LPForget: A System of Forgetting in Answer Set Programming

Fu-Leung Cheng[1], Thomas Eiter[2], Nathan Robinson[1], Abdul Sattar[1], and Kewen Wang[1]

[1] Griffith University, Australia,
{fu.cheng,nathan.robinson}@student.gu.edu.au,{a.sattar,k.wang}@gu.edu.au
[2] Technische Universität Wien, Austria, eiter@kr.tuwien.ac.at

Abstract. A novel declarative approach of forgetting in answer set programming (ASP) has been proposed recently. In this paper we report a system prototype of forgetting in ASP, called LPForget. It consists of two modules: (1) Forgetting: computing the result of forgetting about certain literals in logic program under the answer set semantics; (2) CRS: application of forgetting in resolving conflict (or preference recovery) in multi-agent systems. The motivation for developing LPForget is to provide reasoning support for managing ontologies in rule-based ontology language as well as using the system for studying theoretic properties of forgetting.

Key words: Nonmonotonic logic programming; answer sets; forgetting

1 Introduction

Informally, some literals/concepts in a knowledge base may be redundant and can be removed without affecting the reasoning of other literals. This is the intuition behind the notion of *variable forgetting* or *variable elimination*. This technique can be applied in query answering, planning, decision-making, reasoning about actions, knowledge update and revision. The approach to forgetting in answer set programming (ASP) [**?**,**?**] are among the first attempts to define a *declarative* notion of forgetting in nonmonotonic reasoning and logic programming. Forgetting in ASP naturally generalizes and captures the classical forgetting [**?**,**?**].

In this paper we report an implementation prototype of forgetting in ASP, called LPForget. It consists of two modules: (1) Forgetting: the core of the system is computing the result of forgetting about certain literals in logic program under the answer set semantics; (2) CRS: application of forgetting in conflict resolving (or preference recovery) in multi-agent systems.

The motivation for developing LPForget is to provide reasoning support for managing ontologies in rule-based ontology languages as well as using the system for studying theoretic properties of forgetting. Forgetting has an interesting application in editing, merging, and aligning ontologies in the Semantic Web. Consider a scenario in managing ontologies: Suppose we are compiling an ontology called "PetOwner", in which some pet owners, pets and their relations

are specified. We searched the Web and found an ontology "Animal". However, this ontology is large. In particular, it contains a lot of animals that are not considered as pets normally such as tigers and lions. Thus, it is not practical to reason with or process the whole ontology. An interesting approach is to use the technique of forgetting and thus eliminate those animals that we do not want to keep from "Animal". As a result, a (smaller) sub-ontology of "Animal" is obtained, which keeps only the useful and desirable information in "Animal". Our approach is based on hex-programs whose external atoms are ontologies in OWL (see [**?**] for details).

2 Forgetting in Answer Set Programming

In this section we briefly review some basics of *answer set programming* [**?**] and *semantic forgetting* introduced in [**?**].

2.1 Answer sets of disjunctive programs

A *disjunctive program* is a finite set of rules of the form

$$a_1 \vee \cdots \vee a_s \leftarrow b_1, \ldots, b_m, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_n, \tag{1}$$

$s, m, n \geq 0$, 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 $\mathit{not}\ l$ where not is for the negation as failure and l is a (ordinary) literal. For an atom p, p and $\neg p$ are called *complementary*. For any literal l, its complementary literal is denoted $\tilde{l}$. 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), $\mathit{head}(r) = a_1 \vee \cdots \vee a_s$ and $\mathit{body}(r) = \mathit{body}^+(r) \cup \mathit{not}\ \mathit{body}^-(r)$ where $\mathit{body}^+(r) = \{b_1, \ldots, b_m\}$, $\mathit{body}^-(r) = \{c_1, \ldots, c_n\}$, and $\mathit{not}\ \mathit{body}^-(r) = \{\mathit{not}\ q \mid q \in \mathit{body}^-(r)\}$.

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 *false*.

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

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 $P^X = \{\mathit{head}(r) \leftarrow \mathit{body}^+(r) \mid r \in P, \mathit{body}^-(r) \cap X = \emptyset\}$. An interpretation X is an *answer set* of P if X is a minimal model of P^X (by treating each literal as a new atom). $\mathcal{AS}(P)$ 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 $\mathcal{AS}(P) = \mathcal{AS}(P')$.

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

2.2 Semantic forgetting

A set X' is an *l-subset* of another 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 $X' \setminus \{l\} \subset X \setminus \{l\}$. 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 $\mathcal{S}$ of sets of literals, $X \in \mathcal{S}$ is* l-minimal *if there is no $X' \in \mathcal{S}$ such that $X' \subset_l X$. $\min_l(\mathcal{S})$ denotes the collection of all l-minimal elements in $\mathcal{S}$.*
2. *An answer set X of disjunctive program P is an* l-answer set *if X is l-minimal in $\mathcal{AS}(P)$.*

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. *$B_{P'} \subseteq B_P \setminus \{l\}$ where B_Q denotes the set of (classical) literals occurring in Q.*
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. However, different such programs P' might exist. It follows from the above definition that they are all equivalent under the answer set semantics. Thus, we use $\mathsf{forget}(P, l)$ to denote a possible result of forgetting about l in P.

For further discussions about forgetting in disjunctive logic programming, the reader is referred to [**?**].

3 System Structure and Applying LPForget

LPForget has implemented two basic algorithms (Algorithm 1 and 2) for computing the result of forgetting introduced in [**?**]. Two variants of these algorithms (Algorithm 3 and 4) [**?**] are also implemented in the system. Algorithm 3 is aimed for efficient implementation for forgetting while Algorithm 4 is designed for dealing with open-world reasoning tasks [**?**]. Besides these algorithms for

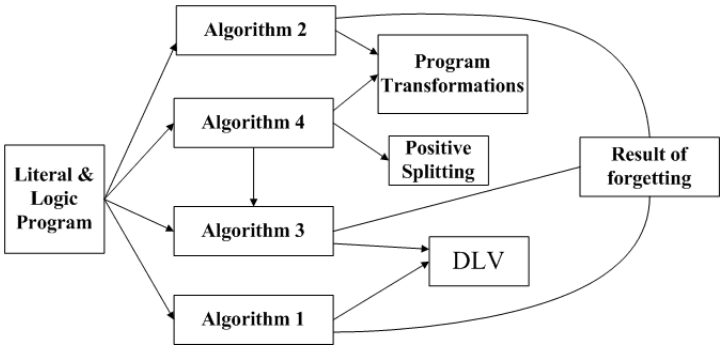

Fig. 1. System Structure of Forgetting

computing forgetting, LPForget contains an application of forgetting for conflict resolution in the setting of multi-agents.

The system LPForget is implemented in Java and its structure is shown in Figure **??**. It currently works on MS Windows since the version of DLV we adopted is of the Windows version. Obviously, the system can be easily adapted to Linux or Unix. The system can be downloaded from: `http://www.cit.gu.edu.au/~kewen/LPForget/`.

The main system can be executed by running the program LPForget.jar. Then one can choose to use the core module Forgetting or the application CRS. If the module Forgetting is chosen, then the users will be prompted to enter a disjunctive program, the literal to be forgotten, and the choice of algorithms. The result of forgetting can be obtained by clicking on the button "Compute". Notice that different algorithms may output syntactically different results of forgetting. However, the results are semantically equivalent.

The syntax of the input logic programs of the system coincides with that of DLV [**?**], as shown by the following example program P:

$$\begin{aligned} red &:- pool, not\ blue. \\ blue &:- pool, not\ red. \\ pool &:- not\ tennisCourt. \end{aligned}$$

This program represents some resident's requirements for a proposal of building a swimming pool and/or tennis court in a community complex: (1) if a swimming pool is built, then its colour can be either red or blue; (2) if a tennis court is not built, then a swimming pool should be built.

Some other residents may also have their requirements on the building proposal. Due to preferences conflict between different residents, the resident may have to give up some of his preferences and, for instance, say "it does not matter for me if the swimming pool is blue or not". Then we can "forget" about the colour "blue" from the above program and then get a new program $\mathsf{forget}(P, blue) = \{pool :- .\}$, which means that only a swimming pool is built.

More examples can be found in the system website (Examples folder).

Acknowledgments. The authors would like to thank Rodney Topor for his helpful comments and contribution to this paper. This work was partially supported by the Australia Research Council (ARC) Discovery Projects 0666107, 0666540, the Austrian Science Funds (FWF) projects 17212, 18019, the European Commission project REWERSE (IST-2003-506779) and NICTA Queensland.

References

1. F. Cheng, T. Eiter, N. Robinson, A. Sattar, and K. Wang. LPForget: a system of forgetting in answer set programming. Technical report, 2006, available at `http://www.cit.gu.edu.au/~kewen/Papers/LPForget_long.pdf`
2. T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, and K. Wang. Forgetting in managing rules and ontologies. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI-06)*, 2006 (accepted for publication).
3. T. Eiter, and K. Wang. Forgetting and conflict resolving in disjunctive logic programming. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-2006)*, pages 238-243, 2006.
4. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proceedings of the International Conference on Logic Programming*, pages 579-597, 1990.
5. J.Lang, P.Liberatore, and P.Marquis. Propositional independence: Formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391-443, 2003.
6. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499-562, 2006.
7. F. Lin and R. Reiter. Forget it. In *Proceedings of the AAAI Fall Symposium on Relevance*, pages 154-159. New Orleans (LA), 1994.
8. K.Wang, A.Sattar and K.Su. A theory of forgetting in logic programming. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-2005)*, pages 682-687, 2006.