Forgetting under the Well-Founded Semantics

José Júlio Alferes[1], Matthias Knorr[1], and Kewen Wang[2]

[1] CENTRIA & Departamento de Informática, Faculdade Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
[2] School of Information and Communication Technology, Griffith University, Brisbane QLD 4111, Australia

Abstract. In this paper, we develop a notion of forgetting for normal logic programs under the well-founded semantics. We show that a number of desirable properties are satisfied by our approach. Three different algorithms are presented that maintain the computational complexity of the well-founded semantics, while partly keeping its syntactical structure.

1 Introduction

Forgetting has drawn considerable attention in knowledge representation and reasoning. This is witnessed by the fact that forgetting has been introduced in many monotonic and nonmonotonic logics [1,5,9,10,11,12,16,18,19], and in particular, in logic programming [6,15,17].

A potential drawback, common to all these three approaches, is the computational (data) complexity of the answer set semantics, which is **coNP**, while the other common semantics for logic programs, the well-founded semantics (WFS), is in **P**, which may be preferable in applications with huge amounts of data. However, to the best of our knowledge, forgetting under the well-founded semantics has not been considered so far. Therefore, in this paper, we develop a notion of forgetting for normal logic programs under the well-founded semantics. We show that forgetting under the well-founded semantics satisfies the properties in [6]. In particular, our approach approximates semantic forgetting of [6] for normal logic programs under answer set semantics as well as forgetting in classical logic, in the sense that whatever is derivable from a logic program under the well-founded semantics after applying our notion of forgetting, is also derivable in each answer set and classical model after applying semantic and classical forgetting to the logic program and its classical representation, respectively. We also present three different algorithms that maintain the favorable computational complexity of the well-founded semantics when compared to computing answer sets.

2 Preliminaries

A *normal logic program* P, or simply *logic program*, is a finite set of rules r of the form $h \leftarrow a_1, \ldots, a_n, \mathit{not}\, b_1, \ldots, \mathit{not}\, b_m$ where h, a_i, and b_j, with $1 \leq i \leq n$ and $1 \leq j \leq m$, are all propositional atoms over a given alphabet Σ.

Given a rule r, we distinguish the *head* of r as $head(r) = h$, and the *body* of r, $body(r) = body^+(r) \cup not\, body^-(r)$, where $body^+(r) = \{a_1, \dots, a_n\}$, $body^-(r) = \{b_1, \dots, b_m\}$ and, for a set S of atoms, $not\, S = \{not\, q \mid q \in S\}$. Rule r is *positive* if $body^-(r) = \emptyset$, *negative* if $body^+(r) = \emptyset$, and a *fact* if $body(r) = \emptyset$.

Given a logic program P, B_P is the set of all atoms appearing in P, and $Lit_P = B_P \cup not\, B_P$. Also, $heads(P)$ denotes the set $\{p \mid p = head(r) \wedge r \in P\}$.

A *three-valued interpretation* $I = I^+ \cup not\, I^-$ with $I^+, I^- \subseteq B_P$ and $I^+ \cap I^- = \emptyset$. Informally, I^+ and I^- contain the atoms that are true and false in I, respectively. Any atom appearing neither in I^+ nor in I^- is undefined.

We recall the definition of the well-founded semantics based on the alternating fixpoint [7]. Given a logic program P and $S \subseteq B_P$, we define $\Gamma_P(S) = least(P^S)$ where $P^S = \{head(r) \leftarrow body^+(r) \mid r \in P, body^-(r) \cap S = \emptyset\}$ and $least(P^S)$ is the least model of the positive logic program P^S. The square of Γ_P, Γ_P^2, is a monotonic operator and thus has both a least fixpoint, $lfp(\Gamma_P^2)$, and a greatest fixpoint $gfp(\Gamma_P^2)$. We obtain the well-founded model $WFM(P)$ of a normal logic program P as $WFM(P) = lfp(\Gamma_P^2) \cup not\,(B_P \setminus gfp(\Gamma_P^2))$.

Two programs P and P' are *equivalent (under WFS)*, denoted by $P \equiv_{wf} P'$, iff $WFM(P) = WFM(P')$. Finally, the inference relation under the WFS is defined for any literal $q \in Lit(P)$ as follows: $P \models_{wf} q$ iff $q \in WFM(P)$.

3 Forgetting under the Well-Founded Semantics

When defining forgetting of an atom p in a given logic program P, we want to obtain a new logic program P' such that it does not contain any occurrence of p or its default negation $not\, p$. Additionally, we want to ensure that only the derivation for p (and $not\, p$) is affected, keeping P' and P equivalent w.r.t. all derivable literals excluding p (and $not\, p$). We want to achieve this based on the semantics rather than the syntax and ground it in forgetting in classical logic.

So, we semantically define the result of forgetting under the WFS by determining the well-founded model, and then providing a logic program that excludes p syntactically, and whose well-founded model excludes (only) p semantically.

Definition 1. *Let P be a logic program and p an atom. The result of* forgetting about p in P*, denoted* $\mathsf{forget}(P, p)$*, is a logic program P' such that the following two conditions are satisfied:*

(1) $B_{P'} \subseteq B_P \setminus \{p\}$, i.e., p does not occur in P', and
(2) $WFM(P') = WFM(P) \setminus (\{p\} \cup \{not\, p\})$

This definition obviously does not introduce new symbols (cf. (F2) in [6]). In the rest of this section, we assume P, P' logic programs and p an atom, and show a number of desirable properties. The first one corresponds to (F3) in [6].

Proposition 2. *For any $l \in Lit \setminus (\{p\} \cup \{not\, p\})$, $\mathsf{forget}(P, p) \models_{wf} l$ iff $P \models_{wf} l$.*

Our definition of forgetting also implies that there are syntactically different logic programs that correspond to $\mathsf{forget}(P, p)$. However, as we show next, all

Algorithm $\mathsf{forget}_1(P, p)$

Input: Normal logic program P and an atom p in P.

Output: A normal logic program P' representing $\mathsf{forget}(P, p)$.

Method:

Step 1. Compute the well-founded model $WFM(P)$ of P.
Step 2. Let M be the three-valued interpretation obtained from $WFM(P)$ by removing p and $not\, p$. Construct a new logic program with $B_{P'} = B_P \setminus \{p\}$ whose well-founded model is exactly M:
$P' = \{a \leftarrow . \mid a \in M^+\} \cup \{a \leftarrow not\, a. \mid a \in B_{P'} \setminus (M^+ \cup M^-)\}$.
Step 3. Output P' as $\mathsf{forget}(P, p)$.

Fig. 1. Algorithm $\mathsf{forget}_1(P, p)$

results of forgetting about p in P are equivalent w.r.t. the well-founded semantics. So, we simply use $\mathsf{forget}(P, p)$ as a generic notation representing any syntactic variant of all semantically equivalent results of forgetting about p in P.

Proposition 3. *If P' and P'' are two results of* $\mathsf{forget}(P, p)$, *then* $P' \equiv_{wf} P''$.

Forgetting also preserves equivalence on $\equiv_{wf}$ (cf. (F4) in [6]).

Proposition 4. *If $P \equiv_{wf} P'$, then* $\mathsf{forget}(P, p) \equiv_{wf} \mathsf{forget}(P', p)$.

However, our definition of forgetting preserves neither strong nor uniform equivalence. Intuitively, the reason is that Def. 1 only specifies the change on the semantics but not the precise syntactic form of the resulting program.

We may also generalize the definition of forgetting to a set of atoms S in the obvious way and show that the elements of the set can be forgotten one-by-one.

Proposition 5. *Let P be a logic program and $S = \{q_1, \ldots, q_n\}$ a set of atoms. Then* $\mathsf{forget}(P, S) \equiv_{wf} \mathsf{forget}(\mathsf{forget}(P, q_1), \ldots, q_n)$.

We show that our notion of forgetting is faithful w.r.t. semantic forgetting in ASP [6] as follows, which also links to classical forgetting (cf. (F1) in [6]).

Theorem 6. *Let P be a logic program and p, q atoms.*

1. *If* $q \in WFM(\mathsf{forget}(P, p))$, *then* $q \in M$ *for all* $M \in \mathcal{AS}(\mathsf{forget}_{ASP}(P, p))$.
2. *If* $not\, q \in WFM(\mathsf{forget}(P, p))$, *then* $q \notin M$ *for all* $M \in \mathcal{AS}(\mathsf{forget}_{ASP}(P, p))$.

4 Computation of Forgetting

4.1 Naïve Semantics-based Algorithm

Def. 1 naturally leads to an algorithm for computing the result of forgetting about p in a given logic program P: compute the well-founded model M of P and construct a logic program from scratch corresponding to $WFM(\mathsf{forget}(P, p))$. This idea is captured in Algorithm $\mathsf{forget}_1(P, p)$ shown in Fig. 1.

Algorithm $\mathsf{forget}_2(P, p)$

Input: Normal logic program P and an atom p in P.

Output: A normal logic program P' representing $\mathsf{forget}(P, p)$.

Method:

Step 1. Query for the truth value of p in $WFM(P)$ of P (e.g., using XSB).
Step 2. Remove all rules whose head is p. Moreover, given the obtained truth value of p in $WFM(P)$, execute one of the three cases:

- **t**: Remove all rules that contain $not\, p$ in the body, and remove p from all the remaining rule bodies.
- **u**: Substitute p and $not\, p$ in each body of a rule r in P by $not\, head(r)$.
- **f**: Remove all rules that contain p in the body, and remove $not\, p$ from all the remaining rule bodies.

Step 3. Output the result P' as $\mathsf{forget}(P, p)$.

Fig. 2. Algorithm $\mathsf{forget}_2(P, p)$

4.2 Query-based Algorithm

Algorithm $\mathsf{forget}_1(P, p)$ has two shortcomings. First, the syntactical structure of the original logic program is completely lost, which is not desirable if the rules are subject to later update or change: the author would be forced to begin from scratch, since the originally intended connections in the rules were lost in the process. Second, the computation is not particularly efficient, e.g., if we consider a huge number of rules from which we want to forget one atom p only.

In the following, we tackle the shortcomings of $\mathsf{forget}_1(P, p)$ based on the fact that the WFS is relevant, in the sense that it allows us to query for one atom in a top-down manner without having to compute the entire model.[3] This means that we only consider a limited number of rules in which the query/goal or one of its subsequent subgoals appear. Once the truth value of p is determined, we only make minimal changes to accommodate the forgetting of p: if p is true (resp. false), then body atoms (resp. entire rules) are removed appropriately; if p is undefined, then all occurrences of p (and $not\, p$) are substituted by the default negation of the rule head, thus ensuring that the rule head will be undefined, unless it is true because of another rule in P whose body is true in $WFM(P)$. The resulting algorithm $\mathsf{forget}_2(P, p)$ is shown in Fig. 2.

4.3 Forgetting as Program Transformations

What if we could actually avoid computing the well-founded-model at all? We investigate how to compute $\mathsf{forget}(P, p)$ using syntactic program transformations instead, thereby handling (F5) and completing the match to the criteria in [6].

[3] See, e.g., XSB (http://xsb.sourceforge.net) for an implementation.

Algorithm $\mathsf{forget}_3(P,p)$

Input: Normal logic program P and an atom p in P.

Output: A normal logic program P' representing $\mathsf{forget}(P,p)$.

Method:

Step 1. Compute $\hat{P}$ by exhaustively applying the transformation rules in $\mapsto_X$ to P.
Step 2. If neither $p \leftarrow . \in \hat{P}$ nor $p \notin heads(\hat{P})$, then substitute p and $not\, p$ in each body of a rule r in $\hat{P}$ by $not\, head(r)$. After that, remove all rules whose head is p.
Step 3. Output the result P' as $\mathsf{forget}(P,p)$.

Fig. 3. Algorithm $\mathsf{forget}_3(P,p)$

The basic idea builds on a set of program transformations $\mapsto_X$ [3], which is a refinement of [2] for the WFS, avoiding the potential exponential size of the resulting program in [2] yielding the program remainder $\hat{P}$. It is shown in [3] that $\mapsto_X$ is always terminating and confluent and that the remainder resulting from applying these syntactic transformations to P relates to the well-founded model $WFM(P)$ in the following way: $p \in WFM(P)$ iff $p \leftarrow . \in \hat{P}$ and $not\, p \in WFM(P)$ iff $p \notin heads(\hat{P})$. We can use this to create the algorithm $\mathsf{forget}_3(P,p)$ shown in Fig. 3 which syntactically computes the result of $\mathsf{forget}(P,p)$.

Theorem 7. *Given logic program P and atom p, $\mathsf{forget}_x(P,p)$, $1 \leq x \leq 3$, computes a correct result of $\mathsf{forget}(P,p)$, terminates, and computing P' is in* $\mathbf{P}$.

5 Conclusions

We have developed a notion of semantic forgetting under the well-founded semantics and presented three different algorithms for computing the result of such forgetting, and in each case the computational complexity is in $\mathbf{P}$.

In terms of future work, we intend to pursue different lines of investigation. First, we may consider a notion of forgetting that also preserves strong equivalence for different programs, similar to [15] for the answer set semantics, possibly based on HT^2 [4] or adapting work on updates using SE-models [13,14]. An important issue then is whether the result is again expressible as a normal logic program. Second, since forgetting has been considered for description logics (DLs), we may also consider forgetting in formalisms that combine DLs and non-monotonic logic programming rules under WFS, such as [8].

Acknowledgments J. Alferes and M. Knorr were partially supported by FCT (Fundação para a Ciência e a Tecnologia) under project "ERRO – Efficient Reasoning with Rules and Ontologies" (PTDC/EIA-CCO/121823/2010), and M. Knorr also by FCT Grant SFRH/BPD/86970/2012. K. Wang was partially supported by Australian Research Council under grants DP110101042 and DP1093652.

References

1. Antoniou, G., Eiter, T., Wang, K.: Forgetting for defeasible logic. In: Bjørner, N., Voronkov, A. (eds.) LPAR. Lecture Notes in Computer Science, vol. 7180, pp. 77–91. Springer (2012)
2. Brass, S., Dix, J.: Semantics of disjunctive logic programs based on partial evaluation. J. Log. Program. 38(3), 167–312 (1999)
3. Brass, S., Dix, J., Freitag, B., Zukowski, U.: Transformation-based bottom-up computation of the well-founded model. TPLP 1(5), 497–538 (2001)
4. Cabalar, P., Odintsov, S.P., Pearce, D.: Logical foundations of well-founded semantics. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) KR. pp. 25–35. AAAI Press (2006)
5. van Ditmarsch, H.P., Herzig, A., Lang, J., Marquis, P.: Introspective forgetting. In: Wobcke, W., Zhang, M. (eds.) Australasian Conference on Artificial Intelligence. Lecture Notes in Computer Science, vol. 5360, pp. 18–29. Springer (2008)
6. Eiter, T., Wang, K.: Semantic forgetting in answer set programming. Artif. Intell. 172(14), 1644–1672 (2008)
7. Gelder, A.V.: The alternating fixpoint of logic programs with negation. J. Comput. Syst. Sci. 47(1), 185–221 (1993)
8. Knorr, M., Alferes, J.J., Hitzler, P.: Local closed world reasoning with description logics under the well-founded semantics. Artif. Intell. 175(9–10), 1528–1554 (2011)
9. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Logic-based ontology comparison and module extraction, with an application to DL-Lite. Artif. Intell. 174(15), 1093–1141 (2010)
10. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: Formula-variable independence and forgetting. J. Artif. Intell. Res. (JAIR) 18, 391–443 (2003)
11. Lin, F., Reiter, R.: Forget it! In: In Proceedings of the AAAI Fall Symposium on Relevance. pp. 154–159 (1994)
12. Lutz, C., Wolter, F.: Foundations for uniform interpolation and forgetting in expressive description logics. In: Walsh, T. (ed.) IJCAI. pp. 989–995. IJCAI/AAAI (2011)
13. Slota, M., Leite, J.: On semantic update operators for answer-set programs. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) ECAI. Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 957–962. IOS Press (2010)
14. Slota, M., Leite, J.: Robust equivalence models for semantic updates of answer-set programs. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) KR. pp. 158–168. AAAI Press (2012)
15. Wang, Y., Zhang, Y., Zhou, Y., Zhang, M.: Forgetting in logic programs under strong equivalence. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) KR. pp. 643–647. AAAI Press (2012)
16. Wang, Z., Wang, K., Topor, R.W., Pan, J.Z.: Forgetting for knowledge bases in DL-Lite. Ann. Math. Artif. Intell. 58(1-2), 117–151 (2010)
17. Zhang, Y., Foo, N.Y., Wang, K.: Solving logic program conflict through strong and weak forgettings. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI. pp. 627–634. Professional Book Center (2005)
18. Zhang, Y., Zhou, Y.: Knowledge forgetting: Properties and applications. Artif. Intell. 173(16-17), 1525–1537 (2009)
19. Zhou, Y., Zhang, Y.: Bounded forgetting. In: Burgard, W., Roth, D. (eds.) AAAI. pp. 280–285. AAAI Press (2011)