Forgetting under the Well-Founded Semantics

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

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

Abstract. Recently, the notion of forgetting has drawn considerable attention in the area of Knowledge Representation and Reasoning. The underlying idea is to discard a part of the knowledge base without affecting the derivations on, e.g., atoms that are not to be forgotten. Several recent papers consider forgetting under answer set semantics, but sometimes the well-founded semantics with data complexity **P** (instead of **coNP**) is preferable in applications. 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, in particular, it approximates forgetting for normal logic programs under answer set semantics as well as forgetting in classical logic. Three different algorithms are presented that maintain the favorable computational complexity of the well-founded semantics when compared to computing answer sets, and whose output is at most of the size of the original program, while partly keeping its syntactical structure.

1 Introduction

In [17], the notion of forgetting has been identified to be necessary in the field of Cognitive Robotics when keeping the knowledge base of an agent up to date. The main idea is to be able to discard information that is no longer of any interest for the current situation of the agent, e.g., certain observations in the past, without affecting any possible future actions or any previously drawn conclusions that may still be of importance. The potential benefit lies in the ability to reduce the space necessary to store the agent's knowledge base and, therefore, also reduce reasoning time for the agent.

Since then, forgetting has drawn considerable attention in the area of Knowledge Representation and Reasoning. This is witnessed by the fact that forgetting has been introduced in propositional logic [15], first-order logic [17,23], in particular description logics [14,20,18], but also in non-classical logics, such as modal logics [7,22], defeasible logic [1], and in Logic Programming [21,9,19].

In the latter case, existing approaches focus on the answer set semantics [12] for logic programs. In [21], the notions of strong and weak forgetting are defined but they are syntax-based, and equivalent programs may have non-equivalent results of forgetting. This is remedied in [9], where semantic forgetting is defined

and a number of properties, that a reasonable theory of forgetting should have, are presented. In [19], the notion of HT-forgetting is introduced based on the Logic of Here-and-There [16], which additionally ensures that strongly equivalent logic programs have strongly equivalent results of forgetting. However, one drawback of that approach is that results of forgetting may not be expressible as logic programs. Another 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, is in **P**, which may be preferable in applications with huge amount 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 [9], namely, given a logic program P and P', the result of forgetting about atom p in P, we have:

(F1) The proposed notion of forgetting should be a "natural" generalization of, and relate to, forgetting in classical logic.
(F2) No new symbols are introduced in P', i.e., the vocabulary stays the same.
(F3) The reasoning under P' is equivalent to the reasoning under P if p is ignored.
(F4) The result of forgetting is not sensitive to syntax in that the results of forgetting about p in semantically equivalent programs should also be semantically equivalent.
(F5) The semantic notion of forgetting is coupled with a syntactic counterpart, i.e., there is effective constructible syntax for representing the result of forgetting.

In particular, our approach approximates semantic forgetting of [9] 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 well-founded semantics after applying our notion of forgetting, also is derivable in each answer set (classical model) after applying semantic forgetting (classical forgetting) to the logic program (its classical representation).

We also present three different algorithms that all maintain the favorable computational complexity of the well-founded semantics when compared to computing answer sets. We study the differences between these algorithms in terms of how much they maintain the structure of the original logic program P and how they affect the size of the resulting program in comparison to the original program P. Unlike [9], where the resulting program may be exponential in the size of the input program, forgetting under well-founded semantics yields in all three algorithms a program which is at most of the size of the original program, which aligns well with one of the intended benefits of forgetting in applications.

The paper is structured as follows. In Sect. 2, we recall preliminary notions on logic programs and the well-founded semantics. We introduce our semantic notion of forgetting under the well-founded semantics in Sect. 3 and discuss its

formal properties. The three algorithms and their comparison are presented in Sect. 4, before we conclude in Sect. 5.[3]

2 Preliminaries

We start by briefly recalling notions and notation of Logic Programming and the well-founded semantics [11] for logic programs.

A *normal logic program* P, or simply *logic program*, is a finite set of rules r of the form

$$h \leftarrow a_1, \ldots, a_n, not\, b_1, \ldots, not\, b_m. \tag{1}$$

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 of the form (1), we distinguish the *head* of r as $head(r) = h$, and the *body* of r, $body(r) = body^+(r) \cup not\, body^-(r)$, where $body^+(r) = \{a_1, \ldots, a_n\}$, $body^-(r) = \{b_1, \ldots, b_m\}$ and, for a set S of atoms, $not\, S = \{not\, q \mid q \in S\}$. If $body^-(r) = \emptyset$, then r is called *positive*; if $body^+(r) = \emptyset$, then r is called *negative*, and if $body(r) = \emptyset$, then r is a *fact*.

Given a logic program P, $B_P = \bigcup_{r \in P} head(r) \cup body^+(r) \cup body^-(r)$ is the set of all atoms appearing in P, and $Lit_P = B_P \cup not\, B_P$ is the set of all *literals*, i.e., all atoms and their default negations, in P. Also, $heads(P)$ denotes the set $\{p \mid p = head(r) \wedge r \in P\}$.

The well-founded semantics is based on three-valued interpretations. A *three-valued interpretation* I for P is defined as $I = I^+ \cup not\, I^-$ 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.

Several alternative, equivalent definitions of the well-founded semantics exist, and we recall the one based on the alternating fixpoint [10] which relies on the Gelfond-Lifschitz transformation for the answer set semantics [12].

Definition 1. *Let P be a logic program and S a set of atoms in P. The* Gelfond-Lifschitz transformation *of P with respect to S, denoted P^S, is the logic program obtained by $P^S = \{head(r) \leftarrow body^+(r) \mid r \in P, body^-(r) \cap S = \emptyset\}$.*

Given a logic program P and $S \subseteq B_P$, we define $\Gamma_P(S) = least(P^S)$ where $least(P^S)$ 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 as follows [10]:

Definition 2. *The* well-founded model *$WFM(P)$ of a normal logic program P is obtained as $WFM(P) = lfp(\Gamma_P^2) \cup not\, (B_P \setminus gfp(\Gamma_P^2))$.*

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

[3] An extended version of this paper containing all the proofs can be found at http://centria.di.fct.unl.pt/~mknorr/ForgettingLPNMR13.pdf.

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 $not\, p$). As outlined in the introduction, we want to achieve this based on the semantics rather than the syntax (F4) and ground it in forgetting in classical logic (F1).

Forgetting p in a classical propositional theory T is defined as $T[p/\mathbf{t}] \vee T[p/\mathbf{f}]$ where $T[p/\mathbf{t}]$ and $T[p/\mathbf{f}]$ are obtained from T by substituting all occurrences of p with the truth value $\mathbf{t}$ and $\mathbf{f}$ respectively. We may generalize this idea to our three-valued setting by defining forgetting p in T as $T[p/\mathbf{t}] \vee T[p/\mathbf{u}] \vee T[p/\mathbf{f}]$, but we face the same problem as [9]. Namely, the notion of disjunctions of logic programs does not exist. In fact, this would be counter-productive when trying to obtain one unique well-founded model for the result of forgetting. So, we tackle the matter semantically 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 3. *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) $WFM(P') = WFM(P) \setminus (\{p\} \cup \{not\, p\})$

Note that, by item (1), no new symbols are introduced, i.e., our definition of forgetting satisfies (F2) of the general requirements of forgetting presented in the introduction. Moreover, by item (2), (F3) is also satisfied, which is also witnessed in the following proposition.

Proposition 4. *Let P be a logic program and p an atom. Then for any literal $l \in Lit \setminus (\{p\} \cup \{not\, p\})$, $\mathsf{forget}(P, p) \models_{wf} l$ if and only if $P \models_{wf} l$.*

Proof. By (2) of Def. 3, $WFM(\mathsf{forget}(P, p)) = WFM(P) \setminus (\{p\} \cup \{not\, p\})$. Hence, for all $l \in Lit \setminus (\{p\} \cup \{not\, p\})$, $l \in WFM(\mathsf{forget}(P, p))$ iff $l \in WFM(P)$. □

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 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 one syntactic variant of all semantically equivalent results of forgetting about p in P.

Proposition 5. *Let P be a logic program and p an atom. If P' and P'' are two results of forgetting about p in P, then $P' \equiv_{wf} P''$.*

Proof. This follows directly from the definition of $\equiv_{wf}$ and (2) of Def. 3. □

Example 6. Consider the following examples.

1. $P = \{p \leftarrow .\ q \leftarrow p.\ a \leftarrow \mathit{not}\, a.\}$ Then $\mathsf{forget}(P, p) = \{q \leftarrow .\ a \leftarrow \mathit{not}\, a.\}$, $\mathsf{forget}(P, q) = \{p \leftarrow .\ a \leftarrow \mathit{not}\, a.\}$, and $\mathsf{forget}(P, a) = \{p \leftarrow .\ q \leftarrow p.\}$. But $\mathsf{forget}(P, a)$ may also be $\{p \leftarrow .\ q \leftarrow .\ p \leftarrow p.\ q \leftarrow \mathit{not}\, q\}$ showing that Def. 3 does not define one unique syntactic result of forgetting.
2. $P = \{p \leftarrow \mathit{not}\, a.\ p \leftarrow \mathit{not}\, b.\ a \leftarrow \mathit{not}\, b.\ b \leftarrow \mathit{not}\, a.\}$. Then $\mathsf{forget}(P, a) = \{b \leftarrow \mathit{not}\, b.\ p \leftarrow \mathit{not}\, b.\}$.

The next property generalizes Prop. 5 and also shows that our definition of forgetting satisfies (F4) entirely.

Proposition 7. *Let P and P' be two logic programs and p an atom. If $P \equiv_{wf} P'$, then $\mathsf{forget}(P, p) \equiv_{wf} \mathsf{forget}(P', p)$.*

Proof. If $P \equiv_{wf} P'$, then $WFM(P) = WFM(P')$ holds. It follows directly that $WFM(\mathsf{forget}(P, p)) = WFM(\mathsf{forget}(P', p))$, so $\mathsf{forget}(P, p) \equiv_{wf} \mathsf{forget}(P', p)$. □

Indeed, Prop. 7 states that forgetting preserves equivalence on $\equiv_{wf}$. In general, we say that forgetting *preserves* an equivalence $\equiv_X$ on logic programs if, for all logic programs P and P' and every atom p, $P \equiv_X P'$ implies $\mathsf{forget}(P, p) \equiv_X \mathsf{forget}(P', p)$. Important notions of equivalence that are studied in the literature w.r.t. logic programs are strong equivalence [16,4] or uniform equivalence [8]. In the former case, two programs P and P' are *strongly equivalent* if $P \cup Q$ and $P' \cup Q$ are equivalent for each logic program Q. In the latter case, two programs P and P' are *uniformly equivalent* if $P \cup Q$ and $P' \cup Q$ are equivalent for each finite set of facts Q.

Our definition of forgetting does preserve neither strong nor uniform equivalence. Intuitively, the reason is that Def. 3 only specifies the change on the semantics but not the precise syntactic form of the resulting program. Consider P in 2. from Ex. 6. P is obviously strongly equivalent to itself, but $\{b \leftarrow \mathit{not}\, b.\ p \leftarrow \mathit{not}\, b.\}$ and $\{b \leftarrow \mathit{not}\, p.\ p \leftarrow \mathit{not}\, b.\}$ are both valid results of forgetting about a from P but not strongly or uniformly equivalent w.r.t., e.g., $\{p \leftarrow .\}$. Following a similar argument as in [9], we can show that our notion of forgetting does not preserve any notion of equivalence that is stronger than $\equiv_{wf}$, where one equivalence $\equiv_1$ is stronger than $\equiv_2$ if $P \equiv_1 P'$ implies $P \equiv_2 P'$ for all programs P and P', but there are programs Q and Q' such that $Q \equiv_2 Q'$, yet $Q \not\equiv_1 Q'$. Note, however, that specific algorithms for computing (representatives) of $\mathsf{forget}(P, p)$ may preserve strong and uniform equivalence. E.g., algorithm $\mathsf{forget}_1(P, p)$ in Sect. 4 actually does preserve strong and uniform equivalence. Also note that [19] presents strongly equivalent forgetting for answer sets based on HT-models, but the result of forgetting in [19] is in general not expressible in normal logic programs. This is why we do not consider this here and leave it for future work.

We may also generalize the definition of forgetting to a set of atoms S as follows: the result of forgetting about S in P, $\mathsf{forget}(P, S)$, is a logic program P' such that the following two conditions are satisfied: (1) $B_{P'} \subseteq B_P \setminus \{p \mid p \in S\}$, and (2) $WFM(P') = WFM(P) \setminus (\{p \mid p \in S\} \cup \{\mathit{not}\, p \mid p \in S\})$. Then we can

show that the same result can be achieved by forgetting the elements of the set of atoms sequentially one-by-one, and independently from the order.

Proposition 8. *Let P be a logic program and $S = \{q_1, \ldots, q_n\}$ a set of atoms. Then, for any permutation $(i_1, \ldots, i_n)$ of $\{1, \ldots, n\}$,*

$$\mathsf{forget}(\mathsf{forget}(P, q_{i_1}), \ldots, q_{i_n}) \equiv_{wf} \mathsf{forget}(\mathsf{forget}(P, q_1), \ldots, q_n).$$

Proof. This follows directly from Def. 3. □

Proposition 9. *Let P be a logic program and $S = \{q_1, \ldots, q_n\}$ a set of atoms. Then*

$$\mathsf{forget}(P, S) \equiv_{wf} \mathsf{forget}(\mathsf{forget}(P, q_1), \ldots, q_n).$$

Proof. This follows directly from the definition of $\mathsf{forget}(P, S)$ and Def. 3. □

Propositions 8 and 9 may be beneficial if we apply one of the algorithms devised in Sect. 4 to the case of forgetting about a set of atoms.

One well-established property of the well-founded semantics is that it is faithful w.r.t. the answer set semantics, i.e., if atom p is true in $WFM(P)$, then p occurs in each answer set of P, and, likewise, if $not\, p \in WFM(P)$, then p occurs in no answer set of P. Next, we establish a result that shows that our notion of forgetting is faithful w.r.t. semantic forgetting in answer set programming [9].

First, we recall from [9] the notion of forgetting in answer set programming (ASP) [12], based on the notion of p-answer set. Given atom p, a set S' is a *p-subset of a set S*, denoted $S' \subseteq_p S$, if $S' \setminus \{p\} \subseteq S \setminus \{p\}$, and a *strict p-subset of a set S*, denoted $S' \subset_p S$, if $S' \setminus \{p\} \subset S \setminus \{p\}$. A set $S \in B_P$ is an *answer set of* logic program P if $\Gamma_P(S) = S$, and $\mathcal{AS}(P)$ denotes the set of all answer sets of P. A p-answer set is defined in [9] as follows.

Definition 10. *Let P be a logic program, $p \in B_P$, and $S \subseteq B_P$. For a set $\mathcal{S}$ of sets of atoms, $S \in \mathcal{S}$ is* p-minimal *if there is no $S' \in S$ such that $S' \subset_p S$. An answer set S of P is a* p-answer set *of P if S is p-minimal in $\mathcal{AS}(P)$. By $\mathcal{AS}_p(P)$ we denote the set of all p-answer sets of P.*

As argued in [9], p-answer sets are necessary in the following definition of ASP-forgetting to avoid that non-minimal answer sets result from forgetting.

Definition 11. *Let P be a logic program and p an atom. A logic program P' represents the result of* ASP-forgetting about p in P, $\mathsf{forget}_{ASP}(P, p)$, *if*

(1) $B_{P'} \subseteq B_P \setminus \{p\}$, i.e., p does not occur in P', and
(2) $\mathcal{AS}(P') = \{S \setminus \{p\} \mid S \in \mathcal{AS}_p(P)\}$.

We can now present the result that shows that our notion of forgetting is faithful w.r.t. ASP-forgetting.

Theorem 12. *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* $M \in \mathcal{AS}(\mathsf{forget}_{ASP}(P,p))$.

Proof. Consider the case $q \in WFM(\mathsf{forget}(P,p))$. Then $q \in WFM(P)$ by (2) of Def. 3, and $q \in M$ for all $M \in \mathcal{AS}(P)$. By (1) of Prop. 1 in [9], we have that every p-answer set S of P is an answer set of P. Hence, $q \in S$ for each such p-answer set S of P. Thus, $q \in S \setminus \{p\}$ for every p-answer set of P, and therefore $q \in M'$ for all $M' \in \mathcal{AS}(\mathsf{forget}_{ASP}(P,p))$, by (2) of Def. 11.

Now consider the case $not\, q \in WFM(\mathsf{forget}(P,p))$. Then $not\, q \in WFM(P)$ by (2) of Def. 3, and $q \notin M$ for all $M \in \mathcal{AS}(P)$. By (2) of Def. 11, we obtain $q \notin M'$ for all $M' \in \mathcal{AS}(\mathsf{forget}_{ASP}(P,p))$. □

Based on Theorem 12 and Theorem 3 in [9], we can establish that forgetting under well-founded semantics also faithfully approximates classical forgetting. We refer to [9] for the definitions of $lcomp(P)$ —the completion of P plus loop formulas—and point out that $\mathsf{f_C}(T,p)$ denotes classically forgetting about p in theory T, and $MM(T)$ the set of classical minimal models in T.

Corollary 13. *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* $M \in MM(\mathsf{f_C}(lcomp(P),p))$.
2. *If* $not\, q \in WFM(\mathsf{forget}(P,p))$*, then* $q \notin M$ *for all* $M \in MM(\mathsf{f_C}(lcomp(P),p))$.

This establishes the relationship to classical forgetting for our notion of forgetting under well-founded semantics (F1). Note that if the well-founded model of P is *total*, i.e., no $p \in B_P$ is undefined in $WFM(P)$, then we achieve the following result.

Corollary 14. *Let* P *be a logic program,* p *an atom, and* $WFM(P)$ *total. Then* $WFM(\mathsf{forget}(P,p)) = \mathcal{AS}(\mathsf{forget}_{ASP}(P,p)) = MM(\mathsf{f_C}(lcomp(P),p))$.

4 Computation of Forgetting

4.1 Naïve Semantics-based Algorithm

Def. 3 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.

Example 15. Consider the following two logic programs:

1. $P = \{a \leftarrow .\ b \leftarrow not\, a.\ c \leftarrow not\, b.\ d \leftarrow a, c.\}$. Here, $a \in WFM(P)$. Then $\mathsf{forget}(P,a) = \{c \leftarrow .\ d \leftarrow .\}$
2. $P = \{a \leftarrow not\, b.\ b \leftarrow not\, a.\ c \leftarrow b, not\, a.\}$. Here, neither a nor $not\, a$ in $WFM(P)$. Then $\mathsf{forget}(P,a) = \{b \leftarrow not\, b.\ c \leftarrow not\, c.\}$.

It is easy to see that Algorithm $\mathsf{forget}_1(P,p)$ yields a correct result.

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

Theorem 16. *Given a logic program P and an atom p, then $\mathsf{forget}_1(P, p)$ computes a correct result of $\mathsf{forget}(P, p)$.*

Proof. The generic program P' created in Algorithm $\mathsf{forget}_1(P, p)$ exactly matches the conditions in Def. 3, i.e., p is not present in P' and $WFM(P') = WFM(P) \setminus (\{p\} \cup \{not\, p\})$. □

The algorithm always terminates and is in **P**.

Theorem 17. *Given a logic program P and an atom p, $\mathsf{forget}_1(P, p)$ terminates and computing P' is in* **P**.

Proof. Due to the finite set of propositional rules, computing the well-founded model and creating P' trivially terminates. Moreover, since computing the well-founded model is in **P** [11], and the creation of the generic program can be done in linear time, we obtain the desired result. □

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 well-founded semantics 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.[4] 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

[4] See, e.g., XSB for an in implementation at `http://xsb.sourceforge.net`.

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 of the form (1) 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 $\mathsf{forget}_2(P, p)$

value of p is determined, we only make minimal changes to accommodate the forgetting of p: if p is true or false, then body atoms or even 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.

Example 18. Consider again the two logic programs from Example 15.

1. $P = \{a \leftarrow .\ b \leftarrow not\, a.\ c \leftarrow not\, b.\ d \leftarrow a, c.\}$. Here, a is **t** in $WFM(P)$, therefore $\mathsf{forget}(P, a) = \{c \leftarrow not\, b.\ d \leftarrow c.\}$.
2. $P = \{a \leftarrow not\, b.\ b \leftarrow not\, a.\ c \leftarrow b, not\, a.\}$. Here, a is **u**, so $\mathsf{forget}(P, a) = \{b \leftarrow not\, b.\ c \leftarrow b, not\, c.\}$.

In both cases, the result is closer to the original program when compared to Example 15, which also indicates the higher efficiency of $\mathsf{forget}_2(P, p)$: only part of the well-founded model is computed and only part of the program is rewritten.

Algorithm $\mathsf{forget}_2(P, p)$ also yields a correct result.

Theorem 19. *Given a logic program P and an atom p, then $\mathsf{forget}_2(P, p)$ computes a correct result of $\mathsf{forget}(P, p)$.*

Proof. After computing the truth value of p, all instances of p are removed from P in each case, so condition (1) of Def. 3 is satisfied. Therefore, p does not appear in $WFM(P')$, but the substitutions/removals of rules are done in all three cases as if p was still present, so $WFM(P') = WFM(P) \setminus (\{p\} \cup \{not\, p\})$. □

The algorithm terminates and is in **P** as well.

Theorem 20. *Given a logic program P and an atom p, $\mathsf{forget}_2(P, p)$ terminates and computing P' is in* **P**.

Proof. SLG resolution used in XSB terminates [5] in particular for the propositional case of finite rules, and the transformations in *Step 2* do terminate as well for the considered class of logic programs. Moreover, SLG resolution is in **P** [5], and, since the transformations w.r.t. p can be done in linear time, we obtain that $\mathsf{forget}_2(P, p)$ can be computed in **P**. □

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 program transformations instead. Program transformations have been widely used in simplifying programs and designing algorithms in logic programming [2]. We introduce an algorithm for computing the result of forgetting about an atom p in logic program P under the well-founded semantics by a series of different program transformations. This ultimately corresponds to tackling (F5), thereby completing the match to the five criteria presented in the introduction.

The basic idea builds on a set of program transformations [3], which is a refinement of [2] for the well-founded semantics avoiding the potential exponential size of the resulting program in [2]. The result is called program remainder, based on which we can obtain a program for forgetting about p in P. We start recalling the program transformations and the program remainder from [3].

Definition 21. *The rewriting system $\mapsto_X$ for logic programs consists of the following transformations: Let P and P' be two logic programs.*

Positive reduction *P' is obtained from P by* positive reduction *iff there is a rule r in P and $c \in body^-(r)$ such that $c \notin heads(P)$ and $P' = (P \setminus \{r\}) \cup \{r_p\}$, where $r_p : head(r) \leftarrow body^+(r), not\, (body^-(r) \setminus \{c\})$.*

Negative reduction *P' is obtained from P by* negative reduction *iff there are two rules in P, r and r', such that r' is the fact $head(r') \leftarrow$, $head(r') \in body^-(r)$ and $P' = P \setminus \{r\}$.*

Success *P' is obtained from p by* success *iff there are two rules in P, r and r', such that r' is the fact $head(r') \leftarrow$, $head(r') \in body^+(r)$ and $P' = (P \setminus \{r\}) \cup \{r_s\}$, where $r_s : head(r) \leftarrow (body^+(r) \setminus \{head(r')\}), not\, body^-(r)$.*

Failure *P' is obtained from P by* failure *iff there is a rule r in P and $c \in body^+(r)$ such that $c \notin heads(P)$ and $P' = P \setminus \{r\}$.*

Loop Detection *P' is obtained from P by* loop detection *iff there is a non-empty set of atoms S such that*

1. *for each rule r in P, if $head(r) \in S$, then $S \cap body^+(r) \neq \emptyset$;*
2. *$P' = P \setminus \{r \mid head(r) \in S\}$.*

The program remainder *$\hat{P}$ is the normal form of $\mapsto_X$ w.r.t. P.*

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 $p \notin \mathit{heads}(\hat{P})$, then substitute p and $\mathit{not}\, p$ in each body of a rule r in $\hat{P}$ of the form (1) by $\mathit{not}\, \mathit{head}(r)$. After that, remove all rules whose head is p.
Step 3. Output the result P' as $\mathsf{forget}(P, p)$.

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

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 $\mathit{WFM}(P)$ in the following way: $p \in \mathit{WFM}(P)$ iff $p \leftarrow . \in \hat{P}$ and $\mathit{not}\, p \in \mathit{WFM}(P)$ iff $p \notin \mathit{heads}(\hat{P})$. We can use this to create the algorithm $\mathsf{forget}_3(P, p)$ shown in Fig. 3 which computes the result of forgetting about p in P syntactically. Note that if $p \leftarrow . \in \hat{P}$ or $p \notin \mathit{heads}(\hat{P})$, then we do not have to do anything, but remove $p \leftarrow .$ in the former case, since the transformation into $\hat{P}$ already simplified P accordingly.

Example 22. Consider again the two logic programs from Ex. 15.

1. $P = \{a \leftarrow .\ b \leftarrow \mathit{not}\, a.\ c \leftarrow \mathit{not}\, b.\ d \leftarrow a, c.\}$. Here, $\hat{P}$ only contains facts for a, c, and d, therefore $\mathsf{forget}(P, a) = \{c \leftarrow .\ d \leftarrow .\}$.
2. $P = \{a \leftarrow \mathit{not}\, b.\ b \leftarrow \mathit{not}\, a.\ c \leftarrow b, \mathit{not}\, a.\}$. Here, $\hat{P} = P$, so $\mathsf{forget}(P, a) = \{b \leftarrow \mathit{not}\, b.\ c \leftarrow b, \mathit{not}\, c.\}$.

Note that the result here is identical to Ex. 15 in case of 1. and identical to Ex. 18 in case of 2., indicating that the changes resulting from $\mathsf{forget}_3(P, p)$ are positioned in between those obtained from $\mathsf{forget}_1(P, p)$ and $\mathsf{forget}_2(P, p)$.

Algorithm $\mathsf{forget}_3(P, p)$ again yields a correct result.

Theorem 23. *Given a logic program P and an atom p, then* $\mathsf{forget}_3(P, p)$ *computes a correct result of* $\mathsf{forget}(P, p)$.

Proof. All instances of p are removed from $\hat{P}$, so condition (1) of Def. 3 is satisfied. Therefore, p also does not appear in $\mathit{WFM}(P')$. Now, by Theorem 23 of [3], $p \in \mathit{WFM}(P)$ iff $p \leftarrow . \in \hat{P}$ and $\mathit{not}\, p \in \mathit{WFM}(P)$ iff $p \notin \mathit{heads}(\hat{P})$, so the substitutions/removals of rules in *Step 2* are done as if p was still present, and therefore $\mathit{WFM}(P') = \mathit{WFM}(P) \setminus (\{p\} \cup \{\mathit{not}\, p\})$. □

The algorithm terminates and is in **P** as well.

Theorem 24. *Given a logic program P and an atom p, $\mathsf{forget}_3(P,p)$ terminates and computing P' is in* **P**.

Proof. The termination of *Step 1* is shown in [3], the remaining transformations in *Step 2* do also terminate for finite sets of propositional programs. In particular, in Lemma 26 in [3], it is shown that computing $\hat{P}$ is in **P**, and since the transformations in *Step 2* can be done in linear time, we finish the proof. □

We obtain the general result on computational complexity for reasoning under forgetting about p in P.

Theorem 25. *Given a logic program P, an atom p, and a three-valued interpretation M, deciding whether M is the well-founded model of $\mathsf{forget}(P,p)$ is* **P**-*complete. Additionally, for a literal l, deciding whether $\mathsf{forget}(P,p) \models_{wf} l$ is* **P**-*complete.*

Proof. We have already shown for all three algorithms in this section that forgetting about p in P is in **P**. In fact, as shown in [11] and recalled in [6], computing the well-founded model is **P**-complete for the propositional case. Since, Def. 3 (and also algorithm $\mathsf{forget}_1(P,p)$) hinges upon this computation, and the creation of the program P' is linear in the worst case, we obtain the result directly for deciding whether M is the well-founded model. Checking whether some literal l is a logical consequence under the well-founded semantics then simply amounts to a look-up in $WFM(\mathsf{forget}(P,p))$. □

We have devised three different algorithms for forgetting under well-founded semantics with the same computational complexity. We have already argued that $\mathsf{forget}_2(P,p)$ is in general more efficient than $\mathsf{forget}_1(P,p)$ and the same result carries over in comparison to $\mathsf{forget}_3(P,p)$ for the same reasons, namely that the whole program does not need to be processed in the reasoning step to obtain the result. We now consider further criteria to compare these three algorithms.

First, unlike the algorithms for ASP-forgetting about p from P, where the output may be exponentially larger than P, our three algorithms produce results for forgetting under well-founded semantics that in general reduce the size of the resulting program, where the size of a program P, $|P|$, is measured by the number of rules in P, and the size of the output of algorithm $\mathsf{forget}_x(P,p)$ is denoted by $|\mathsf{forget}_x(P,p)|$ for $x = \{1,\ldots,3\}$.

Proposition 26. *Let P be a logic program and p an atom. We obtain that $|P| \geq |\mathsf{forget}_2(P,p)| \geq |\mathsf{forget}_3(P,p)| \geq |\mathsf{forget}_1(P,p)|$.*

Proof. Looking at $\mathsf{forget}_2(P,p)$, no new rule is introduced, only existing atoms are substituted or removed, or even entire rules are removed. Hence, $|P| \geq |\mathsf{forget}_2(P,p)|$ holds immediately.

Now consider $|\mathsf{forget}_2(P,p)| \geq |\mathsf{forget}_3(P,p)|$. We show that whenever there is a rule r in $P' = \mathsf{forget}_3(P,p)$, then there is a rule $r' \in P'' = \mathsf{forget}_2(P,p)$. Let $r \in P'$. If $r = a \leftarrow .$ for some atom a, then $a \leftarrow . \in \hat{P}$ and $a \in WFM(P)$. In this case, there has to exist a rule $r' \in P$ with $head(r') = a$ such that either p

does not appear in its body, or $p \in \mathit{body}(r)$, but not $\mathit{not}\, p$ and $p \in \mathit{WFM}(P)$, or $\mathit{not}\, p \in \mathit{body}(r)$, but not p and $\mathit{not}\, p \in \mathit{WFM}(P)$. In all three cases there exists a rule $r'' \in P''$. Alternatively, $r = a \leftarrow \mathit{body}(r)$ with $\mathit{body}(r) \neq \emptyset$. In this case, there also exists a rule $r' \in P$ with $\mathit{head}(r') = a$ and $\mathit{body}(r') \neq \emptyset$ such that $\mathsf{forget}_2(P, p)$ does not remove the rule completely. Hence, there is a rule $r'' \in P''$.

Finally, we consider $|\mathsf{forget}_3(P, p)| \geq |\mathsf{forget}_1(P, p)|$. We show that whenever there is a rule r in $P' = \mathsf{forget}_1(P, p)$, then there is a rule $r' \in P'' = \mathsf{forget}_3(P, p)$. Let $r \in P'$. If $r = a \leftarrow .$ for some atom a, then $a \in \mathit{WFM}(P)$. In this case, $a \leftarrow . \in \hat{P}$ and $r \in P'' = \mathsf{forget}_3(P, p)$. Alternatively, $r = a \leftarrow \mathit{not}\, a$ for some atom a and $a \in B_{P'} \setminus (M^+ \cup M^-)$. In this case, $a \in \mathit{heads}(\hat{P})$, and, since $a \neq p$ there is a rule $r' \in P''$ with $\mathit{head}(r') = a$. □

Note that the inverse $|P| \leq |\mathsf{forget}_2(P, p)| \leq |\mathsf{forget}_3(P, p)| \leq |\mathsf{forget}_1(P, p)|$ does not hold, not even for any two of the four sizes, i.e., in general, no identity exists.

One way to evaluate Prop. 26 is to consider that a smaller resulting program is better, because it helps to save space, which is one of the justifications to apply forgetting in the first place [17]. Another interpretation follows the line of one of our arguments in favor of $\mathsf{forget}_2(P, p)$, namely to try to change as little as possible in the original program. For simplicity we measure this notion by the number of rules changed when applying $\mathsf{forget}_x(P, p)$ for $x \in \{1, \ldots, 3\}$, denoted $\mathit{change}(\mathsf{forget}_x(P, p))$, i.e., those rules that are removed and those that are altered, and we obtain the following corollary.

Corollary 27. *Let P be a logic program and p an atom. Then we obtain that $\mathit{change}(\mathsf{forget}_1(P, p)) \geq \mathit{change}(\mathsf{forget}_3(P, p)) \geq \mathit{change}(\mathsf{forget}_2(P, p))$.*

A more fine-grained result can be achieved if we consider the number of atoms that are changed instead of the number of rules. This can be witnessed already in Examples 15, 18, and 22, where a relation similar to that in the previous corollary can be observed, but we leave the details for future work.

5 Conclusions

We have developed a notion of semantic forgetting under the well-founded semantics. Forgetting under the well-founded semantics indeed satisfies the important properties indicated in [9] for the answer set answers. Moreover, our approach represents a sound approximation of that work in the sense that whatever is derivable from the program that results from forgetting under the well-founded semantics, also is (skeptically) derivable from the result of semantic forgetting under the answer set semantics [9]. Likewise, classical forgetting is approximated by our notion of forgetting.

We have presented three different algorithms for computing the result of forgetting under the well-founded semantics, one that is based on computing the well-founded model, another that relies on only querying for the truth value of the atom to be forgotten, and also one that is entirely syntax-based. In all three cases, the computational complexity is in **P**, which means that we can choose the

preferred method based on the reasoners available and their properties, but also on the size of the resulting program and on how much the result should resemble the original program. The result computed by $\mathsf{forget}_2(P, p)$ makes the smallest amount of changes and contains also in general the largest amount of rules. On the other side, $\mathsf{forget}_1(P, p)$ makes the most changes, but yields the smallest output, while $\mathsf{forget}_3(P, p)$ stays in between. In any case, our algorithms stay tractable and within the computational complexity of the well-founded semantics, and additionally do never increase the size of the output compared to the original program. This means that forgetting under the well-founded semantics is readily usable in practice.

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 [19] for the answer set semantics, possibly based on [4]. An important issue there is whether the result is actually expressible as a normal logic program again. Second, since forgetting has been considered for description logics, we may also consider forgetting in formalisms that combine description logics and non-monotonic logic programming rules under well-founded semantics, such as [13].

References

1. Antoniou, G., Eiter, T., Wang, K.: Forgetting for defeasible logic. In: Bjørner, N., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings. 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.) Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006. pp. 25–35. AAAI Press (2006)
5. Chen, W., Warren, D.S.: Tabled Evaluation with Delaying for General Logic Programs. J. ACM 43(1), 20–74 (1996)
6. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. 33(3), 374–425 (2001)
7. van Ditmarsch, H.P., Herzig, A., Lang, J., Marquis, P.: Introspective forgetting. In: Wobcke, W., Zhang, M. (eds.) AI 2008: Advances in Artificial Intelligence, 21st Australasian Joint Conference on Artificial Intelligence, Auckland, New Zealand, December 1-5, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5360, pp. 18–29. Springer (2008)
8. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Trans. Comput. Log. 8(3) (2007)
9. Eiter, T., Wang, K.: Semantic forgetting in answer set programming. Artif. Intell. 172(14), 1644–1672 (2008)

10. Gelder, A.V.: The alternating fixpoint of logic programs with negation. J. Comput. Syst. Sci. 47(1), 185–221 (1993)
11. Gelder, A.V., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3-4), 365–385 (1991)
13. 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)
14. 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)
15. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: Formula-variable independence and forgetting. J. Artif. Intell. Res. (JAIR) 18, 391–443 (2003)
16. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. 2(4), 526–541 (2001)
17. Lin, F., Reiter, R.: Forget it! In: In Proceedings of the AAAI Fall Symposium on Relevance. pp. 154–159 (1994)
18. Lutz, C., Wolter, F.: Foundations for uniform interpolation and forgetting in expressive description logics. In: Walsh, T. (ed.) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 989–995. IJCAI/AAAI (2011)
19. Wang, Y., Zhang, Y., Zhou, Y., Zhang, M.: Forgetting in logic programs under strong equivalence. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. AAAI Press (2012)
20. 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)
21. Zhang, Y., Foo, N.Y., Wang, K.: Solving logic program conflict through strong and weak forgettings. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005. pp. 627–634. Professional Book Center (2005)
22. Zhang, Y., Zhou, Y.: Knowledge forgetting: Properties and applications. Artif. Intell. 173(16-17), 1525–1537 (2009)
23. Zhou, Y., Zhang, Y.: Bounded forgetting. In: Burgard, W., Roth, D. (eds.) Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011. AAAI Press (2011)