On Syntactic Forgetting with relativized Strong Persistence

Matti Berthold

ScaDS.AI, Universität Leipzig

Abstract

Strong Persistence $(\mathbf{SP})$, since its perception ten years ago, has been at the center of attention in the realm of forgetting in logic programming. So-called forgetting instances, for which it is possible to obtain $(\mathbf{SP})$ were characterized; semantic classes of procedures satisfying $(\mathbf{SP})$ when possible, or various relaxations when it is not, were found; and concrete operators representing these classes were constructed. Recently, $(\mathbf{SP})$ was relaxed in a novel dimension, taking into account the strong persistence of a forgetting result *relativized* to a subset of the remaining atoms. In this paper, we construct a syntactic forgetting operator that satisfies this newly defined desideratum *relativized Strong Persistence* $(\mathbf{rSP})$, whenever possible.

1. Introduction

Logic programming (LP) under answer set semantics is a declarative non-monotonic reasoning formalism with a robust theoretical (and monotonic) foundation based in intuitionistic logic [1]. In essence, answer sets (which are sometimes referred to as stable models) are a second-order notion over classical formulas [2], providing more expressive power than first-order logic, making it possible, for example, to identify Hamiltonian cycles [3].

The question of how a program may be simplified is not simply answered. The surge of research around it, rather suggests that it is very nuanced, where the limits and possibilities vary greatly, depending on the exact definition of what is meant by being *simpler*, and the concrete formalism that is being investigated. Such processes might find application, e.g. in a legal context; in order to exclude dependencies that are no longer deemed relevant; or to reduce the complexity of reasoning tasks.

Forgetting [4, 5, 6], or variable elimination, which is one such possible interpretation of simplification, intuitively means that a programs signature is restricted, while the logical dependencies of the remaining atoms are left unchanged. It has been considered with respect to many properties including *strong persistence* $(\mathbf{SP})$ which seemingly captures best its intuitions [7]. In essence, $(\mathbf{SP})$ ensures that the result of forgetting atoms V from a program P exerts the same behavior as P under the addition of a context program R not containing any forgotten atoms. There are instances for which $(\mathbf{SP})$ is impossible to be achieved [8]. Following this negative result, several relaxations of $(\mathbf{SP})$ were proposed, which are attainable through different semantic means [9, 10, 11].

While results of forgetting using the desired semantics may be obtained by counter-model construction [12] and perhaps be minimized using a version of the Quine-McCluskey algorithm [13], a much more direct and conservative approach is to forget by syntactically manipulating an input program. There are several such syntactical operators in the literature [14, 15, 16, 17, 18, 19, 20], where notably f_{SP}, as its name suggests, satisfies $(\mathbf{SP})$ whenever possible.

Example 1.1. *It is possible that by forgetting an atom we may introduce double negations. E.g. forgetting q from $P = \{a \leftarrow not\, q; q \leftarrow not\, a\}$ results in $P' = \{a \leftarrow not\, not\, a\}$ If an atom is forgotten that appears in such a self-loop, the syntactic derivations are a bit opaque. Consider forgetting q from the following program:*

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

Already, it is impossible to satisfy $(\mathbf{SP})$ [8]. However, a possible result satisfying a relaxation of $(\mathbf{SP})$ may be [17]:

$$a \vee b \leftarrow \qquad b \leftarrow not\, not\, b \qquad a \leftarrow not\, not\, a$$

where a and b each support themselves, and at least one of them is true.

The fact that forgetting in practice should not be done by hand, is probably best witnessed by the fact that forgetting about multiple atoms may not be reduced to forgetting them in iteration. Rather, the result is derived by a recursive derivation tree [19]. [1]

Abstraction by omission [21, 22, 23] can be seen as a relaxed version of forgetting, where atoms are removed from a program, but its answer-sets are only required to behave as before, under no addition of another program R.

The aptly named *Simplification* [24] reduces the signature of a program P by a set of atoms A as well, requiring the result to behave the same as P under a context program R that may contain atoms A in a restricted way.

These different ideas as well as several versions of equivalence were recently captured by an overarching notion of *A-simplifications of P relative to B*, where A is a set of atoms that is to be removed, and B is the signature of possible context programs R [25]. Interestingly, by taking into account the full spectrum of all of these ideas, a novel, relaxed version of $(\mathbf{SP})$, so-called *relativized Strong Persistence* $(\mathbf{rSP})$ emerged.

What is missing from the picture now, is a concrete syntactic transformation $\mathsf{f}_{\mathsf{rSP}}$ that satisfies $(\mathbf{rSP})$, whenever possible. Since $(\mathbf{SP})$ and $(\mathbf{rSP})$ coincide in some cases, it is clear that we should start our search at f_{SP} and see how we get to $\mathsf{f}_{\mathsf{rSP}}$ from there. While the question this paper is out to answer may appear simple at first, its answer is far from it. To be able to construct $\mathsf{f}_{\mathsf{rSP}}$, it turns out that it is necessary to intricately modify sub-procedures of f_{SP}. Finally, with this operator at our disposal, we are then able to construct syntactically general B-relativized A-simplifications.

Given that the class of forgetting operators $\mathsf{F}_{\mathsf{rSS}}$ to satisfy $(\mathbf{rSP})$, whenever possible, is defined methodologically to meet this criterion, and that the operator $\mathsf{f}_{\mathsf{rSP}}$ is defined methodologically to match $\mathsf{F}_{\mathsf{rSS}}$, we assume any intuition about the derivation rules one may find to be 'post hoc'. We therefore leave the task of finding an intuitive explanation of why they are as they are for future studies.

[1] There are accessible implementations of all forgetting operators staying within logic programs available online, including the ones in this paper: https://service.scadsai.uni-leipzig.de/ForgettingWeb https://github.com/mattiberthold/ForgettingWeb

22nd International Workshop on Nonmonotonic Reasoning, November 2-4, 2024, Hanoi, Vietnam

2. Background

Given that $\mathsf{F}_{\mathsf{rSS}}$ and therefore $\mathsf{f}_{\mathsf{rSP}}$ require the understanding of several topics that are 'non-canon' and scientific papers should be self-contained there is a surprising amount of research to be *recalled*, in spite of the fact that the topic of this paper is *forgetting*. Therefore, after recalling the foundations of logic programming, we compile and streamline a rather long list of definitions and results from the literature.

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

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_l, \mathit{not}\, c_1, \ldots, \mathit{not}\, c_m, \\ \mathit{not}\,\mathit{not}\, d_1, \ldots, \mathit{not}\,\mathit{not}\, d_n,$$

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 [26]. Such rules r are also written more succinctly as

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

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

$\Sigma(P)$ 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*[2] *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.[3] 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 $\mathcal{HT}(P_1) = \mathcal{HT}(P_2)$ [27]. Given a set $V \subseteq \Sigma$, the *V-exclusion* of a set of answer sets (a set of HT-interpretations) $\mathcal{M}$, denoted $\mathcal{M}_{\| V}$, is $\{X \backslash V \mid X \in \mathcal{M}\}$ ($\{\langle X\backslash V, Y \backslash V\rangle \mid \langle X, Y\rangle \in \mathcal{M}\}$).

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 [8].

Strong persistence is presumably the best known one [16]. 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: $\mathcal{AS}(P \cup R)_{\| V} = \mathcal{AS}(\mathsf{f}(P, V) \cup R)$ 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: $\mathcal{AS}(\mathsf{f}(P, V)) \supseteq \mathcal{AS}(P)_{\| V}$.

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

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 [8], ($\mathbf{SP}$) was defined for concrete forgetting instances. A forgetting operator f satisfies $(\mathbf{SP})_{\langle P, V\rangle}$, if $\mathcal{AS}(\mathsf{f}(P, V) \cup R) = \mathcal{AS}(P \cup R)_{\| V}$, 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 $(\mathbf{SP})_{\langle P, V\rangle}$.

Definition 2.1 (Gonçalves et al. (2016)). *Let P be a program over Σ and $V \subseteq \Sigma$. The forgetting instance $\langle P, V\rangle$ satisfies criterion Ω if there exists $Y \subseteq \Sigma \backslash V$ such that the set of sets $\mathcal{R}^Y_{\langle P, V\rangle} := \{R^{Y,A}_{\langle P, V\rangle} \mid A \in \mathit{Rel}^Y_{\langle P, V\rangle}\}$ is non-empty and has no least element, where*

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

Corresponding to the Ω criterion the classes F_{R} and F_{SP} specify the HT-models of the forgetting result. It was shown that 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, F_{SP} still exhibits desirable behavior, such as satisfying ($\mathbf{SI}$) and ($\mathbf{wC}$), two of three characterizing criterions of ($\mathbf{SP}$). 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 [9].

The classes F_{R} and F_{SP} are defined as follows:

$$\mathsf{F}_{\mathsf{R}} := \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P, V)) = \{\langle X, Y\rangle \mid Y \subseteq \Sigma(P) \backslash V \wedge$$

[2]Although it is possible to define HT-semantics more broadly over (propositional) formulas, here we use a more succinctly definition over logic programs that is closer to the usual definition of answer sets.

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

$$X \in \bigcup \mathcal{R}^{Y}_{\langle P,V\rangle}\}, \text{ for all programs } P \text{ and } V \subseteq \Sigma\},$$

$$\mathsf{F}_{\mathsf{SP}} := \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X,Y\rangle \mid Y \subseteq \Sigma(P)\backslash V \wedge X \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}\}, \text{ for all programs } P \text{ and } V \subseteq \Sigma\}.$$

Abstraction and Simplification. Abstraction as an over-approximation is defined as follows.

Definition 2.2 (Saributur and Eiter (2018)). *Given P over Σ and Q over Σ' with $|\Sigma| \geqslant |\Sigma'|$, and a mapping $m : \Sigma \to \Sigma' \cup \{\top\}$, Q is an* abstraction *of P w.r.t. m, if $m(\mathcal{AS}(P)) \subseteq \mathcal{AS}(Q)$.*

For an omission abstraction, i.e. $\Sigma' \subseteq \Sigma$, this becomes $(\mathbf{wC})$. An abstraction is called *faithful*, if it additionally satisfies $(\mathbf{sC})$.

This notion was later generalized to take into account context programs, of a certain form, where a program R over Σ is A*-separated*, if there are R_1 over $\Sigma\backslash A$ and R_2 over A, s.t. $R = R_1 \cup R_2$.

Definition 2.3 (Saribatur and Woltran (2023)). *Given, P over Σ, $A \subseteq \Sigma$, and Q over $\Sigma\backslash A$. Q is a* strong A-simplification of P *if for any program R over Σ that is A-separated:*

$$\mathcal{AS}(P \cup R)_{||A} = \mathcal{AS}(Q \cup R_{||A})$$

where:

$$R_{||A} := \{H(r) \leftarrow B(r)\backslash(A \cup \mathit{not\,not}\,A) \mid (H(r) \cup B^{+}(r)) \cap A = \emptyset), r \in R\}$$

P is strong A-simplifiable *if there is such a Q.*

Theorem 2.4 (Saribatur and Woltran (2023)). *If program P is strongly A-simplifiable, then $P_{||A}$ is a strong A-simplification of P.*

Relativized (Strong) Simplification. Recently simplifications have been relaxed to take into account relativized equivalence, i.e. s.t. the simplification Q of P only needs to stay equivalent under addition of any R over a relativized signature $B \subseteq \Sigma$.

Definition 2.5 (Woltran (2004)). *A pair of interpretations $\langle X, Y\rangle$ is a (relativized) B-HT-interpretation iff either $X = Y$ or $X \subset (Y\backslash B)$. The former are called total and the latter non-total B-HT-interpretations. Moreover, a B-HT-interpretation $\langle X, Y\rangle$ is a (relativized) B-HT-model of a program P iff:*

1. *$Y \models P$;*
2. *for all $Y' \subset Y$ with $(Y'\backslash B) = (Y\backslash B) : Y' \not\models P^{Y}$; and*
3. *$X \subset Y$ implies existence of a $X' \subseteq Y$ with $X'\backslash B = X$, such that $X' \models P^{Y}$ holds.*

The set of B-HT-models of P is given by $\mathcal{HT}^{B}(P)$. Two programs P_1 and P_2 are strongly equivalent relative to B iff $\mathcal{HT}^{B}(P_1) = \mathcal{HT}^{B}(P_2)$.

Definition 2.6 (Saribatur and Woltran (2024)). *Given P over Σ, $A, B \subseteq \Sigma$, and Q over $\Sigma\backslash A$, Q is a* (strong) A-simplification of P relative to B *if for any program R over B that is A-separated:*

$$\mathcal{AS}(P \cup R)_{||A} = \mathcal{AS}(Q \cup R_{||A})$$

Program P is B-relativized (strong) A-simplifiable *if there is such a Q.*

The relativized equivalence can similarly be taken into account in forgetting.

Definition 2.7 (Saribatur and Woltran (2024)). *A forgetting operator* f *satisfies* relativized strong persistence *for a relativized forgetting instance $\langle P, V, B\rangle$, $S \subseteq A$, denoted by $(\mathbf{rSP})_{\langle P,V,B\rangle}$, if for all programs R over B, $\mathcal{AS}(\mathsf{f}(P,V,B) \cup R) = \mathcal{AS}(P \cup R)_{||V}$.*

As Ω characterizes instances $\langle P, V\rangle$ for which $(\mathbf{SP})$ is satisfiable, $\Omega_{A,B}$ characterizes instances $\langle P, A, B\rangle$ for which $(\mathbf{rSP})_{\langle P,A,B\rangle}$ is satisfiable.

Definition 2.8 (Saribatur and Woltran (2024)). *Let P be a program over Σ and $A, B \subseteq \Sigma$. P satisfies criterion $\Omega_{A,B}$ if there exists $Y \subseteq \Sigma\backslash A$ such that the set of sets*

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

Proposition 2.9 (Saribatur and Woltran (2024)). *If a forgetting operator* f *satisfies $(\mathbf{SP})_{\langle P,A\rangle}$ then it satisfies $(\mathbf{rSP})_{\langle P,A,B\rangle}$, for any $B \subseteq A$.*

Definition 2.10 (Saribatur and Woltran (2024)). *Given program P over Σ and $A, B \subseteq \Sigma$, the* A-B-HT-models of *P are given by the set*

$$\begin{aligned}\mathcal{HT}^{B}_{A}(P) := &\{\langle Y, Y\rangle_{||A} \mid \langle Y, Y\rangle \in \mathcal{HT}^{B}(P)\} \cup \\ &\{\langle X, Y\rangle_{||A} \mid \langle X, Y\rangle \in \mathcal{HT}^{B}(P), X \subset Y, \\ &\textit{and for all } \langle Y', Y'\rangle \in \mathcal{HT}^{B}(P) \textit{ with } Y'_{||A} = Y_{||A}, \\ &\langle X', Y'\rangle \in \mathcal{HT}^{B}(P) \textit{ with } X' = X_{||A}\}\end{aligned}$$

Definition 2.11 (Saribatur and Woltran (2024)). *Let P be a program. The* relativization of HT-models of P over A to the set B of atoms[4] *is denoted by*

$$\mathcal{HT}^{A,B}(P) := \{\langle X, Y\rangle \mid \langle X, Y\rangle \in \mathcal{HT}^{B}(P), Y \subseteq \Sigma\backslash A\}.$$

Definition 2.12 (Saribatur and Woltran (2024)).

$$\begin{aligned}\mathsf{F}_{\mathsf{rSS}} := \{\mathsf{f} \mid \mathcal{HT}^{A,B}(\mathsf{f}(P,A,B)) = \{\langle X, Y\rangle \mid \\ Y \subseteq \Sigma\backslash V \wedge X \in \bigcap \mathcal{R}^{Y}_{\langle P,A,B\rangle}\}, \\ \textit{for all programs } P \textit{ and } A, B \subseteq \Sigma\}\end{aligned}$$

The class $\mathsf{F}_{\mathsf{rSS}}$ satisfies $(\mathbf{rSP})$ whenever possible.

Theorem 2.13 (Saribatur and Woltran (2024)). *Every $\mathsf{f} \in \mathsf{F}_{\mathsf{rSS}}$ satisfies $(\mathbf{rSP})_{\langle P,A,B\rangle}$, $B \subseteq A$, for every relativized forgetting instance $\langle P, A, B\rangle$, where P does not satisfy $\Omega_{A,B}$.*

The following theorem confirms that forgetting can be used as a stepping-stone to, more generally, derive B-relativized A-simplifications.

Theorem 2.14 (Saribatur and Woltran (2024)). *Let P be B-relativized A-simplifiable, and $\mathsf{f} \in \mathsf{F}_{\mathsf{rSS}}$. Then $\mathsf{f}(P_{||A\cap B}, A\backslash B, B\backslash A)$ is a B-relativized A-simplification of P.*

[4]In order to streamline the presentation the original definition was slightly altered. In particular, $\mathcal{HT}^{A,B}(P)$ behaves as if taking into account the complement $\bar{A}$ of A.

2.1. Syntactic Tools

Defining a syntactic forgetting operators that obeys the semantics of $\mathsf{F}_{\mathsf{rSP}}$ inevitably comes down to modifying the existing operator f_{SP}[5], which is why we recall it and its auxiliary constructions. For succinctness, for examples of established definitions, we refer to [19].

As usual [29, 30, 13, 31, 16, 17] the program is first brought into a normal form, to avoid complications and unnecessary calculations caused by redundant (parts of) rules.

A rule r is *tautological* iff $H(r) \cap B^+(r) \neq \varnothing$, $B^+(r) \cap B^-(r) \neq \varnothing$, or $B^-(r) \cap B^{--}(r) \neq \varnothing$; r is *fundamental*, iff it is not tautological, and $H(r) \cap B^-(r) = \varnothing$ and $B^+(r) \cap B^{--}(r) = \varnothing$.

Definition 2.15 (Cabalar et al. (2007)). *Given two rules r and s, s* subsumes *r, in symbols $s \leqslant r$, 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* $H(s) \cap H(r) = \varnothing$.

Proposition 2.16 (Cabalar et al. (2007)).

$$r \leqslant s \Leftrightarrow \mathcal{HT}(r) \subseteq \mathcal{HT}(s)$$

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

Definition 2.17. *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.*

A program P is in normal form *iff $NF(P) = P$.*

The q-exclusion notation is shorthand to remove an atom.

Definition 2.18 (q-exclusion Berthold (2022)). *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 $L^{\backslash q} := L \backslash \{q, not\, q, not\, not\, q\}$, $r^{\backslash q} := H^{\backslash q}(r) \leftarrow B^{\backslash q}(r)$ and $P^{\backslash q} := \{r^{\backslash q} \mid r \in P\}$.*

We define a partition of a program along the occurrences of a given atom q.

Definition 2.19 (Berthold (2022)). *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:*

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

[5]By f_{SP} we refer to what is called $\mathsf{f}^*_{\mathsf{SP}}$ by Berthold (2022).

There are some correspondences between the models of a program and its rules that we can spot by this partitioning.

Proposition 2.20 (Berthold (2022)). *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 [17] generalizes constructions that collect sets of conjunctions of literals aiming to replace negated occurrences of a literal [15, 16].

Definition 2.21 (Berthold (2022)). *Given a program $P = \{r_1, \ldots, r_n\}$ over Σ and an atom $q \in \Sigma$, then:*

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

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 $not\, not\, not\, not\, p = not\, not\, p$.

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

Proposition 2.22 (Berthold (2022)). *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:*

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

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

The product of rules and programs are defined in order to unite their models.

Definition 2.23 (Product of Rules Berthold (2022)). *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 2.24 (Berthold (2022)). *Let r_1, r_2 be rules over Σ, and $X \subseteq Y \subseteq \Sigma$,*

$$\begin{aligned}
Y \models r_1 \times r_2 &\Leftrightarrow Y \models r_1 \vee Y \models r_2\\
X \models \{r_1 \times r_2\}^Y &\Leftrightarrow X \models \{r_1\}^Y \vee X \models \{r_2\}^Y
\end{aligned}$$

Definition 2.25 (Product of Programs Berthold (2022)). *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 2.26 (Berthold (2022)). *Let P_1, P_2 be programs over Σ, and $X \subseteq Y \subseteq \Sigma$,*

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

The double negation of a rule is such, 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 2.27 (Berthold (2022)). *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 2.28 (Berthold (2022)). *Given a rule r over Σ, and $X \subseteq Y \subseteq \Sigma$, the following statement holds:*

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

We would like to point out that similarly, a formula ψ holds classically iff $\sim\sim\ \psi$ holds intuitionistically. This connection is little surprising, given that HT-logic lies between classical and intuitionistic logic [27]. Further, if we extend the definition of double negation over programs, i.e. $not\,not\,P := \{not\,not\,r \mid r \in P\}$, we are able to construct a program that unites the HT-models of two programs: $\mathcal{HT}(P_1) \cup \mathcal{HT}(P_2) = \mathcal{HT}((P_1 \cup not\,not\,P_1) \times (P_2 \cup not\,not\,P_2))$.

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.

We will also tweak subsumption, since as it is defined above it has some properties that make it impractical to use. For one, it is not anti-symmetrical, two syntactically different rules may subsume each other, such as: $r_1 := \leftarrow not\,not\,a$ and $r_2 := \leftarrow a$, where $r_1 \leqslant r_2$ and $r_2 \leqslant r_1$.

Moreover, even though a rule may subsume another rule, this relation may break, when both of them are conjoined by $\times$ with the same third rule, e.g. let $r_3 := b \leftarrow$, then

$$r_1 \times r_3 = b \leftarrow not\,not\,a < b \leftarrow a = r_2 \times r_3.$$

To avoid these issues, we define a stricter version of subsumption.

Definition 2.29. *Given two fundamental rules r and s, s* strongly subsumes *r, in symbols $s \leqslant_s r$, iff:*

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

Then $s <_s r$, iff $s \leqslant_s r \wedge s \neq r$.

Proposition 2.30. *Strong subsumption is finer than regular subsumption:*

$$s \leqslant_s r \Rightarrow s \leqslant r \tag{1}$$

Strong subsumption is anti-symmetric:

$$s \leqslant_s r \wedge r \leqslant_s s \Rightarrow s = r. \tag{2}$$

Strong subsumption is a greatest subset of regular subsumption, to be anti-symmetric and transitive:

$$s < r \Rightarrow s <_s r \tag{3}$$
$$s \leqslant_s r \wedge r \leqslant_s t \Rightarrow s \leqslant_s t. \tag{4}$$

Strong subsumption is preserved under $\times$:

$$s \leqslant_s r \Rightarrow s \times t \leqslant_s r \times t \text{ for all rules } t. \tag{5}$$

As a consequence, strong subsumption is 'modular' in the following sense:

$$s \leqslant_s r \Leftrightarrow \forall A \subseteq \Sigma : s^{\setminus A} \leqslant_s r^{\setminus A}. \tag{6}$$

Proof:

(1) The requirement for $\leqslant_s$ is stricter than that of $\leqslant$.
(2) Follows from basic set theory and the fact that s and r are fundamental, and therefore $H(t) \cap B^-(t) = \varnothing = B^+(t) \cap B^{--}(t)$ for $t \in \{s, r\}$.
(3) The requirement for $\leqslant_s$ is stricter than that of $\leqslant$.
(4) The subset relation is transitive.
(5) Adding literals to either part of s and r has no effect on the required subset-relationship.
(6) Is a consequence of (5).

■

A rule r is *minimal in P*, iff it is not strongly subsumed by another rule s in P, i.e. iff $\neg\exists s \in P : s <_s r$.

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 2.31 (Berthold 2022). *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 hold:*

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

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. 2.31 constitute the essential pillars of defining forgetting operators.

2.2. Syntactic Forgetting with (SP)

Given that the semantics of $\mathsf{F}_{\mathsf{rSP}}$ and F_{SP} coincide for some inputs, it is not surprising that a representative of $\mathsf{F}_{\mathsf{rSP}}$ needs to be a modification of f_{SP}. We, hence, recall its construction [19]. The operator f_{SP} is defined via two auxiliary operators f_{R} and f_{W}, each of which is again defined using auxiliary operators $\mathsf{f}_{\mathsf{R}}^{A}$ and $\mathsf{f}_{\mathsf{W}}^{A}$, which are defined inductively.

Definition 2.32 ($\mathsf{f}_{\mathsf{R}}^{+}$). *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:*

$$\mathsf{f}_{\mathsf{R}}^{+}(P,q) := nf(\mathbf{1} \cup \mathbf{2} \cup \mathbf{3} \cup \mathbf{4})$$

where:

$$\mathbf{1} := \{\leftarrow D \mid D \in \mathcal{D}^{q}_{as}(R_3 \cup R_4)\}$$
$$\mathbf{2} := \{not\,not\,r^{\backslash q} \mid r \in R_0 \cup R_2\}$$
$$\mathbf{3} := \{r^{\backslash q} \mid r \in R \cup R_2\}$$
$$\mathbf{4} := \{(r_0 \times r')^{\backslash q} \mid r_0 \in R_0, r' \in R_3 \cup R_4\}$$

Proposition 2.33. *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:*

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

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

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

Definition 2.34 ($\mathsf{f}_{\mathsf{R}}^{-}$). *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:*

$$\mathsf{f}_{\mathsf{R}}^{-}(P,q) := nf(\mathbf{1} \cup \mathbf{2})$$

where:

$$\mathbf{1} := \{r'^{\backslash q} \mid r' \in R \cup R_1 \cup R_4\}$$
$$\mathbf{2} := \{not\,not\,r'^{\backslash q} \mid r' \in R_1 \cup R_4\}$$

The operators $\mathsf{f}_{\mathsf{R}}^{A}$ is defined inductively by nested calls on $\mathsf{f}_{\mathsf{R}}^{+}$ and $\mathsf{f}_{\mathsf{R}}^{-}$. In order to fix a concrete forgetting result, we assume an arbitrary order on V, which has no effect on the following propositions.

Definition 2.35 ($\mathsf{f}_{\mathsf{R}}^{A}$). *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:*

$$\mathsf{f}_{\mathsf{R}}^{A}(P, \varnothing) := P$$
$$\mathsf{f}_{\mathsf{R}}^{A}(P, V) := \mathsf{f}_{\mathsf{R}}^{\otimes n}(\mathsf{f}_{\mathsf{R}}^{A\backslash q_n}(P, V\backslash\{q_n\}), q_n)$$

where:

$$\mathsf{f}_{\mathsf{R}}^{\otimes n} := \begin{cases} \mathsf{f}_{\mathsf{R}}^{+}, & \text{if } q_n \in A \\ \mathsf{f}_{\mathsf{R}}^{-}, & \text{otherwise} \end{cases}$$

Proposition 2.36. *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*

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

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

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

Definition 2.37 (f_{R}). *Let P be a program over Σ in normal form and $V \subseteq \Sigma$.*

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

Theorem 2.38. $\mathsf{f}_{\mathsf{R}} \in \mathsf{F}_{\mathsf{R}}$

The operator f_{W}, which contradicts any $\langle X, Y\rangle$ for which $Rel^{Y}_{\langle P,V\rangle} \neq \varnothing$ and $X \notin \mathcal{R}^{Y}_{\langle P,V\rangle}$, is again defined by induction. By uniting f_{R} with f_{W}, we are then able to construct f_{SP}.

Definition 2.39 ($\mathsf{f}_{\mathsf{W}}^{+}$). *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:*

$$\mathsf{f}_{\mathsf{W}}^{+}(P,q) := NF(\mathbf{1} \cup \mathbf{2})$$

where:

$$\mathbf{1} := \{(r_0 \times r' \times not\,not\,r)^{\backslash q} \times \leftarrow D \mid r_0 \in R_0,\ r, r' \in R_3 \cup R_4, D \in \mathcal{D}^{q}_{as}(R_0 \cup R_2)\}$$
$$\mathbf{2} := \{(r \times not\,not\,r')^{\backslash q} \times \leftarrow D \mid r \in R \cup R_2, r' \in R_3 \cup R_4, D \in \mathcal{D}^{q}_{as}(R_0 \cup R_2)\}$$

Definition 2.40 ($\mathsf{f}_{\mathsf{W}}^{-}$). *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:*

$$\mathsf{f}_{\mathsf{W}}^{-}(P,q) := NF(\mathbf{1})$$

where:

$$\mathbf{1} := \{r'^{\backslash q} \times \leftarrow D \mid r' \in R \cup R_1 \cup R_4, D \in \mathcal{D}^{q}_{as}(R_1 \cup R_4)\}$$

Definition 2.41 ($\mathsf{f}_{\mathsf{W}}^{A}$). *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:*

$$\mathsf{f}_{\mathsf{W}}^{A}(P, \varnothing) := P$$
$$\mathsf{f}_{\mathsf{W}}^{A}(P, V) := \mathsf{f}_{\mathsf{W}}^{\otimes n}(\mathsf{f}_{\mathsf{W}}^{A\backslash q_n}(P\backslash R, V\backslash\{q_n\}), q_n) \cup R$$

where:

$$\mathsf{f}_{\mathsf{W}}^{\otimes n} := \begin{cases} \mathsf{f}_{\mathsf{W}}^{+}, & \text{if } q_n \in A \\ \mathsf{f}_{\mathsf{W}}^{-}, & \text{otherwise} \end{cases}$$
$$R := \{r \in P \mid V \cap \Sigma(r) = \varnothing\}$$

Proposition 2.42. *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:*

$$A \in Rel^{Y}_{\langle P,V\rangle} \wedge \forall A'' \subseteq A : \langle XA'', YA\rangle \not\models P$$
$$\Leftrightarrow \langle X, Y\rangle \not\models \mathsf{f}_{\mathsf{W}}^{A}(P,V)$$

Definition 2.43 (f_{W}). *Given a program P in normal form over Σ and $V \subseteq \Sigma$. Then:*

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

Definition 2.44 ($\mathsf{f}_{\mathsf{SP}}^{*}$). *Let P be a program over Σ in normal form and $V \subseteq \Sigma$.*

$$\mathsf{f}_{\mathsf{SP}}(P,V) := NF(\mathsf{f}_{\mathsf{W}}(P,V) \cup \mathsf{f}_{\mathsf{R}}(P,V))$$

Example 2.45. *Let $P_{2.45} = \{a \leftarrow b, q; c \leftarrow d, not\,q; q \leftarrow not\,not\,q\}$, and $V = \{p, q\}$. Then $\mathsf{f}_{\mathsf{SP}}(P,V)$ can be derived by:*

$$\mathsf{f}_{\mathsf{R}}^{\{q\}}(P,V) \subseteq \{c \leftarrow d\} \quad \mathsf{f}_{\mathsf{W}}^{\{q\}}(P,V) = \{a \leftarrow b, not\,not\,a\}$$
$$\mathsf{f}_{\mathsf{R}}^{\varnothing}(P,V) \subseteq \{a \leftarrow b\} \quad \mathsf{f}_{\mathsf{W}}^{\varnothing}(P,V) = \{c \leftarrow d, not\,not\,c\}$$

$$\begin{aligned}\mathsf{f}_{\mathsf{SP}}(P,V) = {} & NF(\mathsf{f}_{\mathsf{R}}^{\{q\}}(P,V) \times \mathsf{f}_{\mathsf{R}}^{\varnothing}(P,V) \\ & \cup\ \mathsf{f}_{\mathsf{W}}^{\{q\}}(P,V) \cup \mathsf{f}_{\mathsf{W}}^{\varnothing}(P,V)) \\ = {} & \{a \vee c \leftarrow b, d; a \leftarrow b, not\,not\,a; c \leftarrow d, not\,not\,d\}\end{aligned}$$

Theorem 2.46. $\mathsf{f}_{\mathsf{SP}} \in \mathsf{F}_{\mathsf{SP}}$

3. Towards Syntactic Forgetting with (rSP)

The idea in the following constructions is to modify the results of the previous operators, to take into account a set B to relativize to. As witnessed in the previous section, half of the construction of $\mathsf{f_{SP}}$ is a member of another class $\mathsf{F_R}$. We define a relaxation $\mathsf{F_{rR}}$ of this class too, to aim for first.

$$\begin{aligned}\mathsf{F_{rR}} := \{\mathsf{f} \mid \mathcal{HT}^{A,B}(\mathsf{f}(P,A,B)) = \{\langle X,Y\rangle \mid \\ Y \subseteq \Sigma\backslash V \wedge X \in \bigcup \mathcal{R}^Y_{\langle P,A,B\rangle}\}, \\ \text{for all programs } P \text{ and } A, B \subseteq \Sigma\}\end{aligned}$$

While the 'r' in $(\mathbf{rSP})$ and the 'R' in $\mathsf{F_R}$ both mean 'relativized', it is not clear in which sense $\mathsf{F_R}$ corresponds to the idea of relativized equivalence. Still we use the subscript 'rR' for this new class in reference to its origin in $\mathsf{F_R}$.

Assume a program P over Σ, $V, B \subseteq \Sigma$, s.t. $V \cap B = \emptyset$, and $Y \subseteq B$. If we take a look at the definition of the class $\mathsf{F_{rR}}$, we can note that there is a similarity in how it treats forgotten atoms V and atoms that it relativizes from $\Sigma(P)\backslash B$:

- Given $A \subseteq V$, a total model $\langle YA, YA\rangle$ of P, s.t. there is a $A' \subseteq A$ with $\langle YA', YA\rangle \models P$ is not considered by $\mathcal{R}^Y_{\langle P,V,B\rangle}$;
- Similarly, given $C \subseteq \Sigma(P)\backslash(B \cup V)$, a total model $\langle YC, YC\rangle$ of P, s.t. there is a $C' \subseteq C$ with $\langle YC', YC\rangle \models P$ is not considered by $\mathcal{R}^Y_{\langle P,V,B\rangle}$, by the construction of $\mathcal{HT}^B(P)$.

The operator $\mathsf{f_R}$ includes an encoding for the first bullet-point. The key-idea therefore is to manipulate the auxiliary constructions of $\mathsf{f_R}$ further, to encode the second bullet-point.

For each $A \subseteq V$ and $C \subseteq \Sigma(P)\backslash(B \cup V)$, we define an auxiliary operator $\mathsf{g}_\mathsf{R}^{A,C}$ that determines for any $Y \subseteq (\Sigma(P) \cap B)\backslash V$ whether (i) $Y \cup A \cup C \models P$, and whether (ii) there are no $A' \subseteq A$ and $C' \subseteq C$, s.t. $A' \cup B' \subset A \cup B$, and $\langle YA'C', YAC\rangle \models P$. Then $\mathsf{g}_\mathsf{R}^{A,C}(P,V)$, satisfies $\langle YC, YC\rangle$, iff (i) and (ii). Further, we let $\mathsf{g}_\mathsf{R}^{A,C}(P,V)$ contradict each $\langle XC'', YC\rangle$ with $X \subset Y$, $C'' \subseteq C$, iff $P \not\models \langle YA'C', YAC\rangle$ for all $A' \subset A$ and $C' \subset C$.

The operators $\mathsf{g}_\mathsf{R}^{A,C}$ are defined taking into account f_R^A as a baseline, i.e. $\mathsf{g}_\mathsf{R}^{A,C}(P,V,B) := \mathsf{g}_\mathsf{R}^C(\mathsf{f}_\mathsf{R}^A(P,V),B)$. These g_R^C we define inductively, starting at $|C| = 1$.

Definition 3.1 (g_R^+). *Given a program P in normal form over Σ and $c \in \Sigma$ s.t. $\mathsf{occ}(P,c) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$. Then:*

$$\mathsf{g}_\mathsf{R}^+(P,c) := nf(\mathbf{1} \cup \mathbf{2} \cup \mathbf{3} \cup \mathbf{4} \cup \mathbf{5})$$

where:

$$\begin{aligned}\mathbf{1} &:= \{\leftarrow D \mid D \in \mathcal{D}^c_{as}(R_3 \cup R_4)\}\\ \mathbf{2} &:= \{r \mid r \in R \cup R_2\}\\ \mathbf{3} &:= \{not\,not\,r \mid r \in R_0 \cup R_2\}\\ \mathbf{4} &:= \{r' \mid r' \in R_3 \cup R_4\}\\ \mathbf{5} &:= \{\leftarrow not\, c\}\end{aligned}$$

Proposition 3.2. *Given a program P over Σ, an atom $c \in \Sigma$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma$, then:*

$$Y \models \mathsf{g}_\mathsf{R}^+(P,c) \Leftrightarrow Y \models P \wedge c \in Y \wedge Y\backslash\{c\} \not\models P^Y$$

If $Y \models \mathsf{g}_\mathsf{R}^+(P,c)$, then:

$$\langle X,Y\rangle \models \mathsf{f}_\mathsf{R}^+(P,q) \Leftrightarrow \langle X,Y\rangle \models P$$

Definition 3.3 (g_R^-). *Given a program P in normal form over Σ and $c \in \Sigma$ s.t. $\mathsf{occ}(P,c) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$. Then:*

$$\mathsf{g}_\mathsf{R}^-(P,c) := nf(\mathbf{1} \cup \mathbf{2} \cup \mathbf{3})$$

where:

$$\begin{aligned}\mathbf{1} &:= \{r' \mid r' \in R \cup R_1 \cup R_4\}\\ \mathbf{2} &:= \{not\,not\,r' \mid r' \in R_1 \cup R_4\}\\ \mathbf{3} &:= \{\leftarrow not\,not\,c\}\end{aligned}$$

Proposition 3.4. *Given a program P over Σ, an atom $c \in \Sigma$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma$, then:*

$$Y \models \mathsf{g}_\mathsf{R}^+(P,c) \Leftrightarrow c \notin Y \wedge Y \models P$$

If $Y \models \mathsf{g}_\mathsf{R}^+(P,c)$, then:

$$\langle X,Y\rangle \models \mathsf{g}_\mathsf{R}^-(P,c) \Leftrightarrow \langle X,Y\rangle \models P$$

As before, in order to fix a concrete forgetting result, we assume an arbitrary order on C.

Definition 3.5 (g_R^C). *Let P be a program over Σ, $B \subseteq \Sigma$ and*
$C \subseteq \{c_1, c_2, \ldots, c_n\} = \bar{B} := \Sigma(P)\backslash B$, s.t. $0 < n$, then:

$$\begin{aligned}\mathsf{g}_\mathsf{R}^C(P,\emptyset) &:= P\\ \mathsf{g}_\mathsf{R}^C(P,B) &:= \mathsf{g}_\mathsf{R}^{\otimes n}(\mathsf{g}_\mathsf{R}^{C\backslash c_n}(P, C\backslash\{c_n\}), c_n)\end{aligned}$$

where:

$$\mathsf{g}_\mathsf{R}^{\otimes n} := \begin{cases}\mathsf{g}_\mathsf{R}^+, & \textit{if } c_n \in C\\ \mathsf{g}_\mathsf{R}^-, & \textit{otherwise}\end{cases}$$

Proposition 3.6. *Given a program P over Σ, sets of atoms $B \subseteq \Sigma$, $C \subseteq \bar{B} := \Sigma(P)\backslash B$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma$ and $X \subseteq B$, then:*

$$\begin{aligned}&Y \models \mathsf{g}_\mathsf{R}^C(P,B)\\ \Leftrightarrow\ &Y \models P \wedge\ Y\backslash B = C\\ &\wedge \nexists Y' \subset Y,\ \text{s.t. } Y' \models P^Y \text{ and } Y' \cap B = Y \cap B\end{aligned}$$

If $Y \models \mathsf{g}_\mathsf{R}^C(P,B)$, then for each $C' \subseteq C$:

$$\langle XC', Y\rangle \models \mathsf{g}_\mathsf{R}^C(P,B) \Leftrightarrow \langle XC', Y\rangle \models P$$

Definition 3.7 ($\mathsf{g_R}$). *Let P be a program over Σ in normal form and $V, B \subseteq \Sigma$. s.t. $V \cap B = \emptyset$. $\bar{B} := \Sigma(P)\backslash B$.*

$$\mathsf{g_R}(P,V,B) := NF(\bigtimes_{\substack{A\subseteq V\\ C\subseteq \bar{B}}} \mathsf{g}_\mathsf{R}^{A,C}(P,V,B))$$

where

$$\mathsf{g}_\mathsf{R}^{A,C}(P,V,B) := \mathsf{g}_\mathsf{R}^C(\mathsf{f}_\mathsf{R}^A(P,V),B)$$

For an illustration of how $\mathsf{g_R}$ functions we refer to Figure 1 on the last page.

Theorem 3.8. $\mathsf{g_R} \in \mathsf{F_{rR}}$

4. Syntactic Forgetting with (rSP)

Again we define g_{W} s.t. it modifies an auxiliary result of f_{W} to take into account whether a $C \subseteq \bar{B} := \Sigma(P)\backslash B$ is relevant.

Remember, that the auxiliary operators $\mathsf{f}^{A}_{\mathsf{W}}$ implement a check for whether A is *relevant* for Y ($A \in Rel^{Y}_{\langle P,V\rangle}$), i.e. that $\langle YA, YA\rangle$ is a model of P, but $\langle YA', YA\rangle$ is not a model of P for all $A' \subset A$. As we have seen in the last section this requirement extends to relativized forgetting, in the sense that we additionally need to check whether $Y \cup C$ can be a stable model of P under addition of rules over B – $\mathsf{g}^{C}_{\mathsf{W}}$ checks, whether $\langle YC, YC\rangle \models P$ and $\langle YC', YC\rangle \VDash P$ for all $C' \subset C$.

By compounding the constructions $\mathsf{f}^{A}_{\mathsf{W}}$ with $\mathsf{g}^{C}_{\mathsf{W}}$, i.e. $\mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B) := \mathsf{g}^{C}_{\mathsf{W}}(\mathsf{f}^{A}_{\mathsf{W}}(P,V),B)$, we get operators with the following properties.

Given a program P over Σ, $V, B \subseteq \Sigma$ with $V \cap B = \emptyset$, $X \subseteq Y \subseteq \Sigma \cap B$, $A \subseteq V$, and $C' \subseteq C \subseteq \Sigma(P)\backslash B$, then, $\mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B)$ contradicts $\langle XC', YC\rangle$ i.e. $\langle XC', YC\rangle \VDash \mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B)$, iff $\langle YCA, YCA\rangle \models P$, for all $A' \subseteq A$ and $C'' \subseteq C$ with $A' \cup C'' \subset A \cup C$: $\langle YA'C'', YAC\rangle \VDash P$, and for all $A' \subseteq A$: $\langle XC'A', YAC\rangle \VDash P$.

The auxiliary operators $\mathsf{g}^{A}_{\mathsf{W}}$ are again inductively defined via $\mathsf{g}^{+}_{\mathsf{W}}$ and $\mathsf{g}^{-}_{\mathsf{W}}$.

Definition 4.1 ($\mathsf{g}^{+}_{\mathsf{W}}$). *Given a program P in normal form over Σ and $c \in \Sigma$ s.t. $\mathsf{occ}(P,c) = \langle R, R_0, R_1, R_2, R_3, R_4\rangle$, then:*

$$\mathsf{g}^{+}_{\mathsf{W}}(P,c) := NF(\boldsymbol{1} \cup \boldsymbol{2})$$

where:

$$\begin{aligned}\boldsymbol{1} := \ & \{r' \times not\,not\,r \times\ \leftarrow D \mid \\ & \quad r, r' \in R_3 \cup R_4, D \in \mathcal{D}^{c}_{as}(R_0 \cup R_2)\}\\ \boldsymbol{2} := \ & \{(r \times not\,not\,r') \times\ \leftarrow D \mid \\ & \quad r \in R \cup R_2, r' \in R_3 \cup R_4, D \in \mathcal{D}^{c}_{as}(R_0 \cup R_2)\}\end{aligned}$$

The building-blocks $\mathsf{g}^{+}_{\mathsf{W}}$ and $\mathsf{g}^{-}_{\mathsf{W}}$ of our inductive definition take into account one atom that is relativized away. We can therefore again use the observations of Prop. 2.31 to see that for the case $|C| = 1$ they have the desired properties.

Proposition 4.2. *Given a program P over Σ, an atom $c \in \Sigma$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma$, then:*

$$\begin{aligned}&\langle X, Y\rangle \VDash \mathsf{g}^{+}_{\mathsf{R}}(P,c)\\ \Leftrightarrow\ & Y \models P \wedge c \in Y \wedge Y\backslash\{c\} \VDash P^{Y} \wedge \langle X, Y\rangle \VDash P\end{aligned}$$

Definition 4.3 ($\mathsf{g}^{-}_{\mathsf{W}}$). *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:*

$$\mathsf{g}^{-}_{\mathsf{W}}(P,c) :=\ NF(\boldsymbol{1})$$

where:

$$\boldsymbol{1} := \{r' \times\ \leftarrow D \mid r' \in R \cup R_1 \cup R_4, D \in \mathcal{D}^{q}_{as}(R_1 \cup R_4)\}$$

Proposition 4.4. *Given a program P over Σ, an atom $c \in \Sigma$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma$, then:*

$$\begin{aligned}&\langle X, Y\rangle \VDash \mathsf{g}^{-}_{\mathsf{R}}(P,c)\\ \Leftrightarrow\ & c \notin Y \wedge Y \models P \wedge \langle X, Y\rangle \VDash P\end{aligned}$$

To define $\mathsf{g}^{C}_{\mathsf{W}}$ for arbitrary $C \subseteq \Sigma(P)\backslash B$, we assume an arbitrary ordering on Σ, e.g. the lexicographic order, and apply repeatedly the operators $\mathsf{g}^{+}_{\mathsf{W}}$ or $\mathsf{g}^{-}_{\mathsf{W}}$, depending on whether an atom c is in C. For example, let P be over $\{a,b,c,d\}$, $B = \{a,b\}$ and $C = \{d\}$, then $\mathsf{g}^{C}_{\mathsf{W}}(P,B) = \mathsf{g}^{+}_{\mathsf{W}}(\mathsf{g}^{-}_{\mathsf{W}}(P,c),d)$.

Definition 4.5 ($\mathsf{g}^{C}_{\mathsf{W}}$). *Let P be a program over Σ, $B \subseteq \Sigma$ and*
$C \subseteq \{c_1, c_2, \ldots, c_n\} = \bar{B} := \Sigma(P)\backslash B$, s.t. $0 < n$, then:

$$\begin{aligned}&\mathsf{g}^{C}_{\mathsf{W}}(P,\emptyset) := P\\ &\mathsf{g}^{C}_{\mathsf{W}}(P,B) := \mathsf{g}^{\otimes n}_{\mathsf{W}}(\mathsf{g}^{C\backslash c_n}_{\mathsf{W}}(P\backslash R, \bar{B}\backslash\{c_n\}), c_n) \cup R\end{aligned}$$

where:

$$\mathsf{g}^{\otimes n}_{\mathsf{W}} := \begin{cases}\mathsf{g}^{+}_{\mathsf{W}}, & \textit{if } c_n \in C\\ \mathsf{g}^{-}_{\mathsf{W}}, & \textit{otherwise}\end{cases}$$
$$R := \{r \in P \mid \Sigma(r) \subseteq B\}$$

The fact that the properties of $\mathsf{g}^{+}_{\mathsf{W}}$ and $\mathsf{g}^{-}_{\mathsf{W}}$ extend to $\mathsf{g}^{C}_{\mathsf{W}}$ can be checked rather straight-forwardly by induction.

Proposition 4.6. *Given a program P over Σ, sets of atoms $B \subseteq \Sigma$, $C \subseteq \Sigma(P)\backslash B$, and sets X and Y, s.t. $X \subset Y \subseteq \Sigma$, then:*

$$\begin{aligned}&\langle X, Y\rangle \VDash \mathsf{g}^{C}_{\mathsf{W}}(P,B)\\ \Leftrightarrow\ & Y \models P \wedge\ Y\backslash B = C\\ & \wedge \nexists Y' \subset Y, \text{ s.t. } Y' \models P^{Y} \text{ and } Y' \cap B = Y \cap B\\ & \wedge \langle X, Y\rangle \VDash P\end{aligned}$$

To construct g_{W}, $\mathsf{f}^{A}_{\mathsf{W}}$ and $\mathsf{g}^{C}_{\mathsf{W}}$ are compounded for each $A \subseteq V$ and $C \subseteq \Sigma(P)\backslash B$, i.e. $\mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B) := \mathsf{g}^{C}_{\mathsf{W}}(\mathsf{f}^{A}_{\mathsf{W}}(P,V),B)$. The resulting rules of each of these compounds are then united.

Definition 4.7 (g_{W}). *Given a program P in normal form over Σ and $V \subseteq \Sigma$. Then:*

$$\mathsf{g}_{\mathsf{W}}(P,V,B) := NF(\bigcup_{\substack{A \subseteq V\\ C \subseteq \bar{B}}} \mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B))$$

where:

$$\mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B) := \mathsf{g}^{C}_{\mathsf{W}}(\mathsf{f}^{A}_{\mathsf{W}}(P,V),B)$$

We would like to remark here that, while this construction may appear rather costly computationally, some factors that may dampen the blow-up that have been discussed in non-relativized forgetting [19], also apply here.

Most importantly, the operator $\mathsf{g}^{A,C}_{\mathsf{W}}$ is such that its result is the empty program, if the combination A, C is 'non-relevant'. If this 'non-relevancy' is detected within a recursive step of $\mathsf{g}^{A,C}_{\mathsf{W}}$ possibly exponentially many calculations can be disregarded.

More concretely, assume for example a program P over $\{a, \ldots, z\}$, $V = \{p, \ldots, z\}$ and $B = \{a, \ldots, h\}$. If $\mathsf{f}^{+}_{\mathsf{W}}(P,p) = \emptyset$, then $\mathsf{g}^{A,C}_{\mathsf{W}}(P,V,B)$ will be the empty program for any $A \supseteq \{p\}$ and any C, which lets us disregard a large part of the recursive calculation-tree.

Definition 4.8 ($\mathsf{f}_{\mathsf{rSP}}$). *Let P be a program over Σ in normal form and $V \subseteq \Sigma$.*

$$\mathsf{f}_{\mathsf{rSP}}(P,V) := NF(\mathsf{g}_{\mathsf{W}}(P,V) \cup \mathsf{g}_{\mathsf{R}}(P,V))$$

Theorem 4.9. $\mathsf{f}_{\mathsf{rSP}} \in \mathsf{F}_{\mathsf{rSS}}$

Corollary 4.10. *Let P be B-relativized A-simplifiable, then $\mathsf{f}_{\mathsf{rSP}}(P_{||A \cap B}, A \backslash B, B)$ is a B-relativized A-simplification of P.*

5. Let's not Forget about Predicates

It remains an open question, as to how forgetting propositional atoms from a program translates to the more general case of forgetting from a program with variables. As has been done for classical formulas [4] one may consider forgetting about terms, ground atoms, or predicate symbols, where the latter probably comes closest to the propositional case. When forgetting about predicate symbols, two obstacles come to mind. (i) A predicate may be recursive, making it impossible to find a (finite) forgetting result. E.g.: consider forgetting t from the following program:

$$t(X,Y) \leftarrow e(X,Y). \qquad a(X,Y) \leftarrow t(X,Y), b(X,Y).$$
$$t(X,Z) \leftarrow t(X,Y), e(Y,Z).$$

One may consider finite forgetting results that have desirable properties up to some bound, as has been similarly done for classical logic [32]. (ii) Even if a non-recursive predicate symbol is forgotten we may have to leave the class of logic programs to represent a result of forgetting. E.g. consider forgetting about about b from the following program where b marks all pairs which are connected through two edges:

$$b(X,Z) \leftarrow e(X,Y), e(Y,Z). \qquad n(X) \leftarrow e(X,Y).$$
$$a(X,Y) \leftarrow \mathit{not}\, b(X,Y), n(X), n(Y). \quad n(Y) \leftarrow e(X,Y).$$

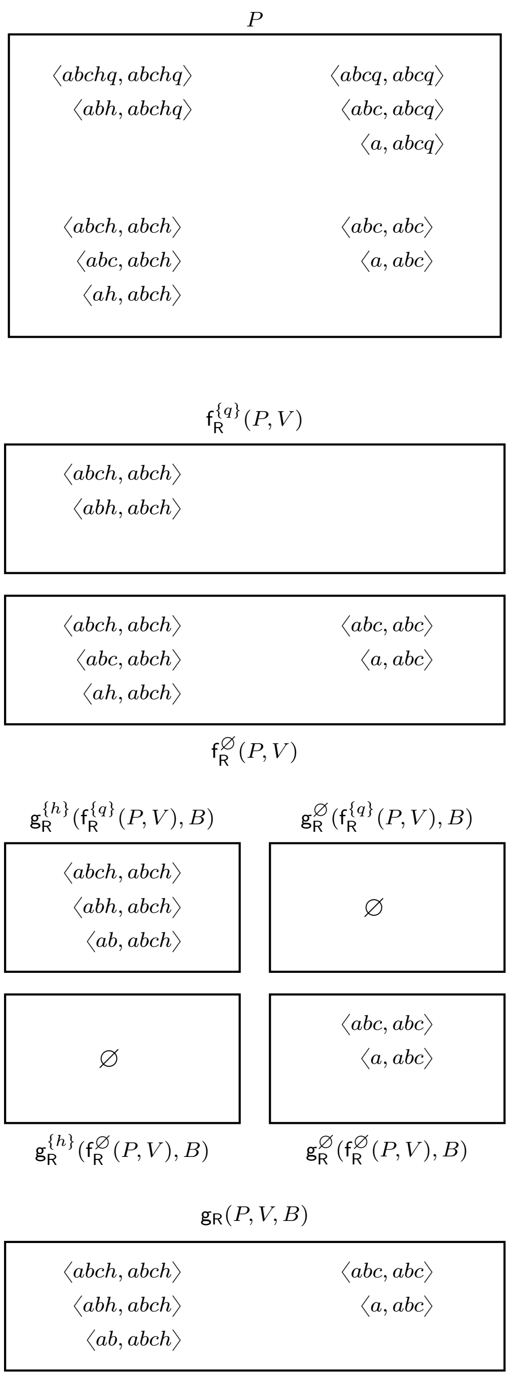
A possible forgetting result in full first order syntax is ψ:

$$\begin{aligned} \forall X,Z : (\neg\exists Y : (e(X,Y) \wedge e(Y,Z)) \wedge n(X) \wedge n(Y) \\ \to a(X,Z) \\ \wedge\, e(X,Z) \to (n(X) \wedge n(Z))) \end{aligned}$$

where the impossiblility of ψ to be put into a prenex-normalform, is inherited from intuitionism. It may be worthwhile to consider subclasses of the full first-order syntax that are well behaved w.r.t. forgetting. Related to this question there are two extensions of logic programs that are able to capture the full polynomial hierarchy, stable-unstable programs [33] and logic programs with quantifiers [34]. A relaxed version of forgetting from logic programs, so-called interpolation has recently been successfully reduced to the classical case [35].

6. Conclusion

The question on how a logic program may be simplified, has become a rather large one, sparking several subtopics that cover different particular aims: forgetting, abstraction, simplification. These ideas have recently been captured under an umbrella-term of (strong) A-simplifications of P relativized to B [25]. The existence of this more abstract version of forgetting tore open a hole between the semantics and syntax that that was just recently closed [19]. In this paper we were able to close it again by intricately modifying f_{SP}, to be able to take into account a relativization set. Given that most of the recent results are limited to the propositional case, we believe that it would be interesting to explore how they translate to forgetting about predicate-symbols next.


P
$\langle abchq, abchq\rangle$
$\langle abh, abchq\rangle$
$\langle abcq, abcq\rangle$
$\langle abc, abcq\rangle$
$\langle a, abcq\rangle$
$\langle abch, abch\rangle$
$\langle abc, abch\rangle$
$\langle ah, abch\rangle$
$\langle abc, abc\rangle$
$\langle a, abc\rangle$
$\mathsf{f}_{\mathsf{R}}^{\{q\}}(P,V)$
$\langle abch, abch\rangle$
$\langle abh, abch\rangle$
$\langle abch, abch\rangle$
$\langle abc, abch\rangle$
$\langle ah, abch\rangle$
$\langle abc, abc\rangle$
$\langle a, abc\rangle$
$\mathsf{f}_{\mathsf{R}}^{\varnothing}(P,V)$
$\mathsf{g}_{\mathsf{R}}^{\{h\}}(\mathsf{f}_{\mathsf{R}}^{\{q\}}(P,V),B)$
$\mathsf{g}_{\mathsf{R}}^{\varnothing}(\mathsf{f}_{\mathsf{R}}^{\{q\}}(P,V),B)$
$\langle abch, abch\rangle$
$\langle abh, abch\rangle$
$\langle ab, abch\rangle$
$\varnothing$
$\varnothing$
$\langle abc, abc\rangle$
$\langle a, abc\rangle$
$\mathsf{g}_{\mathsf{R}}^{\{h\}}(\mathsf{f}_{\mathsf{R}}^{\varnothing}(P,V),B)$
$\mathsf{g}_{\mathsf{R}}^{\varnothing}(\mathsf{f}_{\mathsf{R}}^{\varnothing}(P,V),B)$
$\mathsf{g}_{\mathsf{R}}(P,V,B)$
$\langle abch, abch\rangle$
$\langle abh, abch\rangle$
$\langle ab, abch\rangle$
$\langle abc, abc\rangle$
$\langle a, abc\rangle$
Figure 1: This figure illustrates how g_{R} takes a divide and conquer approach to be able to encode the semantics of F_{rR}. Here we abstract away from the specific syntax of each auxiliary program and only look at their models. For each set of models in a box such a representative exists [13]. The models of an initial program P are on the top most box, the models of the auxiliary results of forgetting $V = \{q\}$ relativized to $B = \{a,b,c\}$ are listed from there on downward. The workings of $\mathsf{f}_{\mathsf{rSP}}$ follow a similar pattern, but are hard to put into an illustration, since the auxiliary operators $\mathsf{g}_{\mathsf{W}}^{V,B}$ satisfy a lot more models.

Acknowledgements

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

[1] A. Heyting, Die formalen regeln der intuitionistischen logik, in: Sitzungsberichte der Preussischen Akademie der Wissenschaften. Physikalisch-mathematische Klasse, 1930.

[2] V. Lifschitz, D. Pearce, A. Valverde, A characterization of strong equivalence for logic programs with variables, in: Proceedings of (LPNMR-07), 2007. doi:10.1007/978-3-540-72200-7_17.

[3] Y. Chen, Y. Zhang, Y. Zhou, First-order indefinability of answer set programs on finite structures, in: Proceedings of (AAAI-10), 2010.

[4] F. Lin, R. Reiter, Forget it, in: Working Notes of AAAI Fall Symposium on Relevance, 1994.

[5] J. P. Delgrande, A knowledge level account of forgetting, Journal of Artificial Intelligence Research (2017). doi:10.1613/jair.5530.

[6] T. Eiter, G. Kern-Isberner, A brief survey on forgetting from a knowledge representation and reasoning perspective, KI - Künstliche Intelligenz (2018). doi:10.1007/s13218-018-0564-6.

[7] R. Gonçalves, M. Knorr, J. Leite, The ultimate guide to forgetting in answer set programming, in: Proceedings of (KR-16), 2016.

[8] R. Gonçalves, M. Knorr, J. Leite, You can't always forget what you want: On the limits of forgetting in answer set programming, in: Proceedings of (ECAI-16), 2016. doi:10.3233/978-1-61499-672-9-957.

[9] R. Gonçalves, M. Knorr, J. Leite, S. Woltran, When you must forget: Beyond strong persistence when forgetting in answer set programming, Theory and Practice of Logic Programming (2017).

[10] R. Gonçalves, T. Janhunen, M. Knorr, J. Leite, S. Woltran, Forgetting in modular answer set programming, in: Proceedings of (AAAI-19), 2019. doi:10.1609/aaai.v33i01.33012843.

[11] R. Gonçalves, T. Janhunen, M. Knorr, J. Leite, On syntactic forgetting under uniform equivalence, in: Proceedings of (JELIA-21), volume 12678, 2021, pp. 297–312. doi:10.1007/978-3-030-75775-5_20.

[12] P. Cabalar, P. Ferraris, Propositional theories are strongly equivalent to logic programs, Theory and Practice of Logic Programming (2007).

[13] P. Cabalar, D. Pearce, A. Valverde, Minimal logic programs, in: Proceedings of (ICLP-07), 2007.

[14] Y. Zhang, N. Y. Foo, Solving logic program conflict through strong and weak forgettings, Artificial Intelligence 170 (2006) 739–778. doi:10.1016/j.artint.2006.02.002.

[15] T. Eiter, K. Wang, Semantic forgetting in answer set programming, Artificial Intelligence (2008).

[16] M. Knorr, J. J. Alferes, Preserving strong equivalence while forgetting, in: Proceedings of (JELIA-14), 2014. doi:10.1007/978-3-319-11558-0_29.

[17] M. Berthold, R. Gonçalves, M. Knorr, J. Leite, A syntactic operator for forgetting that satisfies strong persistence, Theory and Practice of Logic Programming (2019). doi:10.1017/S1471068419000346.

[18] R. Gonçalves, T. Janhunen, M. Knorr, J. Leite, On syntactic forgetting under uniform equivalence, in: Proceedings of (JELIA-21), 2021. doi:10.1007/978-3-030-75775-5_20.

[19] M. Berthold, On syntactic forgetting with strong persistence, in: Proceedings of (KR-22), 2022.

[20] F. Aguado, P. Cabalar, J. Fandinno, D. Pearce, G. Pérez, C. Vidal, Syntactic ASP forgetting with forks, Artificial Intelligence (2024). doi:10.1016/J.ARTINT.2023.104033.

[21] Z. G. Saribatur, T. Eiter, Omission-based abstraction for answer set programs, in: Proceedings of (KR-18), 2018.

[22] T. Eiter, Z. G. Saribatur, P. Schüller, Abstraction for zooming-in to unsolvability reasons of grid-cell problems, CoRR (2019). arXiv:1909.04998.

[23] Z. G. Saribatur, T. Eiter, A semantic perspective on omission abstraction in ASP, in: Proceedings of (KR-20), 2020. doi:10.24963/KR.2020/75.

[24] Z. G. Saribatur, S. Woltran, Foundations for projecting away the irrelevant in ASP programs, in: Proceedings of (KR-23), 2023. doi:10.24963/KR.2023/60.

[25] Z. G. Saribatur, S. Woltran, A unified view on forgetting and strong equivalence notions in answer set programming, in: Proceedings of (AAAI-24), 2024. doi:10.1609/AAAI.V38I9.28940.

[26] V. Lifschitz, L. R. Tang, H. Turner, Nested expressions in logic programs, Annals of Mathematics and Artificial Intelligence (1999). doi:10.1023/A:1018978005636.

[27] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic (2001).

[28] S. Woltran, Characterizations for relativized notions of equivalence in answer set programming, in: Proceedings of (JELIA-04), 2004. doi:10.1007/978-3-540-30227-8_16.

[29] K. Inoue, C. Sakama, Negation as failure in the head, Journal of Logic Programming (1998).

[30] K. Inoue, C. Sakama, Equivalence of logic programs under updates, in: Proceedings of (JELIA-04), 2004.

[31] M. Slota, J. Leite, Back and forth between rules and SE-models, in: Proceedings of (LPNMR-11), 2011.

[32] Y. Zhou, Y. Zhang, Bounded forgetting, in: Proceedings of (AAAI-11), 2011. doi:10.1609/AAAI.V25I1.7842.

[33] B. Bogaerts, T. Janhunen, S. Tasharrofi, Stable-unstable semantics: Beyond NP with normal logic programs, Theory and Practice of Logic Programming (2016). doi:10.1017/S1471068416000387.

[34] G. Amendola, F. Ricca, M. Truszczynski, Beyond NP: quantifying over answer sets, Theory and Practice of Logic Programming (2019). doi:10.1017/S1471068419000140.

[35] J. Heuer, C. Wernhard, Synthesizing strongly equivalent logic programs: Beth definability for answer set programs via craig interpolation in first-order logic, CoRR (2024). doi:10.48550/ARXIV.2402.07696. arXiv:2402.07696.