground literals is called a *possible resulting knowledge base* with respect to $Update(\mathcal{B}, \Pi)$, iff $\mathcal{B}'$ satisfies the following conditions:

(1) if $Update(\mathcal{B}, \Pi)$ has a consistent answer set, say S, then
$\mathcal{B}' = \{L \mid New\text{-}L \in S\}$;

(2) if $Update(\mathcal{B}, \Pi)$ does not have a consistent answer set (i.e. $Update(\mathcal{B}, \Pi)$ is not well defined), then $\mathcal{B}' = \mathcal{B}$.

We use $Res(Update(\mathcal{B}, \Pi))$ to denote the set of all resulting knowledge bases of $Update((\mathcal{B}, \Pi)$.

Example 3.2. Example 3.1 continued. Let $\mathcal{B} = \{\neg A, B, C\}$ and Π consist of the following rules:

[5] x might be a tuple of variables.
[6] Note that an update rule in $Update(\mathcal{B}, \Pi)$ is defeasible if it contains a weak negation *not* in its body.

$\neg B \leftarrow not\ B$,
$A \leftarrow C$.

From Definition 3.2, the specification of updating $\mathcal{B}$ with Π, $Update(\mathcal{B}, \Pi)$, 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 it is easy to see that $Update(\mathcal{B}, \Pi)$ has a unique answer set:

$$\{\neg A, B, C, New\text{-}A, New\text{-}B, New\text{-}C\}.$$

Note that in $Update(\mathcal{B}, \Pi)$, only $N_5 < N_{10}$ is used in the evaluation of $Update(\mathcal{B}, \Pi)$, while other $<$-relations are useless. Hence, from Definition 3.2, the only resulting knowledge base $\mathcal{B}'$ after updating $\mathcal{B}$ with Π is: $\{A, B, C\}$

Example 3.3. Let us consider the secure computer system domain described in section 1 once again. Let

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

and Π consists of the following rules:

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

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

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

3.3 Basic Properties

In this subsection, we investigate basic semantic properties of the generalized simple fact update. Firstly we show that the update specification $Update(\mathcal{B}, \Pi)$ in language $\mathcal{L}_{new}$ defined in Definition 3.1 can be simplified to a PLP in language $\mathcal{L}$.

LEMMA 3.3. *Let $Update(\mathcal{B}, \Pi)$ be a well defined update specification as defined in Definition 3.1. $\mathcal{B}'$ is a resulting knowledge base with respec to $Update(\mathcal{B}, \Pi)$ if and only if $\mathcal{B}'$ is an answer set of prioritized logic program $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <)$, where for each rule $r : L \leftarrow not\overline{L}$ with $L \in \mathcal{B}$, and each rule r' in Π, $\mathcal{N}(r) < \mathcal{N}(r')$*[7].

PROOF. According to Definition 3.1, $\mathcal{B}'$ is a resulting knowledge base with respect to update specification $Update(\mathcal{B}, \Pi)$ if and only if $\mathcal{B}' = \{L \mid New\text{-}L \in S\}$, where S is an answer set of $Update(\mathcal{B}, \Pi)$. Now we show that $Update(\mathcal{B}, \Pi)$ can be reduced to another PLP, say $\mathcal{P}'$, such that each rule in $\mathcal{P}'$ only contains literals $New\text{-}L$ in $\mathcal{L}_{new}$ and without including literals in $\mathcal{L}$. Note that $Update(\mathcal{B}, \Pi)$ is a PLP in language $\mathcal{L}_{new}$. Examining the construction of $Update(\mathcal{B}, \Pi)$, it is easy to see that $\Pi(\mathcal{B}, \mathcal{P})$ consists of three types of rules:

(i) initial knowledge rules: $L \leftarrow$ iff $L \in \mathcal{B}$;

(ii) inertia rules: for each literal L in $\mathcal{L}$, $New\text{-}L \leftarrow L,\ not\ \overline{New\text{-}L}$;

(iii) update rules: for each rule of the form (3) in Π,

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

Since for each rule $L \leftarrow$ in $Update(\mathcal{B}, \Pi)$, L will always occur in each answer set of $Update(\mathcal{B}, \Pi)$, this means that for any literal L in $\mathcal{B}$, the corresponding inertia rule $New\text{-}L \leftarrow L,\ not\ \overline{New\text{-}L}$ can be simplified as $New\text{-}L \leftarrow not\ \overline{New\text{-}L}$. On the other hand, for any literal $L' \notin \mathcal{B}$, the corresponding inertia rule r:

$$New\text{-}L' \leftarrow L',\ not\ \overline{New\text{-}L'}$$

will never be used in the evaluation of $Update(\mathcal{B}, \Pi)$, so it can be omitted. Therefore, $Update(\mathcal{B}, \Pi)$ can be reduced to a PLP $\mathcal{P}' = (\Pi', \mathcal{N}', <')$, where Π' consists of two types of rules:

(i) inertia rules: for each $L \in \mathcal{B}$, $New\text{-}L \leftarrow not\ \overline{New\text{-}L}$;

(ii) update rules: the same as in $Update(\mathcal{B}, \Pi)$;

and for each inertia rule r and update rule r', $\mathcal{N}(r) < \mathcal{N}(r')$. Obviously, this reduction does not have any effect on the evaluation of $Update(\mathcal{B}, \Pi)$.

Now we specify $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}, \mathcal{N}, <)$, where for each rule $r : L \leftarrow not\overline{L}$ with $L \in \mathcal{B}$ and each rule $r' \in \Pi$, $\mathcal{N} < \mathcal{N}(r')$. It is clear that there exists an one-to-one mapping between $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <)$, and $\mathcal{P}' = (\Pi', \mathcal{N}', <')$: each rule in $\mathcal{P}'$ can be obtained from the corresponding rule in $\mathcal{P}$ by replacing each literal $New\text{-}L$ occurring in the rule with literal L, and *vice versa*. This follows that S is an answer set of $\mathcal{P}'$ if and only if $\mathcal{B}'$ is an answer set of $\mathcal{P}$. $\square$

[7] $\overline{L}$ stands for the complement of literal L.

Recall that $\mathcal{B}$ *satisfies* r iff for each ground instance r' of r:

$$L'_0 \leftarrow L'_1, \cdots, L'_m, notL'_{m+1}, \cdots, notL'_n,$$

if facts $L'_1, \cdots, L'_m$ are in $\mathcal{B}$ and facts $L'_{m+1}, \cdots$, and L'_n are not in $\mathcal{B}$, then fact L'_0 is in $\mathcal{B}$. Let Π be a set of rules of the form (3). $\mathcal{B}$ satisfies Π if $\mathcal{B}$ satisfies each rule in Π. We will show next that after updating knowledge base $\mathcal{B}$ with Π, every possible resulting knowledge base $\mathcal{B}'$ satisfies Π as stated in the following proposition.

Proposition 3.4. *Given a knowledge base* $\mathcal{B}$ *and an update program* Π*. Suppose the update specification* $Update(\mathcal{B}, \Pi)$ *is well defined. Let* $\mathcal{B}'$ *be a resulting knowledge base with respect to* $Update(\mathcal{B}, \Pi)$*. Then* $\mathcal{B}'$ *satisfies* Π.

Proof. From Lemma 3.3, $\mathcal{B}'$ is also an answer set of $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <)$, where for each rule $r : L \leftarrow not\overline{L}$ and each rule r' in Π, $\mathcal{N}(r) < \mathcal{N}(r')$. Let Π' be an extended logic program that is a reduct of $\mathcal{P}$ and $\mathcal{B}'$ is an answer set of Π'. If a ground instance r of a rule in Π is in Π', then from the answer set definition for extended logic program, r is satisfied in $\mathcal{B}'$.

Now suppose that a ground instance r of the rule in Π is *not* in Π', i.e. r is eliminated during the reduction of $\mathcal{P}$. From Definition 2.2, there must exist a rule r' of the form $L \leftarrow not\overline{L}$ such that $\mathcal{N}(r') < \mathcal{N}(r)$, and literal L occurs in the body of r with a week negation *not* in front. From the specification of preference relations in $\mathcal{P}$, there is no another rule r'' in $\mathcal{P}$ such that $\mathcal{N}(r'') < \mathcal{N}(r')$. From here it follows $L \in \mathcal{B}'$. In this case, r is still satisfied in $\mathcal{B}'$. □

Now we consider the minimal change in our simple fact update. Let $\mathcal{B}$ and $\mathcal{B}'$ be two knowledge bases. We use $Diff(\mathcal{B}, \mathcal{B}')$ to denote the symmetric set difference on 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 $|L|$ indicates the corresponding ground atom of literal L, and $Min(\mathcal{B}, \Pi)$ to denote the set of all consistent knowledge bases satisfying Π but with minimal differences from $\mathcal{B}$, i.e.

$$Min(\mathcal{B}, \Pi) = \{\mathcal{B}' \mid \mathcal{B}' \text{ satisfies } \Pi \text{ and } Diff(\mathcal{B}, \mathcal{B}') \text{ is minimal in terms of set inclusion}\}.$$

Then we have the following minimal change theorem.

Theorem 3.5. *(***Minimal Change***) Let* $\mathcal{B}$ *be a knowledge base,* Π *be an update program, and* $Update(\mathcal{B}, \Pi)$ *be a well defined update specification. If* $\mathcal{B}'$ *is a resulting knowledge base with respect to* $Update(\mathcal{B}, \Pi)$*, then* $\mathcal{B}' \in Min(\mathcal{B}, \Pi)$.

Proof. From Proposition 3.4, each rule of Π is satisfied in $\mathcal{B}'$. Now we prove $Diff(\mathcal{B}, \mathcal{B}')$ is minimal. From the proof of Lemma 3.3, we can view $\mathcal{B}'$ as an answer set of $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <)$, where for each rule $r : L \leftarrow not\overline{L}$ and each rule r' in Π, $\mathcal{N}(r) < \mathcal{N}(r')$. So $\mathcal{B}'$ is an answer set of some reduct Π' of $\mathcal{P}$. Observing the procedure of computing Π' (i.e. Definition 2.2), it is clear that for every $L \in \mathcal{B}$, rule $L \leftarrow not\overline{L}$ in $\mathcal{P}'$ is retained in Π', and any rule in Π which has *not* L in its body is defeated and has been removed from Π' if $L \in \mathcal{B}'$. Then the result is directly followed from the fact the each consistent answer set of Π' is the smallest set of ground literals in which each rule of Π' is satisfied. □

4. RESTRICTED MONOTONICITY

An important characteristic for many formulations of knowledge representation and reasoning is that they are nonmonotonic. It has been, however, illustrated that under certain conditions, a formulation may satisfy some restricted monotonicity in its reasoning. Restricted monotonicity is a desired property for nonmonotonic formulations as it usually simplifies the reasoning procedure [Engelfreit 1998]. In this section, we explore the restricted monotonicity property for the generalized simple fact update. As we will show next, this property may significantly simplify the evaluation of an update specification. To simplify our presentation, we assume that all prioritized and extended logic programs considered in this section are ground. But note that this assumption does not restrict our results to non-ground programs since each non-ground program is evaluated based on its ground instantiation.

4.1 The Results

Given a set of ground literals $\mathcal{B}$, we use $\overline{\mathcal{B}}$ to denote the set of complement literals of $\mathcal{B}$ with respect to classical negation $\neg$.

THEOREM 4.1. *(**Restricted Monotonicity Theorem 1**) Given two knowledge bases $\mathcal{B}_1$ and $\mathcal{B}_2$ where $\mathcal{B}_1 \subseteq \mathcal{B}_2$ and an update program Π. Suppose that both update specifications $Update(\mathcal{B}_1, \Pi)$ and $Update(\mathcal{B}_2, \Pi)$ are well defined. Let $\mathcal{B}'_1$ be a resulting knowledge base with respect to $Update(\mathcal{B}_1, \Pi)$. Then there exists a resulting knowledge base $\mathcal{B}'_2$ with respect to $Update(\mathcal{B}_2, \Pi)$ such that $\mathcal{B}'_1 \subseteq \mathcal{B}'_2$ if $body(\Pi) \cap (\mathcal{B}_2 - \mathcal{B}_1) = \emptyset$. In this case, $\mathcal{B}'_2 = \mathcal{B}'_1 \cup \{L \mid L \in (\mathcal{B}_2 - \mathcal{B}_1)$ and $\overline{L} \notin \mathcal{B}'_1\}$.*

THEOREM 4.2. *(**Restricted Monotonicity Theorem 2**) Given a knowledge base $\mathcal{B}$ and two update programs Π_1 and Π_2 where $\Pi_1 \subseteq \Pi_2$. Suppose both update specifications $Update(\mathcal{B}, \Pi_1)$ and $Update(\mathcal{B}, \Pi_2)$ are well defined. Let $\mathcal{B}'$ be a resulting knowledge base with respect to $Update(\mathcal{B}, \Pi_1)$. Then there exists a resulting knowledge base $\mathcal{B}''$ with respect to $Update(\mathcal{B}, \Pi_2)$ such that $\mathcal{B}' \subseteq \mathcal{B}''$ if $head(\Pi_2 - \Pi_1) \cap (\overline{\mathcal{B}} \cup body(\Pi_1)) = \emptyset$. In this case, $\mathcal{B}''$ is an answer set of program $\{L \leftarrow \mid L \in \mathcal{B}'\} \cup (\Pi_2 - \Pi_1)$.*

The intuition behind Theorem 4.1 is described as follows. If a knowledge base $\mathcal{B}_1$ is expanded to $\mathcal{B}_2$ by adding more facts and all these added facts do not occur in the body of any rule in Π, then the result of updating $\mathcal{B}_1$ with Π is preserved in the result of updating $\mathcal{B}_2$ with Π. Furthermore, the latter can be simply computed from the result of updating $\mathcal{B}_1$ with Π. On the other hand, Theorem 4.2 says that if an update program Π_1 is expanded to Π_2 by adding more rules and the head of each added rule does not occur in the bodies of rules in update specification $Update(\mathcal{B}, \Pi_1)$ (see Lemma 3.3), then the result of updating $\mathcal{B}$ with Π_1 is preserved in the result of updating $\mathcal{B}$ with Π_2, and the latter is reduced to an answer set of a corresponding extended logic program.

Given a knowledge base $\mathcal{B}$ and an update program Π, we can apply the restricted monotonicity theorems above to simplify the computation of $Update(\mathcal{B}, \Pi)$ when $\mathcal{B}$ or Π can be split into parts. The following examples illustrate such applications.

EXAMPLE 4.1. Let $\mathcal{B} = \{A, B, C, D\}$ and Π consist of the following rules:

$\neg A \leftarrow B,$

$\neg C \leftarrow B,$
$\neg B \leftarrow not\ B.$

We consider the update of $\mathcal{B}$ with Π. Since $body(\Pi) \cap \{A, C, D\} = \emptyset$, from Theorem 4.1, we can actually reduce the update of $\mathcal{B}$ with Π into the update of $\{B\}$ with Π. It is clear that the unique result of $Update(\{B\}, \Pi)$ is $\{\neg A, B, \neg C\}$. So according to Theorem 4.1, the unique resulting knowledge base with respect to $Update(\mathcal{B}, \Pi)$ is $\{\neg A, B, \neg C, D\}$.

Let us consider another situation. Let $\mathcal{B} = \{A, B\}$ and Π be a program consisting of the following rules:

$\neg B \leftarrow not\ C,$
$\neg C \leftarrow A.$

Consider rule $r : \neg C \leftarrow A$ in Π. Since $head(r) \cap (\overline{\mathcal{B}} \cup body(\neg B \leftarrow not\ C)) = \{\neg C\} \cap \{\neg A, \neg B, C\} = \emptyset$, from Theorem 4.2, the update of $\mathcal{B}$ with Π can be reduced to the update of $\mathcal{B}$ with $\{\neg B \leftarrow not\ C\}$, which has the unique result $\{A, \neg B\}$. So the result of $Update(\mathcal{B}, \Pi)$ is an answer set of program $\{A \leftarrow, \neg B \leftarrow, \neg C \leftarrow A\}$, which has the unique answer set $\{A, \neg B, \neg C\}$.

In general, given an update specification $Update(\mathcal{B}, \Pi)$, two restricted monotonicity theorems can be applied alternatively and sequentially to split both $\mathcal{B}$ and Π into smaller parts such that the evaluation of $Update(\mathcal{B}, \Pi)$ can be significantly simplified.

Example 4.2. Let $\mathcal{B} = \{\neg A, B, C\}$ and Π be a program consisting of the following rules:

$\neg B \leftarrow not\ B,$
$A \leftarrow C.$

Consider the update of $\mathcal{B}$ with Π. Firstly, since $body(\Pi) \cap \{\neg A\} = \emptyset$, according to Theorem 4.1, we can split $\mathcal{B}$ into $\mathcal{B}_1 = \{B, C\}$ and $\mathcal{B}_2 = \{\neg A\}$, and the resulting knowledge base $\mathcal{B}'$ with respect to $Update(\mathcal{B}, \Pi)$ is then represented as

$$\mathcal{B}' = \mathcal{B}'_1 \cup \{\neg A \mid \text{ if } A \notin \mathcal{B}'_1\}, \tag{4}$$

where $\mathcal{B}'_1$ is a resulting knowledge base with respect to $Update(\{B, C\}, \Pi)$. Therefore, the evaluation of $Update(\mathcal{B}, \Pi)$ is reduced to the evaluation of $Update(\{B, C\}, \Pi)$.

It is then observed that $head(A \leftarrow C) \cap \{\neg B, \neg C, B\} = \emptyset$, according to Theorem 4.2, Π can be split into $\Pi_1 = \{\neg B \leftarrow not\ B\}$ and $\Pi_2 = \{A \leftarrow C\}$, and $\mathcal{B}'_1$ is then represented as an answer set of the following program:

$$\{L \mid L \in \mathcal{B}'_2\} \cup \{A \leftarrow C\}, \tag{5}$$

where $\mathcal{B}'_2$ is a resulting knowledge base with respect to $Update(\{B, C\}, \{\neg B \leftarrow not\ B\})$.

Since $body(\neg B \leftarrow not\ B) \cap \{C\} = \emptyset$, from Theorem 4.1 again, the update specification $Update(\{B, C\}, \{\neg B \leftarrow not\ B\})$ can be further split. That is, $\mathcal{B}'_2$ can be represented as

$$\mathcal{B}'_2 = \mathcal{B}''_2 \cup \{C \mid \text{ if } \neg C \notin \mathcal{B}''_2\}, \tag{6}$$

where $\mathcal{B}''_2$ is the resulting knowledge base with respect to $Update(\{B\}, \{\neg B \leftarrow not\ B\})$. Now it is easy to see that the unique result of $\mathcal{B}''_2$ is $\{B\}$. Then from (6), (5) and (4), we have

$\mathcal{B}'_2 = \{B, C\}$,
$\mathcal{B}'_1 = \{A, B, C\}$, and
$\mathcal{B}' = \{A, B, C\}$

respectively.

From the above reduction, we can see that the evaluation of update specification $Update(\mathcal{B}, \Pi)$ is finally reduced to the evaluation of update specification $Update(\{B\}, \{\neg B \leftarrow not\ B\})$, where the sizes of knowledge base and update program from initially three facts and two rules are reduced to one fact and one rule respectively.

4.2 Proofs of Restricted Monotonicity Theorems

In order to prove Theorems 4.1 and 4.2 described previously, we need a result about splitting extended logic programs. In particular, our proofs of Theorems 4.1 and 4.2 are based on a special case of a splitting theorem on PLPs proved in [Zhang 2003b], which we state as follows. We first introduce a notion. Given a set of ground literals S, we use $e(\Pi, S)$ to denote the program obtained from program Π by deleting: (1) each rule in Π that has a formula $notL$ in its body with $L \in S$; and (2) all formulas of form L in the bodies of the remaining rules with $L \in S$. Intuitively, $e(\Pi, S)$ can be viewed as a simplicity of Π giving those literals in S to be true.

THEOREM 4.3. *[Zhang 2003b] Let* $\Pi = \Pi_1 \cup \Pi_2$ *be an extended logic program and* $body(\Pi_1) \cap head(\Pi_2) = \emptyset$. *Then a set of literals* S *is a consistent answer set of* Π *if and only if* $S = S_1 \cup S'$, *where* S_1 *is an answer set of* Π_1, S' *is an answer set of* $e(\Pi_2, S_1)$, *and* $S_1 \cup S'$ *is consistent.*

Note that the above splitting result is more general than Lifschitz-Turner's Splitting Set Theorem on extended logic programs [Lifschitz and Turner 1994]. This can be demonstrated by the following example.

EXAMPLE 4.3. Let Π be a program consisting of the following rules:

$A \leftarrow not\ C$,
$A \leftarrow not\ B$,
$B \leftarrow not\ A$.

It is not difficult to see that this program does not have any nontrivial splitting set. Therefore, the above splitting set theorem cannot be used to compute the answer set of Π. However, Π can be split into Π_1 and Π_2 as follows:

Π_1:	Π_2:
$A \leftarrow not\ C$,	$A \leftarrow not\ B$,
	$B \leftarrow not\ A$,

such that $body(\Pi_1) \cap head(\Pi_2) = \emptyset$. It is observed that $\{A\}$ is the unique answer set of Π_1, and the unique answer set of Π is then obtained from Π_1's answer set

$\{A\}$ and the answer set of $e(\Pi_2, \{A\})$, which is also $\{A\}$. So we get the unique answer set of Π $\{A\}$.

Now based on Theorem 4.3, we are already to give complete proofs of the two restricted monotonicity theorems presented in subsection 4.1. We first prove the following lemmas.

LEMMA 4.4. *Let Π_1 and Π_2 be two extended logic programs, where each rule in Π_2 is of the form $L \leftarrow not\overline{L}$, $head(\Pi_2) \cap body(\Pi_2) = \emptyset$ and $body(\Pi_1) \cap head(\Pi_2) = \emptyset$. Suppose both Π_1 and $\Pi_1 \cup \Pi_2$ have consistent answer sets. Then S is a consistent answer set of $\Pi_1 \cup \Pi_2$ if and only if $S = S' \cup \{L \mid L \leftarrow not\overline{L} \in \Pi_2$ and $\overline{L} \notin S'\}$, where S' is a consistent answer set of Π_1.*

PROOF. From the Theorem 4.3, it concludes that S is a consistent answer set of $\Pi_1 \cup \Pi_2$ if and only if $S = S' \cup S''$, where S' is an answer set of Π_1, and S'' is an answer set of $e(\Pi_2, S')$. Since $e(\Pi_2, S') = \{L \leftarrow not\ \overline{L} \mid L \leftarrow not\ \overline{L} \in \Pi_2$ and $\overline{L} \notin S'\}$ and $head(\Pi_2) \cap body(\Pi_2) = \emptyset$, it is clear that $\{L \mid L \leftarrow not\ \overline{L} \in \Pi_2$ and $\overline{L} \notin S'\}$ is an answer set of $e(\Pi_2, S')$. So the result holds. □

LEMMA 4.5. *Let S be a consistent answer set of an extended logic program Π, and $\Pi = \Pi' - \{r_1, \cdots, r_k\}$ where rules $r_1, \cdots, r_k$ are defeated by Π. Then S is also an answer set of Π'.*

The proof of Lemma 4.5 is trivial.

Proof of Restricted Monotonicity Theorem 1 (Theorem 4.1)
From Lemma 3.3, $Update(\mathcal{B}_1, \Pi)$ and $Update(\mathcal{B}_2, \Pi)$ are equivalent to the following two PLPs respectively: $\mathcal{P}_1 = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}_1\}, \mathcal{N}, <)$ and $\mathcal{P}_2 = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}_2\}, \mathcal{N}, <)$, where $<$ relations in $\mathcal{P}_1$ are specified as stated in Lemma 3.3 respectively. Let $\mathcal{B}'_1$ be a consistent answer set of $\mathcal{P}_1$. We assume $\mathcal{B}_1 = \{L_1, \cdots, L_k\}$, and $\mathcal{B}_2 = \mathcal{B}_1 \cup \{L_{k+1}, \cdots, L_m\}$. Observing the construction of $\mathcal{P}_2$, besides rules in $\mathcal{P}_1$, $\mathcal{P}_2$ also contains following rules:

$$
\begin{aligned}
& r_{k+1} : L_{k+1} \leftarrow not\overline{L_{k+1}}, \\
& \cdots, \\
& r_m : L_m \leftarrow not\overline{L_m}.
\end{aligned}
$$

Let $\Pi' \cup \Pi''$ be a reduct of $\mathcal{P}_2$, where each rule in Π' is also in $\mathcal{P}_1$, and $\Pi'' = \{r_{k+1}, \cdots, r_m\}$. Note that since for each rule $r \in \Pi''$, there is no other rule r^* such that $\mathcal{N}(r^*) < \mathcal{N}(r)$, all rules $r_{k+1}, \cdots, r_m$ will be included in each reduct of $\mathcal{P}_2$. Now we show $\mathcal{B}'_2$ is an answer set of $\Pi' \cup \Pi''$. If Π' is a reduct of $\mathcal{P}_1$, then $\mathcal{B}'_1$ is an answer set of Π' and the result is true from Lemma 4.4.

Now suppose Π' is a proper subset of some reduct Π^* of $\mathcal{P}_1$, where $\Pi' = \Pi^* - \{r_p, \cdots, r_q\}$. Clearly, all rules $r_p, \cdots, r_q$ are eliminated from Π^* due to additional rules $r_{k+1}, \cdots, r_m$ are added into $\mathcal{P}_1$ to form $\mathcal{P}_2$. Therefore, we can assume that in the evaluation of reduct $\Pi' \cup \Pi''$ of $\mathcal{P}_2$, there exists some integer h such that

$$
\begin{aligned}
& \Pi_0 = \Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}_2\}, \\
& \quad \cdots, \\
& \Pi_h = \Pi^* \cup \Pi'', \\
& \Pi_{h+1} = \Pi_h - \{r_i, \cdots, r'_i \mid \text{there exists some } r \in \Pi'' \text{ such that}
\end{aligned}
$$

$$
\begin{aligned}
&\qquad\qquad\qquad \mathcal{N}(r) < \mathcal{N}(r_i, \cdots, r'_i) \text{ and } r_i, \cdots, r'_i \\
&\qquad\qquad\qquad \text{are defeated by } \Pi_h - \{r_i, \cdots, r'_i\}\}^{8}, \\
&\cdots, \\
&\Pi_{h+l} = \Pi_{h+l-1} - \{r'_j, \cdots, r_j \mid \text{there exists some } r \in \Pi'' \text{ such that} \\
&\qquad\qquad\qquad \mathcal{N}(r) < \mathcal{N}(r'_j, \cdots, r_j) \text{ and } r'_j, \cdots, r_j \\
&\qquad\qquad\qquad \text{are defeated by } \Pi_{h+l-1} - \{r'_j, \cdots, r_j\}\}, \\
&\cdots,
\end{aligned}
$$

where $r_i, \cdots, r'_i, \cdots, r'_j, \cdots, r_j$ are the rules eliminated from the reduct Π^* of $\mathcal{P}_1$ due to the preferences $\mathcal{N}(r) < \mathcal{N}(r_i)$, $\cdots$, $\mathcal{N}(r) < \mathcal{N}(r_j)$ for some $r \in \Pi''$. Note that since Π^* is a reduct of $\mathcal{P}_1$, no any other rules in Π^* can be further eliminated from preferences between rules in $\{L \leftarrow not\overline{L} \mid L \in \mathcal{B}_1\}$ and rules in Π.

On the other hand, since for any rule $r' \in \{L \leftarrow not\,\overline{L} \mid L \in \mathcal{B}_1\}$, $\mathcal{N}(r') < \mathcal{N}(r_i)$, $\cdots$, $\mathcal{N}(r') < \mathcal{N}(r_j)$, Π_{h+1}, $\cdots$, Π_{h+l} can be also specified as

$$
\begin{aligned}
&\Pi_{h+1} = \Pi_h - \{r_i, \cdots, r'_i \mid \text{there exists some } r' \in \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}_1\} \\
&\qquad\qquad \text{such that } \mathcal{N}(r') < \mathcal{N}(r_i, \cdots, r'_i), \text{ and} \\
&\qquad\qquad r_i, \cdots, r'_i \text{ are defeated by } \Pi_h - \{r_i, \cdots, r'_i\}\}, \\
&\cdots, \\
&\Pi_{h+l} = \Pi_{h+l-1} - \{r'_j, \cdots, r_j \mid \text{there exists some} \\
&\qquad\qquad r' \in \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}_1\} \\
&\qquad\qquad \text{such that } \mathcal{N}(r') < \mathcal{N}(r'_j, \cdots, r_j) \text{ and} \\
&\qquad\qquad r'_j, \cdots, r_j \text{ are defeated by } \Pi_{h+l-1} - \{r'_j, \cdots, r_j\}\}.
\end{aligned}
$$

But this contradicts the fact that Π^* is a reduct of $\mathcal{P}_1$ where no any other rules can be further eliminated from preferences between rule r' and rules $r_i, \cdots, r_j$. So Π' must be a reduct of $\mathcal{P}_1$. □

Proof of Restricted Monotonicity Theorem 2 (Theorem 4.2)
From Lemma 3.3, we know that $Update(\mathcal{B}, \Pi_2)$ is equivalent to a PLP $\mathcal{P} = (\{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\} \cup \Pi_2, \mathcal{N}, <)$, where for each rule $r \in \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}$ and each rule r' in Π_2, $N(r) < N(r')$. Then from Theorem 1 in [Zhang 2003b], we know that each answer set of $\mathcal{P}$ is also an answer set of extended logic program $\{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\} \cup \Pi_2$. Again from Lemma 3.3 and Theorem 1 in [Zhang 2003b], a resulting knowledge base with respect to $Update(\mathcal{B}, \Pi_1)$ can be viewed as an answer set of program $\{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\} \cup \Pi_1$. Therefore, it is sufficient to prove that each consistent answer set $\mathcal{B}''$ of Π', where $\Pi' = \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\} \cup \Pi_2$, is also an answer set of $\{L \leftarrow \mid L \in \mathcal{B}'\} \cup (\Pi_2 - \Pi_1)$, where $\mathcal{B}'$ is an answer set of $\{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\} \cup \Pi_1$. From condition $head(\Pi_2 - \Pi_1) \cap (\overline{\mathcal{B}} \cup body(\Pi_1)) = \emptyset$, this is simply followed from Theorem 4.3. □

5. PROGRAM UPDATE

From this section, we consider program updates, where a knowledge base is represented by an extended logic program and can be updated in terms of another extended logic program.

[8]Note that condition (b) of Definition 2.2 does not apply in this case. Also note that the notion $\mathcal{N}(r) < \mathcal{N}(r_i, \cdots, r'_i)$ means $\mathcal{N}(r) < \mathcal{N}(r_i)$, $\cdots$, $\mathcal{N}(r) < \mathcal{N}(r'_i)$.

5.1 The Approach

From discussions in section 1, we can see that there are two essential steps to perform an update on Π_0 with Π_1: eliminating contradictory rules from Π_0 with respect to Π_1; and solving conflicts between the remaining rules in Π_0 and Π_1. Furthermore, we should also have an explicit syntactic representation for the result of updating Π_0 with Π_1. Before we describe our ideas of dealing with program updates, we first introduce a useful concept. Let S be a consistent set of ground literals. We say S is *coherent* with an extended logic program Π if for any answer set S' of $e(\Pi, S)$ of Π with respect to S, $S \cup S'$ is consistent. Recall that $e(\Pi, S)$ is viewed as a simplified program of Π providing that all ground literals in S are true. Therefore, the intuitive meaning of S's coherence with Π is that given all ground literals of S to be true, no inconsistent fact can be derived from Π. For example, given $S = \{A, \neg B\}$ and $\Pi = \{C \leftarrow A, \neg D \leftarrow notA\}$, the only answer set of $e(\Pi, S)$ is $\{C\}$, from which it is concluded that S is coherent with Π. However, if we change S to $S' = \{A, \neg B, \neg C\}$, then S' is no longer coherent with Π.

Eliminating contradictory rules

To eliminate contradictory rules from Π_0 with respect to Π_1, it cannot be simply to extract a maximal subset Π of Π_0 by requiring $\Pi \cup \Pi_1$ to be well defined. For instance, suppose $\Pi_0 = \{A \leftarrow, B \leftarrow C\}$ and $\Pi_1 = \{C \leftarrow A, \neg B \leftarrow C\}$. Clearly, both $\Pi = \{A \leftarrow\}$ and $\Pi' = \{B \leftarrow C\}$ are maximal subsets of Π_0 such that $\Pi \cup \Pi_1$ and $\Pi' \cup \Pi_1$ are well defined. But intuitively, rule $B \leftarrow C$ represents a contradictory semantics comparing with rule $\neg B \leftarrow C$ in Π_1, and hence we would like to delete $B \leftarrow C$ instead of deleting $A \leftarrow$ from Π_0.

To achieve this purpose, proper semantic justification for contradiction elimination must be taken into account. We first *update* an answer set S_{Π_0} of Π_0 with Π_1. This is achieved by our simple fact update specification $Update(S_{\Pi_0}, \Pi_1)$. The result, as we have described previously, is a consistent set of ground literals, denoted as $S_{(\Pi_0,\Pi_1)}$. If $Update(S_{\Pi_0}, \Pi_1)$ is well defined, then $S_{(\Pi_0,\Pi_1)}$ has minimal difference from S_{Π_0} and satisfies each rule in Π_1. We then *extract* a maximal subset $\Pi_{(\Pi_0,\Pi_1)}$ of Π_0 such that $S_{(\Pi_0,\Pi_1)}$ is coherent with $\Pi_{(\Pi_0,\Pi_1)}$. The intuitive idea behind this is as follows. By updating the answer set of Π_0 with Π_1, we can eliminate the contradictory information implied by Π_0 with respect to rules of Π_1, and the result $S_{(\Pi_0,\Pi_1)}$ presents consistent information with respect to Π_1. Then by requiring $S_{(\Pi_0,\Pi_1)}$ to be coherent with a maximal subset $\Pi_{(\Pi_0,\Pi_1)}$ of Π_0, it is guaranteed that $\Pi_{(\Pi_0,\Pi_1)}$ maximally retains rules of Π_0 that do not imply any contradictory information with respect to rules of Π_1. Program $\Pi_{(\Pi_0,\Pi_1)}$ is called a *transformed program* from Π_0 with respect to Π_1. It should be noted that $\Pi_{(\Pi_0,\Pi_1)}$ could be an empty set sometimes.

Example 5.1. Consider extended logic programs Π_0 consisting of the following rules:

$A \leftarrow,$
$C \leftarrow B,$
$D \leftarrow not\ E,$

and Π_1 consisting of the following rules:

$B \leftarrow A,$
$\neg C \leftarrow B,$
$E \leftarrow not\ D.$

Clearly, Π_0 has a unique answer set $S_{\Pi_0} = \{A, D\}$. Updating S_{Π_0} with Π_1, according to our generalized simple fact update, we have the result $S_{(\Pi_0,\Pi_1)} = \{A, B, \neg C, D\}$, which has a minimal difference from S_{Π_0} and satisfies each rule in Π_1. Then it is easy to see that $\Pi_{(\Pi_0,\Pi_1)} = \{A \leftarrow, D \leftarrow not\ E\}$ is the unique maximal subset of Π_0 such that $S_{(\Pi_0,\Pi_1)}$ is coherent with $\Pi_{(\Pi_0,\Pi_1)}$. Therefore, $\Pi_{(\Pi_0,\Pi_1)}$ is the transformed program from Π_0 with respect to Π_1.

Solving conflicts

After transforming Π_0 to $\Pi_{(\Pi_0,\Pi_1)}$, we need to solve possible conflicts between rules in $\Pi_{(\Pi_0,\Pi_1)}$ and Π_1. To do so, we specify a prioritized logic program $\mathcal{P}_{(\Pi_0,\Pi_1)} = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$, where for each rule r in Π_1 and rule r' in $\Pi_{(\Pi_0,\Pi_1)}$, we specify $\mathcal{N}(r) < \mathcal{N}(r')$ because Π_1 expresses the agent's latest knowledge. Whenever there is a conflict between rules r and r' where $r \in \Pi_1$ and $r' \in \Pi_{(\Pi_0,\Pi_1)}$ respectively, r will override r', otherwise r' will be remained. Finally, we specify the possible resulting program Π'_0 after updating Π_0 with Π_1 to be a reduct of $\mathcal{P}_{(\Pi_0,\Pi_1)} = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$, i.e. $\mathcal{P}^{<}_{(\Pi_0,\Pi_1)}$ (see Definition 2.2).

Example 5.2. (Example 5.1 continued). Consider the same Π_0 and Π_1 as in Example 5.1. From the previous step of eliminating contradictory rules in Π_0, Π_0 is transformed to $\Pi_{(\Pi_0,\Pi_1)} = \{A \leftarrow,\ D \leftarrow not\ E\}$. Now we specify $\mathcal{P}_{(\Pi_0,\Pi_1)} = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$ as follows:

$N_1 : A \leftarrow,$
$N_2 : D \leftarrow not\ E,$
$N_3 : B \leftarrow A,$
$N_4 : \neg C \leftarrow B,$
$N_5 : E \leftarrow not\ D,$
$N_3 < N_1, N_3 < N_2, N_4 < N_1, N_4 < N_2,\ N_5 < N_1, N_5 < N_2.$

From Definition 2.2, it is not difficult to see that $\mathcal{P}_{(\Pi_0,\Pi_1)}$ has a unique reduct:

$A \leftarrow,$
$B \leftarrow A,$
$\neg C \leftarrow B,$
$E \leftarrow not\ D,$

which, as we expect, is the resulting program Π'_0 after updating Π_0 with Π_1.

5.2 Formal Descriptions

The next two definitions specify the transformed program from Π_0, which eliminates all contradictory rules from Π_0 with respect to Π_1, and the final resulting program after updating Π_0 with Π_1 respectively.

Definition 5.1. Given two consistent programs Π_0 and Π_1 and let S_{Π_0} be an answer set of Π_0. Suppose $S_{(\Pi_0,\Pi_1)} \in Res(Update(S_{\Pi_0}, \Pi_1))$ (see section 3.2). An extended logic program $\Pi_{(\Pi_0,\Pi_1)}$ is called a *transformed program* from Π_0 with respect to Π_1, if $\Pi_{(\Pi_0,\Pi_1)}$ is a maximal subset of the ground instantiation of Π_0 such that S_{Π_0} is coherent with $\Pi_{(\Pi_0,\Pi_1)}$.

Definition 5.2. Let $\Pi_{(\Pi_0,\Pi_1)}$ be defined as in Definition 5.1. A *specification of updating* Π_0 *with* Π_1 is specified as a PLP $P\text{-}Update(\Pi_0,\Pi_1) = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$, where for each rule r in Π_1 and each rule r' in $\Pi_{(\Pi_0,\Pi_1)}$, there is a preference relation $\mathcal{N}(r) < \mathcal{N}(r')$. A program Π_0' is called a *possible resulting program* of $P\text{-}Update(\Pi_0,\Pi_1)$ after updating Π_0 with Π_1 if Π_0' is a reduct of the ground instantiation of $P\text{-}Update(\Pi_0,\Pi_1)$.

From Definitions 5.1 and 5.2, it is obvious that given Π_0 and Π_1, both the transformed program $\Pi_{(\Pi_0,\Pi_1)}$ and resulting program Π_0' may not be unique. Ignoring details, it will not be difficult to verify that applying our formal approach described here to problems presented in Examples 5.1 and Example 5.2, we get desired solutions as proposed respectively.

Example 5.3. Given two programs Π_0:

$A \leftarrow,$
$C \leftarrow not\ B,$
$B \leftarrow not\ C,$

and Π_1:

$\neg A \leftarrow.$

Consider an update of Π_0 with Π_1. Firstly, to find out the contradictory rule in Π_0 with respect to Π_1, we update every answer set of Π_0 with Π_1. Clearly, Π_0 has two answer sets $\{A, B\}$ and $\{A, C\}$. Updating these two answer sets with Π_1, we get $\{\neg A, B\}$ and $\{\neg A, C\}$ respectively. From Definition 5.1, it is not difficult to conclude that program $\Pi_{(\Pi_0,\Pi_1)} = \{C \leftarrow not\ B,\ B \leftarrow not\ C\}$ is the unique transformed program from Π_0 with respect to Π_1. Secondly, to solve possible conflicts between rules in $\Pi_{(\Pi_0,\Pi_1)}$ and Π_1, we specify a PLP $P\text{-}Update(\Pi_0,\Pi_1) = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$ as follows:

$N_1 : C \leftarrow not\ B,$
$N_2 : B \leftarrow not\ C,$
$N_3 : \neg A \leftarrow,$
$N_3 < N_1,\ N_3 < N_2.$

Finally, it is concluded that $\{\neg A \leftarrow, C \leftarrow not\ B\}$ and $\{\neg A \leftarrow,\ B \leftarrow not\ C\}$ are the two possible resulting programs after updating Π_0 with Π_1.

Example 5.4. Consider to update Π_0:

$A \leftarrow,$
$C \leftarrow B,$
$\neg C \leftarrow B$

with Π_1:

$B \leftarrow A.$

Π_0 has a unique answer set $S_{\Pi_0} = \{A\}$. Updating S_{Π_0} with Π_1, we obtain a unique result $S_{(\Pi_0,\Pi_1)} = \{A, B\}$. From Definition 5.1, there are two different transformed programs from Π_0 with respect to Π_1 named $\Pi'_{(\Pi_0,\Pi_1)} = \{A \leftarrow,\ C \leftarrow B\}$ and $\Pi''_{(\Pi_0,\Pi_1)} = \{A \leftarrow,\ \neg C \leftarrow B\}$. Then we specify two PLPs:
$P\text{-}Update(\Pi_0, \Pi_1)' = (\Pi'_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$ and
$P\text{-}Update(\Pi_0, \Pi_1)'' = (\Pi''_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$ as follows respectively:

$N_1 : A \leftarrow,$	$N_1 : A \leftarrow,$
$N_2 : C \leftarrow B,$	$N_2 : \neg C \leftarrow B,$
$N_3 : B \leftarrow A,$	$N_3 : B \leftarrow A,$
$N_3 < N_1, N_3 < N_2.$	$N_3 < N_1, N_3 < N_2.$

Obviously, $P\text{-}Update(\Pi_0, \Pi_1)'$ and $P\text{-}Update(\Pi_0, \Pi_1)''$ have reducts:

$P \leftarrow,$
$C \leftarrow B,$
$B \leftarrow A,$

and

$A \leftarrow,$
$\neg C \leftarrow B,$
$B \leftarrow A,$

respectively, which are the possible resulting programs after updating Π_0 with Π_1.

Example 5.5. Consider to update Π_0:

$A \leftarrow,$
$\neg C \leftarrow A,$
$\neg B \leftarrow \neg C$

with Π_1:

$C \leftarrow A.$

Π_0 has a unique answer set $\{A, \neg B, \neg C\}$. By updating $\{A, \neg B, \neg C\}$ with Π_1, we obtain the result $\{A, \neg B, C\}$. Now it is clear that the transformed program from Π_0 with respect to Π_1 is $\{A \leftarrow, \neg B \leftarrow \neg C\}$. Then finally, we have the resulting program

$A \leftarrow,$
$C \leftarrow A,$
$\neg B \leftarrow \neg C,$

after updating Π_0 with Π_1. The interesting point of this example is that during the update, rule $\neg B \leftarrow \neg C$ in Π_0 will not be affected by eliminating rule $\neg C \leftarrow A$ from Π_0.

6. CHARACTERIZING PROGRAM UPDATE

6.1 Basic Properties

Now we investigate properties of the program update. Firstly, it is easy to observe that our previous simple fact update actually is a special case of our logic program update.

Proposition 6.1. *Given two extended logic programs Π_0 and Π_1 where each rule in Π_0 has the form $L \leftarrow$ and L is a ground literal. Let P-$Update(\Pi_0, \Pi_1)$ be an update specification as defined in Definition 5.2 and Π_0' a resulting program of P-$Update(\Pi_0, \Pi_1)$. Then each answer set of Π_0' is also a result with respect to simple fact update specification $Update(S_{\Pi_0}, \Pi_1)$, where $S_{\Pi_0} = \{L \mid L \leftarrow \ \in \Pi_0\}$.*

Proof. Program Π_0 has a unique answer set $S_{\Pi_0} = \{L \mid L \leftarrow \ \in \Pi_0\}$. Suppose S' is a result of $Update(S_{\Pi_0}, \Pi_1)$. Then from Definition 5.1, we need to specify a transformed program $\Pi_{(\Pi_0,\Pi_1)}$ of Π_0. From the special form of rules in Π_0, obviously $\Pi_{(\Pi_0,\Pi_1)}$ must be defined as $\{L \leftarrow \mid L \leftarrow \in \Pi_0 \text{ and } L \in S'\}$. So P-$Update(\Pi_0, \Pi_1)$ is specified as $(\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$.

On the other hand, as there does not exist a rule in $\Pi_{(\Pi_0,\Pi_1)}$ including weak negation in its body, during the computation of the reduct of P-$Update(\Pi_0, \Pi_1)$, no rule in $\Pi_{(\Pi_0,\Pi_1)}$ will be deleted. Furthermore, from the $<$-relation definition in P-$Update(\Pi_0, \Pi_1)$, no any rule in Π_1 can be eliminated either. So P-$Update(\Pi_0, \Pi_1)$ will have a unique reduct - the resulting program $\Pi_0' = \Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1$. This follows that for each answer set $S_{\Pi_0'}$ of Π_0', $S' \subseteq S_{\Pi_0'}$. Again, since S' is also a result of $Update(S_{\Pi_0}, \Pi_1)$, it follows that each rule of Π_1 is satisfied in S'. From the answer set property that each answer set of Π_0' is a smallest set of ground literals in which each rule of Π_0' is satisfied, it concludes that $S_{\Pi_0'} \subseteq S'$. So $S' = S_{\Pi_0'}$. The result holds. $\square$

Given programs Π_0 and Π_1, sometimes we would like to know under what condition(s) a literal derived (entailed) from Π_1 is still derivable from each resulting program Π_0' after updating Π_0 with Π_1. This is so called *persistence property*. Persistence is an interesting property for knowledge base update because it in some sense simplifies the inference problem from the resulting knowledge base. Under our context, persistence property can be formally described as follows: for any answer set S_1 of Π_1, there exists some answer set S_0' of resulting program Π_0' such that $S_1 \subseteq S_0'$. However, due to the defeasibility of rules in extended logic programs, this property does not hold in general. For instance, let $\Pi_0 = \{P \leftarrow\}$ and $\Pi_1 = \{Q \leftarrow notP\}$. After updating Π_0 with Π_1, there is a unique resulting program $\Pi_0 = \{P \leftarrow, Q \leftarrow notP\}$, whose answer set $\{P\}$, obviously, does not include Π_1's answer set $\{Q\}$. Nevertheless, by applying Theorem 4.3, we have the following result.

Theorem 6.2. *Let P-$Update(\Pi_0, \Pi_1)$ be an update specification and Π_0' a resulting program of P-$Update(\Pi_0, \Pi_1)$. If $head(\Pi_0) \cap body(\Pi_1) = \emptyset$, then for each answer set S_0' of Π_0', we have $S_1 \subseteq S_0'$, where S_1 is an answer set of Π_1.*

Proof. According to Definition 5.2, Π_0' is a reduct of P-$Update(\Pi_0, \Pi_1) = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$. From the fact that $\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1 \subseteq \Pi_0 \cup \Pi_1$ and $\Pi_0' \subseteq \Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1$, it follows that $\Pi_1 \subseteq \Pi_0' \subseteq \Pi_0 \cup \Pi_1$. Let $\Pi_0' = X \cup \Pi_1$, where $X \subseteq \Pi_0$. Then from the condition $head(\Pi_0) \cap body(\Pi_1) = \emptyset$, we have $head(X) \cap body(\Pi_1) = \emptyset$. From Theorem 4.3, it follows that each answer set S_0' of $X \cup \Pi_1$ can be expressed as the form $S_0' = S_1 \cup S$, where S_1 is an answer set of Π_1. So the result holds. $\square$

The following theorem further shows that under some conditions, we can also decide which rules in the initial program Π_0 will not be included in the resulting program Π_0' after updating Π_0 with Π_1.

Theorem 6.3. *Let P-$Update(\Pi_0, \Pi_1)$ be an update specification and Π_0' be a resulting program of P-$Update(\Pi_0, \Pi_1)$. If $head(\Pi_0) \cap body(\Pi_1) = \emptyset$, then Π_0' does not include any ground rules, which are ground instances of some rules in Π_0, of the form $L \leftarrow \cdots, notL', \cdots$, where L' is included in every answer set Π_1.*

Proof. Let P-$Update(\Pi_0, \Pi_1) = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$, where $\Pi_{(\Pi_0,\Pi_1)} \subseteq \Pi_0$ and for any rules $r \in \Pi_1$ and $r' \in \Pi_{(\Pi_0,\Pi_1)}$, $\mathcal{N}(r) < \mathcal{N}(r')$. We prove that each ground rule, that is a ground instance of some rule in $\Pi_{(\Pi_0,\Pi_1)}$, of the form $L \leftarrow \cdots, notL', \cdots$, where L' is in each answer set of Π_1, will be eliminated from the computation of the reduct of P-$Update(\Pi_0, \Pi_1)$.

Firstly, it is clear that in the computation of the result of P-$Update(\Pi_0, \Pi_1)$, no ground instance of some rule in Π_1 can be deleted according the the $<$-relation specification in P-$Update(\Pi_0, \Pi_1)$. In each iteration of computing the sequence $\{\Pi^i\}$ $(i = 0, 1, \cdots)$ (See Definition 2.2)[9], Π^i is obtained from Π^{i-1} by eliminating some rule in Π^i (note that this rule is in Π_0):

$$\begin{aligned}
&\Pi^0 = \text{the ground instantiation of } \Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1,\\
&\Pi^i = \Pi^{i-1} - \{r_1, r_2, \cdots \mid r_1, r_2, \cdots \text{ satisfy conditions}\\
&\qquad\qquad\qquad\qquad\qquad \text{(a) and (b) stated in Definition 2.2}\}.
\end{aligned}$$

We can also express Π^i as $\Pi^i = X \cup$ the ground instantiation of Π_1, where $X \subseteq$ the ground instantiation of Π_0. As $head(X) \cap body(\Pi_1) = \emptyset$, from the Theorem 4.3, it concludes that each answer set S^i of Π^i must include an answer set of Π_1.

If there is some rule r', that is a ground instance of a rule in $\Pi_{(\Pi_0,\Pi_1)}$, of the form $L \leftarrow \cdots, notL', \cdots$, where L' is in every answer set of Π_1, then in the sequence $\{\Pi^i\}$ $(i = 0, 1, \cdots)$, there must exist some h, such that $\Pi^h = \Pi^{h-1} - \{r', \cdots\}$ since for any ground instance r of Π_1, $\mathcal{N}(r) < \mathcal{N}(r')$, and L' is in every answer set of Π^h. Therefore, the reduct Π_0' of P-$Update(\Pi_0, \Pi_1)$ does not include any this type of rules in Π_0. $\square$

Directly from Theorem 6.3, we have the following corollary.

Corollary 6.4. *Let P-$Update(\Pi_0, \Pi_1)$ be an update specification. If $head(\Pi_0) \cap body(\Pi_1) = \emptyset$ and each rule in Π_0 has a ground instance of the form $L \leftarrow \cdots, notL', \cdots$, where L' is in every answer set of Π_1, then P-$Update(\Pi_0, \Pi_1)$ has a unique resulting program Π_0' such that Π_0' is the ground instantiation of Π_1.*

Example 6.1. Consider programs Π_0 and Π_1 as follows:

Π_0:
  $E \leftarrow notA$,
  $F \leftarrow notD$,
Π_1:
  $A \leftarrow$,
  $B \leftarrow notC$,
  $C \leftarrow notB$,
  $D \leftarrow not\neg D$.

[9]To distinguish from Π_0 and Π_1, here we use superscript Π^i to denote the program generated in each iteration of computing the sequence.

Since $head(\Pi_0) \cap body(\Pi_1) = \emptyset$, according to the above corollary, updating Π_0 with Π_1 will have a unique result $\Pi'_0 = \Pi_1$. This is also easy to verify from Definitions 5.1 and 5.2. Now we augment Π_0 by adding rule $D \leftarrow notC$ into Π_0. In this case, condition $head(\Pi_0) \cap body(\Pi_1) = \emptyset$ still holds, but in rule $D \leftarrow notC$, C is only in one of the two answer sets of Π_1, so the resulting program is no longer the same as Π_1. Instead, we obtain the unique resulting program $\{D \leftarrow notC\} \cup \Pi_1$.

6.2 Simplifying the Inference in Program Update

An important issue associated with the program update is the *inference problem*. That is, for a given program update specification $P\text{-}Update(\Pi_0, \Pi_1)$ and a ground literal L, whether L is entailed from *every* resulting program with respect to this update specification, i.e. $P\text{-}Update(\Pi_0, \Pi_1) \models L$. To simplify this inference process, we will need a new splitting theorem for prioritized logic programs that the author has studied in [Zhang 2003b]. Given an extended logic program Π and a set X of ground literals. For a rule $r \in e(\Pi, X)$, we use $original(r)$ to denote r's original form in Π. For instance, if Π consists of the following two rules:

$$A \leftarrow B, notC,$$
$$B \leftarrow C, notA,$$

and $X = \{C\}$, then $e(\Pi, X) = \{r : B \leftarrow notA\}$, where $original(r) = B \leftarrow C, notA$. Now we define a split of a PLP as follows.

Definition 6.5. [Zhang 2003b] Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$. We say that $(\mathcal{P}^1, \mathcal{P}^2)$ is a *split* of $\mathcal{P}$, if there exist two disjoint subsets Π_1 and Π_2 of Π, where $\Pi = \Pi_1 \cup \Pi_2$, such that

(1) $head(\Pi_2) \cap body(\Pi_1) = \emptyset$,

(2) $\mathcal{P}^1 = (\Pi_1 \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$[10], where for any $r, r' \in \Pi_1$, $\mathcal{N}(r) < \mathcal{N}(r') \in \mathcal{P}(<)$ implies $\mathcal{N}(r) < \mathcal{N}(r') \in \mathcal{P}^1(<)$, and if there exists some $r'' \in \Pi_2$ and $first^<(r) = r''$, then $N_0 < \mathcal{N}(r) \in \mathcal{P}^1(<)$;

(3) $\mathcal{P}^2 = (e(\Pi_2, S_1) \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where S_1 is an answer set of $\mathcal{P}^1$, for any $r, r' \in e(\Pi_2, S_1)$, $\mathcal{N}(original(r)) < \mathcal{N}(original(r')) \in \mathcal{P}(<)$ implies $\mathcal{N}(r) < \mathcal{N}(r') \in \mathcal{P}^2(<)$, and if there exists some $r'' \in \Pi_1$ and $first^<(original(r)) = r''$, then $N_0 < \mathcal{N}(r) \in \mathcal{P}^2(<)$.

A split $(\mathcal{P}^1, \mathcal{P}^2)$ is called *S-dependent* if S is an answer set of $\mathcal{P}^1$ and $\mathcal{P}^2$ is formed based on S as described in condition 3 above, i.e. $\mathcal{P}^2 = (e(\Pi_2, S) \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$.

Theorem 6.6. *[Zhang 2003b] Let $(\mathcal{P}^1, \mathcal{P}^2)$ be a S_1-dependent split of $\mathcal{P}$ as defined in Definition 6.5. A set of ground literals S is a consistent answer set of $\mathcal{P}$ if and only if $S = S_1 \cup S_2 - \{First\}$, where S_2 is an answer set of $\mathcal{P}^2$, and $S_1 \cup S_2$ is consistent.*

The above result can be extended to a general case.

[10]Here we assume that *First* is a ground literal not occurring in $\mathcal{P}$.

Definition 6.7. [Zhang 2003b] Let $\mathcal{P} = (\Pi, \mathcal{N}, <)$. We say that $(\mathcal{P}^1, \cdots, \mathcal{P}^k)$ is a *split* of $\mathcal{P}$, if there exist k disjoint subsets $\Pi_1, \cdots, \Pi_k$ of Π, where $\Pi = \bigcup_{i=1}^{k} \Pi_i$, such that

(1) $head(\Pi_i) \cap body(\bigcup_{j=1}^{i-1} \Pi_j) = \emptyset$, $(i = 2, \cdots, k)$,

(2) $\mathcal{P}^1 = (\Pi_1 \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where for any $r, r' \in \Pi_1$, $\mathcal{N}(r) < \mathcal{N}(r') \in \mathcal{P}(<)$ implies $\mathcal{N}(r) < \mathcal{N}(r') \in \mathcal{P}^1(<)$, and if there exists some $r'' \notin \Pi_1$ and $first^{<}(r) = r''$, then $N_0 < \mathcal{N}(r) \in \mathcal{P}'(<)$;

(3) $\mathcal{P}^i = (e(\Pi_i, \bigcup_{j=1}^{i-1} S_j) \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where S_j is an answer set of $\mathcal{P}^j$, for any $r, r' \in e(\Pi_i, \bigcup_{j=1}^{i-1} S_j)$, $\mathcal{N}(original(r)) < \mathcal{N}(original(r')) \in \mathcal{P}(<)$ implies $\mathcal{N}(r) < \mathcal{N}(r') \in \mathcal{P}^i(<)$, and if there exists some $r'' \notin \Pi_i$ and $first^{<}(original(r)) = r''$, then $N_0 < \mathcal{N}(r) \in \mathcal{P}^i(<)$.

A split $(\mathcal{P}^1, \cdots, \mathcal{P}^k)$ is called $\bigcup_{i=1}^{k-1} S_i$*-dependent* if S_i is an answer set of $\mathcal{P}^i$ $(i = 1, \cdots, k-1)$ and each $\mathcal{P}^{i+1}$ is formed based on $\bigcup_{j=1}^{i} S_j$ as described in condition 3 above.

Theorem 6.8. *[Zhang 2003b] Let* $(\mathcal{P}^1, \cdots, \mathcal{P}^k)$ *be a* $\bigcup_{i=1}^{k-1} S_i$-dependent *split of* $\mathcal{P}$ *as defined in Definition 6.7. A set of ground literals* S *is a consistent answer set of* $\mathcal{P}$ *if and only if* $S = \bigcup_{i=1}^{k} S_i - \{First\}$, *where* S_k *is an answer set of* $\mathcal{P}^k$, *and* $\bigcup_{i=1}^{k} S_i$ *is consistent.*

Now based on the above splitting theorem for PLPs, we can prove the following result that simplifies the inference problem in program updates.

Theorem 6.9. *Let* $P\text{-}Update(\Pi_0, \Pi_1)$ *be an update specification and* L *be a ground literal. If* $P\text{-}Update(\Pi_0, \Pi_1)$ *has a* S*-dependent split* $(\mathcal{P}^1, \mathcal{P}^2)$, *then* $P\text{-}Update(\Pi_0, \Pi_1) \models L$ *if and only if* $X \cup Y \models L$, *where* $X \cup \{First \leftarrow\}$ *and* $Y \cup \{First \leftarrow\}$ *are reducts of* $\mathcal{P}^1$ *and* $\mathcal{P}^2$ *respectively.*

Proof. We only need to prove that S is an answer set of Π_0' iff S is an answer set of $X \cup Y$. Since Π_0' is a reduct of $P\text{-}Update(\Pi_0, \Pi_1)$, S is certainly an answer set of $P\text{-}Update(\Pi_0, \Pi_1)$ as well. From Theorem 6.6, it shows that S is an answer set of $P\text{-}Update(\Pi_0, \Pi_1)$ iff $S = S_1 \cup S_2 - \{First\}$, where S_1 is an answer set of $\mathcal{P}^1$ and S_2 is an answer set of $\mathcal{P}^2$ (as here we only consider well defined update specification, the consistency condition of $S_1 \cup S_2$ is omitted). Let S_1 be an answer set of $X \cup \{First \leftarrow\}$ and S_2 be an answer set of $Y \cup \{First \leftarrow\}$. So it follows that S is an answer set of Π_0' iff S is an answer set of $X \cup Y$. The result holds. □

As mentioned earlier, to decide whether a literal is entailed from every resulting program with respect to an update specification, we need to evaluate the underlying update specification. The result stated in Theorem 6.9 simplifies such evaluation procedure. For instance, under the condition of Theorem 6.9, the evaluation of $P\text{-}Update(\Pi_0, \Pi_1)$ is reduced to the evaluation of two smaller prioritized logic programs $\mathcal{P}^1$ and $\mathcal{P}^2$. It should be noted that under the splitting condition, although the answer set of a resulting program Π_0' is characterized by the answer sets of two smaller PLPs X and Y, the syntactic form of $X \cup Y$, however, may not be the same as Π_0'. To illustrate this property, consider the following example.

Example 6.2. Let Π_0 and Π_1 be the following programs:

Π_0:
$\quad A \leftarrow notB$,
Π_1:
$\quad C \leftarrow notA$,
$\quad D \leftarrow not\neg D$.

Consider the update of Π_0 with Π_1. From Definitions 5.1 and 5.2, it is easy to see that there is a unique update specification $P\text{-}Update(\Pi_0, \Pi_1)$:

$N_1 : A \leftarrow notB$,
$N_2 : C \leftarrow notA$,
$N_3 : D \leftarrow not\neg D$,
$N_2 < N_1, N_3 < N_1$.

It is clear that $P\text{-}Update(\Pi_0, \Pi_1)$ has a unique reduct $\Pi_0 \cup \Pi_1$.

On the other hand, it is also observed that since $head(\Pi_1) \cap body(\Pi_0) = \emptyset$, a S-dependent split $(\mathcal{P}^1, \mathcal{P}^2)$ can be defined for $P\text{-}Update(\Pi_0, \Pi_1)$:

$\mathcal{P}^1 = (\Pi_0 \cup \{First \leftarrow\}, \mathcal{N}, <)$:
$\quad N_0 : First \leftarrow$,
$\quad N_1 : A \leftarrow notB$,
$\quad N_0 < N_1$,
$\mathcal{P}^2 = (e(\Pi_1, \{A, First\}) \cup \{First \leftarrow\}, \mathcal{N}, <)$:
$\quad N_0 : First \leftarrow$,
$\quad N_1 : D \leftarrow not\neg D$.

Then we obtain the unique reduct of $\mathcal{P}^1$:

$$X \cup \{First \leftarrow\} = \{A \leftarrow notB,\ First \leftarrow\},$$

and the unique reduct of $\mathcal{P}^2$:

$$Y \cup \{First \leftarrow\} = \{D \leftarrow not\neg D,\ First \leftarrow\}.$$

Obviously, $X \cup Y \neq \Pi_0 \cup \Pi_1$ although $X \cup Y$ and $\Pi_0 \cup \Pi_1$ have the same unique answer set $\{A, D\}$.

COROLLARY 6.10. *Let $P\text{-}Update(\Pi_0, \Pi_1)$ be an update specification and L be a ground literal. If $P\text{-}Update(\Pi_0, \Pi_1)$ has a $\bigcup_{i=1}^{k-1} S_i$-dependent split $(\mathcal{P}^1, \cdots, \mathcal{P}^k)$, then $P\text{-}Update(\Pi_0, \Pi_1) \models L$ if and only if $X_1 \cup \cdots \cup X_k \models L$, where $X_i \cup \{First \leftarrow\}$ is a reduct of $\mathcal{P}^i$ $(1 \leq i \leq k)$.*

7. ANALYSIS OF COMPUTATIONAL COMPLEXITY

In this section, we address the issue of computational complexity of simple fact and program updates. In general, for both simple fact and logic program updates, we will consider two complexity problems: model checking and inference - the former says that for a given update problem, deciding whether a set of literals is an answer set for a simple fact or program update specification, while the inference problem is that deciding whether a given literal is entailed from the result of the underlying simple fact or program update. In addition, since contradiction elimination is the first step in a program update, we will also consider the complexity for the check of contradiction elimination in terms of a program update. In the rest of this section, we assume that all programs in the context are finite propositional programs.

We first introduce some basic notions of the complexity theory, where further descriptions are referred to [Garey and Johnson 1979; Papadimitriou 1994]. Two important complexity classes are P and NP. The class of P includes those decision problems solvable by a polynomial-time deterministic Turing machine. The class of NP, on the other hand, consists of those decision problems solvable by a polynomial-time nondeterministic Turing machine.

The class $P^{\mathcal{C}}$ consists of the problems solvable by a polynomial-time deterministic Turing machine with an oracle for a problem from $\mathcal{C}$, where the class $NP^{\mathcal{C}}$ includes the problems solvable by a nondeterministic Turing machine with an oracle for a problem in $\mathcal{C}$. Let $\mathcal{C}$ be a class of decision problems, by co-$\mathcal{C}$ we mean the class consisting of the complements of the problems in $\mathcal{C}$.

The classes Σ_k^P and Π_k^P of the *polynomial hierarchy* are defined as follows:

$\Sigma_0^P = \Pi_0^P = P$ and
$\Sigma_k^P = NP^{\Sigma_{k-1}^P}$, $\Pi_k^P =$co-Σ_k^P for all $k > 1$.

It is easy to see that $NP = \Sigma_1^P$, co-$NP = \Pi_1^P$, and $\Sigma_2^P = NP^{NP}$. A problem A is *complete* for a class $\mathcal{C}$ if $A \in \mathcal{C}$ and for every problem B in $\mathcal{C}$ there is a polynomial transformation of B to A.

7.1 Complexity Results for Simple Fact Update

Now we consider the simple fact update. Given a knowledge base, i.e. a consistent set $\mathcal{B}$ of ground literals and an extended logic program Π, the update of $\mathcal{B}$ with Π is specified by the corresponding update specification $Update(\mathcal{B}, \Pi)$ which is a PLP as defined in Definition 3.1. From Lemma 3.3 in section 3.3, we know that any set $\mathcal{B}'$ is a resulting knowledge base with respect to a well defined $Update(\mathcal{B}, \Pi)$ if and only if $\mathcal{B}'$ is an answer set of a PLP

$$\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <), \qquad (7)$$

where for each $r : L \leftarrow not\overline{L}$ with $L \in \mathcal{B}$ and each $r' \in \Pi$, $\mathcal{N}(r) < \mathcal{N}(r')$. So we call this $\mathcal{P}$ the *equivalent PLP* of update specification $Update(\mathcal{B}, \Pi)$. From this result, it is clear that to evaluate a resulting knowledge base with respect to $Update(\mathcal{B}, \Pi)$, we only need to compute the answer set of $\mathcal{P}$. So the computational complexity of evaluating an update specification $Update(\mathcal{B}, \Pi)$ is equivalent to the the computational complexity of computing the answer set of $\mathcal{P}$.

Proposition 7.1. *Let Π be a well defined program and r a rule. Deciding whether r is defeated by Π is co-NP-complete.*

Proof. Let r have the form: $L \leftarrow L_1, \cdots, L_m, notL_{m+1}, \cdots, notL_n$. According to Definition 2.1, r is defeated by Π if and only if Π has an answer set and for every answer set S of Π, there exists some $L_i \in S$, where $m+1 \leq i \leq n$. Since Π is well defined, that is, it has consistent answer set(s), deciding whether r is defeated by Π is the same as the problem of deciding whether $\Pi \models L_i$ for some i $(m+1 \leq i \leq n)$. Now it is well known that deciding whether $\Pi \models L_i$ is co-NP-complete [Ben-Eliyahu and Dechter 1994]. □

To investigate the complexity with respect to the evaluation of a simple fact update specification, we need to provide a characterization on the answer sets of a particular class of prioritized logic programs that is related to our update formulation. For this purpose, we need a concept called *mutual defeasibility* that was first defined by the author in [Zhang 2003b].

Definition 7.2. [Zhang 2003b] Let Π be a ground extended logic program and r_p and r_q be two rules in Π. We define a set $\mathcal{D}(r_p)$ of literals with respect to r_p as follows:

$$\begin{aligned}&\mathcal{D}_0 = \{head(r_p)\};\\ &\mathcal{D}_i = \mathcal{D}_{i-1} \cup \{head(r) \mid head(r') \in pos(r) \text{ where } r \in \Pi \text{ and } r' \text{ are those}\\ &\qquad\qquad\qquad\qquad \text{rules such that } head(r') \in \mathcal{D}_{i-1}\};\\ &\mathcal{D}(r_p) = \bigcup_{i=1}^{\infty} \mathcal{D}_i.\end{aligned}$$

We say that r_q is *defeasible through* r_p in Π if and only if $neg(r_q) \cap \mathcal{D}(r_p) \neq \emptyset$. r_p and r_q are called *mutually defeasible* in Π, denoted as $mutual(r_p, r_q)$, if r_q is defeasible through r_p and r_p is defeasible through r_q in Π.

Intuitively, if r_q is defeasible through r_p in Π, then there exists a sequence of rules $r_1, r_2, \cdots, r_l, \cdots$ such that $head(r_p)$ occurs in $pos(r_1)$, $head(r_i)$ occurs in $pos(r_{i+1})$ for all $i = 1, \cdots$, and for some k, $head(r_k)$ occurs in $neg(r_q)$. Under this condition, it is clear that by triggering rule r_p in Π, it is possible to defeat rule r_q if rules $r_1, \cdots, r_k$ are triggered as well. As a special case that $\mathcal{D}(r_p) = \{head(r_p)\}$, r_q is defeasible through r_p iff $head(r_p) \in neg(r_q)$. Given an answer set S of Π, r_q is defeasible through r_p, and S indeed defeats r_q but does not defeat r_p and r_p is effective in S (i.e. $pos(r_p) \cup head(r_p) \subseteq S$), we then say that S *defeats* r_q *through* r_p. From Definition 7.2, it is also easy to verify that checking whether two rules in Π are mutually defeasible and whether S defeats r_q through r_p can be achieved in polynomial time.

Given a PLP $\mathcal{P} = (\Pi_1 \cup \Pi_2, \mathcal{N}, <)$, where for each $<$-relation $\mathcal{N}(r) < \mathcal{N}(r')$ in $\mathcal{P}$, $r \in \Pi_1$ and $r' \in \Pi_2$. We specify $R^<(S)$ to be a subset of Π_1 in which each rule is *not* defeated by S. Now we define the maximal $<$-consistency as follows.

Definition 7.3. Given a PLP $\mathcal{P} = (\Pi_1 \cup \Pi_2, \mathcal{N}, <)$, where for each $\mathcal{N}(r) < \mathcal{N}(r')$ in $\mathcal{P}$, $r \in \Pi_1$ and $r' \in \Pi_2$, and S and S' two answer sets of $\Pi_1 \cup \Pi_2$. We say that S is *as $<$-consistent as* S' with respect to $\mathcal{P}(<)$, denoted as $R^<(S') \sqsubseteq R^<(S)$, iff (1) $R^<(S') \subseteq R^<(S)$, and (2) there do not exist rules $r_s \in \Pi_1$, $r_p, r_q \in \Pi_2$ such that (i) S defeats r_q through r_p (not through r_s) and S does not defeat r_s, and (ii) S' defeats r_s and r_p through r_q[11]. $R^<(S') \sqsubset R^<(S)$ if $R^<(S') \sqsubseteq R^<(S)$ and $R^<(S) \not\sqsubseteq R^<(S')$. S is *maximally $<$-consistent* with respect to $\mathcal{P}(<)$ if there does not exist another S'' such that $R^<(S) \sqsubset R^<(S'')$.

The intuitive meaning of the above definition is as follows. A maximal $<$-consistent answer set S of $\Pi_1 \cup \Pi_2$ with respect to $\mathcal{P}(<)$ defeats a minimal number of rules in Π_1, *or* a non-minimal number of rules in Π_1 is defeated by S due to the conflict between rules in Π_2. Consider the following example.

[11]Note that Conditions (i) and (ii) imply that $mutual(r_p, r_q)$.

Π_1:
$r_1 : A \leftarrow notB$,
Π_2:
$r_2 : C \leftarrow notB$,
$r_3 : B \leftarrow notA, notC$.

We specify $\mathcal{P} = (\Pi_1 \cup \Pi_2, \mathcal{N}, <)$, where $\mathcal{N}(r_1) < \mathcal{N}(r_2, r_3)$. It is easy to see that $\Pi_1 \cup \Pi_2$ has two answer sets $S = \{A, C\}$ and $S' = \{B\}$. From Definition 7.3, we know that S is the maximal $<$-consistent answer set of $\Pi_1 \cup \Pi_2$ and S' is not because $R^<(S') \sqsubset R^<(S)$. It is observed that S is also an answer set of $\mathcal{P}$ where S' is not. Now if we change r_1 in Π_1 to $r_1' : D \leftarrow notB$, and all other parts of $\mathcal{P}$ remain the same. Then $S = \{C, D\}$ and $S' = \{B\}$ are both maximal $<$-consistent answer sets of $\Pi_1 \cup \Pi_2$ with respect to $\mathcal{P}(<)$. Again, we can verify that both S and S' are also answer sets of $\mathcal{P}$. This example leads us to gain the following general result.

LEMMA 7.4. *let $\mathcal{P} = (\Pi_1 \cup \Pi_2, \mathcal{N}, <)$ be a PLP, where for each $<$-relation $\mathcal{N}(r) < \mathcal{N}(r')$ in $\mathcal{P}$, $r \in \Pi_1$ and $r' \in \Pi_2$. A set of ground literals S is an answer set of $\mathcal{P}$ if and only if S is a maximal $<$-consistent answer set of $\Pi_1 \cup \Pi_2$ with respect to $\mathcal{P}(<)$.*

PROOF. ($\Rightarrow$) Let S be a maximal $<$-consistent answer set of $\Pi_1 \cup \Pi_2$ with respect to $\mathcal{P}(<)$. We prove that S is also an answer set of $\mathcal{P}$. According to Theorem 1 in [Zhang 2003b], S is an answer set of $\mathcal{P}$ if and only if there exists some reduct chain $\{\Pi_i'\}$ $(i = 0, 1, \cdots)$ such that S is an answer set of each Π_i'. On the other hand, from the $<$-relations specified in $\mathcal{P}$, we know that any rule in Π_1 must be included in every reduct of $\mathcal{P}$. So in fact we can represent a reduct chain as the form $\{\Pi_1 \cup \Pi_i'\}$ $(i = 0, 1, , \cdots)$. We prove our result by induction on i. Firstly, since $\Pi_1 \cup \Pi_0' = \Pi_1 \cup \Pi_2$, S is an answer set of $\Pi_1 \cup \Pi_0'$. Assume that S is an answer set of $\Pi_1 \cup \Pi_p'$. We need to show that S is also an answer set of $\Pi_1 \cup \Pi_{p+1}'$. From Definition 2.2, Π_{p+1}' is obtained from Π_p' as follows:

$$\Pi_1 \cup \Pi_{p+1}' = \Pi_1 \cup \Pi_p' - \{r_1, \cdots, r_l\},$$

where there exists a rule r in Π_1 such that $\mathcal{N}(r) < \mathcal{N}(r_j)$ $(j = 1, \cdots, l)$, and $r_1, \cdots, r_l$ are defeated by $\Pi_1 \cup \Pi_{p+1}'$.

Now assume S is *not* an answer set of $\Pi_1 \cup \Pi_{p+1}'$. Then from Theorem 1 in [Zhang 2003b], S cannot be an answer set for any $\Pi_1 \cup \Pi_j'$ for all $j > p + 1$. Therefore, there must exist another answer set S' of $\Pi_1 \cup \Pi_2$, which is an answer set of each $\Pi_1 \cup \Pi_i'$ in the reduct chain $\{\Pi_1 \cup \Pi_i'\}$ and hence S' is an answer set of $\mathcal{P}$. Note that S' must defeat each rule in $\{r_1, \cdots, r_l\}$. Since S is not an answer set of $\Pi_1 \cup \Pi_{p+1}'$, there must exist some rule $r^* \in \{r_1, \cdots, r_l\}$ such that S does not defeat r^*, where S' defeats r^*, otherwise S becomes an answer set of $\Pi_1 \cup \Pi_{p+1}'$. Accordingly, there must exist a rule $r \in \Pi_1$ such that r is defeated by S through $r^* \in \Pi_p'$.

On the other hand, since each rule of Π_1 cannot be eliminated from the generation of the reduct chain, it implies that each rule in Π_1 defeated by S' is due to (1) some conflict rule(s) in Π_1 itself, or (2) through rule r^* in Π_2. For case (1), it is always possible to select a particular S' such that $R^<(S) \subset R^<(S')$. It is also concluded that there does not exist another rule $r_q \in \Pi_p'$ such that the rule in Π_1

defeated by S' is through r_q. Therefore, from Definition 7.3, $R^<(S) \sqsubset R^<(S')$. But this contradicts the fact that S is maximally $<$-consistent. So S must be an answer set of $\Pi \cup \Pi'_{p+1}$. For case (2), there must exist some rule $r_q \in \Pi'_p$ such that S defeats r_q through r^*. Therefore, we can always construct another reduct chain $\{\Pi_1 \cup \Pi''_i\}$ $(i = 0, 1, \cdots)$ such that for $i \leq p$, $\Pi''_i = \Pi'_i$, and for $i = p+1$, $\Pi_1 \cup \Pi''_{p+1} = \Pi \cup \Pi'_p - \{r_q, \cdots\}$ where S is an answer set of $\Pi_1 \cup \Pi''_{p+1}$. This also prove that S is an answer set of $\mathcal{P}$.

($\Leftarrow$) Suppose that S is not a maximal $<$-consistent answer set of $\Pi_1 \cup \Pi_2$ with respect to $\mathcal{P}(<)$. We will show that S is not an answer set of $\mathcal{P}$. Let S' be a maximal $<$-consistent answer set of $\Pi_1 \cup \Pi_2$ with respect to $\mathcal{P}(<)$ such that $R^<(S) \sqsubset R^<(S')$ and condition (2) in Definition 7.3 also holds. From the above, we know that S' is an answer set of $\mathcal{P}$. Again from Theorem 1 in [Zhang 2003b], to prove S is not an answer set of $\mathcal{P}$, we only need to show that for some reduct chain $\{\Pi^i\}$ $(i = 0, 1, \cdots)$, S is not an answer set for some Π^k.

Without loss of generality, we assume that S' is an answer set of each Π^i from the reduct chain $\{\Pi^i\} = \{\Pi_1 \cup \Pi'_i\}$ $(i = 0, 1, \cdots)$ and prove that S cannot be an answer set for some $\Pi_1 \cup \Pi'_{p+1}$ in the reduce chain $\{\Pi_1 \cup \Pi'_i\}$ $(i = 0, 1, \cdots)$. From $R^<(S) \sqsubset R^<(S')$, it follows that there is some $i \leq p$: $\Pi_1 \cup \Pi'_i$ $(i = 0, \cdots, p)$ such that $R^<(S) = R^<(S')$ for all $\Pi_1 \cup \Pi'_i$ but $R^<(S) \subset R^<(S')$ for $\Pi_1 \cup \Pi'_{p+1}$, where

$$\Pi_1 \cup \Pi'_{p+1} = \Pi_1 \cup \Pi'_p - \{r_1, \cdots, r_l\}.$$

This means that S must defeat some rule r_s in Π_1 where S' does not. From Definition 7.3, it is clear that there *does not* exist rules $r_p, r_q \in \Pi_2$ such that S' defeats r_q through r_p (not through r_s) and S' does not r_s; and S defeats r_s and r_p through r_q. This implies that rule r_s defeated by S is due to a conflict rule $r' \in \Pi'_p$ which is not defeated from Π'_{p+1}. That is, $r' \notin \{r_1, \cdots, r_l\}$. Therefore, S cannot be an answer set of $\Pi_1 \cup \Pi'_{p+1}$. Consequently, S is not an answer set of $\mathcal{P}$. $\square$

THEOREM 7.5. *Let $\mathcal{P}$ be the corresponding PLP of Update$(\mathcal{B}, \Pi)$ as specified by (7) (see the beginning of section 7.1) and S a set of ground literals. Deciding whether S is an answer set of $\mathcal{P}$ is co-NP-complete.*

PROOF. According to our previous description, $\mathcal{P}$ is a PLP of the form $\mathcal{P} = (\Pi \cup \{L \leftarrow \overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <)$, where for each $r : L \leftarrow \overline{L}$ with $L \in \mathcal{B}$ and each $r' \in \Pi$, $\mathcal{N}(r) < \mathcal{N}(r')$. So $\mathcal{P}$ satisfies the condition of Lemma 7.4.

Membership proof. From Lemma 7.4, S is *not* an answer set of $\mathcal{P}$ iff there exists some S' which is an answer set of $\Pi \cup \{L \leftarrow \overline{L} \mid L \in \mathcal{B}\}$, and $R^<(S) \sqsubset R^<(S')$. A guess of such S' and check whether $R^<(S) \sqsubset R^<(S')$ can be done in polynomial time. So the problem is in NP. Consequently, the complement of the problem is in co-NP.

Hardness proof. We reduce the well known NP-complete satisfiability problem to the complement of our problem. In particular, for a given collection of nonempty propositional clauses $\mathcal{C} = \{C_1, \cdots, C_m\}$ on propositional letters $P_1, \cdots, P_n$, we construct a PLP $\mathcal{P} = (\Pi_1 \cup \Pi_2, \mathcal{N}, <)$, where each rule in Π_1 has the form $L \leftarrow not\overline{L}$, and $\mathcal{P}(<)$ is defined to be the set $\mathcal{N}(r) < \mathcal{N}(r')$ for any $r \in \Pi_1$ and $r' \in \Pi_2$, such that an answer set S of $\Pi_1 \cup \Pi_2$ is not maximally $<$-consistent with respect to $\mathcal{P}(<)$ iff $\mathcal{C}$ is satisfiable. Since our construction is in polynomial time, this proves co-NP-hardness.

The program $\mathcal{P} = (\Pi_1 \cup \Pi_2, \mathcal{N}, <)$ involves the following different propositional letters $P_1, \cdots, P_n, X, Y, Sat, Unsat$. Firstly, we define Π_1 just includes one rule:

$r_1 : X \leftarrow not \neg X.$

Π_2, on the other hand, consists of three groups of rules as follows.

G_1:
$P_i \leftarrow not \neg P_i,$
$\neg P_i \leftarrow not P_i$, for $i = 1, \cdots, n$.

For each clause $C_i = \{L_{i,1}, \cdots, L_{i,k}\}$, we specify

G_2;
$Unsat \leftarrow \overline{L_{i,1}}, \cdots, \overline{L_{i,k}}.$

Finally, we specify a group of rules:

G_3:
$Sat \leftarrow not Unsat,$
$\neg X \leftarrow Sat,$
$Y \leftarrow not X.$

Therefore, $\Pi_2 = G_1 \cup G_2 \cup G_3$.

Now we consider the extended logic program $\Pi_1 \cup \Pi_2$. Due to rules in group G_1, it is easy to see that the answer sets of $\Pi_1 \cup \Pi_2$ correspond one-to-one to the truth assignments ϕ to propositional letters $P_1, \cdots, P_n$. For each ϕ, the corresponding answer set S includes those P_i such that $\phi(P_i) = true$, and $\neg P_i$ such that $\phi(P_i) = false$. Furthermore, if ϕ satisfies $\mathcal{C}$, then S contains Sat, $\neg X$ and Y; and if ϕ does not satisfy $\mathcal{C}$, then S contains $Unsat$ and X.

Program $\mathcal{P}$ is then specified as $(\Pi_1 \cup \Pi_2, \mathcal{N}, <)$, where $\mathcal{N}(r_1) < \mathcal{N}(r)$ for any rule $r \in \Pi_2$. It is clear that for each answer set S of $\Pi_1 \cup \Pi_2$, if $\mathcal{C}$ is satisfied, S defeats rule $r_1 : X \leftarrow not \neg X$; and if $\mathcal{C}$ is not satisfied, S defeats rule $Y \leftarrow not X$. This follows that S is *not* maximally $<$-consistent with respect to $\mathcal{P}(<)$ if $\mathcal{C}$ is satisfiable and S is maximally $<$-consistent with respect to $\mathcal{P}(<)$ if $\mathcal{C}$ is not satisfiable. Since $\mathcal{P}$ can be constructed in polynomial time, this proves our result. $\square$

Now we characterize the complexity of the inference problem problem for simple fact update specifications. Let $Update(\mathcal{B}, \Pi)$ be a simple fact update specification and $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L}\}, \mathcal{N}, <)$ be the equivalence PLP of $Update(\mathcal{B}, \Pi)$, where $\mathcal{P}(<)$ is defined as follows: for each rule $r : L \leftarrow not\overline{L}$ with $L \in \mathcal{B}$ and each rule $r' \in \Pi$, $\mathcal{N}(r) < \mathcal{N}(r')$. Clearly, if $\mathcal{P}$ has a unique reduct, say Π^*, then for a give ground literal L, the problem of deciding whether $\mathcal{P} \models L$ is reduced to the problem of deciding whether $\Pi^* \models L$. As Π^* is an extended logic program, and we know that the inference problem for extended logic program is co-NP-complete [Ben-Eliyahu and Dechter 1994]. This observation motivates this following result.

THEOREM 7.6. *Let $Update(\mathcal{B}, \Pi)$ be a simple fact update specification, and $\mathcal{P} = (\Pi \cup \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}, \mathcal{N}, <)$ be the equivalence PLP of $Update(\mathcal{B}, \Pi)$, where $\mathcal{P}(<)$ is specified as before. If $\mathcal{B} \cap body(\Pi) = \emptyset$ and Π is locally stratified, then for a given ground literal L, deciding whether $\mathcal{P} \models L$ is co-NP-complete.*

PROOF. Since $\mathcal{B} \cap body(\Pi) = \emptyset$, from Definition 6.5, it is easy to see that $\mathcal{P}$ has a split $(\mathcal{P}^1, \mathcal{P}^2)$: $\mathcal{P}^1 = (\Pi \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where $N_0 < \mathcal{N}(r)$ for each $r \in \Pi$; $\mathcal{P}^2 = (e(\Pi', S \cup \{First\}) \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where $\Pi' = \{L \leftarrow not\overline{L} \mid L \in \mathcal{B}\}$, S is an answer set of Π, and $\mathcal{P}(<) = \emptyset$. Clearly, $\mathcal{P}^1$ has a unique reduct $\Pi \cup \{N_0 : First \leftarrow\}$, and $\mathcal{P}^2$ has a unique reduct $e(\Pi', S \cup \{First\}) \cup \{N_0 : First \leftarrow\}$. From the given condition, since Π is locally stratified, S is the unique answer set of Π and hence $S \cup \{First\}$ is the unique answer set of $\Pi \cup \{N_0 : First \leftarrow\}$.

Now we show that S^* is an answer set of $\mathcal{P}$ iff S^* is an answer set of $\Pi \cup e(\Pi', S)$. From the Unique Answer Set Theorem - Theorem 4 in [Zhang 2003b], we know that $\mathcal{P}$ has a unique reduct, say Π^*. So it follows that S^* is an answer set of $\mathcal{P}$ iff S^* is an answer set of Π^*. Therefore, we only need to prove that S^* is an answer set of Π^* iff S^* is an answer set of $\Pi \cup e(\Pi', S)$. From Theorem 6.6, it is easy to see that S^* is an answer set of Π^* iff $S^* = S \cup S'$, where $S \cup \{First\}$ is the unique answer set of $\mathcal{P}^1$ and $S' \cup \{First\}$ is an answer set of $\mathcal{P}^2$. So S^* is an answer set of Π^* iff $S^* = S \cup S'$, where S is an answer set of Π and S' is an answer set of $e(\Pi', S)$. Since $head(e(\Pi', S)) \cap body(\Pi) = \emptyset$, from Theorem 4.3, $S \cup S'$ is an answer set $\Pi \cup e(\Pi', S)$. So it follows that S^* is an answer set of Π^* iff S^* is an answer of $\Pi \cup e(\Pi', S)$.

So far, we have showed that for a given ground literal L, deciding whether $\mathcal{P} \models L$ is equivalent to deciding whether $\Pi \cup e(\Pi', S) \models L$, where S is the unique answer set of Π.

Now we prove the membership. A guess for an answer set S of Π and testing whether S is an answer set of Π can be done in polynomial time, and the computation of $\Pi \cup e(\Pi', S)$'s answer set S^* (i.e. $S^* = S \cup \{L \mid L \in \mathcal{B}$ and $\overline{L} \notin S\}$) and testing L not in S^* can be done in polynomial time. So the complement of the problem is in co-NP. The hardness follows from the result Corollary 4.3 in [Ben-Eliyahu and Dechter 1994]. □

COROLLARY 7.7. *Let $Update(\mathcal{B}, \Pi)$ be a simple fact update specification. If $\mathcal{B} \cap body(\Pi) = \emptyset$ and Π has a unique answer set S. Then the resulting knowledge base $\mathcal{B}'$ with respect to $Update(\mathcal{B}, \Pi)$ can be computed in $\mathcal{O}(|m| \cdot |n|)$ time, i.e. in polynomial time, where $|m|$ and $|n|$ are cardinalities of sets $\mathcal{B}$ and S respectively.*

PROOF. From the proof of Theorem 7.6, we know that $Update(\mathcal{B}, \Pi)$ has a unique resulting knowledge $\mathcal{B}' = S \cup \{L \mid L \in \mathcal{B}$ and $\overline{L} \notin \mathcal{B}\}$. Clearly, $\mathcal{B}'$ can be computed in $\mathcal{O}(|m| \cdot |n|)$ time. □

7.2 Complexity Results for Program Update

Now we consider the complexity of program update. Given extended logic programs Π_0 and Π_1, updating Π_0 with Π_1 consists of two steps: the first step is to obtain a subset $\Pi_{(\Pi_0,\Pi_1)}$ with respect to some answer set of Π_0, which is to eliminate those contradictory rules from Π_0, and then to specify the update specification $P\text{-}Update(\Pi_0, \Pi_1)$, which is to solve the conflict between $\Pi_{(\Pi_0,\Pi_1)}$ and Π_1. Now we first analyze the computational complexity for the first step - contradiction elimination.

LEMMA 7.8. *Given an extended logic program Π and a set of ground literals S. S is coherent with Π if and only if program $\Pi \cup \{L \leftarrow \mid L \in S\}$ is well defined.*

Proof. Let $\Pi' = \Pi \cup \{L \leftarrow \mid L \in S\}$. We only need to prove that each answer set of Π' can be expressed as $S' \cup S$, where S' is an answer set of $e(\Pi, S)$.

As for each rule $L \leftarrow$ where $L \in S$, L is in every answer set of Π', so any rule in Π of the form $L' \leftarrow \cdots, notL, \cdots$ can be eliminated from Π' without affecting any answer set of Π'. Also, each rule in Π' of the form $L' \leftarrow \cdots, L, \cdots$ can be simplified by omitting L from its body as L is always true in every answer set of Π'. Therefore, Π' can be simplified as $\Pi' = e(\Pi, S) \cup \{L \leftarrow \mid L \in S\}$. Let $\Pi'' = \{L \leftarrow \mid L \in S\}$. Since $head(e(\Pi, S)) \cap body(\Pi'') = \emptyset$ and $head(\Pi'') \cap body(e(\Pi, S)) = \emptyset$, from Theorem 4.3, we conclude that each answer set of Π' can be expressed as $S' \cup S$, where S' is an answer set of $e(\Pi, S)$. □

Theorem 7.9. *Let S be a consistent set of ground literals and Π a program. Deciding whether S is coherent with Π is NP-complete.*

Proof. From Lemma 7.8, this problem is equivalent to the problem of deciding whether $\Pi \cup \{L \leftarrow \mid L \in S\}$ is well defined. This problem can be equivalently interpreted to be the problem of deciding whether a corresponding propositional normal logic program has a stable model, which is known to be NP-complete [Marek and Truszczyński]. □

Theorem 7.10. *Let S be a consistent set of ground literals, Π a program, and Π' a subset of Π. Deciding whether Π' is a maximal subset of Π such that S is coherent with Π' is co-NP-complete.*

Proof. Membership proof. If Π' is not such a subset of Π, then there must exist a Π'' including Π' such that S is coherent with Π''. From Theorem 7.9, it is clear that such Π'' can be guessed and verified in polynomial time. So the problem is in NP. Consequently, the complement of the problem is in co-NP.

Hardness proof. From Lemma 7.8, the statement that Π' is a maximal subset of Π such that S is coherent with Π' can be equivalently expressed that Π' is a maximal subset of Π such that $\Pi' \cup \{L \leftarrow \mid L \in S\}$ is well defined. We will reduce the well known NP-complete 3-satisfiability problem to the complement of our problem. In particular, for a given collect of nonempty propositional clauses $\mathcal{C} = \{C_1, C_2, \cdots, C_m\}$ on propositional letters $P_1, \cdots, C_n$, where $C_i = \{L_{i,1}, L_{i,2}, L_{i,3}\}$ and $L_{i,j}$ is a literal over $P_1, \cdots, P_n$, we construct extended logic programs Π and Π^* over propositional letters $P_1, \cdots, P_n$, and $Sat, Unsat, X_1, \cdots, X_k$. Then we prove that $\mathcal{C}$ is *not* satisfiable iff a subset Π' of Π, where $\Pi' \cup \Pi^*$ has a consistent answer set, is maximal.

Firstly, we specify Π to include the following group of rules:

G_1:
$P_i \leftarrow not\neg P_i$,
$\neg P_i \leftarrow notP_i$, for each $i = 1, \cdots, n$.

For each clause $C_i = \{L_{i,1}, L_{i,2}, L_{i,3}\}$, we specify:

G_2:
$Unsat \leftarrow \overline{L_{i,1}}, \overline{L_{i,2}}, \overline{L_{i,3}}$.

Finally, we specifythe following group of rule:

G_3:
  $Sat \leftarrow notUnsat$,
  $\neg X_1 \leftarrow Unsat$.

So far, we have $\Pi = G_1 \cup G_2 \cup G_3$. It is easy to observe that the answer sets of Π correspond one-to-one the the truth assignments ϕ to propositional letters $P_1, \cdots, P_n$. For each ϕ, the corresponding answer set S of Π includes those P_i such that $\phi(P_i) = true$ and $\neg P_i$ such that $\phi(P_i) = false$. Furthermore, if $\mathcal{C}$ is satisfiable, then S includes Sat and otherwise S includes $Unsat$ and $\neg X_1$. Let Π' be a subset of Π by setting

$$\Pi' = \Pi - \{\neg X_1 \leftarrow Unsat\}.$$

We specify Π^* to only include rules of the following form:

$$X_j \leftarrow, \text{ for each } j = 1, \cdots, k.$$

Now we consider program $\Pi' \cup \Pi^*$. Clearly, $\Pi' \cup \Pi^*$ has consistent answer sets. If $\mathcal{C}$ is *not* satisfiable, then we can verified that Π' is the unique maximal subset of Π such that $\Pi' \cup \Pi^*$ has a consistent answer set. This is because that if $\mathcal{C}$ is *not* satisfiable, $Unsat$ will be included in the corresponding answer set of $\Pi' \cup \Pi^*$. As rule $\neg X_1 \leftarrow Unsat$ is not included in Π', this ensures that the answer set of $\Pi' \cup \Pi^*$ is still consistent. If add rule $\neg X_1 \leftarrow Unsat$ into Π', the consistency of the answer set of $\neg X_1 \leftarrow Unsat$ will no longer remain. Since Π, Π' and Π^* are constructed in polynomial time, this proves the hardness of the problem. □

The following theorem states that the check for contradiction elimination in the first step of a program update is co-NP-complete.

THEOREM 7.11. *Let P-Update*(Π_0, Π_1) *be a logic program update specification. Deciding whether a set of ground literals S is an answer set of some resulting program of P-Update*(Π_0, Π_1) *is co-NP-complete.*

PROOF. The hardness can be proved in a similar way described in the proof of Theorem 7.5. Now we show that membership. Suppose S is *not* an answer set of *P-Update*(Π_0, Π_1). Then according to Lemma 7.4, there must exist an answer set S' of $\Pi_{(\Pi_0,\Pi_1)}$ such that $R^<(S) \sqsubset R^<(S')$. Obviously, a guess for an answer set S' of Π and testing $R^<(S) \sqsubset R^<(S')$ can be done in polynomial time. So the problem is in NP. Consequently, the complement of the problem is in co-NP. □

Similarly to the case of the simple fact update, we can also characterize a particular class of program update specifications where the inference problem associated to these update problems is reduced to co-NP-complete.

THEOREM 7.12. *Let P-Update*$(\Pi_0, \Pi_1) = (\Pi_{(\Pi_0,\Pi_1)} \cup \Pi_1, \mathcal{N}, <)$ *be a program update specification. If* $head(\Pi_1) \cap body(\Pi_0) = \emptyset$ *and* Π_0 *is locally stratified, then for a given L, deciding whether P-Update*$(\Pi_0, \Pi_1) \models L$ *is co-NP-complete.*

PROOF. Since $head(\Pi_1) \cap body(\Pi_0) = \emptyset$, according to Definition 6.5, *P-Update*(Π_0, Π_1) has a split $(\mathcal{P}^1, \mathcal{P}^2)$: $\mathcal{P}^1 = (\Pi_{(\Pi_0,\Pi_1)} \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where $N_0 < \mathcal{N}(r)$ for all $r \in \Pi_{(\Pi_0,\Pi_1)}$; $\mathcal{P}^1 = (e(\Pi_1, S \cup \{First\}) \cup \{N_0 : First \leftarrow\}, \mathcal{N}, <)$, where S is an answer set of $\Pi_{(\Pi_0,\Pi_1)}$, and $\mathcal{P}^2(<) = \emptyset$. Obviously, $\mathcal{P}^1$

has a unique reduct $\Pi_{(\Pi_0,\Pi_1)} \cup \{N_0 : First \leftarrow\}$, where $\mathcal{P}^2$ has a unique reduct $e(\Pi_1, S \cup \{First\}) \cup \{N_0 : First \leftarrow\}$. Since Π_0 is locally stratified, it follows that $\Pi_{(\Pi_0,\Pi_1)}$ is also locally stratified. So S is the unique answer set of $\Pi_{(\Pi_0,\Pi_1)}$.

Now we show that S^* is an answer set of $P\text{-}Update(\Pi_0, \Pi_1)$ iff S^* is an answer set of $\Pi_{(\Pi_0,\Pi_1)} \cup e(\Pi_1, S)$. Since $\Pi_{(\Pi_0,\Pi_1)}$ is locally stratified, from Proposition 7.1 and the Unique Answer Set Theorem (Theorem 4) in [Zhang 2003b], $P\text{-}Update(\Pi_0, \Pi_1)$ has a unique reduct, say Π^*. Hence we only need to show that S^* is an answer set of Π^* iff S^* is an answer set of $\Pi_{(\Pi_0,\Pi_1)} \cup e(\Pi_1, S)$. From Theorem 6.6, S^* is an answer set of Π^* iff $S^* = S \cup S'$, where S is an answer set of $\Pi_{(\Pi_0,\Pi_1)}$ and S' is an answer set of $e(\Pi_1, S)$. On the other hand, as $head(e(\Pi_1, S)) \cap body(\Pi_{(\Pi_0,\Pi_1)}) = \emptyset$, from Theorem 4.3, $S \cup S'$ is an answer set of $\Pi_{(\Pi_0,\Pi_1)} \cup e(\Pi_1, S)$.

Therefore, for a given literal L, deciding whether $P\text{-}Update(\Pi_0, \Pi_1) \models L$ is equivalent to deciding whether $\Pi_{(\Pi_0,\Pi_1)} \cup e(\Pi_1, S) \models L$. This, from the proof of Theorem 7.10, is co-NP-complete. $\square$

8. COMPARISONS WITH RELATED WORK

As we mentioned earlier, problems of logic program based updates have been extensively studied in recent years, e.g. [Alferes et al. 1998; Alferes et al. 2002; Dekhtyar et al. 1998; Guessoum and Lloyd 1991; Kakas and Mancarella 1990]. In general, we can classify most of current approaches into two major categories: model based and syntax based. A logic program update formulation is *model based* if an update is achieved by considering the underlying semantics of logic programs and the final update result is usually characterized in terms of the corresponding semantics of the logic programs, e.g. stable model/answer set semantics, e.g. [Alferes et al. 1998; Eiter et al. 2002]. On the other hand, a logic program update formulation is *syntax based* if an update is performed in a syntactic way and the update result is characterized by one or more specific resulting logic programs, e.g. [Kakas and Mancarella 1990; Sakama and Inoue 1999].

It has been argued that neither of these two methodologies could precisely express the nature of logic program based updates. On one hand, model based approaches usually provide proper semantic justification for an update procedure, they, however, may also lose significant information that is only expressible in the form of a logic program; on the other hand, syntax based approaches can achieve syntactic results for updates but may not be able to provide semantic justification for the results. Our approach developed in this paper is actually an integration of model and syntax based methodologies, which, as has been shown in previous sections, reflects both semantics and syntax features for logic program based updates.

In the rest of this section, we will focus our comparisons with three typical approaches - Eiter et al.'s approach [Eiter et al. 2002], Sakama and Inoue's approach, and Alferes, Leite, Pereira et al.'s approach [Alferes et al. 1998; Alferes and Pereira 2000]. The first and third approaches are model based, and the second is syntax based. We provide formal characterizations between our approach and these three approaches.

8.1 Relations to Eiter et al.'s Approach

Eiter et al. recently proposed an approach for handling (extended) logic program updates where a sequence of logic program updates is allowed [Eiter et al. 2002].

As our approach only deals with one-step update[12], we will restrict Eiter et al.'s formulation to the one-step update form as follows.

Let $\mathbf{P} = (\Pi_0, \Pi_1)$, where Π_0 and Π_1 are extended logic programs, be an *update sequence* and $\mathcal{A}$ a set of atoms. We say that $\mathbf{P}$ is *over* $\mathcal{A}$ iff $\mathcal{A}$ represents the set of all atoms occurring in the rules in Π_0 and Π_1. We assume a set $\mathcal{A}^*$ of atoms extending $\mathcal{A}$ by new and pairwise distinct atoms $rej(r)$ and A_i, for each rule r occurring in Π_0 or Π_1 and each atom $A \in \mathcal{A}$. Given a rule r, we use $B(r)$ to denote the body part of r and $H(r)$ denote the literal occurring in the head of r (note that $B(r)$ and $H(r)$ are different from $body(r)$ and $head(r)$ which are sets of literals occurring in the body and head of r respectively). Then Eiter *et al*'s update process is defined by the following two definitions (for simplicity, here we only consider ground extended logic programs in our investigation).

Definition 8.1. Given an update sequence $\mathbf{P} = (\Pi_0, \Pi_1)$ over a set of atoms $\mathcal{A}$, the *update program* $\mathbf{P}_\triangleleft = \Pi_0 \triangleleft \Pi_1$ over $\mathcal{A}^*$ consisting of the following items:

(1) all constraints in Π_0 and Π_1 (recall that constraint is a rule with an empty head);

(2) for each r in Π_i $(i = 0, 1)$:
$L_i \leftarrow B(r), not\ rej(r)$ if $H(r) = L$;

(3) for each $r \in \Pi_0$:
$rej(r) \leftarrow B(r), \neg L_1$ if $H(r) = L$;

(4) for each literal L occurring in $\Pi_0 \cup \Pi_1$:
$L_0 \leftarrow L_1$;
$L \leftarrow L_0$.

Intuitively, this program expresses the derivability of a literal L, beginning at Π_1 downwards to Π_0. A rule in Π_1 is always applicable where a rule in Π_0 can only be applicable if it is not refuted by a literal derived at Π_1 that is incompatible with $head(r)$. Persistence of a literal L propagates a locally derived value for L downwards to Π_0, where the local value of L is made global.

Definition 8.2. Let $\mathbf{P} = (\Pi_0, \Pi_1)$ be an update sequence over $\mathcal{A}$. Then $S \subset Lit_{\mathcal{A}}$[13] is an *update answer set* of $\mathbf{P}$ iff $S = S' \cap Lit_{\mathcal{A}}$ for some answer set S' of $\mathbf{P}_\triangleleft$. The collection of all update answer sets of $\mathbf{P}$ is denoted by $\mathcal{U}(\mathbf{P})$.

As an example, consider the update of Π_0 by Π_1, where Π_0 and Π_1 consist of the following rules respectively [Eiter et al. 2002],

Π_0:
$r_1 : sleep \leftarrow not\ tv\_on$,
$r_2 : night \leftarrow$,
$r_3 : tv\_on \leftarrow$,
$r_4 : watch\_tv \leftarrow tv\_on$;
Π_1:

[12]Readers may refer to the author's recent work for a generalization of this approach to handle epistemic logic programs and sequence updates [Zhang 2003a].
[13]$Lit_{\mathcal{A}}$ is the set of literals whose corresponding atoms occur in $\mathcal{A}$.

$r_5 : \neg tv\_on \leftarrow power\_failure,$
$r_6 : power\_failure \leftarrow.$

According to Definitions 8.1 and 8.2, it is easy to see that $\mathbf{P} = (\Pi_0, \Pi_1)$ has a unique update answer set $S = \{power\_failure, \neg tv\_on, sleep, night\}$, which is desired from our intuition.

It is quite easy to observe the difference between Eiter et al.'s approach and ours by noting that no conflict resolution is considered in Eiter et al.'s approach. For instance, updating program $\{A \leftarrow notB\}$ by $\{B \leftarrow notA\}$ will have a unique semantic result $\{B\}$ by using our approach, but two possible results $\{A\}$ and $\{B\}$ by using Eiter et al.'s approach. Also, Eiter et al.'s approach characterizes an update result through the answer set semantics only. However, apart from such superficial difference between two approaches, the fundamental issue of minimal change in updates is handled in very different ways in these two approaches. To reveal this fact, we restrict out attention to the simple fact update in which no conflict resolution is explicitly considered.

PROPOSITION 8.3. *There exists some update sequence* $\mathbf{P} = (\Pi_0, \Pi_1)$ *where each rule in* Π_0 *is of the form* $L \leftarrow$, *such that for some update answer set* S *of* $\mathbf{P}$, $S \notin Min(S', \Pi_1)$, *where* S' *is the answer set of* Π_0, *i.e.* $S' = \{L \mid L \leftarrow \textit{is in } \Pi_0\}$.

PROOF. Consider an update sequence $\mathbf{P} = (\Pi_0, \Pi_1)$ where $\Pi_0 = \{r_1 : A \leftarrow, r_2 : B \leftarrow\}$ and $\Pi_1 = \{r_3 : \neg A \leftarrow not\ A\}$. By Definition 8.2, $\mathbf{P}_\triangleleft$ consists of the following rules:

$$\begin{array}{ll}
\neg A_2 \leftarrow not\ A, not\ rej(r_3), & \\
A_1 \leftarrow not\ rej(r_1), & \\
B_1 \leftarrow not\ rej(r_2), & \\
rej(r_1) \leftarrow \neg A_2, & \\
rej(r_2) \leftarrow \neg B_2, & \\
A_1 \leftarrow A_2, & \neg A_1 \leftarrow \neg A_2, \\
B_1 \leftarrow B_2, & \neg B_1 \leftarrow \neg B_2, \\
A \leftarrow A_1, & \neg A \leftarrow \neg A_1, \\
B \leftarrow B_1, & \neg B \leftarrow \neg B_1.
\end{array}$$

It is easy to see that $\mathbf{P}_\triangleleft$ has two answer sets $\{\neg A, \neg A_1, \neg A_2, rej(r_1), B, B_1, B_2\}$ and $\{A, A_1, B, B_1, B_2\}$, from which we know that $S_1 = \{\neg A, B\}$ and $S_2 = \{A, B\}$ are the two update answer sets of $\mathbf{P}$. As $S' = \{A, B\}$ is the only answer set of Π_0, it is obvious that $S_1 \notin Min(S', \Pi_1)$. $\square$

Now we give a sufficient and necessary condition under which our approach and Eiter et al.'s coincide for simple fact updates. Given an update sequence $\mathbf{P} = (\Pi_0, \Pi_1)$, where each rule in Π_0 is of the form $L \leftarrow$, the corresponding simple fact update in our formulation is represented as $Update(S, \Pi_1)$, where $S = \{L \mid L \leftarrow \text{is in } \Pi_0\}$. For Π_0 in $\mathbf{P}$, we also define a corresponding program Π'_0 as follows:

$$\Pi'_0 = \{L \leftarrow not\overline{L} \mid L \leftarrow \text{is in } \Pi_0\}.$$

Clearly, S is consistent iff Π_0 is consistent, from which it follows that there is no pair of mutual defeasible rules in Π'_0. In the following, we will only consider consistent update sequence, that is, both Π_0 and Π_1 are consistent in $\mathbf{P} = (\Pi_0, \Pi_1)$, and also assume that the corresponding simple fact update $Update(S, \Pi_1)$ is well defined (see Definition 3.2).

Theorem 8.4. *Let* $\mathbf{P} = (\Pi_0, \Pi_1)$ *be a consistent update sequence where each rule in* Π_0 *is of the form* $L \leftarrow$, *and* $\mathcal{U}(\mathbf{P})$ *be the set of all update answer sets of* $\mathbf{P}$. *Given* $S = \{L \mid L \leftarrow \text{ is in } \Pi_0\}$ *and* $\Pi_0' = \{L \leftarrow not\overline{L} \mid L \leftarrow \text{ is in } \Pi_0\}$. *Then* $\mathcal{U}(\mathbf{P}) = Res(Update(S, \Pi_1))$ *iff one of the following conditions holds:*

(1) *there do not exist rules* r_p, r_q *such that* $r_p \in \Pi_0'$ *and* $r_q \in \Pi_1$ *and* $mutual(r_p, r_q)$;

(2) *for any rules* r_p, r_q *such that* $r_p \in \Pi_0'$ *and* $r_q \in \Pi_1$ *and* $mutual(r_p, r_q)$, *and for any* $S' \in \mathcal{U}(\mathbf{P})$, $S' \not\models pos(r_q)$.

Proof. We only prove one direction and the other direction can be proved in a similar way. Consider $\mathcal{U}(\mathbf{P}) = Res(Update(S, \Pi_1))$. In this case, each result of $Update(S, \Pi_1)$ is also an answer set in $\mathcal{U}(\mathbf{P})$. Note that from Lemma 3.3 (section 3.3), $Res(Update(S, \Pi))$ is the set of all answer sets of program $\mathcal{P} = (\Pi_0' \cup \Pi_1, \mathcal{N}, <)$, where for each rule $r \in \Pi_0'$ and rule $r' \in \Pi_1$, $\mathcal{N}(r) < \mathcal{N}(r')$. Then from Theorem 3.5 (section 3.3), we know that each answer set in $\mathcal{U}(\mathbf{P})$ represents a minimal change with respect to the answer set of Π_0, i.e. S, under the condition of satisfying Π_1. From Definitions 8.1 and 8.2, we can see that the only way to ensure such minimal change is that there is no conflict between Π_0' and Π_1 under any $S' \in \mathcal{U}(\mathbf{P})$. That is, there is no pair of mutual defeasible rules between Π_0' and Π_1, say $r_p \in \Pi_0'$ and $r_q \in \Pi_1$ such that for one answer set $S' \in \mathcal{U}(\mathbf{P})$, $S' \models pos(r_p)$, and for another answer set $S^* \in \mathcal{U}(\mathbf{P})$, $S^* \models pos(r_q)$. This implies that either no any mutual defeasible rules exists between Π_0' and Π_1; or for any pair of mutual defeasible rules $mutual(r_p, r_q)$ where $r_p \in \Pi_0'$ and $r_q \in \Pi_1$, no $S' \in \mathcal{U}(\mathbf{P})$ satisfies the relation $S' \models pos(r_q)$. Note that if $S \models pos(r_q)$, rule r_p in Π_0' will be defeated by S' and then $S' \notin Min(\mathcal{B}, \Pi_1)$. This contradicts the condition $\mathcal{U}(\mathbf{P}) = Res(Update(S, \Pi_1))$ □

Corollary 8.5. *Let* $\mathbf{P} = (\Pi_0, \Pi_1)$ *be a consistent update sequence where each rule in* Π_0 *is of the form* $L \leftarrow$, *and* $S = \{L \mid L \leftarrow \text{ is in } \Pi_0\}$. *Then* $\mathcal{U}(\mathbf{P}) \neq Res(Update(S, \Pi_1))$ *iff there exists some* $S' \in \mathcal{U}(\mathbf{P})$ *such that* $S' \notin Min(S, \Pi_1)$.

Proof. Directly from Theorem 8.4. □

8.2 Relations to Sakama and Inoue's Approach

Sakama and Inoue recently proposed an approach for the logic program update through abduction [Sakama and Inoue 1999]. In particular, they considered three types of updates: view updates, theory updates and inconsistency removal. In the view update, Sakama and Inoue distinguished a knowledge base, i.e. an extended logic program, to consist of variable and invariant knowledge. Basically, variable knowledge is viewed as the basic observation about the current state and is changeable, where invariable knowledge is viewed as constraints of the domain and hence is unchangeable. Nevertheless, view update only deals with the case of inserting or deleting literals. The theory update, on the other hand, is the general case of logic program update where the entire knowledge is variable and can be updated via another logic program. Finally, the inconsistency removal is to deal with the problem of revising an inconsistent program to be consistent. In the rest of this subsection, we will focus on the theory update which has the same setting as other logic program update approaches.

Definition 8.6. Given extended logic programs Π_0 and Π_1, Π' *accomplishes* a theory update of Π_0 by Π_1 if

(1) Π' is consistent,
(2) $\Pi_1 \subseteq \Pi' \subseteq \Pi_0 \cup \Pi_1$, and
(3) there is no any other consistent program Π'' such that $\Pi' \subset \Pi'' \subseteq \Pi_0 \cup \Pi_1$.

The following proposition shows the connection between Sakama and Inoue's theory update and abduction[14].

PROPOSITION 8.7. *[Sakama and Inoue 1999] Let* $(\Pi_0 \cup \Pi_1, \Pi_0 - \Pi_1)$ *be an abductive program. Then* Π' *accomplishes a theory update of* Π_0 *by* Π_1 *iff* $\Pi' = (\Pi_0 \cup \Pi_1) - \Pi^*$*, where* $(\emptyset, \Pi^*)$ *is a minimal anti-explanation of the observation* $G = \mathtt{F}$ *with respect to* $(\Pi_0 \cup \Pi_1, \Pi_0 - \Pi_1)$.

According to Definition 8.6, by using Sakama and Inoue's approach, the resulting program Π' can be expressed as $\Pi^* \cup \Pi_1$, where Π^* is a maximal subset of Π_0 such that $\Pi^* \cup \Pi_1$ is consistent. We use $\mathit{SI\text{-}Update}(\Pi_0, \Pi_1)$ to denote the set of all resulting programs, and $\bigcup \mathcal{S}(\mathit{SI\text{-}Update}(\Pi_0, \Pi_1))$ to denote the set of answer sets of all resulting programs.

Basic differences between Sakama and Inoue's approach and ours are quite obvious. Like Eiter et al.'s approach, Sakama and Inoue's approach does not consider any conflict resolution issue, while our approach does. Furthermore, since changes are not based on any semantic considerations, by applying Sakama and Inoue's approach, an update result may not be properly justified from a semantic viewpoint. Consider a simple example as follows. Let $\Pi_0 = \{A \leftarrow, B \leftarrow A\}$ and $\Pi_1 = \{\neg B \leftarrow A\}$. Both our approach and Eiter et al.'s approach will produce a unique result $\{A, \neg B\}$, while Sakama and Inoue's approach will generate two possible results $\{A, \neg B\}$ and $\emptyset$, which are answer sets of resulting programs $\{A \leftarrow, \neg B \leftarrow A\}$ and $\{B \leftarrow A, \neg B \leftarrow A\}$ respectively. Intuitively, we would reject the second result since there is no semantic justification on the fact that both truth values of A and B become *unknow* after the update.

Though these differences, surprisingly, Sakama and Inoue's approach coincides with our approach for simple fact updates.

THEOREM 8.8. *Let* Π_0 *and* Π_1 *be two consistent programs where each rule in* Π_0 *is of the form* $L \leftarrow$ *and* $S = \{L \mid L \leftarrow \text{ is in } \Pi_0\}$. *Suppose* $\mathit{Update}(S, \Pi_1)$ *is well defined. Then* $\bigcup \mathcal{S}(\mathit{SI\text{-}Update}(\Pi_0, \Pi_1)) = \mathit{Res}(\mathit{Update}(S, \Pi_1))$.

PROOF. From Lemma 3.3, we know that a result of $\mathit{Update}(S, \Pi_1)$ can be viewed as an answer set of program $\mathcal{P} = (\Pi^* \cup \Pi_1, \mathcal{N}, <)$, where $\Pi^* = \{L \leftarrow \mathit{not}\overline{L} \mid L \in S\}$, and for each rule $r \in \Pi^*$ and each rule $r' \in \Pi_1$, $\mathcal{N}(r) < \mathcal{N}(r')$. Let S' be an answer set of $\mathcal{P}$. Then S' is an answer set of $\mathcal{P}$'s reduct: $\Pi^* \cup \Pi'_1$, where $\Pi'_1 \subseteq \Pi_1$.

From answer set S', we can *generate* an extended logic program of the form $\Pi'_0 \cup \Pi_1$ where $\Pi'_0 \subseteq \Pi_0$ such that $L \leftarrow$ is in Π'_0 iff $L \in S'$. Clearly, each rule of

[14]Readers may refer to [Eiter et al. 1997; Sakama and Inoue 1999] for the detail of abductive logic programming.

$\Pi'_0 \cup \Pi_1$ is satisfied in S'. Furthermore, it also can be verify that S' is an answer set of $\Pi'_0 \cup \Pi_1$. Now we show that Π'_0 is a maximal subset of Π_0 to make $\Pi'_0 \cup \Pi_1$ consistent. Suppose that Π'_0 is not such a maximal subset of Π_0. Then without loss of generality, we assume that there is a rule $L \leftarrow$ from Π_0 such that $\Pi'_0 \cup \{L \leftarrow\} \cup \Pi_1$ is consistent. Firstly, $S' \cup \{L\}$ cannot be an answer set of $\Pi'_0 \cup \{L \leftarrow\} \cup \Pi_1$ because $\overline{L} \in S'$ (otherwise $L \in S'$ and this implies S' is not an answer set of $\Pi'_0 \cup \Pi_1$). Therefore, there is a rule of the form $r' : \overline{L} \leftarrow \cdots$ is in Π_1. Also, it is observed that r' and rule $r : L \leftarrow not\overline{L}$ in Π^* must be mutually defeasible, i.e. $mutual(r, r')$, otherwise $\Pi'_0 \cup \{L \leftarrow\} \cup \Pi_1$ cannot be consistent. Since $\mathcal{N}(r) < \mathcal{N}(r')$ in $\mathcal{P}$, from the construction of $\mathcal{P}$'s reduct, we know that rule r' should be defeated during the generation of the reduct $\Pi^* \cup \Pi'_1$. This follows that $L \in S'$ and $\overline{L} \notin S'$. But this contradicts the fact that $L \notin S'$ and $\overline{L} \in S'$. So $\Pi'_0 \cup \{L \leftarrow\} \cup \Pi_1$ cannot be consistent. Thus, we prove $Res(Update(S, \Pi_1)) \subseteq \bigcup \mathcal{S}(SI\text{-}Update(\Pi_0, \Pi_1))$.

Now we prove $\bigcup \mathcal{S}(SI\text{-}Update(\Pi_0, \Pi_1)) \subseteq Res(Update(S, \Pi_1))$. Suppose $\Pi'_0 \cup \Pi_1$ is a resulting program after the update of Π_0 by Π_1, i.e. $\Pi'_0 \cup \Pi_1 \in SI\text{-}Update(\Pi_0, \Pi_1)$. Let S' be an answer set of $\Pi'_0 \cup \Pi_1$. We need to show that S' is also an answer set of program $\mathcal{P} = (\Pi^* \cup \Pi_1, \mathcal{N}, <)$, where $\Pi^* = \{L \leftarrow not\overline{L} \mid L \in S\}$, and for each rule $r \in \Pi^*$ and each rule $r' \in \Pi_1$, $\mathcal{N}(r) < \mathcal{N}(r')$.

Let $\Pi^* \cup \Pi'_1$ be a reduct of $\mathcal{P}$. We will show that S' is an answer set of $\Pi^* \cup \Pi'_1$. According to Definition 2.2, suppose the reduct $\Pi^* \cup \Pi'_1$ of $\mathcal{P}$ is generated from the following reduct chain:

$$\begin{aligned}
&\Pi^{(1)} = \Pi^* \cup \Pi_1,\\
&\cdots,\\
&\Pi^{(i+1)} = \Pi^{(i)} - \{r_p, \cdots, r_q \mid \text{there exists a rule } r \in \Pi^{(i)} \text{ such that}\\
&\qquad\qquad\qquad\qquad \mathcal{N}(r) < \mathcal{N}(r_p, \cdots, r_q) \text{ and } r_p, \cdots, r_q\\
&\qquad\qquad\qquad\qquad \text{are defeated by } \Pi^{(i)} - \{r_p, \cdots, r_q\}\},\\
&\cdots
\end{aligned}$$

From Theorem 1 in [Zhang 2003b], we know that a set of ground literals is an answer set of $\mathcal{P}$ iff it is an answer set of each $\Pi^{(i)}$ $(i = 1, 2, \cdots)$ in a reduct chain of $\mathcal{P}$. Now we prove that S' is answer set of each $\Pi^{(i)}$ by induction on i.

Consider $i = 1$. In this case, we show that S' is an answer set of $\Pi^* \cup \Pi_1$. Let $(\Pi^* \cup \Pi_1)^{S'}$ be $\Pi^* \cup \Pi_1$'s Gelfond -Lifschitz transformation with respect to S'. Since S' is an answer set of $\Pi'_0 \cup \Pi_1$, where $\Pi'_0 \subseteq \Pi_0$ is a maximal consistent subset of Π_0 with Π_1, this follows that each rule in Π_1 and each rule of Π'_0 are satisfied in S'. Note that rules from Π'_0 are of the form $L \leftarrow$, where rules from Π^* are of the form $L \leftarrow not\overline{L}$. Therefore, for each rule $r : L \leftarrow$ in Π_0 but not in Π'_0, it must be the case that there is a rule $\overline{L} \leftarrow \cdots$ in Π_1 and $\overline{L} \in S'$ (note that Π'_0 is the maximal subset of Π_0 with Π_1). In this case, the rule in Π^* $r' : L \leftarrow not\overline{L}$ - that corresponds to rule $r : L \leftarrow$ in Π'_0, is defeated by S'. So we conclude that each rule from $(\Pi^* \cup \Pi_1)^{S'}$ is satisfied in S'. Then we need to show that S' is the smallest such set. Suppose S' is *not* the smallest set. In this case, we assume that $S' - \{L^*\}$ still satisfies each rule of $(\Pi^* \cup \Pi_1)^{S'}$. Since S' is an answer set of $\Pi'_0 \cup \Pi_1$ and $L^* \in S'$, there must be some rule of the form $r : L^* \leftarrow \cdots$ in $\Pi'_0 \cup \Pi_1$. If r is in Π'_0, then r is of the form $L^* \leftarrow$, and accordingly, there is a rule $r' : L^* \leftarrow not\overline{L^*}$ in Π^*. It is easy to observe that the only way that the answer set of $(\Pi^* \cup \Pi_1)^{S'}$ does not contain literal L^* is that there must be a rule of the form $\overline{L^*} \leftarrow \cdots$ in Π_1

which makes rule r' be defeated by S'. However, this implies that literal $\overline{L^*}$ should be in S'. This contradicts the fact that S' is a consistent set. On the other hand, if r is in Π_1, then from the fact that S' is an answer set of $\Pi'_0 \cup \Pi_1$ and $L^* \in S'$, we have $S' \models pos(r)$. However, this implies that $S' - \{L^*\}$ cannot satisfy the rule $L^* \leftarrow pos(r)$ in $(\Pi^* \cup \Pi_1)^{S'}$ (the corresponding rule of r after Gelfond-Lifschitz transformation with respect to S'. That is, $S' - \{L^*\}$ cannot satisfy all rules in $(\Pi^* \cup \Pi_1)^{S'}$. So S' must be the smallest set that satisfies each rule of $(\Pi^* \cup \Pi_1)^{S'}$.

Now assume that S' is an answer set for all $\Pi^{(i)}$ $(i = 1, \cdots, k)$. We show S' is also an answer set of $\Pi^{(k+1)}$ Firstly, since $\Pi^{(k+1)} \subseteq \Pi^* \cup \Pi_1$, and $\Pi^{(k+1)S'} \subseteq (\Pi^* \cup \Pi_1)^{S'}$, it follows that each rule in $\Pi^{(k+1)S'}$ is satisfied in S'.

Then we need to show that S' is the smallest set that satisfies $\Pi^{(k+1)S'}$. Again, suppose that S' is not the smallest set of literals that satisfies program $\Pi^{(k+1)S'}$. Without loss of generality, we assume that $S' - \{L'\}$ still satisfies all rules of $\Pi^{(k+1)S'}$. However, since S' is an answer set of $\Pi^{(k)}$, there must be a rule of the form $r : L' \leftarrow \cdots$ in $\Pi^{(k)}$, where $S' \models pos(r)$. On the other hand, because $S' - \{L'\}$ satisfies all rules in $\Pi^{(k+1)S'}$, it implies that rule $r : L' \leftarrow \cdots$ is defeated during the generation of $\Pi^{(k+1)}$. That is, $r \in \{r_p, \cdots, r_q\}$, and r is defeated from each answer set of $\Pi^{(k+1)}$. So we can rewrite r as the form: $r : L' \leftarrow \cdots, notL^*$. On the other hand, a rule of the form $r' : L^* \leftarrow \cdots$ should be in $\Pi^{(k)}$ and *not* triggered so that L' is derived in S'. In $\Pi^{(k+1)}$, as r is defeated by $\Pi^{(k+1)}$, L^* is in each answer set of $\Pi^{(k+1)}$. This follows that rules r and r' are mutually defeasible, i.e. $mutual(r, r')$. To simplify our discussion, we may assume r' has the form: $L^* \leftarrow L_1, \cdots, L_h, notL'$ and all $L_1, \cdots, L_h$ are in S' (this assumption allows our to only consider the non-trivial situation that r' is defeated from $\Pi^{(k)}$ because of r. Otherwise, r must be defeated by other rule(s)).

Note that $L' \in S'$ and $L^* \notin S'$. However, as we show earlier, since S' satisfies each rule of $\Pi^{(k+1)}$ where $r : L' \leftarrow \cdots notL^*$ has been removed, S' should satisfy the transformed form of r': $L^* \leftarrow L_1, \cdots, L_h$. Since $L_1, \cdots, L_h$ are in S', we have $L^* \in S'$. But this is a contradiction from our previous argument that $L^* \notin S'$. So S' must be an answer set of $\Pi^{(k+1)}$. $\Box$

8.3 Relations to Dynamic Logic Programming (DLP)

Dynamic logic programming (DLP), proposed by Alferes et al. [Alferes et al. 1998], is one of the earliest efforts to generalize Marek and Truszczyński's approach of revision programs, in which a knowledge base is represented as a logic program, and it can be updated iteratively by a sequence of logic programs. DLP is also a model based approach. In the framework of DLP, the concept of *generalized logic programs* is developed where negation as failure is allowed to occur in the head of a rule, and accordingly, the stable model definition is also extended to capture the semantics of generalized logic programs.

Formally, given a set $\mathcal{K}$ of propositional atoms, the *propositional language $\mathcal{L}_\mathcal{K}$ of generalized logic programs* contains propositional atoms from the set $\{A \mid A \in \mathcal{K}\} \cup \{notA \mid A \in \mathcal{K}\}$, where A is called *objective atom* and *not* A is called *default atom*. Therefore, the *interpretation* of $\mathcal{L}_\mathcal{K}$ will contain two types of atoms: objective and default. By given an interpretation M, let $M^+ = \{A \mid A \in M\}$ and $M^- = \{not\ A \mid not\ A \in M\}$. Then a *generalized logic program* in $\mathcal{L}_\mathcal{K}$ is a set of rules of the following form:

$L \leftarrow L_1, \cdots, L_n,$

where each L and L_i are objective or default atoms of $\mathcal{L}_{\mathcal{K}}$.

Definition 8.9. [Alferes et al. 1998] Given a generalized logic program Π in $\mathcal{L}_{\mathcal{K}}$, an interpretation M of $\mathcal{L}_{\mathcal{K}}$ is a *stable model* of Π if M is the least model of the Horn theory $\Pi \cup M^{-}$[15].

Consider the following program:

$A \leftarrow not\ B$, $C \leftarrow B$,
$E \leftarrow not\ D$,
$not\ D \leftarrow A, not\ C$,
$D \leftarrow not\ E$.

According to Definition 8.9, this program has one stable model $\{A, not\ B, not\ C, not\ D, E\}$.

Now we can introduce the approach of DLP as follows. Given two generalized logic programs Π_0 and Π_1 in the language $\mathcal{L}_{\mathcal{K}}$, we define

$$\overline{\mathcal{K}} = \mathcal{K} \cup \{\mathcal{A}^- \mid A \in \mathcal{K}\} \cup \{A_{\Pi_0}, A^-_{\Pi_0} \mid A \in \mathcal{K}\} \cup \{A_{\Pi_1}, A^-_{\Pi_1} \mid A \in \mathcal{K}\}.$$

Then the update of Π_0 by Π_1 is a generalized logic program, denoted as $\Pi_0 \oplus \Pi_1$, consists of the following rules:

(1) **Rewritten rules for** Π_0 **and** Π_1: for each rule $r \in \Pi_i$ $(i = 0, 1)$, with $head(r) \leftarrow B_1, \cdots, B_m, not\ C_1, \cdots, not\ C_n$:
$A_{\Pi_i} \leftarrow B_1, \cdots, B_m, C_1^-, \cdots, C_n^-$ if $head(r) = \{A\}$;
$A^-_{\Pi_i} \leftarrow B_1, \cdots, B_m, C_1^-, \cdots, C_n^-$ if $head(r) = \{not\ A\}$;

(2) **Update rules**: for all objective atoms A occurring in Π_0 and Π_1:
$A \leftarrow A_{\Pi_1}$;
$A^- \leftarrow A^-_{\Pi_1}$;

(3) **Inertia rules**: for all objective atoms A occurring in Π_0 and Π_1:
$A \leftarrow A_{\Pi_0}, not\ A^-_{\Pi_1}$;
$A^- \leftarrow A^-_{\Pi_0}, not\ A_{\Pi_1}$;

(4) **Default rules**: for all objective atoms A occurring in Π_0 and Π_1:
$A^- \leftarrow not\ A_{\Pi_0}, not\ A_{\Pi_1}$;
$not\ A \leftarrow A^-$.

Finally, the semantics of the update of Π_0 by Π_1 is characterized by the stable model of $\Pi_0 \oplus \Pi_1$. If S is a stable model of $\Pi_0 \oplus \Pi_1$, we also define $S' = S \cap (\mathcal{K} \cup not\ \mathcal{K})$[16] to be a *dynamic stable model* of the *update sequence* $\mathbf{P} = (\Pi_0, \Pi_1)$. Clearly, S' contains all objective and default atoms that should be true after the update. This approach can be easily generalized to a sequence of updates as shown in [Alferes et al. 1998].

[15]Note that here we view each default atom *not* A as a new atom in the Horn theory.
[16]Here *not* $\mathcal{K}$ denotes the set $\{not\ A \mid A \in \mathcal{K}\}$.

8.3.1 *Comparison with the Simple Fact Update.* As before, we first consider the comparison between our approach and DLP in the case of simple fact updates.

Definition 8.10. Given a simple fact update specification $Update(\mathcal{B}, \Pi)$ as defined in Definition 3.2, we construct an update sequence $\mathbf{P} = (\Pi_0, \Pi_1)$ in DLP as follows:

(i) For each propositional atom A occurring in $Update(\mathcal{B}, \Pi)$, we introduce a new atom A^N, which is used to replace the classical negation of A;

(ii) program Π_0 consists of rules as follows: for each $L \in \mathcal{B}$, (1) if L is A, then $A \leftarrow$ is in Π_0, and (2) if L is $\neg A$, then $A^N \leftarrow$ is in Π_0;

(iii) for each rule r in Π, forming a new rule r' by replacing every classical negation atom $\neg A$ occurring in r with A^N, and include r' into Π_1;

(iv) For each propositional atom A occurring in $Update(\mathcal{B}, \Pi)$, Π_1 also contains rules of the form: $not\ A \leftarrow A^N$, and $not\ A^N \leftarrow A$.

Proposition 8.11. *Given a simple fact update specification $Update(\mathcal{B}, \Pi)$ and its resulting knowledge base $\mathcal{B}'$. Let $\mathbf{P} = (\Pi_0, \Pi_1)$ be the corresponding DLP update sequence as defined in Definition 8.10. Then a literal $L \in \mathcal{B}'$ iff there is a dynamic stable model S of $\mathbf{P}$ such that if L is A, then $A \in S$, if L is $\neg A$ then A^N is in S.*

Proposition 8.11 simply states that our approach coincides with DLP for the simple fact update. The proof of Proposition 8.11 is based on an investigation of one-to-one relationship between the answer set of prioritized logic program $Update(\mathcal{B}, \Pi)$ and the stable model of generalized logic program $\Pi_0 \oplus \Pi_1$. As the proof is rather tedious, here we only briefly address the intuition behind Proposition 8.11.

By observing $Update(\mathcal{B}, \Pi)$, it is clear that the key factor of our simple fact update approach is to solve the possible conflict between the inertia rules and update rules when both are defeasible. To achieve the minimal change criterion during the update, we specify that inertia rules have higher priorities than update rules (see Definition 3.1 in section 3.2). Therefore, when conflicts occur between these two types of rules, inertia rules will override the corresponding update rules. In DLP, this idea is implemented by transforming update rules into a form of

$$A_U \leftarrow B_1, \cdots, B_m, C_1^-, \cdots, C_n^- \text{ or}$$
$$A_U^- \leftarrow B_1, \cdots, B_m, C_1^-, \cdots, C_n^-,$$

where the negation as failure atoms $\{not\ C_1, \cdots, not\ C_n\}$ in the original update rule's body are replaced by atoms $\{C_1^-, \cdots, C_n^-\}$. In this way, the inertia rules of the form

$$A \leftarrow A_{\Pi_0}, not\ A_{\Pi_1}^-,$$
$$A^- \leftarrow A_{\Pi_0}^-, not\ A_{\Pi_1}$$

will no longer conflict with the rewritten update rules. Furthermore, since these inertia rules are defeasible, their heads will be derived under a higher priority than those heads of rewritten rules.

From Theorem 3.5 and Proposition 8.11, it also implies that DLP satisfies the minimal change for the simple fact update. It is quite interesting to note that without employing preferences, DLP also solves the main difficulty of the simple fact update that most model based approaches have. In this sense, it concludes that the information conflict in update problems usually can be handled by both prioritized and non-prioritized logic programs. But we should indicate that in prioritized logic programming approach, such conflict is solved in an explicit manner while in DLP it is implicitly handled by using transforming update rules.

8.3.2 *Comparison with the Program Update.* Although our approach coincides with DLP for the simple fact update, the difference between these two approaches for the program update are quite obvious - our approach considers the conflict resolution while DLP does not. However, probably the most important difference is the way of handling contradictions (we will discuss the issue of conflict resolution in next subsection). For example, given two programs $\Pi_0 = \{A \leftarrow B, \neg A \leftarrow B\}$ and $\Pi_1 = \{B \leftarrow\}$, consider the update of Π_0 by Π_1. In our approach, we get two possible results $\{A \leftarrow B,\ B \leftarrow\}$ and $\{\neg A \leftarrow B,\ B \leftarrow\}$, while DLP will conclude an inconsistent result.

For the above example, people may argue that our approach gives less intuitive result as it seems to undertake both update and revision. We note that not only DLP but also other model based update approaches i.e. Eiter et al.'s approach [Eiter et al. 2002], have adapted similar strategies for contradiction removal, which, without explicitly considering the syntactic forms of the update result, may be inevitable to result in inconsistency in some situations as illustrated in the above example. This, as argued by some researchers, may be more intuitive from a model theoretic viewpoint.

8.3.3 *Preference and Conflict Resolution.* One of the other major difference between our update approach and others is that we use prioritized logic programming to specify an update procedure while conflict between the original program and the update program is solvable by using preferences. Although it is arguable whether it is always necessary to solve the conflict between the original program and update program, it should be noted that in our approach, preference is not just used for solving this type of conflict, instead it plays a key role in achieving a minimal change in the stage of contradiction elimination. As we will address in next section, our approach can be easily extended to deal with domain-dependent conflict resolution in which only certain type of conflict may be considered in deriving an update result.

DLP has been extended by Alferes and Pereira to combine the reasoning about preference and updating into a unified framework [Alferes and Pereira 2000]. In particular, Alferes and Pereira proposed a formulation which combines Brewka and Eiter's prioritized logic programming [Brewka and Eiter 1999] and DLP such that the *Dynamic Prioritized Programming* (DPP) can deal with prioritized logic program update which includes to update the preference relation itself.

The detailed comparison between our prioritized logic programs and Brewka and Eiter's is beyond the scope of this paper. However, it is worth to highlighting the major difference between these two prioritized logic program formalisms. In Brewka and Eiter's preferred answer set semantics, the preference is *not defeasible*, while in our answer set semantics, the preference is treated defeasible. By viewing the preference defeasible, in our formulation, every prioritized logic program has an answer set if and only if the underlying extended logic program has an answer set. This is not always true in Brewka and Eiter's preferred answer set semantics. Such semantic difference implies the major differentiation between our approach and the DPP for program updates with preference.

Let us consider a scenario about employment and pension. If no evidence showing that someone is receiving pension, then it is believed that the person is currently employed. Now suppose we want to add a new rule into the domain saying that if no evidence indicating a person is employed, then it is believed that this person is unemployed. We represent these two statements as the following rules:

$r_1 : \mathit{Employed} \leftarrow \mathit{not}\ \mathit{Receiving\text{-}pension}$,
$r_2 : \neg \mathit{Employed} \leftarrow \mathit{not}\ \mathit{Employed}$,

Let $\Pi_0 = \{r_1\}$ and $\Pi_1 = \{r_2\}$. Then in our update framework, the above problem is represented as the update of Π_0 by Π_1, i.e. $P\text{-}\mathit{Update}(\Pi_0, \Pi_1)$ (note that the preference relation $r_2 < r_1$ is embedded in the specification of $P\text{-}\mathit{Update}(\Pi_0, \Pi_1)$. See Definition 5.2 in section 5.2). The resulting program is simply $\Pi_0 \cup \Pi_1$, which has a unique answer set $\{\mathit{Employed}\}$. This result seems reasonable from our intuition.

It is not difficult to show that we will get the same result by using DLP to specify this update problem without considering the preference relation $r_2 < r_1$. However, the situation changes when we use DPP to specify it. Consider the dynamic prioritized program $(\Pi_0, R_0) \oplus (\Pi_1, R_1)$, where $R_0 = \emptyset$, and $R_1 = \{r_2 < r_1\}$[17]. This program has no preferred stable model at state 1. That means, updating Π_0 by Π_1, by taking the rule of Π_1 as a higher preference, will not get any result. Such solution seems not quite intuitive. Because r_1 and r_2 actually do not conflict with each other, and in this case we would expect the preference relation $r_2 < r_1$ does not play a role in the program evaluation. Nevertheless, this property was not considered in Brewka and Eiter's preferred answer set semantics, and therefore in DPP neither.

9. CONCLUSION

In this paper, we developed an approach for logic program based updates. We also investigated various desirable properties for logic program based updates and their computational complexity.

We observed that information conflict is a complex and important issue in logic program based updates and was not considerably studied by existing approaches. To deal with this problem and its related issues, we used a prioritized logic programming as a basis to specify our update problems. Due to the nature of logic programs, we believe that both declarative semantics and syntax sensitivity must be carefully taken into account in logic program based updates. Previous approaches only focused on one side or another and the link between them was missing. For the first time, our update formulation was developed by integrating these two aspects into a unified framework and presented the most distinguishing feature between

[17]Readers are referred to [Alferes and Pereira 2000] for the detailed definition of dynamic prioritized programs and their associated semantics. Also, for a reason of simplicity, here we omit the process of converting an extended logic program to a generalized logic program.

our approach and other relevant approaches. We also analyzed the computational complexity of our approach in detail and provided nontrivial solutions to simplify various update evaluation procedures under certain conditions.

Our work can be further extended in several directions. First, our update formulation can be enhanced to handle update sequences where more than two logic programs are involved. Although some other approaches can deal with update sequences, like Eiter et al.'s approach and DLP [Eiter et al. 2002; Alferes et al. 1998], their approaches do not meet the requirement of minimality of change for update sequences [Eiter et al. 2002][18]. By omitting the issue of conflict solving, the author recently extended the approach presented in this paper to handle epistemic logic program updates in which update sequences are considered and the requirement of the minimality of change for update sequences is satisfied [Zhang 2003a]. It would be an interesting topic to consider the conflict solving problem in the extent of sequence updates on epistemic logic programs.

The other important issue that was not addressed in this paper is *dynamic contradiction elimination and conflict solving* in logic program based updates. In our current approach, contradictory rules are removed from the final resulting program when a program update takes place. It appears that a more flexible way to deal with contradictory information is to *weaken* the contradictory rules when new added update rules are effective. Consider two programs as follows:

Π_0:
$r_1 : A \leftarrow,$
$r_2 : B \leftarrow A, notC,$
Π_1:
$r_3 : \neg B \leftarrow, A, notD.$

Now updating Π_0 with Π_1, by using our approach, we will have a resulting program $\Pi_3 = \{r_1, r_3\}$. Suppose we further update Π_3 with $\Pi_4 = \{r_4 : D \leftarrow\}$. Then the final result will be $\Pi_5 = \{r_1, r_3, r_4\}$, from which we can derive A and D. However, we may have a more flexible way to perform such updates. For instance, when we update Π_0 with Π_1, since rule r_2 contradicts rule r_3 in Π_1, instead of eliminating r_2 from Π_3, we can weaken it:

$$r'_2 : B \leftarrow A, notC, not\neg B,$$

and include r'_2 into $\Pi'_3 = \{r_1, r'_2, r_3\}$. Then after updating Π'_3 with Π_4, we will have $\Pi'_5 = \{r_1, r'_2, r_3, r_4\}$. Thus, from Π'_5, we can derive A, B and D. This result seems plausible because r_3 now is not effective and r'_2 provides justification for deriving B. It would be an interesting issue how this dynamic contradiction elimination can be formalized in program updates[19].

On the other hand, in terms of conflict solving, in our current approach *domain-independent* preference was specified through a prioritized logic program to solve

[18]Intuitively, this means that to perform an update sequence, it is always preferred to remove those conflict (or/and) contradictory rules appeared earlier in the sequence.

[19]It should be also noted that without discarding any contradictory rules, the final program will be getting bigger and bigger. From an implementation point of view, a proper boundary about this should be carefully defined.

different types of information conflict. However, in many applications, *domain-dependent* preference - which may vary through different instances [Brewka 1996; Zhang and Foo 1997], could substantially influence the way of an update performance. In this case, a mechanism of handling *dynamic conflict solving* is essential. This will be a major challenge in the further research on logic program based updates.

Finally, *mutual updates* on logic programs is an important topic for reasoning about logic program based multiagent systems [de Vos and Vermeir 2000]. Consider a multiagent system where each agent's knowledge base is represented as an extended logic program. To achieve an agreement among all agents, each agent receives other agents' knowledge and tries to maximally maintain her own knowledge if contradictions or/and conflicts exist between this agent's knowledge and others'. This is so-called mutual update - each agent uses her knowledge base to update other agents' knowledge bases. The process continues until all agents reach a common knowledge base. To formalize this task, conflict solving is one of the most essential issues we should consider. Furthermore, since an agent's knowledge base may constantly change before a common knowledge base is reached, maintaining an explicitly syntactic representation of the agent's update result at each stage is important in order to avoid too much information loose from the entire update procedure. We expect our work presented in this paper to provide a foundational basis for this study.

ACKNOWLEDGMENT

The author specially thanks Michael Gelfond, Gerhard Brewka, Norman Foo, Thomas Eiter and Joao A. Leite for useful discussions on many related issues of logic programming and updates. The author also thanks three anonymous referees for valuable comments on an early version of this paper.

REFERENCES

Alferes, J., Leite, J., Pereira, L., Przymusinska, H., and Przymusinski, T. 1998. Dynamic logic programming. In *Proceedings of International Conference on Knowledge Representation and Reasoning (KR-98)*. 98–109.

Alferes, J. and Pereira, L. 2000. Updates plus preference. In *Proceedings of JELIA2000*.

Alferes, J., Pereira, L., Przymusinska, H., and Przymusinski, T. 2002. Lups - a language for updating logic programs. *Artificial Intelligence 138*, 87–116.

Alves, M., Laurent, D., and Spyratos, N. 1995. Update rules in datalog programs. In *Proceedings of 1995 Workshop on Logic Programming and Nonmonotonic Reasoning (LPNMR'95)*. 71–84.

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

Ben-Eliyahu, R. and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence 12*, 53–87.

Boutilier, C. 1996. Abduction to plausible causes: An event-based model of belief update. *Artificial Intelligence 83*, 143–166.

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

Brewka, G. and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence 109*, 297–356.

Crescini, V. and Zhang, Y. 2004. Policyupdater: A system for dynamic access control. Tech. rep., University of Western Sydney.

de Vos, M. and Vermeir, D. 2000. A logic for modeling decision making with dynamic preferences. In *Proceedings of JELIA 2000*. 391–406.

Dekhtyar, M., Dikovsky, A., and Spyratos, N. 1998. On logically justified updates. In *Proceedings of 1998 Joint International Conference and Symposium on Logic Programming (JICSLP-98)*.

Delgrande, J., Schaub, and Tompits, H. 2000. Logic programs with compiled preferences. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI2000)*. 392–398.

Eiter, T., Fink, M., Sabbatini, G., and Tompits, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming 2*, 711–767.

Eiter, T., Gottlob, G., and Leone, N. 1997. Abduction from logic programs: semantics and complexity. *Theoretical Computer Science 189*, 129–177.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. 1997. A deductive system for nonmonotonic reasoning. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*. 363–374.

Engelfreit, J. 1998. Monotonicity and persistence in preferential logics. *Journal of Artificial Intelligence Research 8*, 1–12.

Garey, M. and Johnson, D. 1979. *Computers and Intractability.*

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

Guessoum, A. and Lloyd, J. 1991. Updating knowledge bases ii. *New Generation Computing 10*, 73–100.

Herzig, A. and Rifi, O. 1999. Propositional belief base update and minimal change. *Artificial Intelligence 115*, 107–138.

Kakas, A. and Mancarella, P. 1990. Database updates through abduction. In *Proceedings of the 16th VLDB Conference*. 650–661.

Katsuno, H. and Mendelzon, A. 1991. On the difference between updating a knowledge database and revising it. In *Proceedings of Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*. 387–394.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *Proceedings of Eleventh International Conference on Logic Programming*. 23–37.

Marek, V. and Truszczyński, M. Autoepistemic logic. *Journal of the Association of Computing Machinery 38*, 588–619.

Marek, V. and Truszczyński, M. 1994. Revision program. In *Proceedings of JELIA'94*.

Marek, V. and Truszczyński, M. 1998. Revision programming. *Theoretical Computer Science*, 241–277.

Nemela, I. and Simons, P. 1996. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP'96)*. 289–303.

Papadimitriou, C. 1994. *Computational Complexity.*

Przymusinski, T. and Turner, H. 1997. Update by means of inference rules. *Journal of Logic Programming 30*, 125–143.

Rao, P., Sagonas, K., Swift, T., Warren, D., and Freire, J. 1997. Xsb: A system for efficiently computing wfs. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*. 2–17.

Sakama, C. and Inoue, K. 1999. Updating extended logic programs through abduction. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*. 147–161.

Schaub, T. and Wang, K. 2001. A comparative study of logic programs with preference. In *roceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI01)*. 597–602.

Winslett, M. 1988. Reasoning about action using a possible models approach. In *Proceedings of AAAI-88*. 89–93.

Zhang, Y. 1999. Monotonicity in rule based update. In *Proceedings of the 1999 International Conference on Logic Programming (ICLP'99)*. 471–485.

Zhang, Y. 2001. The complexity of logic program updates. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence (AI2001)*. 630–643.

Zhang, Y. 2003a. Minimal change and maximal coherence for epistemic logic program updates. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*. 112–117.

Zhang, Y. 2003b. Two results for prioritized logic programming. *Theory and Practice of Logic Programming 3(2)*, 223–242.

Zhang, Y. and Foo, N. 1997. Answer sets for prioritized logic programs. In *Proceedings of the 1997 International Logic Programming Symposium (ILPS'97)*. 69–83.

Zhang, Y. and Foo, N. 1998. Updating logic programs. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98)*. 403–407.

Zhang, Y. and Foo, N. 2001. Towards generalized rule-based updates. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*. 82–88.

Zhang, Y., Wu, C.-M., and Bai, Y. 2001. Implementing prioritized logic programming. *AI Communications 14(4)*, 183–196.

Received September 2003; revised March 2004; accepted May 2004