Forgetting and Conflict Resolving in Disjunctive Logic Programming

Author
Eiter, Thomas, Wang, Kewen

Published
2006

Conference Title
Proceedings, Twenty-First National Conference on Artificial Intelligence (AAAI-06) and the Eighteenth Innovative Applications of Artificial Intelligence Conference (IAAI-06)

Copyright Statement
Copyright 2006 AAAI Press. Reproduced in accordance with the copyright policy of the publisher. Use hypertext link for access to conference website.

Downloaded from
http://hdl.handle.net/10072/11585

Link to published version
http://www.aaai.org/Press/Proceedings/aaai06.php
Forgetting and Conflict Resolving in Disjunctive Logic Programming

Thomas Eiter
Technische Universität Wien, Austria
eiter@kr.tuwien.ac.at

Kewen Wang†
Griffith University, Australia
k.wang@griffith.edu.au

Abstract

We establish a declarative theory of forgetting for disjunctive logic programs. The suitability of this theory is justified by a number of desirable properties. In particular, one of our results shows that our notion of forgetting is completely captured by the classical forgetting. We also provide an analysis of computational complexity. As an application of our approach, a fairly general framework for resolving conflicts in inconsistent knowledge bases represented by disjunctive logic programs is defined.

Introduction

Forgetting (Lin & Reiter 1994; Lang, Liberatore, & Marquis 2003) is a key issue for adequately handle a range of classical tasks such as query answering, planning, decision-making, reasoning about actions, or knowledge update and revision. It is, moreover, also important in recently emerging issues such as design and engineering of Web-based ontology languages. Suppose we start to design an ontology of Pets, which is a knowledge base of various pets (like cats, dogs but not lions or tigers). Currently, there are numerous ontologies on the Web. We navigated the Web and found an ontology Animals which is a large ontology on various animals including cats, dogs, tigers and lions. It is not a good idea to download the whole ontology Animals. The approach in the current Web ontology language standard OWL ¹ is to discard those terminologies that are not desired (although this function is still very limited in OWL). For example, we may discard (or forget) tigers and lions from the ontology Animals. If our ontology is only a list of relations, we can handle the forgetting (or discarding) easily. However, an ontology is often represented as a logical theory, and the removal of one term may influence other terms in the ontology. Thus, more advanced methods are needed.

Disjunctive logic programming (DLP) under the answer set semantics (Gelfond & Lifschitz 1990) is now widely accepted as a major tool for knowledge representation and commonsense reasoning (Baral 2002). DLP is expressive in that it allows disjunction in rule heads and strong negation in both heads and bodies. Studying forgetting within DLP is thus a natural issue, and we make in this paper the following contributions:

- We establish a declarative, semantically defined notion of forgetting for disjunctive logic programs, which is a generalization of the corresponding notion for nondisjunctive programs proposed in (Wang, Sattar, & Su 2005). The suitability of this theory is justified by a number of desirable properties.
- We present a transformation-based algorithm for computing the result of forgetting. This method allows to obtain the result of forgetting a literal \( l \) in a logic program via a series of program transformations and other rewritings.
- Connected with the transformation algorithm, we settle some complexity issues for reasoning under forgetting. They provide useful insight into feasible representations of forgetting.
- As an application of our approach, we present a fairly general framework for resolving conflicts in inconsistent knowledge bases. The basic idea of this framework is to weaken the preferences of each agent by forgetting certain knowledge that causes inconsistency. In particular, we show how to use the notion of forgetting to provide an elegant solution for preference elicitation in DLP.

Preliminaries

We briefly review some basic definitions and notation used throughout this paper.
A disjunctive program is a finite set of rules of the form
\[ a_1 \land \cdots \land a_s \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n, \]
where \( a, b \)'s and \( c \)'s are classical literals in a propositional language. A literal is a positive literal \( p \) or a negative literal \( \neg p \) for some atom \( p \). An NAF-literal is of the form \( \text{not } l \) where \( \text{not } \) is for the negation as failure and \( l \) is an (ordinary) literal. For an atom \( p, p \) and \( \neg p \) are called complementary. For any literal \( l \), its complementary literal is denoted \( \neg l \).

To guarantee the termination of some program transformations, the body of a rule is a set of literals rather than a multiset.

Given a rule \( r \) of form (1), \( \text{head}(r) = a_1 \lor \cdots \lor a_s \) and \( \text{body}(r) = \text{body}^+(r) \cup \text{not } \text{body}^-(r) \) where \( \text{body}^+(r) = \{ b_1, \ldots, b_m \} \), \( \text{body}^-(r) = \{ c_1, \ldots, c_n \} \), and \( \text{not } \text{body}^-(r) = \{ q | q \in \text{body}^-(r) \} \).

A rule \( r \) of the form (1) is normal or non-disjunctive, if \( s \leq 1 \); positive, if \( n = 0 \); negative, if \( m = 0 \); constraint, if \( s = 0 \); fact, if \( m = 0 \) and \( n = 0 \), in particular, a rule with \( s = n = m = 0 \) is the constant \( \text{false} \).

A disjunctive program \( P \) is called normal program (resp. positive program, negative program), if every rule in \( P \) is normal (resp. positive, negative).

Let \( P \) be a disjunctive program and let \( X \) be a set of literals. A disjunction \( a_1 \lor \cdots \lor a_s \) is satisfied by \( X \), denoted \( X \models a_1 \lor \cdots \lor a_s \) if \( a_i \in X \) for some \( i \) with \( 1 \leq i \leq s \). A rule \( r \) is satisfied by \( X \), denoted \( X \models r \) iff \( \text{body}^+(r) \subseteq X \) and \( \text{body}^-(r) \cap X = \emptyset \) imply \( X \models \text{head}(r) \). \( X \) is a model of \( P \), denoted \( X \models P \) if every rule of \( P \) is satisfied by \( X \).

An interpretation \( X \) is a set of literals that contains no pair of complementary literals.

The answer set semantics The reduct of \( P \) on \( X \) is defined as \( \text{P}^X = \{ \text{head}(r) \leftarrow \text{body}^+(r) \mid r \in P, \text{body}^-(r) \cap X = \emptyset \} \). An interpretation \( X \) is an answer set of \( P \) if \( X \) is a minimal model of \( \text{P}^X \) (by treating each literal as a new atom). \( \text{AS}(P) \) denotes the collection of all answer sets of \( P \), \( P \) is consistent if it has at least one answer set.

Two disjunctive programs \( P \) and \( P' \) are equivalent, denoted \( P \equiv P' \), if \( \text{AS}(P) = \text{AS}(P') \).

As usual, \( B_P \) is the Herbrand base of logic program \( P \), that is, the set of all (ground) literals in \( P \).

Forgetting in Logic Programming

In this section, we want to define what it means to forget about a literal \( l \) in a disjunctive program \( P \). The idea is to obtain a logic program which is equivalent to the original disjunctive program, if we ignore the existence of the literal \( l \). We believe that forgetting should go beyond syntactic removal of rules/literals and be close to classical forgetting and answer set semantics (keeping its spirit) at the same time. Thus, the definition of forgetting in this section is given in semantics terms, i.e., based on answer sets, and naturally generalizes the corresponding one in (Wang, Sattar, & Su 2005).

In propositional logic, the result of forgetting \( \text{forget}(T, p) \) about a property \( T \) in a theory \( T \) is conveniently defined as \( T(p/\text{true}) \lor T(p/\text{false}) \). This way cannot be directly generalized to logic programming since there is no notion of the "disjunction" of two logic programs. However, if we examine the classical forgetting in model-theoretic point of view, we can obtain the models of \( \text{forget}(T, p) \) in this way: first compute all models of \( T \) and remove \( p \) from each model if it contains \( p \). The resulting collection of sets \( \{ M \setminus \{ p \} \mid M \models T \} \) is exactly the set of all models of \( \text{forget}(T, p) \).

Similarly, given a consistent disjunctive program \( P \) and a literal \( l \), we naïvely could define the result of forgetting about \( l \) in \( P \) as an extended disjunctive program \( P' \) whose answer sets are exactly \( \text{AS}(P) \setminus \{ l \} \setminus X \in \text{AS}(P) \). However, this notion of forgetting cannot guarantee the existence of \( P' \) for even simple programs. For example, consider \( P = \{ b \leftarrow p, \neg q \} \) then \( \text{AS}(P) = \{ \{ a, p \}, \{ a, q \} \} \) and thus \( \text{AS}(P) \setminus \{ p \} = \{ \{ a, q \} \} \). Since \( \{ a \} \subset \{ a, q \} \) and, as well-known, answer sets are incomparable under set inclusion, \( \text{AS}(P) \setminus \{ p \} \) cannot be the set of answer sets of any disjunctive program.

A solution to this problem is a suitable notion of minimal answer set such that the definition of answer sets, minimality, and forgetting can be fruitfully combined. To this end, we call a set \( X \) an l-subset of a set \( X \), denoted \( X' \subseteq_l X \), if \( X' \setminus \{ l \} \subseteq X \setminus \{ l \} \). Similarly, a set \( X' \) is a strict l-subset of \( X \), denoted \( X' \subset_l X \), if \( X' \subseteq l X \setminus \{ l \} \). Two sets \( X \) and \( X' \) of literals are l-equivalent, denoted \( X \sim_l X' \), if \( (X \setminus X') \cup (X' \setminus X) \subseteq \{ l \} \).

Definition 1 Let \( P \) be a consistent disjunctive program, let \( l \) be a literal in \( P \) and let \( X \) be a set of literals.

1. For a collection \( S \) of sets of literals, \( X \in S \) is l-minimal if there is no \( X' \in S \) such that \( X' \subseteq_l X \). \( \text{min}_l(S) \) denotes the collection of all l-minimal elements in \( S \).

2. An answer set \( X \) of disjunctive program \( P \) is an l-answer set if \( X \) is l-minimal in \( \text{AS}(P) \). \( \text{AS}_l(P) \) consists of all l-answer sets of \( P \).

To make \( \text{AS}(P) \) \( - p \) incomparable, we could take either minimal elements or maximal elements from \( \text{AS}(P) \) \( - p \). However, selecting minimal answer sets is in line with semantic principles to minimize positive information.

For example, \( P = \{ a \leftarrow p \lor q \} \), has two answer sets \( X = \{ a, p \} \) and \( X' = \{ a, q \} \). \( X \) is a p-answer set of \( P \), but \( X' \) is not. This example shows that, for a disjunctive program \( P \) and a literal \( l \), not every answer set is an l-answer set.

In the rest of this paper, we assume that \( P \) is a consistent program. The following proposition collects some easy properties of l-answer sets.

Proposition 1 For any consistent program \( P \) and a literal \( l \) in \( P \), the following four items are true:

1. An l-answer set \( X \) of \( P \) must be an answer set of \( P \).
2. For any answer set \( X \) of \( P \), there is an l-answer set \( X' \) of \( P \) such that \( X' \subseteq_l X \).
3. Any answer set \( X \) of \( P \) with \( l \in X \) is an l-answer set of \( P \).
4. If an answer set \( X \) of \( P \) is not an l-answer set, then (1) \( l \notin X \); (2) there exists an l-answer set \( Y \) of \( P \) such that \( l \in Y \subseteq_l X \).
Having the notion of minimality about forgetting a literal, we are now in a position to define the result of forgetting about a literal in a disjunctive program.

**Definition 2** Let $P$ be a consistent disjunctive program and $l$ be a literal. A disjunctive program $P'$ is a result of forgetting about $l$ in $P$, if $P'$ represents $l$-answer sets of $P$, i.e., the following conditions are satisfied:

1. $B_{P'} \subseteq B_P \setminus \{l\}$ and
2. For any set $X'$ of literals with $l \notin X'$, $X'$ is an answer set of $P'$ iff there is an $l$-answer set $X$ of $P$ such that $X' \sim_l X$.

Notice that the first condition implies that $l$ does not appear in $P'$. An important difference of the notion of forgetting here from existing approaches to updating and merging logic programs is that only $l$ and possibly some other literals are removed. In particular, no new symbol is introduced in $P'$.

For a consistent extended program $P$ and a literal $l$, some program $P'$ as in the above definition always exists (cf. Algorithm 1 for details). However, different such programs $P'$ might exist. It follows from the above definition that they are all equivalent under the answer set semantics.

**Proposition 2** Let $P$ be a disjunctive program and $l$ a literal in $P$. If $P'$ and $P''$ are two results of forgetting about $l$ in $P$, then $P'$ and $P''$ are equivalent.

We use $\text{forget}(P, l)$ to denote a possible result of forgetting about $l$ in $P$.

**Example 1** 1. If $P_1 = \{ q \leftarrow \neg p \}$, then $\text{forget}(P_1, q) = \emptyset$ and $\text{forget}(P_1, p) = \{ q \leftarrow \}$. 2. If $P_2 = \{ p \land q \leftarrow \}$, then forgetting $P_2, p$ is the same as forgetting $P_2, q$. 3. $P_3 = \{ p \lor q \leftarrow \neg p, c \leftarrow q \}$ has the unique answer set $\{q, c\}$ and $\text{forget}(P_3, p) = \{ q \leftarrow, c \leftarrow \}$. 4. $P_4 = \{ a \lor p \leftarrow \neg b, c \leftarrow \neg p, b \leftarrow \}$. Then $\text{forget}(P_4, p) = \{ c \leftarrow, b \leftarrow \}$.

We will explain how to obtain $\text{forget}(P, l)$ in the next section. The following proposition generalizes Proposition 2.

**Proposition 3** Let $P$ and $P'$ be two equivalent disjunctive programs and $l$ a literal in $P$. Then $\text{forget}(P, l)$ and $\text{forget}(P', l)$ are also equivalent.

However, forgetting here does not preserve some special equivalences of logic programs stronger than ordinary equivalence like strong equivalence (Lifschitz, Tang, & Turner 1999) or uniform equivalence (Eiter & Fink 2003). This will be discussed elsewhere.

**Proposition 4** For any consistent program $P$ and a literal $l$ in $P$, the following items are true:

1. $\text{AS}(\text{forget}(P, l)) = \{ X \setminus \{l\} \mid X \in \text{AS}_l(X) \}$.
2. If $X \in \text{AS}_l(X)$ with $l \notin X$, then $X \in \text{AS}(\text{forget}(P, l))$.
3. For any $X \in \text{AS}(P)$ such that $l \in X$, $X \setminus \{l\} \in \text{AS}(\text{forget}(P, l))$.
4. For any $X' \in \text{AS}(\text{forget}(P, l))$, either $X'$ or $X' \cup \{l\}$ is in $\text{AS}(P)$.
5. For any $X \in \text{AS}(P)$, there exists $X' \in \text{AS}(\text{forget}(P, l))$ such that $X' \subseteq X$.

6. If $l$ does not appear in $P$, then $\text{forget}(P, l) = P$.

Let $|=_s$ and $|=_c$ be the skeptical and credulous reasoning defined by the answer sets of a disjunctive program $P$, respectively: for any literal $l$,

$P |=_s l$ iff $l \in S$ for every $S \in \text{AS}(P)$.  
$P |=_c l$ iff $l \in S$ for some $S \in \text{AS}(P)$.

**Proposition 5** Let $l$ be a specified literal in disjunctive program $P$. For any literal $l' \neq l$,

1. $P |=_s l'$ iff $\text{forget}(P, l) |=_s l'$.
2. $P |=_c l'$ only if $\text{forget}(P, l) |=_c l'$.

This proposition says that, if $l$ is ignored, $\text{forget}(P, l)$ is equivalent to $P$ under skeptical reasoning, but weaker under credulous reasoning (i.e., inferences are lost).

Similar to the case of normal programs, the above definitions of forgetting about a literal $l$ can be extended to forgetting about a set $F$ of literals. Specifically, we can similarly define $X_1 \subseteq F \ X_2$, $X_1 \sim_l X_2$ and $F$-answer sets of a disjunctive program. The properties of forgetting about a single literal can also be generalized to the case of forgetting about a set. Moreover, the result of forgetting about a set $F$ can be obtained one by one forgetting each literal in $F$.

**Proposition 6** Let $P$ be a consistent disjunctive program and $F = \{l_1, \ldots, l_m\}$ be a set of literals. Then

$\text{forget}(P, F) \equiv \text{forget}(\text{forget}(\text{forget}(P, l_1), l_2), \ldots), l_m)$.  

We remark that for removing a proposition $p$ entirely from a program $P$, it is suggestive to remove both the literals $p$ and $\neg p$ in $P$ (i.e., all positive and negative information about $p$). This can be easily accomplished by $\text{forget}(P, \{p, \neg p\})$.

Let $\text{lcomp}(P)$ be Clark’s completion plus the loop formulas for an ordinary disjunctive program $P$ (Lee & Lifschitz 2003; Lin & Zhao 2004). Then $X$ is an answer set of $P$ iff $X$ is a model of $\text{lcomp}(P)$.

Now we have two kinds of operators $\text{forget}(\cdot)$ and $\text{lcomp}(\cdot)$. Thus for a disjunctive program and an atom $p$, we have two classical logical theories $\text{lcomp}(\text{forget}(P, p))$ and $\text{forget}(\text{lcomp}(P), p)$ on the signature $B_P \setminus \{p\}$. It is natural to ask what the relationship between these two theories is. Intuitively, the models of the first theory are all minimal models while the models of the second theory may not be minimal. Let $P = \{ p \leftarrow \neg q, q \leftarrow \neg p \}$. Then $\text{forget}(\text{forget}(P, p)) = \{ \neg q \}$ and $\text{forget}(\text{lcomp}(P), p) = \{ T \leftarrow \neg q \lor F \leftarrow \neg q \} \equiv T$, which has two models $\{ q \}$ and $\emptyset$.

However, we have the following result.

**Theorem 1** Let $P$ be a logic program without strong negation and $p$ an atom in $P$. Then $X$ is an answer set of $\text{forget}(P, p)$ if and only if $X$ is a minimal model of the result of classical forgetting $\text{lcomp}(P, p)$. That is,

$\text{AS}(\text{forget}(P, p)) = \text{MMOD}(\text{forget}(\text{lcomp}(P), p))$

Here $\text{MMOD}(T)$ denotes the set of all minimal models of a theory $T$ in classical logic.

\footnotetext[2]{Thanks to Esra Erdem and Paolo Ferraris for pointing this out to us.}
Thus forget($P, p$) can be characterized by forgetting in classical logic. Notice that it would not make much sense if we replace lcomp($P$) with a classical theory which is not equivalent to lcomp($P$) in Theorem 1. In this sense, the notion of forgetting for answer set programming is unique.

We use forget$_{min}(T, p)$ to denote a set of classical formulas whose models are the minimal models of the classical forgetting forget($T, p$). Then the conclusion of Theorem 1 is reformulated as lcomp(forget($P, p$)) $\equiv$ forget$_{min}$(lcomp($P$), $p$).

The result is a nice property, since it means that one can “bypass” the use of an LP engine entirely, and represent also the answer sets of forget($P, p$) in terms of a circumscription of classical forgetting, applied to lcomp($P$).

**Theorem 2** Let $P$ be a logic program without strong negation and $p$ an atom in $P$. Then $S'$ is an answer set of forget($P, p$) if and only if either $S = S'$ or $S = S'$ $\cup \{ p \}$ is a model of Circ($B_P \ \{ p \}, \{ p \},$ lcomp($P$)).

**Computation of Forgetting**

As we have noted, forget($P, l$) exists for any consistent disjunctive program $P$ and literal $l$. In this section, we discuss some issues on computing the result of forgetting.

**Naive Algorithm**

By Definition 2, we can easily obtain a naive algorithm for computing forget($P, l$) using some ASP solvers for DLP, like DLV (Leone et al. 2004) or GrN (Jahangiri et al. 2000).

**Algorithm 1 (Computing a result of forgetting)**

**Input:** disjunctive program $P$ and a literal $l$ in $P$.

**Procedure:**

1. Using DLV compute AS($P$);
2. Remove the literal $l$ from every element of AS($P$) and denote the resulting collection as $A'$;
3. Obtain $A''$ by removing non-minimal elements from $A'$;
4. Construct $P'$ whose answer sets are exactly $A''$: Let $A'' = \{ A_1, ... , A_m \}$ and for each $A_i$, $P_i = \{ l' \leftarrow not A_i | l' \in A_i \}$. $P' = \cup_{i=1}^{m} P_i$. Here $A_i = B_P \ \{ A_i \}$.
5. Output $P'$ as forget($P, l$).

This algorithm is complete w.r.t. the semantic forgetting defined in Definition 2.

**Theorem 3** For any consistent disjunctive program $P$ and a literal $l$, Algorithm 1 always outputs forget($P, l$).

**Basic Program Transformations**

In this subsection, we develop an algorithm for computing the result of forgetting in $P$ using program transformations and other modifications. Here we use the set $T_{WFS}$ of program transformations investigated in (Brass & Dix 1999; Wang & Zhou 2005). In our algorithm, an input program $P$ is first translated into a negative program and the result of forgetting is represented as a nested program (under the minimal answer sets defined by Lifschitz et al. (1999)).

**Elimination of Tautologies:** $P'$ is obtained from $P$ by the elimination of tautologies if there is a rule $r$: head($r$) $\leftarrow$ body$^+$($r$), not body$^-$($r$) in $P$ such that head($r$) $\cap$ body$^+$($r$) $\neq \emptyset$ and $P' = P \ \{ r \}$.

**Elimination of Head Redundancy**

$P'$ is obtained from $P$ by the elimination of head redundancy if there is a rule $r$ in $P$ such that an atom $a$ is in both head($r$) and body$^-$($r$) and $P' = P \ \{ r \} \cup \{ head(r) - a \leftarrow body(r) \}$.

The above two transformations guarantee that those rules whose head and body have common literals are removed.

**Positive Reduction:** $P'$ is obtained from $P$ by the positive reduction if there is a rule $r$: head($r$) $\leftarrow$ body$^+$($r$), not body$^-$($r$) in $P$ and $c \in$ body$^-$($r$) such that $c \notin$ head($P$) and $P'$ is obtained from $P$ by removing not $c$ from $r$. That is, $P' = P \ \{ r \} \cup \{ head(r) \leftarrow body^+(r), not (body^-(r) \ \{ c \}) \}$.

**Negative Reduction:** $P'$ is obtained from $P$ by negative reduction if there are two rules $r$: head($r$) $\leftarrow$ body$^+$($r$), not body$^-$($r$) and $r'$: head($r'$) $\leftarrow$ in $P$ such that head($r'$) $\subseteq$ body$^-$($r$) and $P' = P \ \{ r \}$.

**Definition 3** Let $r$ and $r'$ be two rules. We say that $r'$ is an s-implication of $r$ if $r' \neq r$ and at least one of the following two conditions is satisfied:

1. $r'$ is an implication of $r$: head($r$) $\subseteq$ head($r'$), body($r$) $\subseteq$ body($r'$) and at least one inclusion is proper; or
2. $r$ can be obtained by changing some negative body literals of $r'$ into head atoms and removing some head atoms and body literals from $r'$ if necessary.

**Elimination of s-Implications:** $P_2$ is obtained from $P_1$ by elimination of s-implications if there are two distinct rules $r$ and $r'$ of $P_1$ such that $r'$ is an s-implication of $r$ and $P_2 = P_1 \ \{ r \}$.

**Unfolding:** $P'$ is obtained from $P$ by unfolding if there is a rule $r$ such that $P' = P \ \{ r \} \cup \{ head(r) \leftarrow (body^+(r) \ \{ b \}) \leftarrow (\exists r' \in P s.t. b \in head(r')) \}$.

Here head($r'$) $\leftarrow b$ is the disjunction obtained from head($r'$) by removing $b$.

Since an implication is always an s-implication, the following result is a direct corollary of Theorem 4.1 in (Brass & Dix 1999).

**Lemma 1** Each disjunctive program $P$ can be equivalently transformed into a negative program $N$ via the program transformations in $T_{WFS}$ such that on no rule $r$ in $N$, a literal appears in both the head and the body of $r$.

**Transformation-Based Algorithm**

**Algorithm 2 (Computing a result of forgetting)**

**Input:** disjunctive program $P$ and a literal $l$ in $P$.

**Procedure:**

1. Fully apply the program transformations in $T_{WFS}$ on program $P$ and then obtain a negative program $N_0$.
2. Separate $l$ from head disjunction via semi-shifting: For each (negative) rule $r \in N_0$ such that head($r$) = $l \lor A$
and $A$ is a non-empty disjunction, it is replaced by two rules:
\[ l \leftarrow \not A, \text{body}(r) \text{ and } A \leftarrow \not l, \text{body}(r), \]
where $not A$ is the conjunction of all $not l'$ with $l'$ in $A$. The resulting


Step 3. Suppose that $N$ has $n$ rules with head $l$:
\[ r_j \leftarrow not l_{j1}, \ldots, not l_{jm}, \]
where $n \geq 0$, $j = 1, \ldots, n$ and $m_j \geq 0$ for all $j$.

If $n = 0$, then let $Q$ denote the program obtained from $N$
by removing all appearances of $not l$.

If $n = 1$ and $m_1 = 0$, then $l \leftarrow$ is the only rule in $N$
having head $l$. In this case, remove every rule in $N$ whose
body contains $not l$. Let $Q$ be the resulting program.

For $n \geq 1$ and $m_1 > 0$, let $D_1, \ldots, D_r$ be all possible
conjunctions (not not $l_{1k_1}, \ldots, not not l_{nk_n}$) where $0 \leq k_1 \leq m_1, \ldots, 0 \leq k_n \leq m_n$. Replace in $N$ each occurrence
of $not l$ in $N$ by all possible $D_i$. Let $Q$ be the result.

Step 4. Remove all rules with head $l$ from $Q$ and output the resulting program $N'$.

Some remarks: (1) This is only a general algorithm. Some
program transformations could be omitted for some special
programs and various heuristics could also be employed to
make the algorithm more efficient; (2) In this process, a result
of forgetting is represented by a logic program allowing
nested negation as failure. This form seems more intuitive
than using ordinary logic programs; (3) In the construction
of $D_i$, not not $l_{ij}$ cannot be replaced with $l_{ij}$ (even for
a normal logic program). As one can see, if they are re-
placed, the resulting program represents only a subset of
$AS_l(P)$ (see Example 2). This also implies that Algorithm 1
in (Wang, Sattar, & Su 2005) is incomplete in general.
(4) Algorithm 2 above essentially improves the correspon-
ding algorithm (Algorithm 1) in (Wang, Sattar, & Su 2005)
at least in two ways: (i) our algorithm works for a more expres-
sive class of programs (i.e. disjunctive programs) and (ii) the
next result shows that our algorithm is complete under the
minimal answer set semantics of nested logic programs.

**Theorem 4** Let $P$ be a consistent disjunctive program and $l$ a literal. Then $X$ is an answer set of forget($P, l$) iff $X$ is a minimal answer set of $N'$.

**Example 2** Consider $P_4 = \{ c \leftarrow not q, \ p \leftarrow not q, \ q \leftarrow not p \}$. Then, by Algorithm 2, forget($P_4, p$) is the
nested program $\{ c \leftarrow not q, \ q \leftarrow not q \}$, whose
minimal answer sets are exactly the same as the answer sets of forget($P_4, p$). Note that Algorithm 1 in (Wang, Sattar, & Su 2005)
outputs a program $N'$ = $\{ c \leftarrow not q, \ q \leftarrow q \}$
which has a unique answer set $\{ c \}$. However, forget($P_4, p$) has two answer sets $\{ c \}$ and $\{ q \}$. This implies that the algo-

The above algorithm is worst case exponential, and might
also output an exponentially large program. As follows from
complexity considerations, there is no program $P'$ that
represents the result of forgetting which can be constructed in
polynomial time, even if auxiliary literals might be used
which are projected from the answer sets of $P'$. This is a conse-
quence of the complexity results below. However, the
number of rules containing $l$ may not be very large and some
conjunctions $D_i$ may be omitted because of redundacy.

**Resolving Conflicts in Multi-Agent Systems**

In this section, we present a general framework for resolving
conflicts in multi-agents systems, which is inspired from the
preference recovery problem (Lang & Marquis 2002). Suppose
that there are $n$ agents who may have different preferences
on the same issue. In many cases, these preferences (or
constraints) have conflicts and thus cannot be satisfied at the
same time. It is an important issue in constraint reasoning
to find an intuitive criteria so that preferences with higher
priorities are satisfied. Consider the following example.

**Example 3** (Lang & Marquis 2002) Suppose that a group of
four residents in a complex tries to reach an agreement on
building a swimming pool and/or a tennis court. The prefer-
ences and constraints are as follows.

1. Building a tennis court or a swimming pool costs each
one unit of money.
2. A swimming pool can be either red or blue.
3. The first resident would not like to spend more than one
money unit, and prefers a red swimming pool.
4. The second resident would like to build at least one of
tennis court and swimming pool. If a swimming pool is
built, he would prefer a blue one.
5. The third resident would prefer a swimming pool but ei-
ther colour is fine with him.
6. The fourth resident would like both tennis court and swim-
ming pool to be built. He does not care about the colour of
the pool.

Obviously, the preferences of the group are jointly inconsis-
tent and thus it is impossible to satisfy them at the same time.

In the following, we will show how to resolve this kind of
preference conflicts using the theory of forgetting.

An $n$-agent system $S$ is an $n$-tuple ($P_1, P_2, \ldots, P_n$) of
disjunctive programs, $n > 0$, where $P_i$ represents agent $i$’s
knowledge (including preferences, constraints).

As shown in Example 3, $P_1 \cup P_2 \cup \cdots \cup P_n$ may be
inconsistent. The basic idea in our approach is to forget some
literals for each agent so that conflicts can be resolved.

**Definition 4** Let $S = (P_1, P_2, \ldots, P_n)$ be an $n$-agent
system. A compromise of $S$ is a sequence $C = (F_1, F_2, \ldots, F_n)$
where each $F_i$ is a set of literals. An agreement
of $S$ on $C$ is an answer set of the disjunctive pro-
gram forget($S, C$) where forget($S, C$) = forget($F_1, F_2$)$\cup$
forget($P_2, F_2$) $\cup \cdots \cup$ forget($P_n, F_n$).

For a specific application, we may need to impose certain
conditions on each $F_i$.

**Example 4** (Example 3 continued) The scenario can be en-
coded as a collection of five disjunctive programs ($P_0$ stands
for general constraints): $S = (P_0, P_1, P_2, P_3, P_4)$ where
\[
P_0 = \{ \text{red} \lor \text{blue} \leftarrow s, \ \text{not red, blue} \}.
\]
\[
\begin{align*}
  u_1 & \leftarrow not s, t. \quad u_1 \leftarrow s, not t. \\
  u_2 & \leftarrow s, t. \quad u_0 \leftarrow not s, not t; \\
\end{align*}
\]
\[
P_1 = \{ u_0 \lor u_1 \leftarrow, \ \text{red} \leftarrow s \};
\]
\[
P_2 = \{ s \lor t \leftarrow, \ \text{blue} \leftarrow s \};
\]
\[
P_3 = \{ s \leftarrow \}; \text{ and } P_4 = \{ s \leftarrow, \ t \leftarrow \}.
\]
Since this knowledge base is jointly inconsistent, each resident may have to weaken some of her preferences so that an agreement is reached. Some possible compromises are:

1. $C_1 = (\emptyset, F, F, F)$ where $F = \{s, \text{blue}, \text{red}\}$: Every resident would be willing to weaken her preferences on the swimming pool and its colour. Since $\text{forget}(S, C_1) = P_0 \cup \{u_0 \lor u_1 \leftarrow . \ t \leftarrow \}, S$ has a unique agreement $\{t, u_1\}$ on $C_1$. That is, only a tennis court is built.

2. $C_2 = (\emptyset, \{\text{blue}, \text{red}\}, \emptyset, \emptyset, \{t\})$: The first resident can weaken her preference on pool colour and the fourth resident can weaken her preference on tennis court. Since $\text{forget}(S, C_2) = P_0 \cup P_2 \cup P_3 \cup \{u_0 \lor u_1 \leftarrow . \ s \lor t \leftarrow . \ s \leftarrow \}, S$ has a unique agreement $\{s, \text{blue}, u_1\}$ on $C_2$. That is, only a swimming pool will be built and its colour is blue.

As shown in the example, different compromises lead to different results. We do not consider the issue of how to reach compromises here, which is left for future work.

### Computational Complexity

In this section we address the computational complexity of forgetting for different classes of logic programs. Our results show that for general disjunctive programs, (1) the model checking of forgetting is $\Pi_2^P$-complete; (2) the credulous reasoning of forgetting is $\Sigma_3^P$-complete. However, for normal programs or negative disjunctive programs, the complexity levels are lower: (1) the model checking of forgetting is co-NP-complete; (2) the credulous reasoning of forgetting is $\Sigma_2^P$-complete.

<table>
<thead>
<tr>
<th>model checking</th>
<th>disjunctive</th>
<th>negative</th>
<th>normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\models_c$</td>
<td>$\Pi_2^P$</td>
<td>co-NP</td>
<td>co-NP</td>
</tr>
<tr>
<td></td>
<td>$\Sigma_3^P$</td>
<td>$\Sigma_2^P$</td>
<td>$\Sigma_2^P$</td>
</tr>
</tbody>
</table>

**Theorem 5** Given a disjunctive program $P$, a literal $l$, and set of literals $X$, deciding whether $X$ is an $l$-answer set of $P$ is $\Pi_2^P$-complete.

Intuitively, in order to show that $X$ is an $l$-answer set, we have to witness that $X$ is an answer set (which is co-NP-complete to test), and that there is no answer set $X'$ of $P$ such that $X' \subseteq X$. Any $X'$ disproving this can be guessed and checked using an NP-oracle in polynomial time. Thus, $l$-answer set checking is in $\Pi_2^P$, as stated in Theorem 5.

The construction in the proof of Theorem 5 can be extended to show $\Sigma_2^P$-hardness of credulous inference.

**Theorem 6** Given a disjunctive program $P$ and literals $l$ and $l'$, deciding whether $\text{forget}(P, l) \models_c l'$ is $\Sigma_2^P$-complete.

In Theorem 6 a suitable $l$-answer set containing $l'$ can be guessed and checked, by Theorem 5 using $\Sigma_2^P$-oracle. Hence, credulous inference $\text{forget}(P, l) \models_c l'$ is in $\Sigma_2^P$. The matching lower bounds, $\Pi_2^P$ resp. $\Sigma_2^P$-hardness can be shown by encodings of suitable quantified Boolean Formulas (QBFs).

In Theorems 5 and 6, the complexity is coNP- and $\Sigma_2^P$-complete, respectively, if $P$ is either negative or normal.

**Theorem 7** Given a negative or normal program $N$, a literal $l$, and set of literals $X$. Then

1. Deciding $X \in \mathcal{AS}_I(N)$ is co-NP-complete.

2. Deciding whether $\text{forget}(N, l) \models_c l'$ is $\Sigma_2^P$-complete.

### Conclusion

We have proposed a theory of forgetting literals in disjunctive programs. Although our approach is purely declarative, we have proved that it is coupled by a syntactic counterpart based on program transformations. The properties of forgetting show that our approach captures the classical notion of forgetting. As we have explained before, the approach in this paper naturally generalizes the forgetting for normal programs investigated in (Wang, Sattar, & Su 2005). As an application of forgetting, we have also presented a fairly general framework for resolving conflicts in disjunctive logic programming. In particular, this framework provides an elegant solution to the preference recovery problem.

### References


