On Some Properties of Forgetting in ASP

Ricardo Gonçalves and **Matthias Knorr** and **João Leite**[1]

Abstract. Many approaches for forgetting in Answer Set Programming (ASP) have been proposed in recent years, in the form of specific operators, or classes of operators, following different principles and obeying different properties. Whereas each approach was developed to somehow address some particular view on forgetting, thus aimed at obeying a specific set of properties deemed adequate for such view, only a recently published comprehensive overview of existing operators and properties provided a uniform and complete picture, including many novel (even surprising) results on relations between properties and operators. Yet, this overview ignored to a large extent a different set properties for forgetting in ASP, and in this paper we close this gap. It turns out that, while some of these properties are closely related to the properties previously studied, four of them are distinct providing novel results and insights further strengthening established relations between existing operators.

1 Introduction

Forgetting – or variable elimination – is an operation that allows the removal, from a knowledge base, of *middle* variables no longer deemed relevant. The importance of forgetting is witnessed by its application to cognitive robotics [23, 24, 27], resolving conflicts [16, 37, 7, 17], and ontology abstraction and comparison [33, 15, 13, 14]. With its early roots in Boolean Algebra [20], it has been extensively studied in the context of classical logic [1, 16, 18, 19, 25, 26, 34].

Only more recently, the operation of forgetting began to receive attention in the context of logic programming and non-monotonic reasoning, notably of Answer Set Programming (ASP). It turns out that the rule-based nature and non-monotonic semantics of ASP create very unique challenges to the development of forgetting operators – just as to the development of other belief change operators such as those for revision and update, c.f. [28] – making it a special endeavour with unique characteristics distinct from those for classical logic.

Over the years, many have proposed different approaches to forgetting in ASP, through the characterization of the result of forgetting a set of atoms from a given program up to some equivalence class, and/or through the definition of concrete operators that produce a specific program for each input program and atoms to be forgotten [37, 7, 36, 31, 30, 12, 32, 6].

All these approaches were typically proposed to obey some specific set of properties that their authors deemed adequate, some adapted from the literature on *classical* forgetting [38, 31, 32], others specifically introduced for the case of ASP [7, 36, 31, 30, 12, 6]. Examples of such properties include *strengthened consequence*, which requires that the answer sets of the result of forgetting be bound to the answer-sets of the original program modulo the forgotten atoms, or 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.

The result is a *complex* landscape filled with operators and properties, with very little effort put into drawing a map that could help to better understand the relationships between properties and operators. Recently, this problem has been tackled in [9] by presenting a systematic study of *forgetting* in ASP, thoroughly investigating the different approaches found in the literature, their properties and relationships – including many novel results – giving rise to a comprehensive guide aimed at helping users navigate this topic's complex landscape and ultimately assist them in choosing suitable operators for each application.

However, this study ignores to a large extent the postulates on forgetting in ASP introduced in [36], and it is our aim to close this gap here.[2] As a result of our study of these postulates, we are able to conclude that, while some of them are implied by one of the previously studied properties in [9], hence ultimately weaker than these and thus of less importance, others are distinct and provide additional novel results further strengthening the relations between properties and classes of operators as established previously.

To make the presentation self-contained, we first adapt part of the material presented in [9]. Namely, we present general notation on HT-models, logic programs, and answer sets, a section on forgetting in ASP, recall existing properties of forgetting, as discussed in [9], and the classes of operators existing in the literature. In the latter two cases, we also include results on relations of properties and classes of properties. Subsequently, we introduce the postulates from [36] and present our results on relations w.r.t. previously established properties and concerning which of the different classes of operators satisfies which postulate. We conclude with an outlook on future work.

2 Preliminaries

We assume a propositional language $\mathcal{L}_{\mathcal{A}}$ over a *signature* $\mathcal{A}$, a finite set of propositional atoms[3]. The *formulas* of $\mathcal{L}_{\mathcal{A}}$ are inductively defined using connectives $\bot$, $\wedge$, $\vee$, and $\supset$:

$$\varphi ::= \bot \mid p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \supset \varphi \qquad (1)$$

where $p \in \mathcal{A}$. In addition, $\neg\varphi$ and $\top$ are resp. shortcuts for $\varphi \supset \bot$ and $\bot \supset \bot$. Given a finite set S of formulas, $\bigvee S$ and $\bigwedge S$ denote resp. the disjunction and conjunction of all formulas in S. In particular, $\bigvee \emptyset$ and $\bigwedge \emptyset$ stand for resp. $\bot$ and $\top$, and $\neg S$ and $\neg\neg S$ represent resp. $\{\neg\varphi \mid \varphi \in S\}$ and $\{\neg\neg\varphi \mid \varphi \in S\}$. Unless otherwise stated, we assume that the underlying signature for a particular formula φ is $\mathcal{A}(\varphi)$, the set of atoms appearing in φ.

[1] NOVA LINCS, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, email: {rjrg|mkn|jleite}@fct.unl.pt.

[2] We use the term *postulate* to follow [36] and easily distinguish them from the *properties* discussed in [9]. However, their role is the same as the role of other properties.

[3] Often, the term propositional variable is used synonymously.

HT-models Regarding the semantics of propositional formulas, we consider the monotonic logic here-and-there (HT) and equilibrium models [21]. An *HT-interpretation* is a pair $\langle H, T\rangle$ s.t. $H \subseteq T \subseteq \mathcal{A}$. The satisfiability relation in HT, denoted $\models_{\mathsf{HT}}$, is recursively defined as follows for $p \in \mathcal{A}$ and formulas φ and ψ:

- $\langle H, T\rangle \models_{\mathsf{HT}} p$ if $p \in H$;
- $\langle H, T\rangle \not\models_{\mathsf{HT}} \bot$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \wedge \psi$ if $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$ and $\langle H, T\rangle \models_{\mathsf{HT}} \psi$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \vee \psi$ if $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$ or $\langle H, T\rangle \models_{\mathsf{HT}} \psi$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \supset \psi$ if both (i) $T \models \varphi \supset \psi$,[4] and (ii) $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$ implies $\langle H, T\rangle \models_{\mathsf{HT}} \psi$.

A given HT-interpretation is an *HT-model* of a formula φ if $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$. We denote by $\mathcal{HT}(\varphi)$ the set of *all HT-models of* φ. In particular, $\langle T, T\rangle \in \mathcal{HT}(\varphi)$ is an *equilibrium model* of φ if there is no $T' \subset T$ s.t. $\langle T', T\rangle \in \mathcal{HT}(\varphi)$.

Given two formulas φ and ψ, if $\mathcal{HT}(\varphi) \subseteq \mathcal{HT}(\psi)$, then φ *entails* ψ in HT, written $\varphi \models_{\mathsf{HT}} \psi$. Also, φ and ψ are *HT-equivalent*, written $\varphi \equiv_{\mathsf{HT}} \psi$, if $\mathcal{HT}(\varphi) = \mathcal{HT}(\psi)$.

For sets of atoms X, Y and $V \subseteq \mathcal{A}$, $Y \sim_V X$ denotes that $Y \setminus V = X \setminus V$. For HT-interpretations $\langle H, T\rangle$ and $\langle X, Y\rangle$, $\langle H, T\rangle \sim_V \langle X, Y\rangle$ denotes that $H \sim_V X$ and $T \sim_V Y$. Then, for a set $\mathcal{M}$ of HT-interpretations, $\mathcal{M}_{\dagger V}$ denotes the set $\{\langle X, Y\rangle \mid \langle H, T\rangle \in \mathcal{M}$ and $\langle X, Y\rangle \sim_V \langle H, T\rangle\}$.

Logic Programs An *(extended) logic program* P is a finite set of *(extended) rules*, i.e., formulas of the form

$$\bigwedge \neg\neg D \wedge \bigwedge \neg C \wedge \bigwedge B \supset \bigvee A\ , \tag{2}$$

where all elements in $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_l\}$, $C = \{c_1, \ldots, c_m\}$, $D = \{d_1, \ldots, d_n\}$ are atoms.[5] Such rules r are also commonly written as

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, ..., b_l, not\, c_1, ..., not\, c_m, \\ not\, not\, d_1, ..., not\, not\, d_n\ , \tag{3}$$

and we will use both forms interchangeably. Given r, we distinguish its *head*, $head(r) = A$, and its *body*, $body(r) = B \cup \neg C \cup \neg\neg D$, representing a disjunction and a conjunction.

As shown by Cabalar and Ferraris [4], any set of (propositional) formulas is HT-equivalent to an (extended) logic program which is why we can focus solely on these.

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*, and *fact* if also $l = 0$. The classes of *disjunctive*, *normal* and *Horn programs*, $\mathcal{C}_d$, $\mathcal{C}_n$, and $\mathcal{C}_H$, are defined resp. as a finite set of disjunctive, normal, and Horn rules. We also call extended rules with $k \leq 1$ *non-disjunctive*, thus admitting a non-standard class $\mathcal{C}_{nd}$, called *non-disjunctive programs*, different from normal programs. We have $\mathcal{C}_H \subset \mathcal{C}_n \subset \mathcal{C}_d \subset \mathcal{C}_e$ and also $\mathcal{C}_n \subset \mathcal{C}_{nd} \subset \mathcal{C}_e$.

We now recall the *answer set semantics* [8] for logic programs. Given a program P and a set I of atoms, the *reduct* P^I is defined as $P^I = \{A \leftarrow B : r \text{ of the form (3) in } P, C \cap I = \emptyset, D \subseteq I\}$. A set I' of atoms is a model of P^I if, for each $r \in P^I$, $I' \models B$ implies $I' \models A$. I is minimal in a set S, denoted by $I \in \mathcal{MIN}(S)$, if there is no $I' \in S$ s.t. $I' \subset I$. Then, I is an *answer set* of P iff I is a minimal model of P^I. Note that, for $\mathcal{C}_{nd}$ and its subclasses, this minimal model is in fact unique. The set of all answer sets of P is denoted by $\mathcal{AS}(P)$. 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*. Also, the *V-exclusion* of a set of answer sets $\mathcal{M}$, denoted $\mathcal{M}_{\|V}$, is $\{X \setminus V \mid X \in \mathcal{M}\}$. Two programs P_1, P_2 are *equivalent* if $\mathcal{AS}(P_1) = \mathcal{AS}(P_2)$ and *strongly equivalent* if $P_1 \equiv_{\mathsf{HT}} P_2$. It is well-known that answer sets and equilibrium models coincide [21], but since the former notion is frequently used in the literature and arguably easier to use, we will mainly rely on it. Finally, determining if program P has an answer set is Σ_2^p-complete, and NP-complete if P is non-disjunctive [5].

[4] $\models$ is the standard consequence relation from classical logic.

[5] Extended logic programs [22] are actually more expressive, but this form is sufficient here.

3 Forgetting

The principal idea of forgetting in logic programming is to remove or hide certain atoms from a given program, while preserving its semantics for the remaining atoms.

Example 1 *Consider the following program P:*

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

The result of forgetting about atom e from P should be a program over the remaining atoms of P, i.e., it should not contain e. Intuitively, in the result, the fact $b \leftarrow$ should persist since it is independent of e. In addition, the link between a and b should be preserved in some way, even if e is absent. Also, d should still follow from the result of forgetting as the original rule $d \leftarrow not\, c$ does not contain e.

As the example indicates, preserving the semantics for the remaining atoms is not necessarily tied to one unique program. Rather 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 set of atoms V to be forgotten, returns a unique program, the result of forgetting about V from P.

Definition 1 *Given a class of logic programs $\mathcal{C}$ over $\mathcal{A}$, a* forgetting operator *is a partial function $\mathsf{f} : \mathcal{C} \times 2^{\mathcal{A}} \rightarrow \mathcal{C}$ s.t. $\mathsf{f}(P, V)$ is a program over $\mathcal{A}(P) \setminus V$, for each $P \in \mathcal{C}$ and $V \in 2^{\mathcal{A}}$. We call $\mathsf{f}(P, V)$ the* result of forgetting about V from P. *Furthermore, f is called* closed *for $\mathcal{C}' \subseteq \mathcal{C}$ if, for every $P \in \mathcal{C}'$ and $V \in 2^{\mathcal{A}}$, we have $\mathsf{f}(P, V) \in \mathcal{C}'$. A* class F of forgetting operators *is a set of forgetting operators.*

Note that the requirement for being a partial function is a natural one given the existing notions in the literature, where some are not closed for certain classes of programs.

To remain as general and uniform as possible, we focus on classes of operators. Whenever a notion of forgetting in the literature is defined through a concrete forgetting operator only, we consider the class containing that single operator.

It is worth noting that some notions of forgetting do not explicitly require that atoms to be forgotten be absent from the result of forgetting, but instead that they be *irrelevant*:

(IR) $\mathsf{f}(P, V) \equiv_{\mathsf{HT}} P'$ for some P' not containing any $v \in V$.

Although **(IR)** allows (irrelevant) occurrences of atoms in a result of forgetting, in the literature they are subsequently assumed to be not occurring, which is sanctioned by **(IR)**. Hence, focusing on operators that yield programs without the atoms to be forgotten is not a restriction in these cases.

4 Properties of Forgetting

Previous work on forgetting in ASP has introduced a variety of desirable properties. In this section, we recall the relevant properties found in the literature and investigate existing relations between them.

Unless otherwise stated, F is a class of forgetting operators, and $\mathcal{C}$ the class of programs over $\mathcal{A}$ of a given $\mathsf{f} \in \mathsf{F}$.

The first three properties were proposed by Eiter and Wang [7], though not formally introduced as such. The first two were in fact guiding principles for defining their notion of forgetting, while the third was later formalized by Wang et al. [30].

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

Strengthened Consequence requires that the answer sets of the result of forgetting be answer sets of the original program, ignoring the atoms to be forgotten.

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

Weak Equivalence requires that forgetting preserves equivalence of programs (equality of answer sets).

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

Strong Equivalence requires that forgetting preserves strong equivalence of programs.

The next three properties, together with **(IR)**, were introduced by Zhang and Zhou [38] in the context of forgetting in modal logics, and later adopted by Wang et al. [31, 32] for forgetting in ASP.

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

Weakening requires that the HT-models of P also be HT-models of $\mathsf{f}(P,V)$, thus implying that $\mathsf{f}(P,V)$ has at most the same consequences as P.

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

Positive Persistence requires that the consequences of P not containing atoms to be forgotten be preserved in the result of forgetting.

(NP) F satisfies *Negative Persistence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$: if $P \not\models_{\mathsf{HT}} P'$, with $P' \in \mathcal{C}$ and $\mathcal{A}(P') \subseteq \mathcal{A} \setminus V$, then $\mathsf{f}(P,V) \not\models_{\mathsf{HT}} P'$.

Negative Persistence requires that a program not containing atoms to be forgotten not be a consequence of $\mathsf{f}(P,V)$, unless it was already a consequence of P.

The following property was introduced by Wong [36], but the more descriptive name is novel here.

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

Strong Invariance requires that it be (strongly) equivalent to add a program without the atoms to be forgotten before or after forgetting.

The property called *existence* was discussed by Wang et al. [31] and formalized by Wang et al. [30]. It requires that a result of forgetting for P in $\mathcal{C}$ exists in the class $\mathcal{C}$, important to iterate. We extend this property s.t. it be explicitly tied to a class $\mathcal{C}$, thus allowing to speak about F being closed/not closed for different classes $\mathcal{C}$.

($\mathbf{E}_{\mathcal{C}}$) 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 $\mathcal{C}$.

In the literature, classes of operators are often defined in ways such that only some of its members are closed for a certain class. Thus, class F being closed for some $\mathcal{C}$ only requires that there exists some "witness in favor of it", instead of having to restrict the class to the closed operators.

The next property was introduced by Wang et al. [30] building on the ideas behind **(sC)** by Eiter and Wang [7].

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

Consequence persistence requires that the answer sets of the result of forgetting correspond exactly to the answer sets of the original program, ignoring the atoms to be forgotten.

The following property was introduced by Knorr and Alferes [12] with the aim of imposing the preservation of all dependencies contained in the original program.

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

This strengthens **(CP)** by imposing that the correspondence between answer-sets of the result of forgetting and those of the original program be preserved in the presence of any additional set of rules not containing the atoms to be forgotten.

The final property here is due to Delgrande and Wang [6], although its name is novel as well.

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

Weakened Consequence requires that the answer sets of the original program be preserved while forgetting, ignoring the atoms to be forgotten.

These properties are not orthogonal to one another, and several relations between them exist. The following proposition (from [9]) establishes all known relevant relations between properties, some novel and some to be found in the literature.

Proposition 1 *The following relations hold for all* F*:*[6]

1. **(CP)** *is incompatible with* **(W)** *as well as with* **(NP)** *(for* F *closed for* $\mathcal{C}$*, where* $\mathcal{C}$ *contains normal logic programs);*
2. **(W)** *is equivalent to* **(NP)***;*
3. **(SP)** *implies* **(PP)***;*
4. **(SP)** *implies* **(SE)***;*
5. **(W)** *and* **(PP)** *together imply* **(SE)***;*
6. **(CP)** *and* **(SI)** *together are equivalent to* **(SP)***;*
7. **(sC)** *and* **(wC)** *together are equivalent to* **(CP)***;*

[6] To ease the reading, here "**(P)**" stands for "F satisfies **(P)**".

8. **(CP)** *implies* **(wE)***;*
9. **(SE)** *and* **(SI)** *together imply* **(PP)***.*

Items 1.-4. are known from the literature: 1. in [30], 2.-3. in [11], and 4. in [12]. The remainder are novel.

Note first, that 1. and 2. also rely on **(IR)** in their original formulation, in particular, **(W)** is equivalent to **(NP)** and **(IR)**. As **(IR)** is incorporated directly into our definition of forgetting operators, this reliance is ensured implicitly. This means that, by 2., the four properties proposed by Zhang and Zhou [38] actually reduce to two distinct ones. In addition, 5. ensures that these two imply **(SE)**, which is considered desirable by Wang et al. [31] in addition to the former. So, five desired properties can be represented by two.

As indicated by 3. and 4., **(SP)** seems to be an expressive property, further confirmed by the new result 6. that provides a non-trivial decomposition of **(SP)** into **(CP)** and **(SI)**. These two are themselves expressive, as witnessed by other new results. Namely, 7. shows that **(CP)** is the combination of **(sC)** and **(wC)**, and 8. that it implies preservation of equivalence, while 9. provides the non-trivial result that Strong Equivalence and Strong Invariance imply Positive Persistence. The latter means that 3. can actually be obtained without relying on **(CP)**. Altogether, **(SP)** implies all properties in this section, except for **(W)** and **(NP)**, to which it is incompatible under the condition given in 1., and **(E$_C$)**, where we need to consider concrete classes of programs.

5 Operators of Forgetting

We now turn our attention to operators of forgetting in ASP, reviewing the approaches found in the literature and establishing novel relations between them.

Strong and Weak Forgetting The first proposals are due to Zhang and Foo [37] introducing two syntactic operators for normal logic programs, termed Strong and Weak Forgetting. Both start with computing a reduction corresponding to the well-known weak partial evaluation (WGPPE) [2], defined as follows: for a normal logic program P and $a \in \mathcal{A}$, $R(P,a)$ is the set of all rules in P and all rules of the form $head(r_1) \leftarrow body(r_1) \setminus \{a\} \cup body(r_2)$ for each $r_1, r_2 \in P$ s.t. $a \in body(r_1)$ and $head(r_2) = a$. Then, the two operators differ on how they subsequently remove rules containing a, the atom to be forgotten. In Strong Forgetting, all rules containing a are simply removed:

$$\mathsf{f}_{strong}(P,a) = \{r \in R(P,a) \mid a \notin \mathcal{A}(r)\}$$

In Weak Forgetting, rules with occurrences of $not\, a$ in the body are kept, after $not\, a$ is removed.

$$\mathsf{f}_{weak}(P,a) = \{head(r) \leftarrow body(r) \setminus \{not\, a\} \mid r \in R(P,a), a \notin head(r) \cup body(r)\}$$

The motivation for this difference is whether such $not\, a$ is seen as support for the rule head (Strong) or not (Weak). In both cases, the actual operator for a set of atoms V is defined by the sequential application of the respective operator to each $a \in V$. Both operators are closed for $\mathcal{C}_n$. The corresponding singleton classes are defined as follows.

$$\mathsf{F}_{strong} = \{\mathsf{f}_{strong}\} \qquad \mathsf{F}_{weak} = \{\mathsf{f}_{weak}\}$$

Semantic Forgetting Eiter and Wang [7] proposed Semantic Forgetting to improve on some of the shortcomings of the two purely syntax-based operators f_{strong} and f_{weak}. Semantic Forgetting introduces a class of operators for consistent disjunctive programs[7] defined as follows:

$$\mathsf{F}_{sem} = \{\mathsf{f} \mid \mathcal{AS}(\mathsf{f}(P,V)) = \mathcal{MIN}(\mathcal{AS}(P)_{\|V})\}$$

The basic idea is to characterize a result of forgetting just by its answer sets, obtained by considering only the minimal sets among the answer sets of P ignoring V. Three concrete algorithms are presented, two based on semantic considerations and one syntactic. Unlike the former, the latter is not closed for classes[8] $\mathcal{C}_d^+$ and $\mathcal{C}_n^+$, since double negation is required in general.

Semantic Strong and Weak Forgetting Wong [36] argued that semantic forgetting should not be focused on answer sets only, as these do not contain all the information present in a program. He defined two classes of forgetting operators for disjunctive programs, building on HT-models.[9] First, given a program P and an atom a, the set of all consequences of P is defined as $Cn(P,a) = \{r \mid r \text{ disjunctive}, P \models_{\mathsf{HT}} r, \mathcal{A}(r) \subseteq \mathcal{A}(P)\}$. We obtain $P_S(P,a)$ and $P_W(P,a)$, the results of strongly and weakly forgetting a single atom a from P, as follows:

1. Consider $P_1 = Cn(P,a)$.
2. Obtain P_2 by removing from P_1: (i) r with $a \in body(r)$, (ii) a from the head of each r with $not\, a \in body(r)$.
3. Given P_2, obtain $P_S(P,a)$ and $P_W(P,a)$ by replacing/removing certain rules r in P_2 as follows:

	r with $not\, a$ in body	r with a in head
S	(remove)	(remove)
W	remove only $not\, a$	remove only a

The generalization to sets of atoms V, i.e., $P_S(P,V)$ and $P_W(P,V)$, can be obtained by simply sequentially forgetting each $a \in V$, yielding the following classes of operators.

$$\mathsf{F}_S = \{\mathsf{f} \mid \mathsf{f}(P,V) \equiv_{\mathsf{HT}} P_S(P,V)\}$$
$$\mathsf{F}_W = \{\mathsf{f} \mid \mathsf{f}(P,V) \equiv_{\mathsf{HT}} P_W(P,V)\}$$

While steps 2. and 3. are syntactic, different strongly equivalent representations of $Cn(P,a)$ exist, thus providing different instances. Wong [36] defined one construction based on inference rules for HT-consequence, closed for $\mathcal{C}_d$.

HT-Forgetting Wang et al. [31, 32] introduced HT-Forgetting, building on properties introduced by Zhang and Zhou [38] in the context of modal logics, with the aim of overcoming problems with Wongs notions, namely that each of them did not satisfy one of the properties **(PP)** and **(W)**. HT-Forgetting is defined for extended programs and uses representations of sets of HT-models directly.

$$\mathsf{F}_{\mathsf{HT}} = \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = \mathcal{HT}(P)_{\dagger V}\}$$

A concrete operator is presented [32] that is shown to be closed for $\mathcal{C}_e$ and $\mathcal{C}_H$, and it is also shown that no operator exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$.

[7] Actually, classical negation can occur in scope of not, but due to the restriction to consistent programs, this difference is of no effect [8], so we ignore it here.

[8] Here, $^+$ denotes the restriction to consistent programs.

[9] Wong [36] considers SE-models [29]. Without loss of generality, we consider the more general HT-models.

	sC	**wE**	**SE**	**W**	**PP**	**NP**	**SI**	**CP**	**SP**	**wC**	$\mathbf{E}_{\mathcal{C}_H}$	$\mathbf{E}_{\mathcal{C}_n}$	$\mathbf{E}_{\mathcal{C}_d}$	$\mathbf{E}_{\mathcal{C}_{nd}}$	$\mathbf{E}_{\mathcal{C}_e}$
F_{strong}	×	×	×	✓	×	✓	✓	×	×	×	✓	✓	-	-	-
F_{weak}	×	×	×	×	✓	×	✓	×	×	×	✓	✓	-	-	-
F_{sem}	✓	✓	×	×	×	×	×	×	×	×	✓	✓	✓	-	-
F_{S}	×	×	✓	✓	✓	✓	×	×	×	×	✓	×	✓	-	-
F_{W}	✓	✓	✓	×	✓	×	✓	×	×	×	✓	✓	✓	-	-
F_{HT}	×	×	✓	✓	✓	✓	✓	×	×	×	✓	×	×	×	✓
F_{SM}	✓	✓	✓	×	✓	×	×	✓	×	✓	✓	×	×	×	✓
F_{Sas}	✓	✓	✓	×	✓	×	✓	✓	✓	✓	✓	×	×	×	×
F_{SE}	×	×	✓	✓	✓	✓	×	×	×	×	✓	×	✓	-	-

Figure 1. Satisfaction of properties for known classes of forgetting operators. For class F and property **P**, '✓' represents that F satisfies **P**, '×' that F does not satisfy **P**, and '-' that F is not defined for the class $\mathcal{C}$ in consideration.

SM-Forgetting Wang et al. [30] defined a modification of HT-Forgetting, SM-Forgetting, for extended programs, with the objective of preserving the answer sets of the original program (modulo the forgotten atoms).

$$\mathsf{F}_{\mathsf{SM}} = \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) \text{ is a maximal subset of } \mathcal{HT}(P)_{\dagger V} \text{ s.t. } \mathcal{AS}(\mathsf{f}(P,V)) = \mathcal{AS}(P)_{\| V}\}$$

A concrete operator is provided that, like for F_{HT}, is shown to be closed for $\mathcal{C}_e$ and $\mathcal{C}_H$. It is also shown that no operator exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$.

Strong AS-Forgetting Knorr and Alferes [12] introduced Strong AS-Forgetting with the aim of preserving not only the answer sets of P itself but also those of $P \cup R$ for any R over the signature without the atoms to be forgotten. The notion is defined abstractly for classes of programs $\mathcal{C}$.

$$\mathsf{F}_{Sas} = \{\mathsf{f} \mid \mathcal{AS}(\mathsf{f}(P,V) \cup R) = \mathcal{AS}(P \cup R)_{\| V} \text{ for all programs } R \in \mathcal{C} \text{ with } \mathcal{A}(R) \subseteq \mathcal{A}(P) \setminus V\}$$

A concrete operator is defined for $\mathcal{C}_{nd}$, but not closed for $\mathcal{C}_n$ and only defined for certain programs with double negation.

SE-Forgetting Delgrande and Wang [6] recently introduced SE-Forgetting based on the idea that forgetting an atom from program P is characterized by the set of those SE-consequences, i.e., HT-consequences, of P that do not mention the atoms to be forgotten. The notion is defined for disjunctive programs building on an inference system by Wong [35] that preserves strong equivalence. Given that $\vdash_s$ is the consequence relation of this system, $Cn_{\mathcal{A}}(P)$ is $\{r \in \mathcal{L}_{\mathcal{A}} \mid r \text{ disjunctive}, P \vdash_s r\}$. The class is defined by:

$$\mathsf{F}_{SE} = \{\mathsf{f} \mid \mathsf{f}(P,V) \equiv_{\mathsf{HT}} Cn_{\mathcal{A}}(P) \cap \mathcal{L}_{\mathcal{A}(P)\setminus V}\}$$

An operator is provided, which is closed for $\mathcal{C}_d$.

While all these classes were introduced with differing motivations, they coincide under certain conditions, e.g., when restricted to specific classes of programs [9].

Proposition 2 *For all Horn programs P, every $V \subseteq \mathcal{A}(P)$, and all forgetting operators $\mathsf{f}_1, \mathsf{f}_2$ in the classes F_{strong}, F_{weak}, F_S, F_{HT}, F_{SM}, F_{Sas}, and F_{SE}, it holds that $\mathsf{f}_1(P,V) \equiv_{\mathsf{HT}} \mathsf{f}_2(P,V)$.*

Example 2 *Consider the subset of rules of P in Ex. 1 that are Horn, $P' = \{a \leftarrow e, e \leftarrow b, b \leftarrow\}$. Then, $\mathsf{f}(P',\{e\})$ for any f in any of these classes is strongly equivalent to $a \leftarrow b$ and $b \leftarrow$. All three known operators in F_{sem} actually also satisfy this condition, but the class is not sufficiently restricted to ensure this in general. F_W completely differs since any operator in F_W must include $\leftarrow b$ in its result.*

Wang et al. [31, 32] additionally show that, for $\mathcal{C}_H$, the result of F_{HT} is strongly equivalent to that of classical forgetting. We thus obtain as a corollary that this holds for all classes of forgetting operators mentioned in Prop. 2.

Perhaps surprisingly, two classes of operators coincide [9].

Theorem 1 *Consider the class of disjunctive programs. Then, F_S and F_{SE} coincide.*

This coincidence can be traced back to the fact that the inference system used for F_{SE} is the same as that used to define the example operator for F_S. This correspondence can be extended to F_{HT} in a particular case [9].

Proposition 3 *Let P be a disjunctive logic program, $V \subseteq \mathcal{A}(P)$, $\mathsf{f}_S \in \mathsf{F}_S$, $\mathsf{f}_{\mathsf{HT}} \in \mathsf{F}_{\mathsf{HT}}$, and $\mathsf{f}_{SE} \in \mathsf{F}_{SE}$. Then, $\mathsf{f}_S(P,V) \equiv_{\mathsf{HT}} \mathsf{f}_{\mathsf{HT}}(P,V) \equiv_{\mathsf{HT}} \mathsf{f}_{SE}(P,V)$ whenever $\mathsf{f}_{\mathsf{HT}}(P,V)$ is strongly equivalent to a disjunctive program.*

This does not hold in general, as the next example shows.

Example 3 *Given $P = \{a \leftarrow not\, b, b \leftarrow not\, a, \leftarrow a, b\}$, consider forgetting about b from P. For any f_{HT}, $\mathsf{f}_{\mathsf{HT}}(P,\{b\})$ must contain $a \leftarrow not\, not\, a$, which is not disjunctive.*

This also means that item 1. in Prop. 2 [6] actually does not hold.

For the sake of completeness and to ease later comparisons, we also include a table with the results on satisfaction of properties for known classes of forgetting operators obtained in [9] – these results are shown in Fig. 1.

Complexity All approaches show or mention that computing the result of forgetting with one particular operator is in EXP. The only exception is $\mathsf{f} \in \mathsf{F}_{SE}$, where forgetting one atom leads only to at most a quadratic increase in program size. Still, if a set of atoms is forgotten, then, e.g., Ex. 9 by Brass et al. [3] applies, hence, it is also in EXP. Sometimes the complexity of other problems is established, such as satisfiability of $\mathsf{f}(P,V)$ or whether some a holds in some or all $S \in \mathcal{AS}(\mathsf{f}(P,V))$. In most cases, these results match those considering P itself, with the exception of F_{sem} where slight modifications are due to the additional minimality test.

6 Wongs Properties of Forgetting

With all notions and notation in place regarding forgetting in ASP, the commonly considered properties and the existing classes of forgetting operators, we can now turn to the postulates introduced by Wong [36]. These postulates were in fact defined in a somewhat different way compared to the properties presented in Sec. 4. Namely, they are only defined for forgetting a single atom, for disjunctive programs (as this is the maximal class of programs considered in [36]), and in a generic way for different notions of equivalence, which is then instantiated with two different notions to account for different classes of operators. Here, we only consider HT-equivalence, i.e., strong equivalence, as, in the literature, this is clearly the more relevant of the two notions considered in [36] and in line with previously presented material here and in [9].

We start by recalling these postulates[10] with some minor adjustments to our notation including the extension to the most general class of extended logic programs considered here, but leaving the restriction to forgetting only single atoms as is.

(F0) F satisfies **(F0)** if, for each $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}$ and $a \in \mathcal{A}$: if $P \equiv_{\mathsf{HT}} P'$, then $\mathsf{f}(P, \{a\}) \equiv_{\mathsf{HT}} \mathsf{f}(P', \{a\})$.

(F1) F satisfies **(F1)** if, for each $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}$ and $a \in \mathcal{A}$: if $P \models_{\mathsf{HT}} P'$, then $\mathsf{f}(P, \{a\}) \models_{\mathsf{HT}} \mathsf{f}(P', \{a\})$.

(F2) F satisfies **(F2)** if, for each $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}$ and $a \in \mathcal{A}$: if a does not appear in R, then $\mathsf{f}(P \cup R, \{a\}) \equiv_{\mathsf{HT}} \mathsf{f}(P', \{a\}) \cup R$ for all $R \in \mathcal{C}$.

(F2-) F satisfies **(F2-)** if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$, and $a \in \mathcal{A}$: if $P \models_{\mathsf{HT}} r$ and a does not occur in r, then $\mathsf{f}(P, \{a\}) \models_{\mathsf{HT}} r$ for all rules r expressible in $\mathcal{C}$.

(F3) F satisfies **(F3)** if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $a \in \mathcal{A}$: $\mathsf{f}(P, \{a\})$ does not contain any atoms that are not in P.

(F4) F satisfies **(F4)** if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $a \in \mathcal{A}$: if $\mathsf{f}(P, \{a\}) \models_{\mathsf{HT}} r$, then $\mathsf{f}(\{s\}, \{a\}) \models_{\mathsf{HT}} r$ for some $s \in Cn_{\mathcal{A}}(P)$.

(F5) F satisfies **(F5)** if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $a \in \mathcal{A}$: if $\mathsf{f}(P, \{a\}) \models_{\mathsf{HT}} A \leftarrow B \cup \neg C \cup \neg\neg D$, then $P \models_{\mathsf{HT}} A \leftarrow B \cup \neg C \cup \{\neg a\} \cup \neg\neg D$.

(F6) F satisfies **(F6)** if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $a, b \in \mathcal{A}$: $\mathsf{f}(\mathsf{f}(P, \{b\}), \{a\}) \equiv_{\mathsf{HT}} \mathsf{f}(\mathsf{f}(P, \{a\}), \{b\})$.

These (formal) postulates represent the following: If two programs are HT-equivalent, then forgetting about atom a from both preserves HT-equivalence **(F0)**; if a program is a HT-consequence of another program, then forgetting about atom a from both preserves this HT-consequence **(F1)**; when forgetting about an atom a, it does not matter if we add a set of rules over the remaining language before or after forgetting **(F2)**; any consequence of the original program not mentioning atom a is also a consequence of the result of forgetting about a **(F2-)**; the result of forgetting about an atom from a program does only contain atoms occurring in the original program **(F3)**; any rule which is a consequence of the result of forgetting about an atom from program P, is actually a consequence of the result of forgetting about that atom from a single rule among the HT-consequences of P **(F4)**; for any rule which is a HT-consequence of the result of forgetting about an atom a from program P, this rule with its body extended by $not\, a$ is a HT-consequence of P itself **(F5)**; and the order is not relevant when sequentially forgetting two atoms **(F6)**.

Note that $Cn_{\mathcal{A}}(P)$ for **(F4)** is defined here necessarily over the maximal class of programs considered in the class of operators, and that the kind of rules considered in **(F5)** is restricted according to the maximal class of programs considered in a given class of operators.

First, we establish relations between these postulates and between them and the properties discussed in Sec. 4.

Proposition 4 *The following relations hold for all* F*:*

1. **(F1)** *implies* **(F0)***;*
2. **(F2)** *and* **(F1)** *imply* **(F2-)***;*
3. **(SE)** *implies* **(F0)***;*
4. **(W)** *and* **(PP)** *together imply* **(F1)***;*
5. **(SI)** *implies* **(F2)***;*
6. **(PP)** *implies* **(F2-)***;*
7. **(W)** *implies* **(F5)***.*

The first two results have been shown in [36], the remainder are novel. We can observe that postulates **(F0)**, **(F2)**, **(F2-)**, and **(F5)** are generalized by properties presented in [9], and **(F1)** by a pair of these. This latter observation is also related to the result 5. of Prop. 1 and the fact that both **(F1)** and **(SE)** imply **(F0)**.

We now proceed by presenting further intuitions for each of these postulates and in particular we show which operators presented in Sec. 5 satisfy which of the new postulates.

We start with **(F0)** which is a special case of **(SE)** because it only considers forgetting one atom instead of a set. It shares with **(SE)** the intuition that forgetting the same atom(s) should preserve strong equivalence of programs.

Proposition 5 *For postulate* **(F0)** *the following holds:*

- F_S*,* F_W*,* F_{HT}*,* F_{SM}*,* F_{Sas} *and* F_{SE} *satisfy* **(F0)***;*
- F_{strong}*,* F_{weak} *and* F_{sem} *do not satisfy* **(F0)***.*

The fact that classes F_S, F_W, F_{HT}, F_{SM}, F_{Sas} and F_{SE} satisfy **(F0)** follows from Prop. 4 since they all satisfy **(SE)**. In [36], F_{strong} and F_{weak} are shown to not satisfy **(F0)**. For F_{sem}, the argument given in [7] to show that F_{sem} does not satisfy **(SE)** also applies to **(F0)**. Hence, even though **(F0)** is weaker than **(SE)** the results for all classes of operators coincide with those for **(SE)** (see [9]), thus, in the bigger picture of existing properties of forgetting, **(F0)** seems negligible.

For **(F1)**, the idea is that forgetting the same atom(s) should preserve HT-consequence between two programs. As already argued in [36], this postulate can be considered a strengthening of **(F0)**.

Proposition 6 *For postulate* **(F1)** *the following holds:*

- F_S*,* F_W*,* F_{HT} *and* F_{SE} *satisfy* **(F1)***;*
- F_{strong}*,* F_{weak}*,* F_{sem}*,* F_{SM} *and* F_{Sas} *do not satisfy* **(F1)***.*

The fact that F_S and F_W satisfy **(F1)** was proved in [36]. For F_{HT} and F_{SE}, this result follows from Prop. 4 and the fact that F_{HT} and F_{SE} satisfy both **(W)** and **(PP)**.

For the negative results, F_{strong}, F_{weak} and F_{sem} cannot satisfy **(F1)** since they do not satisfy **(F0)**. For F_{SM} and F_{Sas}, consider the following programs $P = \{a \leftarrow not\, p, p \leftarrow not\, a\}$ and $P' = \{a \leftarrow not\, p\}$. Then clearly $P \models_{\mathsf{HT}} P'$, but since $\mathsf{f}(P, p) \equiv_{\mathsf{HT}} \{a \leftarrow not\, not\, a\}$ and $\mathsf{f}(P', p) \equiv_{\mathsf{HT}} \{a \leftarrow\}$ for any $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}} \cup \mathsf{F}_{Sas}$ we have that $\mathsf{f}(P, p) \not\models_{\mathsf{HT}} \mathsf{f}(P', p)$.

Thus, **(F1)** is not only distinct per se, but in fact, compared to the results on properties satisfied by known classes of operators, it provides a unique set of classes of operators of forgetting for which it is satisfied (cf. Fig. 1). This means that it would be worth including

[10] As mentioned before, we use the term *postulate* to follow [36] and ease readability. Technically, they are treated as every other *property*.

this postulate in the set of properties considered in the full study of properties of forgetting operators.

As argued in [36], it should not matter whether we add the new rules before or after forgetting, as long as these rules do not refer the the forgotten atom(s). Similar to **(F0)**, postulate **(F2)** is a special case of one of the properties considered in Sec. 4.

Proposition 7 *For postulate* **(F2)** *the following holds:*

- F_{strong}, F_{weak}, F_W, F_{HT} *and* F_{Sas} *satisfy* **(F2)***;*
- F_S, F_{sem}, F_{SM} *and* F_{SE} *do not satisfy* **(F2)***.*

It was proved in [36] that F_W satisfies **(F2)**. The classes F_{strong}, F_{weak}, F_{HT} and F_{Sas} do satisfy **(F2)**, since they satisfy **(SI)** and given Prop. 4. Regarding the negative results, it was proved in [36] that F_S and F_{sem} do not satisfy **(F2)**. For F_{SM} and F_{SE}, the counterexample given in [9] for **(SI)** also applies for **(F2)**. Thus, all results coincide with those of **(SI)**, hence, **(F2)** also seems negligible.

In [36], **(F2-)** was introduced as a weakening of **(F2)**. As our new result shows in Prop. 4, it rather is a special case of the already considered property **(PP)**.

Proposition 8 *For postulate* **(F2-)** *the following holds:*

- F_{weak}, F_S, F_W, F_{HT}, F_{SM}, F_{Sas} *and* F_{SE} *satisfy* **(F2-)***;*
- F_{strong} *and* F_{sem} *do not satisfy* **(F2-)***.*

The positive results follow from Prop. 4 and the fact that all such operators satisfy **(PP)**. Regarding the two negative results, the counterexamples given in [32] for **(PP)** also apply for **(F2-)**. Thus, all results coincide with those of **(PP)**, hence, similar to **(F2)**, **(F2-)** seems negligible as well in the broader picture of properties being satisfied by classes of forgetting operators.

In [36], two further variations of **(F2)** are considered. One, called **(F2')**, is discarded right away as being insufficient to solve the incompatibility between F_S and **(F2)**. The other, called **(F2*)** restricts the program R to a single rule, only for the sake of F_S satisfying this restricted version of **(F2)**. But in our view, permitting only the addition of single rules is of little value, which is why we have omitted this variant from our considerations.

The intuition of **(F3)** is that forgetting is intended to simplify the language of a program by removing unnecessary or unwanted atoms. This is reasonable since otherwise, if atoms not occurring in a program were allowed to appear in the result of forgetting, then a trivial solution for the result of forgetting would be to simply rename the atoms to be forgotten using such extra atoms. Given our definition of (classes of) forgetting operators, this postulate is trivially satisfied.

Proposition 9 *All classes of operators* F_{strong}, F_{weak}, F_{sem}, F_S, F_W, F_{HT}, F_{SM}, F_{Sas} *and* F_{SE} *satisfy* **(F3)***.*

Hence, **(F3)**, though a very reasonable postulate as such, is of little value for discriminating differences and commonalities between different classes of forgetting operators, so it can be neglected as well.

The postulate **(F4)** states that every rule which is a HT-consequence of the result of forgetting about atom a from P is a HT-consequence of the result of forgetting about a from a single rule which is itself a HT-consequence of P.

Proposition 10 *For postulate* **(F4)** *the following holds:*

- F_{strong}, F_{weak}, F_S, F_W, F_{HT}, *and* F_{SE} *satisfy* **(F4)***;*
- F_{sem}, F_{SM} *and* F_{Sas} *do not satisfy* **(F4)***.*

The positive result for F_S, F_W and F_{SE} was shown in [36]. For F_{HT}, this follows directly from the alternative definition of HT-forgetting in [32]. For F_{strong} and F_{weak}, the result follows from the fact that this postulate is already shown to hold for a stronger notion of equivalence in [36], and since the additional derivation rules distinguishing this notion of equivalence and HT-equivalence do not affect the result.

The negative result for F_{Sas} and F_{SM} can be shown with a counterexample based on program $P = \{a \leftarrow p, p \leftarrow \mathit{not}\,\mathit{not}\,p\}$. For any operator in either class of forgetting operators, the result of forgetting about p from P is strongly equivalent to $a \leftarrow \mathit{not}\,\mathit{not}\,a$. However neither this nor any other rule over $\{a\}$, which has this rule as a HT-consequence, appears in $Cn_{\mathcal{A}}(P)$. In the case of F_{sem}, the negative result follows from the rather relaxed definition of the class and the fact that for satisfying **(F4)** any operator in F_{sem} has to satisfy it: we can easily define an operator that is still in F_{sem}, but returns an arbitrary program, then **(F4)** clearly does not hold.

Thus, similar to **(F1)**, this postulate turns out to be of interest as no previously studied property exists which is satisfied for precisely the same set of classes of forgetting operators.

The intuition of **(F5)**, as presented in [36], is that any rule which is a HT-consequence of the result of forgetting must be a HT-consequence of the program itself in the situations where the atom to be forgotten is not known.

Proposition 11 *For postulate* **(F5)** *the following holds:*

- F_{strong}, F_{weak}, F_S, F_W, F_{HT} *and* F_{SE} *satisfy* **(F5)***;*
- F_{sem}, F_{SM} *and* F_{Sas} *do not satisfy* **(F5)***.*

The positive result for F_S, F_W and F_{SE} was shown in [36]. A similar argument can be used for F_{weak}. For F_{strong} and F_{HT}, the result follows from Prop. 4 and the fact that these classes satisfy **(W)**. The negative result for F_{sem} was shown in [36]. For F_{SM} and F_{Sas}, consider the program $P = \{a \leftarrow p, p \leftarrow \mathit{not}\,\mathit{not}\,p\}$. Then, for $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}}$ or $\mathsf{f} \in \mathsf{F}_{Sas}$, we have that $\mathsf{f}(P, \{p\}) \equiv_{\mathsf{HT}} \{a \leftarrow \mathit{not}\,\mathit{not}\,a\}$. Therefore, $\mathsf{f}(P, \{p\}) \models_{\mathsf{HT}} a \leftarrow \mathit{not}\,\mathit{not}\,a$, but it is not the case that $P \models_{\mathsf{HT}} a \leftarrow \mathit{not}\,\mathit{not}\,a, \mathit{not}\,p$.

Thus, surprisingly, even though the postulate is implied by an existing property, the set of classes of forgetting operators does not coincide with that of the stronger property, which makes **(F5)** also a property of interest in the context of distinguishing existing classes of forgetting operators.

Finally, **(F6)** allows that the order in which two atoms are forgotten does not matter.

Proposition 12 *For postulate* **(F6)** *the following holds:*

- F_{strong}, F_{weak}, F_{sem}, F_S, F_W, F_{HT}, F_{SM} *and* F_{SE} *satisfy* **(F6)***;*
- F_{Sas} *does not satisfy* **(F6)***.*

The positive result for each operator was proved in the same paper where such operator was defined (cf. Sec. 5). The negative result for F_{Sas} follows from the fact that F_{Sas} satisfies **(SP)**, an important property which is argued to capture the essence of forgetting in ASP, but which unfortunately in certain cases does not allow forgetting while satisfying this property [10]. Take the program $P = \{p \leftarrow \mathit{not}\,\mathit{not}\,p; a \leftarrow p; b \leftarrow \mathit{not}\,p\}$. Forgetting about b from P first is strongly equivalent to removing the third rule, and subsequently forgetting about p is strongly equivalent to $\{a \leftarrow \mathit{not}\,\mathit{not}\,a\}$. However, forgetting about p from P first while satisfying **(SP)** is simply not allowed. Hence, the order of forgetting matters for F_{Sas}. This postulate is succinct and there is no property considered in [9] which is satisfied by all classes but F_{Sas}. On the other hand, the problem of the failure of F_{Sas} seems tightly connected to the fact that this class is the only one which satisfies **(SP)**, i.e., **(SP)** seems to be incompatible with **(F6)** and therefore this postulate is most likely of less interest concerning the distinction of classes of forgetting operators.

7 Conclusions

We have studied eight postulates of forgetting in ASP, as introduced in [36], to fill a gap in a recent comprehensive guide on properties and classes of operators for forgetting in ASP and relations between these [9].

It turns out that four of them are actually directly implied by previously considered single properties and for three among these, the sets of classes of forgetting operators which satisfy the stronger and the weaker property do precisely coincide. This suggests that these three, **(F0)**, **(F2)**, and **(F2-)** can safely be ignored. Postulate **(F3)** is as such different, but always satisfied by definition of forgetting operators, so we can safely ignore this one as well. Postulate **(F6)** is also different, and not always satisfied, but it seems that this is solely tied to the incompatibility with an already existing important property, **(SP)**. So this postulate is of interest, but it probably will not contribute much to further distinguishing classes of operators.

The remaining three properties, **(F1)**, **(F4)**, and **(F5)**, are in fact distinct (even though **(F5)** is implied by an existing property), and no other already existing property is satisfied by precisely the same set of classes of forgetting operators in each of these cases. This means that these three are worth being considered for inclusion in the set of relevant properties as they would provide further distinguishing criteria for existing classes of operators. This would most certainly be of help to further clarify the relation between properties **(SE)**, **(W)**, and **(PP)** considered before and provide further means to actually create precise characterizations of many classes of forgetting operators, an open issue in [9].

REFERENCES

[1] W. W. Bledsoe and Larry M. Hines, 'Variable elimination and chaining in a resolution-based prover for inequalities', in *Procs. of CADE*, eds., Wolfgang Bibel and Robert A. Kowalski, volume 87 of *LNCS*, pp. 70–87. Springer, (1980).

[2] Stefan Brass and Jürgen Dix, 'Semantics of (disjunctive) logic programs based on partial evaluation', *J. Log. Program.*, **40**(1), 1–46, (1999).

[3] Stefan Brass, Jürgen Dix, Burkhard Freitag, and Ulrich Zukowski, 'Transformation-based bottom-up computation of the well-founded model', *TPLP*, **1**(5), 497–538, (2001).

[4] Pedro Cabalar and Paolo Ferraris, 'Propositional theories are strongly equivalent to logic programs', *TPLP*, **7**(6), 745–759, (2007).

[5] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov, 'Complexity and expressive power of logic programming', *ACM Comput. Surv.*, **33**(3), 374–425, (2001).

[6] James P. Delgrande and Kewen Wang, 'A syntax-independent approach to forgetting in disjunctive logic programs', in *Procs. of AAAI*, eds., Blai Bonet and Sven Koenig, pp. 1482–1488. AAAI Press, (2015).

[7] Thomas Eiter and Kewen Wang, 'Semantic forgetting in answer set programming', *Artif. Intell.*, **172**(14), 1644–1672, (2008).

[8] Michael Gelfond and Vladimir Lifschitz, 'Classical negation in logic programs and disjunctive databases', *New Generation Comput.*, **9**(3-4), 365–385, (1991).

[9] Ricardo Gonçalves, Matthias Knorr, and João Leite, 'The ultimate guide to forgetting in ASP', in *Procs. of KR*, eds., Chitta Baral, James P. Delgrande, and Frank Wolter, pp. 135–144. AAAI Press, (2016).

[10] Ricardo Gonçalves, Matthias Knorr, and João Leite, 'You can't always forget what you want', in *Procs. of ECAI*. IOS Press, (2016). accepted for publication.

[11] Jianmin Ji, Jia-Huai You, and Yisong Wang, 'On forgetting postulates in answer set programming', in *Procs. of IJCAI*, eds., Qiang Yang and Michael Wooldridge, pp. 3076–3083. AAAI Press, (2015).

[12] Matthias Knorr and José Júlio Alferes, 'Preserving strong equivalence while forgetting', in *Procs. of JELIA*, eds., Eduardo Fermé and João Leite, volume 8761 of *LNCS*, pp. 412–425. Springer, (2014).

[13] Boris Konev, Michel Ludwig, Dirk Walther, and Frank Wolter, 'The logical difference for the lightweight description logic EL', *J. Artif. Intell. Res. (JAIR)*, **44**, 633–708, (2012).

[14] Boris Konev, Carsten Lutz, Dirk Walther, and Frank Wolter, 'Model-theoretic inseparability and modularity of description logic ontologies', *Artif. Intell.*, **203**, 66–103, (2013).

[15] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev, 'Logic-based ontology comparison and module extraction, with an application to dl-lite', *Artif. Intell.*, **174**(15), 1093–1141, (2010).

[16] Jérôme Lang, Paolo Liberatore, and Pierre Marquis, 'Propositional independence: Formula-variable independence and forgetting', *J. Artif. Intell. Res. (JAIR)*, **18**, 391–443, (2003).

[17] Jérôme Lang and Pierre Marquis, 'Reasoning under inconsistency: A forgetting-based approach', *Artif. Intell.*, **174**(12-13), 799–823, (2010).

[18] Javier Larrosa, 'Boosting search with variable elimination', in *Procs. of CP*, ed., Rina Dechter, volume 1894 of *LNCS*, pp. 291–305. Springer, (2000).

[19] Javier Larrosa, Enric Morancho, and David Niso, 'On the practical use of variable elimination in constraint optimization problems: 'still-life' as a case study', *J. Artif. Intell. Res. (JAIR)*, **23**, 421–440, (2005).

[20] C. I. Lewis, *A survey of symbolic logic*, University of California Press, 1918. Republished by Dover, 1960.

[21] Vladimir Lifschitz, David Pearce, and Agustín Valverde, 'Strongly equivalent logic programs', *ACM Trans. Comput. Log.*, **2**(4), 526–541, (2001).

[22] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner, 'Nested expressions in logic programs', *Ann. Math. Artif. Intell.*, **25**(3-4), 369–389, (1999).

[23] Fangzhen Lin and Raymond Reiter, 'How to progress a database', *Artif. Intell.*, **92**(1-2), 131–167, (1997).

[24] Yongmei Liu and Ximing Wen, 'On the progression of knowledge in the situation calculus', in *Procs. of IJCAI*, ed., Toby Walsh, pp. 976–982. IJCAI/AAAI, (2011).

[25] Aart Middeldorp, Satoshi Okui, and Tetsuo Ida, 'Lazy narrowing: Strong completeness and eager variable elimination', *Theor. Comput. Sci.*, **167**(1&2), 95–130, (1996).

[26] Yves Moinard, 'Forgetting literals with varying propositional symbols', *J. Log. Comput.*, **17**(5), 955–982, (2007).

[27] David Rajaratnam, Hector J. Levesque, Maurice Pagnucco, and Michael Thielscher, 'Forgetting in action', in *Procs. of KR*, eds., Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter. AAAI Press, (2014).

[28] Martin Slota and João Leite, 'The rise and fall of semantic rule updates based on se-models', *TPLP*, **14**(6), 869–907, (2014).

[29] Hudson Turner, 'Strong equivalence made easy: nested expressions and weight constraints', *TPLP*, **3**(4-5), 609–622, (2003).

[30] Yisong Wang, Kewen Wang, and Mingyi Zhang, 'Forgetting for answer set programs revisited', in *Procs. of IJCAI*, ed., Francesca Rossi. IJCAI/AAAI, (2013).

[31] Yisong Wang, Yan Zhang, Yi Zhou, and Mingyi Zhang, 'Forgetting in logic programs under strong equivalence', in *Procs. of KR*, eds., Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, pp. 643–647. AAAI Press, (2012).

[32] Yisong Wang, Yan Zhang, Yi Zhou, and Mingyi Zhang, 'Knowledge forgetting in answer set programming', *J. Artif. Intell. Res. (JAIR)*, **50**, 31–70, (2014).

[33] Zhe Wang, Kewen Wang, Rodney W. Topor, and Jeff Z. Pan, 'Forgetting for knowledge bases in DL-Lite', *Ann. Math. Artif. Intell.*, **58**(1-2), 117–151, (2010).

[34] Andreas Weber, 'Updating propositional formulas', in *Expert Database Conf.*, pp. 487–500, (1986).

[35] Ka-Shu Wong, 'Sound and complete inference rules for SE-consequence', *J. Artif. Intell. Res. (JAIR)*, **31**, 205–216, (2008).

[36] Ka-Shu Wong, *Forgetting in Logic Programs*, Ph.D. dissertation, The University of New South Wales, 2009.

[37] Yan Zhang and Norman Y. Foo, 'Solving logic program conflict through strong and weak forgettings', *Artif. Intell.*, **170**(8-9), 739–778, (2006).

[38] Yan Zhang and Yi Zhou, 'Knowledge forgetting: Properties and applications', *Artif. Intell.*, **173**(16-17), 1525–1537, (2009).