On the limits of forgetting in Answer Set Programming[☆]

Ricardo Gonçalves[a], Matthias Knorr[a], João Leite[a], Stefan Woltran[b]

[a] *NOVA LINCS, Departamento de Informática, FCT NOVA, 2829-516 Caparica, Portugal*
[b] *DBAI, TU Wien, Favoritenstrasse 9-11, A-1040 Vienna, Austria*

Abstract

Selectively forgetting information while preserving what matters the most is becoming an increasingly important issue in many areas, including in knowledge representation and reasoning. Depending on the application at hand, forgetting operators are defined to obey different sets of desirable properties. It turns out that, of the myriad of desirable properties discussed in the context of forgetting in Answer Set Programming, *strong persistence*, which imposes certain conditions on the correspondence between the answer sets of the program pre- and post-forgetting, and a certain independence from non-forgotten atoms, seems to best capture its essence, and be desirable in general. However, it has remained an open problem whether it is always possible to forget a set of atoms from a program while obeying strong persistence. In this paper, we investigate the limits of forgetting in Answer Set Programming. After showing that it is not always possible to forget a set of atoms from a program while obeying this property, we move forward and precisely characterize what can and cannot be forgotten from a program, by presenting a necessary and sufficient criterion. This characterisation allows us to draw some important conclusions regarding the existence of forgetting operators for specific classes of logic programs, to characterize the class of forgetting operators that achieve the correct result whenever forgetting is possible, and investigate the related question of determining what we can forget from some specific logic program. Subsequently, we address the issue of what to do when we *must* forget a set of atoms, but cannot without violating this property. To this end, we investigate three natural alternatives to forget when forgetting without violating strong persistence is not possible, which turn out to correspond to the different natural possible relaxations of the characterization of strong persistence. Additionally, before concluding, we address computational complexity issues – namely of checking whether the novel criterion holds and whether a certain program is a result according to the different classes of forgetting operators we introduce – and discuss the related literature.

Keywords: Forgetting, Answer Set Programming, Strong Equivalence,

[☆] This is a combined, revised and substantially extended version of the conference papers [1] and [2].

1. Introduction

Forgetting is often considered a problem in our everyday life, because of the negative effects it causes, such as missing an appointment, or failing to recall where we placed the car keys this time. Since the early experiments by Ebbinghaus [3], the causes of forgetting have been studied in Cognitive Psychology. According to [4], they vary from fading (or decay) over time; interference, where learning new information can affect previously learned information, and vice-versa; retrieval failure, either because we never stored the information in the first place or because links, so-called cues, that are required to retrieve the memory are missing; as well as repression, e.g., in the case of trauma, or amnesia.

Recent work in Neuroscience suggests that while some forms of forgetting, such as fading or loss of cues, are passive in nature, others must be the result of actively forgetting. Experiments show that our brain seems to be equipped with mechanisms that allow the removal of information no longer deemed necessary [5], which indicates that forgetting is indeed desirable. In fact, despite its common negative connotation, in the face of the huge amounts of information we acquire in our daily life, the loss of information actually turns out to be highly beneficial, since it allows us to distinguish important memories and abstract irrelevant details. As argued in [6], this fosters better decision making in future situations under changing conditions.

A similar motivation can be found in recent work on *intentional forgetting* in organizations/enterprises [7]. Whereas it is well-established that knowledge acquisition is paramount for the success of organizations, only more recently it has been observed that ever growing amounts of knowledge acquired over time can also become an obstacle for the continued success of the organization. Forgetting can then provide the means to ease the process of interpreting the available knowledge and turn the organization more competitive. Current proposed methods of forgetting are often inspired by forms of forgetting considered in cognitive psychology, such as forgetting the retrieval cues for organizational routines (rather than unlearning them) [7].

In Computer Science, the enormous increase of available data and information due to the digital transformation creates huge problems, because of the limitations in terms of space (of physical storage) and runtime (of algorithms). The latter is especially true in the area of Artificial Intelligence, where algorithms of exponential worst-case complexity are common. Furthermore, forgetting is becoming increasingly necessary to properly deal with legal and privacy issues, including, for example, the implementation of court orders to eliminate certain pieces of illegally acquired or maintained information, or even to enforce the new EU General Data Protection Regulation [8], which includes the *right to be forgotten* – the person's right to ask a corporation to eliminate private data.

This has resulted in a growing attention to forgetting with the aim to omit irrelevant information for faster and better decision processes and tools [9]. For example, in Machine Learning, methods are proposed to forget/fade out irrelevant features or use data selection to improve/speed up the training of neural networks. In the area of Knowledge Representation and Reasoning, two distinct kinds of forgetting have been considered in the literature [10]. The less common kind amounts to logically eliminate a certain formula from a knowledge base, and is in fact closely related to the operation of contraction in belief revision. The more common kind, and the one that corresponds to the focus of this paper, considers forgetting – or variable elimination – as an operation that suppresses part of the vocabulary of the knowledge base, which may result in an additional refinement of formulas over the remaining vocabulary. This is most useful when we wish to eliminate (temporary) variables introduced to represent auxiliary concepts, with the goal of restoring the declarative nature of some knowledge base, for data protection related issues, or just to simplify it. Within the area of Knowledge Representation and Reasoning, recent applications of forgetting to cognitive robotics [11, 12, 13], resolving conflicts [14, 15, 16, 17, 18], and ontology abstraction and comparison [19, 20, 21, 22], further witness its importance.

In this paper, we will focus on forgetting in Answer Set Programming – or Logic Programming under the Stable Models Semantics [23, 24] – a well established rule-based language for knowledge representation and reasoning, characterised by a simple and well-understood non-monotonic declarative semantics, with known relationships with other logic-based formalisms such as Default Logic, Autoepistemic Logic, SAT, etc., and strengthened by the existence of efficient implementations such as CLASP [25] and DLV [26]. Within this context, we show that it is not always possible to forget some set of atoms from an answer set program while preserving all existing relations between the atoms not to be forgotten, and we thoroughly investigate the *when*, *what*, and *how* related to adequately forgetting a set of atoms from an answer set program, and how to proceed when that is not possible, but we nevertheless need to forget.

1.1. Literature Review on Forgetting on Answer Set Programming

With its early roots in Boolean Algebra [27], forgetting has been extensively studied in the context of classical logic [28, 14, 29, 30, 31, 32, 33, 34] and, more recently, in the context of logic programming, notably of Answer Set Programming (ASP) (cf. also the broad survey [10]). The non-monotonic rule-based nature of ASP called for the development of specific methods and techniques – just as it happened with other belief change operations such as revision and update, cf. [35, 36, 37, 38, 39, 40, 41] – resulting in a significant number of different forgetting operators [15, 16, 42, 43, 44, 45, 46, 18], obeying different sets of properties deemed desirable, some adapted from the literature on *classical* forgetting [47, 43, 46], others specifically introduced for the case of ASP [16, 42, 43, 44, 45, 18], and often defined for different classes of answer set programs.

Examples of such properties include the so-called *strengthened consequence* **(sC)**, which requires that the answer sets of the result of forgetting be answer sets of the original program, ignoring the atoms to be forgotten; the so-called *weakened consequence* **(wC)**, which requires that the answer sets of the original program be preserved while forgetting, ignoring the atoms to be forgotten; the so-called *consequence persistence* **(CP)**, the conjoin of the previous two properties, which requires that the answer sets of the result of forgetting correspond exactly to those of the original program, ignoring the atoms to be forgotten; the so-called *strong invariance* **(SI)**, which requires that it be (strongly) equivalent to add a program without the atoms to be forgotten before or after forgetting; and the so-called *existence*, which requires that the result of forgetting belongs to the same class of programs admitted by the forgetting operator, so that the same reasoners can be used and the operator be iterated, among many others (cf. Sec. 2).

The operators proposed over the years follow rather distinct approaches. Some operators are purely syntactical, defining forgetting through a suitable transformation of the original program, such as the early proposals of Zhang and Foo [15] and, more recently, of Knorr and Alferes [45]. For example, Strong Forgetting [15] defined the result of forgetting by starting with computing a reduction corresponding to the well-known weak partial evaluation (WGPPE) [48], and then simply removing all rules containing the atom to be forgotten. Other operators characterise the result of forgetting by focusing on the answer sets of the original program and of the resulting program such as the proposal of Eiter and Wang [16], where the operators they define focus on considering only the minimal sets among the answer sets of the initial program ignoring the atoms to be forgotten to obtain the desired result. A larger set of operators proposed in the literature focus on the richer characterisation of programs based on their HT-models, such as the proposals of Wong [42], Wang et al. [43, 46], Wang et al. [44], and Delgrande and Wang [18]. For example, Semantic Strong Forgetting [42] defines the result of forgetting through a suitable construction which, as shown in [49], is characterised by the set of those HT-consequences of the original program that do not mention the atoms to be forgotten. A complete picture of the existing forgetting operators and properties they obey can be found in a recent survey [49].

1.2. Motivation and Contributions

From observing the landscape of existing operators and properties, one can conclude that there cannot be a one-size-fits-all forgetting operator for ASP, but rather a family of operators, each obeying a specific set of properties. Furthermore, it is clear that not all properties bear the same relevance. Whereas some properties can be very important, such as *existence*, since it guarantees that we can use the same automated reasoners after forgetting, other properties are less important, sometimes perhaps even questionable, as discussed in [49].

There is nevertheless one property – *strong persistence* **(SP)** [45] – which seems to best capture the essence of forgetting in the context of ASP. The property **(SP)** essentially requires that all existing relations between the atoms not

to be forgotten be preserved, captured by requiring that there be a correspondence between the answer sets of a program before and after forgetting a set of atoms, and that such correspondence be preserved in the presence of additional rules not containing the atoms to be forgotten. With a slight abuse of using notation that has not been introduced yet, an operator f is said to obey **(SP)** if, for every program P and every set of atoms to be forgotten V, it holds that $\mathcal{AS}(\mathsf{f}(P,V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, for all programs R not containing atoms in V, where $\mathsf{f}(P,V)$ denotes the result of forgetting V from P, $\mathcal{AS}(P)$ the answer sets of P, and $\mathcal{AS}(P)_{\|V}$ their restriction to atoms not in V. Notably, this characterization of the property is very much aligned with the conceptual idea of the more common view on Forgetting in Knowledge Representation and Reasoning and Answer Set Programming in particular.

Whereas it seems rather undisputed that **(SP)** is a desirable property, it is not clear to what extent we can define operators that satisfy it. In [45], the authors propose an operator that obeys such property, but which is only defined for a restricted class of programs and can only be applied to forget a single atom from a program in a very limited range of situations.

In the first part of this paper, in Secs. 3 and 4, we thoroughly investigate the limits of *forgetting* under **(SP)**. We begin, in Sec. 3, by checking whether it is always possible to forget some set of atoms from an answer set program while obeying **(SP)**. This is perhaps the most fundamental question that remained open in [49], not being clear whether an operator that achieves it does not exist, or simply no one had found it yet. It turns out that the answer is negative: sometimes it is simply not possible to forget some set of atoms from a program, while maintaining the relevant relations between other atoms, since the atoms to be forgotten play a pivotal role.

From this negative result, the following research questions become central, which we subsequently address in Sec. 4.

- *When can we not forget some set of atoms from an answer set program while obeying* **(SP)***?* We answer this by characterizing when a specific set of atoms cannot be forgotten from a specific program. We define a criterion (Ω) on a program and set of atoms which, when satisfied, implies that such atoms cannot be forgotten from the program.

- *When (and how) can we forget some set of atoms from an answer set program while obeying* **(SP)***?* We answer this by presenting a class of operators, dubbed $\mathsf{F_{SP}}$, that, provided Ω does not hold, satisfies strong persistence, among many other properties, and show that Ω is both sufficient and necessary to determine when some set of atoms can be forgotten from a program.

- *What can we forget from a specific answer set program while obeying* **(SP)***?* We answer this by providing a constructive definition of the sets of atoms that can be forgotten from a given program. While investigating the answer to this question, we uncover certain classes of programs from which we can always forget every single atom.

Having established and characterised the limits of forgetting under **(SP)**, in particular that it is not always possible to forget under **(SP)** (according to criterion Ω), the second part of this paper is devoted to investigate how to proceed when that is the case, but we nevertheless *must* forget. Sometimes we simply cannot avoid the need to forget. In applications of forgetting for conflict resolution [14, 15, 16, 17, 18], it seems rather clear that forgetting is indeed imperative, just as when we simply wish to restore the declarative nature of some knowledge base by eliminating (temporary) variables introduced to represent auxiliary concepts. The same argument applies to implementing court orders to eliminate certain pieces of illegally acquired or maintained information, even if it still is an open question whether the notion of forgetting required to ensure such right is the one under investigation in this line of research. In any case, tools that can help companies and users automate the operation of forgetting should be able to handle not only situations where we can achieve the required forgetting without violating **(SP)**, but also situations where such *ideal* forgetting is not possible.

Towards developing a theoretical ground on which such universally applicable tools can be based, we address the question of how to forget when Ω is true, along three different ways.

- We *first* take a closer look at the class F_{SP}, defined for the case when Ω is false, and investigate how it behaves in general. One crucial observation is that it overestimates answer sets, i.e., forgetting preserves all existing answer sets, but new ones may be added, which indicates a violation of property **(sC)**.

- Our *second* approach borrows from the notion of *relativized equivalence* [50], a generalization of strong equivalence that considers equivalence only w.r.t. a given subset of the language, and is characterized by the so-called V-HT-models, which lead us to consider a specific operator that simply returns all rules that are relativized equivalent to the original program w.r.t. the atoms not to be forgotten, which turns out to be a member of a class of operators whose result is characterized by the set of V-HT-models, omitting the atoms to be forgotten. Whereas this class never overestimates answer sets, i.e., it obeys **(sC)**, it may lose some of the original answer sets, which indicates a violation of property **(wC)**.

- The *third* approach tries to overcome a weakness of the second, i.e., its result diverges from F_{SP} even when it is possible to forget, and proposes a case-based definition that can be seen as a mixture of the previous two. Whereas it preserves all answer sets, i.e., it obeys both **(sC)** and **(wC)**, it no longer satisfies **(SI)** (strong invariance).

The three alternatives are fully investigated in Sec. 5. We characterize each of them by showing which subset of the properties previously considered in the literature they obey, and relate them by considering further additional properties, which also helps clarify their preferable usage. Perhaps one of the most

interesting features of this set of alternatives stems from a characterisation of **(SP)** according to which a forgetting operator obeys **(SP)** if and only if it obeys **(sC)**, **(wC)** and **(SI)**. Hence, each of the three alternatives exactly corresponds to the relaxation of one of these three properties that jointly characterize **(SP)**.

Before concluding the paper, we focus on the computational complexity of the problems considered in this paper, and discuss the related literature. First, in Sec. 6, we provide a detailed study of the computational complexity of verifying whether the criterion Ω holds and the computational complexity of determining the correct output for forgetting a set of atoms from a given program w.r.t. a class of forgetting operators, in the ideal case as well as in the case of the three alternatives when we cannot forget without changing some consequences. Subsequently, in Sec. 7, we consider other approaches for forgetting previously proposed in the literature, discussing how each relates to the problems addressed in this paper, and to what extent they are not suitable to our purposes.

Throughout the paper, other relevant intermediate results are presented, and the main concepts are illustrated with examples. The proofs of all results are in Appendix.

2. Forgetting in ASP

We start by recalling the necessary notions and notation on logic programs under the answer set semantics and on forgetting for answer set programming.

2.1. Answer Set Programming

We assume a *propositional signature* $\mathcal{A}$, i.e., a finite set of propositional atoms, also termed propositional variables synonymously. An *(extended) logic program* P over $\mathcal{A}$ is a finite set of *(extended) rules* of the form

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, ..., b_l, not\, c_1, ..., not\, c_m, not\, not\, d_1, ..., not\, not\, d_n\ , \qquad (1)$$

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 $\mathcal{A}$. Such rules r are also commonly written in a more succinct way as

$$A \leftarrow B, not\, C, not\, not\, D\ , \qquad (2)$$

where we have $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_l\}$, $C = \{c_1, \ldots, c_m\}$, $D = \{d_1, \ldots, d_n\}$, and we use both forms interchangeably. We write the set of atoms appearing in P as $\mathcal{A}(P)$ and the class of (extended) logic programs as $\mathcal{C}_e$.[1]

This class of logic programs, $\mathcal{C}_e$, includes a number of special kinds of rules r: if $n = 0$, then we call r *disjunctive*; if, in addition, $k \leq 1$, then r is *normal*; if on top of that $m = 0$, then we call r *Horn*; if moreover $k = l = 1$, then we call r *unary*, and *fact* if instead $l = 0$. We also admit *constraints*, which are

[1] Extended logic programs [51] are actually more expressive, but the form used here suffices. Also note that double negation is commonly required in the context of forgetting in ASP [49] and supported by answer set solvers such as clingo (`https://potassco.org/`).

(extended) rules where $k = 0$. The classes of *disjunctive*, *normal*, *Horn*, and *unary programs*, $\mathcal{C}_d$, $\mathcal{C}_n$, $\mathcal{C}_H$, and $\mathcal{C}_u$, are defined as a finite set of disjunctive, normal, Horn, and unary rules, respectively. We have $\mathcal{C}_u \subset \mathcal{C}_H \subset \mathcal{C}_n \subset \mathcal{C}_d \subset \mathcal{C}_e$.

Given a program P and a set I of atoms, the *reduct* P^I [52] is defined as

$$P^I = \{A \leftarrow B : r \text{ of the form (2) in } P \text{ such that } C \cap I = \emptyset \text{ and } D \subseteq I\}.$$

This reduct is used to define the answer sets of a program, i.e., its models, and we recall this notion based on HT-models as defined in the context of the logic of here-and-there [53], the monotonic logic underpinning ASP [54].

An *HT-interpretation* is a pair $\langle X, Y\rangle$ s.t. $X \subseteq Y \subseteq \mathcal{A}$. Given a program P, an HT-interpretation $\langle X, Y\rangle$ is an *HT-model of* P if $Y \models P$ and $X \models P^Y$, where $\models$ represents the standard consequence relation for classical logic and where programs are interpreted as conjunctions of classical implications

$$b_1 \wedge ... \wedge b_l \wedge \neg c_1 \wedge ... \wedge \neg c_m \wedge \neg\neg d_1 \wedge ... \wedge \neg\neg d_n \rightarrow a_1 \vee \ldots \vee a_k$$

corresponding to its rules r of the form (1) where $\neg$, $\wedge$, and $\vee$ denote classical negation, conjunction and disjunction, respectively. We occasionally admit for the sake of readability that the HT-models of a program P are restricted to $\mathcal{A}(P)$ even if $\mathcal{A}(P) \subset \mathcal{A}$. The set of *all HT-models of* P is written $\mathcal{HT}(P)$. A set of atoms Y is an *answer set* of P if $\langle Y, Y\rangle \in \mathcal{HT}(P)$, and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(P)$. The set of all answer sets of P is written $\mathcal{AS}(P)$. Note that, for $\mathcal{C}_d$ and its subclasses, all $I \in \mathcal{AS}(P)$ are pairwise incomparable. If P has an answer set, then P is *consistent*.

Example 1. *Consider the following program P:*

$$a \leftarrow not\, b \qquad b \leftarrow not\, c \qquad e \leftarrow d \qquad d \leftarrow a$$

We can show that $\langle b, bde\rangle$ is an HT-model of P because $\{b, d, e\} \models P$ and $\{b\} \models P^{\{b,d,e\}}$ where $P^{\{b,d,e\}}$ is as follows:[2]

$$b \leftarrow \qquad e \leftarrow d \qquad d \leftarrow a$$

It is then easy to see that $\{b, d, e\}$ is not an answer set of P. Similarly, $\{b, d\}$ is not an answer set of P because $\{b, d\} \not\models P$. In fact, we can verify that $\{b\}$ is the only answer set of P and that P is therefore consistent.

Relations between programs can be established based on notions of equivalence and consequence. We say that two programs P_1, P_2 are *equivalent* if $\mathcal{AS}(P_1) = \mathcal{AS}(P_2)$ and *strongly equivalent*, written $P_1 \equiv P_2$, if $\mathcal{AS}(P_1 \cup R) = \mathcal{AS}(P_2 \cup R)$ for every $R \in \mathcal{C}_e$. It has been shown that, for this test, it suffices to consider programs $R \in \mathcal{C}_u$ [50], and it is well-known that $P_1 \equiv P_2$ exactly when $\mathcal{HT}(P_1) = \mathcal{HT}(P_2)$ [55]. We also say that P' is an *HT-consequence* of P, written $P \models_{\mathsf{HT}} P'$, whenever $\mathcal{HT}(P) \subseteq \mathcal{HT}(P')$.

[2] We follow a common convention and abbreviate sets in HT-interpretations such as $\{a, b\}$ with the sequence of its elements, ab.

Example 2. *Consider* $P_1 = \{b \leftarrow\}$. *Then* P_1 *and* P *from Example 1 are equivalent, but not strongly equivalent, e.g., because adding* $R = \{c \leftarrow\}$ *to both yields different answer sets, namely* $\{b, c\}$ *and* $\{a, c, d, e\}$, *respectively. Also, neither is an HT-consequence of the other, since* $\langle \emptyset, bc\rangle \in \mathcal{HT}(P)$, *but not in* $\mathcal{HT}(P_1)$, *and* $\langle b, bd\rangle \in \mathcal{HT}(P_1)$, *but not in* $\mathcal{HT}(P)$.

Occasionally, we want to omit certain elements of the signature from sets of interpretations and to identify certain sets whenever they coincide on all elements of a restricted part of the signature. Given a set of atoms V, the V-*exclusion* of a set of answer sets (resp. a set of HT-interpretations) $\mathcal{M}$, written $\mathcal{M}_{\|V}$, is $\{X\backslash V \mid X \in \mathcal{M}\}$ (resp. $\{\langle X\backslash V, Y\backslash V\rangle \mid \langle X, Y\rangle \in \mathcal{M}\}$). Also, given two sets of atoms $X, X' \subseteq \mathcal{A}$, we write $X \sim_V X'$ whenever $X\backslash V = X'\backslash V$.

The different classes of logic programs introduced previously can be precisely characterized in terms of HT-models. These characterizations appear in the literature, but are spread out over different papers. To ease the reading, we collect and recall them here in a uniform way, starting with the most general class, the class of extended programs.

Lemma 1. *([56]) Let* $\mathcal{M}$ *be a set of HT-interpretations. Then, there exists an extended program* $P \in \mathcal{C}_e$ *such that* $\mathcal{M} = \mathcal{HT}(P)$ *iff* $\mathcal{M}$ *satisfies the following condition:*

(E_1) $\langle X, Y\rangle \in \mathcal{M} \Rightarrow \langle Y, Y\rangle \in \mathcal{M}$.

The characterization for the (restricted) class of disjunctive programs requires an additional condition.

Lemma 2. *([57]) Let* $\mathcal{M}$ *be a set of HT-interpretations. Then, there exists a disjunctive program* $P \in \mathcal{C}_d$ *such that* $\mathcal{M} = \mathcal{HT}(P)$ *iff* $\mathcal{M}$ *satisfies the following conditions:*

(D_1) $\langle X, Y\rangle \in \mathcal{M} \Rightarrow \langle Y, Y\rangle \in \mathcal{M}$;

(D_2) $\langle X, Y\rangle \in \mathcal{M}$, *and* $\langle Y', Y'\rangle \in \mathcal{M}$ *with* $Y \subseteq Y' \Rightarrow \langle X, Y'\rangle \in \mathcal{M}$.

Normal programs are even further restricted and this is reflected in the characterization.

Lemma 3. *([58]) Let* $\mathcal{M}$ *be a set of HT-interpretations. Then, there exists a normal program* $P \in \mathcal{C}_n$ *such that* $\mathcal{M} = \mathcal{HT}(P)$ *iff* $\mathcal{M}$ *satisfies the following conditions:*

(N_1) $\langle X, Y\rangle \in \mathcal{M} \Rightarrow \langle Y, Y\rangle \in \mathcal{M}$;

(N_2) $\langle X, Y\rangle \in \mathcal{M}$, *and* $\langle Y', Y'\rangle \in \mathcal{M}$ *with* $Y \subseteq Y' \Rightarrow \langle X, Y'\rangle \in \mathcal{M}$;

(N_3) $\langle X, Y\rangle \in \mathcal{M}$ *and* $\langle X', Y\rangle \in \mathcal{M} \Rightarrow \langle X \cap X', Y\rangle \in \mathcal{M}$.

While in the previous cases, each further restriction on the admitted programs introduces an additional condition, the case of Horn programs further restricted variants of the conditions used so far. This result is presented in [46], but it builds on previous work [59].

Lemma 4. *([46]) Let $\mathcal{M}$ be a set of HT-interpretations. Then, there exists a Horn program $P \in \mathcal{C}_H$ such that $\mathcal{M} = \mathcal{HT}(P)$ iff $\mathcal{M}$ satisfies the following conditions:*

(H_1) $\langle X, Y\rangle \in \mathcal{M}$, *and* $\langle Y', Y'\rangle \in \mathcal{M}$ *with* $Y \subseteq Y' \Rightarrow \langle X, Y'\rangle \in \mathcal{M}$;

(H_2) $\langle X, Y\rangle \in \mathcal{M}$ *iff* $X \subseteq Y, \langle X, X\rangle \in \mathcal{M}$ *and* $\langle Y, Y\rangle \in \mathcal{M}$;

(H_3) $\langle X, Y\rangle \in \mathcal{M}$ *and* $\langle X', Y'\rangle \in \mathcal{M} \Rightarrow \langle X \cap X', Y \cap Y'\rangle \in \mathcal{M}$.

Note that (H_2) implies (E_1), (D_1) and (N_1), which is why a property corresponding to the latter is missing in the case of Horn programs. Interestingly, it can be shown that (H_2) implies (H_1).

Lemma 5. *Let $\mathcal{M}$ be a set of HT-interpretations. Then, (H_2) implies (H_1).*

This makes (H_1) redundant in the characterization of the class of Horn programs, as it is implied by (H_2), but we leave it here for the sake of clarity in some of the forthcoming arguments.

Besides the notion of HT-model, we also recall the notion of A-SE-models [50], a generalization of HT-models for ASP that allow to consider only part of the signature, but here adapted to V-HT-models that focus on $V \subseteq \mathcal{A}$, the atoms to be omitted, instead of on $A = \mathcal{A}\backslash V$, the atoms that remain. Given a set of atoms V, an HT-interpretation $\langle X, Y\rangle$ is called a *V-HT-interpretation* if either $X = Y$ or $X \subset Y\backslash V$. A V-HT-interpretation $\langle X, Y\rangle$ is a *(relativized) V-HT-model of P* if: (i) $Y \models P$; (ii) for all $Y' \subset Y$ with $Y \sim_V Y'$, $Y' \not\models P^Y$; and (iii) if $X \subset Y$, then there exists $X' \subseteq Y$ such that $X = X'\backslash V$ and $X' \models P^Y$. The set of *all V-HT-models of P* is written $\mathcal{HT}_V(P)$. Programs P_1, P_2 are *relativized equivalent* w.r.t. $V \subseteq \mathcal{A}$, written $P_1 \equiv_V P_2$, if $\mathcal{AS}(P_1 \cup R) = \mathcal{AS}(P_2 \cup R)$ for every $R \in \mathcal{C}_e$ s.t. $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$. Here, again, such $R \in \mathcal{C}_u$ suffice for verifying the condition, and we have that $P_1 \equiv_V P_2$ exactly when $\mathcal{HT}_V(P_1) = \mathcal{HT}_V(P_2)$ [50].

Example 3. *Consider again program P from Ex. 1 and $V = \{b\}$. Then, $\langle b, b\rangle$ and $\langle a, abde\rangle$ are V-HT-interpretations while $\langle e, be\rangle$ is not. We can see that $\langle b, b\rangle$ is a V-HT-model of P because (i) $\{b\} \models P$, (ii) only $\emptyset \subset \{b\}$ and $\emptyset \not\models P^{\{b\}}$ and since (iii) trivially holds. On the other hand, $\langle a, abde\rangle$ is not a V-HT-model of P, because (iii) does not hold, since neither $\{a\} \models P^{\{a,b,d,e\}}$ nor $\{a, b\} \models P^{\{a,b,d,e\}}$. Yet, $\langle e, abde\rangle$ is a V-HT-model of P.*

Note that the definitions of V-HT-interpretation and V-HT-model precisely coincide with those of HT-interpretation and HT-model for the case where nothing is omitted/forgotten, i.e., $V = \emptyset$. For the detailed justification of the more involved definition of V-HT models we refer to [50].

2.2. Forgetting

The principal idea of forgetting in ASP is to remove or hide certain atoms from a given program, while preserving its semantics for the remaining atoms. As the result, often a representative up to some notion of equivalence between programs is considered. In this sense, many notions of forgetting for logic programs are defined semantically, i.e., they introduce a class of operators that satisfy a certain semantic characterization. Each single operator in such a class is then a concrete function that, given a program P and a non-empty set of atoms V to be forgotten, returns a unique program, the result of forgetting about V from P. Formally, given a class of logic programs $\mathcal{C}$ over $\mathcal{A}$, a *forgetting operator (over $\mathcal{C}$)* is a partial function[3] $\mathsf{f} : \mathcal{C} \times 2^{\mathcal{A}} \to \mathcal{C}$ such that $\mathsf{f}(P, V)$ is a program over $\mathcal{A}(P) \backslash V$, for each $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$. We call $\mathsf{f}(P, V)$ the *result of forgetting about V from P* and denote the domain of f by $\mathcal{C}(\mathsf{f})$. Unless stated otherwise, in what follows, we focus on $\mathcal{C} = \mathcal{C}_e$, and we leave $\mathcal{C}$ implicit. Furthermore, f is called *closed* for $\mathcal{C}' \subseteq \mathcal{C}(\mathsf{f})$ if, for every $P \in \mathcal{C}'$ and $V \subseteq \mathcal{A}$, we have $\mathsf{f}(P, V) \in \mathcal{C}'$. A *class* F *of forgetting operators (over $\mathcal{C}$)* is a set of forgetting operators f such that $\mathcal{C}(\mathsf{f}) \subseteq \mathcal{C}$.[4] Such classes are usually described by a common definition/condition that each operator in the class has to satisfy (see [49] for an overview on the many different kinds and forms of defining such classes).

Most previous work on forgetting in ASP accompanies the definition of classes of operators with the introduction of a variety of desirable properties. In the following, we recall these properties, leaving the details, e.g., on which class of forgetting operators satisfies which properties, to related work in the literature [49].[5] Unless stated otherwise, F is a class of forgetting operators.

($\mathbf{E}_{\mathcal{C}}$) F satisfies *Existence for $\mathcal{C}$*, i.e., F is *closed for a class of programs* $\mathcal{C}$ if there exists $\mathsf{f} \in \mathsf{F}$ s.t. f is closed for $\mathcal{C}$.

(wE) F satisfies *weak Equivalence* if, for each $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(\mathsf{f}(P, V)) = \mathcal{AS}(\mathsf{f}(P', V))$ whenever $\mathcal{AS}(P) = \mathcal{AS}(P')$.

(SE) F satisfies *Strong Equivalence* if, for each $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$: if $P \equiv P'$, then $\mathsf{f}(P, V) \equiv \mathsf{f}(P', V)$.

(PP) F satisfies *Positive Persistence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$: if $P \models_{\mathsf{HT}} P'$, with $P' \in \mathcal{C}(\mathsf{f})$ and $\mathcal{A}(P') \subseteq \mathcal{A} \backslash V$, then $\mathsf{f}(P, V) \models_{\mathsf{HT}} P'$.

(W) F satisfies *Weakening* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $P \models_{\mathsf{HT}} \mathsf{f}(P, V)$.

[3] The admission of partial functions is due to the fact that some operators existing in the literature are not always defined.

[4] There are classes of operators in the literature defined for class C which include concrete operators only defined for a subclass C'.

[5] We omit the property dubbed *Negative Persistence* **(NP)** from the list, as it has been shown to coincide with **(W)** [49].

(SI) F satisfies *Strong (addition) Invariance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathsf{f}(P,V) \cup R \equiv \mathsf{f}(P \cup R, V)$ for all programs $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$.

(sC) F satisfies *strengthened Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(\mathsf{f}(P,V)) \subseteq \mathcal{AS}(P)_{\|V}$.

(wC) F satisfies *weakened Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(P)_{\|V} \subseteq \mathcal{AS}(\mathsf{f}(P,V))$.

(CP) F satisfies *Consequence Persistence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(\mathsf{f}(P,V)) = \mathcal{AS}(P)_{\|V}$.

(SP) F satisfies *Strong Persistence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(\mathsf{f}(P,V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, for all programs $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$.

Throughout the paper, whenever we write that a single operator f obeys some property, we mean that the singleton class composed of that operator, $\{\mathsf{f}\}$, obeys such property.

Example 4. *Consider program P from Ex. 1.*

$$a \leftarrow not\, b \qquad b \leftarrow not\, c \qquad e \leftarrow d \qquad d \leftarrow a$$

First, if we want to forget about some atom, then we expect all rules that do not mention this atom to persist, while rules that mention it to no longer occur. For example, when forgetting about d from P, the first two rules should be contained in the result of forgetting, while the latter two should not. At the same time, implicit dependencies, such as e depending on a via d, should be preserved. Hence, we expect the following result:

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

In fact, many existing notions of forgetting in the literature (cf. [49]) provide precisely this result.

Now, consider forgetting about b from P. Note that P contains an implicit dependency between a and c, namely, whenever c becomes true, then so does a, i.e., if we add, e.g., $c \leftarrow$ to the program, then a is necessarily true. Different notions of (classes of) forgetting operators f existing in the literature (see [49]) would return the result of forgetting $\mathsf{f}(P,\{b\}) = \emptyset$, but if we want to preserve the indirect dependency of a on c, then $\mathsf{f}(P,\{b\})$ must contain the rule $a \leftarrow not\, not\, c$ to ensure that adding $c \leftarrow$ results in a single answer set $\{a,c,d,e\}$ for both P and $\mathsf{f}(P,\{b\})$. In fact, a valid result for $\mathsf{f}(P,\{b\})$ such that f satisfies **(SP)**, *arguably the central property for forgetting in ASP (see Sec. 3), would be:*

$$a \leftarrow not\, not\, c \qquad e \leftarrow d \qquad d \leftarrow a$$

Finally, if the atom to be forgotten does not appear at the same time in some rule body and some rule head, usually no dependencies need to be preserved. Consider forgetting about c from P, then, since c only appears (negated) in the body of the rule with head b, b becomes unconditionally true, and the expected result $\mathsf{f}(P, \{c\})$ *would be:*

$$a \leftarrow not\, b \qquad b \leftarrow \qquad e \leftarrow d \qquad d \leftarrow a$$

3. The Limits of Forgetting

Among the desirable properties of classes of forgetting operators recalled in the previous section, *strong persistence* **(SP)** [45] is of particular interest, as it ensures that forgetting preserves all existing relations between all atoms occurring in the program, but the forgotten. In this sense, a class of operators satisfying **(SP)** removes the desired atoms, but has no negative semantic effects on the remainder. The importance of **(SP)** is also witnessed by the fact that a class of operators that satisfies **(SP)** satisfies all the other previously mentioned properties with the exception of **(W)**, which has been shown to be incompatible with the preservation of answer sets while forgetting [49]. Hence, ideally, when forgetting in ASP, we want to ensure that **(SP)** is satisfied. In this section, we investigate in detail whether there are any limits on preserving all the existing relations between all the remaining atoms occurring in the program, while forgetting.

3.1. Can We Always Forget?

If we had an operator that satisfies **(SP)**, then answering the question of whether we can always forget while preserving all the existing relations between non-forgotten atoms would be straightforwardly positive. However, determining a forgetting operator that satisfies **(SP)** is a difficult problem, since, for the verification whether a certain program P' should be the result of forgetting about a set of atoms V from program P, none of the well-established equivalence relations can be used, i.e., neither equivalence nor strong equivalence hold in general between P and P', not even relativized equivalence [50], even though it is close in spirit to the ideas of **(SP)**. Hence, maybe not surprisingly, there is no known general class of operators that satisfies **(SP)** and that is closed (for the considered class of logic programs).

The two known positive results concerning the satisfiability of **(SP)** are the existence of several known classes of operators that satisfy **(SP)** *when restricted to Horn programs* [49], and the existence of one specific operator that, in a very restricted range of situations based on a non-trivial syntactical criterion, permits forgetting about V from P while satisfying the definition of **(SP)** [45]. However, the former result is probably of little relevance given the crucial role played by (default) negation in ASP, while the criterion required in the latter result is certainly too strong, excluding large classes of cases where forgetting about V from P is possible.

All this begs the question of whether there even exists a forgetting operator, or a class of these, defined over a class of programs $\mathcal{C}$ beyond the class of Horn programs, that satisfies **(SP)**. The following theorem provides a negative answer to this question.

Theorem 1. *There is no forgetting operator over $\mathcal{C} \supseteq \mathcal{C}_n$ that satisfies* **(SP)**.

Proof. Let $\mathcal{C}$ be a class of programs with $\mathcal{C} \supseteq \mathcal{C}_n$ and suppose there exists f over $\mathcal{C}$ that satisfies **(SP)**. Then, for each $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(\mathsf{f}(P, V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, for all programs $R \in \mathcal{C}$ with $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$. Consider $P \in \mathcal{C}_n$:

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

We construct $\mathcal{HT}(\mathsf{f}(P, \{p, q\}))$, the set of HT-models of the result of forgetting about $V = \{p, q\}$ from P.

We know that $\langle ab, ab\rangle$ must be part of $\mathcal{HT}(\mathsf{f}(P, \{p, q\}))$, otherwise it would not be possible to obtain the answer set $\{a, b\}$ for the result when adding $R = \{a \leftarrow; b \leftarrow\}$ to $\mathsf{f}(P, \{p, q\})$.

At the same time, since $\{a, b\}$ (modulo V) is not an answer set of the original program, $\langle X, ab\rangle \in \mathcal{HT}(\mathsf{f}(P, \{p, q\}))$ for at least one $X \subset \{a, b\}$ to prevent $\{a, b\}$ from being an answer set of $\mathsf{f}(P, \{p, q\})$. Consider the three alternatives:

- $\langle \emptyset, ab\rangle \notin \mathcal{HT}(\mathsf{f}(P, \{p, q\}))$, as adding $R = \{a \leftarrow b; b \leftarrow a\}$, whose HT-models are $\{\langle \emptyset, \emptyset\rangle, \langle \emptyset, ab\rangle, \langle ab, ab\rangle\}$, yields one answer set $\{a, b\}$ for $P \cup R$ (modulo V) which the forgetting result has to preserve;
- $\langle a, ab\rangle \notin \mathcal{HT}(\mathsf{f}(P, \{p, q\}))$, since adding $R = \{a \leftarrow\}$, whose HT-models include $\langle a, ab\rangle$, yields one answer set $\{a, b\}$ for $P \cup R$ (modulo V) which the forgetting result has to preserve;
- $\langle b, ab\rangle \notin \mathcal{HT}(\mathsf{f}(P, \{p, q\}))$ symmetrically for $R = \{b \leftarrow\}$.

We derive a contradiction. ■

Consequently, it is not always possible to forget a set of atoms from a given logic program while satisfying the property **(SP)**.

3.2. When Can't We Forget?

Whereas Thm. 1 shows that in general it is not always possible to forget while satisfying **(SP)**, its proof provides some hints on why this is the case. Some atoms play an important role in the program, being pivotal in establishing the relations between the remaining atoms. Therefore, it is simply not possible to forget them and expect that the relations between other atoms be preserved. That is precisely what happens with the pair of atoms p and q in the program

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

presented in the proof of Thm. 1. It is simply not possible to forget them both and expect all the semantic relations between a and b to be kept. No program

over atoms $\{a, b\}$ would have the same answer sets as those of the original program (modulo p and q), when both are extended with an arbitrary set of rules over $\{a, b\}$.

This observation immediately leads to one of the central questions here: under what circumstances is it not possible to forget about a given set of atoms V from program P while satisfying **(SP)**? In particular, given a concrete program, which sets of atoms play such a pivotal role that they cannot be jointly forgotten without affecting the semantic relations between the remaining atoms in the original program?

To deal with these questions that no longer require the satisfaction of certain properties in general for all programs P and all sets of forgotten atoms V, but rather that **(SP)** holds for a concrete program P and set of atoms V, we introduce the notion of a *forgetting instance*.

Definition 1 (Forgetting Instance). *Let $\mathcal{C}$ be a class of programs over $\mathcal{A}$. A* (forgetting) instance (over $\mathcal{C}$) *is a pair $\langle P, V\rangle$ s.t. $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$.*

Thus, given P from the proof of Thm. 1, $\langle P, \{p, q\}\rangle$ is a forgetting instance over $\mathcal{C}_e$ (or alternatively over $\mathcal{C}_n$).

This allows us to introduce a restriction of property **(SP)** to operators of forgetting and such forgetting instances.

Definition 2 (Strong Persistence for Forgetting Instance). *A forgetting operator f over $\mathcal{C}$ satisfies $\textbf{(SP)}_{\langle P,V\rangle}$, for some forgetting instance $\langle P, V\rangle$ over $\mathcal{C}$, if $\mathcal{AS}(\mathsf{f}(P, V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, for all programs $R \in \mathcal{C}$ with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$.*

Before we proceed, it is worth noting that the previous definition not only requires that operator f be defined for the class of programs to which program P belongs, but it also restricts that programs R belong to such class. Whereas these conditions affect operators defined only for $\mathcal{C}_n$ or $\mathcal{C}_d$, it turns out that operators restricted to such domain are not desirable in the context of **(SP)**. The reason is that, in general, even if there is a forgetting operator f that satisfies $\textbf{(SP)}_{\langle P,V\rangle}$ for a normal program P and atoms V, there may be none over normal or disjunctive programs, as shown by the following example.

Example 5. *Consider forgetting about q from P:*

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

The only correct result that satisfies the condition of **(SP)** *is strongly equivalent to $P' = \{p \leftarrow not\, not\, p\}$ with $\mathcal{AS}(P') = \{\emptyset, \{p\}\}$. Thus, the answer sets are comparable, i.e., $\emptyset \subset \{p\}$, hence P' cannot be strongly equivalent to any normal or disjunctive program.*

Therefore, even if we can forget about some V from a normal program, the result will in general be an extended program, hence, requiring an operator over this general class. As a consequence, in what follows, unless otherwise stated, we will focus on forgetting operators over the entire general class $\mathcal{C}_e$.

We now proceed with the introduction of a criterion (Ω) which will play a fundamental role in characterizing the instances for which we cannot expect forgetting operators to satisfy $\mathbf{(SP)}_{\langle P,V\rangle}$. This is a rather technically involved part of the paper, so we beg the reader to bear with us.

First, note that property **(SP)** in general requires strong equivalence between the program P and its result of forgetting modulo the set of forgotten atoms V, i.e., it preserves the answer sets modulo V no matter which program R over the remaining language is added to both. Since the HT-models of the considered program P are paramount for determining the answer sets of P together with changing programs R, it is maybe not surprising that these HT-models are crucial for the criterion. However, certain HT-models can never give rise to an answer set over the remaining language.

Example 6. *Recall $P \in \mathcal{C}_n$ used in the proof of Thm. 1:*

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

Here, $\mathcal{HT}(P)$ contains 15 elements:

$$\begin{array}{cccc} \langle ap, ap\rangle & \langle bq, abq\rangle & \langle b, abpq\rangle & \langle abp, abpq\rangle \\ \langle bq, bq\rangle & \langle abq, abq\rangle & \langle ab, abpq\rangle & \langle abq, abpq\rangle \\ \langle ap, abp\rangle & \langle \emptyset, abpq\rangle & \langle ap, abpq\rangle & \langle abpq, abpq\rangle \\ \langle abp, abp\rangle & \langle a, abpq\rangle & \langle bq, abpq\rangle & \end{array}$$

Note that P has HT-models $\langle Y, Y\rangle$ and $\langle Y', Y\rangle$ with $Y'\backslash V = Y\backslash V$ and $Y' \subset Y$, namely $\langle abpq, abpq\rangle$ and any of $\langle ab, abpq\rangle$, $\langle abp, abpq\rangle$ and $\langle abq, abpq\rangle$ where $V = \{p, q\}$. Then, $Y = \{a, b, p, q\}$ can never become an answer set of a program $P \cup R$, no matter which R over the remaining language is added to P. This happens since any program R over $\{a, b\}$ cannot distinguish these HT-models, i.e., either all are HT-models of R or none is.[6]

We formalize the idea that such entire sets of HT-models are not relevant for the possible answer sets of $P \cup R$ by determining, for each possible Y over the remaining atoms, those subsets of the forgotten atoms that are indeed relevant.

Definition 3 (Relevant Forgotten Atoms). *Let P be a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, and $Y \subseteq \mathcal{A}\backslash V$. We define the* set of sets of relevant forgotten atoms for Y w.r.t. $\langle P, V\rangle$, $Rel^Y_{\langle P,V\rangle}$, *as follows:*

$$\begin{aligned} Rel^Y_{\langle P,V\rangle} = \{A \subseteq V \mid\ & \langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P) \text{ and} \\ & \nexists A' \subset A \text{ such that } \langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)\}. \end{aligned}$$

Example 7. *Recall P from Ex. 6 with the atoms to be forgotten $V = \{p, q\}$. We have $Rel^{\emptyset}_{\langle P,V\rangle} = \emptyset$, $Rel^{\{a\}}_{\langle P,V\rangle} = \{\{p\}\}$, $Rel^{\{b\}}_{\langle P,V\rangle} = \{\{q\}\}$, and $Rel^{\{a,b\}}_{\langle P,V\rangle} = \{\{p\}, \{q\}\}$. Thus, as expected by Ex. 6, $\{p, q\} \notin Rel^{\{a,b\}}_{\langle P,V\rangle}$.*

[6] A similar argument exists for the exclusion of certain HT-models from relativized HT-models. [50]

Note that, for each set of atoms A in any $Rel^Y_{\langle P,V\rangle}$, those HT-models of the form $\langle X, Y \cup A\rangle$ guarantee an answer set $Y \cup A$ of $P \cup R$, for at least one R over the remaining atoms.

In accordance with **(SP)**, we are interested in a forgetting result whose answer sets coincide with those of P, no matter which R we add. This raises the question of how to deal with cases where $Rel^Y_{\langle P,V\rangle}$ contains more than one element, such as $Rel^{\{a,b\}}_{\langle P,V\rangle}$ in Ex. 7. In particular, there may be at least two sets of HT-models of P, of the form $\langle X, Y \cup A\rangle$ and $\langle X, Y \cup A'\rangle$, with $A, A' \in Rel^Y_{\langle P,V\rangle}$, that result in answer sets for different R, but since Y coincides, they basically collapse into one case over the remaining language. As it turns out, the problematic cases are exactly those where such sets of HT-models are in conflict over for which programs R an answer set is obtained.

Before we formalize this in our criterion, we introduce notation that joins the first components of the HT-models in each such set of HT-models. Namely, for P a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, and $Y \subseteq \mathcal{A}\backslash V$, we denote

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

A conflict occurs, and thus forgetting while preserving **(SP)** is not possible, when there is no least element among these $R^{Y,A}_{\langle P,V\rangle}$ (for relevant A) for one Y.

We are now ready to formulate the main concept of this Section, criterion Ω, which will be used to characterize the limits of forgetting under **(SP)**.

Definition 4 (Criterion Ω). *Let P be a program over $\mathcal{A}$ and $V \subseteq \mathcal{A}$. An instance $\langle P, V\rangle$* satisfies criterion Ω *if there exists $Y \subseteq \mathcal{A}\backslash V$ such that the set of sets*

$$\mathcal{R}^Y_{\langle P,V\rangle} = \{R^{Y,A}_{\langle P,V\rangle} \mid A \in Rel^Y_{\langle P,V\rangle}\}$$

is non-empty and has no least element.

The following example illustrates how criterion Ω can be checked.

Example 8. *Recall P from Example 6 with $V = \{p, q\}$. To prove that instance $\langle P, \{p, q\}\rangle$ satisfies Ω, we need to find $Y \subseteq \mathcal{A}\backslash V = \{a, b, p, q\}\backslash\{p, q\} = \{a, b\}$ such that $\mathcal{R}^Y_{\langle P,V\rangle}$ is non-empty and has no least element. For that, we only need to focus on those sets $Y' \subseteq \{a, b\}$ for which $Rel^{Y'}_{\langle P,V\rangle}$ is not empty. For $Y' = \{b\}$, $\mathcal{R}^{Y'}_{\langle P,V\rangle}$ has only one element and therefore necessarily a least one. The same holds for $Y' = \{a\}$.*

We are left to inspect $Y' = \{a, b\}$. We already know from Ex. 7 that $\mathcal{R}^{\{a,b\}}_{\langle P,V\rangle}$ has only two elements, $R^{\{a,b\},\{p\}}_{\langle P,V\rangle} = \{\{a, b\}, \{a\}\}$ and $R^{\{a,b\},\{q\}}_{\langle P,V\rangle} = \{\{a, b\}, \{b\}\}$. Since these two sets are incomparable, $\mathcal{R}^{\{a,b\}}_{\langle P,V\rangle}$ has no least element. Therefore, taking $Y = \{a, b\}$, we conclude that $\langle P, \{p, q\}\rangle$ satisfies Ω.

Our objective now is to show that this criterion can be used for determining whether we can forget while satisfying **(SP)**. In the particular case of Horn programs, we can show that the criterion is never satisfied.

Proposition 1. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then $\langle P, V\rangle$ does not satisfy Ω.*

This result aligns with our expectations given that several known classes of forgetting operators do satisfy **(SP)** if restricted to Horn programs [49].

In general, Ω is of course satisfiable as we have seen, e.g., in Ex. 8. Hence, we have to inspect the relation between Ω and $\mathbf{(SP)}_{\langle P,V\rangle}$ more closely.

Since criterion Ω heavily relies on HT-models, it helps to observe that each total model of the result of a forgetting operator that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ is related with at least one HT-model of the original program.

Lemma 6. *Let f be a forgetting operator over $\mathcal{C}$ that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ for an instance $\langle P, V\rangle$ over $\mathcal{C}$. If $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, then $Rel^Y_{\langle P,V\rangle} \neq \emptyset$.*

This relation between the HT-models of the result of forgetting and those of the original program is in fact stronger since, for all instances $\langle P, V\rangle$, every forgetting operator that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ produces a result whose HT-models are a subset of the HT-models of the original program (modulo the forgotten atoms).

Proposition 2. *If a forgetting operator f over $\mathcal{C}$ satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ for an instance $\langle P, V\rangle$ over $\mathcal{C}$, then*

$$\mathcal{HT}(\mathsf{f}(P, V)) \subseteq \mathcal{HT}(P)_{\|V}.$$

We are now ready to state the main result of this Section, namely that Ω is a sufficient condition to determine that some set of atoms V cannot be forgotten from a program P while satisfying *strong persistence*.

Theorem 2. *If $\langle P, V\rangle$ satisfies Ω, then no forgetting operator f over $\mathcal{C} \supseteq \mathcal{C}_d$ satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$.*

Example 9. *Recall $P \in \mathcal{C}_n$ from Ex. 6. Since $\langle P, \{p, q\}\rangle$ satisfies criterion Ω (cf. Ex. 8), no forgetting operator f satisfies $\mathbf{(SP)}_{\langle P,\{p,q\}\rangle}$.*

Thm. 2 is explicitly stated for operators defined for the classes of programs $\mathcal{C}_e$ and $\mathcal{C}_d$. This is no mere coincidence. While the result would vacuously hold for operators defined for $\mathcal{C}_H$ by Prop. 1, it turns out that Thm. 2 does not hold for operators that are restricted to normal programs.

Example 10. *Consider forgetting about $V = \{p, q\}$ from program P.*

$$\begin{array}{lll} p \leftarrow b, not\ q & b \leftarrow p, not\ q & a \leftarrow b, not\ q \\ q \leftarrow b, not\ p & b \leftarrow q, not\ p & a \leftarrow b, not\ p \end{array}$$

This program P has a large number of HT-models, but many of them can never give rise to an answer set. More precisely, we can verify that $Rel^{\emptyset}_{\langle P,V\rangle} = \{\emptyset\}$, $Rel^{\{a\}}_{\langle P,V\rangle} = \emptyset$, $Rel^{\{b\}}_{\langle P,V\rangle} = \emptyset$, and $Rel^{\{a,b\}}_{\langle P,V\rangle} = \{\{p\}, \{q\}\}$ with $\mathcal{R}^{\emptyset}_{\langle P,V\rangle} = \{\{\emptyset\}\}$ and

$\mathcal{R}^{\{a,b\}}_{\langle P,V\rangle} = \{\{\emptyset,\{a\},\{a,b\}\},\{\emptyset,\{b\},\{a,b\}\}\}$. *This corresponds to the following HT-models:*

$$\langle\emptyset,\emptyset\rangle \qquad \langle\emptyset,abp\rangle \qquad \langle a,abp\rangle \qquad \langle abp,abp\rangle$$
$$\langle\emptyset,abq\rangle \qquad \langle b,abq\rangle \qquad \langle abq,abq\rangle$$

Since $\mathcal{R}^{\{a,b\}}_{\langle P,V\rangle}$ *has no least element,* Ω *is satisfied. Also, applying the same argument as in the proof of Thm. 1 to construct the HT-models of the result of forgetting, we obtain that the HT-models should contain precisely three HT-models (unlike in the proof,* $\langle\emptyset,ab\rangle$ *does not cause a contradiction):*

$$\langle\emptyset,\emptyset\rangle \qquad \langle\emptyset,ab\rangle \qquad \langle ab,ab\rangle$$

Consider f *over* $\mathcal{C}_n$ *such that* $\mathcal{HT}(\mathsf{f}(P,V))$ *contains exactly the HT-models above. By Def. 2,* 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* $R\in\mathcal{C}_n$ *with* $\mathcal{A}(R)\subseteq\mathcal{A}\backslash V$*. Thus, for Thm. 2 to hold we need to find* $R\in\mathcal{C}_n$ *that violates the previous condition. Taking into account the set of HT-models of* $\mathsf{f}(P,V)$*, for every program* R *that violates the* $\mathbf{(SP)}_{\langle P,V\rangle}$ *condition,* $\mathcal{HT}(R)$ *is required to contain* $\langle ab,ab\rangle$*,* $\langle a,ab\rangle$*, and* $\langle b,ab\rangle$*, but not* $\langle\emptyset,ab\rangle$*. However, such set of HT-models does not satisfy the condition of* here-intersection *for HT-models of normal programs (cf. [59] and* $(N3)$ *in Lemma 3 in the Appendix), hence no such* $R\in\mathcal{C}_n$ *exists.*

An interesting consequence of this observation is that, unlike verifying relativized and strong equivalence, it does not suffice to consider only unary programs R for verifying the condition of $\mathbf{(SP)}_{\langle P,V\rangle}$. This effectively prevents the applicability of Thm. 2 to operators defined over normal programs. Still, this is not problematic, since, as already pointed out in Example 5, operators over $\mathcal{C}_n$ (and $\mathcal{C}_d$) are of limited interest in the context of forgetting while preserving **(SP)**.

Turning back to the reasons why we cannot forget some sets of atoms from a given program, we can observe – both in the proof of Thm. 1 as well as when the criterion Ω is satisfied – that they seem to be strongly connected to the presence of atoms to be forgotten that depend on themselves via an even (and non-zero) number of negations. This could raise the question of whether it is possible to forget about sets of atoms under certain restrictions. For example, we can verify that criterion Ω is not satisfied for forgetting only about p from the normal program used in the proof of Thm. 1, i.e., we can forget about p in this case. So it would seem that forgetting about certain sets of atoms should be a more admissible problem. Unfortunately, this is not the case, since even if we were to delimit V, in general it is always possible to find a program from which we cannot forget V while satisfying **(SP)**.

Proposition 3. *Let* $V\subseteq\mathcal{A}$ *be some set of atoms. Then, there is* $P\in\mathcal{C}_e$ *such that every forgetting operator* f *over* $\mathcal{C}_e$ *does not satisfy* $\mathbf{(SP)}_{\langle P,V\rangle}$.

Example 11. *Consider forgetting about p from P:*

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

Even though we forget only about one atom p, the argument in the proof of Thm. 1 applies just as well, and no program $\mathsf{f}(P, \{p\})$ exists whose HT-models match the requirements to satisfy **(SP)**$_{\langle P,\{p\}\rangle}$*. Examples for sizes of V greater than one can then straightforwardly be obtained from this example.*

In addition, even if it is possible to forget $V \cup V'$, it may not be possible to forget V and V' in any arbitrary order.

Example 12. *Consider again program P from Ex. 11. Forgetting about b from P first is strongly equivalent to removing the third rule, and subsequently forgetting about p is strongly equivalent to $\{a \leftarrow not\, not\, a\}$. However, forgetting about p from P first while satisfying* **(SP)** *is simply not possible (as shown in Ex. 11).*

This raises another important question: *If a set of atoms V can be forgotten as a whole, is there also at least one proper subset of V that can be forgotten?* Previous arguments, such as in Example 11, might suggest that there is, but it turns out not to be the case.

Proposition 4. *Let P be a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, and f a forgetting operator that satisfies* **(SP)**$_{\langle P,V\rangle}$*. There may not exist any V' with $\emptyset \subset V' \subset V$ such that f satisfies* **(SP)**$_{\langle P,V'\rangle}$*.*

Hence, even if it is possible to forget a set of atoms, it may be impossible to forget arbitrary proper subsets of it.

4. Forgetting with Strong Persistence

So far we have focused on what cannot be forgotten. We now turn our attention to what can be forgotten and clarify under which circumstances we can forget without losing indirect dependencies between atoms that remain.

4.1. When Can We Forget?

While Ω allows us to test when it is not possible to forget without sacrificing **(SP)**, we do not yet know whether this is a necessary criterion. That being the case would ensure that whenever we forget about V from P, and Ω is not satisfied, **(SP)**$_{\langle P,V\rangle}$ could be obeyed. In the following, we investigate this matter with the aim of being able to determine when we can forget, and, if possible, to characterise those operators that can obtain the desirable result.

To this end, we start by introducing a new class of forgetting operators that is defined by the characterization of the HT-models that represent a result of forgetting about V from P. The definition of this class of operators strongly relies on the sets of sets $\mathcal{R}^{Y}_{\langle P,V\rangle}$, which contains $R^{Y,A}_{\langle P,V\rangle}$ (potentially several

for different $A \subseteq V$) that indicate for which programs R an answer set can be obtained for $(P \cup R)_{\|V}$. As already mentioned, these indications may not coincide for different A, so the intuitive idea here is to only consider those for which there is "consensus". This translates into computing the intersections which amounts to determining the least element in each $\mathcal{R}^Y_{\langle P,V\rangle}$ if they exist, i.e., whenever criterion Ω is not satisfied (see also Ex. 10).

Definition 5 (SP-Forgetting). *The class of forgetting operators* F_{SP} *is defined by the following set:*

$$\{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X,Y\rangle \mid Y \subseteq \mathcal{A}(P)\backslash V \wedge X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}\} \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}$$

Note that some operator f is part of the class if the condition holds for all programs in the class for which f is defined.

Example 13. *Consider the following program* P*:*

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

Forgetting $\{p\}$ *from* P *while satisfying* **(SP)** *should preserve all dependencies between* a *and* b*. Such dependencies are internalized in the set of HT-models of* P*. In this case, all models in* $\mathcal{HT}(P)$ *are of the form* $\langle X,Y\rangle$ *with* $Y = \{p\}$*,* $Y = \{a,p\}$*,* $Y = \{b,p\}$*,* $Y = \{a,b\}$ *or* $Y = \{a,b,p\}$*. Some of these models are however not relevant from the point of view of* **(SP)***. For example, since* $\langle b, bp\rangle \in \mathcal{HT}(P)$*, we have that* $\{b,p\}$ *can never be an answer set of* $(P \cup R)$*, for a program* R *over* $\{a,b\}$*. This is due to the fact that no* R *over* $\{a,b\}$ *can distinguish the models* $\langle b, bp\rangle$ *and* $\langle bp, bp\rangle$*, i.e., one is a model of* R *if and only if the other one is. The same type of argument applies to* $Y = \{a,b,p\}$ *since* $\langle ab, abp\rangle \in \mathcal{HT}(P)$*. Therefore,* $\{p\} \notin Rel^{\{b\}}_{\langle P,V\rangle}$ *and* $\{p\} \notin Rel^{\{a,b\}}_{\langle P,V\rangle}$*, which intuitively means that models of* P *of the form* $\langle X, bp\rangle$ *and of the form* $\langle X, abp\rangle$ *are not relevant for the existence of models of the form* $\langle X, b\rangle$ *and* $\langle X, ab\rangle$ *in* $\mathsf{f}(P,\{p\})$*, respectively. We obtain that* $\mathcal{R}^{\{b\}}_{\langle P,V\rangle} = \emptyset$*, which means that* $\mathsf{f}(P,V)$ *has no HT-model of the form* $\langle X, \{b\}\rangle$*. For the other possible models we have* $Rel^{\emptyset}_{\langle P,\{p\}\rangle} = \{\{p\}\}$*,* $Rel^{\{a\}}_{\langle P,\{p\}\rangle} = \{\{a\}\}$ *and* $Rel^{\{a,b\}}_{\langle P,\{p\}\rangle} = \{\{a,b\}\}$*. In each of the three cases, the set* $\mathcal{R}^{Y\setminus\{p\}}_{\langle P,V\rangle}$ *has only one element, thus the intersection is precisely that element. This immediately implies that the forgetting instance* $\langle P,\{p\}\rangle$ *does not satisfy criterion* Ω*. The resulting set of HT-models of* $\mathsf{f}(P,V)$ *for* $f \in \mathsf{F}_{\mathsf{SP}}$*, is* $\{\langle\emptyset,\emptyset\rangle, \langle\emptyset,a\rangle, \langle a,a\rangle, \langle a,ab\rangle, \langle ab,ab\rangle\}$*. This means that for every* $f \in \mathsf{F}_{\mathsf{SP}}$*, the program* $\mathsf{f}(P,V)$ *is strongly equivalent to* $\{a \leftarrow not\, not\, b\}$*.*

Example 14. *Recall program* P *from Ex. 6 and the observations in Ex. 7 and 8. Namely,* $Rel^{\emptyset}_{\langle P,V\rangle} = \emptyset$*,* $Rel^{\{a\}}_{\langle P,V\rangle} = \{\{p\}\}$*,* $Rel^{\{b\}}_{\langle P,V\rangle} = \{\{q\}\}$*, and* $Rel^{\{a,b\}}_{\langle P,V\rangle} = \{\{p\},\{q\}\}$*. In the first two cases, the intersection* $\bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ *coincides with the only existing element, and in the latter case, we have* $R^{\{a,b\},\{p\}}_{\langle P,V\rangle} = \{\{a,b\},\{a\}\}$ *and* $R^{\{a,b\},\{q\}}_{\langle P,V\rangle} = \{\{a,b\},\{b\}\}$*. Thus, forgetting instance* $\langle P,\{p,q\}\rangle$ *satisfies*

criterion Ω *and* $\bigcap \mathcal{R}^{\{a,b\}}_{\langle P,V\rangle} = \{\{a,b\}\}$. *The resulting set of HT-models of* $\mathsf{f}(P,V)$ *for* $f \in \mathsf{F_{SP}}$, *is* $\{\langle a,a\rangle, \langle b,b\rangle, \langle ab,ab\rangle\}$. *This means that, for every* $f \in \mathsf{F_{SP}}$, *the program* $\mathsf{f}(P,V)$ *is strongly equivalent to*

$$a \leftarrow not\ b \quad b \leftarrow not\ a \quad a \leftarrow not\ not\ a, not\ not\ b \quad b \leftarrow not\ not\ a, not\ not\ b.$$

In Sec. 3, we have argued that forgetting operators over $\mathcal{C}_n$ and $\mathcal{C}_d$ are not desirable in the context of forgetting while preserving **(SP)**. This is further strengthened by the fact that $\mathsf{F_{SP}}$ does not contain operators defined (only) over these classes of programs.

Example 15. *Let* P *be the normal program in Example 5. This program has 6 HT-models, namely* $\langle p,p\rangle, \langle q,q\rangle, \langle pq,pq\rangle, \langle p,pq\rangle, \langle q,pq\rangle, \langle \emptyset,pq\rangle$. *Therefore, for every* $f \in \mathsf{F_{SP}}$, *we have that* $\mathcal{HT}(\mathsf{f}(P,\{q\})) = \{\langle p,p\rangle, \langle \emptyset,\emptyset\rangle\}$. *Then* $\mathcal{AS}(\mathsf{f}(P,\{q\})) = \{\emptyset, \{p\}\}$. *But no normal nor disjunctive program has comparable answer sets. Therefore, there is no operator* $f \in \mathsf{F_{SP}}$ *such that* $\mathcal{C}(\mathsf{f})$ *is* $\mathcal{C}_n$ *or* $\mathcal{C}_d$.

In principle, the previous observation could raise the question whether the class $\mathsf{F_{SP}}$ is indeed well-defined. However, the close connection between the definition of $\mathsf{F_{SP}}$ and criterion Ω is not a mere coincidence. It turns out that every operator in $\mathsf{F_{SP}}$ in fact satisfies **(SP)** for those instances $\langle P,V\rangle$ that do not satisfy Ω.

Theorem 3. *Every* $\mathsf{f} \in \mathsf{F_{SP}}$ *satisfies* $\mathbf{(SP)}_{\langle P,V\rangle}$ *for every* $\langle P,V\rangle$ *over* $\mathcal{C}(\mathsf{f})$ *that does not satisfy* Ω.

It immediately follows from Thms. 2 and 3 that Ω is a necessary and sufficient criterion for determining whether it is possible to forget about V from P and preserve **(SP)**.

Corollary 1. *Every forgetting operator* $\mathsf{f} \in \mathsf{F_{SP}}$ *satisfies* $\mathbf{(SP)}_{\langle P,V\rangle}$ *iff* $\langle P,V\rangle$ *over* $\mathcal{C}(\mathsf{f})$ *does not satisfy* Ω.

Also, whenever it is possible to forget, the class $\mathsf{F_{SP}}$ precisely characterises the result of forgetting.

Proposition 5. *Let* $\langle P,V\rangle$ *be an instance over* $\mathcal{C}$ *that does not satisfy* Ω. *Then, for every forgetting operator* f *satisfying* $\mathbf{(SP)}_{\langle P,V\rangle}$ *and every* $\mathsf{f}' \in \mathsf{F_{SP}}$ *with* $\mathcal{C} \subseteq \mathcal{C}(\mathsf{f})$ *and* $\mathcal{C} \subseteq \mathcal{C}(\mathsf{f}')$, *we have that* $\mathsf{f}(P,V) \equiv_{\mathsf{HT}} \mathsf{f}'(P,V)$.

For the particular case of Horn programs, it is possible to simplify the construction of the result since the set of HT-models of the result of forgetting coincides with the set of HT-models of the original program, modulo the forgotten atoms.

Proposition 6. *Let* f *be in* $\mathsf{F_{SP}}$. *Then, for every* $V \subseteq \mathcal{A}$:

$$\mathcal{HT}(\mathsf{f}(P,V)) = \mathcal{HT}(P)_{\|V} \text{ for } P \in \mathcal{C}_H.$$

Example 16. *Consider the following Horn program P and $V = \{b\}$:*

$$a \leftarrow b \qquad\qquad b \leftarrow c$$

Here, $\mathcal{HT}(P)$ contains 10 elements:

$$\begin{array}{ccccc} \langle \emptyset, \emptyset \rangle & \langle a, a \rangle & \langle a, ab \rangle & \langle \emptyset, abc \rangle & \langle ab, abc \rangle \\ \langle \emptyset, a \rangle & \langle \emptyset, ab \rangle & \langle ab, ab \rangle & \langle a, abc \rangle & \langle abc, abc \rangle \end{array}$$

We have $Rel^{\emptyset}_{\langle P,V \rangle} = \{\emptyset\}$, $Rel^{\{a\}}_{\langle P,V \rangle} = \{\emptyset\}$, $Rel^{\{c\}}_{\langle P,V \rangle} = \emptyset$, and $Rel^{\{a,c\}}_{\langle P,V \rangle} = \{\{b\}\}$. Thus, according to Def. 5, $\mathcal{HT}(\mathsf{f}(P,V))$ for $f \in \mathsf{F_{SP}}$ contains the following 6 elements:

$$\langle \emptyset, \emptyset \rangle \qquad \langle \emptyset, a \rangle \qquad \langle a, a \rangle \qquad \langle \emptyset, ac \rangle \qquad \langle a, ac \rangle \qquad \langle ac, ac \rangle$$

It is easy to verify that the same result can be obtained using Prop. 6 directly by simply omitting b from the 10 HT-models of P (some of which collapse into the same resulting HT-model). Thus, for every $f \in \mathsf{F_{SP}}$, the program $\mathsf{f}(P,V)$ is strongly equivalent to $\{a \leftarrow c\}$.

In the previous example, the forgetting result of a Horn program is again a Horn program and this raises the question whether this is also the case in general, i.e., whether $\mathsf{F_{SP}}$ is closed for Horn programs. Subsequently, we can pose the same question for the other classes of programs considered. As it turns out, we can show that $\mathsf{F_{SP}}$ is closed in the general case (i.e. for extended programs) and for Horn programs, but not for disjunctive nor normal programs. This is in fact similar to previous classes of forgetting operators defined for the class of extended programs that are based on manipulating HT-models, namely HT-forgetting [46] and SM-forgetting [44].

Theorem 4. *$\mathsf{F_{SP}}$ is closed for extended programs and Horn programs, but neither for disjunctive programs nor normal programs.*

We state the result in the general case here, i.e., independently of whether we are allowed to forget or not, but these results obviously apply in exactly the same way if we restrict our attention to the cases where we can forget about some V from a given program P (also see Ex. 15).

These results actually refer to the property **($\mathbf{E}_\mathcal{C}$)** for different $\mathcal{C}$ which raises the question of how $\mathsf{F_{SP}}$ fares w.r.t. the other properties mentioned in the literature. Here, we restrict our attention to the cases where we can forget, i.e., where the considered instance does not satisfy Ω, because that is the primary intended use case of $\mathsf{F_{SP}}$. Then, most of the properties mentioned in Sec. 2 are satisfied.

Theorem 5. *Restricted to instances $\langle P,V \rangle$ that do not satisfy Ω, $\mathsf{F_{SP}}$ satisfies* **(wE)**, **(SE)**, **(PP)**, **(SI)**, **(sC)**, **(wC)**, **(CP)**, *and* **(SP)**.

The only property which is not satisfied, **(W)**, has been proved orthogonal to **(SP)** – in fact, to every property that aims at preserving answer sets [49].

4.2. How Can We Forget?

We have seen that, given an instance $\langle P, V\rangle$, we can test whether Ω is not satisfied, i.e., whether we are allowed to forget V from P while preserving **(SP)**, in which case we can compute the HT-models that characterize a result of forgetting using the definition of F_{SP}. However, this does not provide an actual forgetting result, nor guarantee that an operator actually exists. We overcome this limitation and provide a concrete forgetting operator based on the notion of countermodels in here-and-there [56], which has been used previously in a similar manner for computing concrete results of forgetting for classes of forgetting operators based on HT-models [44, 46].

Intuitively, the idea of HT-countermodels can be summarized as follows. Given a program P, the HT-interpretations that are not HT-models of P (hence the name countermodels) can be used to determine rules, that, if conjoined, result in a program P' that is strongly equivalent to P. Here, we utilize this method to create an actual program, viz. a forgetting result, whose set of HT-models corresponds to the semantic characterization of any forgetting result for F_{SP}. For that purpose, we first recall some notation on countermodels from [56] adjusting it to the syntax of rules we use.

Let P be a program and $X \subseteq Y \subseteq \mathcal{A}$. An HT-interpretation $\langle X, Y\rangle$ is an *HT-countermodel* of P if $\langle X, Y\rangle \not\models P$. We also define the following rules:[7]

$$r^{\mathcal{A}}_{X,Y} = (Y \backslash X) \leftarrow X, not\,(\mathcal{A} \backslash Y), not\,not\,(Y \backslash X) \tag{3}$$

$$r^{\mathcal{A}}_{Y,Y} = \emptyset \leftarrow Y, not\,(\mathcal{A} \backslash Y) \tag{4}$$

Countermodels and the resulting rules are in fact closely connected [56]. Here, we recall these results as presented in [46].

Proposition 7 ([46]). *Let $X \subset Y \subseteq \mathcal{A}$ and $U \subseteq V \subseteq \mathcal{A}$.*

(i) $\langle U, V\rangle$ is an HT-countermodel of $r^{\mathcal{A}}_{X,Y}$ iff $U = X$ and $V = Y$.

(ii) $\langle U, V\rangle$ is an HT-countermodel of $r^{\mathcal{A}}_{Y,Y}$ iff $V = Y$.

This allows us to determine a program for a set of HT-models provided such program exists. Recall that not all sets of HT-interpretations correspond to the set of HT-models of some program. A set of HT-interpretations S is *HT-expressible* iff $\langle X, Y\rangle \in S$ implies $\langle Y, Y\rangle \in S$. In this case, we are able to determine a corresponding program.

Proposition 8 ([46]). *Let M be a set of HT-interpretations which is HT-expressible and define the program P_M as*

$$P_M = \{r^{\mathcal{A}}_{X,Y} \mid \langle X, Y\rangle \notin M \text{ and } \langle Y, Y\rangle \in M\} \cup \{r^{\mathcal{A}}_{Y,Y} \mid \langle Y, Y\rangle \notin M\}.$$

Then, $\mathcal{HT}(P_M) = M$.

[7] In the context of forgetting, we also make the signature used for these rules explicit. This does not affect the previous technical results, but eases the reading.

Thus, we can determine a program based on the HT-countermodels that has precisely the same HT-countermodels, and thus the same HT-models.

Example 17. *Consider the program P (over $\{a, b\}$) consisting of two rules $a \leftarrow b$ and $b \leftarrow a$. Its HT-models are $\langle \emptyset, \emptyset \rangle$, $\langle \emptyset, ab \rangle$, and $\langle ab, ab \rangle$. Thus, the HT-countermodels relevant for the construction of P_M are $\langle a, a \rangle$, $\langle b, b \rangle$, $\langle a, ab \rangle$, and $\langle b, ab \rangle$. This yields P_M containing the following four rules:*

$$r^{\{a,b\}}_{\{a\},\{a\}} = \bot \leftarrow a, not\ b \tag{5}$$

$$r^{\{a,b\}}_{\{b\},\{b\}} = \bot \leftarrow b, not\ a \tag{6}$$

$$r^{\{a,b\}}_{\{a\},\{ab\}} = b \leftarrow a, not\ not\ b \tag{7}$$

$$r^{\{a,b\}}_{\{b\},\{ab\}} = a \leftarrow b, not\ not\ a \tag{8}$$

We can verify that $\mathcal{HT}(P_M) = \mathcal{HT}(P)$ and that P_M can in fact be simplified to P without much effort (the additional double-negated atoms in (7) and (8) require the existence of rules (6) and (5), respectively).

Note that the formulation in Prop. 8 differs from the original one [56] in that, whenever $\langle Y, Y \rangle \notin M$ only $r^{\mathcal{A}}_{Y,Y}$ itself is considered, whereas in [56] also all $r^{\mathcal{A}}_{X,Y} \mid \langle X, Y \rangle$ with $X \subset Y$ are included.

We are now ready to provide an algorithm, Alg. 1, that computes an actual forgetting result for $\mathsf{F_{SP}}$, which we denote $\mathsf{f_{SP}}$. It proceeds as follows. After initializing the resulting program P' and fixing its signature $\mathcal{A}' = \mathcal{A} \backslash V$, we determine for which $Y \subseteq \mathcal{A}'$ we have that $\mathcal{R}^Y_{\langle P,V \rangle}$ is empty. In this case, $\langle Y, Y \rangle \notin \mathcal{HT}(\mathsf{f}(P, V))$ for every $\mathsf{f} \in \mathsf{F_{SP}}$, hence $\langle Y, Y \rangle$ is an HT-countermodel and we add $r^{\mathcal{A}'}_{Y,Y}$ to the forgetting result. Notably, we use $\mathcal{A}'$ to avoid that atoms to be forgotten appear in the added rule. Alternatively, if $\mathcal{R}^Y_{\langle P,V \rangle}$ is not empty, we determine which X do not appear in $\bigcap \mathcal{R}^Y_{\langle P,V \rangle}$. For those, by definition, $\langle X, Y \rangle \notin \mathcal{HT}(\mathsf{f}(P, V))$ for every $\mathsf{f} \in \mathsf{F_{SP}}$, hence $\langle X, Y \rangle$ is an HT-countermodel and we add $r^{\mathcal{A}'}_{X,Y}$ to the forgetting result.

We can show that Alg. 1 does terminate and that the HT-models of the result correspond to the semantic characterization of $\mathsf{F_{SP}}$.

Theorem 6. *Alg. 1 terminates and $\mathsf{f_{SP}} \in \mathsf{F_{SP}}$.*

Thus, $\mathsf{f_{SP}}$ as described in Alg. 1 is actually an operator of $\mathsf{F_{SP}}$ and serves as a witness for the existence of such operators in the general case, unlike, e.g., the class F_{Sas} which aims at forgetting and satisfying **(SP)**, but only achieves this in a very restricted setting [45].

Example 18. *Consider again program P from Ex. 6 with $V = \{p, q\}$. We can compute $\mathsf{f_{SP}}(P, V)$ and obtain the following program:*

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

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

Algorithm 1: Computing $\mathsf{f}_{\mathsf{SP}}(P, V)$ for forgetting about V from P

Input : Program P and $V \subseteq \mathcal{A}$
Output: Program $P' = \mathsf{f}_{\mathsf{SP}}(P, V)$

$P' := \emptyset$;
$\mathcal{A}' := \mathcal{A} \backslash V$;
for $Y \subseteq \mathcal{A}'$ **do**
  if $\mathcal{R}^Y_{\langle P,V \rangle} = \emptyset$ **then**
    $P' := P' \cup \{r^{\mathcal{A}'}_{Y,Y}\}$;
  end
  else
    for $X \subseteq Y$ *s.t.* $X \notin \bigcap \mathcal{R}^Y_{\langle P,V \rangle}$ **do**
      $P' := P' \cup \{r^{\mathcal{A}'}_{X,Y}\}$;
    end
  end
end

Comparing this result with that obtained in Ex. 14 we observe slight syntactic differences that can be explained in a similar way as in Ex. 17.

Note that, by Prop. 6, there is an easier way to determine the HT-models for the characterization of the result of forgetting w.r.t. F_{SP} if the considered program is Horn. In this case, we can re-use the operator presented for Semantic Forgetting [46], but restricted to Horn programs, which is simpler and provides a further instance of an operator in F_{SP}.

4.3. What Can We Forget?

We now approach the problem from a different angle, and determine which sets of atoms can be forgotten from a specific program.

We begin with the case where the set of atoms to be forgotten is a singleton and the program normal. As it turns out, we can always forget single atoms from normal programs without having to test Ω.

Proposition 9. *There is a forgetting operator satisfying* $\mathbf{(SP)}_{\langle P,V \rangle}$*, for every* $P \in \mathcal{C}_n$ *and every* V*, such that* $|V| = 1$.

Essentially, this result follows from the fact that if we forget only one atom, then every $\mathcal{R}^Y_{\langle P,V \rangle}$ contains at most two elements, namely $R^{Y,\emptyset}_{\langle P,V \rangle}$ and $R^{Y,V}_{\langle P,V \rangle}$. Then, we can use specific conditions on the HT-models of normal programs to show that the former is always least.

Example 19. *Recall again program P from Ex. 6. We already know that we cannot forget about $V = \{p, q\}$ (cf. the proof of Thm. 1). However, by Prop. 9, we can forget about p or q alone, as well as forget only about a or b. By forgetting*

about q alone from P we obtain the program of Ex. 11. Since this is no longer a normal program, we cannot use again Prop. 9 to subsequently forget about p, as illustrated in Ex. 11.

It turns out that such result also holds for the class of disjunctive programs and the rationale is based on a very similar argument as for normal programs.

Proposition 10. *There is a forgetting operator satisfying* $\mathbf{(SP)}_{\langle P,V\rangle}$*, for every $P \in \mathcal{C}_d$ and every V, such that $|V| = 1$.*

As indicated in Thm. 4, no operator in F_{SP} is closed for normal or disjunctive programs, hence quite likely the result will not be applicable throughout an iterative process of forgetting one atom at a time. But for a one-time forgetting operation, they might be useful.

We now provide a general way to determine which sets of atoms can be forgotten from a given program.

Theorem 7. *Let P be a program. Then the set of sets of atoms*

$$\mathcal{V}_P = \{V \subseteq \mathcal{A}(P) \mid \bigcap \mathcal{R}^Y_{\langle P,V\rangle} \in \mathcal{R}^Y_{\langle P,V\rangle} \text{ for every } \mathcal{R}^Y_{\langle P,V\rangle} \neq \emptyset\}$$

is the set of all sets V of atoms for which it is possible to forget V from P while satisfying $\mathbf{(SP)}_{\langle P,V\rangle}$*.*

Thus, this result provides a general way to obtain all sets of atoms V that can be forgotten from a given program P while preserving **(SP)**. Notably, all possible sets contained in $\mathcal{V}_P$ have to be determined individually as neither sub- nor supersets are necessarily contained in the result (cf. Prop. 4 for subsets and the observation in Ex. 19 for supersets).

Example 20. *Recall once more program P given in Ex. 6. Clearly, $\mathcal{V}_P$ contains $\{p\}$, $\{q\}$ as well as $\{a\}$ and $\{b\}$ as indicated in Ex. 19 already. In addition, by Thm. 7, we can forget about, e.g., $\{p, q, a\}$ (and symmetrically $\{p, q, b\}$, but not $\{p, q\}$. In fact, we can verify that, among all $V \subseteq \mathcal{A}(P)$, $V = \{p, q\}$ is the only (non-empty) set of atoms which cannot be forgotten.*

5. Beyond Strong Persistence when Forgetting

We have seen that, although **(SP)** is the central property one wants to ensure to hold when forgetting atoms from an answer set program, this is not always possible. In the previous sections, we have drawn precise limits on forgetting with **(SP)** and studied the problem within such limits. We now focus on the problem of forgetting beyond the limits of **(SP)**, that is, what can we do if it is not possible to forget while satisfying **(SP)**, but we must nevertheless forget. To this end, we investigate three possible alternatives.

5.1. Reusing SP-Forgetting

We first consider $\mathsf{F_{SP}}$ itself as a possible solution. This has the obvious benefit that, whenever we can forget under **(SP)**, i.e. when Ω is not satisfied, we obtain the desired result. When Ω is satisfied, then there are certain sets of rules R over the remaining language for which the condition of **(SP)** does not hold. Intuitively, this is the case because different sets of HT-models of P, which collapse into one set due to the forgotten atoms, are conflicting over which programs R an answer set can be obtained for $(P \cup R)_{\|V}$. Using $\mathsf{F_{SP}}$ anyway, in such cases, thus provides a way to resolve these conflicts by only adopting those R for which there exists "consensus" between these different collapsing sets of HT-models.

One way of measuring how well this proposed solution works is to look at the properties that are satisfied when applying $\mathsf{F_{SP}}$ in general, and compare to the ideal case (when forgetting is possible). Regarding *existence*, we already showed in general that $\mathsf{F_{SP}}$ is closed for extended programs and Horn programs, but not for disjunctive nor normal programs (Thm. 4). Regarding the other properties, we have seen (Thm. 5) that, when restricted to those instances that do not satisfy Ω, $\mathsf{F_{SP}}$ satisfies most of the properties of forgetting mentioned in Sec. 2.

Before we can proceed to establishing which properties are satisfied by $\mathsf{F_{SP}}$ in general, i.e., independently of whether Ω is satisfied or not, we need to establish some relevant technical results first. Namely, given the pivotal role of programs R over the remaining language, a closer inspection of these programs is necessary. First, we can show that two interpretations that only differ on V, the atoms to be forgotten, are either both models of R or none is.

Lemma 7. *Let $V \subseteq \mathcal{A}$, R a program over $\mathcal{A}\backslash V$, and $\langle X, Y\rangle, \langle X', Y'\rangle$ HT-interpretations s.t. $X \sim_V X'$ and $Y \sim_V Y'$. Then, $\langle X, Y\rangle \in \mathcal{HT}(R)$ iff $\langle X', Y'\rangle \in \mathcal{HT}(R)$.*

This result is useful as it allows us to establish that HT-models of R from which we remove the atoms to be forgotten (that are not mentioned in R) are naturally HT-models of R.

Lemma 8. *Let $V \subseteq \mathcal{A}$, R a program over $\mathcal{A}\backslash V$, and $X, Y \subseteq \mathcal{A}\backslash V$. Then, $\langle X, Y\rangle \in \mathcal{HT}(R)_{\|V}$ iff $\langle X, Y\rangle \in \mathcal{HT}(R)$.*

In addition, Lemma 7 can be used to show that the set of relevant forgotten atoms for Y w.r.t. $\langle P \cup R, V\rangle$, $Rel^Y_{\langle P\cup R,V\rangle}$, can be characterized using $Rel^Y_{\langle P,V\rangle}$, i.e., the set of relevant forgotten atoms w.r.t. $\langle P, V\rangle$ only.

Lemma 9. *Let P be a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, R a program over $\mathcal{A}\backslash V$, and $A \subseteq V$. Then, $A \in Rel^Y_{\langle P,V\rangle}$ and $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$ iff $A \in Rel^Y_{\langle P\cup R,V\rangle}$.*

This result is important whenever we want to correlate HT-models of program P and program $P \cup R$, which is particularly relevant for property **(SI)**.

We can now characterize which properties are satisfied by $\mathsf{F_{SP}}$ in general.

Proposition 11. $\mathsf{F_{SP}}$ *satisfies* **(SE)**, **(PP)**, **(SI)**, *and* **(wC)**, *but does not satisfy* **(wE)**, **(W)**, **(sC)**, *and* **(CP)**.

From the previous proposition, we observe that **(W)** is no longer the only property that does not hold. Notably, the fact that $\mathsf{F_{SP}}$ does not satisfy **(CP)**, and in particular **(sC)**, means that there are instances $\langle P, V\rangle$ for which the result of forgetting about V from P has answer sets that do not correspond to answer sets in the original program P, which is also why (**wE**) does not hold.

Example 21. *Consider again program P of Ex. 11.*

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

Clearly, P has six HT-models, $\langle ap, ap\rangle, \langle b, b\rangle, \langle b, ab\rangle, \langle ab, ab\rangle, \langle ap, abp\rangle, \langle abp, abp\rangle$, and two answer sets $\{a, p\}$ and $\{b\}$. Intuitively, p yields an exclusive choice between a and b. If we take $V = \{p\}$, then, $\mathcal{R}^{\emptyset}_{\langle P,V\rangle} = \emptyset$, $\mathcal{R}^{\{a\}}_{\langle P,V\rangle} = \{\{a\}\}$, $\mathcal{R}^{\{b\}}_{\langle P,V\rangle} = \{\{b\}\}$, and $\mathcal{R}^{\{a,b\}}_{\langle P,V\rangle} = \{\{b, ab\}, \{a, ab\}\}$. From this we have that $\bigcap \mathcal{R}^{\emptyset}_{\langle P,V\rangle} = \emptyset$, $\bigcap \mathcal{R}^{\{a\}}_{\langle P,V\rangle} = \{a\}$, $\bigcap \mathcal{R}^{\{b\}}_{\langle P,V\rangle} = \{b\}$, and $\bigcap \mathcal{R}^{\{a,b\}}_{\langle P,V\rangle} = \{ab\}$. This means that, for every $\mathsf{f} \in \mathsf{F_{SP}}$, $\mathsf{f}(P, V)$ has three HT-models, $\langle a, a\rangle, \langle b, b\rangle, \langle ab, ab\rangle$, which means that $\mathsf{f}(P, V)$ has three answer sets, the two from P ignoring p, $\{a\}$ and $\{b\}$, and additionally $\{a, b\}$. Intuitively, this happens because using the intersection essentially discards both $\langle b, ab\rangle$ and $\langle ap, abp\rangle$ (modulo the forgotten p).

This is, in fact, rather atypical, as so far no class of forgetting operators that satisfies **(wC)**, but not **(sC)**, and thus not **(CP)**, was known [49]. Since the violation of **(sC)** may be seen as sufficient cause to render $\mathsf{F_{SP}}$ inadequate when Ω is satisfied – notably when the introduction of new answer sets as the result of forgetting cannot be accepted – alternatives need to be investigated.

5.2. Relativized Forgetting

One way to proceed is to explore alternative ways to forgetting in ASP by borrowing from the notion of relativized equivalence [50]. Relativized equivalence is a generalization of strong equivalence that considers equivalence w.r.t. a given subset of the language, such that equivalence and strong equivalence are their special cases (for the empty and the entire language respectively). This fits naturally within the idea of forgetting in ASP, in particular w.r.t. property **(SP)**, inasmuch as after forgetting about V from P we only allow the addition of programs over $\mathcal{A}\backslash V$, so relativized (strong) equivalence should be applied accordingly.

Based on this idea, we first define a forgetting operator that simply considers all logical consequences w.r.t. relativized equivalence. This way, the result of forgetting about V from P amounts to the set of all rules (over $\mathcal{A}\backslash V$) that can be added to P while preserving relativized equivalence. Given a program P and $V \subseteq \mathcal{A}$, we consider the closure of P given V:

$$Cn(P, V) = \{r \mid \{r\} \in \mathcal{C}_e \text{ and } P \cup \{r\} \equiv_V P\}.$$

Then, the result of forgetting about a set of atoms V from program $P \in \mathcal{C}_e$ is defined as

$$\mathsf{f}_\mathsf{r}(P, V) = \{r \mid r \in Cn(P, V) \text{ and } \mathcal{A}(\{r\}) \cap V = \emptyset\}.$$

Clearly, the resulting program does not mention the forgotten atoms, and we can show that this operator does not belong to F_SP.

Example 22. *Recall program P from Ex. 21 with $V = \{p\}$. Since we are interested in relativized equivalence, we focus on the V-HT-models of P:*

$$\langle b, b\rangle \qquad \langle ap, ap\rangle \qquad \langle b, ab\rangle \qquad \langle ab, ab\rangle \qquad \langle a, abp\rangle \qquad \langle abp, abp\rangle$$

It can be verified that $\mathsf{f}_\mathsf{r}(P, \{p\})$ is strongly equivalent to the following program:

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

Thus, for these four rules r, we have $P \cup \{r\} \equiv_V P$, i.e., $\mathcal{HT}_V(P \cup \{r\}) = \mathcal{HT}_V(P)$, and every other rule over $\{a, b\}$ which also satisfies this condition does not affect the HT-models of this result (in principle, the definition of f_r requires us to present all the possible rules, but that is evidently not a good idea due to tautological rules, rules whose HT-models coincide but are syntactically different, etc.). Notably, this program does not have the answer set $\{a, b\}$, which indicates that this operator does not belong to F_SP (cf. the observation in Ex. 21).

It is however unclear whether two rules that individually satisfy the condition in $Cn(P, V)$ will also satisfy this condition together. Fortunately, we can show that f_r is well-defined, in the sense that testing relativized equivalence for each rule individually is the same as testing the entire set of rules as a whole.

Proposition 12. *Let P be a program, $V \subseteq \mathcal{A}$ and R_1, R_2 programs over $\mathcal{A}\backslash V$. Then, $P \cup R_1 \cup R_2 \equiv_V P$ iff $P \cup R_1 \equiv_V P$ and $P \cup R_2 \equiv_V P$.*

As a consequence of the above result, we can test whole sets of rules R directly and include them all in $Cn(P, V)$ collectively provided the test for relativized equivalence succeeds. Moreover, $\mathsf{f}_\mathsf{r}(P, V)$ is in fact the largest set of rules over $\mathcal{A}\backslash V$ that can be safely added to P without changing its set of V-HT-models.

Proposition 13. *Let P be a program and $V \subseteq \mathcal{A}$. Then, $\mathsf{f}_\mathsf{r}(P, V)$ is the largest set of rules R over the alphabet $\mathcal{A}\backslash V$ such that $P \cup R \equiv_V P$.*

We could now define a (possibly singleton) class of operators that generalizes the idea of f_r in a straightforward manner, and then study this class, but its definition would not be very concise since, in the worst case, we would have to check whether each possible rule over the remaining language is relativized equivalent to the original program. In addition, as discussed in Ex. 22, we would have to determine a set of rules that suffice to represent the result, unless we

are ready to present exponentially many rules most of which are semantically irrelevant in terms of their HT-models.

Instead, inspired by knowledge forgetting [46], we follow a different path. We define a class of forgetting operators that consider the V-HT-models of P and omit all occurrences of elements of V from these.

Definition 6 (Relativized Forgetting). *The class of forgetting operators* $\mathsf{F_R}$ *is defined by the following set:*

$$\{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \mathcal{HT}_V(P)_{\|V} \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}$$

Similar to SP-Forgetting, this definition is based on a characterization of the HT-models of the forgetting result for any operator f, program P, and atoms V to be forgotten. Note that while $\mathsf{f_r}$ is defined for $\mathcal{C}_e$, operators in $\mathsf{F_R}$ can be defined over subclasses of programs. Nevertheless, we can show, in a similar way as shown in Ex. 15 for the case of $\mathsf{F_{SP}}$, that there are no operators in $\mathsf{F_R}$ over $\mathcal{C}_n$ or $\mathcal{C}_d$. Namely, for the program considered in this example, we obtain $\mathcal{HT}_V(P)_{\|V} = \{\langle p,p\rangle, \langle \emptyset,\emptyset\rangle\}$. Hence, there are two comparable answer sets, $\emptyset$ and $\{p\}$, which is not possible for normal or disjunctive programs.

Example 23. *Recall program P from Ex. 21 and its V-HT models for $V = \{p\}$ from Ex. 22. Then, by Def. 6, we obtain $\mathcal{HT}(\mathsf{f}(P,V))$ as follows:*

$$\langle b,b\rangle \qquad \langle a,a\rangle \qquad \langle a,ab\rangle \qquad \langle b,ab\rangle \qquad \langle ab,ab\rangle$$

It can be verified that the program presented for $\mathsf{f_r}(P,\{p\})$ in Ex. 22 has precisely these HT-models (over $\{a,b\}$), i.e., the result of forgetting about p from P for every $\mathsf{f} \in \mathsf{F_R}$ is strongly equivalent with that for $\mathsf{f_r}$.

It turns out that this correspondence is no mere coincidence. In fact, we show in the following that $\mathsf{f_r} \in \mathsf{F_R}$, and in the course of that, we establish a precise relation between the HT-models and the V-HT-models of a program. This is an important contribution, since it allows the usage of well-known properties of HT-models, such as monotonicity, that are not satisfied by V-HT-models (see [50]).

First, we introduce an alternative characterization of the V-HT-models of a program P based on its HT-models using the following notion.

Definition 7. *Let P be a program and $Y, V \subseteq \mathcal{A}$. Then, Y is* relevant for P w.r.t. V *if*

(i) $\langle Y,Y\rangle \in \mathcal{HT}(P)$

(ii) $\langle Y',Y\rangle \notin \mathcal{HT}(P)$ *for every* $Y' \subset Y$ *s.t.* $Y \sim_V Y'$.

$Rel(P,V)$ denotes the set of all sets relevant for P w.r.t. V.

This notion is tightly connected to the set of sets of relevant forgotten atoms for Y in Def. 3 used in the definition of criterion Ω.

Proposition 14. *Let $A \subseteq V \subseteq \mathcal{A}$ and $Y \subseteq \mathcal{A} \backslash V$. Then,*

$$Y \cup A \in Rel(P, V) \text{ iff } A \in Rel^{Y}_{\langle P, V \rangle}.$$

This allows the alternative definition of a V-HT-model in terms of HT-models.

Proposition 15. *Let P be a program and $V \subseteq \mathcal{A}$. Then, a V-HT-interpretation $\langle X, Y \rangle$ is a V-HT-model of P iff the following conditions hold:*

(1) $Y \in Rel(P, V)$;

(2) If $X \subset Y$, then there exists $X' \subset Y$ with $X = X' \backslash V$ such that $\langle X', Y \rangle \in \mathcal{HT}(P)$.

This result can be used to obtain a precise characterization of the total V-HT-models of a program P, a result that will prove useful subsequently.

Lemma 10. *Let P be a program and $V \subseteq \mathcal{A}$. Then, $\langle Y, Y \rangle \in \mathcal{HT}_V(P)$ iff $Y \in Rel(P, V)$.*

We can now present an alternative characterization of the set of V-HT-models of a program in terms of its set of HT-models. This result is particularly useful, even beyond the scope of forgetting, since it shows how the set of V-HT-models of a program can be directly obtained from its set of HT-models.

Proposition 16. *Let P be a program and $V \subseteq \mathcal{A}$. Then,*

$$\mathcal{HT}_V(P) = \bigcup_{Y \in Rel(P,V)} (\{\langle X \backslash V, Y \rangle : \langle X, Y \rangle \in \mathcal{HT}(P) \text{ and } X \subset Y\} \cup \{\langle Y, Y \rangle\}).$$

Example 24. *Consider again program P from Ex. 21 with $V = \{p\}$. We have that $\mathcal{HT}(P)$ contains the following HT-models:*

$$\langle b, b \rangle \quad \langle ap, ap \rangle \quad \langle b, ab \rangle \quad \langle ab, ab \rangle \quad \langle ap, abp \rangle \quad \langle abp, abp \rangle$$

We obtain $Rel(P, V) = \{\{a, p\}, \{b\}, \{a, b\}, \{abp\}\}$ and thus $\mathcal{HT}_V(P)$ contains the following V-HT-models

$$\langle b, b \rangle \quad \langle ap, ap \rangle \quad \langle b, ab \rangle \quad \langle ab, ab \rangle \quad \langle a, abp \rangle \quad \langle abp, abp \rangle$$

which corresponds precisely to the set listed in Ex. 22.

Recall that the class $\mathsf{F_R}$ is defined using $\mathcal{HT}_V(P)_{\|V}$, the V-HT-models of P modulo the atoms to be forgotten. We can show that this set corresponds to the set of HT-models of some program over the remaining language.

Lemma 11. *Let P be a program and $V \subseteq \mathcal{A}$. Then, there is a program R over $\mathcal{A} \backslash V$ such that $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$.*

In fact, such program R over $\mathcal{A}\backslash V$ must be a subset of the result of forgetting according to the concrete operator $\mathsf{f_r}$.

Lemma 12. *Let P be a program, $V \subseteq \mathcal{A}$ and R a program over $\mathcal{A}\backslash V$. If $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$, then $R \subseteq \mathsf{f_r}(P, V)$, i.e., $P \cup \{r\} \equiv_V P$ for every $r \in R$.*

Based on that, we can show that $\mathsf{f_r}$ is indeed a concrete forgetting operator in the class $\mathsf{F_R}$.

Theorem 8. *Let P be a program and $V \subseteq \mathcal{A}$. Then,*

$$\mathcal{HT}(\mathsf{f_r}(P, V))_{\|V} = \mathcal{HT}_V(P)_{\|V}.$$

Although the above result implies that $\mathsf{f_r}$ is a concrete operator in the class $\mathsf{F_R}$, the result of forgetting about V from a program P according to $\mathsf{f_r}$ contains all the possible rules (over the remaining atoms) that are V-HT-equivalent to the given program. Thus, we can show that forgetting according to every operator in the class $\mathsf{F_R}$ produces at most as many rules as $\mathsf{f_r}$.

Proposition 17. *Let $\mathsf{f} \in \mathsf{F_R}$. Then, for every program $P \in \mathcal{C}(\mathsf{f})$ and $V \subset \mathcal{A}$, we have $\mathsf{f}(P, V) \subseteq \mathsf{f_r}(P, V)$.*

Therefore, though $\mathsf{f_r}$ is indeed a concrete operator in the class $\mathsf{F_R}$, it is not recommended for computing results of forgetting according to $\mathsf{F_R}$, since the result will always include, among others, e.g., all possible tautological rules and equivalent rules. So, instead of computing all the possible rules (over the remaining atoms) that are V-HT-equivalent to the given program, we can compute the HT-models according to the definition of Relativized Forgetting and adopt the countermodel construction employed in the previous section to construct a program which is strongly equivalent to the desired forgetting result.

We now provide an algorithm, Alg. 2, that computes an actual forgetting result for $\mathsf{F_R}$, which we denote $\mathsf{f_R}$. It proceeds as follows. After initializing the resulting program P' and fixing its signature $\mathcal{A}' = \mathcal{A}\backslash V$, we determine for which $Y \subseteq \mathcal{A}'$ we have that $Rel^Y_{\langle P,V\rangle}$ is empty. The fact that the set $Rel^Y_{\langle P,V\rangle}$ is empty means, according to Prop. 14, that there is no $A \subseteq V$ such that $Y \cup A \in Rel_{\langle P,V\rangle}$, and, according to Prop. 16, there is no $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}_V(P)$, which implies that $\langle Y, Y\rangle \notin \mathcal{HT}_V(P)_{\|V}$, i.e., $\langle Y, Y\rangle$ is a HT-countermodel of $\mathsf{f}(P, V)$ for every $\mathsf{f} \in \mathsf{F_R}$. Hence, we add $r^{\mathcal{A}'}_{Y,Y}$ to the forgetting result. Note that, as in Alg. 1, we use $\mathcal{A}'$ to avoid that atoms to be forgotten appear in the added rule. Now, in case $Rel^Y_{\langle P,V\rangle}$ is not empty, then $\langle Y, Y\rangle$ is an HT-model of $\mathsf{f}(P, V)$ for every $\mathsf{f} \in \mathsf{F_R}$. Therefore, for the countermodels construction, we need to find those $X \subset Y$ such that $\langle X, Y\rangle$ is an HT-countermodel of $\mathsf{f}(P, V)$ for every $\mathsf{f} \in \mathsf{F_R}$. According to the definition of the class $\mathsf{F_R}$, these $X \subset Y$ are such that $\langle X, Y\rangle \notin \mathcal{HT}_V(P)_{\|V}$, and therefore, for each of such X, we add $r^{\mathcal{A}'}_{X,Y}$ to the forgetting result.

We can show that Alg. 2 does terminate and that the HT-models of the result correspond to the semantic characterization of $\mathsf{F_R}$.

Algorithm 2: Computing $\mathsf{f}_{\mathsf{R}}(P, V)$ for forgetting about V from P

Input : Program P and $V \subseteq \mathcal{A}$
Output: Program $P' = \mathsf{f}_{\mathsf{R}}(P, V)$

$P' := \emptyset$;
$\mathcal{A}' := \mathcal{A} \backslash V$;
for $Y \subseteq \mathcal{A}'$ **do**
  if $Rel^{Y}_{\langle P,V \rangle} = \emptyset$ **then**
    $P' := P' \cup \{r^{\mathcal{A}'}_{Y,Y}\}$;
  end
  else
    for $X \subset Y$ *s.t.* $\langle X, Y \rangle \notin \mathcal{HT}_V(P)_{\|V}$ **do**
      $P' := P' \cup \{r^{\mathcal{A}'}_{X,Y}\}$;
    end
  end
end

Theorem 9. *Alg. 2 terminates and* $\mathsf{f}_{\mathsf{R}} \in \mathsf{F}_{\mathsf{R}}$.

Example 25. *Recall program P from Ex. 21 with $V = \{p\}$. We have seen in Ex. 22 that P has six V-HT-models, that correspond, as shown in Ex. 23, to the six HT-models of $\mathsf{f}(P, V)$ for every $\mathsf{f} \in \mathsf{F}_{\mathsf{R}}$. The result of forgetting V from P according to the concrete operator f_{R}, $\mathsf{f}_{\mathsf{R}}(P, V)$, is the following set of rules, each corresponding, according to Alg. 2, to a countermodel of the desired result of forgetting:*

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

We can verify that this set of rules has the six mentioned HT-models, thus confirming Thm. 9. By comparison, the program $\mathsf{f}_{\mathsf{r}}(P, V)$ contains many more rules than $\mathsf{f}_{\mathsf{R}}(P, V)$. First of all, it contains all tautologies, such as $a \leftarrow a$, $b \leftarrow a, b$, $b \leftarrow a, not\, a$ etc. Then, according to Prop. 17, it contains all four rules of $\mathsf{f}_{\mathsf{R}}(P, V)$. In addition, $\mathsf{f}_{\mathsf{r}}(P, V)$, even in this simple example, contains many more rules that are not tautologies, such as all four rules mentioned in Ex. 22 and, e.g., $a \leftarrow b, not\, not\, a$ and $b \leftarrow a, not\, not\, b$. This means that, although both concrete operators f_{R} and f_{r} are in class F_{R}, f_{R} is by far preferable over f_{r}, since it provides a concise representation of the intended result of forgetting.

The definition of the concrete operators f_{SP} and f_{R} are rather similar in spirit, which is not surprising given their common formal basis of HT-models. However, the two classes F_{SP} and F_{R} of which the latter are instances, have a (maybe) surprisingly close relation, as a result of Props. 14 and 16.

Theorem 10. *Let P be a program and $V \subseteq \mathcal{A}$. Then, $\mathsf{F_R}$ can be defined as*

$$\{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}\} \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}.$$

Thus, this notion of forgetting based on relativized equivalence differs from $\mathsf{F_{SP}}$ by considering the union of the relevant HT-models instead of the intersection, which explains the differences observed in Ex. 21 and 22.

Of course, whenever $\mathcal{R}^Y_{\langle P,V\rangle}$ contains only one element, union and intersection coincide, which is always the case for Horn programs.

Proposition 18. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then, for every $Y \subseteq \mathcal{A}\backslash V$, we have that $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element.*

Thus, when restricted to $\mathcal{C}_H$, $\mathsf{F_R}$ coincides with $\mathsf{F_{SP}}$.

Proposition 19. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then, for every $\mathsf{f} \in \mathsf{F_{SP}}$ and $\mathsf{f}' \in \mathsf{F_R}$ we have that $\mathsf{f}(P,V) \equiv \mathsf{f}'(P,V)$.*

Since this correspondence does not hold in general, we also establish which properties are satisfied by $\mathsf{F_R}$.

Proposition 20. $\mathsf{F_R}$ *satisfies* $(\mathbf{E}_{\mathcal{C}_H})$, $(\mathbf{E}_{\mathcal{C}_e})$, **(SE)**, **(PP)**, **(SI)**, *and* **(sC)**, *but not* $(\mathbf{E}_{\mathcal{C}_n})$, $(\mathbf{E}_{\mathcal{C}_d})$, **(wE)**, **(W)**, **(wC)**, *and* **(CP)**.

In terms of the set of properties under consideration, $\mathsf{F_R}$ and $\mathsf{F_{SP}}$ only differ with respect to **(sC)** and **(wC)**. This difference, however, is crucial. Since $\mathsf{F_R}$ satisfies **(sC)**, it approximates the set of answer sets of P, but, contrary to $\mathsf{F_{SP}}$, never ends up adding new answer sets to the result of forgetting. However, it's not all roses, as will become clear next.

5.3. Merging SP-Forgetting and Relativized Forgetting

We have shown that $\mathsf{F_R}$, i.e., Relativized Forgetting, is a better alternative than $\mathsf{F_{SP}}$ if our objective is to approximate the set of answer sets modulo the forgotten atoms, but not introduce new answer sets. However, $\mathsf{F_R}$ has a drawback: there are cases where it is possible to forget while satisfying **(SP)**, but the result for any $\mathsf{f} \in \mathsf{F_R}$ does not coincide with the desired result (obtainable with operators from $\mathsf{F_{SP}}$).

Example 26. *Consider the following program P and that we want to forget about p from P:*

$$a \leftarrow p \qquad\qquad p \leftarrow not\ not\ p$$

It is easy to check that $\langle P, V\rangle$ does not satisfy Ω, i.e., it is possible to forget about V from P while satisfying **(SP)**. *The result returned by every operator in $\mathsf{F_{SP}}$ is strongly equivalent to $\{a \leftarrow not\ not\ a\}$. However, $\mathsf{f}(P,V)$ for every $\mathsf{f} \in \mathsf{F_R}$ is strongly equivalent to the empty program.*

The difference between $\mathsf{F_{SP}}$ and $\mathsf{F_R}$, as shown in Thm. 10, lies in the usage of intersection and union in their respective definitions. The key point is that whenever $\mathcal{R}^Y_{\langle P,V\rangle}$ has more than one element, even if there is a least one, union and intersection will not coincide. Taking this idea into account, we define a class of operators that aims at combining the delineated positive aspects of both $\mathsf{F_{SP}}$ and $\mathsf{F_R}$.

Definition 8 ($\mathbf{SP}_M$-Forgetting). *The class of forgetting operators $\mathsf{F_M}$ is defined by the following set:*

$$\{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X,Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \text{ and } X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}, \text{ if } \mathcal{R}^Y_{\langle P,V\rangle} \text{ has no least element, or } X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}, \text{ otherwise}\} \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}.$$

Whenever $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element, then $\mathsf{F_M}$ employs the intersection, whose result is precisely the least element, similar to $\mathsf{F_{SP}}$ and does therefore coincide with the desired ideal solution in this case, and whenever there is no least element it uses the union instead, just like $\mathsf{F_R}$.

Example 27. *Consider the program of Ex. 21. The result of forgetting about p from that program, for every $\mathsf{f} \in \mathsf{F_M}$, is strongly equivalent with that given in Ex. 22 for every $\mathsf{f}' \in \mathsf{F_R}$. On the other hand, for the program given in Ex. 26, the result of forgetting about p from that program, for every $\mathsf{f} \in \mathsf{F_M}$, is strongly equivalent to $\{a \leftarrow not\, not\, a\}$, and the same also holds for every operator in $\mathsf{F_{SP}}$.*

Note that due to the nature of the definition of $\mathsf{F_M}$, it does not contain operators over $\mathcal{C}_n$ or $\mathcal{C}_d$ either. Furthermore, a concrete operator of the class $\mathsf{F_M}$ can be defined using the very same approach as for the two classes of forgetting operators previously considered.

We also provide an algorithm, Alg. 3, that computes an actual forgetting result for $\mathsf{F_M}$, which we denote $\mathsf{f_M}$. Its definition follows the case-based definition of the class $\mathsf{F_M}$. More precisely, after initializing the resulting program P' and fixing its signature $\mathcal{A}' = \mathcal{A}\backslash V$, we determine for each $Y \subseteq \mathcal{A}'$ whether $\langle Y,Y\rangle$ itself gives rise to a countermodel and add the corresponding rule if this the case. If not, one of two cases of the definition of $\mathsf{F_M}$ is applied, adding further rules to the resulting program in a similar fashion as done before in Algs. 2 and 3, respectively.

We can again show that Alg. 3 does terminate and that the HT-models of the result correspond to the semantic characterization of $\mathsf{F_M}$.

Theorem 11. *Alg. 3 terminates and $\mathsf{f_M} \in \mathsf{F_M}$.*

Also, note that the case distinction in its definition is done for each $\mathcal{R}^Y_{\langle P,V\rangle}$ individually, so there are programs where the forgetting result for every $\mathsf{f} \in \mathsf{F_M}$ is neither strongly equivalent to that of any $\mathsf{f} \in \mathsf{F_{SP}}$ nor of any $\mathsf{f} \in \mathsf{F_R}$.

Algorithm 3: Computing $\mathsf{f}_{\mathsf{M}}(P, V)$ for forgetting about V from P

Input : Program P and $V \subseteq \mathcal{A}$
Output: Program $P' = \mathsf{f}_{\mathsf{M}}(P, V)$

$P' := \emptyset$;
$\mathcal{A}' := \mathcal{A} \backslash V$;
for $Y \subseteq \mathcal{A}'$ **do**
  if $Rel^{Y}_{\langle P,V \rangle} = \emptyset$ **then**
    $P' := P' \cup \{r^{\mathcal{A}'}_{Y,Y}\}$;
  end
  else if $\mathcal{R}^{Y}_{\langle P,V \rangle}$ *has no least element* **then**
    for $X \subset Y$ *s.t.* $\langle X, Y \rangle \notin \mathcal{HT}_V(P)_{\|V}$ **do**
      $P' := P' \cup \{r^{\mathcal{A}'}_{X,Y}\}$;
    end
  end
  else
    for $X \subseteq Y$ *s.t.* $X \notin \bigcap \mathcal{R}^{Y}_{\langle P,V \rangle}$ **do**
      $P' := P' \cup \{r^{\mathcal{A}'}_{X,Y}\}$;
    end
  end
end

Example 28. *Consider a program P whose HT-models are the following:*

$$\langle \emptyset, a \rangle \quad \langle a, a \rangle \quad \langle ap, ap \rangle \quad \langle b, ab \rangle \quad \langle ab, ab \rangle \quad \langle ap, abp \rangle \quad \langle abp, abp \rangle$$

In this case, the set $\mathcal{R}^{\{a\}}_{\langle P,V \rangle} = \{\{\{a\}, \{\}\}, \{\{a\}\}\}$ has a least element, $\{\{a\}\}$, whereas $\mathcal{R}^{\{a,b\}}_{\langle P,V \rangle} = \{\{\{a\}, \{a, b\}\}, \{\{b\}, \{a, b\}\}\}$ has no least element. The result of forgetting $\{p\}$ from P according to f_{M}, or any other operator in F_{M}, has the following four HT-models:

$$\langle a, a \rangle \qquad \langle a, ab \rangle \qquad \langle b, ab \rangle \qquad \langle ab, ab \rangle$$

The result of forgetting $\{p\}$ from P according to F_{R} has the following five HT-models:

$$\langle \emptyset, a \rangle \qquad \langle a, a \rangle \qquad \langle b, ab \rangle \qquad \langle a, ab \rangle \qquad \langle ab, ab \rangle$$

Finally, the result of forgetting $\{p\}$ from P according to F_{SP} has the following two HT-models:

$$\langle a, a \rangle \qquad \langle ab, ab \rangle$$

For this forgetting instance all three classes provide different results of forgetting that are not strongly equivalent. In fact, even in terms of answer sets, these three

classes yield different results. The result of forgetting $\{p\}$ *from* P *according to* $\mathsf{F_R}$ *has no answer sets, according to* $\mathsf{F_M}$ *one answer set,* $\{a\}$*, and according to* $\mathsf{F_{SP}}$ *two answer sets, namely* $\{a\}$ *and* $\{a, b\}$*.*

Expectedly, if we consider only Horn programs, then this definition of $\mathsf{F_M}$ coincides with both its constituents.

Proposition 21. *Let* $P \in \mathcal{C}_H$ *and* $V \subseteq \mathcal{A}$*. Then, for every* $\mathsf{f} \in (\mathsf{F_{SP}} \cup \mathsf{F_R})$ *and* $\mathsf{f}' \in \mathsf{F_M}$ *we have that* $\mathsf{f}(P, V) \equiv \mathsf{f}'(P, V)$*.*

Moreover, unlike $\mathsf{F_R}$, we are able to show that $\mathsf{F_M}$ coincides with $\mathsf{F_{SP}}$ whenever it is possible to forget.

Proposition 22. *Let* P *be a program and* $V \subseteq \mathcal{A}$*, such that* $\langle P, V\rangle$ *does not satisfy* Ω*. Then, for every* $\mathsf{f} \in \mathsf{F_{SP}}$ *and* $\mathsf{f}' \in \mathsf{F_M}$ *we have that* $\mathsf{f}(P, V) \equiv \mathsf{f}'(P, V)$*.*

This is so because in the case of Horn programs, the least element exists for every $\mathcal{R}^Y_{\langle P,V\rangle}$, so the intersection is always used.

The particular definition of $\mathsf{F_M}$ ensures that it satisfies a set of properties that is different from that of its predecessors.

Proposition 23. $\mathsf{F_M}$ *satisfies* $(\mathbf{E}_{\mathcal{C}_H})$*,* $(\mathbf{E}_{\mathcal{C}_e})$*,* **(wE)***,* **(SE)***,* **(PP)***,* **(sC)***,* **(wC)***,* **(CP)***, but not* $(\mathbf{E}_{\mathcal{C}_n})$*,* $(\mathbf{E}_{\mathcal{C}_d})$*,* **(W)***, and* **(SI)***.*

Contrary to $\mathsf{F_{SP}}$ and $\mathsf{F_R}$, the class $\mathsf{F_M}$ satisfies both **(wC)** and **(sC)**, and consequently **(CP)**. Therefore, the result of forgetting according to $\mathsf{F_M}$ preserves the answer sets of P, but, unlike the other two, no longer satisfies **(SI)**.

In fact, the answer sets are no longer preserved if a (non-empty) program over $\mathcal{A}\backslash V$ is added to P. To capture this in a more precise way, we introduce generalizations of **(wC)** and **(sC)**, which correspond to the two inclusions of **(SP)**.

(wSP) F satisfies *weakened Strong Persistence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(P \cup R)_{\|V} \subseteq \mathcal{AS}(\mathsf{f}(P, V) \cup R)$, for all $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$.

(sSP) F satisfies *strengthened Strong Persistence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathcal{AS}(\mathsf{f}(P, V) \cup R) \subseteq \mathcal{AS}(P \cup R)_{\|V}$, for all $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$.

Property **(wSP)** guarantees that all answer sets of P are preserved when forgetting, no matter which rules R over $\mathcal{A}\backslash V$ are added to P, but, for some such R, does not prevent that the result of forgetting has more answer sets than P. Vice versa, **(sSP)** does not guarantee the preservation of all answer sets of P for some added R over $\mathcal{A}\backslash V$, but it ensures that all answer sets of the result of forgetting indeed correspond to answer sets of P, independently of the added rules R.

We can show that each of the three considered classes of forgetting operators only satisfies one of the two properties.

Theorem 12. $\mathsf{F_{SP}}$ *satisfies* **(wSP)**, *whereas* $\mathsf{F_R}$ *and* $\mathsf{F_M}$ *satisfy* **(sSP)**.

Since there is no nonempty class of forgetting operators that satisfies **(SP)**(cf. Thm. 1), it follows from this result that $\mathsf{F_{SP}}$ does not satisfy **(sSP)**, and that $\mathsf{F_R}$ and $\mathsf{F_M}$ do not satisfy **(wSP)**. Thus, even though $\mathsf{F_R}$ satisfies **(wC)**, i.e., **(wSP)** for an empty R, it does not for arbitrary R's. Although both $\mathsf{F_R}$ and $\mathsf{F_M}$ satisfy **(sSP)**, there is an inclusion in terms of HT-models.

Lemma 13. *Let* $\mathsf{f} \in \mathsf{F_R}$, $\mathsf{f}' \in \mathsf{F_M}$, $P \in \mathcal{C}(\mathsf{f}) \cap \mathcal{C}(\mathsf{f}')$, *and* $V \subseteq \mathcal{A}\backslash V$. *Then,*

$$\mathcal{HT}(\mathsf{f}'(P,V)) \subseteq \mathcal{HT}(\mathsf{f}(P,V)).$$

This means that $\mathsf{F_M}$ indeed provides a better approximation in terms of property **(SP)**, as the following result states.

Proposition 24. *Let* $P \in \mathcal{C}_e$ *be a program,* $V \subseteq \mathcal{A}$, $\mathsf{f} \in \mathsf{F_R}$, *and* $\mathsf{f}' \in \mathsf{F_M}$ *with* $\mathcal{C}(\mathsf{f}) = \mathcal{C}(\mathsf{f}') = \mathcal{C}_e$. *Then, for every* $R \in \mathcal{C}_e$ *with* $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$,

$$\mathcal{AS}(\mathsf{f}(P,V) \cup R) \subseteq \mathcal{AS}(\mathsf{f}'(P,V) \cup R).$$

Note that the result is stated for the general class of programs only: for Horn programs the classes coincide (recall Prop. 21), and for normal and disjunctive programs no operators exist in the considered classes of forgetting operators.

Clearly, a class F satisfies **(SP)** iff it satisfies **(wSP)** and **(sSP)**. Since no F can in general satisfy **(SP)**, we basically obtain two kinds of relaxations on the conditions of **(SP)**. But we can do even better: following results from [49], we know that F satisfies **(SP)** iff it satisfies **(wC)**, **(sC)**, and **(SI)**. From the results in Props. 11, 20, and 23, we obtain that each of the three discussed classes corresponds to a unique relaxation of the conditions of **(SP)**: class $\mathsf{F_{SP}}$ satisfies **(wC)** and **(SI)**, class $\mathsf{F_R}$ satisfies **(sC)** and **(SI)**, and class $\mathsf{F_M}$ satisfies **(wC)** and **(sC)**. This implies that our study gives a complete account on which forgetting operators to use when **(SP)** cannot be satisfied, but only approximated.

Arguably, $\mathsf{F_M}$ is also more flexible in situations where we have to forget several atoms for which $\mathsf{F_{SP}}$ and $\mathsf{F_R}$ do not provide the optimal overall choice.

Example 29. *Consider the following program* P *from which we want to forget about* c *and* p.

$$d \leftarrow c \qquad c \leftarrow not\ not\ c \qquad a \leftarrow p \qquad b \leftarrow not\ p \qquad p \leftarrow not\ not\ p$$

Clearly, $\mathsf{F_{SP}}$ *allows us to correctly capture the result of forgetting about* c, *in the sense that* $d \leftarrow not\ not\ d$ *is part of the result of forgetting, but, at the same time, will introduce new answer sets in which both* a *and* b *are true. On the other hand,* $\mathsf{F_R}$ *will avoid the latter problem, but will simply cancel all rules mentioning* d *and* c. *Here,* $\mathsf{F_M}$ *certainly provides the best alternative as it avoids both problems and provides the desired result.*

The choice between the three classes greatly depends on the application at hand. To help making this decision, we now identify, for each of the three classes, a set of conditions in favor of its choice over the other two.

The class F_{SP} should be chosen whenever:

- **(SP)** should hold for those instances that do not satisfy Ω;
- Rules that do not mention atoms to be forgotten should be preserved;
- All answer sets should be preserved; and
- We do not mind the appearance of new answer sets.

The class F_{R} should be chosen whenever:

- Rules that do not mention atoms to be forgotten should be preserved;
- No new answer sets should appear;
- We do not mind that some answer sets may disappear; and
- We do not mind that **(SP)** does not hold even if Ω does not hold.

The class F_{M} should be chosen whenever:

- **(SP)** should hold for those instances that do not satisfy Ω;
- Answer sets should be preserved precisely (modulo the forgotten atoms); and
- We do not mind to change rules that do not mention atoms to be forgotten.

These conditions directly stem from the properties each of the classes of forgetting operators satisfies, and can be seen as a guideline for a more informed choice between the three alternatives.

6. Complexity

In this section, we study the computational complexity of several important decision problems, most notably the question of whether a certain program is the forgetting result of one of the classes we have introduced here, as well as determining whether Ω holds, which allows us to decide whether forgetting is possible and preserve **(SP)**.

We assume familiarity with standard complexity concepts, such as **NP**. Given a complexity class **C**, a **C** oracle decides a given sub-problem from **C** in one computation step. The class $\mathbf{\Sigma_k^P}$ contains the problems that can be decided in polynomial time by a non-deterministic Turing machine with unrestricted access to a $\mathbf{\Sigma_{k-1}^P}$ oracle. $\mathbf{\Pi_k^P}$ is the complementary class of $\mathbf{\Sigma_k^P}$. Thus, $\mathbf{\Sigma_1^P} = \mathbf{NP}$, and $\mathbf{\Pi_1^P} = \mathbf{coNP}$. We also recall that a language is in complexity class $\mathbf{D_i^P}$ iff it is the intersection of a language in $\mathbf{\Sigma_i^P}$ and a language in $\mathbf{\Pi_i^P}$. Instead of $\mathbf{D_1^P}$ we use the more common name $\mathbf{D^P}$.

We start by considering the decision problems related to relativized equivalence, building on the following already known result which will be useful due to the established correspondence between HT-and V-HT-models in Prop. 16.

Proposition 25 ([50],Theorem 6.12.). *Given a program P, an HT-interpretation $\langle X, Y\rangle$, and $V \subseteq \mathcal{A}$, deciding whether $\langle X, Y\rangle \in \mathcal{HT}_V(P)$ is $\mathbf{D^P}$-complete.*

Our first result is in the spirit of model-checking.

Proposition 26. *Given program P, $V \subseteq \mathcal{A}$, and HT-interpretation $\langle X, Y\rangle$, deciding whether $\langle X, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$ is $\mathbf{\Sigma_2^P}$-complete. Hardness holds already for disjunctive programs.*

Membership follows from guessing an interpretation $Y' \sim_V Y$ and checking $(X, Y') \in \mathcal{HT}_V(P)$ (cf. Prop. 25), while the hardness result can be adapted from the $\mathbf{\Sigma_2^P}$-hardness of ASP consistency (cf. [60]). By means of this, we can determine the complexity of deciding whether a given program is strongly equivalent to the result of forgetting obtained by every $\mathsf{f} \in \mathsf{F_R}$.

Theorem 13. *Given programs P, Q, and $V \subseteq \mathcal{A}$, deciding whether $P \equiv \mathsf{f}(Q, V)$ (for $\mathsf{f} \in \mathsf{F_R}$) is $\mathbf{\Pi_3^P}$-complete. Hardness holds already for disjunctive programs.*

Essentially, for the complementary problem, we guess an HT-interpretation $\langle X, Y\rangle$ and check that either $(X, Y) \in \mathcal{HT}(P)$ or $(X, Y) \in \mathcal{HT}_V(P)_{\|V}$, but not both. The hardness result is then obtained by a reduction from $(3, \forall)$-QSAT.

The next result provides the complexity of determining whether some X occurs in the intersection of $\mathcal{R}^Y_{\langle P,V\rangle}$ used in the definition of $\mathsf{F_{SP}}$, $\mathsf{F_M}$ and Ω.

Proposition 27. *Given program P, $V \subseteq \mathcal{A}$, and HT-interpretation $\langle X, Y\rangle$ with $Y \subseteq \mathcal{A}\backslash V$, deciding whether $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ is in $\mathbf{D_2^P}$.*

Basically, we have to perform a $\mathbf{\Sigma_2^P}$- and a $\mathbf{\Pi_2^P}$-test. The former decides whether $\mathcal{R}^Y_{\langle P,V\rangle} \neq \emptyset$, while the latter determines that for all $A \subseteq V$, either $\langle Y \cup A, Y \cup A\rangle \notin \mathcal{HT}_V(P)$ or $\langle X, Y \cup A\rangle \in \mathcal{HT}_V(P)$.

This Lemma allows us to obtain an identical result to Thm. 13 for $\mathsf{F_{SP}}$.

Theorem 14. *Given programs P, Q, and $V \subseteq \mathcal{A}$, deciding whether $P \equiv \mathsf{f}(Q, V)$ (for $\mathsf{f} \in \mathsf{F_{SP}}$) is $\mathbf{\Pi_3^P}$-complete. Hardness holds already for disjunctive programs.*

The basic proof idea is very similar to the one sketched for Thm. 13, but substituting the test $(X, Y) \in \mathcal{HT}_V(P)_{\|V}$ with $(X, Y) \in \mathcal{HT}(\mathsf{f}(P, V))$ for $\mathsf{f} \in \mathsf{F_{SP}}$.

Since the definition of $\mathsf{F_M}$ is based on cases, deciding whether its condition holds is computationally more expensive than the previous two (in Lemmas 26 and 27).

Proposition 28. *Given program P, $V \subseteq \mathcal{A}$, and HT-interpretation $\langle X, Y\rangle$ with $Y \subseteq \mathcal{A}\backslash V$, deciding whether $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$ if $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element, and $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ otherwise, is in $\mathbf{\Sigma_3^P}$ and in $\mathbf{\Pi_3^P}$.*

Fortunately, though, since this test is both in $\mathbf{\Sigma_3^P}$ and in $\mathbf{\Pi_3^P}$, in the next result, we can basically solve the complementary problem of guessing an HT-interpretation $\langle X, Y\rangle$ and check that either $(X, Y) \in \mathcal{HT}(P)$ or $(X, Y) \in \mathcal{HT}(\mathsf{f}(P, V))$ for $\mathsf{f} \in \mathsf{F_M}$, but not both, in one step.

Theorem 15. *Given programs P, Q, and $V \subseteq \mathcal{A}$, deciding whether $P \equiv \mathsf{f}(Q, V)$ (for $\mathsf{f} \in \mathsf{F_M}$) is $\mathbf{\Pi_3^P}$-complete. Hardness holds already for disjunctive programs.*

Thus, determining whether $P \equiv \mathsf{f}(Q, V)$ for f of every of the three considered classes of forgetting operators is always $\mathbf{\Pi_3^P}$-complete. This shows that the choice of which of the three classes of forgetting operators to use in a concrete situation is not influenced by their computational complexity.

Finally, we also provide the complexity result for criterion Ω.

Theorem 16. *Let P be a program over $\mathcal{A}$ and $V \subseteq \mathcal{A}$. Deciding whether $\langle P, V\rangle$ satisfies criterion Ω is $\mathbf{\Sigma_3^P}$-complete. Hardness holds already for disjunctive programs.*

Thus, determining whether Ω holds is not on a lower level in the polynomial hierarchy than determining whether a given program is strongly equivalent to the result of forgetting for every of the three considered classes. Checking Ω is still of major importance, be it for determining whether we can forget without losses in case not forgetting is an option, or for helping making a decision on which class of operators to be used according to the indications given at the end of the previous section.

7. Related Work

In this section, we discuss other approaches for forgetting in ASP previously considered in the literature and discuss their relation to the material presented so far, occasionally reusing notation and results introduced/compiled in [49].

To begin with, there are further classes of forgetting operators that, like $\mathsf{F_{SP}}$, $\mathsf{F_R}$, and $\mathsf{F_M}$, are defined through a characterization of the set of HT-models of the result of forgetting based on the HT-models of the original program.

Namely, Wang et al. introduced HT-forgetting [43, 46] and the corresponding class $\mathsf{F_{HT}}$ of forgetting operators, which is defined for extended programs, characterizing the set of HT-models of the result of forgetting as being composed of the HT-models of the original program, modulo any occurrence of the atoms to be forgotten. This class was shown to be closed for the most general class ($\mathcal{C}_e$) and Horn programs ($\mathcal{C}_H$), but not for disjunctive ($\mathcal{C}_d$) or normal programs ($\mathcal{C}_n$. Although $\mathsf{F_{HT}}$ satisfies **(SI)**, it does not satisfy **(CP)**. In fact, $\mathsf{F_{HT}}$ does

not even satisfy either of **(sC)** and **(wC)**. This immediately implies that it does not satisfy neither **(sSP)** nor **(wSP)**, the two new properties we introduced to further clarify the proximity to a class satisfying **(SP)**. Therefore, though the semantic definition of $\mathsf{F_{HT}}$ is quite similar to $\mathsf{F_R}$ in principle, the fact that the latter utilizes V-HT-models instead of HT-models (and thus, e.g., omits HT-models that can never result in an answer set, no matter which program R we add over the remaining language) makes a huge difference. This is also reflected by the properties that are satisfied by $\mathsf{F_R}$ and not by $\mathsf{F_{HT}}$, namely **(sC)** and **(sSP)**, i.e., $\mathsf{F_R}$ never ends up introducing new answer sets to the result of forgetting. Arguably, to some extent, this may be due to the fact that $\mathsf{F_{HT}}$ drew inspiration from forgetting in (monotonic) modal logic S5 [47].

Starting from the observation that HT-forgetting does not satisfy **(CP)**, Wang et al. introduced SM-Forgetting [44] and the class $\mathsf{F_{SM}}$ of forgetting operators was defined with the aim of preserving the answer sets of the original program (modulo the forgotten atoms). Like HT-Forgetting, this class is defined for extended programs through a characterization of the HT-models of the result of forgetting. Namely, maximal subsets of the HT-models of the original program are taken, modulo any occurrence of the atoms to be forgotten, such that the set of their answer sets coincides with the set of answer sets of the original program, modulo the forgotten atoms. It is therefore no surprise that $\mathsf{F_{SM}}$ satisfies **(CP)**. Moreover, like $\mathsf{F_{HT}}$, this class is shown to be closed for $\mathcal{C}_e$ and $\mathcal{C}_H$, but not for $\mathcal{C}_d$ nor $\mathcal{C}_n$. However, $\mathsf{F_{SM}}$ does not satisfy **(SI)**. Further inspection shows that, although $\mathsf{F_{SM}}$ satisfies **(sC)** and **(wC)** (a consequence of satisfying **(CP)**), it does not satisfy any of **(sSP)** and **(wSP)**.

Example 30. *For* **(sSP)**, *consider program* P*:*

$$a \leftarrow p \qquad\qquad \bot \leftarrow not\ a \qquad\qquad \bot \leftarrow not\ p$$

with $\mathcal{HT}(P) = \{\langle ap, ap\rangle, \langle a, ap\rangle, \langle \emptyset, ap\rangle\}$*. Then, for every* $\mathsf{f} \in \mathsf{F_{SM}}$*, we have that* $\mathcal{HT}(\mathsf{f}(P, \{p\})) = \{\langle a, a\rangle, \langle \emptyset, a\rangle\}$*. Clearly,* $\mathcal{AS}(\mathsf{f}(P, \{p\})) = \emptyset = \mathcal{AS}(P)_{\|\{p\}}$*, but if we add* $R = \{a \leftarrow\}$ *to both* P *and* $\mathsf{f}(P, \{p\})$*, we obtain with* $\mathcal{AS}(\mathsf{f}(P, \{p\}) \cup R) = \{\{a\}\}$*, whereas* $\mathcal{AS}(P \cup R)_{\|\{p\}} = \emptyset$*.*

To show that $\mathsf{F_{SM}}$ *does not satisfy* **(wSP)**, *consider the program* P *of Ex. 11, whose HT-models are given in Ex. 21. In this case, for every* $\mathsf{f} \in \mathsf{F_{SM}}$*, we have that* $\mathcal{HT}(\mathsf{f}(P, \{p\})) = \{\langle a, a\rangle, \langle b, b\rangle, \langle ab, ab\rangle, \langle a, ab\rangle, \langle b, ab\rangle\}$*. Though* $\mathcal{AS}(\mathsf{f}(P, \{p\})) = \{\{a\}, \{b\}\} = \mathcal{AS}(P)_{\|\{p\}}$*, if we add* $R = \{a \leftarrow\}$ *to both* P *and* $\mathsf{f}(P, \{p\})$*, we obtain* $\mathcal{AS}(\mathsf{f}(P, \{p\}) \cup R) = \{\{a\}\}$*, whereas* $\mathcal{AS}(P \cup R)_{\|\{p\}} = \{\{a\}, \{a, b\}\}$*.*

Therefore, contrarily to the three classes here defined, $\mathsf{F_{SP}}$, $\mathsf{F_R}$ and $\mathsf{F_M}$, the class $\mathsf{F_{SM}}$ does not satisfy any of the inclusions of **(SP)**, i.e., forgetting operators in $\mathsf{F_{SM}}$ may end up removing or adding answer sets to the result of forgetting depending on the chosen program R added to the result of forgetting. Hence, though $\mathsf{F_{SM}}$ is in principle close to $\mathsf{F_M}$ in terms of satisfied properties (both satisfy **(sC)** and **(wC)** and not **(SI)**), it does not achieve the preservation of answer sets while forgetting in general, even if, unlike $\mathsf{F_{HT}}$, it does indeed focus on preserving answer sets. The results in terms of computational complexity

mirror this: the decision problem equivalent to Thms. 13, 14, and 15 is $\mathbf{\Pi_1^P}$-complete for $\mathsf{F_{HT}}$ and $\mathbf{\Pi_2^P}$-complete for $\mathsf{F_{SM}}$. Hence, forgetting and preserving (or approximating the preservation of) all semantic dependencies between the atoms not to be forgotten is a complicated problem, and preserving more dependencies increases the computational complexity.

Wong defined two further classes of forgetting operators that build on HT-models, more precisely on HT-consequence, namely Semantic Strong Forgetting (F_S) and Semantic Weak Forgetting (F_W) [42], which are defined for disjunctive programs.

In the case of F_S, the basic idea is to take the set of rules HT-entailed by the original program excluding those that mention atoms to be forgotten. This class is closed for disjunctive programs but not for normal ones, and it was shown that F_S does not satisfy **(sC)**, **(wC)**, nor **(SI)**, which immediately implies that it also does not satisfy **(sSP)** nor **(wSP)**. In some way, F_S can be seen as an approximation of $\mathsf{F_{HT}}$ defined only for disjunctive programs (which is why **(SI)** does not hold for F_S). Independently, Delgrande and Wang introduced SE-Forgetting (F_{SE}) [18], which is defined for disjunctive programs building on an inference system [61] that preserves strong equivalence. Although the underlying definitions differ, SE-Forgetting in fact coincides with Semantic Strong Forgetting. More recently, this approach based on consequence has also been generalized to a wider setting of logics by Delgrande [62], who investigated forgetting as an abstract belief change operator, independent of any specific formal system and syntax.

In the case of F_W, we also take the set of rules HT-entailed by the original program, but instead of removing all the rules that mention atoms to be forgotten, some rules are kept, just omitting the forgotten atoms themselves. While the approach satisfies **(SI)** and **(sC)**, which would eventually imply proximity to $\mathsf{F_R}$, it was also shown that operators in this class, contrarily to most classes of operators in the literature, do not give the expected results even for very simple Horn programs.

With the aim of preserving both the answer sets of the original program, and also those of the original program augmented by any set of rules over the signature without the atoms to be forgotten, Knorr and Alferes [45] introduced Strong AS-Forgetting. Although the defined class F_{Sas} indeed satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ for those forgetting instances for which it is defined, this class can only be used in a very restricted setting. This is witnessed by the fact that the only known operator in F_{Sas}, dubbed f_{Sas}, is only defined for a non-standard class of programs (permitting double negation but no disjunctions), and can only be applied to forget about single atoms p if a sufficient (but not necessary) criterion, called p-forgettable (see [45]), holds, which is considerably stronger than Ω, hence unnecessarily excluding many possible cases. Moreover, this operator cannot in general be iterated, since the result of forgetting an atom p from a p-forgettable program may well end up being a program that is not q-forgettable for some other atom q to be forgotten. For the sake of completeness, we formally relate such operator and the class $\mathsf{F_{SP}}$.

Corollary 2. *Let f_{Sas} be the operator defined in [45] for F_{Sas} and P a program for which f_{Sas} is defined. For every $\mathsf{f} \in \mathsf{F_{SP}}$, if a single atom p is p-forgettable from P, then $\mathcal{HT}(\mathsf{f}(P, \{p\})) = \mathcal{HT}(\mathsf{f}_{Sas}(P, \{p\}))$.*

Eiter and Wang [16] proposed Semantic Forgetting, F_{sem}, with the basic idea of characterizing the result of forgetting just by its answer sets. These are obtained by considering only the minimal sets among the answer sets of the initial program ignoring the atoms to be forgotten. Three concrete algorithms are presented, two based on semantic considerations and one syntactic. This class F_{sem} is closed for all classes of programs for which it is defined (up to disjunctive), but only satisfies **(sC)** and **(wE)** among the properties presented in Sec. 2. This is ultimately not surprising, since answer sets do not contain all the information present in a program, which is why many properties related to strong equivalence cannot be satisfied.

A further approach by Zhang and Foo [15] introduced two syntactic operators for normal logic programs, termed Strong and Weak Forgetting. Both operators start with computing a reduction corresponding to the well-known weak partial evaluation (WGPPE) [48]. Then, in Strong Forgetting, all rules containing the atom to be forgotten are simply removed, while in Weak Forgetting, first, negative occurrences of the atom to be forgotten are omitted, and only then rules (still) containing the atom to be forgotten are omitted. Both operators are closed for $\mathcal{C}_n$. Although these operators satisfy **(SI)**, they do not satisfy **(CP)**, nor any of the inclusions **(sC)** and **(wC)**, which implies that they also do not satisfy **(sSP)** and **(wSP)**, and due to their syntactic nature, are rather orthogonal to the preservation of answer sets (or even dependencies between atoms to be forgotten).

Finally, also with the aim of preserving Strong Persistence, although with an alternative approach, Aguado et al. [63, 64] propose a solution based on the extension of the syntax of programs by a new connective, called fork. This has the benefit that preserving Strong Persistence becomes always possible. Such solution, however, assumes that one is willing to trade the elimination of atoms by the addition of a new syntactic construct.

We end this section with an interesting connection with variable elimination in propositional logic. First, we formally recall a well-known result, namely that when we have choice rules of the form $a \leftarrow \mathit{not}\, \mathit{not}\, a$ for every atom a in the signature, the answer sets of a program coincide with the classical models of the program. Recall that a classical model of a program is a set $I \subseteq \mathcal{A}$ such that $I \models P$. We denote by $Mod(P)$ the set of classical models of program P. We define the *completion* of a program P over $\mathcal{A}$ as the program $P^* = P \cup P_{\mathcal{A}}$, where $P_{\mathcal{A}} = \{a \leftarrow \mathit{not}\, \mathit{not}\, a \mid a \in \mathcal{A}\}$.

Proposition 29. *Let P be a program over $\mathcal{A}$. Then,*

$$Mod(P) = \mathcal{AS}(P^*)$$

Example 31. *Let $P = \{a \leftarrow \mathit{not}\, b; b \leftarrow \mathit{not}\, a\}$ be a program over $\mathcal{A} = \{a, b\}$, and its completion $P^* = \{a \leftarrow \mathit{not}\, b;\ b \leftarrow \mathit{not}\, a;\ a \leftarrow \mathit{not}\, \mathit{not}\, a;\ b \leftarrow \mathit{not}\, \mathit{not}\, b\}$.*

In this case, P has three classical models $\{a\}, \{b\}$ and $\{a, b\}$, which coincide with the answer sets of P^*. Note that the two choice rules added to P make $\{a, b\}$ an answer set of P^*, whereas this set is not an answer set of P.

Interestingly, we can always forget a set of atoms from the completion of program while satisfying **(SP)**.

Proposition 30. *Let P be a program over $\mathcal{A}$. Then, for every $V \subseteq \mathcal{A}$, we have that $\langle P^*, V\rangle$ does not satisfy Ω.*

Since the models of a program coincide with the answer sets of its completion, a question that naturally arises is whether the result of forgetting according to the class $\mathsf{F_{SP}}$ coincides with classical forgetting, also known as variable elimination. In order to be able to draw such connection, we briefly recall the semantical characterization of variable elimination. Given a program P over $\mathcal{A}$ and $V \subseteq \mathcal{A}$, classical variable elimination of V from P, denoted by $\mathsf{f}_{CL}(P, V)$, is semantically characterized (over the signature $\mathcal{A}\backslash V$) as $Mod(\mathsf{f}_{CL}(P, V)) = Mod(P)_{\|V}$. We can now present the connection between forgetting in ASP and variable elimination.

Proposition 31. *Let P be a program over $\mathcal{A}$ and $V \subseteq \mathcal{A}$. Then, for every $\mathsf{f} \in \mathsf{F_{SP}}$, we have that the following conditions hold*

- $\mathcal{HT}(\mathsf{f}(P^*, V)) = \{\langle T, T\rangle \mid T \in Mod(\mathsf{f}_{CL}(P, V))\}$
- $\mathcal{AS}(\mathsf{f}(P^*, V)) = Mod(\mathsf{f}_{CL}(P, V))$.

This result shows that the result of forgetting a set of variables from the completion of any program can be completely characterized using the semantic characterization of variable elimination applied to the original program.

Example 32. *Recall program P from Ex. 21 with $V = \{p\}$. This program has four classical models, namely $\{a, p\}$, $\{b\}$, $\{a, b\}$, and $\{a, b, p\}$. The completion P^* of P is composed of the following rules:*

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

*Whereas the instance $\langle P, \{p\}\rangle$ satisfies Ω, the same does not hold for the completion of P, i.e., $\langle P^*, \{p\}\rangle$ does not satisfy Ω. It is therefore possible to forget p from P^* and satisfy **(SP)**. The resulting program is strongly equivalent to*

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

and therefore has the HT-models

$$\langle a, a\rangle \qquad\qquad \langle b, b\rangle \qquad\qquad \langle ab, ab\rangle$$

each corresponding to a classical model of P modulo p. The corresponding answer sets of P^* are $\{a\}$, $\{b\}$, and $\{a, b\}$.

A result similar to Prop. 31 can be shown for the classes $\mathsf{F_R}$ and $\mathsf{F_M}$, which further strengthens the idea that these three classes of operators are indeed closely related.

8. Conclusions

We have studied forgetting in ASP, focusing on what is perhaps its most crucial property – *strong persistence* **(SP)** – which captures the essence of forgetting in ASP by ensuring that all semantic relations between the atoms not forgotten are preserved.

We began by answering an important open question, showing that it is not always possible to forget a set of atoms while obeying **(SP)**.

Departing from this impossibility result, we conducted a thorough study of the limits of forgetting in ASP, including a necessary and sufficient criterion (Ω) to determine whether a particular set of atoms can be forgotten from a program while obeying **(SP)**. Whereas at a technical level, criterion Ω is closely tied to certain conditions on the HT-models of the program at hand, it seems that what cannot be forgotten from a program are atoms used in rules that are somehow equivalent to *choice rules* [65], and those atoms are pivotal in the sense that they play an active role in determining the truth of other atoms in some answer sets i.e., there are rules whose bodies mention these atoms and they are true at least in some answer sets. Further investigating this conjecture is an interesting line of future work.

We have also introduced a new class of operators that allows us to show how to forget a set of atoms from a given program while preserving **(SP)**, whenever that is possible. We also provided a general condition to determine all sets of atoms that can be forgotten from a given program, as well as special cases in which a set of atoms can always be forgotten, namely singleton sets in the case of normal and disjunctive programs.

The second part of this paper was devoted to investigating the problem of forgetting in ASP when we *must* forget, even if satisfying the fundamental desirable property **(SP)** is not possible. Three alternatives were pursued which, despite stemming from different starting points – one reusing a known class of forgetting operators, one exploring the concept of relativized equivalence, and one trying to get the best of the previous two – turn out to each correspond to the relaxation of one of three properties – **(wC)**, **(sC)** and **(SI)** – that together characterize **(SP)**. We characterized the three classes by showing which of the usually considered properties each obeys, established links between them, and showed that none of the other operators and classes of operators mentioned in the literature satisfy the properties satisfied by these three alternatives.

We also established relevant novel results concerning a correspondence between V-HT-models and HT-models and a full complexity result for checking whether the criterion (Ω) that indicates whether it is possible to forget while satisfying **(SP)** holds.

The computational complexity of the proposed operators turns out to be high, which is not surprising given, for example, the fact that in classical logic forgetting can only be expressed as a second-order axiom. Nevertheless, on the one hand, forgetting is an operation not expected to be done as regularly as for example model computation or query answering, while, on the other hand, at least for those classes that satisfy **(SI)**, F_{SP} and F_{R}, we can perform forgetting

in a modular way focusing only on the relevant part of the program. Whether this can be extended also to F_{M} remains an interesting open problem for future research.

Other avenues for future research include investigating different forms of forgetting which may be required in practice, such as those that preserve some aggregated meta-level information about the forgotten atoms, or even going beyond maintaining all relationships between non-forgotten atoms which may be required by certain legislation. Also, we may investigate the relation of Ω to projections of answer sets [57] in particular related to modules [66, 67], study concrete operators of forgetting in the line of [68], and investigate the limits of forgetting for semantics other than ASP, such as [46] based on the FLP-semantics [69], or [70, 45] based on the well-founded semantics [71].

Acknowledgments. We would like to thank the reviewers of this paper for their comments and suggestions. R. Gonçalves, M. Knorr and J. Leite were partially supported by FCT project FORGET (PTDC/CCI-INF/32219/2017) and by strategic project NOVA LINCS (UIDB/04516/2020). S. Woltran was supported by the Austrian Science Fund (FWF): Y698, P25521.

Appendix

This appendix contains the proofs for all results in the paper (with the exception of Thm. 1 whose proof appears in the main body).

Proofs for Sec. 2

Lemma 5. *Let $\mathcal{M}$ be a set of HT-interpretations. Then, (H_2) implies (H_1).*

Proof. Suppose that $\langle X, Y\rangle \in \mathcal{M}$, $\langle Y', Y'\rangle \in \mathcal{M}$ with $Y \subseteq Y'$, and that (H_2) holds. Then, $\langle X, X\rangle \in \mathcal{M}$ follows by (H_2) from $\langle X, Y\rangle \in \mathcal{M}$. Also, from $X \subseteq Y \subseteq Y'$, $\langle X, X\rangle \in \mathcal{M}$, and $\langle Y', Y'\rangle \in \mathcal{M}$, we have $\langle X, Y'\rangle \in \mathcal{M}$ by (H_2). ■

Proofs for Sec. 3

Proposition 1. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then $\langle P, V\rangle$ does not satisfy Ω.*

Proof. We need to show that for every $Y \subseteq \mathcal{A}\backslash V$, either $\mathcal{R}^Y_{\langle P,V\rangle}$ is empty or it has a least element. Let $Y \subseteq \mathcal{A}\backslash V$ such that $\mathcal{R}^Y_{\langle P,V\rangle}$ is not empty. We prove that $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element. First, since $\mathcal{R}^Y_{\langle P,V\rangle}$ is not empty, there exists $A \in Rel^Y_{\langle P,V\rangle}$. We now prove that $R^{Y,A}_{\langle P,V\rangle}$ is a least element of $\mathcal{R}^Y_{\langle P,V\rangle}$. For that, let $X \in R^{Y,A}_{\langle P,V\rangle}$ and $A' \in Rel^Y_{\langle P,V\rangle}$. We prove that $X \in R^{Y,A'}_{\langle P,V\rangle}$, thus showing that $R^{Y,A}_{\langle P,V\rangle}$ is a least element of $\mathcal{R}^Y_{\langle P,V\rangle}$. Since $X \in R^{Y,A}_{\langle P,V\rangle}$ we have that $\langle X \cup A'', Y \cup A\rangle \in \mathcal{HT}(P)$, for some $A'' \subseteq A$. Since $A' \in Rel_{\langle P,V\rangle}$ we have that $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$. Using condition (H_3) of Lemma 4, we can conclude that $\langle (X \cup A'') \cap (Y \cup A'), (Y \cup A) \cap (Y \cup A')\rangle \in \mathcal{HT}(P)$, i.e., $\langle X \cup (A'' \cap A'), Y \cup (A'' \cap A')\rangle \in \mathcal{HT}(P)$. Since $Y \cup (A'' \cap A') \subseteq Y \cup A'$ we can use condition (H_1) of Lemma 4 to conclude that $\langle X \cup (A'' \cap A'), Y \cup A'\rangle \in \mathcal{HT}(P)$. This then implies that $X \in R^{Y,A'}_{\langle P,V\rangle}$. ■

Lemma 6. *Let f be a forgetting operator over $\mathcal{C}$ that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ for an instance $\langle P, V\rangle$ over $\mathcal{C}$. If $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, then $Rel^Y_{\langle P,V\rangle} \neq \emptyset$.*

Proof. We show the contrapositive, i.e., if $Rel^Y_{\langle P,V\rangle} = \emptyset$, then $\langle Y, Y\rangle \notin \mathcal{HT}(\mathsf{f}(P, V))$. Suppose that $Rel^Y_{\langle P,V\rangle} = \emptyset$. By Def. 3, there are two cases to consider.

(i) There is no $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$. Thus, no matter which program R over $\mathcal{A}\backslash V$ we consider, $Y \cup A$ is not an answer set of $P \cup R$ for any $A \subseteq V$. Therefore, by Def. 2, Y is not an answer set of $\mathsf{f}(P, V) \cup R$ for any such R. Since this holds in particular for the set of facts $R = \{a \leftarrow \mid a \in Y\}$ and $\langle Y, Y\rangle$ is its only HT-model of the form $\langle X, Y\rangle$, we conclude that $\langle Y, Y\rangle \notin \mathcal{HT}(\mathsf{f}(P, V))$.

(ii) For each $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$, there is $A' \subset A$ such that $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$. Consider some R over $\mathcal{A}\backslash V$ with $\langle Y, Y\rangle \in \mathcal{HT}(R)$, such as the above set R of facts. We have $\langle Y \cup A'', Y \cup A\rangle \in \mathcal{HT}(R)$ for every $A \subseteq V$ and $A'' \subseteq A$. Therefore, for each $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$, we have that both $\langle Y \cup A, Y \cup A\rangle$ and $\langle Y \cup A', Y \cup A\rangle$ are HT-models of $P \cup R$. Thus, $Y \cup A$ is not an answer set of $P \cup R$ for any $A \subseteq V$ and, following the same argument as in (i), we conclude that $\langle Y, Y\rangle \notin \mathcal{HT}(\mathsf{f}(P, V))$. ■

Proposition 2. *If a forgetting operator* f *over* $\mathcal{C}$ *satisfies* $\mathbf{(SP)}_{\langle P,V\rangle}$ *for an instance* $\langle P, V\rangle$ *over* $\mathcal{C}$*, then*

$$\mathcal{HT}(\mathsf{f}(P, V)) \subseteq \mathcal{HT}(P)_{\|V}.$$

Proof. Let f be a forgetting operator over some $\mathcal{C}$ that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ for an instance $\langle P, V\rangle$ over $\mathcal{C}$. We consider two cases.

First, let $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. Let R be a program over $\mathcal{A}(P)\backslash V$, such as the set of facts $R = \{a \leftarrow \mid a \in Y\}$, whose only HT-model of the form $\langle X, Y\rangle$ is $\langle Y, Y\rangle$. Then $Y \in \mathcal{AS}(\mathsf{f}(P, V) \cup R)$. Since f satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$, we have that $Y \in \mathcal{AS}(P \cup R)_{\|V}$. Then, there exists $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. In particular, $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$, and hence $\langle Y, Y\rangle \in \mathcal{HT}(P)_{\|V}$.

Now let $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$ with $X \subset Y$. We then have that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, and, as we proved above, $\langle Y, Y\rangle \in \mathcal{HT}(P)_{\|V}$. Let R be a program over $\mathcal{A}(P)\backslash V$ whose only HT-models of the form $\langle X', Y\rangle$ are $\langle X, Y\rangle$ and $\langle Y, Y\rangle$, such as the unary program $R = \{a \leftarrow \mid a \in X\} \cup \{b \leftarrow c \mid b, c \in Y\backslash X\}$. Then, $Y \notin \mathcal{AS}(\mathsf{f}(P, V) \cup R)$. Since f satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$, we have that $Y \notin \mathcal{AS}(P \cup R)_{\|V}$. Since R is a program over $\mathcal{A}(P)\backslash V$, the only HT-models of R of the form $\langle X', Y \cup A\rangle$ with $A \subseteq V$ are of the form (a) $\langle X \cup A', Y \cup A\rangle$ and (b) $\langle Y \cup A', Y \cup A\rangle$ with $A' \subseteq A$. In addition, as $\langle Y, Y\rangle \in \mathcal{HT}(P)_{\|V}$, we have $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$ for some $A' \subseteq A \subseteq V$. Hence, $\langle Y, Y\rangle \in \mathcal{HT}(P \cup R)_{\|V}$. Since $Y \notin \mathcal{AS}(P \cup R)_{\|V}$, $\mathcal{HT}(P)$ contains at least one HT-model of the form (a) or (b) with $A' \subset A$ in the latter case. Now, since $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, we can use Lemma 6 to conclude that there is $A \in Rel^Y_{\langle P,V\rangle}$, i.e., there is $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$ and there is no $A' \subset A$ such that $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$. Thus, for such A, an HT-model of the form (a) occurs in $\mathcal{HT}(P)$. Therefore, $\langle X, Y\rangle \in \mathcal{HT}(P)_{\|V}$. ■

Theorem 2. *If* $\langle P, V\rangle$ *satisfies* Ω*, then no forgetting operator* f *over* $\mathcal{C} \supseteq \mathcal{C}_d$ *satisfies* $\mathbf{(SP)}_{\langle P,V\rangle}$*.*

Proof. We show the contrapositive, i.e., if there exists a forgetting operator f that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$, then $\langle P, V\rangle$ does not satisfy Ω.

Consider f that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$. To show that $\langle P, V\rangle$ does not satisfy Ω, we need to show that there is no $Y \subseteq \mathcal{A}\backslash V$ such that the set of sets $\mathcal{R}^Y_{\langle P,V\rangle} = \{R^{Y,A}_{\langle P,V\rangle} \mid A \in Rel^Y_{\langle P,V\rangle}\}$ is non-empty and has no least element. Take some $Y \subseteq \mathcal{A}\backslash V$. We have to consider the following three cases:

(i) $Rel^Y_{\langle P,V\rangle}$ is empty. Then $\mathcal{R}^Y_{\langle P,V\rangle}$ is empty as well.

(ii) $Rel^Y_{\langle P,V\rangle}$ contains exactly one element A. Then, $R^{Y,A}_{\langle P,V\rangle}$ is least in $\mathcal{R}^Y_{\langle P,V\rangle}$.
(iii) There is more than one element in $Rel^Y_{\langle P,V\rangle}$. We proceed as follows.
We first show that,

$$\text{if } \langle X,Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V)),\ \text{then } X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}. \tag{9}$$

Assume that $\langle X,Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Consider a program R over $\mathcal{A}\backslash V$ whose HT-models over $\mathcal{A}\backslash V$ of the form $\langle X',Y\rangle$ are only $\langle Y,Y\rangle$ and $\langle X,Y\rangle$, such as the unary program $R = \{a \leftarrow \mid a \in X\} \cup \{b \leftarrow c \mid b,c \in Y\backslash X\}$. Then, $Y \notin \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. Thus, $Y \notin \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. By Def. 2, we have $Y \notin \mathcal{AS}(P \cup R)_{\|V}$, i.e., $Y \cup A \notin \mathcal{AS}(P \cup R)$, for any $A \subseteq V$. In particular, $Y \cup A \notin \mathcal{AS}(P \cup R)$, for every $A \in Rel^Y_{\langle P,V\rangle}$. Since R is a program over $\mathcal{A}(P)\backslash V$, the only HT-models of R of the form $\langle X', Y \cup A\rangle$ with $A \subseteq V$ are of the form (a) $\langle X \cup A', Y \cup A\rangle$ and (b) $\langle Y \cup A', Y \cup A\rangle$ with $A' \subseteq A$. For each $A \in Rel^Y_{\langle P,V\rangle}$we have, by Def. 3, that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$. This, together with $Y \cup A \notin \mathcal{AS}(P \cup R)$, implies $\langle X \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$ for some $A' \subseteq A$. Thus, $X \in R^{Y,A}_{\langle P,V\rangle}$, for each $A \in Rel^Y_{\langle P,V\rangle}$, and we can therefore conclude that $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$.

Now suppose that $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element, i.e., no element of $\mathcal{R}^Y_{\langle P,V\rangle}$ coincides with $\bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. Then, for each $A \in Rel^Y_{\langle P,V\rangle}$, there exists $X_A \in R^{Y,A}_{\langle P,V\rangle}$ s.t. $X_A \notin \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. Take a program R over $\mathcal{A}\backslash V$ whose HT-models of the form $\langle X',Y\rangle$ are exactly $\langle Y,Y\rangle$ and $\langle X_A,Y\rangle$ for each $A \in Rel^Y_{\langle P,V\rangle}$. Note that by Lemma 2 there is a disjunctive program with exactly that set of HT-models. We have $Y \notin \mathcal{AS}(P \cup R)_{\|V}$. Since each $\langle X_A,Y\rangle$ cannot belong to $\mathcal{HT}(\mathsf{f}(P,V))$, by (9), we have that $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. This contradicts the assumption that f satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$, which finishes the proof. ■

Proposition 3. *Let $V \subseteq \mathcal{A}$ be some set of atoms. Then, there is $P \in \mathcal{C}_e$ such that every forgetting operator f over $\mathcal{C}_e$ does not satisfy $\mathbf{(SP)}_{\langle P,V\rangle}$.*

Proof. First consider the case where V has only the element p. Take $a,b \in \mathcal{A}$ both distinct and distinct from p, and consider the program

$$a \leftarrow p \qquad\qquad b \leftarrow not\ p \qquad\qquad p \leftarrow not\ not\ p$$

The proof for this case where $|V|= 1$ follows precisely the argument in the proof of Thm. 1. For every V such that $|V|> 1$, we can consider the above program for some $p \in V$, together with a rule $p_i \leftarrow p_i$ for each of the remaining $p_i \in V$. The proof argument applies in the very same manner. ■

Proposition 4. *Let P be a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, and f a forgetting operator that satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$. There may not exist any V' with $\emptyset \subset V' \subset V$ such that f satisfies $\mathbf{(SP)}_{\langle P,V'\rangle}$.*

Proof. Consider the HT-models $\langle \emptyset, ap\rangle$, $\langle ap, ap\rangle$, $\langle \emptyset, aq\rangle$, $\langle aq, aq\rangle$, $\langle pq, apq\rangle$, and $\langle apq, apq\rangle$. By Lemma 1 there is a program P whose HT-models are exactly these. Take $V = \{p, q\}$. We can easily check that $\langle P, V\rangle$ does not satisfy Ω. Therefore, according to Thm. 3, every $\mathsf{f} \in \mathsf{F_{SP}}$ satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$[8]. Nevertheless, it is also easy to check that both $\langle P, \{p\}\rangle$ and $\langle P, \{q\}\rangle$ satisfy Ω. Therefore, according to Thm. 2, there is no forgetting operator that satisfies $\mathbf{(SP)}_{\langle P,\{p\}\rangle}$ nor $\mathbf{(SP)}_{\langle P,\{q\}\rangle}$. ■

Proofs of Sec. 4

Theorem 3. *Every* $\mathsf{f} \in \mathsf{F_{SP}}$ *satisfies* $\mathbf{(SP)}_{\langle P,V\rangle}$ *for every* $\langle P, V\rangle$ *over* $\mathcal{C}(\mathsf{f})$ *that does not satisfy* Ω.

Proof. Let $\mathsf{f} \in \mathsf{F_{SP}}$ and let $\langle P, V\rangle$ be an instance such that $\langle P, V\rangle$ does not satisfy Ω. We have to show that $\mathcal{AS}(\mathsf{f}(P, V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, for all programs $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$.

"$\subseteq$": Suppose that $Y \in \mathcal{AS}(\mathsf{f}(P, V) \cup R)$. Then, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$ and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$. By Def. 5, we can conclude that there is no $X \subset Y$ with $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ and $\langle X, Y\rangle \in \mathcal{HT}(R)$. Since $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$, we can conclude that $\mathcal{R}^Y_{\langle P,V\rangle}$ is non-empty, and given that we are assuming that $\langle P, V\rangle$ does not satisfy Ω, $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element, which is precisely $\bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. Let $A \in Rel^Y_{\langle P,V\rangle}$ be such that $R^{Y,A}_{\langle P,V\rangle} = \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. Then, since $\langle Y, Y\rangle \in \mathcal{HT}(R)$ and $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we have that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$. Therefore, $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. Since $A \in Rel^Y_{\langle P,V\rangle}$ such that $R^{Y,A}_{\langle P,V\rangle} = \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ and there is no $X \subset Y$ with $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ such that $\langle X, Y\rangle \in \mathcal{HT}(R)$, we can conclude that there is no $X \subset Y \cup A$ such that $\langle X, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. Therefore, $Y \cup A \in \mathcal{AS}(P \cup R)$, and so $Y \in \mathcal{AS}(P \cup R)_{\|V}$.

"$\supseteq$": Suppose $Y \in \mathcal{AS}(P \cup R)_{\|V}$. Then there exists $A \subseteq V$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$ and there is no $X \subset Y \cup A$ such that $\langle X, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. Therefore, $\mathcal{R}^Y_{\langle P,V\rangle}$ is non-empty and $A \in Rel^Y_{\langle P,V\rangle}$. In fact, $R^{Y,A}_{\langle P,V\rangle} = \{Y\}$, so there is no $X' \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ with $X' \subset Y$ such that $\langle X', Y\rangle \in \mathcal{HT}(R)$. Therefore, there is no $\langle X', Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$ such that $X' \subset Y$. In order to conclude that $Y \in \mathcal{AS}(\mathsf{f}(P, V) \cup R)$ we thus just need to prove that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$. Since $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$ and $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we have that $\langle Y, Y\rangle \in \mathcal{HT}(R)$. Since $Y \in R^{Y,A}_{\langle P,V\rangle}$ for all $A \in Rel^Y_{\langle P,V\rangle}$, we clearly have that $Y \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ and therefore, by Def. 5, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. We can then conclude that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$. Hence, $Y \in \mathcal{AS}(\mathsf{f}(P, V) \cup R)$. ■

[8] We safely use Theorem 3, which is only proved afterwards, since its proof does not depend on the current result.

Corollary 1. *Every forgetting operator* $\mathsf{f} \in \mathsf{F_{SP}}$ *satisfies* $\mathbf{(SP)}_{\langle P,V\rangle}$ *iff* $\langle P, V\rangle$ *over* $\mathcal{C}(\mathsf{f})$ *does not satisfy* Ω.

Proof. Let $\mathsf{f} \in \mathsf{F_{SP}}$. First, suppose that f satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$. Then, by Thm. 2, $\langle P, V\rangle$ does not satisfy Ω. Now suppose that $\langle P, V\rangle$ does not satisfy Ω. Then, since $\mathsf{f} \in \mathsf{F_{SP}}$, we have, by Thm. 3, that f satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$. ■

Proposition 5. *Let* $\langle P, V\rangle$ *be an instance over* $\mathcal{C}$ *that does not satisfy* Ω. *Then, for every forgetting operator* f *satisfying* $\mathbf{(SP)}_{\langle P,V\rangle}$ *and every* $\mathsf{f}' \in \mathsf{F_{SP}}$ *with* $\mathcal{C} \subseteq \mathcal{C}(\mathsf{f})$ *and* $\mathcal{C} \subseteq \mathcal{C}(\mathsf{f}')$, *we have that* $\mathsf{f}(P, V) \equiv \mathsf{f}'(P, V)$.

Proof. Let f and f' be forgetting operators such that f satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ and $\mathsf{f}' \in \mathsf{F_{SP}}$. Since both $\mathsf{f}(P, V)$ and $\mathsf{f}'(P, V)$ are programs over $\mathcal{A}\backslash V$, all we need to prove is that $\mathcal{AS}(\mathsf{f}(P, V) \cup R) = \mathcal{AS}(\mathsf{f}'(P, V) \cup R)$, for every program R over $\mathcal{A}\backslash V$. Since $\mathsf{f}' \in \mathsf{F_{SP}}$, we have, by Thm. 3, that f' satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$. Since both f and f' satisfy $\mathbf{(SP)}_{\langle P,V\rangle}$, we have that $\mathcal{AS}(\mathsf{f}(P, V)\cup R) = \mathcal{AS}(P\cup R)_{\|V} = \mathcal{AS}(\mathsf{f}'(P, V) \cup R)$, by Def. 2, for every program R over $\mathcal{A}\backslash V$. We can therefore conclude that $\mathsf{f}(P, V) \equiv \mathsf{f}'(P, V)$. ■

Proposition 6. *Let* f *be in* $\mathsf{F_{SP}}$. *Then, for every* $V \subseteq \mathcal{A}$:

$$\mathcal{HT}(\mathsf{f}(P, V)) = \mathcal{HT}(P)_{\|V} \text{ for } P \in \mathcal{C}_H.$$

Proof. "$\subseteq$": Suppose that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. Then $Y \subseteq \mathcal{A}(P)\backslash V$ and $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$, i.e., there is at least one $A \in Rel^Y_{\langle P,V\rangle}$ such that $X \in R^{Y,A}_{\langle P,V\rangle}$. Hence, there is $\langle X', Y \cup A\rangle \in \mathcal{HT}(P)$ with $X = X'\backslash V$, and thus $\langle X, Y\rangle \in \mathcal{HT}(P)_{\|V}$.

"$\supseteq$": Now, suppose that $\langle X, Y\rangle \in \mathcal{HT}(P)_{\|V}$. Then, there exists $A' \subseteq A \subseteq V$ such that $\langle X \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$. Note that, by (H_2) of Lemma 4, $Rel^Y_{\langle P,V\rangle}$ cannot be empty. Thus, consider some $A'' \in Rel^Y_{\langle P,V\rangle}$. Then $\langle Y \cup A'', Y \cup A''\rangle \in \mathcal{HT}(P)$. Since $P \in \mathcal{C}_H$, we can use (H_3) of Lemma 4 to conclude that $\langle (X\cup A')\cap (Y\cup A''), (Y\cup A)\cap(Y\cup A'')\rangle \in \mathcal{HT}(P)$, i.e., $\langle X\cup(A'\cap A''), Y\cup(A\cap A'')\rangle \in \mathcal{HT}(P)$. Since $Y \cup (A \cap A'') \subseteq Y \cup A''$, and using (H_1) of Lemma 4, we can conclude that $\langle X \cup (A' \cap A''), Y \cup A''\rangle \in \mathcal{HT}(P)$. Consequently $X \in R^{Y,A''}_{\langle P,V\rangle}$. Hence $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$, and therefore $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. ■

Theorem 4. $\mathsf{F_{SP}}$ *is closed for extended programs and Horn programs, but neither for disjunctive programs nor normal programs.*

Proof. First, we consider extended programs. Let $\mathsf{f} \in \mathsf{F_{SP}}$ and $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$ for every P and every $V \subseteq \mathcal{A}$. By Lemma 1, we need to show that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. If $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, then $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ by Def. 5. Thus, $X \in R^{Y,A}_{\langle P,V\rangle}$ for every $A \in Rel^Y$, and, for each such A, we have $\langle X \cup V', Y \cup A\rangle \in \mathcal{HT}(P)$ for some $V' \subseteq A$. By Lemma 1, we have $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$ for each such A, and thus $Y \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. Hence, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$ by Def. 5.

Now consider Horn programs. Let $\mathsf{f} \in \mathsf{F_{SP}}$. Using Lemmas 4 and 5, we only need to prove that conditions (H_2) and (H_3) hold for $\mathcal{HT}(\mathsf{f}(P,V))$ for every Horn program P and every $V \subseteq \mathcal{A}$. First, consider condition (H_2).

If: take $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. We show that $\langle X, X\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ ($X \subseteq Y$ holds by definition of HT-models). Since $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$, we have $X \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}$ by Def. 5. Then, $\langle X \cup A', Y \cup A''\rangle \in \mathcal{HT}(P)$ for some $A' \subseteq A'' \subseteq V$. Using condition (H_2) for $\mathcal{HT}(P)$ we have that $\langle X \cup A', X \cup A'\rangle \in \mathcal{HT}(P)$ and $\langle Y \cup A'', Y \cup A''\rangle \in \mathcal{HT}(P)$. Then, for every $A''' \in Rel^{X}_{\langle P,V\rangle}$ we have $\langle X \cup A''', X \cup A'''\rangle \in \mathcal{HT}(P)$. Hence, $X \in \bigcap \mathcal{R}^{X}_{\langle P,V\rangle}$, and therefore $\langle X, X\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ by Def. 5. In the same way, for every $A''' \in Rel^{Y}_{\langle P,V\rangle}$, we have $\langle Y \cup A''', Y \cup A'''\rangle \in \mathcal{HT}(P)$. Hence, $Y \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}$, and therefore, by Def. 5, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$.

Only-if: assume that $\langle X, X\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ with $X \subseteq Y$. We aim to prove that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle X, X\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ we have that $X \in \bigcap \mathcal{R}^{X}_{\langle P,V\rangle}$ and $Y \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}$ by Def. 5. Then, there is $A' \subseteq V$ such that $\langle X \cup A', X \cup A'\rangle \in \mathcal{HT}(P)$ and $A'' \subseteq V$ such that $\langle Y \cup A'', Y \cup A''\rangle \in \mathcal{HT}(P)$. Also, for every $A'' \in Rel^{Y}_{\langle P,V\rangle}$ we have $\langle Y \cup A'', Y \cup A''\rangle \in \mathcal{HT}(P)$. Since $X \subseteq Y$ and using condition (H_3) we have that $\langle X \cup (A' \cap A''), X \cup (A' \cap A'')\rangle \in \mathcal{HT}(P)$ for every $A'' \in Rel^{Y}_{\langle P,V\rangle}$. Since $X \cup (A' \cap A'') \subseteq Y \cup A''$ we can use condition (H_2) to conclude that $\langle X \cup (A' \cap A''), Y \cup A'')\rangle \in \mathcal{HT}(P)$ for every $A'' \in Rel^{Y}_{\langle P,V\rangle}$, therefore $X \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}$. Hence, $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$.

For condition (H_3), take $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle X', Y'\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. We aim to prove that $\langle X \cap X', Y \cap Y'\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle H, T\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$, by Def. 5, we have that $X \in \bigcap \mathcal{R}^{Y}_{\langle P,V\rangle}$ and $X' \in \bigcap \mathcal{R}^{Y'}_{\langle P,V\rangle}$. Then, there are $A, A', A'', A''' \subseteq V$ such that $\langle X \cup A, Y \cup A'\rangle \in \mathcal{HT}(P)$ and $\langle X' \cup A'', Y' \cup A'''\rangle \in \mathcal{HT}(P)$. Using condition (H_3) for $\mathcal{HT}(P)$ we have that $\langle (X \cap X') \cup (A \cap A''), (Y \cap Y') \cup (A' \cap A''')\rangle \in \mathcal{HT}(P)$. Then, by (H_2), $\langle (X \cap X') \cup (A \cap A''), (X \cap X') \cup (A \cap A'')\rangle \in \mathcal{HT}(P)$. Moreover, for every $A^* \in Rel^{Y \cap Y'}_{\langle P,V\rangle}$, we have $\langle (Y \cap Y') \cup A^*, (Y \cap Y') \cup A^*\rangle \in \mathcal{HT}(P)$. Then, by (H_3), $\langle (X \cap X') \cup (A \cap A'' \cap A^*), (X \cap X') \cup (A \cap A'' \cap A^*)\rangle \in \mathcal{HT}(P)$, for every $A^* \in Rel^{Y \cap Y'}_{\langle P,V\rangle}$, since $(X \cap X') \subseteq (Y \cap Y')$. Therefore, by (H_2), it follows that $\langle (X \cap X') \cup (A \cap A'' \cap A^*), (Y \cap Y') \cup A^*\rangle \in \mathcal{HT}(P)$ for every $A^* \in Rel^{Y \cap Y'}_{\langle P,V\rangle}$. Thus, $(X \cap X') \in \bigcap \mathcal{R}^{Y \cap Y'}_{\langle P,V\rangle}$, and, hence, $\langle X \cap X', Y \cap Y'\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$.

Finally, the negative results for $(\mathbf{E}_{\mathcal{C}_d})$ and $(\mathbf{E}_{\mathcal{C}_n})$ follow from the forgetting instance given in Example 5. ■

Theorem 5. *Restricted to instances $\langle P, V\rangle$ that do not satisfy Ω, $\mathsf{F_{SP}}$ satisfies* **(wE)**, **(SE)**, **(PP)**, **(SI)**, **(sC)**, **(wC)**, **(CP)**, *and* **(SP)**.

Proof. By Thm. 3, every $\mathsf{f} \in \mathsf{F_{SP}}$ satisfies $\mathbf{(SP)}_{\langle P,V\rangle}$ for every $\langle P, V\rangle$ that does not satisfy Ω. Hence, $\mathsf{F_{SP}}$ satisfies **(SP)** under this restriction. Then, by Prop. 1 of [49], $\mathsf{F_{SP}}$ also satisfies **(wE)**, **(SE)**, **(PP)**, **(SI)**, **(sC)**, **(wC)**, **(CP)**. ■

Theorem 6. *Alg. 1 terminates and* $\mathsf{f}_{\mathsf{SP}} \in \mathsf{F}_{\mathsf{SP}}$.

Proof. Termination follows straightforwardly from the fact that we consider a finite signature $\mathcal{A}$, and thus Y and X in lines 3 and 8 are finite, and in each possible case of augmentation of P' only one rule is added.

Regarding correctness, i.e., $\mathsf{f}_{\mathsf{SP}} \in \mathsf{F}_{\mathsf{SP}}$, let $M = \mathsf{f}(P, V)$ for $\mathsf{f} \in \mathsf{F}_{\mathsf{SP}}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}(P)$. Note that the rules added in lines 5 and 9 are precisely those rules in the definition of P_M that, by Prop. 8, correspond to the countermodels of the HT-models of the characterization of the forgetting result for SP-Forgetting (cf. Def. 5). Then, by Prop. 8, we have that $\mathcal{HT}(P_M) = M$. ■

Proposition 9. *There is a forgetting operator satisfying* $\mathbf{(SP)}_{\langle P,V\rangle}$*, for every* $P \in \mathcal{C}_n$ *and every* V*, such that* $|V| = 1$.

Proof. Let $P \in \mathcal{C}_n$ and assume that $|V| = 1$. By Cor. 1, all we need to prove is that $\langle P, V\rangle$ does not satisfy Ω. Since $|V| = 1$, for every $Y \subseteq \mathcal{A}\backslash V$, the set $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most two elements: $R^{Y,\emptyset}_{\langle P,V\rangle}$ and $R^{Y,V}_{\langle P,V\rangle}$. If $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element, then either the set is empty or it has a least element, and therefore in this case $\langle P, V\rangle$ does not satisfy Ω. Now suppose that $\mathcal{R}^Y_{\langle P,V\rangle}$ has two elements, i.e., both $R^{Y,\emptyset}_{\langle P,V\rangle}$ and $R^{Y,V}_{\langle P,V\rangle}$ belong to $\mathcal{R}^Y_{\langle P,V\rangle}$. Then both $\langle Y, Y\rangle$ and $\langle Y \cup V, Y \cup V\rangle$ are HT-models of P. We now prove that in this case $R^{Y,\emptyset}_{\langle P,V\rangle} \subseteq R^{Y,V}_{\langle P,V\rangle}$, i.e., $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element. Let $X \in R^{Y,\emptyset}_{\langle P,V\rangle}$. Then $\langle X, Y\rangle \in \mathcal{HT}(P)$. Since $\langle X, Y\rangle \in \mathcal{HT}(P)$ and $\langle Y \cup V, Y \cup V\rangle \in \mathcal{HT}(P)$, by (N_2) of Lemma 3, we have that $\langle X, Y \cup V\rangle \in \mathcal{HT}(P)$, and therefore $X\backslash V = X \in R^{Y,V}_{\langle P,V\rangle}$. ■

Proposition 10. *There is a forgetting operator satisfying* $\mathbf{(SP)}_{\langle P,V\rangle}$*, for every* $P \in \mathcal{C}_d$ *and every* V*, such that* $|V| = 1$.

Proof. Let $P \in \mathcal{C}_d$ and assume that $|V| = 1$. By Cor. 1, all we need to prove is that $\langle P, V\rangle$ does not satisfy Ω. Since $|V| = 1$, for every $Y \subseteq \mathcal{A}\backslash V$, the set $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most two elements: $R^{Y,\emptyset}_{\langle P,V\rangle}$ and $R^{Y,V}_{\langle P,V\rangle}$. If $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element, then either the set is empty or it has a least element, and therefore in this case $\langle P, V\rangle$ does not satisfy Ω. Now suppose that $\mathcal{R}^Y_{\langle P,V\rangle}$ has two elements, i.e., both $R^{Y,\emptyset}_{\langle P,V\rangle}$ and $R^{Y,V}_{\langle P,V\rangle}$ belong to $\mathcal{R}^Y_{\langle P,V\rangle}$. Then, both $\langle Y, Y\rangle$ and $\langle Y \cup V, Y \cup V\rangle$ are HT-models of P. We now prove that in this case $R^{Y,\emptyset}_{\langle P,V\rangle} \subseteq R^{Y,V}_{\langle P,V\rangle}$, i.e., $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element. Let $X \in R^{Y,\emptyset}_{\langle P,V\rangle}$, i.e., $\langle X, Y\rangle \in \mathcal{HT}(P)$. Since $\langle X, Y\rangle \in \mathcal{HT}(P)$ and $\langle Y \cup V, Y \cup V\rangle \in \mathcal{HT}(P)$, by (D_2) of Lemma 2, we have that $\langle X, Y \cup V\rangle \in \mathcal{HT}(P)$, and therefore $X\backslash V = X \in R^{Y,V}_{\langle P,V\rangle}$. ■

Theorem 7. *Let* P *be a program. Then the set of sets of atoms*

$$\mathcal{V}_P = \{V \subseteq \mathcal{A}(P) \mid \bigcap \mathcal{R}^Y_{\langle P,V\rangle} \in \mathcal{R}^Y_{\langle P,V\rangle} \text{ for every } \mathcal{R}^Y_{\langle P,V\rangle} \neq \emptyset\}$$

is the set of all sets V *of atoms for which it is possible to forget* V *from* P *while satisfying* $\mathbf{(SP)}_{\langle P,V\rangle}$.

Proof. Let P be a program. Using Cor. 1, we need to check that $\mathcal{V}_P$ is exactly the set of all $V \subseteq \mathcal{A}(P)$ such that the instance $\langle P, V\rangle$ does not satisfies criterion Ω.

Let $V \in \mathcal{V}_P$. By definition of $\mathcal{V}_P$, we have that for every $Y \subseteq \mathcal{A}(P)\backslash V$ either $\mathcal{R}^Y_{\langle P,V\rangle} = \emptyset$ or $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} \in \mathcal{R}^Y_{\langle P,V\rangle}$. Since clearly $\bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ is the least element of $\mathcal{R}^Y_{\langle P,V\rangle}$, $\langle P, V\rangle$ cannot satisfy Ω.

Now let $V \notin \mathcal{V}_P$. Then, by definition of $\mathcal{V}_P$, there exists $Y \subseteq \mathcal{A}(P)\backslash V$ such that $\mathcal{R}^Y_{\langle P,V\rangle} \neq \emptyset$ and $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} \notin \mathcal{R}^Y_{\langle P,V\rangle}$. Suppose that $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element, call it L. Then $L \subseteq R$, for every $R \in \mathcal{R}^Y_{\langle P,V\rangle}$. Therefore $L \subseteq \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. Since $L \in \mathcal{R}^Y_{\langle P,V\rangle}$ we have that $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} \subseteq L$. Thus $L = \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$, which contradicts the fact that $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} \notin \mathcal{R}^Y_{\langle P,V\rangle}$. Therefore, $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element, and we can conclude that $\langle P, V\rangle$ satisfies criterion Ω. ■

Proofs for Sec. 5

Lemma 7. *Let $V \subseteq \mathcal{A}$, R a program over $\mathcal{A}\backslash V$, and $\langle X, Y\rangle, \langle X', Y'\rangle$ HT-interpretations s.t. $X \sim_V X'$ and $Y \sim_V Y'$. Then, $\langle X, Y\rangle \in \mathcal{HT}(R)$ iff $\langle X', Y'\rangle \in \mathcal{HT}(R)$.*

Proof. Suppose $\langle X, Y\rangle \in \mathcal{HT}(R)$. Then we have that $Y \models R$ and $X \models R^Y$. Since R does not contain atoms from V and $Y \sim_V Y'$, it is well-known from classical logic that also $Y' \models R$. Also, it follows immediately from the definition of program reduct that $R^Y = R^{Y'}$. Finally, since R does not contain atoms from V, neither does $R^{Y'}$. Moreover, since $X \sim_V X'$, we have that $X' \models R^Y = R^{Y'}$. Therefore, $\langle X', Y'\rangle \in HT(R)$. The converse direction follows easily using the same argument. ■

Lemma 8. *Let $V \subseteq \mathcal{A}$, R a program over $\mathcal{A}\backslash V$, and $X, Y \subseteq \mathcal{A}\backslash V$. Then, $\langle X, Y\rangle \in \mathcal{HT}(R)_{\|V}$ iff $\langle X, Y\rangle \in \mathcal{HT}(R)$.*

Proof. If: Suppose that $\langle X, Y\rangle \in \mathcal{HT}(R)_{\|V}$. Then, there exists $\langle X', Y'\rangle \in HT(R)$ such that $X'\backslash V = X$ and $Y'\backslash V = Y$. Since $X, Y \subseteq \mathcal{A}\backslash V$, we have that $X\backslash V = X$ and $Y\backslash V = Y$. Therefore, $X \sim_V X'$ and $Y \sim_V Y'$, and we can use Lemma 7 to conclude that $\langle X, Y\rangle \in \mathcal{HT}(R)$.

Only-if: Suppose that $\langle X, Y\rangle \in \mathcal{HT}(R)$. Since $X \subseteq Y \subseteq \mathcal{A}\backslash V$, we have that $X\backslash V = X$ and $Y\backslash V = Y$. Therefore, $\langle X, Y\rangle = \langle X\backslash V, Y\backslash V\rangle \in \mathcal{HT}(R)_{\|V}$. ■

Lemma 9. *Let P be a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, R a program over $\mathcal{A}\backslash V$, and $A \subseteq V$. Then, $A \in Rel^Y_{\langle P,V\rangle}$ and $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$ iff $A \in Rel^Y_{\langle P\cup R,V\rangle}$.*

Proof. If: Consider $A \in Rel^Y_{\langle P,V\rangle}$ and $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$. By Def. 6, $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$ and $\nexists A' \subset A$ such that $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$. Then, $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. Suppose there is $A'' \subset A$ such that $\langle Y \cup A'', Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. Then, in particular, $\langle Y \cup A'', Y \cup A\rangle \in \mathcal{HT}(P)$. We derive a contradiction and conclude $A \in Rel^Y_{\langle P\cup R,V\rangle}$ by Def. 6.

Only-if: Consider $A \in Rel^Y_{\langle P\cup R,V\rangle}$. By Def. 6, $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P \cup R)$ and $\nexists A' \subset A$ such that $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. Then, $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$ and $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$. Suppose there is $A'' \subset A$ such that $\langle Y \cup A'', Y \cup A\rangle \in \mathcal{HT}(P)$. Given $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(R)$, we can conclude $\langle Y \cup A'', Y \cup A\rangle \in \mathcal{HT}(R)$ by Lemma 7, and thus $\langle Y \cup A'', Y \cup A\rangle \in \mathcal{HT}(P \cup R)$. We derive a contradiction and conclude $A \in Rel^Y_{\langle P,V\rangle}$ by Def. 6. ■

Proposition 11. $\mathsf{F_{SP}}$ *satisfies* **(SE)**, **(PP)**, **(SI)**, *and* **(wC)**, *but does not satisfy* **(wE)**, **(W)**, **(sC)**, *and* **(CP)**.

Proof. $\mathsf{F_{SP}}$ satisfies **(SE)**, since the coinciding sets of HT-models of two strongly equivalent programs P and P' yield precisely the same set of HT-models when forgetting about some V from P and P' using some $\mathsf{f} \in \mathsf{F_{SP}}$ with $P, P' \in \mathcal{C}(\mathsf{f})$.

Regarding **(PP)**, let $\mathsf{f} \in \mathsf{F_{SP}}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$. We prove the result by contraposition. Suppose $\mathsf{f}(P,V) \not\models_{\mathsf{HT}} P'$, for $P' \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(P') \subseteq \mathcal{A}\backslash V$. Then, there is $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ such that $\langle X, Y\rangle \notin \mathcal{HT}(P')$. By definition of $\mathsf{F_{SP}}$, for all $A^* \in Rel^Y_{\langle P,V\rangle}$ there is $A' \subseteq A^*$ such that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$, and at least one such A^* exists. Since $\langle X, Y\rangle \notin \mathcal{HT}(P')$ and $\mathcal{A}(P') \subseteq \mathcal{A}\backslash V$, we have, by Lemma 7, that $\langle X \cup A', Y \cup A^*\rangle \notin \mathcal{HT}(P')$ for each such A' and A^*. Therefore, $P \not\models_{\mathsf{HT}} P'$.

For **(SI)**, let $\mathsf{f} \in \mathsf{F_{SP}}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$. For each $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we prove $\mathsf{f}(P,V)\cup R \equiv \mathsf{f}(P\cup R, V)$ by showing $\mathcal{HT}(\mathsf{f}(P,V)\cup R) = \mathcal{HT}(\mathsf{f}(P \cup R, V))$.

"$\subseteq$": Let $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. Then, $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle X, Y\rangle \in \mathcal{HT}(R)$. Therefore, $Rel^Y_{\langle P,V\rangle} \neq \emptyset$ and for all $A^* \in Rel^Y_{\langle P,V\rangle}$ there is $A' \subseteq A^*$ such that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$, and at least one such A^* exists. Since $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we have, by Lemma 7, that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(R)$. Therefore, $\langle X\cup A', Y\cup A^*\rangle \in \mathcal{HT}(P\cup R)$ for each such A' and A^*. In particular, $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(R)$ and $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$ by Lemma 1. Then, by Lemma 9, $A^* \in Rel^Y_{\langle P\cup R,V\rangle}$ for all such A^*. By Def. 5, we obtain $\langle X, Y\rangle = \langle (X \cup A')\backslash V, Y\rangle \in \mathcal{HT}(\mathsf{f}(P \cup R, V))$.

"$\supseteq$": Let $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P \cup R, V))$. Then, by definition of $\mathsf{F_{SP}}$, for all $A^* \in Rel^Y_{\langle P\cup R,V\rangle}$ there is $A' \subseteq A^*$ such that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$, and at least one such A^* exists. This implies that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$ and $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(R)$. By Lemma 7, we have that $\langle X, Y\rangle \in \mathcal{HT}(R)$. Also, by Lemma 9, we can conclude that $A^* \in Rel^Y_{\langle P,V\rangle}$ for all $A^* \in Rel^Y_{\langle P\cup R,V\rangle}$. Now, since $\langle X\cup A', Y\cup A^*\rangle \in \mathcal{HT}(P)$ and $A^* \in Rel^Y_{\langle P,V\rangle}$, we have, by definition of $\mathsf{F_{SP}}$, that $\langle X, Y\rangle = \langle (X \cup A')\backslash V, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle X, Y\rangle \in \mathcal{HT}(R)$, we can conclude that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$.

Regarding **(wC)**, let $\mathsf{f} \in \mathsf{F_{SP}}$, $P \in \mathcal{C}(\mathsf{f})$, $V \subseteq \mathcal{A}$ a set of atoms to be forgotten, and S an answer set of P. Then, $\langle S, S\rangle$ is an HT-model of P and there is no $X \subset S$ such that $\langle X, S\rangle$ is also an HT-model of P. Therefore, $R^{S\backslash V, S\cap V}_{\langle P,V\rangle} = \{S\backslash V\}$ is the least element of $\mathcal{R}^{S\backslash V}_{\langle P,V\rangle}$. This means that $\bigcap \mathcal{R}^{S\backslash V}_{\langle P,V\rangle} = \{S\backslash V\}$. Hence, $\langle S\backslash V, S\backslash V\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and there is no $X \subset S\backslash V$ such that $\langle X, S\backslash V\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$, i.e., $S\backslash V$ is an answer set of $\mathsf{f}(P,V)$.

For **(W)**, consider $P = \{a \leftarrow \mathit{not}\, b;\ b \leftarrow \mathit{not}\, c\}$. We can verify that, for every $\mathsf{f} \in \mathsf{F_{SP}}$, $\mathsf{f}(P,\{b\})$ is strongly equivalent to $\{a \leftarrow \mathit{not}\,\mathit{not}\, c\}$, which is not a consequence of P, hence **(W)** is not satisfied.

For **(sC)** and **(CP)**, consider $P = \{a \leftarrow p; b \leftarrow \mathit{not}\, p; p \leftarrow \mathit{not}\,\mathit{not}\, p\}$. Now, for every $\mathsf{f} \in \mathsf{F_{SP}}$, $\mathcal{HT}(\mathsf{f}(P,\{p\}))$ contains precisely three HT-models, namely $\langle a, a\rangle$, $\langle b, b\rangle$, and $\langle ab, ab\rangle$, which means that $\mathsf{f}(P,\{p\})$ has the three answer sets $\{\{a\},\{b\},\{a,b\}\}$. Since the answer sets of P modulo $\{p\}$ are $\{\{a\},\{b\}\}$, we know that **(sC)** is not satisfied, nor **(CP)** by Prop. 1 of [49].

For **(wE)**, consider $P' = \{a \leftarrow p; b \leftarrow \mathit{not}\, a; p \leftarrow \mathit{not}\,\mathit{not}\, p\}$, which has the same answer sets as P in the counterexample for **(sC)** (and **(CP)**), but, for every $\mathsf{f} \in \mathsf{F_{SP}}$, the answer sets of $\mathsf{f}(P',\{p\})$ are $\{\{a\},\{b\}\}$, whereas those of $\mathsf{f}(P,\{p\})$ are $\{\{a\},\{b\},\{a,b\}\}$. Therefore, **(wE)** is not satisfied. ■

Proposition 12. *Let P be a program, $V \subseteq \mathcal{A}$ and R_1, R_2 programs over $\mathcal{A}\backslash V$. Then, $P \cup R_1 \cup R_2 \equiv_V P$ iff $P \cup R_1 \equiv_V P$ and $P \cup R_2 \equiv_V P$.*

Proof. We start with an observation. Recall that, by definition, $P \equiv_V Q$ iff for all programs R over $\mathcal{A}\backslash V$, $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$. From this, it follows that $P \equiv_V Q$ implies $P \cup R \equiv_V Q \cup R$, for every R over $\mathcal{A}\backslash V$.

Only-if: Assume that $P \cup R_1 \equiv_V P$ and $P \cup R_2 \equiv_V P$. From the above observation we get that $P \cup R_1 \cup R_2 \equiv_V P \cup R_2$. By transitivity of $\equiv_V$ and the assumption that $P \cup R_2 \equiv_V P$, we obtain $P \cup R_1 \cup R_2 \equiv_V P$.

If: Assume that $P \cup R_1 \cup R_2 \equiv_V P$. Then, again using the above observation, we have that $P \cup R_1 \cup R_2 \cup R_1 \equiv_V P \cup R_1$. Clearly, $P \cup R_1 \cup R_2 \cup R_1 = P \cup R_1 \cup R_2$. Then, by transitivity, we can conclude that $P \cup R_1 \equiv_V P$. The proof that $P \cup R_2 \equiv_V P$ can be done in a similar way. ■

Proposition 13. *Let P be a program and $V \subseteq \mathcal{A}$. Then, $\mathsf{f_r}(P,V)$ is the largest set of rules R over the alphabet $\mathcal{A}\backslash V$ such that $P \cup R \equiv_V P$.*

Proof. Assume that there is $R \supset \mathsf{f_r}(P,V)$ such that $P \cup R \equiv_V P$. Let $r \in R$ such that $r \notin \mathsf{f_r}(P,V)$. Using Prop. 12, we have that $P \cup R \equiv_V P$ iff $P \cup R\backslash\{r\} \equiv_V P$ and $P \cup \{r\} \equiv_V P$. But then, $P \cup \{r\} \equiv_V P$ contradicts $r \notin \mathsf{f_r}(P,V)$. ■

Proposition 14. *Let $A \subseteq V \subseteq \mathcal{A}$ and $Y \subseteq \mathcal{A}\backslash V$. Then,*

$$Y \cup A \in \mathit{Rel}(P,V) \text{ iff } A \in \mathit{Rel}^{Y}_{\langle P,V\rangle}.$$

Proof. Just by inspecting the definitions of $Rel(P,V)$ and $Rel^Y_{\langle P,V\rangle}$, it is immediate that the conditions for $Y \cup A \in Rel(P,V)$ are equivalent to those for $A \in Rel^Y_{\langle P,V\rangle}$. ■

Proposition 15. *Let P be a program and $V \subseteq \mathcal{A}$. Then, a V-HT-interpretation $\langle X,Y\rangle$ is a V-HT-model of P iff the following conditions hold:*

(1) $Y \in Rel(P,V)$*;*

(2) If $X \subset Y$, then there exists $X' \subset Y$ with $X = X'\backslash V$ such that $\langle X',Y\rangle \in \mathcal{HT}(P)$.

Proof. If: Let $\langle X,Y\rangle \in \mathcal{HT}_V(P)$. Then, (i) and (ii) of the definition of V-HT-model immediately imply (i) and (ii) of Def. 7, resp., i.e., $Y \in Rel(P,V)$, and therefore (1) is satisfied. Also, (i) and (iii) of the definition of V-HT-model together with the definition of V-HT-interpretations imply (2).

Only-if: Now, let $\langle X,Y\rangle$ be a V-HT-interpretation satisfying conditions (1) and (2). Clearly, (1) implies (i) and (ii) of Def. 7, and thus (i) and (ii) of the definition of V-HT-model. Moreover, (2) implies (iii) of the definition of V-HT-model. Therefore, $\langle X,Y\rangle$ is a V-HT-model of P. ■

Lemma 10. *Let P be a program and $V \subseteq \mathcal{A}$. Then, $\langle Y,Y\rangle \in \mathcal{HT}_V(P)$ iff $Y \in Rel(P,V)$.*

Proof. If: Suppose that $Y \in Rel(P,V)$. Then, (1) of Prop. 15 is satisfied. Moreover, (2) of Prop. 15 is trivially satisfied since $X = Y$. Therefore, $\langle Y,Y\rangle \in \mathcal{HT}_V(P)$.

Only-if: Now suppose that $\langle Y,Y\rangle \in \mathcal{HT}_V(P)$. Then, by (1) of Prop. 15, $Y \in Rel(P,V)$. ■

Proposition 16. *Let P be a program and $V \subseteq \mathcal{A}$. Then,*

$$\mathcal{HT}_V(P) = \bigcup_{Y \in Rel(P,V)} (\{\langle X\backslash V, Y\rangle : \langle X,Y\rangle \in \mathcal{HT}(P) \text{ and } X \subset Y\} \cup \{\langle Y,Y\rangle\})$$

Proof. Let $\Phi = \bigcup_{Y \in Rel(P,V)}\{\langle Y,Y\rangle\} \cup \{\langle X\backslash V,Y\rangle : \langle X,Y\rangle \in \mathcal{HT}(P) \text{ and } X \subset Y\}$. We show that $\mathcal{HT}_V(P) = \Phi$.

"$\subseteq$": Suppose that $\langle Y,Y\rangle \in \mathcal{HT}_V(P)$. By Lemma 10, we have that $Y \in Rel(P,V)$. Therefore, $\langle Y,Y\rangle \in \Phi$. Now let $\langle X,Y\rangle \in \mathcal{HT}_V(P)$ s.t. $X \subset Y$. By (1) of Prop. 15, we have that $Y \in Rel(P,V)$. By (2) of Prop. 15, there exists $\langle X',Y\rangle \in \mathcal{HT}(P)$ s.t. $X = X'\backslash V$. But then clearly $\langle X,Y\rangle = \langle X'\backslash V,Y\rangle \in \Phi$.

"$\supseteq$": Suppose that $\langle Y,Y\rangle \in \Phi$. Then, $Y \in Rel(P,V)$, and we can conclude, by Lemma 10, that $\langle Y,Y\rangle \in \mathcal{HT}_V(P)$. Now let $\langle X,Y\rangle \in \Phi$ with $X \subset Y$. First note that $\langle X,Y\rangle$ is a V-HT-interpretation since, by definition of Φ, $X = X'\backslash V$ for some $X' \subset Y$, and therefore $X \subset Y\backslash V$. Since $Y \in Rel(P,V)$, condition (1) of Prop. 15 is satisfied. By definition of Φ, there exists $\langle X',Y\rangle \in \mathcal{HT}(P)$ such that $X = X'\backslash V$. This implies that (2) of Prop. 15 is also satisfied. Therefore, by Prop. 15, $\langle X,Y\rangle \in \mathcal{HT}_V(P)$. ■

Lemma 11. *Let P be a program and $V \subseteq \mathcal{A}$. Then, there is a program R over $\mathcal{A}\backslash V$ such that $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$.*

Proof. We prove that if $\langle X, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$, then also $\langle Y, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$. In this case, since $\mathcal{HT}_V(P)_{\|V}$ is a set of HT-interpretations over $\mathcal{A}\backslash V$, Prop. 4 of [46] allows us to conclude that there is a program R over the alphabet $\mathcal{A}\backslash V$ whose set of HT-models over $\mathcal{A}\backslash V$ is precisely $\mathcal{HT}_V(P)_{\|V}$, which, together with Lemma 8, implies that $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$.

So, suppose that $\langle X, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$. Then, there exists $\langle X', Y'\rangle \in \mathcal{HT}_V(P)$ such that $X'\backslash V = X$ and $Y'\backslash V = Y$. By Prop. 16, we have that $Y' \in Rel(P, V)$. Therefore, by Lemma 10, we have that $\langle Y', Y'\rangle \in \mathcal{HT}_V(P)$, which implies that $\langle Y, Y\rangle = \langle Y'\backslash V, Y'\backslash V\rangle \in \mathcal{HT}_V(P)_{\|V}$. ■

Lemma 12. *Let P be a program, $V \subseteq \mathcal{A}$ and R a program over $\mathcal{A}\backslash V$. If $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$, then $R \subseteq \mathsf{f_r}(P, V)$, i.e., $P \cup \{r\} \equiv_V P$ for every $r \in R$.*

Proof. Suppose $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$. Then, using Prop. 12, we just need to prove that $P \cup R \equiv_V P$, i.e., $\mathcal{HT}_V(P \cup R) = \mathcal{HT}_V(P)$.

"$\subseteq$": We start by proving that $Rel(P \cup R, V) \subseteq Rel(P, V)$. Let $Y \in Rel(P \cup R, V)$. Then $\langle Y, Y\rangle \in \mathcal{HT}(P \cup R)$ by Def. 7, and in particular $\langle Y, Y\rangle \in \mathcal{HT}(P)$ and $\langle Y, Y\rangle \in \mathcal{HT}(R)$. Since R does not contain atoms from V, we can conclude, from Lemma 7, that $\langle Y\backslash V, Y\backslash V\rangle \in \mathcal{HT}(R)$. Assume that $Y \notin Rel(P, V)$, i.e., there is some $Y' \subset Y$ with $Y \sim_V Y'$ such that $\langle Y', Y\rangle \in \mathcal{HT}(P)$. Since R does not contain atoms from V, $\langle Y\backslash V, Y\backslash V\rangle \in \mathcal{HT}(R)$ and $Y \sim_V Y'$, we have, again by Lemma 7, that $\langle Y', Y\rangle \in \mathcal{HT}(P \cup R)$. Since this contradicts $Y \in Rel(P \cup R, V)$, we can conclude that in fact $Y \in Rel(P, V)$.

Now assume that $\langle Y, Y\rangle \in \mathcal{HT}_V(P \cup R)$. Then, by Lemma 10, we have that $Y \in Rel(P \cup R, V)$, and, as we proved above, $Y \in Rel(P, V)$. Therefore, again by Lemma 10, we have $\langle Y, Y\rangle \in \mathcal{HT}_V(P)$. Now let $\langle X, Y\rangle \in \mathcal{HT}_V(P \cup R)$ with $X \subset Y$. By Prop. 16, we have that $Y \in Rel(P \cup R, V)$. Then, as we proved above, $Y \in Rel(P, V)$. Also, by Prop. 16, there is $X' \subset Y$ with $X'\backslash V = X$ such that $\langle X', Y\rangle \in \mathcal{HT}(P \cup R)$. Then $\langle X', Y\rangle \in \mathcal{HT}(P)$, and since $Y \in Rel(P, V)$, we have, by Prop. 16, that $\langle X, Y\rangle = \langle X'\backslash V, Y\rangle \in \mathcal{HT}_V(P)$.

"$\supseteq$": We start by proving that $Rel(P, V) \subseteq Rel(P \cup R, V)$. Let $Y \in Rel(P, V)$. Then, by Lemma 10, $\langle Y, Y\rangle \in \mathcal{HT}_V(P)$, and we immediately have that $\langle Y\backslash V, Y\backslash V\rangle \in \mathcal{HT}_V(P)_{\|V}$. Since we are assuming that $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$, we can conclude, by Lemma 8, that $\langle Y\backslash V, Y\backslash V\rangle \in \mathcal{HT}(R)$. Since R does not contain atoms from V we have, by Lemma 7, that $\langle Y, Y\rangle \in \mathcal{HT}(R)$. Therefore, $\langle Y, Y\rangle \in \mathcal{HT}(P \cup R)$. Assume that $Y \notin Rel(P \cup R, V)$, i.e., there is $Y' \subset Y$ with $Y' \sim_V Y$ such that $\langle Y', Y\rangle \in \mathcal{HT}(P \cup R)$. Then, in particular, $\langle Y', Y\rangle \in \mathcal{HT}(P)$. Since this contradicts the fact that $Y \in Rel(P, V)$, we can conclude that $Y \in Rel(P \cup R, V)$.

Now suppose that $\langle Y, Y\rangle \in \mathcal{HT}_V(P)$. Then, $Y \in Rel(P, V)$, and, as we proved above, $Y \in Rel(P \cup R, V)$. Therefore, $\langle Y, Y\rangle \in \mathcal{HT}_V(P \cup R)$ by Lemma 10. Now suppose that $\langle X, Y\rangle \in \mathcal{HT}_V(P)$ with $X \subset Y$. Then, by

Prop. 16, we have $Y \in \mathit{Rel}(P,V)$. Since we proved above that $\mathit{Rel}(P,V) \subseteq \mathit{Rel}(P \cup R,V)$, we can conclude $Y \in \mathit{Rel}(P \cup R,V)$. Since $\langle X,Y\rangle \in \mathcal{HT}_V(P)$, and $X = X\backslash V$ by definition of V-HT-interpretations, and since we are assuming that $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$, we can conclude that $\langle X, Y\backslash V\rangle \in \mathcal{HT}(R)_{\|V}$. Using Lemma 7 we can conclude that $\langle X, Y\backslash V\rangle \in \mathcal{HT}(R)$. Since $\langle X,Y\rangle \in \mathcal{HT}_V(P)$, by Prop. 16, there is $X' \subseteq Y$ with $X'\backslash V = X$ such that $\langle X',Y\rangle \in \mathcal{HT}(P)$. Since R does not contain atoms from V and $X \sim_V X'$, we can use Lemma 7 to conclude that $\langle X',Y\rangle \in \mathcal{HT}(R)$. Therefore, $\langle X',Y\rangle \in \mathcal{HT}(P \cup R)$. Since $Y \in \mathit{Rel}(P \cup R,V)$, we can conclude, by Prop. 16, that $\langle X,Y\rangle = \langle X'\backslash V,Y\rangle \in \mathcal{HT}_V(P \cup R)$. ■

Theorem 8. *Let P be a program and $V \subseteq \mathcal{A}$. Then,*

$$\mathcal{HT}(\mathsf{f_r}(P,V))_{\|V} = \mathcal{HT}_V(P)_{\|V}.$$

Proof. "$\subseteq$": First suppose that $\langle X,Y\rangle \in \mathcal{HT}(\mathsf{f_r}(P,V))_{\|V}$. Lemma 11 guarantees the existence of a program R over $\mathcal{A}\backslash V$ such that $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$. Since, by Lemma 12, we have that $R \subseteq \mathsf{f_r}(P,V)$, we clearly have that $\langle X,Y\rangle \in \mathcal{HT}(R)_{\|V}$. Then, since $\mathcal{HT}(R)_{\|V} = \mathcal{HT}_V(P)_{\|V}$, we can conclude that $\langle X,Y\rangle \in \mathcal{HT}_V(P)_{\|V}$.

"$\supseteq$": Now let $\langle X,Y\rangle \in \mathcal{HT}_V(P)_{\|V}$. Then, there is $\langle X',Y'\rangle \in \mathcal{HT}_V(P)$ such that $X'\backslash V = X$ and $Y'\backslash V = Y$. Using Prop. 16, we have that $Y' \in \mathit{Rel}(P,V)$. We now consider two cases:

First suppose that $X = Y$. First, since $Y' \in \mathit{Rel}(P,V)$, we have, by Lemma 10, that $\langle Y',Y'\rangle \in \mathcal{HT}_V(P)$. Recall that, by Prop. 13, we have that $\mathcal{HT}_V(P \cup \mathsf{f_r}(P,V)) = \mathcal{HT}_V(P)$. Therefore, $\langle Y',Y'\rangle \in \mathcal{HT}_V(P \cup \mathsf{f_r}(P,V))$. Using Prop. 16, we have that $\langle Y',Y'\rangle \in \mathcal{HT}(P \cup \mathsf{f_r}(P,V))$. In particular, $\langle Y',Y'\rangle \in \mathcal{HT}(\mathsf{f_r}(P,V))$. Therefore, $\langle Y,Y\rangle = \langle Y'\backslash V,Y'\backslash V\rangle \in \mathcal{HT}(\mathsf{f_r}(P,V))_{\|V}$.

Now suppose that $X \subset Y$. By definition of V-HT-interpretation, we have that either $X' = Y'$ or $X' \subset Y'\backslash V$. The former is not possible since it contradicts the assumption that $X \subset Y$. We can then conclude that $X' \subset Y'\backslash V$, which implies that $X' = X'\backslash V = X$. Therefore $\langle X,Y'\rangle \in \mathcal{HT}_V(P)$. Using Prop. 13, we can conclude that $\langle X,Y'\rangle \in \mathcal{HT}_V(P \cup \mathsf{f_r}(P,V))$. Therefore, by Prop. 16, there exists $X'' \subset Y'$ with $X''\backslash V = X$ such that $\langle X'',Y'\rangle \in \mathcal{HT}(P \cup \mathsf{f_r}(P,V))$. In particular, $\langle X'',Y'\rangle \in \mathcal{HT}(\mathsf{f_r}(P,V))$. We can then conclude that $\langle X,Y\rangle = \langle X''\backslash V,Y'\backslash V\rangle \in \mathcal{HT}(\mathsf{f_r}(P,V))_{\|V}$. ■

Proposition 17. *Let $\mathsf{f} \in \mathsf{F_R}$. Then, for every program $P \in \mathcal{C}(\mathsf{f})$ and $V \subset \mathcal{A}$, we have $\mathsf{f}(P,V) \subseteq \mathsf{f_r}(P,V)$.*

Proof. Let $\mathsf{f} \in \mathsf{F_R}$. Then, for every program $P \in \mathcal{C}(\mathsf{f})$ and $V \subset \mathcal{A}$, we have, by definition of $\mathsf{F_R}$, that $\mathcal{HT}(\mathsf{f}(P,V)) = \mathcal{HT}_V(P)_{\|V}$. But then, since $\mathsf{f}(P,V)$ is a program over $\mathcal{A}\backslash V$, it follows immediately from Lemma 12 that $\mathsf{f}(P,V) \subseteq \mathsf{f_r}(P,V)$. ■

Theorem 9. *Alg. 2 terminates and $\mathsf{f_R} \in \mathsf{F_R}$.*

Proof. Termination follows straightforwardly from the fact that we consider a finite signature $\mathcal{A}$, and thus Y and X in lines 3 and 8 are finite, and in each possible case of augmentation of P' only one rule is added.

Regarding correctness, i.e., $\mathsf{f}_\mathsf{R} \in \mathsf{F}_\mathsf{R}$, let $M = \mathsf{f}(P, V)$ for $\mathsf{f} \in \mathsf{F}_\mathsf{R}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \mathcal{A}(P)$. Note that the rules added in lines 5 and 9 are precisely those rules in the definition of P_M that, by Prop. 8, correspond to the countermodels of the HT-models of the characterization of the forgetting result for SP-Forgetting (cf. Def. 6). Then, by Prop. 8, we have that $\mathcal{HT}(P_M) = M$. ■

Theorem 10. *Let P be a program and $V \subseteq \mathcal{A}$. Then, F_R can be given by the set*

$$\{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}\} \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}.$$

Proof. We need to show the following, for each $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$:

$$\mathcal{HT}_V(P)_{\|V} = \{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}\}.$$

"$\subseteq$": Suppose first that $\langle X, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$. Clearly, $Y \subseteq \mathcal{A}\backslash V$. Also, there is $\langle X', Y'\rangle \in \mathcal{HT}_V(P)$ such that $Y = Y'\backslash V$ and $X = X'\backslash V$. By Prop. 16, we can conclude that $Y' \in Rel(P, V)$. Using Prop. 14 we have that $A = Y'\backslash Y \in Rel^Y_{\langle P,V\rangle}$. We now consider two cases:
First, let $X' = Y'$. Since $Y' \in Rel(P, V)$ we have that $\langle Y', Y'\rangle \in \mathcal{HT}(P)$. Since $Y' = Y \cup A$, we can conclude that $X = X'\backslash V \in R^{Y,A}_{\langle P,V\rangle}$. Therefore, since $A \in Rel^Y_{\langle P,V\rangle}$, we can conclude that $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$.
Second, let $X' \subset Y'$. In this case, by definition of V-HT-model, we have that $X' \subset Y'\backslash V$, and therefore $X = X'\backslash V = X'$. By Prop. 16, there exists $\langle X'', Y'\rangle \in \mathcal{HT}(P)$ such that $X''\backslash V = X$. Hence, $X = X''\backslash V \in R^{Y,A}_{\langle P,V\rangle}$. Therefore, since $A \in Rel^Y_{\langle P,V\rangle}$, we can conclude that $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$.

"$\supseteq$": Conversely, consider $\langle X, Y\rangle$ such that $Y \subseteq \mathcal{A}\backslash V$ and $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$. First, it follows from $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$ that there exists some $A \subseteq V$ such that $\langle X', Y \cup A\rangle \in \mathcal{HT}(P)$ with $X'\backslash V = X$ and $A \in Rel^Y_{\langle P,V\rangle}$. By Prop. 14 we can conclude that $(Y \cup A) \in Rel(P, V)$. We now consider two cases:
First, let $X = Y$. Then, since $(Y \cup A) \in Rel(P, V)$, we have by Prop. 16 that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}_V(P)$. Therefore, $\langle Y, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$.

Second, let $X \subset Y$. Since $(Y \cup A) \in Rel(P, V)$ and $\langle X', Y \cup A\rangle \in \mathcal{HT}(P)$, we have by Prop. 16 that $\langle X'\backslash V, Y \cup A\rangle \in \mathcal{HT}_V(P)$. Therefore, $\langle X, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$. ■

Proposition 18. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then, for every $Y \subseteq \mathcal{A}\backslash V$, we have that $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element.*

Proof. Suppose there are at least two different elements in $\mathcal{R}^Y_{\langle P,V\rangle}$. Then, there are $A, A' \in Rel^Y_{\langle P,V\rangle}$ such that $R^{Y,A}_{\langle P,V\rangle}$ and $R^{Y,A'}_{\langle P,V\rangle}$ are distinct, i.e., there is

$X \in R^{Y,A}_{\langle P,V\rangle}$ such that $X \notin R^{Y,A'}_{\langle P,V\rangle}$, or there is $X \in R^{Y,A'}_{\langle P,V\rangle}$ such that $X \notin R^{Y,A}_{\langle P,V\rangle}$. Without loss of generality, assume there is $X \in R^{Y,A}_{\langle P,V\rangle}$ such that $X \notin R^{Y,A'}_{\langle P,V\rangle}$. Then, there is $A'' \subseteq A$ such that $\langle X \cup A'', Y \cup A\rangle \in \mathcal{HT}(P)$. Since $P \in \mathcal{C}_H$, we can use condition (H_2) of Lemma 4 to conclude that $\langle X \cup A'', X \cup A''\rangle \in \mathcal{HT}(P)$ and $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)$. Since $A' \in Rel^Y_{\langle P,V\rangle}$ we have that $\langle Y \cup A', Y \cup A'\rangle \in \mathcal{HT}(P)$. Using again condition (H_2) of Lemma 4, we can conclude that $\langle X \cup A'', Y \cup A'\rangle \in \mathcal{HT}(P)$, which contradicts $X \notin R^{Y,A'}_{\langle P,V\rangle}$. We can therefore conclude that $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element. ■

Proposition 19. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then, for every $\mathsf{f} \in \mathsf{F_{SP}}$ and $\mathsf{f}' \in \mathsf{F_R}$ we have that $\mathsf{f}(P,V) \equiv \mathsf{f}'(P,V)$.*

Proof. Since we are assuming that P is a Horn program, we can use Prop. 18 to conclude that $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element, for every $Y \subseteq \mathcal{A}\backslash V$. Since in this case $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} = \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$, the result follows immediately from the definition of $\mathsf{F_{SP}}$ and the alternative characterization of $\mathsf{F_R}$ given in Thm. 10. ■

Proposition 20. *$\mathsf{F_R}$ satisfies $(\mathbf{E}_{\mathcal{C}_H})$, $(\mathbf{E}_{\mathcal{C}_e})$, **(SE)**, **(PP)**, **(SI)**, and **(sC)**, but not $(\mathbf{E}_{\mathcal{C}_n})$, $(\mathbf{E}_{\mathcal{C}_d})$, **(wE)**, **(W)**, **(wC)**, and **(CP)**.*

Proof. The fact that $\mathsf{F_R}$ satisfies $(\mathbf{E}_{\mathcal{C}_H})$ follows from Prop. 19 and the fact that $\mathsf{F_{SP}}$ satisfies $(\mathbf{E}_{\mathcal{C}_H})$ by Thm. 4.

The fact that $\mathsf{F_R}$ satisfies $(\mathbf{E}_{\mathcal{C}_e})$ naturally follows from the fact that, for every $\mathsf{f} \in \mathsf{F_R}$, $P \in \mathcal{C}_e$, and $V \subseteq \mathcal{A}$, $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ implies, by definition of $\mathsf{F_R}$, that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$, which suffices by Lemma 1.

For **(SE)**, just note that the definition of $\mathsf{F_R}$ is completely semantic, in the sense that every $\mathsf{f} \in \mathsf{F_R}$ returns the same set of HT-models for identical sets of HT-models.

For **(PP)**, let $\mathsf{f} \in \mathsf{F_R}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$. We prove the result by contraposition. Suppose $\mathsf{f}(P,V) \not\models_{\mathsf{HT}} P'$, for $P' \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(P') \subseteq \mathcal{A}\backslash V$. Then, there is $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ such that $\langle X, Y\rangle \notin \mathcal{HT}(P')$. From the former we have, by Thm. 10, that $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$. Therefore, there is $A^* \in Rel^Y_{\langle P,V\rangle}$ such that $X \in R^{Y,A^*}_{\langle P,V\rangle}$, i.e., there is $A' \subseteq A^*$ such that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$. Since $\langle X, Y\rangle \notin \mathcal{HT}(P')$ and $\mathcal{A}(P') \subseteq \mathcal{A}\backslash V$, we have, by Lemma 7, that $\langle X \cup A', Y \cup A^*\rangle \notin \mathcal{HT}(P')$. Therefore, $P \not\models_{\mathsf{HT}} P'$.

For **(SI)**, let $\mathsf{f} \in \mathsf{F_R}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$. For each $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we show $\mathsf{f}(P,V) \cup R \equiv \mathsf{f}(P \cup R, V)$, i.e., $\mathcal{HT}(\mathsf{f}(P,V) \cup R) = \mathcal{HT}(\mathsf{f}(P \cup R, V))$.

"$\subseteq$": First let $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. Then, $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle X, Y\rangle \in \mathcal{HT}(R)$. From the former we have, by Thm. 10, that there is $A^* \in Rel^Y_{\langle P,V\rangle}$ and $A' \subseteq A^*$ such that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$. From the latter, and since $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we have, by Lemma 7, that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(R)$. Therefore, $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$. Also, $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$ by Lemma 1, and, thus, $A^* \in Rel^Y_{\langle P \cup R,V\rangle}$ by Lemma 9. Since $\langle X \cup A', Y \cup A^*\rangle \in$

$\mathcal{HT}(P \cup R)$, we have that $X = (X \cup A')\backslash V \in R^{Y,A^*}_{\langle P\cup R,V\rangle}$, thus implying that $X \in \bigcup \mathcal{R}^Y_{\langle P\cup R,V\rangle}$. By Thm. 10, we finally have that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P \cup R, V))$.

"$\supseteq$": For the reverse inclusion, let $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P \cup R, V))$. Then, by Thm. 10, we have $X \in \bigcup \mathcal{R}^Y_{\langle P\cup R,V\rangle}$, i.e., there is $A^* \in Rel^Y_{\langle P\cup R,V\rangle}$ and $A' \subseteq A^*$ such that $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$. Then, $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$ and $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(R)$. By Lemma 7, we have that $\langle X, Y\rangle \in \mathcal{HT}(R)$. Also, by Lemma 9, $A^* \in Rel^Y_{\langle P,V\rangle}$. Now, since $\langle X \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$ and $A^* \in Rel^Y_{\langle P,V\rangle}$, we have, by Thm. 10, that $\langle X, Y\rangle = \langle (X \cup A')\backslash V, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. Since $\langle X, Y\rangle \in \mathcal{HT}(R)$, we can conclude that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V) \cup R)$.

For **(sC)**, let $\mathsf{f} \in \mathsf{F_R}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$. Now, let $Y \in \mathcal{AS}(\mathsf{f}(P, V))$. Then, we have that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, which, by Thm. 10, implies that $Rel^Y_{\langle P,V\rangle} \neq \emptyset$, i.e., there is $A^* \in Rel^Y_{\langle P,V\rangle}$. Then, $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(P)$. Suppose there is $X' \subset Y \cup A^*$ such that $\langle X', Y \cup A^*\rangle \in \mathcal{HT}(P)$. Thus, $X'\backslash V \in R^{Y,A^*}_{\langle P,V\rangle}$. Since $A^* \in Rel^Y_{\langle P,V\rangle}$, we have that $X'\backslash V \subset Y$. By Thm. 10, we have that $\langle X'\backslash V, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, which, together with $X'\backslash V \subset Y$, contradicts $Y \in \mathcal{AS}(\mathsf{f}(P, V))$. Hence, there is no $X' \subset Y \cup A^*$ such that $\langle X', Y \cup A^*\rangle \in \mathcal{HT}(P)$, thus $Y \cup A^* \in \mathcal{AS}(P)$, and therefore $Y \in \mathcal{AS}(P)_{\|V}$.

For the negative results for $(\mathbf{E}_{\mathcal{C}_n})$ and $(\mathbf{E}_{\mathcal{C}_d})$, consider the normal (and thus also disjunctive) program $P = \{a \leftarrow not\, p; p \leftarrow not\, a\}$, and $V = \{p\}$. We have that $\mathcal{HT}_V(P) = \{\langle p, p\rangle, \langle a, a\rangle\}$. Thus, for every $\mathsf{f} \in \mathsf{F_R}$, we have, by definition of $\mathsf{F_R}$, that $\mathcal{HT}(\mathsf{f}(P, V)) = \{\langle \emptyset, \emptyset\rangle, \langle a, a\rangle\}$. Then, Lemma 3 and Lemma 2 guarantee that $\mathsf{f}(P, V)$ cannot be a normal nor a disjunctive program.

For the negative result for **(wE)**, let $\mathcal{A} = \{a, b, p\}$, $V = \{p\}$, and P_1, P_2 programs such that $\mathcal{HT}(P_1) = \{\langle abp, abp\rangle, \langle a, ab\rangle, \langle ab, ab\rangle\}$ and $\mathcal{HT}(P_2) = \{\langle abp, abp\rangle\}$. Lemma 1 guarantees that such programs exist. In this case, since $\mathcal{AS}(P_1) = \mathcal{AS}(P_2) = \{\{a, b, p\}\}$, P_1 and P_2 are weakly equivalent. By Thm. 10, we have, for every $\mathsf{f} \in \mathsf{F_R}$, that $\mathcal{HT}(\mathsf{f}(P_1, V)) = \{\langle a, ab\rangle, \langle ab, ab\rangle\}$, and $\mathcal{HT}(\mathsf{f}(P_2, V)) = \{\langle ab, ab\rangle\}$. Thus, $\{a, b\} \notin \mathcal{AS}(\mathsf{f}(P_1, V))$, but $\{a, b\} \in \mathcal{AS}(\mathsf{f}(P_2, V))$, i.e., $\mathsf{f}(P_1, V)$ and $\mathsf{f}(P_2, V)$ are not weakly equivalent.

For the negative result for **(W)**, let $\mathcal{A} = \{a, p\}$, $V = \{p\}$, and P a program such that $\mathcal{HT}(P) = \{\langle ap, ap\rangle, \langle a, ap\rangle\}$. Lemma 1 guarantees that such program exists. In this case, $Rel^{\emptyset}_{\langle P,V\rangle} = \emptyset$ and $Rel^{\{a\}}_{\langle P,V\rangle} = \emptyset$. By Thm. 10 we have, for every $\mathsf{f} \in \mathsf{F_R}$, that $\mathcal{HT}(\mathsf{f}(P, V)) = \emptyset$. Thus, $\mathcal{HT}(P) \not\subseteq \mathcal{HT}(\mathsf{f}(P, V))$.

To prove the negative result for **(wC)**, consider $\mathcal{A} = \{a, b, p\}$ and $V = \{p\}$. Let P be a program such that $\mathcal{HT}(P) = \{\langle abp, abp\rangle, \langle b, ab\rangle, \langle ab, ab\rangle\}$. Lemma 1 guarantees that such programs exists. In this case, we have that $\{a, b\} \in \mathcal{AS}(P)_{\|V}$ since $\{a, b, p\} \in \mathcal{AS}(P)$. By Thm. 10 we have that $\langle b, ab\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$. But this means that $\{a, b\} \notin \mathcal{AS}(\mathsf{f}(P, V))$. Therefore, $\mathcal{AS}(P)_{\|V} \not\subseteq \mathcal{AS}(\mathsf{f}(P, V))$.

The negative result for **(CP)** follows directly from the fact that **(wC)** is not satisfied, and by Prop. 1 in [49]. ■

Theorem 11. *Alg. 3 terminates and* $\mathsf{f_M} \in \mathsf{F_M}$.

Proof. Termination follows straightforwardly from the fact that we consider a finite signature $\mathcal{A}$, and thus Y and X in lines 3, 8, and 13 are finite, and in each possible case of augmentation of P' only one rule is added.

Regarding correctness, i.e., $\mathsf{f}_{\mathsf{M}} \in \mathsf{F}_{\mathsf{M}}$, let $M = \mathsf{f}(P, V)$ for $\mathsf{f} \in \mathsf{F}_{\mathsf{M}}$, $P \in \mathcal{C}(\mathsf{f})$ and $V\mathcal{A}(P)$. Note that the rules added in lines 5, 9, and 14 are precisely those rules in the definition of P_M that, by Prop. 8, correspond to the countermodels of the HT-models of the characterization of the forgetting result for SP-Forgetting (cf. Def. 8). Then, by Prop. 8, we have that $\mathcal{HT}(P_M) = M$. ■

Proposition 21. *Let $P \in \mathcal{C}_H$ and $V \subseteq \mathcal{A}$. Then, for every $\mathsf{f} \in (\mathsf{F}_{\mathsf{SP}} \cup \mathsf{F}_{\mathsf{R}})$ and $\mathsf{f}' \in \mathsf{F}_{\mathsf{M}}$ we have that $\mathsf{f}(P, V) \equiv \mathsf{f}'(P, V)$.*

Proof. Since we are assuming that P is a Horn program, we can use Prop. 18 to conclude that $\mathcal{R}^Y_{\langle P,V\rangle}$ has at most one element, for every $Y \subseteq \mathcal{A}\backslash V$. Since in this case $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} = \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$, the result follows immediately from the definitions of F_{SP} and F_{M}, together with the alternative characterization of F_{R} given in Thm. 10. ■

Proposition 22. *Let P be a program and $V \subseteq \mathcal{A}$, such that $\langle P, V\rangle$ does not satisfy Ω. Then, for every $\mathsf{f} \in \mathsf{F}_{\mathsf{SP}}$ and $\mathsf{f}' \in \mathsf{F}_{\mathsf{M}}$ we have that $\mathsf{f}(P, V) \equiv \mathsf{f}'(P, V)$.*

Proof. Since $\langle P, V\rangle$ does not satisfy Ω then, for every $Y \subseteq \mathcal{A}\backslash V$, $\mathcal{R}^Y_{\langle P,V\rangle}$ is either empty or has a least element. The result then follows immediately from the definition of F_{M}. ■

Proposition 23. F_{M} *satisfies* **(E**$_{\mathcal{C}_H}$**)**, **(E**$_{\mathcal{C}_e}$**)**, **(wE)**, **(SE)**, **(PP)**, **(sC)**, **(wC)**, **(CP)**, *but not* **(E**$_{\mathcal{C}_n}$**)**, **(E**$_{\mathcal{C}_d}$**)**, **(W)**, *and* **(SI)**.

Proof. The fact that F_{M} satisfies **(E**$_{\mathcal{C}_H}$**)** follows from Prop.19 and the fact that F_{SP} satisfies **(E**$_{\mathcal{C}_H}$**)** by Thm. 4.

The fact that F_{M} satisfies **(E**$_{\mathcal{C}_e}$**)** naturally follows from the fact that, for every $\mathsf{f} \in \mathsf{F}_{\mathsf{M}}$, $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$ implies, by construction, that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$, which suffices by Lemma 1.

F_{M} satisfies **(SE)** since the coinciding sets of HT-models of two strongly equivalent programs P and P' yield precisely the same set of HT-models when forgetting about some V from P and P'.

Regarding **(PP)**, let $\mathsf{f} \in \mathsf{F}_{\mathsf{M}}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$. For each $P' \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(P') \subseteq \mathcal{A}\backslash V$, we prove the result by contraposition. Suppose $\mathsf{f}(P, V) \not\models_{\mathsf{HT}} P'$. Then, there is $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P, V))$ such that $\langle X, Y\rangle \notin \mathcal{HT}(P')$. By definition of F_{M}, we have two cases: *a)* $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element: in this case, $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle} = R^{Y,A^*}_{\langle P,V\rangle}$, for some $A^* \in Rel^Y_{\langle P,V\rangle}$. *b)* $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element: in this case, $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$, which means that there is some $A^* \in Rel^Y_{\langle P,V\rangle}$ such that $X \in R^{Y,A^*}_{\langle P,V\rangle}$. For both cases, there is $A \subseteq A^*$ such that $\langle X \cup A, Y \cup A^*\rangle \in \mathcal{HT}(P)$. Since $\langle X, Y\rangle \notin \mathcal{HT}(P')$ and $\mathcal{A}(P') \subseteq \mathcal{A}\backslash V$, we have, by Lemma 7, that $\langle X \cup A, Y \cup A^*\rangle \notin \mathcal{HT}(P')$. Therefore, $P \not\models_{\mathsf{HT}} P'$.

For **(sC)**, let $\mathsf{f} \in \mathsf{F_M}$, $P \in \mathcal{C}(\mathsf{f})$, $V \subseteq \mathcal{A}$, and $Y \in \mathcal{AS}(\mathsf{f}(P,V))$. Then, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. By definition of $\mathsf{F_M}$, we have that $\mathcal{R}^Y_{\langle P,V\rangle}$ has only the element $\{Y\}$, which is necessarily the least element of $\mathcal{R}^Y_{\langle P,V\rangle}$. Therefore, there is some $A \in Rel^Y_{\langle P,V\rangle}$ such that $R^{Y,A}_{\langle P,V\rangle} = \{Y\}$, i.e., $\langle Y \cup A, Y \cup A\rangle \in HT(P)$ and there is no $X \subset Y \cup A$ such that $\langle X, Y \cup A\rangle \in HT(P)$. This implies that $Y \cup A \in \mathcal{AS}(P)$, and therefore $Y \in \mathcal{AS}(P)_{\|V}$.

For **(wC)**, let $\mathsf{f} \in \mathsf{F_M}$, $P \in \mathcal{C}(\mathsf{f})$, $V \subseteq \mathcal{A}$, and $Y \in \mathcal{AS}(P)_{\|V}$. Then, $Y \cup A \in \mathcal{AS}(P)$, for some $A \subseteq V$. Suppose $A \notin Rel^Y_{\langle P,V\rangle}$. Then, there is $A' \subset A$ such that $\langle Y \cup A', Y \cup A\rangle \in \mathcal{HT}(P)$, which contradicts $Y \cup A \in \mathcal{AS}(P)$. Therefore, $A \in Rel^Y_{\langle P,V\rangle}$. This means that $R^{Y,A}_{\langle P,V\rangle} = \{Y\}$, and thus $R^{Y,A}_{\langle P,V\rangle}$ is the least element of $\mathcal{R}^Y_{\langle P,V\rangle}$. In this case, $\bigcap \mathcal{R}^Y_{\langle P,V\rangle} = \{Y\}$. By definition of $\mathsf{F_M}$, we have that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$, which means that $Y \in \mathcal{AS}(\mathsf{f}(P,V))$.

By Prop. 1 of [49], we know that **(sC)** and **(wC)** together imply **(CP)**. Therefore, we can conclude that $\mathsf{F_M}$ satisfies **(CP)**. Then, since by Prop. 1 of [49] **(CP)** implies **(wE)**, we know that $\mathsf{F_M}$ satisfies **(wE)**.

For the negative results for $(\mathbf{E}_{\mathcal{C}_n})$ and $(\mathbf{E}_{\mathcal{C}_d})$, consider the normal (and therefore also disjunctive) program $P = \{a \leftarrow not\, p; p \leftarrow not\, a\}$, and $V = \{p\}$. The set of HT-models of P is $\mathcal{HT}(P) = \{\langle p,p\rangle, \langle a,a\rangle, \langle ap,ap\rangle, \langle a,ap\rangle, \langle p,ap\rangle, \langle \emptyset, ap\rangle\}$. In this case we have $Rel^{\{a\}}_{\langle P,V\rangle} = \{\emptyset\}$ and $Rel^{\emptyset}_{\langle P,V\rangle} = \{\{p\}\}$. Note that $\{p\} \notin Rel^{\{a\}}_{\langle P,V\rangle}$ due to the HT-model $\langle a, ap\rangle \in \mathcal{HT}(P)$. We then have that $\mathcal{R}^{\{a\}}_{\langle P,V\rangle} = \{\{a\}\}$ and $\mathcal{R}^{\emptyset}_{\langle P,V\rangle} = \{\emptyset\}$. Since each of these sets has a least element, we have, by definition of $\mathsf{F_M}$, that $\mathcal{HT}(\mathsf{f}(P,V)) = \{\langle \emptyset, \emptyset\rangle, \langle a,a\rangle\}$, for every $\mathsf{f} \in \mathsf{F_M}$. Then, Lemma 3 and Lemma 2 guarantee that $\mathsf{f}(P,V)$ cannot be a normal nor a disjunctive program.

For the negative result for **(W)**, let $\mathcal{A} = \{a,p\}$, $V = \{p\}$, and P a program such that $\mathcal{HT}(P) = \{\langle ap,ap\rangle, \langle a,ap\rangle\}$. Lemma 1 guarantees that such program exists. In this case, $Rel^{\emptyset}_{\langle P,V\rangle} = \emptyset$ and $Rel^{\{a\}}_{\langle P,V\rangle} = \emptyset$. Therefore, $\mathcal{R}^{\emptyset}_{\langle P,V\rangle} = \emptyset$ and $\mathcal{R}^{\{a\}}_{\langle P,V\rangle} = \emptyset$. By definition of $\mathsf{F_M}$, we have that $\mathcal{HT}(\mathsf{f}(P,V)) = \emptyset$, for every $\mathsf{f} \in \mathsf{F_M}$. Thus, $\mathcal{HT}(P) \not\subseteq \mathcal{HT}(\mathsf{f}(P,V))$.

The negative result for **(SI)** follows from the fact, shown in Prop. 1 of [49], that properties **(CP)** and **(SI)** together are equivalent to property **(SP)**. By Thm. 1, there is no class of forgetting operators over $\mathcal{C}_e$ that satisfies **(SP)**. In particular, $\mathsf{F_M}$ does not satisfy **(SP)**. Therefore, since $\mathsf{F_M}$ satisfies **(CP)**, we can conclude that it cannot satisfy **(SI)**. ■

Theorem 12. $\mathsf{F_{SP}}$ *satisfies* **(wSP)**, *whereas* $\mathsf{F_R}$ *and* $\mathsf{F_M}$ *satisfy* **(sSP)**.

Proof. First we prove that $\mathsf{F_{SP}}$ satisfies **(wSP)**.

Let $\mathsf{f} \in \mathsf{F_{SP}}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$. We aim to prove that $\mathcal{AS}(P \cup R)_{\|V} \subseteq \mathcal{AS}(\mathsf{f}(P,V) \cup R)$, for $R \in \mathcal{C}(\mathsf{f})$ such that $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$. Let $Y \in \mathcal{AS}(P \cup R)_{\|V}$. Then, there is $A^* \subseteq V$ such that $Y \cup A^* \in \mathcal{AS}(P \cup R)$. Therefore, $\langle Y \cup A^*, Y \cup$

$A^*\rangle \in \mathcal{HT}(P \cup R)$ and there is no $X \subset Y \cup A^*$ such that $\langle X, Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$. We then have that $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(P)$ and $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$ as well as $A^* \in Rel^Y_{\langle P \cup R, V\rangle}$. Then, by Lemma 9, $A^* \in Rel^Y_{\langle P,V\rangle}$, and we have that $\mathcal{R}^Y_{\langle P,V\rangle}$ is non-empty and therefore $Y \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. By definition of $\mathsf{F_{SP}}$, we have that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$ and $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$, we have, by Lemma 7, that $\langle Y, Y\rangle \in \mathcal{HT}(R)$. Therefore, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$.

Suppose there is $X^* \subset Y$ such that $\langle X^*, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. In particular, $\langle X^*, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle X^*, Y\rangle \in \mathcal{HT}(R)$. By definition of $\mathsf{F_{SP}}$, $X^* \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. In particular $X^* \in R^{Y,A^*}_{\langle P,V\rangle}$. This means that there is $A' \subseteq A^*$ such that $\langle X^* \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P)$. Since $\langle X^*, Y\rangle \in \mathcal{HT}(R)$, and using again Lemma 7, we can conclude that $\langle X^* \cup A', Y \cup A^*\rangle \in \mathcal{HT}(R)$. Therefore, $\langle X^* \cup A', Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$. The fact that $X^* \cup A' \subset Y \cup A^*$ then contradicts $Y \cup A^* \in \mathcal{AS}(P \cup R)$. Hence, there is no $X^* \subset Y$ such that $\langle X^*, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$, thus $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$.

We now prove that $\mathsf{F_R}$ satisfies **(sSP)**.

Let $\mathsf{f} \in \mathsf{F_R}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$. We aim to prove that $\mathcal{AS}(\mathsf{f}(P,V) \cup R) \subseteq \mathcal{AS}(P \cup R)_{\|V}$, for $R \in \mathcal{C}(\mathsf{f})$ such that $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$. Let $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. Therefore, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$ and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. We then have that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle Y, Y\rangle \in \mathcal{HT}(R)$. The former implies that $Rel^Y_{\langle P,V\rangle} \neq \emptyset$, i.e., there is some $A^* \in Rel^Y_{\langle P,V\rangle}$. Then, $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(P)$. Since $\langle Y, Y\rangle \in \mathcal{HT}(R)$, we have, by Lemma 7, that $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$. Therefore, $\langle Y \cup A^*, Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$.

Suppose there is $X^* \subset Y \cup A^*$ such that $\langle X^*, Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$. Then, $\langle X^*, Y \cup A^*\rangle \in \mathcal{HT}(P)$ and $\langle X^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$. From $\langle X^*, Y \cup A^*\rangle \in \mathcal{HT}(P)$ and $A^* \in Rel^Y_{\langle P,V\rangle}$ we can conclude that $X^*\backslash V \in R^{Y,A^*}_{\langle P,V\rangle}$. Therefore, $X^*\backslash V \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$. By Thm. 10, we have that $\langle X^*\backslash V, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle X^*, Y \cup A^*\rangle \in \mathcal{HT}(R)$, we have, by Lemma 7, that $\langle X^*\backslash V, Y\rangle \in \mathcal{HT}(R)$. Therefore, $\langle X^*\backslash V, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. Moreover, since $A^* \in Rel^Y_{\langle P,V\rangle}$, we must have that $(X^*\backslash V) \subset Y$, which contradicts $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. Hence, there is no $X^* \subset Y \cup A^*$ such that $\langle X^*, Y \cup A^*\rangle \in \mathcal{HT}(P \cup R)$, thus $Y \cup A^* \in \mathcal{AS}(P \cup R)$ and therefore $Y \in \mathcal{AS}(P \cup R)_{\|V}$.

Finally, we prove that $\mathsf{F_M}$ satisfies **(sSP)**.

Let $\mathsf{f} \in \mathsf{F_M}$, $P \in \mathcal{C}\mathsf{f}$ and $V \subseteq \mathcal{A}$. We aim to prove that $\mathcal{AS}(\mathsf{f}(P,V) \cup R) \subseteq \mathcal{AS}(P \cup R)_{\|V}$, for $R \in \mathcal{C}(\mathsf{f})$ such that $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$. Let $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. Therefore, $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$ and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. We then have that $\langle Y, Y\rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle Y, Y\rangle \in \mathcal{HT}(R)$. The former then implies that $Rel^Y_{\langle P,V\rangle} \neq \emptyset$. We then have two possible cases:

a) $\mathcal{R}^Y_{\langle P,V\rangle}$ has a least element: we choose some $A^* \in Rel^Y_{\langle P,V\rangle}$ such that $R^{Y,A^*}_{\langle P,V\rangle}$ is the least element of $\mathcal{R}^Y_{\langle P,V\rangle}$;

b) $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element: we choose some A^* from $Rel^Y_{\langle P,V\rangle}$.

For both cases, we show that $Y \cup A^* \in \mathcal{AS}(P \cup R)$ holds, from which $Y \in \mathcal{AS}(P \cup R)_{\|V}$ directly follows. First, since $A^* \in Rel^Y_{\langle P,V\rangle}$ we have that

$\langle Y \cup A^*, Y \cup A^* \rangle \in \mathcal{HT}(P)$. Since $\langle Y, Y \rangle \in \mathcal{HT}(R)$, we have, by Lemma 7, that $\langle Y \cup A^*, Y \cup A^* \rangle \in \mathcal{HT}(R)$. Therefore, $\langle Y \cup A^*, Y \cup A^* \rangle \in \mathcal{HT}(P \cup R)$.

Suppose there is $X^* \subset Y \cup A^*$ such that $\langle X^*, Y \cup A^* \rangle \in \mathcal{HT}(P \cup R)$. Then, $\langle X^*, Y \cup A^* \rangle \in \mathcal{HT}(P)$ and $\langle X^*, Y \cup A^* \rangle \in \mathcal{HT}(R)$. From $\langle X^*, Y \cup A^* \rangle \in \mathcal{HT}(P)$ and $A^* \in Rel^Y_{\langle P,V \rangle}$ we can conclude that $X^* \backslash V \in R^{Y,A^*}_{\langle P,V \rangle}$. We again consider the two possible cases:

a) $\mathcal{R}^Y_{\langle P,V \rangle}$ has a least element: since we are assuming that $R^{Y,A^*}_{\langle P,V \rangle}$ is the least element of $\mathcal{R}^Y_{\langle P,V \rangle}$, we have that $R^{Y,A^*}_{\langle P,V \rangle} = \bigcap \mathcal{R}^Y_{\langle P,V \rangle}$. Therefore, $X^* \backslash V \in \bigcap \mathcal{R}^Y_{\langle P,V \rangle}$. By definition of $\mathsf{F_M}$, we have that $\langle X^* \backslash V, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V))$;

b) $\mathcal{R}^Y_{\langle P,V \rangle}$ has no least element: then, since $X^* \backslash V \in \bigcup \mathcal{R}^Y_{\langle P,V \rangle}$, we have, by definition of $\mathsf{F_M}$, that $\langle X^* \backslash V, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V))$.

In both cases, we conclude that $\langle X^* \backslash V, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle X^*, Y \cup A^* \rangle \in \mathcal{HT}(R)$, we have, by Lemma 7, that $\langle X^* \backslash V, Y \rangle \in \mathcal{HT}(R)$. Therefore, $\langle X^* \backslash V, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. Since $A^* \in Rel^Y_{\langle P,V \rangle}$, we have $(X^* \backslash V) \subset Y$, which contradicts $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. Hence, there is no $X^* \subset Y \cup A^*$ such that $\langle X^*, Y \cup A^* \rangle \in \mathcal{HT}(P \cup R)$, thus $Y \cup A^* \in \mathcal{AS}(P \cup R)$. ■

Lemma 13. *Let* $\mathsf{f} \in \mathsf{F_R}$, $\mathsf{f}' \in \mathsf{F_M}$, $P \in \mathcal{C}(\mathsf{f}) \cap \mathcal{C}(\mathsf{f}')$, *and* $V \subseteq \mathcal{A} \backslash V$. *Then,*

$$\mathcal{HT}(\mathsf{f}'(P,V)) \subseteq \mathcal{HT}(\mathsf{f}(P,V)).$$

Proof. Let $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}'(P,V))$. By definition of $\mathsf{F_M}$, we have two cases: *a*) $\mathcal{R}^Y_{\langle P,V \rangle}$ has no least element, in which case $X \in \bigcup \mathcal{R}^Y_{\langle P,V \rangle}$, or *b*) $\mathcal{R}^Y_{\langle P,V \rangle}$ has a least element, in which case $X \in \bigcap \mathcal{R}^Y_{\langle P,V \rangle}$. In both cases, since $\bigcap \mathcal{R}^Y_{\langle P,V \rangle} \subseteq \bigcup \mathcal{R}^Y_{\langle P,V \rangle}$, we have that $X \in \bigcup \mathcal{R}^Y_{\langle P,V \rangle}$. Then, using the alternative characterization of $\mathsf{F_R}$ given in Thm. 10 we can conclude that $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. ■

Proposition 24. *Let* $P \in \mathcal{C}_e$ *be a program,* $V \subseteq \mathcal{A}$, $\mathsf{f} \in \mathsf{F_R}$, *and* $\mathsf{f}' \in \mathsf{F_M}$ *with* $\mathcal{C}(\mathsf{f}) = \mathcal{C}(\mathsf{f}') = \mathcal{C}_e$. *Then, for every* $R \in \mathcal{C}_e$ *with* $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$,

$$\mathcal{AS}(\mathsf{f}(P,V) \cup R) \subseteq \mathcal{AS}(\mathsf{f}'(P,V) \cup R).$$

Proof. Let $R \in \mathcal{C}_e$ with $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$, and $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. Then, $\langle Y, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$ and there is no $X \subset Y$ such that $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$. We then have that $\langle Y, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V))$ and $\langle Y, Y \rangle \in \mathcal{HT}(R)$. From the former we can conclude that $Rel^Y_{\langle P,V \rangle} \neq \emptyset$, and, therefore, $\langle Y, Y \rangle \in \mathcal{HT}(\mathsf{f}'(P,V) \cup R)$.

Suppose there exists $X \subset Y$ such that $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}'(P,V) \cup R)$. Then, $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}'(P,V))$ and $\langle X, Y \rangle \in \mathcal{HT}(R)$. From the former, and using Lemma 13, we can conclude that $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V))$. Since $\langle X, Y \rangle \in \mathcal{HT}(R)$, we have that $\langle X, Y \rangle \in \mathcal{HT}(\mathsf{f}(P,V) \cup R)$, which contradicts $Y \in \mathcal{AS}(\mathsf{f}(P,V) \cup R)$. We can therefore conclude that $Y \in \mathcal{AS}(\mathsf{f}'(P,V) \cup R)$. ■

Proofs for Sec. 6

Proposition 26. *Given program P, $V \subseteq \mathcal{A}$, and HT-interpretation $\langle X, Y\rangle$, deciding whether $\langle X, Y\rangle \in \mathcal{HT}_V(P)_{\|V}$ is $\mathbf{\Sigma_2^P}$-complete. Hardness holds already for disjunctive programs.*

Proof. Membership in $\mathbf{\Sigma_2^P}$ is argued as follows: we guess interpretation $Y' \sim_V Y$ and check $(X, Y') \in \mathcal{HT}_V(P)$. Since the latter check is in $\mathbf{D^P}$ (cf. Proposition 25), two **NP**-oracle calls suffice; $\mathbf{\Sigma_2^P}$-membership thus follows.

The hardness result follows quite easily from the $\mathbf{\Sigma_2^P}$-hardness of ASP consistency, cf. [60]. We recall the construction (in a slightly adapted way) here, since we will reuse it in some of the forthcoming results. The reduction maps $(2, \exists)$-QBFs $\Phi = \exists X \forall Y \phi$ with $\phi = \bigvee_{i=1}^{n}(l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$ – with negative atoms written as $\overline{a}$ – to programs

$$\begin{aligned}\mathcal{T}[X, Y, \phi] \;=\; & \{x \vee \overline{x} \leftarrow; \bot \leftarrow x, \overline{x} \mid x \in X\} \cup \\ & \{y \vee \overline{y} \leftarrow; y \leftarrow w; \overline{y} \leftarrow w; w \leftarrow y, \overline{y} \mid y \in Y\} \cup \\ & \{w \leftarrow l_{i,1}, l_{i,2}, l_{i,3} \mid 1 \leq i \leq n\} \cup \\ & \{\bot \leftarrow \mathit{not}\ w\};\end{aligned}$$

It is shown in [60] that $\mathcal{T}[X, Y, \phi]$ has an answer set iff Φ is true. Recall that M is answer set of a program P iff $\langle M, M\rangle$ is HT-model of P and for each $M' \subset M$, it holds that $M \not\models P^M$. Now let V denote all atoms occurring in $\mathcal{T}[X, Y, \phi]$. We observe that $\langle M, M\rangle$ is a V-HT model of $\mathcal{T}[X, Y, \phi]$ iff M is an answer set of $\mathcal{T}[X, Y, \phi]$. Consequently, $\langle \emptyset, \emptyset\rangle \in \mathcal{HT}_V(P)_{\|V}$ iff $\mathcal{T}[X, Y, \phi]$ has at least one answer set. This show $\mathbf{\Sigma_2^P}$-hardness. ■

Proposition 27. *Given program P, $V \subseteq \mathcal{A}$, and HT-interpretation $\langle X, Y\rangle$ with $Y \subseteq \mathcal{A}\backslash V$, deciding whether $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ is in $\mathbf{D_2^P}$.*

Proof. We have to prove that the problem can be decided via conjoining a $\mathbf{\Sigma_2^P}$- and a $\mathbf{\Pi_2^P}$-test. The $\mathbf{\Sigma_2^P}$-test is used to decide whether $\mathcal{R}^Y_{\langle P,V\rangle} \neq \emptyset$. For this, we guess $A \subseteq V$ and check that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}_V(P)$. The $\mathbf{\Pi_2^P}$-test checks that for all $A \subseteq V$, either $\langle Y \cup A, Y \cup A\rangle \notin \mathcal{HT}_V(P)$ or $\langle X, Y \cup A\rangle \in \mathcal{HT}_V(P)$. Recalling that V-HT-model checking is in $\mathbf{D^P}$ (cf. Proposition 25), the claimed membership is seen to hold. ■

Proposition 28. *Given program P, $V \subseteq \mathcal{A}$, and HT-interpretation $\langle X, Y\rangle$ with $Y \subseteq \mathcal{A}\backslash V$, deciding whether $X \in \bigcup \mathcal{R}^Y_{\langle P,V\rangle}$ if $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element, and $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$ otherwise, is in $\mathbf{\Sigma_3^P}$ and in $\mathbf{\Pi_3^P}$.*

Proof. $\mathbf{\Sigma_3^P}$-membership: The condition holds iff there exists $(X, Y') \in \mathcal{HT}_V(P)$ with $Y' \sim_V Y$, such that (i) $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element or (ii) $Y' \cap V$ provides the least element of $\mathcal{R}^Y_{\langle P,V\rangle}$. Test (i) holds iff there exist X, X', Z, Z' with $Z \neq Z'$, $Y \sim_V Z \sim_V Z'$, and $X \neq X'$, such that $\langle X, Z\rangle \in \mathcal{HT}_V(P)$, $\langle X', Z'\rangle \in$

$\mathcal{HT}_V(P)$, $\langle X', Z\rangle \notin \mathcal{HT}_V(P)$, $\langle X, Z'\rangle \notin \mathcal{HT}_V(P)$, and for each $Z'' \sim_V Y$ either $\langle X, Z''\rangle \in \mathcal{HT}_V(P)$ or $\langle X', Z''\rangle \in \mathcal{HT}_V(P)$. Test (ii) can be decided by checking whether for all X'', Y'' with $Y'' \sim_V Y$ it holds that $\langle Y'', Y''\rangle \notin \mathcal{HT}_V(P)$, $\langle X'', Y'\rangle \notin \mathcal{HT}_V(P)$, or $\langle X'', Y''\rangle \in \mathcal{HT}_V(P)$. For giving a nondeterministic procedure combining the above tests, we thus need to guess Y', X, X', Z, Z' such that for all X'', Y'', Z'' certain properties hold. Among these properties, V-HT-model checking is the most costly and in $\mathbf{D^P}$. All together this yields a $\mathbf{\Sigma_3^P}$-procedure.

$\mathbf{\Pi_3^P}$-membership: we show $\mathbf{\Sigma_3^P}$ for the complementary problem which holds if (i) $(X, Z) \notin \mathcal{HT}_V(P)$ for each $Z \sim_V Y$ or (ii) the least element of $\mathcal{R}^Y_{\langle P,V\rangle}$ does not contain X. Note that (ii) can be decided by guessing $Y' \sim_V Y$ and checking that $(Y', Y') \in \mathcal{HT}_V(P)$, $(X, Y') \notin \mathcal{HT}_V(P)$, and whether for all X' and Y'' with $Y'' \sim_V Y$, it holds that $\langle Y'', Y''\rangle \notin \mathcal{HT}_V(P)$, $\langle X', Y'\rangle \notin \mathcal{HT}_V(P)$, or $\langle X', Y''\rangle \in \mathcal{HT}_V(P)$. Since V-HT-model checking is in $\mathbf{D^P}$, (i) is a $\mathbf{\Pi_2^P}$-test which is followed by $\mathbf{\Sigma_3^P}$-procedure for (ii). Together, this yields a $\mathbf{\Sigma_3^P}$-procedure. ■

Theorem 13. *Given programs P, Q, and $V \subseteq \mathcal{A}$, deciding whether $P \equiv \mathsf{f}(Q, V)$ (for $\mathsf{f} \in \mathsf{F_R}$) is $\mathbf{\Pi_3^P}$-complete. Hardness holds already for disjunctive programs.*

Proof. We give $\mathbf{\Sigma_3^P}$-membership for the complementary problem: Guess HT-interpretation $\langle X, Y\rangle$ and check that either $(X, Y) \in \mathcal{HT}(P)$ or $(X, Y) \in \mathcal{HT}_V(P)_{\|V}$, but not both. By Lemma 26, deciding $(X, Y) \in \mathcal{HT}_V(P)_{\|V}$ is in $\mathbf{\Sigma_2^P}$; moreover, deciding $(X, Y) \in \mathcal{HT}(P)$ is tractable.

For $\mathbf{\Pi_3^P}$-hardness, we reduce from $(3, \forall)$-QSAT. In what follows, we frequently use $\overline{U} = \{\overline{u} \mid u \in U\}$. Let $\Phi = \forall Z \exists X \forall Y \phi$ with $\phi = \bigvee_{i=1}^{n}(l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$ (again, negative literals written as $\overline{a}$). We set $V = X \cup \overline{X} \cup Y \cup \overline{Y} \cup \{w\}$ and define

$$\begin{aligned} P \quad = \quad & \{z \vee \overline{z} \leftarrow; \bot \leftarrow z, \overline{z} \mid z \in Z\} \cup \\ & \{\bot \leftarrow v \mid v \in V\} \end{aligned}$$

Finally, let $Q = \mathcal{T}[Z \cup X, Y, \phi]$ with $\mathcal{T}[\cdot]$ as in the proof of Lemma 26. We claim that Φ is true iff $P \equiv \mathsf{f}(Q, V)$, i.e. $\mathcal{HT}(P) = \mathcal{HT}_V(Q)_{\|V}$.

To this end, let us compute $\mathcal{HT}(P)$ and $\mathcal{HT}_V(Q)_{\|V}$. We have

$$\mathcal{HT}(P) = \{\langle I \cup (\overline{Z} \backslash \overline{I}), I \cup (\overline{Z} \backslash \overline{I})\rangle \mid I \subseteq Z\}.$$

For Q, we first give the HT-models. To do so, we define for $I \subseteq Z$, $J \subseteq Z$, the abbreviations $[IJ] = I \cup (\overline{Z} \backslash \overline{I}) \cup J \cup (\overline{X} \backslash \overline{I})$ and $[IJ]^+ = [IJ] \cup Y \cup \overline{Y} \cup \{w\}$. We have

$$\begin{aligned} \mathcal{HT}(Q) \quad = \quad & \{\langle [IJ]^+, [IJ]^+\rangle \mid I \subseteq Z, J \subseteq X\} \cup \\ & \{\langle [IJ] \cup K \cup (\overline{Y} \backslash \overline{K}), [IJ]^+\rangle \mid \\ & \quad I \subseteq Z, J \subseteq X, K \subseteq Y, I \cup J \cup K \not\models \phi\}. \end{aligned}$$

Recall that $V = X \cup \overline{X} \cup Y \cup \overline{Y} \cup \{w\}$. We thus have

$$\begin{aligned}\mathcal{HT}_V(Q) &= \{\langle [IJ]^+, [IJ]^+\rangle \mid I \subseteq Z, J \subseteq X, \\ &\qquad \text{s.t. } \forall K \subseteq Y, I \cup J \cup K \models \phi\}.\end{aligned}$$

Finally, the projection to V yields

$$\begin{aligned}\mathcal{HT}_V(Q)_{\|V} &= \{\langle I \cup (\overline{Z}\backslash\overline{I}), I \cup (\overline{Z}\backslash\overline{I})\rangle \mid I \subseteq Z, \\ &\qquad \text{s.t. } \exists J \subseteq X \forall K \subseteq Y, I \cup J \cup K \models \phi\}.\end{aligned}$$

It is now rather easy to see that $\mathcal{HT}(P) = \mathcal{HT}_V(Q)_{\|V}$ iff Φ is true. ■

Theorem 14. *Given programs P, Q, and $V \subseteq \mathcal{A}$, deciding whether $P \equiv \mathsf{f}(Q, V)$ (for $\mathsf{f} \in \mathsf{F_{SP}}$) is $\mathbf{\Pi_3^P}$-complete. Hardness holds already for disjunctive programs.*

Proof. The proof is similar to the one of Theorem 13. We give $\mathbf{\Sigma_3^P}$-membership for the complementary problem: Guess HT-interpretation $\langle X, Y\rangle$ and check that either $(X, Y) \in \mathcal{HT}(P)$ or $(X, Y) \in \mathcal{HT}(\mathsf{f}(P, V))$ but not both. For deciding $(X, Y) \in \mathcal{HT}(\mathsf{f}(P, V))$ we need to check that $Y \subseteq \mathcal{A}\backslash V$ and $X \in \bigcap \mathcal{R}^Y_{\langle P,V\rangle}$. By Lemma 27, the latter is in $\mathbf{D_2^P}$ (which requires two calls to a $\mathbf{\Sigma_2^P}$-oracle); moreover, deciding $(X, Y) \in \mathcal{HT}(P)$ is tractable.

For $\mathbf{\Pi_3^P}$-hardness, we reduce from $(3, \forall)$-QSAT. We let $\Phi = \forall Z \exists X \forall Y \phi$ with $\phi = \bigvee_{i=1}^n (l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$, $V = X \cup \overline{X} \cup Y \cup \overline{Y} \cup \{w\}$, and programs P and Q as in the proof of Theorem 13. We claim that Φ is true iff $P \equiv \mathsf{f}(Q, V)$, i.e. $\mathcal{HT}(P) = \{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcap \mathcal{R}^Y_{\langle Q,V\rangle}\}$.

Recall that

$$\mathcal{HT}(P) = \{\langle I \cup (\overline{Z}\backslash\overline{I}), I \cup (\overline{Z}\backslash\overline{I})\rangle \mid I \subseteq Z\}.$$

and

$$\begin{aligned}\mathcal{HT}_V(Q) &= \{\langle [IJ]^+, [IJ]^+\rangle \mid I \subseteq Z, J \subseteq X, \\ &\qquad \text{s.t. } \forall K \subseteq Y, I \cup J \cup K \models \phi\}.\end{aligned}$$

with $[IJ]^+ = I \cup (\overline{Z}\backslash\overline{I}) \cup J \cup (\overline{X}\backslash\overline{I}) \cup Y \cup \overline{Y} \cup \{w\}$.

Since, we have no non-total V-HT-models, we observe that $X \in \bigcap \mathcal{R}^Y_{\langle Q,V\rangle}$ implies $X = Y$. For our setting here, $\langle I \cup (\overline{Z}\backslash\overline{I}), I \cup (\overline{Z}\backslash\overline{I})\rangle$ are thus the only candidates for being contained in $\{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcap \mathcal{R}^Y_{\langle Q,V\rangle}\}$ and it holds that $\langle I \cup (\overline{Z}\backslash\overline{I}), I \cup (\overline{Z}\backslash\overline{I})\rangle \in \{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcap \mathcal{R}^Y_{\langle Q,V\rangle}\}$ iff $\exists J \subseteq X \forall K \subseteq Y, I \cup J \cup K \models \phi$. From this observation, we get that $\mathcal{HT}(P) = \{\langle X, Y\rangle \mid Y \subseteq \mathcal{A}\backslash V \wedge X \in \bigcap \mathcal{R}^Y_{\langle Q,V\rangle}\}$ iff Φ is true. ■

Theorem 15. *Given programs P, Q, and $V \subseteq \mathcal{A}$, deciding whether $P \equiv \mathsf{f}(Q, V)$ (for $\mathsf{f} \in \mathsf{F_M}$) is $\mathbf{\Pi_3^P}$-complete. Hardness holds already for disjunctive programs.*

Proof. We give $\mathbf{\Sigma_3^P}$-membership for the complementary problem: Guess HT-interpretation $\langle X, Y\rangle$ and check that (i) $(X,Y) \in \mathcal{HT}(P)$ and $(X,Y) \notin \mathcal{HT}(\mathsf{f}(P,V))$ or (ii) $(X,Y) \notin \mathcal{HT}(P)$ and $(X,Y) \in \mathcal{HT}(\mathsf{f}(P,V))$. $(X,Y) \notin \mathcal{HT}(\mathsf{f}(P,V))$ holds if $Y \not\subseteq \mathcal{A}\backslash V$ or the condition of Lemma 28 does not hold. The latter can be done via a $\mathbf{\Sigma_3^P}$-test (due to the $\mathbf{\Pi_3^P}$-membership result of Lemma 28). Similarly, $(X,Y) \in \mathcal{HT}(\mathsf{f}(P,V))$ holds if $Y \subseteq \mathcal{A}\backslash V$ and the condition of Lemma 28 holds. The latter can be done via a $\mathbf{\Sigma_3^P}$-test, as well (cf. Lemma 28). Since both $\mathbf{\Sigma_3^P}$-tests are embedded solely in a guess, we stay within $\mathbf{\Sigma_3^P}$.

Hardness follows essentially by the same arguments as in Theorem 14, since $\mathcal{HT}_V(Q)$ only contains total HT-interpretations; thus the difference between the operators becomes immaterial in the reduction from QSAT. ■

Theorem 16. *Let P be a program over $\mathcal{A}$ and $V \subseteq \mathcal{A}$. Deciding whether $\langle P, V\rangle$ satisfies criterion Ω is $\mathbf{\Sigma_3^P}$-complete. Hardness holds already for disjunctive programs.*

Proof. Membership: We guess an interpretation Y and check that $\mathcal{R}^Y_{\langle P,V\rangle} = \{R^{Y,A}_{\langle P,V\rangle} \mid A \in Rel^Y_{\langle P,V\rangle}\}$ is (i) non-empty and (ii) has no least element. For (i) it suffices to guess Z such that $Y \sim_V Z$, and check $\langle Z, Z\rangle \in \mathcal{HT}_V(P)$; this ensures that $A = Z\backslash Y \in Rel^Y_{\langle P,V\rangle}$ and thus $R^{Y,A}_{\langle P,V\rangle} \in \mathcal{R}^Y_{\langle P,V\rangle}$. For (ii), we further guess Y', Y'', X', X'' with $Y \sim_V Y' \sim_V Y''$, $Y' \neq Y''$, $X' \neq X''$ and check that (a) $\langle X', Y'\rangle \in \mathcal{HT}_V(P)$, $\langle X'', Y''\rangle \in \mathcal{HT}_V(P)$, $\langle X'', Y'\rangle \notin \mathcal{HT}_V(P)$, $\langle X', Y''\rangle \notin \mathcal{HT}_V(P)$; and (b) for each $Z \sim_V Y$ either $\langle X', Z\rangle \in \mathcal{HT}_V(P)$ or $\langle X'', Z\rangle \in \mathcal{HT}_V(P)$. The rationale behind (a) is to identify $A' = Y'\backslash Y$ and $A'' = Y''\backslash Y$ such that $X' \in R^{Y,A'}_{\langle P,V\rangle}\backslash R^{Y,A''}_{\langle P,V\rangle}$ and $X'' \in R^{Y,A''}_{\langle P,V\rangle}\backslash R^{Y,A'}_{\langle P,V\rangle}$. Thus, $\mathcal{R}^Y_{\langle P,V\rangle}$ would have a least element, only if it contains a set $R^{Y,A}$ which neither has X' nor X'' as an element. Consequently, we can ensure that $\mathcal{R}^Y_{\langle P,V\rangle}$ has no least element if for each $A \in Rel^Y_{\langle P,V\rangle}$, $R^{Y,A}_{\langle P,V\rangle}$ contains either X' or X''; this check is performed by (b). Since V-HT-model checking is in $\mathbf{D^P}$, this yields in overall a $\mathbf{\Sigma_3^P}$-procedure.

Hardness. We reduce from $(3,\exists)$-QSAT and let $\Phi = \exists X \forall Y \exists Z \phi$ with $\phi = \bigwedge_{i=1}^n (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ (negative literals written as $\overline{a}$). W.l.o.g. we assume that X, Y, Z are all non-empty. We set $V = X \cup \overline{X} \cup Y \cup \overline{Y} \cup \{w\}$, and construct a program P_Φ over atoms $A = X \cup \overline{X} \cup Y \cup \overline{Y} \cup Y' \cup \overline{Y}' \cup Z \cup \overline{Z}$ such that (P_Φ, V) satisfies criterion Ω iff Φ is true.

Before constructing P_Φ, we give its intended HT-models. To this end, we use the folloing abbreviations, for $I \subseteq X$, $J \subseteq Y$:

- $[IJ] = I \cup (\overline{X}\backslash\overline{I}) \cup J \cup (\overline{Y}\backslash\overline{J})$
- $[IJ]^+ = [IJ] \cup Y' \cup \overline{Y}' \cup Z \cup \overline{Z}$.

We have

$$\begin{aligned}\mathcal{HT}(P_\Phi) = \; & \{\langle [IJ], [IJ]^+\rangle, \langle [IJ]^+, [IJ]^+\rangle \mid I \subseteq X, J \subseteq Y\} \cup \\ & \{\langle [IJ] \cup J' \cup (\overline{Y}'\backslash\overline{J}') \cup K \cup (\overline{Z}\backslash\overline{K}), [IJ]^+\rangle \mid \\ & \quad I \subseteq X, J \subseteq Y, K \subseteq Z, \text{ s.t. } I \cup J \cup K \models \phi\}\end{aligned}$$

We show that (P_Φ, V) satisfies Ω iff Φ is true.

Only-if: assume there is a set $M \subseteq A\backslash V$ satisfying the conditions for Ω. Then M is of the form $I \cup (\overline{X}\backslash\overline{I}) \cup Y' \cup \overline{Y}'$ for some $I \subseteq X$. Moreover, $\mathcal{R}^M_{(P_\Phi,V)}$ cannot contain the set $\{[IJ]\backslash V, M\}$ (i.e. the set stemming solely from the ϕ-independent tuples from $\mathcal{HT}(P_\Phi)$), since then it would have a least element. But then, by the definition of the HT-models of P_Φ, we know that for each $J \subseteq Y$ there exists an HT-model of P_Φ of the form $\langle [IJ] \cup J' \cup (\overline{Y}'\backslash\overline{J}') \cup K \cup (\overline{Z}\backslash\overline{K}), [IJ]^+\rangle$, witnessing that for each $J \subseteq Y$ there exists a $K \subseteq Z$, such that $I \cup J \cup K \models \phi$. It follows that Φ is true.

if: assume Φ is true and let $I \subseteq X$ be a model of $\forall Y \exists Z \phi$. We show that $M = I \cup (X'\backslash I) \cup Y' \cup \overline{Y}'$ satisfies the conditions for Ω. Since Φ is true there is, for each $J \subseteq Y$, at least one $K \subseteq Z$ making ϕ true. From this we obtain that $\mathcal{R}^M_{(P_\Phi,V)}$ is of the form $\{[IJ]\backslash V, I \cup (\overline{X}\backslash\overline{I}) \cup J' \cup (\overline{Y}'\backslash\overline{J}'), M \mid J \subseteq Y\}$ which is clearly non-empty (recall that $Y \neq \emptyset$) and does not have a least element.

We finally give the required polynomial-time construction of P_Φ, given Φ. P_Φ contains the following rules:

$$\begin{aligned}&\{a \vee \overline{a} \leftarrow;\ \bot \leftarrow a, \overline{a} \mid a \in X \cup Y\} \\ &\{\bot \leftarrow not\, a;\ \bot \leftarrow not\, \overline{a} \mid a \in Y' \cup Z\} \\ &\{a \vee \overline{a} \leftarrow b;\ a \vee \overline{a} \leftarrow \overline{b} \mid a, b \in Y' \cup Z\} \\ &\{a \leftarrow b, \overline{b}';\ a \leftarrow \overline{b}, b';\ \overline{a} \leftarrow b, \overline{b}';\ \overline{a} \leftarrow \overline{b}, b' \mid a \in Y' \cup Z, b \in Y\} \\ &\{a \leftarrow l_{i,1}, l_{i,2}, l_{i,3};\ \overline{a} \leftarrow l_{i,1}, l_{i,2}, l_{i,3} \mid a \in Y' \cup Z, 1 \leq i \leq n\}\end{aligned}$$

It can be checked that $\mathcal{HT}(P_\Phi)$ behaves as intended. ■

Proofs for Sec. 7

Corollary 2. *Let f_{Sas} be the operator defined in [45] for F_{Sas} and P a program for which f_{Sas} is defined. For every $\mathsf{f} \in \mathsf{F}_{\mathsf{SP}}$, if a single atom p is p-forgettable from P, then $\mathcal{HT}(\mathsf{f}(P, \{p\})) = \mathcal{HT}(\mathsf{f}_{Sas}(P, \{p\}))$.*

Proof. The result follows easily from Proposition 5, the fact that if p is p-forgettable from P then $\langle P, \{p\}\rangle$ does not satisfy Ω, and the fact that f_{Sas} satisfies $\mathbf{(SP)}_{\langle P,\{p\}\rangle}$. ■

Proposition 29. *Let P be a program over $\mathcal{A}$. Then,*

$$Mod(P) = \mathcal{AS}(P^*)$$

Proof. First of all, it is obvious that $\mathcal{AS}(P^*) \subseteq Mod(P)$ since $A \in \mathcal{AS}(P^*)$ is a classical model P^* and therefore also a classical model of P.

For the reverse inclusion, just take $A \in Mod(P)$. Note that $P^{*A} = P^A \cup \{a \leftarrow | \ a \in A\}$. Since A is a model of P, it is also a model of P^A. Clearly, A is also a model of $\{a \leftarrow | \ a \in A\}$, and therefore a model of $P^A \cup \{a \leftarrow | \ a \in A\}$. Since no strict subset of A can satisfy $\{a \leftarrow | \ a \in A\}$, A is the minimal model of $P^A \cup \{a \leftarrow | \ a \in A\}$, therefore $A \in \mathcal{AS}(P^*)$. ■

Proposition 30. *Let P be a program over $\mathcal{A}$. Then, for every $V \subseteq \mathcal{A}$, we have that $\langle P^*, V\rangle$ does not satisfy Ω.*

Proof. We start by proving that all HT-models of P^* are total. Let $\langle X, Y\rangle$ be an HT-model of P^*. Then, by definition, $Y \models P^*$ and $X \models (P^*)^Y$. Since $(P^*)^Y = P^Y \cup \{a \leftarrow | \ a \in Y\}$ and since $X \models (P^*)^Y$, we have that in particular $X \models \{a \leftarrow | \ a \in Y\}$. This means that $Y \subseteq X$. Since, by definition of HT-interpretation, $X \subseteq Y$, we can conclude that $X = Y$.

Now, since we already proved that all HT-models of P^* are total, we have that, for each $Y \subseteq \mathcal{A}\backslash V$ and $A \subseteq V$, the set $R^{Y,A}_{\langle P^*,V\rangle}$ is either empty or the set $\{Y\}$. This means that, for each $Y \subseteq \mathcal{A}\backslash V$, $\mathcal{R}^Y_{\langle P^*,V\rangle}$ is either empty or the set $\{\{Y\}\}$. There is therefore no $Y \subseteq \mathcal{A}\backslash V$ for which $\mathcal{R}^Y_{\langle P^*,V\rangle}$ is non-empty and has no least element, meaning that $\langle P^*, V\rangle$ does not satisfy Ω. ■

Proposition 31. *Let P be a program over $\mathcal{A}$ and $V \subseteq \mathcal{A}$. Then, for every $\mathsf{f} \in \mathsf{F_{SP}}$, we have that the following conditions hold*

- $\mathcal{HT}(\mathsf{f}(P^*, V)) = \{\langle T, T\rangle \mid T \in Mod(\mathsf{f}_{CL}(P, V))\}$
- $\mathcal{AS}(\mathsf{f}(P^*, V)) = Mod(\mathsf{f}_{CL}(P, V))$.

Proof. We start by proving that all HT-models of P^* are total. Let $\langle X, Y\rangle$ be an HT-model of P^*. Then, by definition, $Y \models P^*$ and $X \models (P^*)^Y$. Since $(P^*)^Y = P^Y \cup \{a \leftarrow | \ a \in Y\}$ and since $X \models (P^*)^Y$, we have that in particular $X \models \{a \leftarrow | \ a \in Y\}$. This means that $Y \subseteq X$. Since, by definition of HT-interpretation, $X \subseteq Y$, we can conclude that $X = Y$. Now, since we already proved that all HT-models of P^* are total, we have that, for each $Y \subseteq \mathcal{A}\backslash V$ and $A \subseteq V$, the set $R^{Y,A}_{\langle P^*,V\rangle}$ is either empty or the set $\{Y\}$, implying that $\mathcal{R}^Y_{\langle P^*,V\rangle}$ is either empty or the set $\{\{Y\}\}$. Given this, since $\mathsf{f} \in \mathsf{F_{SP}}$ and taking into account the definition of $\mathsf{F_{SP}}$, we can conclude that $\mathcal{HT}(\mathsf{f}(P^*, V)) = \{\langle Y, Y\rangle \mid \mathcal{R}^Y_{\langle P^*,V\rangle} \neq \emptyset\}$. Note just that $\mathcal{R}^Y_{\langle P^*,V\rangle} \neq \emptyset$ iff there exists $A \in Rel^Y_{\langle P^*,V\rangle}$ such that $\langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P^*)$. This means that $Y \cup A \models P$, thus $Y \cup A \in Mod(P)$, and by definition of classical forgetting, $Y \in Mod(\mathsf{f}_{CL}(P, V))$. Then, we can conclude that $\mathcal{HT}(\mathsf{f}(P^*, V)) = \{\langle Y, Y\rangle \mid Y \in Mod(\mathsf{f}_{CL}(P, V))\}$.

The fact that $\mathcal{AS}(\mathsf{f}(P^*, V)) = Mod(\mathsf{f}_{CL}(P, V))$ follows immediately from the already proved fact $\mathcal{HT}(\mathsf{f}(P^*, V)) = \{\langle T, T\rangle \mid T \in Mod(\mathsf{f}_{CL}(P, V))\}$. ■

References

[1] R. Gonçalves, M. Knorr, J. Leite, You can't always forget what you want: on the limits of forgetting in answer set programming, in: G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), IOS Press, 2016, pp. 957–965.

[2] 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 17 (5-6) (2017) 837–854, Special Issue for the Proceedings of the 33rd International Conference on Logic Programming, (ICLP).

[3] H. Ebbinghaus, Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie, Duncker & Humblot, Leipzig, 1885.

[4] P. Shrestha, Meaning and causes of forgetting, Psychstudy, November 17 (2017).

[5] R. L. Davis, Y. Zhong, The biology of forgetting - a perspective, Neuron 95 (3) (2017) 490–503.

[6] B. A. Richards, P. W. Frankland, The persistence and transience of memory, Neuron 94 (6) (2017) 1071–1084.

[7] A. Kluge, N. Gronau, Intentional forgetting in organizations: The importance of eliminating retrieval cues for implementing new routines, Front. Psychol. 01 February.

[8] European Parliament, General data protection regulation, Official Journal of the European Union L119/59.
URL `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=OJ:L:2016:119:TOC`

[9] C. Beierle, I. J. Timm, Intentional forgetting: An emerging field in ai and beyond, KI 33 (1) (2019) 5–8.

[10] T. Eiter, G. Kern-Isberner, A brief survey on forgetting from a knowledge representation and reasoning perspective, KI 33 (1) (2019) 9–33.

[11] F. Lin, R. Reiter, How to progress a database, Artificial Intelligence 92 (1-2) (1997) 131–167.

[12] Y. Liu, X. Wen, On the progression of knowledge in the situation calculus, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, IJCAI/AAAI, 2011, pp. 976–982.

[13] D. Rajaratnam, H. J. Levesque, M. Pagnucco, M. Thielscher, Forgetting in action, in: C. Baral, G. De Giacomo, T. Eiter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014, AAAI Press, 2014, pp. 498–507.

[14] J. Lang, P. Liberatore, P. Marquis, Propositional independence: Formula-variable independence and forgetting, Journal of Artificial Intelligence Research 18 (2003) 391–443.

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

[16] T. Eiter, K. Wang, Semantic forgetting in answer set programming, Artificial Intelligence 172 (14) (2008) 1644–1672.

[17] J. Lang, P. Marquis, Reasoning under inconsistency: A forgetting-based approach, Artificial Intelligence 174 (12-13) (2010) 799–823.

[18] J. P. Delgrande, K. Wang, A syntax-independent approach to forgetting in disjunctive logic programs, in: B. Bonet, S. Koenig (Eds.), Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA, AAAI Press, 2015, pp. 1482–1488.

[19] Z. Wang, K. Wang, R. W. Topor, J. Z. Pan, Forgetting for knowledge bases in DL-Lite, Annals of Mathematics and Artificial Intelligence 58 (1-2) (2010) 117–151.

[20] R. Kontchakov, F. Wolter, M. Zakharyaschev, Logic-based ontology comparison and module extraction, with an application to dl-lite, Artificial Intelligence 174 (15) (2010) 1093–1141.

[21] B. Konev, M. Ludwig, D. Walther, F. Wolter, The logical difference for the lightweight description logic EL, Journal of Artificial Intelligence Research 44 (2012) 633–708.

[22] B. Konev, C. Lutz, D. Walther, F. Wolter, Model-theoretic inseparability and modularity of description logic ontologies, Artificial Intelligence 203 (2013) 66–103.

[23] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: Procs. of ICLP, Seattle, Washington, 1988.

[24] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (3-4) (1991) 365–385.

[25] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. T. Schneider, Potassco: The potsdam answer set solving collection, AI Commun. 24 (2) (2011) 107–124.

[26] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (3) (2006) 499–562.

[27] C. I. Lewis, A survey of symbolic logic, University of California Press, 1918, republished by Dover, 1960.

[28] W. W. Bledsoe, L. M. Hines, Variable elimination and chaining in a resolution-based prover for inequalities, in: W. Bibel, R. A. Kowalski (Eds.), 5th Conference on Automated Deduction, Les Arcs, France, July 8-11, 1980, Proceedings, Vol. 87 of LNCS, Springer, 1980, pp. 70–87.

[29] J. Larrosa, Boosting search with variable elimination, in: R. Dechter (Ed.), Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings, Vol. 1894 of LNCS, Springer, 2000, pp. 291–305.

[30] J. Larrosa, E. Morancho, D. Niso, On the practical use of variable elimination in constraint optimization problems: 'still-life' as a case study, Journal of Artificial Intelligence Research 23 (2005) 421–440.

[31] A. Middeldorp, S. Okui, T. Ida, Lazy narrowing: Strong completeness and eager variable elimination, Theoretical Computer Science 167 (1&2) (1996) 95–130.

[32] Y. Moinard, Forgetting literals with varying propositional symbols, Journal of Logic and Computation 17 (5) (2007) 955–982.

[33] A. Weber, Updating propositional formulas, in: Expert Database Systems, Proceedings From the First International Conference, Charleston, South Carolina, USA, April 1-4, 1986, 1986, pp. 487–500.

[34] D. M. Gabbay, R. A. Schmidt, A. Szalas, Second Order Quantifier Elimination: Foundations, Computational Aspects and Applications, College Publications, 2008.

[35] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, T. C. Przymusinski, Dynamic updates of non-monotonic knowledge bases, The Journal of Logic Programming 45 (1-3) (2000) 43–70.

[36] T. Eiter, M. Fink, G. Sabbatini, H. Tompits, On properties of update sequences based on causal rejection, Theory and Practice of Logic Programming 2 (6) (2002) 721–777.

[37] C. Sakama, K. Inoue, An abductive framework for computing knowledge base updates, Theory and Practice of Logic Programming 3 (6) (2003) 671–713.

[38] M. Slota, J. Leite, Robust equivalence models for semantic updates of answer-set programs, in: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012, AAAI Press, 2012, pp. 158–168.

[39] M. Slota, J. Leite, A unifying perspective on knowledge updates, in: L. F. del Cerro, A. Herzig, J. Mengin (Eds.), Logics in Artificial Intelligence - 13th European Conference, JELIA 2012, Toulouse, France, September 26-28, 2012. Proceedings, Vol. 7519 of LNAI, Springer, 2012, pp. 372–384.

[40] J. P. Delgrande, T. Schaub, H. Tompits, S. Woltran, A model-theoretic approach to belief change in answer set programming, ACM Transactions on Computational Logic 14 (2) (2013) 14.

[41] M. Slota, J. Leite, The rise and fall of semantic rule updates based on se-models, Theory and Practice of Logic Programming 14 (6) (2014) 869–907.

[42] K.-S. Wong, Forgetting in logic programs, Ph.D. thesis, The University of New South Wales (2009).

[43] Y. Wang, Y. Zhang, Y. Zhou, M. Zhang, Forgetting in logic programs under strong equivalence, in: G. Brewka, T. Eiter, S. A. McIlraith (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012, AAAI Press, 2012, pp. 643–647.

[44] Y. Wang, K. Wang, M. Zhang, Forgetting for answer set programs revisited, in: F. Rossi (Ed.), IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013, IJCAI/AAAI, 2013, pp. 1163–1168.

[45] M. Knorr, J. J. Alferes, Preserving strong equivalence while forgetting, in: E. Fermé, J. Leite (Eds.), Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings, Vol. 8761 of LNCS, Springer, 2014, pp. 412–425.

[46] Y. Wang, Y. Zhang, Y. Zhou, M. Zhang, Knowledge forgetting in answer set programming, Journal of Artificial Intelligence Research 50 (2014) 31–70.

[47] Y. Zhang, Y. Zhou, Knowledge forgetting: Properties and applications, Artificial Intelligence 173 (16-17) (2009) 1525–1537.

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

[49] R. Gonçalves, M. Knorr, J. Leite, The ultimate guide to forgetting in answer set programming, in: C. Baral, J. Delgrande, F. Wolter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth

International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016, AAAI Press, 2016, pp. 135–144.

[50] T. Eiter, M. Fink, S. Woltran, Semantical characterizations and complexity of equivalences in answer set programming, ACM Transactions on Computational Logic 8 (3) (2007) 1–53.

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

[52] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (3-4) (1991) 365–385.

[53] A. Heyting, Die Formalen Regeln der Intuitionistischen Logik, Sitzungsberichte der Preussischen Akademie der Wissenschaften. Physikalisch-mathematische Klasse, 1930.

[54] D. Pearce, Stable inference as intuitionistic validity, Journal of Logic Programming 38 (1) (1999) 79–91.

[55] V. Lifschitz, D. Pearce, A. Valverde, Strongly equivalent logic programs, ACM Transactions on Computational Logic 2 (4) (2001) 526–541.

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

[57] T. Eiter, H. Tompits, S. Woltran, On solution correspondences in answer-set programming, in: L. P. Kaelbling, A. Saffiotti (Eds.), IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005, Professional Book Center, 2005, pp. 97–102.

[58] T. Eiter, M. Fink, H. Tompits, S. Woltran, On eliminating disjunctions in stable logic programming, in: D. Dubois, C. A. Welty, M. Williams (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004, AAAI Press, 2004, pp. 447–458.

[59] T. Eiter, M. Fink, H. Tompits, S. Woltran, Simplifying logic programs under uniform and strong equivalence, in: V. Lifschitz, I. Niemelä (Eds.), Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings, Vol. 2923 of LNCS, Springer, 2004, pp. 87–99.

[60] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Annals of Mathematics and Artificial Intelligence 15 (3-4) (1995) 289–323.

[61] K.-S. Wong, Sound and complete inference rules for SE-consequence, Journal of Artificial Intelligence Research 31 (2008) 205–216.

[62] J. P. Delgrande, A knowledge level account of forgetting, Journal of Artificial Intelligence Research 60 (2017) 1165–1213.

[63] F. Aguado, P. Cabalar, J. Fandinno, D. Pearce, G. Pérez, C. Vidal, Forgetting auxiliary atoms in forks, in: B. Bogaerts, A. Harrison (Eds.), Proceedings of the 10th Workshop on Answer Set Programming and Other Computing Paradigms co-located with the 14th International Conference on Logic Programming and Nonmonotonic Reasoning, ASPOCP@LPNMR 2017, Espoo, Finland, July 3, 2017, Vol. 1868 of CEUR Workshop Proceedings, CEUR-WS.org, 2017.

[64] F. Aguado, P. Cabalar, J. Fandinno, D. Pearce, G. Pérez, C. Vidal, Forgetting auxiliary atoms in forks, Artif. Intell. 275 (2019) 575–601.

[65] V. Lifschitz, What is answer set programming?, in: D. Fox, C. P. Gomes (Eds.), Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008, AAAI Press, 2008, pp. 1594–1597.

[66] T. Janhunen, E. Oikarinen, H. Tompits, S. Woltran, Modularity aspects of disjunctive stable models, J. Artif. Intell. Res. 35 (2009) 813–857.

[67] R. Gonçalves, T. Janhunen, M. Knorr, J. Leite, S. Woltran, Forgetting in modular answer set programming, in: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, AAAI Press, 2019, pp. 2843–2850.

[68] M. Berthold, R. Gonçalves, M. Knorr, J. Leite, A syntactic operator for forgetting that satisfies strong persistence, Theory Pract. Log. Program. 19 (5-6) (2019) 1038–1055.

[69] M. Truszczynski, Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs, Artificial Intelligence 174 (16-17) (2010) 1285–1306.

[70] J. J. Alferes, M. Knorr, K. Wang, Forgetting under the well-founded semantics, in: P. Cabalar, T. C. Son (Eds.), Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings, Vol. 8148 of LNCS, Springer, 2013, pp. 36–41.

[71] A. V. Gelder, K. A. Ross, J. S. Schlipf, The well-founded semantics for general logic programs, Journal of the ACM 38 (3) (1991) 620–650.