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 (SP) captures best the essence of forgetting in logic programming. While classes of operators, such as FR and FSP, that satisfy immediate relaxations of (SP), and in the case of FSP, even (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 FR and FSP that forget sets of atoms syntactically. 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 or, to simplify it by removing unnecessary data.

Recent studies investigated several forgetting properties and identified Strong Persistence (SP) to most accurately describe the essence of forgetting (Gonçalves, Knorr, and Leite 2016a). (SP), however, is not always obtainable (Gonçalves, Knorr, and Leite 2016b). Therefore forgetting operators have been defined semantically, satisfying different relaxations of (SP), among them FR and FSP, where notably FSP does satisfy (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 (SP) (cf. Berthold et al. 2019 for an overview).

The only syntactic operator to satisfy (SP), whenever possible, fSP, only forgets one atom. To this end, it was proven to conform to the semantics of class FSP. But can fSP be iterated to correctly forget arbitrary sets of atoms?

Example 1. Consider forgetting p and q from program P1:

\[
\begin{align*}
\text{a} & \leftarrow \text{p}, \text{q}.
\text{q} & \leftarrow \text{not p}.
\text{p} & \leftarrow \text{not p}.
\end{align*}
\]

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. And indeed, if we first forget q and then p via fSP, we arrive at our desired result. Forgetting q first yields fSP(P1, q):

\[
\begin{align*}
\text{a} & \leftarrow \text{p}, \text{not q}.
\text{p} & \leftarrow \text{not p}.
\end{align*}
\]

where the appearance of q in the rule a → 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 unsatisfactory together. Forgetting p subsequently naturally yields the empty program, i.e. fSP(fSP(P1, q), p) = ∅.

Observe what happens if the order of forgetting p and q is reversed: Forgetting p first yields fSP(P1, p):

\[
\begin{align*}
\text{a} & \leftarrow \text{q}, \text{not a}.
\text{q} & \leftarrow \text{not q}.
\end{align*}
\]

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 fSP(fSP(P1, p), q):

\[
\begin{align*}
\text{a} & \leftarrow \text{not a}.
\end{align*}
\]

which is not our desired forgetting result.

Apparently, fSP is sensitive to the order in which atoms are forgotten. We will show that, worse yet, for both classes, FR and FSP, 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 fSP insufficient for forgetting arbitrary sets of atoms from programs by FSP semantics, and illustrates that concrete forgetting procedures for both FR and FSP necessarily are quite complex.

This paper introduces the syntactic operators fR and fSP, to overcome these issues, that forget any set of atoms correctly by the semantics of FR and FSP.

1This intuition coincides with FSP.
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 \( f_P \) and \( f_{\mathcal{S}P} \) 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 \( \Sigma \). A logic program \( P \) over \( \Sigma \) is a finite set of rules of the form

\[
a_1 \lor \ldots \lor a_k \leftarrow b_1, \ldots, b_l, \neg c_1, \ldots, \neg c_m,
\]

\[
\text{not } d_1, \ldots, \text{not } d_n,
\]

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 \( \Sigma \) (Lifschitz, Tang, and Turner 1999). Such rules \( r \) are also written more succinctly as

\[
H(r) \leftarrow B^+(r), \text{not } B^-(r), \text{not not } B^-(r),
\]

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 \( B(r) = B^+(r) \lor \text{not } B^-(r) \lor \text{not not } B^-(r) \) is called the body of \( r \), where, for a set \( A \) of atoms, \( \text{not } A = \{\text{not } q \mid q \in A\} \) and \( \text{not not } A = \{\text{not not } q \mid q \in A\} \).

\( \Sigma(P) \) 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 \( P^I = \{H(r) \leftarrow B^+(r) \mid r \in P \text{ such that } B^-(r) \cap I = \emptyset \text{ and } B^-(r) \subseteq I\} \). An HT-interpretation is a pair \((X, Y)\) s.t. \( X \subseteq Y \subseteq \Sigma \). Given a program \( P \), an HT-interpretation \((X, Y)\) is an HT-model of \( P \), \((X, Y) \models P \), iff \( Y \models P \) and \( X \models P_Y \), where \( \models \) denotes the standard satisfaction relation for classical logic and for HT-logic. An HT-interpretation \((X, Y)\) is total iff \( X = Y \). Given a rule \( r \), \((X, Y) \models r \) iff \( (X, Y) \not\models \{r\} \). 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 \( (Y, Y) \in \mathcal{HT}(P) \), and there is no \( X \subset Y \) such that \((X, Y) \in \mathcal{HT}(P) \). We term HT-models \((X, Y)\) s.t. \( X \subsetneq Y \) 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 \( \mathcal{M} \mathcal{V} \), is \( \{X \setminus V \mid X \in \mathcal{M}\} \). We assume that \( \emptyset, Y, \emptyset \cup \{p, q\} \) means \( \emptyset, Y, \emptyset \cup \{p, q\} \).

Forgetting: Properties and Operators
Let \( \mathcal{P} \) be the set of all logic programs. A forgetting operator \( f : \mathcal{P} \times 2^\Sigma \to \mathcal{P} \). The program \( f(P, V) \) is interpreted as the result of forgetting about \( V \) from \( P \). Moreover, \( \mathcal{AS}(P, V) \subseteq \mathcal{AS}(P) \setminus V \) 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 \( f(P, V) \) is strongly equivalent to the original program \( P \), modulo the forgotten atoms.

\[ (SP) \text{ } f \text{ satisfies strong persistence iff, for each program } P \text{ and each set of atoms } V, \text{ we have: } \mathcal{AS}(P \cup R) \models \mathcal{AS}(f(P, V) \cup R) \text{ for all programs } R \text{ with } \Sigma(R) \subseteq \Sigma \setminus V. \]

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

\[ (wC) \text{ } f \text{ satisfies weakened consequence iff, for each program } P \text{ and each set of atoms } V: \mathcal{AS}(f(P, V)) \subseteq \mathcal{AS}(P | V). \]

\[ (sC) \text{ } f \text{ satisfies strengthened consequence iff, for each program } P \text{ and each set of atoms } V: \mathcal{AS}(f(P, V)) \subseteq \mathcal{AS}(P | V). \]

\[ (SI) \text{ } f \text{ satisfies strong invariance iff, for each program } P \text{ and each set of atoms } V, \text{ we have: } f(P, V) \cup R \equiv f(P \cup R, V) \text{ for all programs } R \text{ with } \Sigma(R) \subseteq \Sigma \setminus V. \]

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 (SP) for all pairs \((P, V)\), 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), (SP) was defined for concrete forgetting instances. A forgetting operator \( f \) satisfies (SP) \((P, V)\), if \( \mathcal{AS}(f(P, V) \cup R) = \mathcal{AS}(P \cup R) \mid V \), for all programs \( R \) with \( \Sigma(R) \subseteq \Sigma \setminus V \). A sound and complete criterion \( \Omega \) characterizes when it is not possible to forget while satisfying (SP) \((P, V)\).

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

\[ \text{Rel}(P, V) := \{A \subseteq V \mid (Y' \cup A, Y, A) \in \mathcal{HT}(P) \text{ and } \exists A' \subseteq A : (Y' \cup A', Y, A) \in \mathcal{HT}(P)\}. \]

Then, an instance \((P, V)\) satisfies criterion \( \Omega \) iff there exist the considered total models with matching unforgotten part \( Y \), there is no less element, with respect to their witnesses modulo \( V \), more formally, iff there is \( Y' \subseteq \Sigma \setminus V \) such that the set of sets \( \mathcal{R}(P, V) := \{R(P, V) \mid A \in \text{Rel}(P, V)\} \) is non-empty and has no less element, where

\[ \mathcal{R}(P, V) := \{X \setminus V \mid (X, Y, A) \in \mathcal{HT}(P)\}. \]
Example 2. Consider the forgetting instance \( \{P_2, \{p\}\} \), where \( P_2 \) has the following HT-models\(^3\):

\[
\begin{align*}
\{apq, apq\} & \quad \{aq, aq\} & \quad \{ap, ap\} \\
\{\emptyset, apq\} & \quad \{a, aq\} & \quad \{a, ap\}
\end{align*}
\]

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

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

The classes \( F_R \) and \( F_{SP} \) are defined as follows:

\[
F_R := \{f | \mathcal{HT}(f(P, V)) = \{X, Y\} | Y \subseteq \Sigma(P) \wedge X \in \bigcup \mathcal{R}_{(P,V)} \}.
\]

\[
F_{SP} := \{f | \mathcal{HT}(f(P, V)) = \{X, Y\} | Y \subseteq \Sigma(P) \wedge X \in \mathcal{R}_{(P,V)} \}.
\]

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

Example 3. We assume \( f \in F_R \). 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 \( f(P_i, V) = P_j \). An edge from \( P_i \) to \( P_j \) is additionally labeled with \( \Omega \), iff \( \{P_i, V\} \) satisfies \( \Omega \). The program \( P_2 \) from Exm. 2 is in the topmost box.

\[
\begin{align*}
\{apq, apq\} & \quad \{aq, aq\} & \quad \{ap, ap\} \\
\{\emptyset, apq\} & \quad \{a, aq\} & \quad \{a, ap\}
\end{align*}
\]

If \( p \) is forgotten from \( P_2 \), then \( \text{Rel}_{(P_2, \{p\})} = \{\emptyset, \{p\}\} \). Hence \( \{aq, aq\} \neq f(P_2, \{p\}) \). Since \( \{a, aq\} = P_2 \), and \( \{\emptyset, apq\} = P_2 \), we get \( \{a, ap\} = f(P_2, \{p\}) \). And \( \{\emptyset, aq\} = f(P_2, \{p\}) \).

For all \( Y \subseteq \Sigma(P_2) \wedge V \), s.t. \( Y \neq \{a, q\} \), we have \( \text{Rel}_{(P_2, \{p\})} = \emptyset \), which entails \( \{Y, Y\} \neq f(P_2, \{p\}) \)

Example 4. Now assume \( f \in F_{SP} \) instead, and the same graphical representation of forgetting as in Exm. 3. When \( p \) is forgotten from \( P_2 \), then \( \{aq, aq\} = f(P_2, \{p\}) \), for the same reason as in Exm. 3, since \( \text{Rel}_{(P_2, \{p\})} = \emptyset \), but, since \( \{ap, ap\} \neq P_2 \), and \( \{a, ap\} \neq P_2 \), we get \( \{a, aq\} \neq f(P_2, \{p\}) \). In addition, \( \{a, q\} \neq P_2 \), and \( \{\emptyset, aq\} \neq P_2 \) entail \( \{\emptyset, aq\} \neq f(P_2, \{p\}) \). As before, for all \( Y \subseteq \Sigma(P_2) \wedge V \), s.t. \( Y \neq \{a, q\} \), we have \( \text{Rel}_{(P_2, \{p\})} = \emptyset \), which entails \( \{Y, Y\} \neq f(P_2, \{p\}) \)

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

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

3 Syntactic Tools

Defining syntactic forgetting operators obeying the semantics of \( F_R \) and \( F_{SP} \) 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; Słota 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:

\[
\begin{align*}
& a \lor b \leftarrow a. & a \lor b \leftarrow \text{not } a. \\
& b \leftarrow a, \text{not } a. & b \leftarrow a, \text{not } a.
\end{align*}
\]
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 ≤ 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. \( B^+(s) \cap B^-(r) = \emptyset \) or \( H(s) \cap H(r) = \emptyset \).

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, if it is not (strictly) subsumed by another rule s in P, i.e. iff \( \neg \exists s \in P : s \leq r \) and \( r \notin s \).

**Example 6.** Consider the following program \( P_6 \):

\[ a \leftarrow \text{not not } b. \quad a \leftarrow c, \text{not not } b. \quad \leftarrow b, \text{not } a. \]

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 \( H(r) \cap B^+(r) \neq \emptyset \), or \( B^+(r) \cap B^-(r) \neq \emptyset \), or \( B^-(r) \cap B^-(r) \neq \emptyset \).

**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 \( a \notin B(r) \);
- if \( a \in B(r) \), then \( a \notin B(r) \);
- 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 \( \Sigma \), the q-exclusions are \( L^q := L \setminus \{ q, \text{not } q, \text{not not } q \} \), \( r^q := H^q(r) \leftarrow B^q(r) \) and \( P^q := \{ r^q \mid r \in P \} \).

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 \( \Sigma \) and an atom \( q \in \Sigma \), P is partitioned according to the occurrence of \( q \), i.e. \( \text{occ}(P,q) := \{ R, R_0, R_1, R_2, R_3, R_4 \} \), where

- \( R := \{ r \in P \mid q \notin \Sigma(r) \} \)
- \( R_0 := \{ r \in P \mid q \in B(r) \} \)
- \( R_1 := \{ r \in P \mid \text{not } q \in B(r) \} \)
- \( R_2 := \{ r \in P \mid \text{not not } q \in B(r), q \notin H(r) \} \)
- \( R_3 := \{ r \in P \mid \text{not not } q \in B(r), q \in H(r) \} \)
- \( R_4 := \{ r \in P \mid \text{not } q \in B(r), q \in H(r) \} \)

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 \( \Sigma \), \( X \subseteq Y \subseteq \Sigma \), and an atom \( q \in \Sigma \), with \( q \notin Y \), and \( \text{occ}(P,q) = \{ R, R_0, R_1, R_2, R_3, R_4 \} \). Then the following equivalencies hold:

\[
\begin{align*}
(X,Y) \neq P & \iff \exists r \in R \cup R_1 \cup R_4 : (X,Y) \neq r \\
(Xq,Yq) \neq P & \iff \exists r \in R_0 \cup R_2 : (Xq,Yq) \neq r \\
(X,Yq) \neq P & \iff \exists r \in R \cup R_2 \cup R_3 \cup R_4 : (X,Yq) \neq r
\end{align*}
\]

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 \( P_7 \):

\[ a \lor q \leftarrow b. \quad q \leftarrow c. \]
If $q$ 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 a, not c\}.

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

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

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

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:

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

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

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

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 $\Sigma$, $Y \subseteq \Sigma$, and an atom $q \in \Sigma$, with $q \notin Y$, and occ($P, q$) $\subseteq \Sigma$, then the following implications hold:

\begin{align*}
(Y, Y) \equiv P & \rightarrow \exists D \in D^q_{as}(R_1 \cup R_4) : (Y, Y) \not \leftrightarrow D \\
(Y_q, Y_q) \equiv P & \rightarrow \exists D \in D^q_{as}(R_0 \cup R_2) : (Y_q, Y_q) \not \leftrightarrow D \\
(Y, Y_q) \equiv P & \rightarrow \exists D \in D^q_{as}(R_3 \cup R_4) : (Y, Y_q) \not \leftrightarrow D
\end{align*}

In the case that $R = \emptyset$ 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. 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 $\Sigma$, and $X \subseteq Y \subseteq \Sigma$,

$$Y \equiv r_1 \times r_2 \equiv Y \equiv r_1 \vee Y \equiv r_2$$

$$X \equiv \{r_1 \times r_2\}^Y \leftarrow X \leftarrow \{r_1\}^Y \vee X \leftarrow \{r_2\}^Y$$

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

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

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

\begin{align*}
Y & \equiv P_1 \times P_2 \equiv Y \equiv P_1 \vee Y \equiv P_2 \\
X & \equiv (P_1 \times P_2)^Y \equiv X \equiv P_1^Y \vee X \equiv P_2^Y
\end{align*}

In general, $H_T(P_1) \cup H_T(P_2) = H_T(P_1 \times P_2)$ does not hold, e.g. $(\emptyset, ab) \not\equiv (a \leftarrow \emptyset)$ and $(\emptyset, ab) \not\equiv (\emptyset, b)$, but $(\emptyset, ab) \equiv (a \leftarrow b)$. 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 $(X, Y)$ is a classical model, and therefore whether the corresponding total model $(Y, Y)$ is a potential answer set.

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

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

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

\begin{align*}
Y \equiv r \equiv (X, Y) & \equiv \not not r
\end{align*}

Using the double negation, for example, we can construct $P = \{a \leftarrow \emptyset \times \{\not not \leftarrow \emptyset\} = \{a \leftarrow \not not \leftarrow \emptyset\}$, such that $(\emptyset, ab) \not\equiv P$, and $H_T(P) = H_T((a \leftarrow \emptyset \times \{\not not \leftarrow \emptyset\} \cup \{\not not \leftarrow \emptyset\} \cup H_T).$

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 $\Sigma$, $X \subseteq Y \subseteq \Sigma$, and an atom $q \in \Sigma$, with $q \notin Y$, and occ($P, q$) $\subseteq (R, R_0, R_1, R_2, R_3, R_4)$, then the following equivalences hold:

\begin{align*}
(Y, Y) \not\equiv P & \leftarrow \exists r \in R_1 \cup R_4 : (Y, Y) \not\equiv not r_q

\vee \exists r \in R : (Y, Y) \not\equiv r

\{X, Y\} \not\equiv P & \leftarrow \exists r \in R_0 \cup R_2 : (Y, Y) \not\equiv not r_q

\vee \exists r \in R : (Y, Y) \not\equiv r

\{Y, Y_q\} \not\equiv P & \leftarrow (Y_q, Y_q) \not\equiv P

\vee \exists r \in R_3 \cup R_4 : (Y, Y) \not\equiv not r_q

\{Y, Y_q\} \not\equiv P & \leftarrow (Y_q, Y_q) \not\equiv P

\land \exists D \in D^q_{as}(R_3 \cup R_4) : (Y, Y) \not\leftrightarrow D

\{X_q, Y_q\} \not\equiv P & \leftarrow \exists r \in R_0 \cup R_2 : (X, Y) \not\equiv not r_q

\{X, Y_q\} \not\equiv P & \leftarrow (X_q, Y_q) \not\equiv P

\land \exists D \in D^q_{as}(R_3 \cup R_4) : (X, Y) \not\leftrightarrow D

\{X_q, Y_q\} \not\equiv P & \leftarrow \exists r \in R_0 \cup R_2 : (X, Y) \not\equiv not r_q

\{X, Y_q\} \not\equiv P & \leftarrow (X_q, Y_q) \not\equiv P

\land \exists D \in D^q_{as}(R_3 \cup R_4) : (X, Y) \not\leftrightarrow D
\end{align*}

If additionally $R = \emptyset$, then:

\begin{align*}
(Y, Y) \equiv P & \leftarrow \exists D \in D^q_{as}(R_1 \cup R_4) : (Y, Y) \not\leftrightarrow D

\{X_q, Y_q\} \not\equiv P & \leftarrow \exists r \in R_0 \cup R_2 : (X, Y) \not\equiv not r_q
\end{align*}

For all $r_2 \in R_2$:

\begin{align*}
\{Y_q, Y_q\} \not\equiv r_2 & \leftarrow (Y, Y_q) \not\equiv r_2, \text{ and }

\{X_q, Y_q\} \not\equiv r_2 & \leftarrow (X, Y_q) \not\equiv r_2.
\end{align*}

The rules identified in Prop. 8 constitute the essential building blocks, used to define $f_R$ and $f^a_R$. 47
4 Relativized Forgetting

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

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

The operators $f^A_R$ 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 $\text{Rel}^Y_{(P,q)} \subseteq \emptyset \cup \{q\}$. For forgetting one atom $q$ it holds that $\{q\} \in \text{Rel}^Y_{(P,q)}$, iff $(Y q, Y q) = P$ and $(Y, Y q) \notin P$. Using Prop. 8, we are able to construct rules, contradicting $(Y, Y)$, if $q \notin \text{Rel}^Y_{(P,q)}$, and otherwise, if $\{q\}$ is relevant, satisfying $(X, Y)$ iff $(X q, Y q) = P$ or $(X, Y q) = P$.

**Definition 10 ($f^P_R$).** Given a program $P$ over $\Sigma$, and $q \in \Sigma$ s.t. $\text{occ}(P, q) = \{R, R_0, R_1, R_2, R_3, R_4\}$, then:

$$f^P_R(P, q) := nf(A \cup B \cup C \cup D)$$

where:

$$A := \{\neg D \mid D \in D^q_{as}(R_3 \cup R_4)\}$$

$$B := \{\neg \neg r' \mid r_0 \in R_0 \cup R_2\}$$

$$C := \{r^q \mid r \in R \cup R_2\}$$

$$D := \{(r_0 \times r')^q \mid r_0 \in R_0, r' \in R_3 \cup R_4\}$$

The four sets $A, B, C, D$ making up $f^P_R(P, q)$ directly correspond to the conclusions in Prop. 8. If $(Y q, Y q) \notin P$, a rule $r \in B \cup C$ contradicts $(Y, Y)$. If $(Y, Y q) = P$, then a rule $r \in A$ contradicts $(Y, Y)$. Otherwise, if both $(X q, Y q) \notin P$ and $(X, Y q) \notin P$, then a rule in $C \cup D$ contradicts $(X, Y)$.

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

$$(Y, Y) = f^P_R(P, q) \iff (q) \in \text{Rel}^Y_{(P,q)}$$

If $\{q\} \in \text{Rel}^Y_{(P,q)}$, then:

$$(X, Y) = f^P_R(P, q) \iff (X q, Y q) = P \vee (X, Y q) = P$$

It holds that $\emptyset \in \text{Rel}^Y_{(P,q)} \iff (Y, Y) = P$. Once again, we utilize Prop. 8 to construct an auxiliary program.

**Definition 11 ($f^P_R$).** Given a program $P$ in normal form over $\Sigma$ and $q \in \Sigma$ s.t. $\text{occ}(P, q) = \{R, R_0, R_1, R_2, R_3, R_4\}$, then:

$$f^P_R(P, q) := nf(A \cup B)$$

where:

$$A := \{r^q \mid r' \in R_1 \cup R_4\}$$

$$B := \{\neg \neg r^q \mid r' \in R_1 \cup R_4\}$$

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

$$\langle Y, Y \rangle = f^P_R(P, q) \iff \emptyset \in \text{Rel}^Y_{(P,q)}$$

If $\emptyset \in \text{Rel}^Y_{(P,q)}$, then:

$$\langle X, Y \rangle = f^P_R(P, q) \iff \langle X, Y \rangle = P$$

Now, using $f^P_R$ and $f^P_R$ as building blocks, we inductively construct $f^P_R$ for $V$ of arbitrary sizes. We assume an arbitrary ordering on the signature $\Sigma$ 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 $f^P_R(P,V)$ is correct anyhow.

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

$$f^P_R(P, \emptyset) := P$$

$$f^P_R(P, V) := f^{P_n}_R(f^{P_{n-1}}_R(P, \{q_n\}), q_n)$$

where:

$$f^{P_n}_R := \begin{cases} f^P_R, &\text{if } q_n \in A \\ f^{P}_{R}, &\text{otherwise} \end{cases}$$

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

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

$$\langle Y, Y \rangle = f^P_R(P, V) \iff A \in \text{Rel}^Y_{(P,V)}$$

If $A \in \text{Rel}^Y_{(P,V)}$ then:

$$\langle X, Y \rangle = f^P_R(P, V) \iff \exists A'' \subseteq A : (X A'', Y A) = P$$

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

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

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

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

$$f_R(P, V) := NF(\times \limits_{A \subseteq V} f^A_R(P, V))$$

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

**Theorem 1.** $f_R \in F_R$
5 Forgetting with Strong Persistence

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

\[
(Y,Y) \models f_{SP}^*(P,V) \iff (Y,Y) \models f_R(P,V).
\]

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

Example 9. Consider \( P_2 \) from earlier, where

\[
H(T(f_R(P_2, \{p\})) = \{\{aq, aq\}, \{a, aq\}, \{\varnothing, aq\} \}
\]

\( f_{W}(P_2, \{p\}) \) specifically contradicts \( \{a, aq\} \) and \( \{\varnothing, aq\}, \) s.t.

\[
H(T(f_R(P_2, \{p\}) \cup f_W(P_2, \{p\})) = \{\{aq, aq\} \}.
\]

As for \( f_R \), we define \( f_W \) by the help of auxiliary operators for every \( A \in V \). Each \( f_W^*(P,V) \) contradicts \( (X,Y) \), i.e. \( (X,Y) \notin f_W^*(P,V) \), iff \( A \in Rel^{f_W^*(P,V)} \) and \( \forall A'' \in A : (X A'', Y A) \notin P \), where \( X \subseteq \Sigma \).

Also as before, the auxiliary operators \( f_W^* \) are defined inductively by nested calls of \( f_W \).

Definition 14 \( (f_W^*) \). Given a program \( P \) in normal form over \( \Sigma \) and \( q \in \Sigma \) s.t. \( \text{occ}(P,q) = (R_0, R_1, R_2, R_3, R_4) \), then:

\[
f_W^*(P,V) := NF(A \cup B)
\]

where:

\[
A := \{ (r_0 \times r' \times \text{not not} r)^{\vartheta} \times \rightarrow D | r_0 \in R_0, r, r' \in R_3 \cup R_4, D \in \mathcal{D}^q_{\vartheta}(R_0 \cup R_2) \}
\]

\[
B := \{ (r \times \text{not not} r')^{\vartheta} \times \rightarrow D | r \in R \cup R_2, r' \in R_3 \cup R_4, D \in \mathcal{D}^q_{\vartheta}(R_0 \cup R_2) \}
\]

The operator \( f_W \) creates a rule contradicting \( (X,Y) \) iff \( \{q\} \in Rel^{f_W^*(P,V)} \) and \( (Xq,Yq) \in H(T(P)) \). According to Prop. 8, \( \{q\} \in Rel^{f_W^*(P,V)} \) entails the existence of \( D \in \mathcal{D}^q_{\vartheta}(P) \) and \( r \in R_3 \cup R_4 \), s.t. \( (X,Y) \notin \rightarrow D \) and \( (X,Y) \notin \text{not not} r^{\vartheta} \). Also, \( (Xq,Yq) \notin P \land (X,Yq) \notin P \) implies the existence of an \( r \in R \cup R_2 \), s.t. \( (X,Y) \notin r^{\vartheta} \), or \( r_0 \in R_0 \), and \( r' \in R_3 \cup R_4 \), s.t. \( (X,Y) \notin r_0^{\vartheta} \times \text{not not} r^{\vartheta} \). Hence the sets \( A \) and \( B \).

Proposition 12. Given a program \( P \) over \( \Sigma \), an atom \( q \in \Sigma \), and sets \( X \) and \( Y \), s.t. \( X \subseteq Y \subseteq \Sigma \), then:

\[
\{q\} \in Rel^{f_W^*(P,V)} \land \{Xq,Yq\} \notin P \land \{X,Yq\} \notin P \iff (X,Y) \notin f_W^*(P,q)
\]

Definition 15 \( (f_{SP}^*) \). Given a program \( P \) in normal form over \( \Sigma \) and \( q \in \Sigma \) s.t. \( \text{occ}(P,q) = (R_0, R_1, R_2, R_3, R_4) \), then:

\[
f_{SP}^*(P,V) := NF(A)
\]

where:

\[
A := \{ r^{\vartheta} \times \rightarrow D | r' \in R \cup R_1 \cup R_4, D \in \mathcal{D}^q_{\vartheta}(R_1 \cup R_4) \}
\]

Again, using Prop. 8, \( \varnothing \in Rel^{f_W^*(P,V)} \) implies the existence of \( D \in \mathcal{D}^q_{\vartheta}(P) \) s.t. \( (X,Y) \notin \rightarrow D \) and if \( (X,Y) \notin P \) then there is a rule \( r \in R_1 \cup R_3 \). s.t. \( (X,Y) \notin r^{\vartheta} \).

Proposition 13. Given a program \( P \) over \( \Sigma \), an atom \( q \in \Sigma \), and sets \( X \) and \( Y \), s.t. \( X \subseteq Y \subseteq \Sigma \), then:

\[
\varnothing \in Rel^{f_W^*(P,V)} \land (X,Y) \notin P \iff (X,Y) \notin f_{SP}^*(P,q)
\]

Example 10. Recall \( P_1 \) from the beginning,

\[
f_{W}(P_1, \{p\}) = \{ a \leftarrow \theta, \text{not not} a \} \cup \varnothing
\]

Now, the operators \( f_{SP}^* \) are defined inductively.

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

\[
f_{SP}^*(P, \varnothing) := P
\]

\[
f_{SP}^*(P,V) := f_{W}^{\vartheta}((f_{SP}^*)_{\vartheta}(P \setminus R, V \setminus \{q_n\}), q_n) \cup R
\]

where:

\[
f_{W}^{\vartheta} := \begin{cases} f_{W}, & \text{if } q_n \in A \\ f_{W}, & \text{otherwise} \end{cases}
\]

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

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 \( \Sigma \), and sets \( X, Y, A \) and \( V \), s.t. \( A \subseteq V \subseteq \Sigma \), \( X \subseteq Y \subseteq \Sigma \setminus V \), and

\[
R = \{ r \in P \mid V \cap \Sigma(r) = \varnothing \}
\]

\[
A \in Rel^{f_W^*(P,V)} \land \forall A'' \in A : (X A'', Y A) \notin P
\]

\[
\rightarrow (X,Y) \notin f_{SP}^*(P,V)
\]

Now, \( f_{W}(P,V) \) is the union of all \( f_{SP}^*(P,V) \).

Definition 17 \( (f_{W}) \). Given a program \( P \) in normal form over \( \Sigma \) and \( V \subseteq \Sigma \), then:

\[
f_{W}(P,V) := NF( \bigcup_{A 

\text{The union of programs satisfies the intersections of their models. Hence the next proposition.}

Proposition 15. Given a program \( P \) over \( \Sigma \) in normal form, three sets of atoms \( X, Y, V \), s.t. \( X \subseteq Y \subseteq \Sigma \), \( V \subseteq \Sigma \), and

\[
R = \{ r \in P \mid V \cap \Sigma(r) = \varnothing \}
\]

\[
A \in Rel^{f_W^*(P,V)} \land \forall A'' \in A : (X A'', Y A) \notin P
\]

\[
\rightarrow (X,Y) \notin f_{W}(P,V)
\]

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 \( \Sigma \) in normal form and \( V \subseteq \Sigma \).

\[
f_{SP}^*(P,V) := NF(f_W(P,V) \cup f_R(P,V))
\]

Theorem 2. \( f_{SP}^* \in F_{SP} \)
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 ‘x’. 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 $\Sigma$, $q \in \Sigma$, and $occ(P, q) = \{R, R_0, R_1, R_2, R_3, R_4\}$, then $f_{SP}(P, q) = NF(\emptyset \cup 1a \cup 2a \cup 3a \cup 1b \cup 2b \cup 3b \cup 4 \cup 5 \cup 6)$, where:

$$0 = R$$

$$1a = \{(r_0 \times r_3) \times q \mid r_0 \in R_0, r_3 \in R_4\}$$

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

$$3a = \{(r_0 \times r_3 \times not not r^\prime q) \times q \times \times D \mid r_0 \in R_0, r_3 \in R_3, D \in D_{\emptyset}(R_0 \cup R_2)\}$$

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

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

$$3b = \{(r_2^\prime q \times not not r^\prime q) \times \times D \mid r_2 \in R_2, r_3 \in R_3, D \in D_{\emptyset}(R_0 \cup R_2)\}$$

$$4 = \{r^\prime q \times \times D \mid r^\prime \in R_1 \cup R_4, D \in D_{\emptyset}(R_3 \cup R_4)\}$$

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

$$6 = \{r^\prime q \times \times D \mid r^\prime \in R_1 \cup R_4, D \in D_{\emptyset}(R_1 \cup R_4)\}$$

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 $f_R^x$ respectively gather half of each derivation rule. These rules are then conjoined by $\times$. The outputs of $f_R$ (resp. $f_R^x$) are indicated in the definition above in blue (resp. red) text colors. For example, given that $r_0 \in R_0$ and $r_1 \in R_3$, $f_R$ produces $(r_0 \times r_1)^q$, while $f_R^x$ produces $r_1^q$. Also $(r_0 \times r_1)^q \times r_1^q = (r_0 \times r_1)^q$ – hence the purple text in 1a.

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

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

Then $f_{SP}(P_{11}, q)$ is derived by 1a, 4 and 5; $f_R^x(P_{11}, q)$ and $f_R(P_{11}, q)$ each include half of these derived rules.

$$f_R(P_{11}, q) = f_{SP}(P_{11}, q) = \{a \leftarrow c; b not c; b \leftarrow not a\}$$

$$f_R^x(P_{11}, q) \supseteq \{a \leftarrow c; \leftarrow not c; \leftarrow not a\}$$

After conjoining the results $f_R^x$ and $f_R$ by $\times$, 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, $f_R(P, q_1)$ and $f_R(P, q_1)$ are calculated – the rules that are indicated in blue (resp. red), but they are not immediately conjoined by $\times$. Before, $f_R^x$ 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 $P_3$ via $f_{SP}$ in one order produces an incorrect result. Why is that? $P_1 = \{r_1, r_2, r_3\}$, where:

$$r_1 = a \leftarrow p, q. \ r_2 = q \leftarrow not p. \ r_3 = p \leftarrow not p.$$.

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

$$r_4 = a \leftarrow q, not not a. \ r_5 = q \leftarrow not q.$$.

The initial program contains a self-cycle $r_3$, which amounts to a choice of whether $p$ is true or not. Rules $r_3$ and $r_5$, which are derived by 3a and 6 in $f_{SP}(P_1, p)$ 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 $f_{SP}(P_1, p)$, $r_5$ is derived by 3a, as if $r_4$ and $r_5$ (and hence $r_1$ and $r_2$) are independent of each other. $f_{SP}(f_{SP}(P_1, p), q) = \{r_5\}$, where:

$$r_6 = a \leftarrow not not a. \ r_7 = q \leftarrow not q.$$.

The initial program contains a self-cycle $r_3$, which amounts to a choice of whether $p$ is true or not. Rules $r_3$ and $r_5$, which are derived by 3a and 6 in $f_{SP}(P_1, p)$ 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 $f_{SP}(P_1, p)$, $r_5$ is derived by 3a, as if $r_4$ and $r_5$ (and hence $r_1$ and $r_2$) are independent of each other. $f_{SP}(P_1, p) = \{r_3\}$.

Example 13. Consider forgetting $q$ from the following program $P_{13}$:

$$a \leftarrow not q. \ q \leftarrow b_1, c_1. \ \ldots \ q \leftarrow b_n, c_n.$$.

The programs $f_R(P_{13}, q)$ and $f_{SP}(P_{13}, q)$ 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 $f_R^{ex}$ and $f_{SP}^{ex}$ respectively.
Example 14 (Computational Blowup). Consider forgetting \( q \) from \( P_{14} \):
\[
av \leftarrow q, \quad q \leftarrow b_1, \ldots, b_n.
\]
where
\[
f_R(P_{14}, \{q\}) = f^2_{SP}(P_{14}, \{q\}) = \{a \leftarrow b_1, \ldots, b_n\}
\]
are derived rather quickly through
\[
f_R(P_{14}, \{q\}) = NF(f^2_R(P_{14}, q) \times f^2_R(P_{14}, q))
\]
\[
= \{a \leftarrow b_1, \ldots, b_n\},
\]
where
\[
f^2_R(P_{14}, q) = \{a \leftarrow b_1, \ldots, b_n; \leftarrow \text{not } a, \text{not } b_1, \ldots, \text{not } b_n\}
\]
and
\[
f^2_{SP}(P_{14}, \{q\}) = f_R(P_{14}, \{q\}) \cup f^2_W(P_{14}, q) \cup f^2_W(P_{14}, q)
\]
\[
= f_R(P_{14}, \{q\}) \cup \emptyset \cup \emptyset
\]
\[
= \{a \leftarrow b_1, \ldots, b_n\},
\]
but deriving
\[
f^2_{sem}(P_{14}, \{q\}) = f^{sem}_{SP}(P_{14}, \{q\}) = \{a \leftarrow b_1, \ldots, b_n, \text{not } a; \leftarrow a \leftarrow b_1, \ldots, b_n, \text{not } a\}
\]
Involves checking \(3^{n+2}\) interpretations.

Example 15 (Blowup of the Forgetting Result). Consider forgetting \( q \) from \( P_{15} \):
\[
av \leftarrow b, \text{not } q, \quad c \leftarrow d, \text{not } q, \quad e \leftarrow f, \text{not } q.
\]
then \(f_R(P_{15}, \{q\}) = f^2_{SP}(P_{15}, \{q\}) = \{a \leftarrow b, \leftarrow c \leftarrow d, \text{not } a; \leftarrow e \leftarrow f, \text{not } a\}
\]
while \(f^2_R(P_{15}, \{q\})\) and \(f^{sem}_{SP}(P_{15}, \{q\})\) both contain 127 rules, which, no doubt, in no way resemble \(P_{15}\).

7 Conclusion and Discussion

Strong persistence (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 (SP) have been proposed and investigated. Notably, (SP) decomposes into the three properties (sC), (wC) and (SI), where an operator \(f\) satisfies (SP) iff \(f\) satisfies all (wC) and (SI).

The classes of operators \(f_R\) and \(f_{SPA}\) each guarantee two of those three properties, and in the case of \(f_{SPA}\), even all three, and thus (SP), whenever possible.

\(f_R\) and \(f_{SPA}\) are defined by the semantics of their forgetting result, i.e. they do not define concrete functions, and can 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 \(f_{SPA}\), 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 \(f_R\) resp. \(f_{SPA}\), and therefore that \(f_R\) and \(f_{SPA}\) inherit several (desirable) properties (Gonçalves et al. 2017).

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

Corollary 4. \(f_{SPA}\) satisfies (wC), (SI), (PP) and (SE), but not (wE), (W), (sC), nor (CP).

Further, if \(\{P, V\}\) satisfies \(\neg \Omega\), then \(f_{SPA}\) satisfies (SP)\((P, V)\).

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

In the context of modular logic programming, (SP) can be relaxed to (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 \(F_{UP}\), that guarantees (UP), and interestingly for us, also guarantees (sC) and (wC), with a representative \(f_u \in F_{UP}\) (Gonçalves et al. 2021), that too, if applicable, forgets atoms through syntactic manipulation. In other words, with \(f_R, f^2_{SPA}\), and \(f_u\) at our disposal, we can now syntactically forget satisfying any two properties of (wC), (sC) and (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


