On Syntactic Forgetting with Strong Persistence

Matti Berthold
Universität Leipzig
berthold@informatik.uni-leipzig.de

Abstract

It is generally agreed upon that so-called *strong persistence* $(\mathbf{SP})$ captures best the essence of *forgetting* in logic programming. While classes of operators, such as $\mathsf{F_R}$ and $\mathsf{F_{SP}}$, that satisfy immediate relaxations of $(\mathbf{SP})$, and in the case of $\mathsf{F_{SP}}$, even $(\mathbf{SP})$ whenever possible, have been characterized semantically, many practical questions have yet to be addressed. This technical paper aims to answer one of them: How can atoms be forgotten from a program without having to calculate its exponential number of models? To this end, we introduce two concrete representatives of $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ that forget sets of atoms by syntactical manipulation of a program's rules. This may in many cases prevent exponential blowups and produce a forgetting result that is close to the original program.

1 Introduction

The research area of *forgetting* has recently seen ample amounts of progress in the context of logic programming. Intuitively, forgetting means that bits of a program are removed in such a way that the logical connections between the remaining bits are left intact. Forgetting might be necessary, for example, to tend to legal requests to remove information of a user from a knowledge base, or, to simplify it by removing unnecessary data.

Recent studies investigated several forgetting properties and identified *Strong Persistence* $(\mathbf{SP})$ to most accurately describe the essence of forgetting (Gonçalves, Knorr, and Leite 2016a). $(\mathbf{SP})$, however, is not always obtainable (Gonçalves, Knorr, and Leite 2016b). Therefore forgetting operators have been defined semantically, satisfying different relaxations of $(\mathbf{SP})$, among them $\mathsf{F_R}$ and $\mathsf{F_{SP}}$, where notably $\mathsf{F_{SP}}$ does satisfy $(\mathbf{SP})$, whenever possible (Gonçalves et al. 2017).

While it is possible to obtain specific forgetting results from the desired semantics by counter-models construction (Cabalar and Ferraris 2007), this process necessarily involves calculating an exponential number of interpretations. In many cases, such blowups can be avoided by constructing a forgetting result purely by syntactically manipulating the input program.

In the literature there exists a number of such syntactic forgetting operators (Zhang and Foo 2006; Eiter and Wang 2008; Knorr and Alferes 2014; Gonçalves et al. 2021), however, most of them do not satisfy the most basic properties, much less $(\mathbf{SP})$ (cf. (Berthold et al. 2019) for an overview).

The only syntactic operator to satisfy $(\mathbf{SP})$, whenever possible, f_{SP}, only forgets one atom. To this end, it was proven to conform to the semantics of class $\mathsf{F_{SP}}$. But can f_{SP} be iterated to correctly forget arbitrary sets of atoms?

Example 1. *Consider forgetting p and q from program P_1:*

$$a \leftarrow p, q. \qquad q \leftarrow not\, p. \qquad p \leftarrow not\, not\, p.$$

The only remaining atom, a, is implied by p and q, who are mutually exclusive. Since a can therefore not be deduced, one would expect the result of forgetting to be the empty program.[1] *And indeed, if we first forget q and then p via f_{SP}, we arrive at our desired result. Forgetting q first yields $\mathsf{f}_{SP}(P_1, q)$:*

$$\cancel{a \leftarrow p, not\, p.} \qquad p \leftarrow not\, not\, p.$$

where the appearance of q in the rule $a \leftarrow p, q.$ is simply replaced by $not\, p$, which implied q before. The resulting rule can then be disregarded, as p and $not\, p$ are unsatisfiable together. Forgetting p subsequently naturally yields the empty program, i.e. $\mathsf{f}_{SP}(\mathsf{f}_{SP}(P_1, q), p) = \emptyset$.

Observe what happens if the order of forgetting p and q is reversed: Forgetting p first yields $\mathsf{f}_{SP}(P_1, p)$:

$$a \leftarrow q, not\, not\, a. \qquad q \leftarrow not\, not\, q.$$

where the cyclic dependency on p is transferred to a and q, but the information that a was only implied by a contradiction is lost. Forgetting q thereafter yields $\mathsf{f}_{SP}(\mathsf{f}_{SP}(P_1, p), q)$:

$$a \leftarrow not\, not\, a.$$

which is not our desired forgetting result.

Apparently, f_{SP} is sensitive to the order in which atoms are forgotten. We will show that, worse yet, for both classes, $\mathsf{F_R}$ and $\mathsf{F_{SP}}$, there are instances such that forgetting atoms iteratively in *any* order will create a result deviating from forgetting all atoms at once. This proves the existing forgetting operator f_{SP} insufficient for forgetting arbitrary sets of atoms from programs by $\mathsf{F_{SP}}$ semantics, and illustrates that concrete forgetting procedures for both $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ necessarily are quite complex.

This paper introduces the syntactic operators f_R and f^*_{SP} to overcome these issues, that forget any set of atoms correctly by the semantics of $\mathsf{F_R}$ and $\mathsf{F_{SP}}$.

[1]This intuition coincides with $\mathsf{F_{SP}}$.

The paper is structured as follows: Sec. 2 recalls the basics of logic programming and the semantics of forgetting. Sec. 3 collects a number of auxiliary syntactic operators that are used when forgetting. We put a particular emphasis on their connection to HT-semantics and provide some insights that are essential to our main results.

The operators f_R and f^*_{SP} are defined and proven to be correct in Sec. 4 and 5 respectively. Finally Sec. 6 contains the intuitions behind the operators, as well as some positive results demonstrating their performance. We close the paper with a discussion in Sec. 7.

2 Background

Logic Programs We assume a *propositional signature* Σ. A *logic program* P over Σ is a finite set of *rules* of the form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, ..., b_l, not\, c_1, ..., not\, c_m, \\ not\, not\, d_1, ..., not\, not\, d_n,$$

where all $a_1, \ldots, a_k, b_1, \ldots, b_l, c_1, \ldots, c_m$, and $d_1, \ldots, d_n$ are atoms of Σ (Lifschitz, Tang, and Turner 1999). Such rules r are also written more succinctly as

$$H(r) \leftarrow B^+(r), not\, B^-(r), not\, not\, B^{--}(r),$$

where $H(r) = \{a_1, \ldots, a_k\}$, $B^+(r) = \{b_1, \ldots, b_l\}$, $B^-(r) = \{c_1, \ldots, c_m\}$, and $B^{--}(r) = \{d_1, \ldots, d_n\}$, and we will use both forms interchangeably. Given a rule r, $H(r)$ is called the *head* of r, and $B(r) = B^+(r) \cup not\, B^-(r) \cup not\, not\, B^{--}(r)$ is called the *body* of r, where, for a set A of atoms, $not\, A = \{not\, q \mid q \in A\}$ and $not\, not\, A = \{not\, not\, q \mid q \in A\}$.

$\Sigma(P)$ and $\Sigma(r)$ denote the set of atoms appearing in P and r, respectively.

Given a program P and an *interpretation*, i.e., a set $I \subseteq \Sigma$ of atoms, the *reduct* of P given I, is defined as $P^I = \{H(r) \leftarrow B^+(r) \mid r \in P$ such that $B^-(r) \cap I = \emptyset$ and $B^{--}(r) \subseteq I\}$. An *HT-interpretation* is a pair $\langle X, Y\rangle$ s.t. $X \subseteq Y \subseteq \Sigma$. Given a program P, an HT-interpretation $\langle X, Y\rangle$ is an *HT-model of* P, $\langle X, Y\rangle \models P$, iff $Y \models P$ and $X \models P^Y$, where $\models$ both denotes the standard satisfaction relation for classical logic and for HT-logic.[2] An HT-interpretation $\langle X, Y\rangle$ is *total* iff $X = Y$. Given a rule r, $\langle X, Y\rangle \models r$, iff $\langle X, Y\rangle \models \{r\}$. We admit that the set of HT-models of a program P is restricted to $\Sigma(P)$ even if $\Sigma(P) \subset \Sigma$. We denote by $\mathcal{HT}(P)$ the set of *all HT-models of P*. A set of atoms Y is an *answer set* of P iff $\langle Y, Y\rangle \in \mathcal{HT}(P)$, and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(P)$. We term HT-models $\langle X, Y\rangle$ s.t. $X \subset Y$ *witnesses*. The set of all answer sets of P is denoted by $\mathcal{AS}(P)$. Two programs P_1, P_2 are *equivalent* iff $\mathcal{AS}(P_1) = \mathcal{AS}(P_2)$ and *strongly equivalent*, $P_1 \equiv P_2$, iff $\mathcal{AS}(P_1 \cup R) = \mathcal{AS}(P_2 \cup R)$ for any program R. It is well-known that $P_1 \equiv P_2$ exactly when $\mathcal{HT}(P_1) = \mathcal{HT}(P_2)$ (Lifschitz, Pearce, and Valverde 2001). Given a set $V \subseteq \Sigma$, the V*-exclusion* of a set of answer sets (a set of HT-interpretations) $\mathcal{M}$, denoted $\mathcal{M}_{\parallel V}$, is $\{X \backslash V \mid X \in \mathcal{M}\}$ ($\{\langle X\backslash V, Y\backslash V\rangle \mid \langle X, Y\rangle \in \mathcal{M}\}$).

[2]For brevity, parentheses, commas and union signs within HT-interpretations may be omitted, such that, for example, $\langle \emptyset, Ypq\rangle$ means $\langle \emptyset, Y \cup \{p, q\}\rangle$.

Forgetting: Properties and Operators Let $\mathcal{P}$ be the set of all logic programs. A *forgetting operator* is a (partial) function $\mathsf{f} : \mathcal{P} \times 2^\Sigma \to \mathcal{P}$. The program $\mathsf{f}(P, V)$ is interpreted as the *result of forgetting about V from P*. Moreover, $\Sigma(\mathsf{f}(P, V)) \subseteq \Sigma(P)\backslash V$ is usually required. In the following we introduce some well-known properties for forgetting operators (Gonçalves, Knorr, and Leite 2016a).

Strong persistence is presumably the best known one (Knorr and Alferes 2014). It requires that the result of forgetting $\mathsf{f}(P, V)$ is strongly equivalent to the original program P, modulo the forgotten atoms.

($\mathbf{SP}$) f satisfies *strong persistence* iff, for each program P and each set of atoms V, we have: $\mathcal{AS}(P \cup R)_{\parallel V} = \mathcal{AS}(\mathsf{f}(P, V) \cup R)$ for all programs R with $\Sigma(R) \subseteq \Sigma\backslash V$.

Notably, ($\mathbf{SP}$) can be decomposed into the following three properties, i.e. an operator f satisfies ($\mathbf{SP}$) iff f satisfies all ($\mathbf{wC}$), ($\mathbf{sC}$) and ($\mathbf{SI}$), where

($\mathbf{wC}$) f satisfies *weakened consequence* iff, for each P and each set of atoms V: $\mathcal{AS}(\mathsf{f}(P, V)) \supseteq \mathcal{AS}(P)_{\parallel V}$.

($\mathbf{sC}$) f satisfies *strengthened consequence* iff, for each P and each set of atoms V: $\mathcal{AS}(\mathsf{f}(P, V)) \subseteq \mathcal{AS}(P)_{\parallel V}$.

Strong invariance requires that rules not mentioning atoms to be forgotten can be added before or after forgetting.

($\mathbf{SI}$) f satisfies *strong invariance* iff, for each program P and each set of atoms V, we have: $\mathsf{f}(P, V) \cup R \equiv \mathsf{f}(P \cup R, V)$ for all programs R with $\Sigma(R) \subseteq \Sigma\backslash V$.

Note that the presented properties are often considered for certain subclasses such as *disjunctive*, *normal* or *Horn programs*. Moreover, they naturally extend over classes of forgetting operators, where a class satisfies a property, iff all its members do.

In the light of the impossibility for a forgetting operator to satisfy ($\mathbf{SP}$) for all pairs $\langle P, V\rangle$, called *forgetting instances*, where P is a program and V is a set of atoms to be forgotten from P (Gonçalves, Knorr, and Leite 2016b), ($\mathbf{SP}$) was defined for concrete forgetting instances. A forgetting operator f satisfies $(\mathbf{SP})_{\langle P,V\rangle}$, if $\mathcal{AS}(\mathsf{f}(P, V) \cup R) = \mathcal{AS}(P \cup R)_{\parallel V}$, for all programs R with $\Sigma(R) \subseteq \Sigma\backslash V$. A sound and complete criterion Ω characterizes when it is not possible to forget while satisfying $(\mathbf{SP})_{\langle P,V\rangle}$.

The definition of Ω is rather technical. To decide Ω for a given $\langle P, V\rangle$, all total models $\langle YA, YA\rangle$ with $Y \subseteq \Sigma(P)\backslash V$ and $A \subseteq V$ are *considered* for which there are no witnesses $\langle YA', YA\rangle \in \mathcal{HT}(P)$ with $A' \subset A$. For each $Y \subseteq \Sigma(P)\backslash V$ all such *relevant* $A \subseteq V$ are collected:

$$Rel^Y_{\langle P,V\rangle} := \{A \subseteq V \mid \langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P) \text{ and } \\ \nexists A' \subset A \text{ s.t. } \langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)\}.$$

Then, an instance $\langle P, V\rangle$ *satisfies criterion* Ω iff between the considered total models with matching unforgotten part Y, there is no least element, with respect to their witnesses modulo V, more formally, iff there is $Y \subseteq \Sigma\backslash V$ such that the set of sets $\mathcal{R}^Y_{\langle P,V\rangle} := \{R^{Y,A}_{\langle P,V\rangle} \mid A \in Rel^Y_{\langle P,V\rangle}\}$ is non-empty and has no least element, where

$$R^{Y,A}_{\langle P,V\rangle} := \{X\backslash V \mid \langle X, Y \cup A\rangle \in \mathcal{HT}(P)\}.$$

Example 2. *Consider the forgetting instance $\langle P_2, \{p\}\rangle$, where P_2 has the following HT-models*[3]*:*

$$\boxed{\begin{array}{ccc} \langle apq, apq\rangle & \langle aq, aq\rangle & \langle ap, ap\rangle \\ \langle \varnothing, apq\rangle & \langle a, aq\rangle & \langle a, ap\rangle \end{array}}$$

Both $\langle apq, apq\rangle$ and $\langle aq, aq\rangle$ are considered, whereas $\langle ap, ap\rangle$ is not, because of $\langle a, ap\rangle$. $\langle apq, apq\rangle$ and $\langle aq, aq\rangle$ match w.r.t. $Y = \{a, q\}$, but none of them is least w.r.t. their witnesses modulo p. Hence $\langle P_2, \{p\}\rangle$ satisfies Ω.

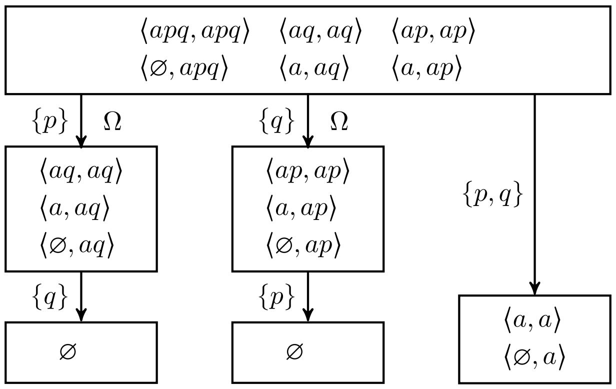
Corresponding to the Ω criterion the classes $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ specify the HT-models of the forgetting result. It was shown that $\mathsf{F_{SP}}$ satisfies $(\mathbf{SP})_{\langle P,V\rangle}$ for all instances $\langle P, V\rangle$ that do not satisfy Ω. Moreover, in the case where Ω is satisfied, $\mathsf{F_{SP}}$ still exhibits desirable behavior, such as satisfying $(\mathbf{SI})$ and $(\mathbf{wC})$, two of three characterizing criterions of $(\mathbf{SP})$. $\mathsf{F_R}$ on the other hand always satisfies $(\mathbf{sC})$ and $(\mathbf{SI})$, which makes it an ideal choice, if no new answer sets should be created (Gonçalves et al. 2017).

The classes $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ are are defined as follows:

$$\begin{aligned} \mathsf{F_R} \coloneqq \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X, Y\rangle \mid Y \subseteq \Sigma(P)\backslash V \wedge \\ X \in \bigcup \mathcal{R}^{Y}_{\langle P,V\rangle}\},\ \text{for all programs } P \text{ and } V \subseteq \Sigma\}, \\ \mathsf{F_{SP}} \coloneqq \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X, Y\rangle \mid Y \subseteq \Sigma(P)\backslash V \wedge \\ X \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}\},\ \text{for all programs } P \text{ and } V \subseteq \Sigma\}. \end{aligned}$$

where $\mathsf{F_R}$ collects the union of all witnesses modulo V of the considered models, and $\mathsf{F_{SP}}$ collects their intersection.

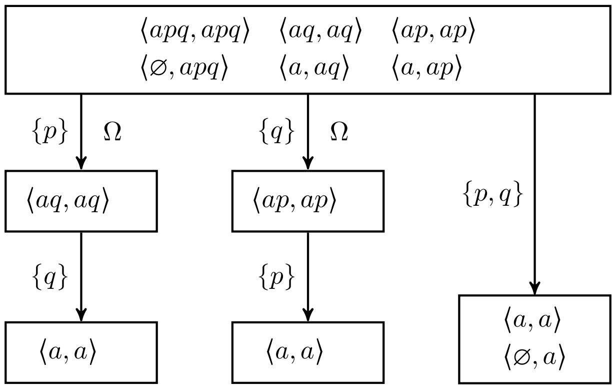
Example 3. *We assume $\mathsf{f} \in \mathsf{F_R}$. The following graph contains programs as nodes, represented by their respective set of HT-models. Two nodes P_i and P_j are connected by an edge labeled V, iff $\mathsf{f}(P_i, V) = P_j$. An edge from P_i to P_j is additionally labeled with Ω, iff $\langle P_i, V\rangle$ satisfies Ω. The program P_2 from Exm. 2 is in the topmost box.*



If p is forgotten from P_2, then $Rel^{\{a,q\}}_{\langle P_2,\{p\}\rangle} = \{\varnothing, \{p\}\}$. Hence $\langle aq, aq\rangle \models \mathsf{f}(P_2, \{p\})$. Since $\langle a, aq\rangle \models P_2$, and $\langle \varnothing, apq\rangle \models P_2$, we get $\langle a, aq\rangle \models \mathsf{f}(P_2, \{p\})$ and $\langle \varnothing, aq\rangle \models \mathsf{f}(P_2, \{p\})$. For all $Y \subseteq \Sigma(P_2)\backslash V$, s.t. $Y \neq \{a, q\}$, we have $Rel^{Y}_{\langle P_2,\{p\}\rangle} = \varnothing$, which entails $\langle Y, Y\rangle \not\models \mathsf{f}(P_2, \{p\})$

[3]Given a set of HT-interpretations M, satisfying $\langle X, Y\rangle \in M \Rightarrow \langle Y, Y\rangle \in M$, there exists a program P with $\mathcal{HT}(P) = M$ (Cabalar and Ferraris 2007). In the following examples each set of models represents the program that can be derived by the counter-models construction.

Example 4. *Now assume $\mathsf{f} \in \mathsf{F_{SP}}$ instead, and the same graphical representation of forgetting as in Exm. 3. When p is forgotten from P_2, then $\langle aq, aq\rangle \models \mathsf{f}(P_2, \{p\})$, for the same reason as in Exm. 3, since $Rel^{\{a,q\}}_{\langle P_2,\{p\}\rangle} \neq \varnothing$, but, since $\langle ap, apq\rangle \not\models P_2$, and $\langle a, apq\rangle \not\models P_2$, we get $\langle a, aq\rangle \not\models \mathsf{f}(P_2, \{p\})$. In addition, $\langle q, aq\rangle \not\models P_2$, and $\langle \varnothing, aq\rangle \not\models P_2$ entail $\langle \varnothing, aq\rangle \not\models \mathsf{f}(P_2, \{p\})$. As before, for all $Y \subseteq \Sigma(P_2)\backslash V$, s.t. $Y \neq \{a, q\}$, we have $Rel^{Y}_{\langle P_2,\{p\}\rangle} = \varnothing$, which entails $\langle Y, Y\rangle \not\models \mathsf{f}(P_2, \{p\})$*



Whereas arbitrary sets of atoms can be forgotten, based on the HT-models defined by $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ via the counter-models construction (Cabalar and Ferraris 2007) – a procedure not unlike determining a formula in CNF based on classical models – purely syntactical manipulations following $\mathsf{F_{SP}}$ semantics have only been defined to forget single atoms, and for $\mathsf{F_R}$ up until now there are none whatsoever.

Examples 3 and 4 witness that in general atoms cannot be forgotten iteratively by the semantics of $\mathsf{F_R}$ and $\mathsf{F_{SP}}$, which, for one, proves f_{SP} unfit to correctly forget multiple atoms.

3 Syntactic Tools

Defining syntactic forgetting operators obeying the semantics of $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ comes down to two points: i) identifying which rules an interpretation is contradicted by, and ii) constructing rules that contradict interpretations as semantically defined. In this chapter we collect and introduce essential tools needed to do these two things.

Firstly, to avoid complications and unnecessary calculations caused by redundant (parts of) rules, we bring programs into a normal formal as has been done in previous related work (Inoue and Sakama 1998; 2004; Cabalar, Pearce, and Valverde 2007; Slota and Leite 2011; Knorr and Alferes 2014). The construction used by Berthold et al. in 2019 applies to programs with disjunctive heads and double negation, and eliminates non-minimal rules (Brass and Dix 1999).

First off, there is a number of redundancies that can be removed easily, by considering each rule individually.

Example 5. *Consider the following rules:*

$$\begin{array}{ll} a \vee b \leftarrow a. & a \vee b \leftarrow not\, a. \\ \quad b \leftarrow a, not\, a. & \quad b \leftarrow a, not\, not\, a. \\ \quad b \leftarrow not\, a, not\, not\, a. & \end{array}$$

The three rules in the left column are tautological, they are satisfied by any interpretation, and can thus be disregarded.

The two rules in the right column contain redundant literals. An interpretation satisfying the body of the top rule, i.e. $not\, a$ cannot at the time satisfy a in the rule's head. Therefore a can be omitted. For the bottom rule it holds that any interpretation satisfying a in the rule's body also satisfies $not\, not\, a$, hence $not\, not\, a$ can be omitted.

Further, some rules are weaker (have more models) than other rules. Such *non-minimal* rules can also be disregarded. We broaden the notion of non-minimality, that has been used by Berthold et al., by adapting subsumption of rules (Cabalar, Pearce, and Valverde 2007).

Definition 1. *Given two rules r and s, s* subsumes *r, in symbols $s \leq r$, iff:*

1. $H(s) \subseteq H(r) \cup B^-(r)$,
2. $B^+(s) \subseteq B^+(r) \cup B^{--}(r)$,
3. $B^-(s) \subseteq B^-(r)$,
4. $B^{--}(s) \subseteq B^{--}(r) \cup B^+(r)$, *and*
5. $B^+(s) \cap B^{--}(r) = \varnothing$ *or* $H(s) \cap H(r) = \varnothing$.

Subsumption is such that, if a rule s subsumes a rule r, then s has no more HT-models than r. In other words it contains at least as much information, and r is redundant.

A rule r is *minimal in P*, iff it is not (strictly) subsumed by another rule s in P, i.e. iff $\neg\exists s \in P : s \leq r \wedge r \nleq s$.

Example 6. *Consider the following program P_6:*

$$a \leftarrow not\, not\, b. \qquad a \leftarrow c, not\, not\, b. \qquad \leftarrow b, not\, a.$$

In (Berthold et al. 2019), only the middle rule is non-minimal in P_6, since it contains more literals than the rule on the left. In this paper, we additionally deem the right hand rule non-minimal, since it is subsumed by the left hand rule.

Programs in normal form do not contain redundant parts of rules, non-minimal rules, nor tautologies, where a rule r is *tautological* iff $H(r) \cap B^+(r) \neq \varnothing$, or $B^+(r) \cap B^-(r) \neq \varnothing$, or $B^-(r) \cap B^{--}(r) \neq \varnothing$.

Definition 2. *Let P be a program. We say that P is in* normal form *iff the following conditions hold:*

- *P does not contain tautological rules;*
- *if $a \in H(r)$, then $not\, a \notin B(r)$;*
- *if $a \in B(r)$, then $not\, not\, a \notin B(r)$;*
- *all rules in P are minimal.*

Any program can be transformed into a strongly equivalent program in normal form.

Definition 3. *Let P be a program. The normal form $NF(P)$ is obtained from P by:*

1. *removing all tautological rules;*
2. *removing all atoms a from $B^{--}(r)$ in the remaining rules r, whenever $a \in B^+(r)$;*
3. *removing all atoms a from $H(r)$ in the remaining rules r, whenever $a \in B^-(r)$;*
4. *removing from the resulting program all rules that are not minimal.*

Proposition 1. *Let P be a program. Then, $NF(P)$ is in normal form and is strongly equivalent to P.*

The first three steps of the normal form construction only require checking each rule by itself. Checking whether a rule in a program P is non-minimal requires a comparison to each other rule in P. Hence the normal form construction has polynomial complexity.

Proposition 2. *Let P be a program. Then, the normal form $NF(P)$ can be computed in PTIME.*

The q-exclusion notation is short hand to remove an atom.

Definition 4 (q-exclusion)**.** *Given an atom $q \in \Sigma$, and a set of literals L, a rule r and a program P over Σ, the q-exclusions are $L^{\setminus q} := L \setminus \{q, not\, q, not\, not\, q\}$, $r^{\setminus q} := H^{\setminus q}(r) \leftarrow B^{\setminus q}(r)$ and $P^{\setminus q} := \{r^{\setminus q} \mid r \in P\}$.*

A rule in a program in normal form can only contain a given atom q in six different ways. We define a partition of a program along these occurrences.

Definition 5. *Given a program P in normal form over Σ and an atom $q \in \Sigma$, P is partitioned according to the occurrence of q, i.e. $\mathsf{occ}(P,q) := \langle R, R_0, R_1, R_2, R_3, R_4\rangle$, where*

$$\begin{aligned}
R &:= \{r \in P \mid q \notin \Sigma(r)\}\\
R_0 &:= \{r \in P \mid q \in B(r)\}\\
R_1 &:= \{r \in P \mid not\ q \in B(r)\}\\
R_2 &:= \{r \in P \mid not\ not\ q \in B(r), q \notin H(r)\}\\
R_3 &:= \{r \in P \mid not\ not\ q \in B(r), q \in H(r)\}\\
R_4 &:= \{r \in P \mid not\ not\ q \notin B(r), q \in H(r)\}
\end{aligned}$$

By partitioning a program along the occurrences of an atom, we are able to identify some first correspondences between rules and models – if an interpretation is not a model of a program, it is contradicted by a rule with a certain occurrence of q.

Proposition 3. *Given a program P in normal form over Σ, $X \subseteq Y \subseteq \Sigma$, and an atom $q \in \Sigma$, with $q \notin Y$, and $\mathsf{occ}(P,q) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$. Then the following equivalencies hold:*

$$\begin{aligned}
\langle X,Y\rangle \not\models P &\Leftrightarrow \exists r \in R \cup R_1 \cup R_4 : \langle X,Y\rangle \not\models r\\
\langle Xq,Yq\rangle \not\models P &\Leftrightarrow \exists r \in R \cup R_0 \cup R_2 : \langle Xq,Yq\rangle \not\models r\\
\langle X,Yq\rangle \not\models P &\Leftrightarrow \exists r \in R \cup R_2 \cup R_3 \cup R_4 : \langle X,Yq\rangle \not\models r
\end{aligned}$$

The next construction conversely identifies, which interpretations *are* models of a program.

The *as-dual* construction (Berthold et al. 2019) generalizes constructions that collect sets of conjunctions of literals aiming to replace negated occurrences of a literal (Eiter and Wang 2008; Knorr and Alferes 2014). It takes into account disjunctions and double negations, and can be used more generally to collect literals describing possible ways to satisfy rules independently of a given atom. Applying the construction to the rules containing q in the head, the as-dual describe exactly how q can be falsified.

Example 7. *Consider the following program P_7:*

$$a \vee q \leftarrow b. \qquad\qquad q \leftarrow c.$$

If b does not hold, q cannot be derived from the first rule. Likewise, q cannot be derived from the first rule, if a is not false. Further, q cannot be derived from the second rule, if c does not hold. Hence q is falsified by the following two sets of literals: $\{not\, b, not\, c\}$ and $\{not\, not\, a, not\, c\}$.

Definition 6. *Given a program $P = \{r_1, \ldots, r_n\}$ over Σ and an atom $q \in \Sigma$, then:*

$$\mathcal{D}^q_{as}(P) := \{\{l_1, \ldots, l_n\} \mid l_i \in not\, B^{\setminus q}(r_i) \cup not\, not\, H^{\setminus q}(r_i), 1 \leq i \leq n\},$$

where, for a set S of literals, $not\, S = \{not\, s \mid s \in S\}$ and $not\, not\, S = \{not\, not\, s \mid s \in S\}$, where, for $p \in \Sigma$, we assume the simplification $not\, not\, not\, p = not\, p$ and $not\, not\, not\, not\, p = not\, not\, p$.

The as-dual can also be applied more generally, to any program P to gain sets of literals proving P to be satisfied independently of an atom q.

Example 8. *Let P_8 consist of the following two rules:*

$$a \leftarrow q, b. \qquad c \leftarrow d, not\, not\, q.$$

Then each set in $\mathcal{D}^q_{as}(P_8)$ =

$$\{ \quad \{not\, not\, a, not\, not\, c\}, \quad \{not\, b, not\, not\, c\}, \quad \{not\, not\, a, not\, d\}, \quad \{not\, b, not\, d\} \quad \}$$

describes how P_8 can be satisfied independently of q.

By applying the as-dual to certain subsets of a program, we are able to construct rules that contradict some models of a program.

Proposition 4. *Given a program P in normal form over Σ, $Y \subseteq \Sigma$, and an atom $q \in \Sigma$, with $q \notin Y$, and $\mathsf{occ}(P, q) = \langle R, R_0, R_1, R_2, R_3, R_4 \rangle$. Then the following implications hold:*

$$\langle Y, Y \rangle \models P \Rightarrow \exists D \in \mathcal{D}^q_{as}(R_1 \cup R_4) : \langle Y, Y \rangle \not\models \leftarrow D$$
$$\langle Yq, Yq \rangle \models P \Rightarrow \exists D \in \mathcal{D}^q_{as}(R_0 \cup R_2) : \langle Yq, Yq \rangle \not\models \leftarrow D$$
$$\langle Y, Yq \rangle \models P \Rightarrow \exists D \in \mathcal{D}^q_{as}(R_3 \cup R_4) : \langle Y, Yq \rangle \not\models \leftarrow D$$

In the case that $R = \varnothing$ the first and second implication hold in both directions.

We define the product of rules (resp. of programs) and double negation of rules to be able to construct rules (resp. programs) that unite the models of two given rules (resp. of programs).

Definition 7 (Product of Rules). *Let r_1 and r_2 be rules. Their* product *$r_1 \times r_2$, is defined as:*

$$H(r_1) \cup H(r_2) \leftarrow B(r_1) \cup B(r_2)$$

Proposition 5. *Let r_1, r_2 be rules over Σ, and $X \subseteq Y \subseteq \Sigma$,*

$$Y \models r_1 \times r_2 \Leftrightarrow Y \models r_1 \vee Y \models r_2$$
$$X \models \{r_1 \times r_2\}^Y \Leftrightarrow X \models \{r_1\}^Y \vee X \models \{r_2\}^Y$$

Definition 8 (Product of Programs). *Let P_1 and P_2 be programs. Their* product *$P_1 \times P_2$, is defined as:*

$$\{r_1 \times r_2 \mid r_1 \in P_1 \wedge r_2 \in P_2\}$$

Proposition 6. *Let P_1, P_2 be programs over Σ, and $X \subseteq Y \subseteq \Sigma$,*

$$Y \models P_1 \times P_2 \Leftrightarrow Y \models P_1 \vee Y \models P_2$$
$$X \models (P_1 \times P_2)^Y \Leftrightarrow X \models P_1^Y \vee X \models P_2^Y$$

In general, $\mathcal{HT}(P_1) \cup \mathcal{HT}(P_2) = \mathcal{HT}(P_1 \times P_2)$ does not hold, e.g. $\langle \varnothing, ab \rangle \not\models \{a \leftarrow\}$ and $\langle \varnothing, ab \rangle \not\models \{\leftarrow b\}$, but $\langle \varnothing, ab \rangle \models \{a \leftarrow b\}$. To get around this, we define the double negation of a rule to be able to reason about, whether the second item Y of an HT-model $\langle X, Y \rangle$ is a classical model, and therefore whether the corresponding total model $\langle Y, Y \rangle$ is a potential answer set.

Definition 9. *Given a rule r, we define the* double negation *of r, i.e. $not\, not\, r$, as:*

$$not\, not\, r := \leftarrow not\, H(r) \cup not\, not\, B(r)$$

Proposition 7. *Given a rule r over Σ, and $X \subseteq Y \subseteq \Sigma$, the following statement holds:*

$$Y \models r \Leftrightarrow \langle X, Y \rangle \models not\, not\, r$$

Using the double negation, for example, we can construct $P = \{a \leftarrow\} \times \{not\, not\, \leftarrow b\} = \{a \leftarrow not\, not\, b\}$, such that $\langle \varnothing, ab \rangle \not\models P$, and $\mathcal{HT}(P) = \mathcal{HT}(\{a \leftarrow\}) \cup \mathcal{HT}(\{\leftarrow b\})$.

Any rule r subsumes $not\, not\, r$. In order not to lose double negated rules, we therefore restrict the normal form construction NF to its first three steps, denoted nf, when necessary (in Def. 10 and Def. 11).

All the aforementioned correspondences between models and rules remain, when an atom q is removed from a rule as well as from an interpretation.

Proposition 8. *Given a program P in normal form over Σ, $X \subset Y \subseteq \Sigma$, and an atom $q \in \Sigma$, with $q \notin Y$, and $\mathsf{occ}(P, q) = \langle R, R_0, R_1, R_2, R_3, R_4 \rangle$. Then the following equivalences hold:*

$$\langle Y, Y \rangle \not\models P \Leftrightarrow \exists r \in R_1 \cup R_4 : \langle Y, Y \rangle \not\models not\, not\, r^{\setminus q} \vee \exists r \in R : \langle Y, Y \rangle \not\models r$$
$$\langle X, Y \rangle \not\models P \Leftrightarrow \exists r \in R \cup R_1 \cup R_4 : \langle X, Y \rangle \not\models r^{\setminus q}$$
$$\langle Yq, Yq \rangle \not\models P \Leftrightarrow \exists r \in R_0 \cup R_2 : \langle Y, Y \rangle \not\models not\, not\, r^{\setminus q} \vee \exists r \in R : \langle Y, Y \rangle \not\models r$$
$$\langle Y, Yq \rangle \not\models P \Leftrightarrow \langle Yq, Yq \rangle \not\models P \vee \exists r \in R_3 \cup R_4 : \langle Y, Y \rangle \not\models not\, not\, r^{\setminus q}$$
$$\langle Y, Yq \rangle \models P \Leftrightarrow \langle Yq, Yq \rangle \models P \wedge \exists D \in \mathcal{D}^q_{as}(R_3 \cup R_4) : \langle Y, Y \rangle \not\models \leftarrow D$$
$$\langle Xq, Yq \rangle \not\models P \Leftrightarrow \exists r \in R \cup R_0 \cup R_2 : \langle X, Y \rangle \not\models r^{\setminus q}$$
$$\langle X, Yq \rangle \not\models P \Leftrightarrow \langle Yq, Yq \rangle \not\models P \vee \exists r \in R \cup R_2 \cup R_3 \cup R_4 : \langle X, Y \rangle \not\models r^{\setminus q}$$

If additionally $R = \varnothing$, then

$$\langle Y, Y \rangle \models P \Leftrightarrow \exists D \in \mathcal{D}^q_{as}(R_1 \cup R_4) : \langle Y, Y \rangle \not\models \leftarrow D$$
$$\langle Yq, Yq \rangle \models P \Leftrightarrow \exists D \in \mathcal{D}^q_{as}(R_0 \cup R_2) : \langle Y, Y \rangle \not\models \leftarrow D$$

For all $r_2 \in R_2$:

$$\langle Yq, Yq \rangle \not\models r_2 \Leftrightarrow \langle Y, Yq \rangle \not\models r_2, \text{ and}$$
$$\langle Xq, Yq \rangle \not\models r_2 \Leftrightarrow \langle X, Yq \rangle \not\models r_2.$$

The rules identified in Prop. 8 constitute the essential building blocks, used to define f_R and f^*_{SP}.

4 Relativized Forgetting

Now, we define f_R, such that $\mathsf{f}_R \in \mathsf{F_R}$. To that end, for each $A \subseteq V$, we define an auxiliary operator f_R^A that determines, whether A is relevant for a given $Y \subseteq \Sigma(P)\backslash V$ ($A \in Rel^Y_{\langle P,V\rangle}$), and whether for this A there exists a witness $\langle XA'', YA\rangle$, with $A'' \subseteq A$. We construct these $\mathsf{f}_R^A(P,V)$, such that they contradict $\langle Y,Y\rangle$, iff A is not relevant w.r.t. Y, i.e. $\langle Y,Y\rangle \not\models \mathsf{f}_R^A(P,V) \Leftrightarrow A \notin Rel^Y_{\langle P,V\rangle}$, and such that, if A is relevant, $\mathsf{f}_R^A(P,V)$ satisfies all the witnesses modulo V that belong to $\langle YA, YA\rangle$, i.e. $\langle X,Y\rangle \models \mathsf{f}_R^A(P,V) \Leftrightarrow A \in Rel^Y_{\langle P,V\rangle} \wedge \exists A'' \subseteq A : \langle XA'', YA\rangle \models P$.

$\mathsf{F_R}$ is such that $\mathcal{HT}(\mathsf{f}_R(P,V)) = \bigcup_{A\subseteq V} \mathcal{HT}(\mathsf{f}_R^A(P,V))$, so we define f_R as the conjunction all f_R^A by $\times$, to construct a program that unites all of their the models.

The operators f_R^A are defined inductively. We therefore first define them for one atom. If one atom q is forgotten from a program P, there are two subsets of $\{q\}$. In other words, for any $Y \subseteq \Sigma$, we have $Rel^Y_{\langle P,\{q\}\rangle} \subseteq \{\varnothing, \{q\}\}$.

For forgetting one atom q it holds that $\{q\} \in Rel^Y_{\langle P,\{q\}\rangle}$, iff $\langle Yq, Yq\rangle \models P$ and $\langle Y, Yq\rangle \not\models P$. Using Prop. 8, we are able to construct rules, contradicting $\langle Y,Y\rangle$, iff $\{q\} \notin Rel^Y_{\langle P,\{q\}\rangle}$, and otherwise, if $\{q\}$ is relevant, satisfying $\langle X,Y\rangle$ iff $\langle Xq, Yq\rangle \models P$ or $\langle X, Yq\rangle \models P$.

Definition 10 (f_R^+). *Given a program P in normal form over Σ and $q \in \Sigma$ s.t.* $\mathsf{occ}(P,q) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$. *Then:*

$$\mathsf{f}_R^+(P,q) := nf(A \cup B \cup C \cup D)$$

where:

$$A := \{\leftarrow D \mid D \in \mathcal{D}^q_{as}(R_3 \cup R_4)\}$$
$$B := \{not\,not\, r^{\backslash q} \mid r_0 \in R_0 \cup R_2\}$$
$$C := \{r^{\backslash q} \mid r \in R \cup R_2\}$$
$$D := \{(r_0 \times r')^{\backslash q} \mid r_0 \in R_0, r' \in R_3 \cup R_4\}$$

The four sets A, B, C and D making up $\mathsf{f}_R^+(P,q)$ directly correspond to the conclusions in Prop. 8. If $\langle Yq, Yq\rangle \not\models P$, a rule $r \in B \cup C$ contradicts $\langle Y,Y\rangle$. If $\langle Y, Yq\rangle \models P$, then a rule $r \in A$ contradicts $\langle Y,Y\rangle$. Otherwise, if both $\langle Xq, Yq\rangle \not\models P$ and $\langle X, Yq\rangle \not\models P$, then a rule in $C \cup D$ contradicts $\langle X,Y\rangle$.

Proposition 9. *Given a program P over Σ, an atom $q \in \Sigma$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma\backslash\{q\}$, then:*

$$\langle Y,Y\rangle \models \mathsf{f}_R^+(P,q) \Leftrightarrow \{q\} \in Rel^Y_{\langle P,\{q\}\rangle}$$

If $\{q\} \in Rel^Y_{\langle P,\{q\}\rangle}$, then:

$$\langle X,Y\rangle \models \mathsf{f}_R^+(P,q) \Leftrightarrow \langle Xq, Yq\rangle \models P \vee \langle X, Yq\rangle \models P$$

It holds that $\varnothing \in Rel^Y_{\langle P,\{q\}\rangle} \Leftrightarrow \langle Y,Y\rangle \models P$. Once again, we utilize Prop. 8 to construct an auxiliary program.

Definition 11 (f_R^-). *Given a program P in normal form over Σ and $q \in \Sigma$ s.t.* $\mathsf{occ}(P,q) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$. *Then:*

$$\mathsf{f}_R^-(P,q) := nf(A \cup B)$$

where:

$$A := \{r'^{\backslash q} \mid r' \in R \cup R_1 \cup R_4\}$$
$$B := \{not\,not\, r'^{\backslash q} \mid r' \in R_1 \cup R_4\}$$

Proposition 10. *Given a program P over Σ, an atom $q \in \Sigma$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma\backslash\{q\}$, then:*

$$\langle Y,Y\rangle \models \mathsf{f}_R^-(P,q) \Leftrightarrow \varnothing \in Rel^Y_{\langle P,\{q\}\rangle}$$

If $\varnothing \in Rel^Y_{\langle P,\{q\}\rangle}$, then:

$$\langle X,Y\rangle \models \mathsf{f}_R^-(P,q) \Leftrightarrow \langle X,Y\rangle \models P$$

Now, using f_R^+ and f_R^- as building blocks, we inductively construct f_R^A for V of arbitrary sizes. We assume an arbitrary ordering on the signature Σ to be able to fix a concrete forgetting result. However, the subsequent propositions are independent of this ordering and hence there is no loss of generality. We leave the question about whether the given ordering has a meaningful impact on the forgetting result for future studies – the resulting program $\mathsf{f}_R(P,V)$ is correct anyhow.

Definition 12 (f_R^A). *Let P be a program over Σ, and $A \subseteq \{q_1, q_2, \ldots, q_n\} = V \subseteq \Sigma$, s.t. $0 < n$, then:*

$$\mathsf{f}_R^A(P,\varnothing) := P$$
$$\mathsf{f}_R^A(P,V) := \mathsf{f}_R^{\otimes_n}(\mathsf{f}_R^{A^{\backslash q_n}}(P, V\backslash\{q_n\}), q_n)$$

where:

$$\mathsf{f}_R^{\otimes_n} := \begin{cases} \mathsf{f}_R^+, & \textit{if } q_n \in A \\ \mathsf{f}_R^-, & \textit{otherwise} \end{cases}$$

For $|V| = n$ the auxiliary functions f_R^A consist of n nested functions calls f_R^+ or f_R^-. For example, given a program P, $V = \{p,q\}$ and $A = \{p\}$, $\mathsf{f}_R^A(P,V) = \mathsf{f}_R^-(\mathsf{f}_R^+(P,p),q)$. Prop. 11 generalizes Prop. 9 and Prop. 10, and can be proven by induction.

Proposition 11. *Given a program P over Σ, and sets X, Y, A and V, s.t. $A \subseteq V \subseteq \Sigma$, and $X \subseteq Y \subseteq \Sigma\backslash V$, then*

$$\langle Y,Y\rangle \models \mathsf{f}_R^A(P,V) \Leftrightarrow A \in Rel^Y_{\langle P,V\rangle}$$

If $A \in Rel^Y_{\langle P,V\rangle}$, then:

$$\langle X,Y\rangle \models \mathsf{f}_R^A(P,V) \Leftrightarrow \exists A'' \subseteq A : \langle XA'', YA\rangle \models P$$

$\mathsf{F_R}$ is such that all considered total models and their witnesses (modulo V) are collected. In other words, for any $\mathsf{f} \in \mathsf{F_R}$ we have

$$\mathcal{HT}(\mathsf{f}(P,V)) = \bigcup_{A\subseteq V} \mathcal{HT}(\mathsf{f}_R^A(P,V))$$

Therefore, we define the result of forgetting f_R, by using the $\times$ operator.

Definition 13 (f_R). *Let P be a program over Σ in normal form and $V \subseteq \Sigma$.*

$$\mathsf{f}_R(P,V) := NF(\bigtimes_{A\subseteq V} \mathsf{f}_R^A(P,V))$$

Using Prop. 6, Prop. 11 and the fact that any $\langle Y,Y\rangle$ is contradicted by a double negated rule, we arrive at:

Theorem 1. $\mathsf{f}_R \in \mathsf{F_R}$

5 Forgetting with Strong Persistence

Next, we define f^*_{SP}, such that $\mathsf{f}^*_{SP} \in \mathsf{F}_{\mathsf{SP}}$. For any program P and set V, the forgetting result $\mathsf{f}^*_{SP}(P,V)$ has a subset of the models of $\mathsf{f}_R(P,V)$, but the same total models, i.e.:

$$\langle Y,Y\rangle \models \mathsf{f}^*_{SP}(P,V) \Leftrightarrow \langle Y,Y\rangle \models \mathsf{f}_R(P,V).$$

In other words, $\mathsf{f}^*_{SP}(P,V)$ and $\mathsf{f}_R(P,V)$ only differ in their witnesses. We hence construct $\mathsf{f}^*_{SP}(P,V)$ by adding rules to $\mathsf{f}_R(P,V)$ that contradict the remaining witnesses. We call these additional rules $\mathsf{f}_W(P,V)$.

Example 9. *Consider P_2 from earlier, where*

$$\mathcal{HT}(\mathsf{f}_R(P_2,\{p\})) = \{\langle aq,aq\rangle, \langle a,aq\rangle, \langle \varnothing,aq\rangle\}.$$

$\mathsf{f}_W(P_2,\{p\})$ *specifically contradicts $\langle a,aq\rangle$ and $\langle \varnothing,aq\rangle$, s.t.*

$$\mathcal{HT}(\mathsf{f}_R(P_2,\{p\}) \cup \mathsf{f}_W(P_2,\{p\})) = \{\langle aq,aq\rangle\}.$$

As for f_R, we define f_W by the help of auxiliary operators for every $A \subseteq V$. Each $\mathsf{f}^A_W(P,V)$ contradicts $\langle X,Y\rangle$, i.e. $\langle X,Y\rangle \not\models \mathsf{f}^A_W(P,V)$, iff $A \in Rel^Y_{\langle P,V\rangle}$ and $\forall A'' \subseteq A : \langle XA'',YA\rangle \not\models P$, where $X \subseteq Y \subseteq \Sigma$.

Also as before, the auxiliary operators f^A_W are defined inductively by nested calls of f^+_W and f^-_W.

Definition 14 (f^+_W)**.** *Given a program P in normal form over Σ and $q \in \Sigma$ s.t. $\mathsf{occ}(P,q) = \langle R,R_0,R_1,R_2,R_3,R_4\rangle$, then:*

$$\mathsf{f}^+_W(P,V) := NF(A \cup B)$$

where:

$$\begin{aligned} A :=\ &\{(r_0 \times r' \times not\,not\,r)^{\backslash q} \times\ \leftarrow D \mid \\ &r_0 \in R_0,\ r,r' \in R_3 \cup R_4, D \in \mathcal{D}^q_{as}(R_0 \cup R_2)\} \\ B :=\ &\{(r \times not\,not\,r')^{\backslash q} \times\ \leftarrow D \mid \\ &r \in R \cup R_2, r' \in R_3 \cup R_4, D \in \mathcal{D}^q_{as}(R_0 \cup R_2)\} \end{aligned}$$

The operator f^+_W creates a rule contradicting $\langle X,Y\rangle$ iff $\{q\} \in Rel^Y_{\langle P,V\rangle}$ and $\langle Xq,Yq\rangle, \langle X,Yq\rangle \notin \mathcal{HT}(P)$. According to Prop. 8, $\{q\} \in Rel^Y_{\langle P,V\rangle}$ entails the existence of $D \in \mathcal{D}^q_{as}(P)$ and $r \in R_3 \cup R_4$, s.t. $\langle X,Y\rangle \not\models\ \leftarrow D$ and $\langle X,Y\rangle \not\models not\,not\,r^{\backslash q}$. Also, $\langle Xq,Yq\rangle \not\models P \wedge \langle X,Yq\rangle \not\models P$ implies the existence of an $r \in R \cup R_2$, s.t. $\langle X,Y\rangle \not\models r^{\backslash q}$, or $r_0 \in R_0$, and $r' \in R_3 \cup R_4$, s.t. $\langle X,Y\rangle \not\models r_0^{\backslash q} \times r'^{\backslash q}$. Hence the sets A and B.

Proposition 12. *Given a program P over Σ, an atom $q \in \Sigma$, and sets X and Y, s.t. $X \subseteq Y \subseteq \Sigma\backslash\{q\}$, then:*

$$\begin{aligned} &\{q\} \in Rel^Y_{\langle P,\{q\}\rangle} \wedge \langle Xq,Yq\rangle \not\models P \wedge \langle X,Yq\rangle \not\models P \\ &\Leftrightarrow \langle X,Y\rangle \not\models \mathsf{f}^+_W(P,q) \end{aligned}$$

Definition 15 (f^-_W)**.** *Given a program P in normal form over Σ and $q \in \Sigma$ s.t. $\mathsf{occ}(P,q) = \langle R,R_0,R_1,R_2,R_3,R_4\rangle$, then:*

$$\mathsf{f}^-_W(P,V) :=\ NF(A)$$

where:

$$A := \{r'^{\backslash q} \times\ \leftarrow D \mid r' \in R \cup R_1 \cup R_4, D \in \mathcal{D}^q_{as}(R_1 \cup R_4)\}$$

Again, using Prop. 8, $\varnothing \in Rel^Y_{\langle P,V\rangle}$ implies the existence of $D \in \mathcal{D}^q_{as}(P)$ s.t. $\langle X,Y\rangle \not\models\ \leftarrow D$, and if $\langle X,Y\rangle \not\models P$ then there is a rule $r \in R \cup R_1 \cup R_4$ s.t. $\langle X,Y\rangle \not\models r^{\backslash q}$.

Proposition 13. *Given a program P over Σ, an atom $q \in \Sigma$, and sets X and Y, s.t. $X \subseteq Y \subseteq \Sigma\backslash\{q\}$, then:*

$$\begin{aligned} &\varnothing \in Rel^Y_{\langle P,\{q\}\rangle} \wedge \langle X,Y\rangle \not\models P \\ &\Leftrightarrow \langle X,Y\rangle \not\models \mathsf{f}^-_W(P,q) \end{aligned}$$

Example 10. *Recall P_1 from the beginning,*

$$\begin{aligned} \mathsf{f}^+_W(P_1,\{p\}) &= \{a \leftarrow q, not\,not\,a\} \cup \varnothing \\ \mathsf{f}^-_W(P_1,\{p\}) &= \{q \leftarrow not\,not\,q\} \end{aligned}$$

Now, the operators f^A_W are defined inductively.

Definition 16 (f^A_W)**.** *Let P be a program over Σ, and $A \subseteq \{q_1,q_2,\ldots,q_n\} = V \subseteq \Sigma$, s.t. $0 < n$, then:*

$$\begin{aligned} \mathsf{f}^A_W(P,\varnothing) &:= P \\ \mathsf{f}^A_W(P,V) &:= \mathsf{f}^{\otimes_n}_W(\mathsf{f}^{A\backslash q_n}_W(P\backslash R, V\backslash\{q_n\}), q_n) \cup R \end{aligned}$$

where:

$$\mathsf{f}^{\otimes_n}_W := \begin{cases} \mathsf{f}^+_W, & \textit{if } q_n \in A \\ \mathsf{f}^-_W, & \textit{otherwise} \end{cases}$$

$$R := \{r \in P \mid V \cap \Sigma(r) = \varnothing\}$$

Prop. 14 generalizes Prop. 12 and Prop. 13, and can be proven by induction. For technical reasons we assume R to be empty. This assumption can be made without loss of generality, since F_{SP} satisfies (**SI**).

Proposition 14. *Given a program P over Σ, and sets X, Y, A and V, s.t. $A \subseteq V \subseteq \Sigma$, $X \subseteq Y \subseteq \Sigma\backslash V$, and $R = \{r \in P \mid V \cap \Sigma(r) = \varnothing\} = \varnothing$, then:*

$$\begin{aligned} &A \in Rel^Y_{\langle P,V\rangle} \wedge \forall A'' \subseteq A : \langle XA'',YA\rangle \not\models P \\ &\Leftrightarrow \langle X,Y\rangle \not\models \mathsf{f}^A_W(P,V) \end{aligned}$$

Now, $\mathsf{f}_W(P,V)$ is the union of all $\mathsf{f}^A_W(P,V)$.

Definition 17 (f_W)**.** *Given a program P in normal form over Σ and $V \subseteq \Sigma$. Then:*

$$\mathsf{f}_W(P,V) := NF(\bigcup_{A \subseteq V} \mathsf{f}^A_W(P,V))$$

The union of programs satisfies the intersections of their models. Hence the next proposition.

Proposition 15. *Given a program P over Σ in normal form, three sets of atoms X, Y and V, s.t. $X \subseteq Y \subseteq \Sigma$, $V \subseteq \Sigma$, and $R = \{r \in P \mid V \cap \Sigma(r) = \varnothing\} = \varnothing$, then:*

$$\begin{aligned} &\exists A \in Rel^Y_{\langle P,V\rangle} \textit{ s.t. } \forall A'' \subseteq A. \langle XA'',YA\rangle \not\models P \\ &\Leftrightarrow \langle X,Y\rangle \not\models \mathsf{f}_W(P,V) \end{aligned}$$

We arrive at the end. f^*_{SP} is (the normal form of) the union of f_R and f_W, and a member of F_{SP}.

Definition 18 (f^*_{SP})**.** *Let P be a program over Σ in normal form and $V \subseteq \Sigma$.*

$$\mathsf{f}^*_{SP}(P,V) := NF(\mathsf{f}_W(P,V) \cup \mathsf{f}_R(P,V))$$

Theorem 2. $\mathsf{f}^*_{SP} \in \mathsf{F}_{\mathsf{SP}}$

6 Intuitions and Results

The operators f_R and f^*_{SP} and their components are defined methodologically, to match the semantics F_R and F_SP. Nonetheless, their results can be observed to bear some intuitive sense. In fact, they implement adjusted versions of all the derivation rules of f_{SP}. In other words, all the intuitions about f_{SP} apply to f_R and f^*_{SP}.

To make a comparison, we recall the definition of f_{SP}, rephrased to be more compact, and to match our notation using '×'. Notably, derivation rules **3a** and **7** have been merged into **3a**. This simplification is possible, due to the application of the normal form construction.

Definition 19 (f_{SP})**.** *Let P be a program in normal form over Σ, $q \in \Sigma$, and $\mathsf{occ}(P,q) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$, then $\mathsf{f}_{SP}(P,q) := NF(\boldsymbol{0} \cup \boldsymbol{1a} \cup \boldsymbol{2a} \cup \boldsymbol{3a} \cup \boldsymbol{1b} \cup \boldsymbol{2b} \cup \boldsymbol{3b} \cup \boldsymbol{4} \cup \boldsymbol{5} \cup \boldsymbol{6})$, where:*

0 $= R$

1a $= \{(r_0 \times r_4)^{\setminus q} \mid r_0 \in R_0, r_4 \in R_4\}$

2a $= \{(r_0 \times r_3)^{\setminus q} \times \mathit{not\,not}\, r'^{\setminus q} \mid r_0 \in R_0, r_3 \in R_3, r' \in R_1 \cup R_4\}$

3a $= \{(r_0 \times r_3 \times \mathit{not\,not}\, r'_3)^{\setminus q} \times \leftarrow D \mid r_0 \in R_0, r_3, r'_3 \in R_3, D \in \mathcal{D}^q_{as}(R_0 \cup R_2\})\}$

1b $= \{r_2^{\setminus q} \times r'^{\setminus q} \mid r_2 \in R_2, r' \in R_1 \cup R_4\}$

2b $= \{r_2^{\setminus q} \times \mathit{not\,not}\, r'^{\setminus q} \mid r_2 \in R_2, r' \in R_1 \cup R_4\}$

3b $= \{(r_2 \times \mathit{not\,not}\, r_3)^{\setminus q} \times \leftarrow D \mid r_2 \in R_2, r_3 \in R_3, D \in \mathcal{D}^q_{as}(R_0 \cup R_2\})\}$

4 $= \{r'^{\setminus q} \times \leftarrow D \mid r' \in R_1 \cup R_4, D \in \mathcal{D}^q_{as}(R_3 \cup R_4)\}$

5 $= \{r'^{\setminus q} \times \mathit{not\,not}\, r^{\setminus q} \mid r' \in R_1 \cup R_4, r \in R_0 \cup R_2\}$

6 $= \{r'^{\setminus q} \times \leftarrow D \mid r' \in R_1 \cup R_4, D \in \mathcal{D}^q_{as}(R_1 \cup R_4)\}$

In the case of forgetting one atom, f_R implements the derivation rules **0**, **1a**, **2a**, **1b**, **2b**, **4** and **5**. The operators f^+_R and f^-_R respectively gather half of each derivation rule. These halves are then conjoined by ×. The outputs of f^+_R (resp. f^-_R) are indicated in the definition above in blue (resp. red) text colors. For example, given that $r_0 \in R_0$ and $r_4 \in R_4$, f^+_R produces $(r_0 \times r_4)^{\setminus q}$, while f^-_R produces $r_4^{\setminus q}$. Also $(r_0 \times r_4)^{\setminus q} \times r_4^{\setminus q} = (r_0 \times r_4)^{\setminus q}$ – hence the purple text in **1a**.

Example 11. *Consider forgetting q from the program P_{11}:*

$$a \leftarrow q. \qquad b \leftarrow \mathit{not}\, q. \qquad q \leftarrow c.$$

*Then $\mathsf{f}_{SP}(P_{11},q)$ is derived by **1a**, **4** and **5**; $\mathsf{f}^+_R(P_{11},q)$ and $\mathsf{f}^-_R(P_{11},q)$ each include half of these derived rules.*

$$\begin{aligned} \mathsf{f}_R(P_{11},q) = \mathsf{f}_{SP}(P_{11},q) &= \{a \leftarrow c\,; b \leftarrow \mathit{not}\, c\,; b \leftarrow \mathit{not}\, a\} \\ \mathsf{f}^+_R(P_{11},q) &\supseteq \{a \leftarrow c\,; \ \leftarrow \mathit{not}\, c\,; \ \leftarrow \mathit{not}\, a\} \\ \mathsf{f}^-_R(P_{11},q) &\supseteq \{\ \leftarrow c\,; \qquad b \leftarrow \qquad\} \end{aligned}$$

After conjoining the results f^+_R and f^-_R by ×, all halves that have been put together 'incorrectly' are eliminated by the normal form construction. For the scope of this paper, we rely on the normal form construction to eliminate these redundancies – An efficient implementation might not produce them in the first place.

When multiple atoms are forgotten via f_R the pattern as indicated in Def. 19 repeats recursively. Consider, for example, forgetting $V = \{q_1, q_2\}$ from P. Then, first, $\mathsf{f}^+_R(P,q_1)$ and $\mathsf{f}^-_R(P,q_1)$ are calculated – the rules that are indicated in blue (resp. red), but they are not immediately conjoined by ×. Before, f^+_R and f^-_R are applied to them.

Now, the operator f_W implements the remaining derivation rules **3a**, **3b** and **6** in a way that circumvents the issue we alluded to in the introduction.

Example 12. *Consider again the first example, where forgetting p and q from program P_1 via f_{SP} in one order produces an incorrect result. Why is that? $P_1 = \{r_1, r_2, r_3\}$, where:*

$$r_1 = a \leftarrow p, q. \quad r_2 = q \leftarrow \mathit{not}\, p. \quad r_3 = p \leftarrow \mathit{not\,not}\, p.$$

Forgetting p yields $\mathsf{f}_{SP}(P_1,p) = \{r_4, r_5\}$, where:

$$r_4 = a \leftarrow q, \mathit{not\,not}\, a. \qquad r_5 = q \leftarrow \mathit{not\,not}\, q.$$

*The initial program contains a self-cycle r_3, which amounts to a choice of whether p is true or not. Rules r_4 and r_5 which are derived by **3a** and **6** in $\mathsf{f}_{SP}(P_1,p)$ mimic this choice by giving self-cycles to rules r_1 and r_2 that depend on p. However, the information that they were derived from mutually exclusive rules is lost. Therefore, when q is forgotten from $\mathsf{f}_{SP}(P_1,p)$, r_6 is derived by **3a**, as if r_4 and r_5 (and hence r_1 and r_2) are independent of each other. $\mathsf{f}_{SP}(\mathsf{f}_{SP}(P_1,p),q) = \{r_6\}$, where:*

$$r_6 = a \leftarrow \mathit{not\,not}\, a.$$

The operator f_W circumvents the creation of r_6 by compartmentalizing r_4 and r_5:

$$\mathsf{f}^+_W(P_1,p) = \{r_4\} \qquad \mathsf{f}^-_W(P_1,p) = \{r_5\}$$

and

$$\begin{aligned} \mathsf{f}^+_W(\{r_4\},q) = \mathsf{f}^-_W(\{r_4\},q) &= \\ \mathsf{f}^+_W(\{r_5\},q) = \mathsf{f}^-_W(\{r_5\},q) &= \varnothing \end{aligned}$$

Hence $\mathsf{f}_W(P_1,\{p,q\}) = \varnothing$, and thus $\mathsf{f}^*_{SP}(P_1,\{p,q\}) = \varnothing$.

6.1 Computational Advantage

While for some forgetting instances exponential blowups are inevitable (Eiter and Kern-Isberner 2018), forgetting via the semantics of a program will always be exponential with respect to the number of its atoms. In many scenarios forgetting by syntactic manipulations is much simpler.

Example 13 (Inevitable Blowup)**.** *Consider forgetting q from the following program P_{13}:*

$$a \leftarrow \mathit{not}\, q. \qquad q \leftarrow b_1, c_1. \qquad \dots \qquad q \leftarrow b_n, c_n.$$

The programs $\mathsf{f}_R(P_{13},q)$ and $\mathsf{f}^*_{SP}(P_{13},q)$ are exponentially large with respect to n.

We present some scenarios where forgetting via f_R and f^*_{SP} clearly outperforms forgetting via the semantics, and provide formal results proving that the result of forgetting by f_R and f^*_{SP} are more similar to the initial program than the result of forgetting by the semantics.

We denote the results of the counter-models method variation in (Wang, Wang, and Zhang 2013), (Wang et al. 2014) from the expected HT models f^{sem}_R and f^{sem}_{SP} respectively.

Example 14 (Computational Blowup)**.** *Consider forgetting q from P_{14}:*

$$a \leftarrow q. \qquad q \leftarrow b_1, \ldots, b_n.$$

where

$$\mathsf{f}_R(P_{14}, \{q\}) = \mathsf{f}^*_{SP}(P_{14}, \{q\}) = \{a \leftarrow b_1, \ldots, b_n\}$$

are derived rather quickly through

$$\begin{aligned}\mathsf{f}_R(P_{14}, \{q\}) &= NF(\mathsf{f}^+_R(P_{14}, q) \times \mathsf{f}^-_R(P_{14}, q)) \\ &= \{a \leftarrow b_1, \ldots, b_n\},\end{aligned}$$

where

$$\begin{aligned}\mathsf{f}^+_R(P_{14}, q) = \{&a \leftarrow b_1, \ldots, b_n; \\ &\leftarrow not\, a, not\, not\, b_1, \ldots, not\, not\, b_n\}\end{aligned}$$

$$\mathsf{f}^-_R(P_{14}, q) = \{\leftarrow b_1, \ldots, b_n; \leftarrow not\, not\, b_1, \ldots, not\, not\, b_n\}$$

and

$$\begin{aligned}\mathsf{f}^*_{SP}(P_{14}, \{q\}) &= \mathsf{f}_R(P_{14}, \{q\}) \cup \mathsf{f}^+_W(P_{14}, q) \cup \mathsf{f}^-_W(P_{14}, q) \\ &= \mathsf{f}_R(P_{14}, \{q\}) \cup \varnothing \cup \varnothing \\ &= \{a \leftarrow b_1, \ldots, b_n\},\end{aligned}$$

but deriving

$$\begin{aligned}\mathsf{f}^{sem}_R(P_{14}, \{q\}) = \mathsf{f}^{sem}_{SP}(P_{14}, \{q\}) = \\ \{\ \ \leftarrow b_1, \ldots, b_n, not\, a; \\ a \leftarrow b_1, \ldots, b_n, not\, not\, a\ \}\end{aligned}$$

involves checking 3^{n+2} interpretations.

Example 15 (Blowup of the Forgetting Result)**.** *Consider forgetting q from P_{15}:*

$$a \leftarrow b, not\, q. \qquad c \leftarrow d, not\, q. \qquad e \leftarrow f, not\, q.$$

then $\mathsf{f}_R(P_{15}, \{q\}) = \mathsf{f}^*_{SP}(P_{15}, \{q\}) =$

$$a \leftarrow b. \qquad c \leftarrow d. \qquad e \leftarrow f.$$

while $\mathsf{f}^{sem}_R(P_{15}, \{q\})$ and $\mathsf{f}^{sem}_{SP}(P_{15}, \{q\})$ both contain 127 rules, which, no doubt, in no way resemble P_{15}.

The attentive reader might notice that the number of auxiliary operators f^A_R and f^A_W is exponential with respect to the size of V, which seems to counteract the savings of not having to calculate an exponential number of interpretations.

Proposition 16. *Let P_1, P_2 and R be programs over Σ,*

$$(P_1 \cup R) \times (P_2 \cup R) \equiv (P_1 \times P_2) \cup R.$$

Since f^+_R and f^-_R both simply pass on rules R not mentioning q_i, the proposition above can be utilized to factor out any rule not mentioning q_i of its respective recursive step, thus limiting potential exponential blowups. If an intermediate result does not mention q_i at all, the recursive step can be skipped, halving the size of the recursive sub-tree. For f_W the hope is that some intermediate results are empty, to be able to prune the recursive tree as well.

Furthermore, most of the time V is much smaller than $\Sigma(P)$, which is why in practice the complexity is expected to be significantly smaller compared to the semantic approach.

The operator f_{SP} has been proven to produce more favorable results compared to f^{sem}_{SP}. To that end, Berthold et al. defined a distance measure between logic programs, and proved that the result of f_{SP} is closer to the initial program than f^{sem}_{SP}. These results directly translate to f_R and f^*_{SP}.

7 Conclusion and Discussion

Strong persistence $(\mathbf{SP})$ most accurately describes that the relations of remaining atoms are left intact after forgetting. Due to its general impossibility, a number of relaxations of $(\mathbf{SP})$ have been proposed and investigated. Notably, $(\mathbf{SP})$ decomposes into the three properties $(\mathbf{sC})$, $(\mathbf{wC})$ and $(\mathbf{SI})$, where an operator f satisfies $(\mathbf{SP})$ iff f satisfies all $(\mathbf{wC})$, $(\mathbf{sC})$ and $(\mathbf{SI})$.

The classes of operators $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ each guarantee two of those three properties, and in the case of $\mathsf{F_{SP}}$, even all three, and thus $(\mathbf{SP})$, whenever possible.

$\mathsf{F_R}$ and $\mathsf{F_{SP}}$ are defined by the semantics of their forgetting result, i.e. they do not define concrete functions, and up until now the only way to forget via their definitions, was to calculate the exponential number of models of a given input, to apply them, and then to construct a new program from the resulting models. This process is necessarily computationally hard, and its result may in no way resemble the input.

This paper introduced two novel operators f_R and f^*_{SP}, to circumvent these two issues. They forget atoms from a program purely by syntactically manipulating its rules, avoiding having to calculate the exponential number of its models, and creating a result that in some way resembles its origin.

We proved their membership in $\mathsf{F_R}$ resp. $\mathsf{F_{SP}}$, and therefore that f_R and f^*_{SP} inherit several (desirable) properties (Gonçalves et al. 2017).

Corollary 3. *f_R satisfies $(\mathbf{sC})$, $(\mathbf{SI})$, $(\mathbf{PP})$ and $(\mathbf{SE})$, but not $(\mathbf{wE})$, $(\mathbf{W})$, $(\mathbf{wC})$, nor $(\mathbf{CP})$.*

Corollary 4. *f^*_{SP} satisfies $(\mathbf{wC})$, $(\mathbf{SI})$, $(\mathbf{PP})$ and $(\mathbf{SE})$, but not $(\mathbf{wE})$, $(\mathbf{W})$, $(\mathbf{sC})$, nor $(\mathbf{CP})$. Further, if $\langle P, V\rangle$ satisfies $\neg\Omega$, then f^*_{SP} satisfies $(\mathbf{SP})_{\langle P,V\rangle}$.*

Further, we provided examples that illustrate that members of $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ cannot always forget atoms iteratively. Constructing forgetting results according to their semantics is therefore necessarily involved. The operator f_{SP}, for example, that has previously been shown to adhere to $\mathsf{F_{SP}}$ semantics, has thus been proven to be unfit to correctly forget arbitrary sets of atoms by $\mathsf{F_{SP}}$ semantics.

In the context of modular logic programming, $(\mathbf{SP})$ can be relaxed to $(\mathbf{UP})$, where the result of forgetting is only required to have the same answer set (modulo V) under the addition of facts (Gonçalves et al. 2019).

There exists a semantic class $\mathsf{F_{UP}}$, that guarantees $(\mathbf{UP})$, and interestingly for us, also guarantees $(\mathbf{sC})$ and $(\mathbf{wC})$, with a representative $\mathsf{f}_u \in \mathsf{F_{UP}}$ (Gonçalves et al. 2021), that too, if applicable, forgets atoms through syntactic manipulation. In other words, with f_R, f^*_{SP} and f_u at our disposal, we can now syntactically forget satisfying any two properties of $(\mathbf{wC})$, $(\mathbf{sC})$ and $(\mathbf{SI})$.

One future line of research is to explore how forgetting translates to other formalisms, as has been done for abstract argumentation (Baumann and Berthold 2022). Another one is to investigate forgetting from logic programs with variables. In this larger context, forgetting can mean a number of things: the removal of all occurrences of a predicate, the removal of all occurrences of an individual, or perhaps, the removal of all occurrences of certain individuals within some predicates, to, for example, just forget certain information about an entity.

Acknowledgements

Quite obviously, this work is a continuation of my master's thesis, which was mainly supervised by Ricardo Gonçalves. I want, therefore, to take the opportunity to express my gratitude for his support and efforts throughout the process.
Back then, he was the one to tell me that there might be a wrong forgetting order, when iterating f_{SP}, using a notation that I adapted in Exm. 3 and 4. I hope you like my fix!
Further, I thank the anonymous reviewers for all of their useful hints.
This work was supported by the German Federal Ministry of Education and Research (BMBF, 01IS18026B-F) by funding the competence center for Big Data and AI "ScaDS.AI" Dresden/Leipzig.

References

Baumann, R., and Berthold, M. 2022. Limits and possibilities of forgetting in abstract argumentation. In *Proceedings of (IJCAI-22)*. To appear.

Berthold, M.; Gonçalves, R.; Knorr, M.; and Leite, J. 2019. A syntactic operator for forgetting that satisfies strong persistence. *Theory and Practice of Logic Programming* 19(5-6):1038–1055.

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

Cabalar, P., and Ferraris, P. 2007. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming* 7(6):745–759.

Cabalar, P.; Pearce, D.; and Valverde, A. 2007. Minimal logic programs. In *Proceedings of (ICLP-07)*. Springer.

Eiter, T., and Kern-Isberner, G. 2018. A brief survey on forgetting from a knowledge representation and reasoning perspective. *KI - Künstliche Intelligenz.*

Eiter, T., and Wang, K. 2008. Semantic forgetting in answer set programming. *Artificial Intelligence* 172(14):1644–1672.

Gonçalves, R.; Knorr, M.; Leite, J.; and Woltran, S. 2017. When you must forget: Beyond strong persistence when forgetting in answer set programming. *Theory and Practice of Logic Programming* 17(5-6):837–854.

Gonçalves, R.; Janhunen, T.; Knorr, M.; Leite, J.; and Woltran, S. 2019. Forgetting in modular answer set programming. In *Proceedings of (AAAI-19)*, 2843–2850. AAAI Press.

Gonçalves, R.; Janhunen, T.; Knorr, M.; and Leite, J. 2021. On syntactic forgetting under uniform equivalence. In *Proceedings of (JELIA-21)*, volume 12678 of *Lecture Notes in Computer Science*, 297–312. Springer.

Gonçalves, R.; Knorr, M.; and Leite, J. 2016a. The ultimate guide to forgetting in answer set programming. In *Proceedings of (KR-16)*, 135–144.

Gonçalves, R.; Knorr, M.; and Leite, J. 2016b. You can't always forget what you want: On the limits of forgetting in answer set programming. In *Proceedings of (ECAI-16)*, 957–965.

Inoue, K., and Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming* 35(1):39–78.

Inoue, K., and Sakama, C. 2004. Equivalence of logic programs under updates. In *Proceedings of (JELIA-04)*. Springer.

Knorr, M., and Alferes, J. J. 2014. Preserving strong equivalence while forgetting. In *Proceedings of (JELIA-14)*, volume 8761 of *LNCS*, 412–425. Springer.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2(4):526–541.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence* 25(3-4):369–389.

Slota, M., and Leite, J. 2011. Back and forth between rules and SE-models. In *Proceedings of (LPNMR-11)*. Springer.

Wang, Y.; Zhang, Y.; Zhou, Y.; and Zhang, M. 2014. Knowledge forgetting in answer set programming. *Journal of Artificial Intelligence Research* 50:31–70.

Wang, Y.; Wang, K.; and Zhang, M. 2013. Forgetting for answer set programs revisited. In *Proceedings of (IJCAI-13)*, 1162–1168. IJCAI/AAAI.

Zhang, Y., and Foo, N. Y. 2006. Solving logic program conflict through strong and weak forgettings. *Artificial Intelligence* 170(8-9):739–778.