A Unified Framework for Representing Logic Program Updates

Yan Zhang
School of Computing & Information Technology
University of Western Sydney
NSW 1797, Australia
E-mail: yan@cit.uws.edu.au

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

Abstract

As a promising formulation to represent and reason about agents' dynamic behavious, logic program updates have been considerably studied recently. While similarities and differences between various approaches were discussed and evaluated by researchers, there is a lack of method to represent different logic program update approaches under a common framework. In this paper, we continue our study on a general framework for logic program conflict solving based on notions of strong and weak forgettings (Zhang, Foo, & Wang 2005). We show that all major logic program update approaches can be transformed into our framework, under which each update approach becomes a specific conflict solving case with certain constraints. We also investigate related computational properties for these transformations.

Introduction

As a promising formulation to represent and reason about agents' dynamic behavious, logic program updates have been considerably studied recently. Basically, there are three major approaches to deal with logic program updates: *model based*, *syntax based*, and *integrated* approaches. In model based approaches, atoms/literals occurring in a logic program are assumed to be persistent by default, unless there are explicit reasons to cause them to change. Consequently, the result of an update is usually characterized in terms of stable models/answer sets of the underlying update program. Eiter *et al*'s causal rejection based update (Eiter *et al.* 2002) and Alferes, Leite *et al*'s dynamic logic program (DLP) update (Alferes *et al.* 1998; Leite 2003) are typical model based update approaches. The syntax based approach, such as Sakama and Inoue's approach (Sakama & Inoue 1999), aims to generate an explicit syntactic representation of an update result, in which conflicts are solved under a consistency criterion for a maximal syntax coherence. The main problem with this approach is that it may also generate some unwanted update solutions. Zhang and Foo's integrated update approach (Zhang & Foo 1998), on the other hand, combines both semantic and syntactic features of the above two types of approaches. This approach first applies inertia and update rules to solve conflicts between *new* and *old* atoms/literal (as in model based approaches), and then employs a principle of maximal syntax coherence (as in Sakama-Inoue's approach) to extract an explicit resulting program.

While similarities and differences between Eiter *et al*'s causal rejection based approach and DLP update approach can be formally characterized, e.g. (Eiter *et al.* 2002), it is believed that comparing different types of update approaches at some formal level are generally difficult (discussions on this topic are referred to (Eiter *et al.* 2002; Leite 2003; Zhang 2005)).

In our paper (Zhang, Foo, & Wang 2005), we developed a general framework for logic program conflict solving based on notions of strong and weak forgettings in logic programs (Zhang, Foo, & Wang 2005), that can be viewed as an analogous notion of forgetting in propositional theories (Lang, Liberatore, & Marquis 2003; Lin 2001). In this paper, we continue our study on this framework and further show that all major logic program update approaches can be transformed into our framework, under which each update approach becomes a specific conflict solving case with certain constraints.

The rest of the paper is organized as follows. Sections 2 and 3 present an overview on strong and weak forgettings in logic programs and a general framework for conflict solving named *logic program contexts*. Section 4 represents transformations from different logic program update approaches into the framework of logic program contexts. Section 5 provides some related complexity results about these transformations. Finally, section 6 concludes the paper with some discussions.

Strong and Weak Forgettings in Logic Programs

We consider finite propositional normal logic programs in which each rule has the form:

$$a \leftarrow b_1, \cdots, b_m, \texttt{not}\, c_1, \cdots, \texttt{not}\, c_n \qquad (1)$$

where a is either a propositional atom or empty, and $b_1, \cdots, b_m$, $c_1, \cdots, c_n$ are propositional atoms. When a is empty, rule (1) is called a *constraint*. Given a rule r of the form (1), we denote $head(r) = \{a\}$, $pos(r) = \{b_1, \cdots, b_m\}$ and $neg(r) = \{c_1, \cdots, c_n\}$, and therefore, rule (1) may be represented as the form:

$$head(r) \leftarrow pos(r), \texttt{not}\, neg(r). \qquad (2)$$

Sometimes, (2) is also simply denoted by $head(r) \leftarrow body(r)$. We assume that readers are familiar with the stable model semantics of normal logic programs. A program Π is called *consistent* if it has a stable model. An atom a *is entailed* from Π, denoted as $\Pi \models a$ if a is in every stable model of Π.

Given a program Π, a rule $r \in \Pi$ is called *redundant* if r is of one of the following forms: (1) $head(r) \neq \emptyset$ and $head(r) \subseteq pos(r)$, or (2) $pos(r) \cap neg(r) \neq \emptyset$. It is easy to show that removing redundant rules from a program will not affect the stable model of this program.

Definition 1 *(***Program reduction***) Let Π be a program and p an atom. We define the* reduction of Π with respect to p*, denoted as $Reduct(\Pi, \{p\})$, to be a program obtained from Π by (1) for each rule r with $head(r) = p$ and each rule r' with $p \in pos(r')$, replacing r' with a new rule $r'' : head(r') \leftarrow (pos(r) \cup pos(r') - \{p\}), \mathtt{not}(neg(r) \cup neg(r'))$; (2) if there is such rule r' in Π and has been replaced by r'' in (1), then removing rule r from the remaining program. Let P be a set of atoms. Then the* reduction of Π with respect to P *is inductively defined as follows:*

$Reduct(\Pi, \emptyset) = \Pi$,
$Reduct(\Pi, P \cup \{p\}) = Reduct(Reduct(\Pi, \{p\}), P)$.

Note that in our program reduction definition, step (1) is the same as logic program *unfolding* (Brass & Dix 1999). While unfolding is to eliminate positive body occurrences of an atom in a logic program, the reduction, on other hand, is further to remove those rules with heads of this atom.

Example 1 Let $\Pi_1 = \{a \leftarrow \mathtt{not}b, a \leftarrow d, c \leftarrow a, \mathtt{not}e\}$, $\Pi_2 = \{a \leftarrow c, \mathtt{not}b, c \leftarrow \mathtt{not}d\}$, and $\Pi_3 = \{a \leftarrow b, b \leftarrow \mathtt{not}d, c \leftarrow a, \mathtt{not}e\}$. Then $Reduct(\Pi_1, \{a\}) = \{c \leftarrow \mathtt{not}b, \mathtt{not}e, c \leftarrow d, \mathtt{not}e\}$, $Reduct(\Pi_2, \{a\}) = \Pi_2$, and $Reduct(\Pi_3, \{a, b\}) = \{c \leftarrow \mathtt{not}d, \mathtt{not}e\}$. □

Definition 2 *(***Strong forgetting***) Let Π be a logic program, and p an atom. We define a program to be the result of* strongly forgetting *p in Π, denoted as $SForgetLP(\Pi, \{p\})$, if it is obtained from the following transformation:*

1. $\Pi' = Reduct(\Pi, \{p\})$;
2. $\Pi' = \Pi' - \{r \mid r$ *is a redundant rule*$\}$;
3. $\Pi' = \Pi' - \{r \mid head(r) = p\}$;
4. $\Pi' = \Pi' - \{r \mid p \in pos(r)\}$;
5. $\Pi' = \Pi' - \{r \mid p \in neg(r)\}$;
6. $SForgetLP(\Pi, \{p\}) = \Pi'$.

In Definition 2, step 1 is just to perform reduction on Π with respect to atom p. Step 2 is to remove all redundant rules which may be introduced by the reduction of Π with respect to p. Steps 3 and 4 are to remove those rules which have p as the head or in the positive body. The intuitive meaning of Steps 3 and 4 is that after forgetting p, any atom's information in rules having p as their heads or positive bodies will be lost because they are all relevant to p, i.e. these atoms either serve as a support for p or p is in part of the supports for these atoms. On the other hand, Step 5 states that any rule containing p in its negation as failure part will also be removed. The consideration for this step is as follows. If we think $neg(r)$ is a part of support of $head(r)$, then when $p \in neg(r)$ is forgotten, $head(r)$'s entire support is lost as well. Clearly, such treatment of negation as failure in forgetting is quite strong in the sense that more atoms may be lost together with $\mathtt{not}\, p$. Therefore we call this kind of forgetting *strong forgetting*. Definition 2 can be easily extended to the case of forgetting a set of atoms:

$SForgetLP(\Pi, \emptyset) = \Pi$,
$SForgetLP(\Pi, P \cup \{p\}) =$
$\quad SForgetLP(SForgetLP(\Pi, \{p\}), P)$.

With a different way of dealing with negation as failure, we have a weak version of forgetting. We define a program to be the result of *weakly forgetting* p in Π, denoted as $WForgetLP(\Pi, \{p\})$, exactly in the same way as in Definition 2 except that Step 5 is replaced by the following step:

$\Pi' = (\Pi' - \Pi^*) \cup \Pi^\dagger$, where
$\Pi^* = \{r \mid p \in neg(r)\}$ and
$\Pi^\dagger = \{r' \mid r' : head(r) \leftarrow$
$\quad pos(r), \mathtt{not}(neg(r) - \{p\})$ where $r \in \Pi^*\}$.

It is easy to see that weakly forgetting p from a rule having p in its negative body will result in the absence of p in any case. The notion of weakly forgetting a set of atoms, denoted as $WForgetLP(\Pi, P)$, can be defined in a similar way:

$WForgetLP(\Pi, \emptyset) = \Pi$,
$WForgetLP(\Pi, P \cup \{p\}) =$
$\quad WForgetLP(\, WForgetLP(\Pi, \{p\}), P)$.

Example 2 $\Pi = \{a \leftarrow b, \mathtt{not}c,\ a \leftarrow \mathtt{not}e, d \leftarrow a, e,\ e \leftarrow \mathtt{not}a\}$. Then $SForgetLP(\Pi, \{a\}) = \{d \leftarrow b, e, \mathtt{not}c\}$, and $WForgetLP(\Pi, \{a\}) = \{d \leftarrow b, e, \mathtt{not}c,\ e \leftarrow\}$. □

A General Framework for Solving Logic Program Conflicts

We define a general framework called logic program context to represent a knowledge system which consists of multiple agents' knowledge bases. We consider the issue of conflicts occurring in the reasoning within the underlying logic program context. The notions of strong and weak forgettings provide an effective way to solve such conflicts.

Definition 3 *(***Logic program context***) A* logic program context *is a n-ary tuple $\Sigma = (\Phi_1, \cdots, \Phi_n)$, where each Φ_i is a triplet $(\Pi_i, \mathcal{C}_i, \mathcal{F}_i)$ - Π_i and $\mathcal{C}_i$ are two logic programs, and $\mathcal{F}_i \subseteq Atom(\Pi_i)$ is a set of atoms. We also call each Φ_i the ith* component *of Σ. Σ is* consistent *if for each i, $\Pi_i \cup \mathcal{C}_i$ is consistent. Σ is* conflict-free *if for any i and j, $\Pi_i \cup \mathcal{C}_j$ is consistent.*

In the above definition, for a given logic program context Σ, each component Φ_i represents agent i's local situation, where Π_i is agent i's knowledge base, $\mathcal{C}_i$ is a set of constraints that agent i should comply and will not change in any case, and $\mathcal{F}_i$ is a set of atoms that agent i may forget if necessary.

Now the problem of conflict solving under this setting can be stated as follows: given a logic program context $\Sigma = (\Phi_1, \cdots, \Phi_n)$, which may not be consistent or conflict-free, how can we find an alternative logic program context $\Sigma' = (\Phi'_1, \cdots, \Phi'_n)$ such that Σ' is conflict-free *and* is *closest* to the original Σ in some sense?

Definition 4 *(**Solution**) Let $\Sigma = (\Phi_1, \cdots, \Phi_n)$ be a logic program context, where each $\Phi_i = (\Pi_i, \mathcal{C}_i, \mathcal{F}_i)$. We call a logic program context Σ' a* solution *that solves conflicts in Σ, if Σ' satisfies the following conditions:*

1. *Σ' is conflict-free;*
2. *$\Sigma' = (\Phi'_1, \cdots, \Phi'_n)$, where $\Phi'_i = (\Pi'_i, \mathcal{C}_i, \mathcal{F}_i)$, and $\Pi'_i = SForgetLP(\Pi_i, P_i)$ or $\Pi'_i = WForgetLP(\Pi_i, P_i)$ for some $P_i \subseteq \mathcal{F}_i$.*

We denote the set of all solutions of Σ as $Solution(\Sigma)$.

Definition 5 *(**Ordering on solutions**) Let Σ, Σ' and Σ'' be three logic program contexts, where $\Sigma', \Sigma'' \in Solution(\Sigma)$. We say that Σ' is* closer *or* as close to Σ as Σ''*, denoted as $\Sigma' \preceq_\Sigma \Sigma''$, if for each i, $\Phi'_i = (\Pi'_i, \mathcal{C}_i, \mathcal{F}_i) \in \Sigma'$ and $\Phi''_i = (\Pi''_i, \mathcal{C}_i, \mathcal{F}_i) \in \Sigma''$, where $\Pi'_i = SForgetLP(\Pi_i, P_i)$ or $\Pi'_i = WForgetLP(\Pi_i, P_i)$ for some $P_i \subseteq \mathcal{F}_i$, and $\Pi''_i = SForgetLP(\Pi_i, Q_i)$ or $\Pi''_i = WForgetLP(\Pi_i, Q_i)$ for some $Q_i \subseteq \mathcal{F}_i$ respectively, we have $P_i \subseteq Q_i \subseteq \mathcal{F}_i$. We denote $\Sigma' \prec_\Sigma \Sigma''$ if $\Sigma' \preceq_\Sigma \Sigma''$ and $\Sigma'' \not\preceq_\Sigma \Sigma'$.*

Definition 6 *(**Preferred solution**) Let Σ and Σ' be two logic program contexts. We say that Σ' is a* preferred solution *of Σ, if $\Sigma' \in Solution(\Sigma)$ and there does not exist another $\Sigma'' \in Solution(\Sigma)$ such that $\Sigma'' \prec_\Sigma \Sigma'$.*

Applications of this conflict solving approach in complex problem domains, and semantic and computational foundations in relation to strong and weak forgettings and logic program contexts have been studied in detail in our paper (Zhang, Foo, & Wang 2005).

Representing Logic Program Updates

In this section, we show that three major logic program update approaches can be transformed into the framework of logic program contexts, in which all these update approaches become special cases of conflict solving problems with different types of constraints.

Representing Causal Rejection Based Approach

Eiter *et al*'s update approach is based on a principle called *causal rejection* where a sequence of logic program updates is allowed (Eiter *et al.* 2002). Let $\mathbf{P} = (\Pi_1, \cdots, \Pi_n)$, where $\Pi_1, \cdots, \Pi_1$ are extended logic programs, be an (extended logic program) *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 $\Pi_1, \cdots, \Pi_n$. We use $Lit_\mathcal{A}$ to denote the set of all literals whose corresponding atoms are in $\mathcal{A}$. 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 $\Pi_1, \cdots, \Pi_n$ and each atom $a \in \mathcal{A}$. Then Eiter *et al*'s update process is defined by the following two definitions (here we only consider ground extended logic programs in our investigation).

Definition 7 *(Eiter* et al. *2002) Given an update sequence $\mathbf{P} = (\Pi_1, \cdots, \Pi_n)$ over a set of atoms $\mathcal{A}$, the* update program $\mathbf{P}_\lhd = \Pi_1 \lhd \cdots \lhd \Pi_n$ *over $\mathcal{A}^*$ consisting of the following items:*

1. *all constraints in $\Pi_1, \cdots, \Pi_n$ (recall that a constraint is a rule with an empty head);*
2. *for each r in Π_i $(1 \leq i \leq n)$:*
   $l_i \leftarrow body(r), \texttt{not}\, rej(r)$ *if* $head(r) = \{l\}$;
3. *for each $r \in \Pi_{i-1}$ $(2 \leq i \leq n)$:*
   $rej(r) \leftarrow body(r), \neg l_i$ *if* $head(r) = \{l\}$;
4. *for each literal l occurring in $\Pi_1 \cup \cdots \cup \Pi_n$:*
   $l_{i-1} \leftarrow l_i$ $(1 < i \leq n)$, $l \leftarrow l_1$.

Intuitively, this program expresses the derivation of a literal l, beginning at Π_n downwards to Π_1. A rule in Π_n is always applicable where a rule in Π_{i-1} can only be applicable if it is not refuted by a literal derived at Π_i that is incompatible with $head(r)$. Persistence of a literal l propagates a locally derived value for l downwards to Π_1, where the local value of l is made global.

Definition 8 *(Eiter* et al. *2002) Let $\mathbf{P} = (\Pi_1, \cdots, \Pi_n)$ be an update sequence over $\mathcal{A}$. Then $S \subseteq Lit_\mathcal{A}$ is an* update answer set *of $\mathbf{P}$ iff $S = S' \cap Lit_\mathcal{A}$ for some answer set S' of $\mathbf{P}_\lhd$. The collection of all update answer sets of $\mathbf{P}$ is denoted by $\mathcal{U}(\mathbf{P})$.*

As an example, consider an update sequence $\mathbf{P} = (\Pi_1, \Pi_2, \Pi_3)$, where Π_1, Π_2 and Π_3 consist of the following rules respectively (Eiter *et al.* 2002),

Π_1:
$r_1 : sleep \leftarrow \texttt{not}\ tv\_on$,
$r_2 : night \leftarrow$,
$r_3 : tv\_on \leftarrow$,
$r_4 : watch\_tv \leftarrow tv\_on$;
Π_2:
$r_5 : \neg tv\_on \leftarrow power\_failure$,
$r_6 : power\_failure \leftarrow$,
Π_3:
$r_7 : \neg power\_failure \leftarrow$.

According to Definitions 7 and 8, it is easy to see that $\mathbf{P} = (\Pi_1, \Pi_2, \Pi_3)$ has a unique update answer set $S = \{\neg power\_failure, tv\_on, watch\_tv, night\}$, which is consistent with our intuition.

In order to transform this update approach into our framework of logic program context, we first re-formulate this approach in a normal logic program setting. In particular, given an update sequence $\mathbf{P} = (\Pi_1, \cdots, \Pi_n)$ over $\mathcal{A}$, we extend the set $\mathcal{A}$ to $\overline{A}$ by adding atom $\overline{a}$ to $\mathcal{A}$ for each $a \in \mathcal{A}$. Then by replacing each negative atom $\neg a$ occurring in Π_i with $\overline{a}$, we obtain a translated (normal logic program) update sequence $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$ over $\overline{\mathcal{A}}$

We also extend set $\overline{\mathcal{A}}$ to $\overline{\mathcal{A}}^*$ by including new atoms $rej(r)$, a_i and $\overline{a}_i$ for each rule r in $\overline{\Pi}_1, \cdots, \overline{\Pi}_n$ and each pair of atoms $a, \overline{a} \in \overline{\mathcal{A}}$. Then following Definition 7, we can obtain the corresponding update program $\overline{\mathbf{P}}_\lhd$ which is also a normal logic program. We also call a stable model of $\overline{\mathbf{P}}_\lhd$ *update stable model* of $\overline{\mathbf{P}}$.

Proposition 1 *Let $\mathbf{P} = (\Pi_1, \cdots, \Pi_n)$ be an update sequence, $\mathbf{P}_\lhd$ the update program of $\mathbf{P}$, and $\overline{\mathbf{P}}$ and $\overline{\mathbf{P}}_\lhd$ the corresponding translations of $\mathbf{P}$ and $\mathbf{P}_\lhd$ respectively as described above. $S \subseteq Lit_\mathcal{A}$ is an update answer set of $\mathbf{P}$ iff there is an update stable model $\overline{S}$ of $\overline{\mathbf{P}}$ such that $S = (\overline{S} \cap \mathcal{A}) \cup \{\neg a \mid \overline{a} \in \overline{S}\}$.*

Having Proposition 1, we only need to consider a transformation from a normal logic program update sequence $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$, where $\overline{\mathbf{P}}$ is translated from an extended logic program update sequence $\mathbf{P}$ as described above, to a conflict solving problem under the framework of logic program contexts.

Definition 9 *Let $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$ $(n > 1)$ be a normal logic program update sequence over $\overline{\mathcal{A}}$. We specify a sequence of logic program contexts $\Omega_{CR} = (\Sigma_1, \cdots, \Sigma_{n-1})$[1] over the set of atoms $\mathcal{B} = \overline{\mathcal{A}}^* \cup \{l_{a_i}, l_{\overline{a}_i} \mid a_i, \overline{a}_i \in \overline{\mathcal{A}}^*, i = 1, \cdots, n\}$ where l_{a_i} and $l_{\overline{a}_i}$ are newly introduced atoms:*

1. $\Sigma_1 = ((\overline{\Pi}_1^*, \emptyset, \mathcal{F}_1), (\emptyset, \mathcal{C}_1, \emptyset))$, *where $\overline{\Pi}_1^*$ consists of the following rules:*
   (a) *all constraints in $\overline{\Pi}_1, \cdots, \overline{\Pi}_n$;*
   (b) *for each $r \in \overline{\Pi}_i$: $a \leftarrow body(r)$ or $\overline{a} \leftarrow body(r)$ $(1 \leq i \leq n)$,*
   $a_i \leftarrow body(r), \texttt{not}\ l_{\overline{a}_i}$, $\quad \overline{a}_i \leftarrow B(r), \texttt{not}\ l_{a_i}$ *respectively,*
   (c) *for each $a, \overline{a}$ in $\overline{\mathcal{A}}$,*
   $a_{i-1} \leftarrow a_i$, $\quad \overline{a}_{i-1} \leftarrow \overline{a}_i$ $(1 < i \leq n)$,
   $a \leftarrow a_1$, $\quad \overline{a} \leftarrow \overline{a}_1$.
   (d) $\mathcal{F}_1 = \{l_{a_{n-1}}, l_{\overline{a}_{n-1}} \mid \forall a \in \mathcal{A}\}$,
   (e) $\mathcal{C}_1 = \{\leftarrow a_{n-1}, \overline{a}_n, \ \leftarrow \overline{a}_{n-1}, a_n \mid \forall a \in \mathcal{A}\}$;
2. $\Sigma_i = ((\overline{\Pi}_i^*, \emptyset, \mathcal{F}_i), (\emptyset, \mathcal{C}_i, \emptyset))$ $(1 < i \leq n-1)$, *where*
   (a) $\overline{\Pi}_i^* = \overline{\Pi}_{i-1}^\dagger$, *and $\overline{\Pi}_{i-1}^\dagger$ is in a preferred solution of Σ_{i-1}: $\Sigma'_{i-1} = ((\overline{\Pi}_{i-1}^\dagger, \emptyset, \mathcal{F}_{i-1}), (\emptyset, \mathcal{C}_{i-1}, \emptyset))$,*
   (b) $\mathcal{F}_i = \{l_{a_{n-i}}, l_{\overline{a}_{n-i}} \mid \forall a \in \mathcal{A}\}$,
   (c) $\mathcal{C}_i = \{\leftarrow a_{n-i}, \overline{a}_{n-i+1}, \ \leftarrow \overline{a}_{n-i}, a_{n-i+1} \mid \forall a \in \mathcal{A}\}$.

A set of atoms $S' \subseteq \mathcal{B}$ is called a model *of Ω_{CR} if S' is a stable model of $\overline{\Pi}_{n-1}^\dagger$, where $\overline{\Pi}_{n-1}^\dagger$ is in a preferred solution of Σ_{n-1}: $\Sigma'_{n-1} = ((\overline{\Pi}_{n-1}^\dagger, \emptyset, \mathcal{F}_{n-1}), (\emptyset, \mathcal{C}_{n-1}, \emptyset))$.*

Let us take a closer look at Definition 9. Given an update sequence $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$, Definition 9 specifies a sequence of logic program contexts $\Omega_{CR} = (\Sigma_1, \cdots, \Sigma_{n-1})$, where each Σ_i solves certain conflicts embedded in $\overline{\mathbf{P}}$. Σ_1 represents the first level of conflict solving, where $\overline{\Pi}_1^*$ is similar to $\overline{\mathbf{P}}_\lhd$ except that the possible conflict between a_{n-1} and $\overline{a}_n$ (or $\overline{a}_{n-1}$ and a_n) has been reformulated as a constraint $\leftarrow a_{n-1}, \overline{a}_n$ (or $\leftarrow \overline{a}_{n-1}, a_n$ resp.) in $\mathcal{C}_1$. Note that in rules specified in 1.(b) of Definition 9: $a_i \leftarrow body(r), \texttt{not}\ l_{\overline{a}_i}$, $\overline{a}_i \leftarrow body(r), \texttt{not}\ l_{a_i}$, formulas $\texttt{not}\ l_{\overline{a}_{n-1}}$ and $\texttt{not}\ l_{a_{n-1}}$ (here $i = n-1$) are introduced to solve the conflict between a_{n-1} and $\overline{a}_n$ (or $\overline{a}_{n-1}$ and a_n resp.).

Observe that Σ_1 *only* solves conflicts between atoms at level $n-1$. For example, if both a_{n-1} and $\overline{a}_n$ can be derived from $\overline{\Pi}_1^*$, then rule $a_{n-1} \leftarrow body(r), \texttt{not}\ l_{\overline{a}_{n-1}}$ will be eliminated from Π_1 by strongly forgetting atom $l_{\overline{a}_{n-1}}$ under the constraint $\leftarrow a_{n-1}, \overline{a}_n$ in $\mathcal{C}_1$.

[1]Note that when $n = 1$ our transformation becomes trivial since we can simply specify Ω_{CR} to consist of a single logic program context $\Sigma = ((\overline{\Pi}_1, \emptyset, \emptyset), (\emptyset, \emptyset, \emptyset))$. In this case Σ has a (preferred) solution iff $\overline{\Pi}_1$ is consistent. So in the rest of the paper we will only consider the case $n > 1$.

In the sequence $\Omega_{CR} = (\Sigma_1, \cdots, \Sigma_{n-1})$, conflicts are solved in a *downwards* manner with respect to the update sequence $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$, where each Σ_i $(i > 1)$ is specified for the purpose of solving conflicts between atoms a_{n-i} and $\overline{a}_{n-i+1}$ (or $\overline{a}_{n-i}$ and a_{n-i+1}).

Example 3 Consider the TV example mentioned earlier again, where $\mathbf{P} = (\Pi_1, \Pi_2, \Pi_3)$ is an update sequence. It is easy to translate $\mathbf{P}$ to the corresponding normal logic program update sequence $\overline{\mathbf{P}} = (\overline{\Pi}_1, \overline{\Pi}_2, \overline{\Pi}_3)$, where $\neg tv\_on$ and $\neg power\_failure$ are replaced by atoms $\overline{tv\_on}$ and $\overline{power\_failure}$ respectively. According to Definition 9, we then specify a sequence of logic program contexts $\Omega_{CR} = (\Sigma_1, \Sigma_2)$ to solve the conflict occurring in $\overline{\mathbf{P}}$. $\Sigma_1 = ((\overline{\Pi}_1^*, \emptyset, \mathcal{F}_1), (\emptyset, \mathcal{C}_1, \emptyset))$, where $\overline{\Pi}_1^*$ consists of the following rules[2]:

$sleep_1 \leftarrow \texttt{not}\ tv\_on, \texttt{not}\ l_{\overline{sleep_1}}$,
$night_1 \leftarrow \texttt{not}\ l_{\overline{night_1}}$,
$tv\_on_1 \leftarrow \texttt{not}\ l_{\overline{tv\_on_1}}$,
$watch\_tv_1 \leftarrow tv\_on, \texttt{not}\ l_{\overline{watch\_tv_1}}$,
$\overline{tv\_on}_2 \leftarrow power\_failure, \texttt{not}\ l_{tv\_on_2}$,
$power\_failure_2 \leftarrow \texttt{not}\ l_{\overline{power\_failure_2}}$,
$\overline{power\_failure}_3 \leftarrow \texttt{not}\ l_{power\_failure_3}$,
$night \leftarrow night_1$,
$tv\_on \leftarrow tv\_on_1$,
$watch\_tv \leftarrow watch\_tv_1$,
$\overline{tv\_on}_1 \leftarrow \overline{tv\_on}_2$,
$\overline{tv\_on} \leftarrow \overline{tv\_on}_1$,
$\overline{power\_failure}_2 \leftarrow \overline{power\_failure}_3$,
$\overline{power\_failure}_1 \leftarrow \overline{power\_failure}_2$,
$\overline{power\_failure} \leftarrow \overline{power\_failure}_1$,
$power\_failure_1 \leftarrow power\_failure_2$,
$power\_failure \leftarrow power\_failure_1$,

$\mathcal{F}_1 = \{l_{power\_failure_2}, l_{\overline{power\_failure_2}}\}$, and
$\mathcal{C}_1 = \{\leftarrow power\_failure_2, \overline{power\_failure}_3\}$.

It is easy to see that Σ_1 is not conflict free since $\overline{\Pi}_1^* \cup \mathcal{C}_1$ is not consistent (i.e it has no stable model). To specify Σ_2, we first need to obtain a preferred solution of Σ_1. In fact Σ_1 has a unique preferred solution $\Sigma'_1 = ((\overline{\Pi}_1^\dagger, \emptyset, \mathcal{F}_1), (\emptyset, \mathcal{C}_1, \emptyset))$, where $\overline{\Pi}_1^\dagger = SForgetLP(\overline{\Pi}_1^*, \{l_{\overline{power\_failure_2}}\}) = \overline{\Pi}_1^* - \{power\_failure_2 \leftarrow \texttt{not}\ l_{\overline{power\_failure_2}}\}$.

Now we specify $\Sigma_2 = ((\overline{\Pi}_1^\dagger, \emptyset, \mathcal{F}_2), (\emptyset, \mathcal{C}_2, \emptyset))$, where $\mathcal{F}_2 = \{l_{tv\_on_1}, l_{\overline{tv\_on_1}}\}$ and $\mathcal{C}_2 = \{\leftarrow tv\_on_1, \overline{tv\_on}_2\}$. Note that Σ_2 is already conflict free. So by ignoring those atoms with subscripts, Ω_{CR} has a unique model $\{\overline{power\_failure}, tv\_on, watch\_tv, night\}$, which is the same as the update stable model of update sequence $\overline{\mathbf{P}}$. □

Theorem 1 *Let $\overline{\mathbf{P}}$ be a normal logic program update sequence over the set of atoms $\overline{\mathcal{A}}$, and $\Omega_{CR} = (\Sigma_1, \cdots, \Sigma_{n-1})$ a sequence of logic program contexts as specified in Definition 9. A subset $\overline{S}$ of $\overline{\mathcal{A}}$ is an update stable model of $\overline{\mathbf{P}}$ iff Ω_{CR} has a model S' such that $\overline{S} = S' \cap \overline{\mathcal{A}}$.*

[2]To avoid unnecessarily tedious details, here we omit some irrelevant rules and atoms from $\overline{\Pi}_1^*$, $\mathcal{F}_1$ and $\mathcal{C}_1$. The same for Σ_2.

Representing Dynamic Logic Programming

Logic program update based on dynamic logic programming (DLP) (or simply called DLP update approach) was proposed by Alferes, Leite, Pereira, *et al* (Alferes *et al.* 1998), and then extended for various purposes (Leite 2003). Using a similar technique as described in last subsection, we can also transform the DLP update approach into the framework of logic program contexts and prove an equivalence theorem between these two formulations.

Due to a space limit, the details are referred to our full paper. Here we only highlight that: compared to the transformation presented earlier, the main feature to transform a dynamic update program $\mathbf{P}_{\oplus} = \Pi_1 \oplus \cdots \oplus \Pi_n$ (readers are referred to (Leite 2003) for the definition of $\mathbf{P}_{\oplus}$) to a sequence of logic program contexts $\Omega_{DLP} = (\Sigma_1, \cdots, \Sigma_n)$ is to employ an *upwards* conflict solving policy to solve possible conflicts between atoms (recall Definition 9 uses a downwards method to solve conflicts among atoms). For example, for each pair of inheritance rules in $\mathbf{P}_{\oplus}$:

$a_i \leftarrow a_{i-1}, \texttt{not}\ \overline{a}_{P_i}$, and
$\overline{a}_{i-1} \leftarrow \overline{a}_{i-1}, \texttt{not}\ a_{P_i}$

we replace them with the following rules in Ω_{DLP}:

$a_i \leftarrow a_{i-1}, \texttt{not}\ l_{a_i}$, $\quad \overline{a}_{i-1} \leftarrow \overline{a}_{i-1}, \texttt{not}\ l_{\overline{a}_i}$,
$l_{a_i} \leftarrow \texttt{not}\ h_{a_i}$, $\quad l_{\overline{a}_i} \leftarrow \texttt{not}\ h_{\overline{a}_i}$,
$l_{a_i} \leftarrow \overline{a}_{P_i}$, $\quad l_{\overline{a}_i} \leftarrow a_{P_i}$,

and under constraints $\leftarrow a_i, \overline{a}_{P_i}$ and $\leftarrow \overline{a}_i, a_{P_i}$, conflicts are solved by weakly forgetting atoms from $\{h_{a_i}, h_{\overline{a}_i}\}$[3].

Representing Updates as Constraints for Conflict Solving

Sakama and Inoue's update approach (or simply denoted as SI approach) is viewed as a typical syntax based logic program update approach (Sakama & Inoue 1999), which solves conflicts between two programs on a basis of syntactic coherence. In (Zhang, Foo, & Wang 2005), we have showed that there is a simple one-to-one correspondence between SI approach and our conflict solving framework.

From previous descriptions and (Zhang, Foo, & Wang 2005), we observe that the key step to transform an update approach into a sequence of logic program contexts (or one logic program context like the case of SI approach) is to construct the underlying constraints for conflict solving. In both Eiter et al's causal rejection and DLP approaches, constraints are specified based on atoms, e.g. $\leftarrow a_{n-i}, \overline{a}_{n-i+1}$ in Ω_{CR}, and $\leftarrow a_i, a_{P_i^-}$ in Ω_{DLP}.

For SI approach, on the other hand, the underlying constraints are specified as the entire update program. For instance, consider the update of Π_1 by Π_2 using SI approach, the corresponding logic program context for this update is of the form $\Sigma = ((\Pi, \emptyset, \mathcal{F}), (\emptyset, \Pi_2, \emptyset))$, in which program Π_2 serves as constraints for conflict solving (Zhang, Foo, & Wang 2005).

From the above observation, we can see that the main difference between model based and syntax based update approaches is to solve conflicts under different types of constraints, namely atoms based and program based constraints respectively.

[3]Recall that in Ω_{CR} conflicts are solved through strong forgettings.

Representing Integrated Update Approach

Different from both model based and syntax based approaches, Zhang and Foo's update approach integrated both desirable semantic and syntactic features of (extended) logic program updates (Zhang & Foo 1998). Their approach also solves default conflicts caused by negation as failure in logic programs by using a prioritized logic programming language. Consequently, Zhang and Foo's update approach can generate an explicit resulting program for a logic program update and also avoid some undesirable solutions embedded in Sakama-Inoue's approach (Zhang 2005).

Since we do not consider default conflict solving in this paper, we will only focus on the transformation from first part of Zhang-Foo's update approach, that is, the conflict (contradiction) elimination, into a logic program context.

Let Π_1 and Π_2 be two extended logic programs. Updating Π_1 with Π_2 consists of two stages. Step (1): Simple fact update - updating an answer set S of Π_1 by program Π_2. The result of this update is a collection of sets of literals, denoted as $Update(S, \Pi_2)$. Step (2): Select a $S' \in Update(S, \Pi_2)$, and extract a maximal subset Π^* of Π_1 such that program $\Pi^* \cup \{l \leftarrow | \ l \in S'\}$ (or simply represented as $\Pi^* \cup S'$) is consistent. Then $\Pi^* \cup \Pi_2$ is called a *resulting program* of updating Π_1 with Π_2.

Note that in Step (1), the simple fact update is achieved through a prioritized logic programming (Zhang & Foo 1998). Recently, Zhang proved an equivalence relationship between the simple fact update and Sakama and Inoue's program update (Zhang 2005): $Update(S, \Pi_2) = \bigcup \mathcal{S}(SI\text{-}Update(\Pi(S), \Pi_2))$, where $\Pi(S) = \{l \leftarrow | \ l \in S\}$, and $\bigcup \mathcal{S}(SI\text{-}Update(\Pi(S), \Pi_2))$ is the class of all answer sets of resulting programs after updating $\Pi(S)$ by Π_2 using Sakama-Inoue's approach.

Example 4 Consider two extended logic programs Π_1 and Π_2 as follows:

Π_1: $\quad \Pi_2$:
$a \leftarrow$, $\quad b \leftarrow a$,
$c \leftarrow b$, $\quad \neg c \leftarrow b$.
$d \leftarrow \texttt{not}\ e$.

Π_1 has a unique answer set $\{a, d\}$. Then Step (1) Zhang-Foo's simple fact update of $\{a, d\}$ by Π_2, $Update(\{a, d\}, \Pi_2)$, which is equivalently to update $\{a \leftarrow, d \leftarrow\}$ with Π_2 using Sakama-Inoue's approach, will contain a single set $\{a, b, \neg c, d\}$. Applying Step (2), we obtain the final update result $\{a \leftarrow, d \leftarrow \texttt{not}\ e\} \cup \Pi_2$. Note that directly using Sakma-Inoue's approach to update Π_1 with Π_2 will obtain an extra unwanted resulting program $\{c \leftarrow b, d \leftarrow \texttt{not}\ e\} \cup \Pi_2$. □

As we have already provided a transformation from Sakama-Inoue's approach to a logic program context, to show that Zhang-Foo's update approach can also be represented within our framework, it is sufficient to only transform Step (2) above into a conflict solving problem under certain logic program context.

As before, given two extended logic programs Π_1 and Π_2 over the set of atoms $\mathcal{A}$, we extend $\mathcal{A}$ to $\overline{\mathcal{A}}$ with new atom $\overline{a}$ for each $a \in \mathcal{A}$. Then by replacing each $\neg a$ in S' and Π_2 with $\overline{a}$, we obtain the corresponding normal logic programs $\overline{\Pi}_1$ and $\overline{\Pi}_2$ respectively. Suppose $Update(\overline{S}, \overline{\Pi}_2)$ is the result of the simple fact update, where $\overline{S}$ is a stable model of $\overline{\Pi}_1$.

Definition 10 *Let $\overline{\Pi}_1$, $\overline{\Pi}_2$, and $Update(\overline{S}, \overline{\Pi}_2)$ be defined as above, and $\overline{S'} \in Update(\overline{S}, \overline{\Pi}_2)$. We specify a logic program context $\Sigma_{ZF} = ((\overline{\Pi'}_1, \emptyset, \mathcal{F}), (\emptyset, \mathcal{C}, \emptyset))$ over the set of atoms $\overline{A} \cup \{l_r \mid r \in \overline{\Pi}_1\}$ where l_r are newly introduced atoms:*

1. *$\overline{\Pi'}_1$ consists of rules: (a) for each rule $r : head(r) \leftarrow body(r)$ in $\overline{\Pi}_1$, $head(r) \leftarrow body(r), \texttt{not}\ l_r$ is in $\overline{\Pi'}_1$, and (b) $\overline{S'} \subseteq \overline{\Pi'}_1$,*
2. *$\mathcal{F} = \{l_r \mid r \in \overline{\Pi}_1\}$,*
3. *$\mathcal{C} = \{\leftarrow a, \overline{a} \mid a, \overline{a} \in \overline{A}\}$.*

The following theorem shows that Step (2) in Zhang-Foo's approach can be precisely characterized by a logic program context specified in Definition 10.

Theorem 2 *Let $\overline{\Pi}_1$, $\overline{\Pi}_2$, and $Update(\overline{S}, \overline{\Pi}_2)$ be defined as above, and $\overline{S'} \in Update(\overline{S}, \overline{\Pi}_2)$. $\overline{\Pi}^*$ is a maximal subset of $\overline{\Pi}_1$ such that $\overline{\Pi'} = \overline{\Pi}^* \cup \overline{S'}$ is consistent iff $\overline{\Pi'}$ is in a preferred solution of Σ_{ZF}: $\Sigma'_{ZF} = ((\overline{\Pi'}, \emptyset, \mathcal{F}), (\emptyset, \mathcal{C}, \emptyset))$.*

Computational Issues

In this section, we consider the related computational issues regarding the transformations from different update approaches into the framework of logic program contexts.

Given a (normal logic program) update sequence $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$, translating it to a sequence of logic program contexts $\Omega_{CR} = (\Sigma_1, \cdots, \Sigma_{n-1})$ is usually expensive because each Σ_i $(1 < i \leq n-1)$ is defined based on the computation of a preferred solution of Σ_i, which is only achievable in exponential time generally. More precisely, we have the following complexity result.

Theorem 3 *Let $\overline{\mathbf{P}} = (\overline{\Pi}_1, \cdots, \overline{\Pi}_n)$ be an update sequence over $\overline{A}$, and $(\Sigma_1, \cdots, \Sigma_{n-1})$ an arbitrary sequence of logic program contexts built upon $\mathcal{B} = \overline{A}^* \cup \{l_{a_i}, l_{\overline{a}_i} \mid a_i, \overline{a}_i \in \overline{\mathcal{A}}^*, i = 1, \cdots, n\}$. Deciding whether $(\Sigma_1, \cdots, \Sigma_{n-1})$ sequence satisfies conditions of Definition 9 is Π_2^P-complete. The hardness holds even if $n = 3$.*

Theorem 3 tells us that representing Eiter *et al*'s causal rejection based update approach within the framework of logic program contexts has to pay higher computational costs. We can obtain a similar result for the DLP approach. The following theorem, however, shows that performing Zhang-Foo's program updates (and the same for Sakama-Inoue's approach) in our framework are not harder than using their original approaches.

Theorem 4 *The following results hold:*

1. *Logic program context Σ_{ZF} can be constructed in polynomial time;*
2. *Deciding whether a given logic program context Σ' is a preferred solution of Σ_{ZF} is Π_2^P-complete.*

Conclusion

In this paper, we moved one step further to study a general framework for solving logic program conflicts, where conflicts among different agents are solved through strong and weak forgetting operations on corresponding logic programs. We showed that under this framework, all major approaches of logic program updates become special cases of conflict solving within certain logic program contexts (or sequences of logic program contexts), and program updates are achieved by solving conflicts in these contexts. To the best of our knowledge, this is the first general framework to precisely represent all major logic program update approaches. Our complexity results also confirm that achieving syntactic results for solving logic program conflicts has to pay higher computational costs than obtaining model based results.

Recently Eiter et al also developed a general framework to model nonmonotonic knowledge base evolution (Eiter *et al.* 2005). Since their framework is built upon an abstract model, the underlying update semantics has to directly depend on the specific update approach captured by the framework.

References

Alferes, J.; Leite, J.; Pereira, L.; and *et al*. 1998. Dynamic logic programming. In *Proceedings of KR-98*, 98–111.

Brass, S., and Dix, J. 1999. Semantics of (disjunctive) logic programs based on partial evaluation. *Journal of Logic programming* 40(1):1–46.

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.; Fink, M.; Sabbatini, G.; and Tompits, H. 2005. Reasoning about evolving nonmonotonic knowledge bases. *ACM TOCL* 6:389–440.

Lang, J.; Liberatore, P.; and Marquis, P. 2003. Propositional independence - formula-variable independence and forgetting. *Journal of Artificial Intelligence Research* 18:391–443.

Leite, J. 2003. *Evolving Knowledge Bases: Specification and Semantics*. IOS Press.

Lin, F. 2001. On the strongest necessary and weakest sufficient conditions. *Artificial Intelligence* 128:143–159.

Sakama, C., and Inoue, K. 1999. Updating extended logic programs through abduction. In *Proceedings of LPNMR'99*, 2–17.

Zhang, Y., and Foo, N. 1998. Updating logic programs. In *Proceedings of ECAI-1998*, 403–407.

Zhang, Y.; Foo, N.; and Wang, K. 2005. Solving logic program conflicts through strong and weak forgetting. In *Proceedings of IJCAI05*, to appear.

Zhang, Y. 2005. Logic program based updates. *ACM Transaction on Computational Logic* - :http://www.acm.org/pubs/tocl/accepted.html.