The Ultimate Guide to Forgetting in Answer Set Programming

Ricardo Gonçalves and **Matthias Knorr** and **João Leite**
NOVA LINCS, Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
2829-516 Caparica, Portugal

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, we are lacking a comprehensive and uniform overview of existing operators and properties. We aim at overcoming this by thoroughly examining existing properties and (classes of) operators for forgetting in ASP, drawing a complete picture, which includes many novel (even surprising) results on relations between properties and operators. Our goal is to provide a guide to help users in choosing the most adequate operator for their application requirements.

Introduction

In this paper, we present a systematic study of *forgetting* in Answer Set Programming (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.

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 (Lin and Reiter 1997; Liu and Wen 2011; Rajaratnam et al. 2014), resolving conflicts (Lang, Liberatore, and Marquis 2003; Zhang and Foo 2006; Eiter and Wang 2008; Lang and Marquis 2010), and ontology abstraction and comparison (Wang et al. 2010; Kontchakov, Wolter, and Zakharyaschev 2010; Konev et al. 2012; 2013). With its early roots in Boolean Algebra (Lewis 1918), it has been extensively studied in the context of classical logic (Bledsoe and Hines 1980; Lang, Liberatore, and Marquis 2003; Larrosa 2000; Larrosa, Morancho, and Niso 2005; Middeldorp, Okui, and Ida 1996; Moinard 2007; Weber 1986).

Only more recently, the operation of forgetting began to receive attention in the context of logic programming and non-monotonic reasoning, notably of 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. (Slota and Leite 2014) – 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 (Zhang and Foo 2006; Eiter and Wang 2008; Wong 2009; Wang et al. 2012; Wang, Wang, and Zhang 2013; Knorr and Alferes 2014; Wang et al. 2014; Delgrande and Wang 2015).

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 (Zhang and Zhou 2009; Wang et al. 2012; 2014), others specifically introduced for the case of ASP (Eiter and Wang 2008; Wong 2009; Wang et al. 2012; Wang, Wang, and Zhang 2013; Knorr and Alferes 2014; Delgrande and Wang 2015). 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, Ji, Wang and You (2015) sought to characterize an operator that would obey *all* of the properties in the literature. Whereas, in principle, having such an operator would be a good outcome, especially because each property proposed in the literature is *prima facie* desirable, it turns out that any such operator can only be defined for an extremely limited class of programs, which does not include the standard class of normal logic programs nor just about all the programs used in examples found in the literature on forgetting in ASP.

This strengthens the idea 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. The choice of operator will then depend on which properties are deemed more important for the specific application in hand, for which it is important to understand: a) the relationships between different properties, b) which properties are obeyed by which operators, and even c) whether some sets of properties make more sense than others.

To this end, in this paper, we present a systematic, comprehensive and thorough study of *forgetting in ASP*, going well beyond a *simple* survey since many novel results and insights are presented. After a brief section with preliminaries and a section where the concept of forgetting is defined, the paper is divided into four main sections. The first one contains a comprehensive account of the properties found in the literature, together with an investigation into the relationships between them, including several novel results. The subsequent section is devoted to describing the operators defined in the literature, and establishing some results on their relationships, including a surprising equivalence between two of these operators. Then, we devote one section to present a comprehensive account of which properties are satisfied by which operators, some of the results to be found scattered in the literature, but more than half being novel. This section also contains a map of the landscape of existing operators, organized according to their properties, and a thorough discussion to better explain the main results and the (sometimes subtle) details that help understand them. The final main section, before the conclusions, contains a discussion of the landscape beyond the existing literature, including some additional results and thoughts on specific sets of properties for which operators have never been defined, and even show that some of them make little sense since they can be represented by *absurd* operators.

Preliminaries

We assume a propositional language $\mathcal{L}_\mathcal{A}$ over a *signature* $\mathcal{A}$, a finite set of propositional atoms[1]. The *formulas* of $\mathcal{L}_\mathcal{A}$ are inductively defined using connectives $\bot$, $\wedge$, $\vee$, and $\supset$:

$$\varphi ::= \bot \mid p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \supset \varphi \tag{1}$$

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.

HT-models Regarding the semantics of propositional formulas, we consider the monotonic logic here-and-there (HT) and equilibrium models (Lifschitz, Pearce, and Valverde 2001). 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) $T \models \varphi \supset \psi$,[2] and (ii) $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$ implies $\langle H, T\rangle \models_{\mathsf{HT}} \psi$.

An 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 $A = \{a_1, \dots, a_k\}$, $B = \{b_1, \dots, b_l\}$, $C = \{c_1, \dots, c_m\}$, $D = \{d_1, \dots, d_n\}$ are atoms.[3] Such rules r are also commonly written as

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

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 (2007), 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* (Gelfond and Lifschitz 1991) for logic programs. Given a program P and a set I of atoms, the *reduct* P^I is defined as $P^I = \{A \leftarrow B : r$ 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

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

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

[3] Extended logic programs (Lifschitz, Tang, and Turner 1999) are actually more expressive, but this form is sufficient here.

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 (Lifschitz, Pearce, and Valverde 2001), 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 (Dantsin et al. 2001).

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 $\mathsf{f} : \mathcal{C} \times 2^{\mathcal{A}} \to \mathcal{C}$ 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.

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 (2008), 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. (2013).

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

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 (2009) in the context of forgetting in modal logics, and later adopted by Wang et al. (2012; 2014) 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 (2009), 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. (2012) and formalized by Wang et al. (2013). 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 $\mathcal{C}$.

(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. (2013) building on the ideas behind **(sC)** by Eiter and Wang (2008).

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

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 (2014) 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 $\mathcal{AS}(\mathsf{f}(P,V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, 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 (2015), 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 $\mathcal{AS}(P)_{\|V} \subseteq \mathcal{AS}(\mathsf{f}(P,V))$.

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 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:[4]

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

Items 1.-4. are known from the literature: 1. in (Wang, Wang, and Zhang 2013), 2.-3. in (Ji, You, and Wang 2015), and 4. in (Knorr and Alferes 2014). 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 (2009) actually reduce to two distinct ones. In addition, 5. ensures that these two imply **(SE)**, which is considered desirable by Wang et al. (2012) 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$_\mathcal{C}$)**, where we need to consider concrete classes of programs.

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 (2006) 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) (Brass and Dix 1999), 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 $\mathit{head}(r_1) \leftarrow \mathit{body}(r_1) \setminus \{a\} \cup \mathit{body}(r_2)$ for each $r_1, r_2 \in P$ s.t. $a \in \mathit{body}(r_1)$ and $\mathit{head}(r_2) = a$. 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 $\mathit{not}\, a$ in the body are kept, after $\mathit{not}\, a$ is removed.

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

The motivation for this difference is whether such $\mathit{not}\, a$ 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 (2008) proposed Semantic Forgetting to improve on some of the shortcomings of the two purely syntax-based operators f_{strong} and f_{weak}.

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

Semantic Forgetting introduces a class of operators for consistent disjunctive programs[5] 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[6] $\mathcal{C}_d^+$ and $\mathcal{C}_n^+$, since double negation is required in general.

Semantic Strong and Weak Forgetting Wong (2009) 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.[7] 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 (2009) defined one construction based on inference rules for HT-consequence, closed for $\mathcal{C}_d$.

HT-Forgetting Wang et al. (2012; 2014) introduced HT-Forgetting, building on properties introduced by Zhang and Zhou (2009) 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 (Wang et al. 2014) 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 $\mathcal{C}_n$.

SM-Forgetting Wang et al. (2013) 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 (2014) 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 (2015) 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 (2008) 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.

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. (2012; 2014) 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.

[5]Actually, classical negation can occur in scope of not, but due to the restriction to consistent programs, this difference is of no effect (Gelfond and Lifschitz 1991), so we ignore it here.

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

[7]Wong (2009) considers SE-models (Turner 2003). 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. White background means a novel result, and gray a previously known one.[6]

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.

Proposition 3. *Let* P *be a disjunctive 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 (Delgrande and Wang 2015) actually does not hold.

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. (2001) 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.

On the Properties of Existing Operators

The *desirability* of the properties presented before is, to some extent, in the eye of the beholder. Often, a particular novel approach to forgetting is justified by the fact that previous approaches did not obey some new property deemed crucial, neglecting however that this novel approach actually ended up failing to satisfy other properties, themselves deemed crucial by those who introduced them. Whereas the introduction of most known approaches to forgetting was accompanied by a study of some properties they each enjoyed, there are many missing gaps, some because some properties were only introduced later, others because they were simply neglected. Despite the discussion and potential controversy around the adequacy of the properties, which may ultimately depend on the application at hand, the first and perhaps most important step is to draw an exhaustive picture regarding which properties are obeyed by which classes of operators. This takes us to the central theorem of our paper, illustrated in one easy-to-read table.

Theorem 2. *All results in Fig. 1 hold.*

One first observation is that every class of operators obeys a different set of properties (apart from F_S and F_{SE}, which coincide, c.f. Thm. 1). This is a strong indication that these properties play a role in characterizing the classes of operators. In fact, a precise characterization of some classes of operators in terms of the properties they satisfy sometimes exists (Wang et al. 2012; 2014; Delgrande and Wang 2015), although this not the case in general.

Specific Properties

We now focus on analyzing specific properties and how they relate to the known classes of operators.

Starting with **(sC)** and **(wE)**, we know, by Prop. 1, that any F that is known to satisfy **(CP)** also satisfies these two. For the remaining classes, it is worth illustrating why F_S, F_{HT}, and F_{SE} do not satisfy **(sC)** by looking at the example where we forget about a from $P = \{a \leftarrow not\, a\}$: all three classes require the result to be strongly equivalent to $\emptyset$, i.e., the forgetting operation introduces a new answer-set. Turning to **(wE)**, it requires that the results of forgetting about p from $P = \{q \leftarrow not\, p, q \leftarrow not\, q\}$ and from $Q = \{q \leftarrow\}$ have the same answer-sets, while the three classes F_S, F_{HT}, and F_{SE} require that the results be strongly equivalent to $\mathsf{f}(P,p) = \{q \leftarrow not\, q\}$ and $\mathsf{f}(Q,p) = \{q \leftarrow\}$, which are obviously not equivalent. F_W satisfies both properties: in the previous two examples, $\perp$ must be returned in the former, while $\mathsf{f}(P,p)$ includes $q \leftarrow$ in the latter.

[6] Previous results are proved in (Zhang and Foo 2006; Eiter and Wang 2008; Wong 2009; Wang et al. 2012; Wang, Wang, and Zhang 2013; Knorr and Alferes 2014; Wang et al. 2014; Delgrande and Wang 2015; Ji, You, and Wang 2015).

The properties **(SE)**, **(W)**, **(PP)**, and **(NP)** have received more attention in the literature, although focusing more on the properties not satisfied by previous approaches to motivate the introduction of a new one. As a result, several novel positive results were yet to be proved, and are included in the table. It is perhaps worth pointing out that despite Wang et al. (2014) having discussed that F_S and F_W do not satisfy **(PP)**, they did so using a counterexample – Ex. 3 in this paper – that is not really part of the language for which F_S and F_W are defined, since it relies on rules with double negation as missing consequences, hence an unfair argument. For the language for which they are defined, they satisfy **(PP)**. Worth illustrating is why F_{Sas} does not satisfy **(W)**/**(NP)**:

Example 4. *With* $P = \{a \leftarrow \mathit{not}\, b,\ b \leftarrow \mathit{not}\, c\}$, $\mathsf{f}(P, b)$ *for any* $\mathsf{f} \in \mathsf{F}_{Sas}$ *must contain* $a \leftarrow \mathit{not}\, \mathit{not}\, c$ *which is not a consequence of* P*, hence* **(W)** *(and* **(NP)***) is not satisfied.*

(SI) has received less attention, yet often this non-trivial property is satisfied. Wong (2009) showed **(SI)** for strong and weak forgetting, but using t-equivalence instead of HT-equivalence, whose semantics differs. The negative results for F_{SE} follow by correspondence to F_S and, for F_{SM}, from forgetting about b from P as in Ex. 4: $\mathsf{f}(P, b) \equiv_{\mathsf{HT}} \emptyset$ for $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}}$, so adding $c \leftarrow$ results precisely in a program containing this fact. If we add $c \leftarrow$ before forgetting, then the HT-models of the result of forgetting, ignoring all occurrences of b, correspond precisely to $\langle\{c\}, \{c\}\rangle$, $\langle\{c\}, \{a, c\}\rangle$, and $\langle\{a, c\}, \{a, c\}\rangle$. To preserve the answer sets, only the last of these three can be considered. Hence, $a \leftarrow$ and $c \leftarrow$ (or strongly equivalent rules) occur in the result of forgetting for any $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}}$, and **(SI)** does not hold.

The new negative results for **(CP)** and **(SP)** can be illustrated with forgetting about b from $P = \{a \leftarrow \mathit{not}\, b,\ b \leftarrow \mathit{not}\, a\}$, i.e., the first two rules of Ex. 3. Since $\mathcal{AS}(P) = \{\{a\}, \{b\}\}$, the result must have two answer sets $\{a\}$ and $\emptyset$, which is not possible for disjunctive programs obtained from operators in F_S, F_W, and F_{SE}. The same example serves as counterexample for all negative results of **(wC)**, while positive results follow for classes satisfying **(CP)** from Prop. 1. Notably, the counterexample also applies to F_{SE}, thus invalidating Thm. 2 in (Delgrande and Wang 2015), and explaining why all results for **(wC)** are novel.

The results on **(E$_\mathcal{C}$)** for different classes of programs $\mathcal{C}$ reveal that all classes of operators are closed for $\mathcal{C}_H$. Yet, when admitting negation, each F apart from F_{Sas} is closed for the maximal class of programs considered, but often not for intermediate ones, with the exception of F_{sem} and F_W. Interestingly, the two known semantic operators in F_{sem} are not closed for $\mathcal{C}_H$, while the known syntactic one is, despite not being closed in general. Note that '-' was used w.r.t. to the definitions of each F: the singleton classes F_{strong} and F_{weak} are precisely defined for normal programs; the intuition behind minimization embedded in F_{sem}'s definition does not combine well with double negation; and, for F_S, F_W, and F_{SE}, the consequence relation that is applied is defined for disjunctive rules.

We omit any complexity results from the table since they do not contribute to distinguishing classes of operators, as discussed before.

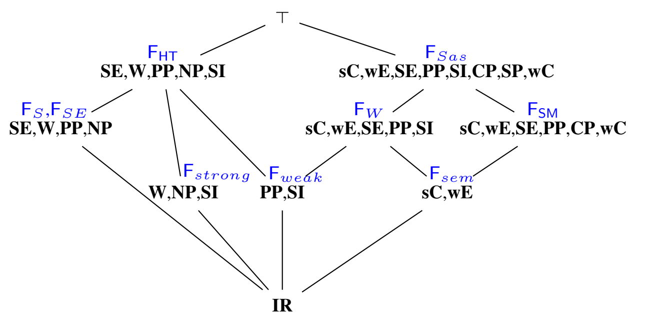

Figure 2: Sets of properties satisfied by known classes of forgetting operators

Classes of Operators

The results in Fig. 1 provide us valuable information to compare classes of operators and provide some guidelines regarding the choice of a forgetting operator.

The first concern is perhaps the required class of programs $\mathcal{C}$. If some class of operators is not closed or even not defined for $\mathcal{C}$, then it is certainly not a good choice. An exception may be the only known operator for F_{Sas}, whose applicability condition can be checked in linear time (on the number of rules), if one accepts that forgetting is not always possible.

With these considerations on **(E$_\mathcal{C}$)** in mind, we can analyze the remaining properties. Fig. 2 shows a lattice of inclusions between the sets of properties satisfied by each known class of forgetting operators, adding $\top$ to represent all the properties (except **(E$_\mathcal{C}$)**) and **(IR)** as the bottom element.

Fig. 2 makes it apparent that there is one kind of property that divides the classes into two groups, namely whether some kind of preservation of answer sets from the original program to its result of forgetting is supported or not. The classes for which this is the case are F_{sem}, F_W, F_{SM}, and F_{Sas} by either satisfying **(sC)** or **(CP)**. Among them, the two classes that satisfy **(CP)** turn out to be separated by only one property as follows (using semi-formal notation):[9]

$$\mathsf{F}_{\mathsf{SM}} + (\mathbf{SI}) \rightsquigarrow \mathsf{F}_{Sas}$$

Hence, the previously observed difference between the two on **(SP)** is only a consequence of the difference on **(SI)**.

Unlike the former, the two that satisfy **(sC)**, F_{sem} and F_W, are closed for all the program classes for which they are considered. For F_W, adding **(wC)** seemingly yields F_{Sas}:

$$\mathsf{F}_W + (\mathbf{wC}) \rightsquigarrow \mathsf{F}_{Sas}$$

However, note that picking an operator of F_W and somehow enforcing **(wC)** will not provide an operator of F_{Sas}. It can be shown that the effect of forgetting V from P for $\mathsf{f} \in \mathsf{F}_W$ yields a result that replaces all $v \in V$ as if they were false, independently of the actual rules in P. E.g., forgetting about p from $p \leftarrow$ would yield $\bot$, which is not aligned with the original idea of forgetting, i.e., removing all $v \in V$ without affecting other derivations, and adding **(wC)** would disallow such operation. In general, an operator of a superclass

[9]Such equations are in fact not to be read as precise characterizations of classes, but rather a form of visualization of differences.

does not necessarily belong to a subclass in the hierarchy of Fig. 2. This also means that at least one of F_W and F_{Sas} is not precisely characterized by their sets of satisfied properties, and given the intrinsic connection of F_{Sas} to **(SP)**, we conjecture that F_W is the culprit.

Among the four classes that do not support preservation of answer sets, F_{strong} and F_{weak} are closely related due to their similar definition. Both coincide on satisfying **(SI)**, a consequence of their syntactic definition that only manipulates rules containing the atoms to be forgotten, and differ on **(W)** and **(PP)**, a consequence of the different treatment of negated occurrences of the atoms to be forgotten.

For the third node without this preservation support, $\mathsf{F}_S/\mathsf{F}_{SE}$, there is also a close proximity to F_W based on their definition, yet, their characterizations differ substantially. Both satisfy **(SE)** and **(PP)**, but differ on five other properties, which means that in this case the variation in the definition has a much more profound effect on the set of satisfied properties. At the same time, we already know that there is a close relation to the remaining class F_{HT} as witnessed in Prop. 3. This is matched by a close correspondence in terms of satisfied properties.

$$\mathsf{F}_S/\mathsf{F}_{SE} + (\mathbf{SI}) \rightsquigarrow \mathsf{F}_{\mathsf{HT}}$$

Again, **(SI)** plays a distinguishing role. Notably this clarifies the apparent mismatch of the characterizations for F_{HT} and F_{SE}, both claiming that it is precisely given by **(IR)**, **(W)**, **(PP)**, and **(NP)**. This is indeed true for each of them for the maximal class of programs considered, but, intuitively, restricting F_{HT} to $\mathcal{C}_d$ cancels **(SI)**.

F_{HT} is also closely connected to F_{SM}, as the latter restricts the HT-models of the result s.t. **(CP)** holds. It turns out that this not only cancels **(W)** (see 1. of Prop. 1), but also **(SI)**.

Beyond the Existing Literature

Whereas the results and discussion so far provide insights into the properties and relations between existing classes of operators, which may help when picking one for a given application, the landscape of forgetting operators may admit additional classes, which will be the focus of this section.

Subclasses of Programs

While the results in Thm. 2 for $(\mathbf{E}_\mathcal{C})$ naturally differentiate between the classes of programs $\mathcal{C}$ considered, the remaining properties are stated for the most general class of programs for which the class of operators is defined. This begs the question whether restricting $\mathcal{C}$ would affect the results shown in Fig. 1, which we will now address.

We first consider Horn programs, for which we already know that all classes of operators are closed. From Prop. 2, we know that several classes that satisfy different sets of properties in the general case actually coincide. As expected, these classes all satisfy the same set of properties when restricted to Horn programs. The reason can be traced back to the incompatibility result 1. in Prop. 1, only stated for program classes above $\mathcal{C}_H$, i.e., it does not apply here.

Theorem 3. *For Horn programs, the following holds:*

- F_{strong}*,* F_{weak}*,* F_S*,* F_{HT}*,* F_{SM}*,* F_{Sas}*, and* F_{SE} *satisfy* **(W)** *and* **(SP)***;*
- F_{sem} *satisfies* **(CP)***;*
- F_W *satisfies* **(sC)***,* **(wE)***,* **(SE)** *and* **(SI)**.

Actually, only the minimally necessary properties are mentioned, the remaining can be obtained from Prop. 1. This means, in particular, that the first group of classes satisfies all the presented properties (with $(\mathbf{E}_\mathcal{C})$ limited to $(\mathbf{E}_H)$, of course), and, surprisingly, that reducing to Horn programs does not change the set of properties satisfied by F_W.

For normal programs, it turns out that the introduction of negation in the body immediately makes the result coincide with the general one. This is witnessed by the fact that all counterexamples are normal programs, in particular those mentioned in the previous section.

Additional Classes of Operators

We now take a step back and look at the bigger picture of the sets of properties beyond those that have been shown to hold for existing classes of operators. Fig. 3 provides a lattice with all possible combinations of properties, taking into consideration the following set of simplifications, that actually also can be understood as indicators as to which properties are important for the comparison.

- Since **(SP)** is only satisfied by F_{Sas}, hence not well-suited for a comparison, we use **(SI)** and **(CP)** instead. Also, all the implications of **(SI)** and **(CP)** are left implicit.
- Since **(W)** and **(NP)** (together with **(IR)**) are equivalent, we only use **(W)**.
- Whenever **(W)** and **(PP)** hold, we omit **(SE)**.
- Whenever **(SE)** and **(SI)** hold, we omit **(PP)**.
- Since **(sC)** is implied by **(CP)**, we only show the former whenever the latter does not hold.
- **(wC)** requires that the set of answer sets can only ever increase through forgetting, which is at odds with the non-monotonic nature of ASP. **(wC)** only seems meaningful if combined with **(sC)**, as **(CP)**, to preserve answer sets, which is seconded by the existing classes of forgetting operators. We therefore omit **(wC)**.
- **(sC)** requires that the answer sets of the result of forgetting from P be a subset of those of P. As the choice of the subset is deterministic for a concrete operator, **(wE)** holds as well. Thus, even though **(sC)** and **(wE)** are not formally related, any concrete operator satisfies either both or none, hence we omit **(wE)**.

Accordingly, the graph only shows **(sC)**, **(SE)**, **(W)**, **(PP)**, **(SI)**, and **(CP)** and each node automatically "inherits" all properties from the nodes below to which it is connected, even if some are not explicitly shown because they are implied according to Prop. 1. The lattice is arranged in levels based on the number of properties that are satisfied, where **(CP)** includes **(sC)**, and implicit properties are taken into account. Thus, e.g., **(SI)**,**(CP)** appears one level below $\top$ because of all the properties implied by the combination of the two. Links are made between neighboring levels apart from one case where a dashed line ensures that the link between **(W)**,**(SI)** and **(W)**,**(PP)**,**(SI)** is not lost.

Though quite a few nodes in the graph correspond to known classes or special cases for Horn, further classes can be explicitly determined. As an example, consider the concrete forgetting operators for Semantic Forgetting that also

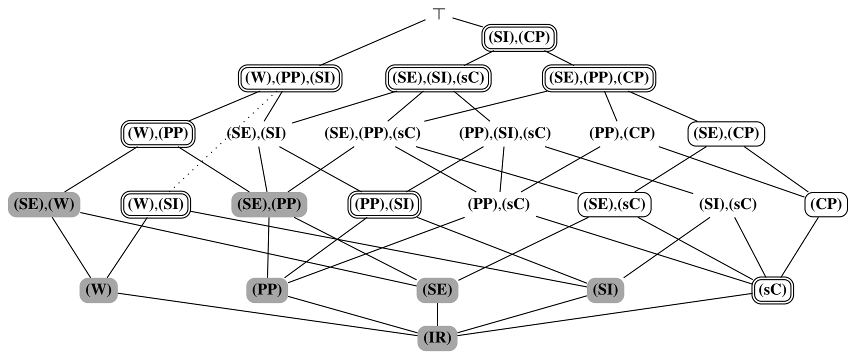

Figure 3: Combinations of properties. Sets of properties with doubled borders are satisfied by known classes, those with single borders match newly considered classes, and for those with gray background absurd operators have been found.

satisfy **(SE)**. Based on these, a new class of forgetting operators can be defined, essentially adding a condition to the definition of Semantic Forgetting that requires that for any two strongly equivalent programs, their results be also strongly equivalent. This provides a distinct class of operators which satisfies both **(sC)** and **(SE)**, and which, if restricted to Horn programs, satisfies **(SE)** and **(CP)**. The two cases for Horn programs, **(CP)** and **(SE)**,**(CP)** give rise to two new classes of operators by simply omitting the minimality condition on answer sets for the result, which also makes it reasonable to consider such classes for programs with double negation. An example operator is sketched for each one:

- **(CP)** : compute all answer sets, remove all atoms to be forgotten, create a program that represents this set of reduced answer sets;
- **(SE)**,**(CP)** : compute all answer sets, remove all atoms to be forgotten, create a canonical program representing the resulting set of answer sets.

Whether these new classes would be preferable to the already existing ones in certain aspects/applications remains an open question that requires further study.

Providing operators that precisely satisfy only a certain set of properties can also be used to show that some set of properties does not suffice to characterize the class:

- **(W)**,**(SI)** : delete all rules with atoms to be forgotten.

This operator matches the properties satisfied by Strong Forgetting, but it clearly does not fit into the class (even if we ignored that we defined F_{strong} as a singleton), since by the way deletion is applied, the idea of (WGPPE) is lost.

Further operators can be defined in a similar style, providing evidence that certain sets of properties are probably of little interest as they are satisfied by absurd operators. We mention those pointed out in Fig. 3:

- **(IR)** : delete all rules; then add some arbitrary rules over the remaining alphabet (after forgetting);[10]
- **(W)** : delete all rules with atoms to be forgotten and an arbitrary 50% of the remaining rules;
- **(PP)** : perform weak forgetting; in the resulting program, pick an arbitrary set of rules and turn them into facts by removing their body;
- **(SE)** : compute all answer sets; remove all atoms to be forgotten from them; create a set of facts that represents the intersection of all such reduced answer sets;
- **(SI)** : perform the WGPPE replacement step of strong and weak forgetting; in the resulting program, arbitrarily delete rules with negative occurrences (in the body) of atoms to be forgotten, or just remove their bodies, and delete all rules with positive occurrences (in the body or head) of atoms to be forgotten;
- **(SE)**,**(W)** : delete the entire program;
- **(SE)**,**(PP)** : add, as facts, to the result of performing SE-forgetting, the atoms that belong to some answer-set of the original program and for which there is some rule of the original program that contains some negative and no positive occurrence of forgotten atoms.

Other sets of properties, unannotated in Fig. 3, do not seem to permit the definition of such simplistic operators, so their interest as a class requires further study.

Conclusions

The landscape of forgetting in ASP comprises many operators and classes of operators defined to obey some subset of a large number of *desirable* properties proposed in the literature, while lacking a systematic account of all these results and their relations, making it all too difficult to get a clear understanding of the state-of-the-art, and even to choose the most adequate operator for some specific application.

This paper aimed at addressing this problem, presenting a systematic study of forgetting in ASP, including a thorough investigation of both properties and existing classes of forgetting operators, going well beyond a survey of the state-of-the-art as many novel results were included, thus achieving a truly comprehensive picture of the landscape. The wider picture of yet unexplored terrain was also approached, revealing new candidates for classes of operators as well as sets of properties whose apparent unreasonableness advises against further consideration.

One reasonable way to move forward is to investigate forgetting with semantics other than ASP, such as (Wang et al. 2014) based on the FLP-semantics (Truszczynski 2010), or (Alferes, Knorr, and Wang 2013; Knorr and Alferes 2014)

[10]The term "arbitrary" is used freely to represent some deterministic set, e.g., the first elements of some specific ordering.

based on the well-founded semantics (Gelder, Ross, and Schlipf 1991).

Acknowledgments

We would like to thank the reviewers for their comments, which helped improve this paper. R. Gonçalves, M. Knorr and J. Leite were partially supported by FCT under strategic project NOVA LINCS (PEst/UID/CEC/04516/2013). R. Gonçalves was partially supported by FCT grant SFRH/BPD/100906/2014 and M. Knorr was partially supported by FCT grant SFRH/BPD/86970/2012.

References

Alferes, J. J.; Knorr, M.; and Wang, K. 2013. Forgetting under the well-founded semantics. In Cabalar, P., and Son, T. C., eds., *Procs. of LPNMR*, volume 8148 of *LNCS*, 36–41. Springer.

Bledsoe, W. W., and Hines, L. M. 1980. Variable elimination and chaining in a resolution-based prover for inequalities. In Bibel, W., and Kowalski, R. A., eds., *Procs. of CADE*, volume 87 of *LNCS*, 70–87. Springer.

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

Brass, S.; Dix, J.; Freitag, B.; and Zukowski, U. 2001. Transformation-based bottom-up computation of the well-founded model. *TPLP* 1(5):497–538.

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

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33(3):374–425.

Delgrande, J. P., and Wang, K. 2015. A syntax-independent approach to forgetting in disjunctive logic programs. In Bonet, B., and Koenig, S., eds., *Procs. of AAAI*, 1482–1488. AAAI Press.

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

Gelder, A. V.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Comput.* 9(3-4):365–385.

Ji, J.; You, J.; and Wang, Y. 2015. On forgetting postulates in answer set programming. In Yang, Q., and Wooldridge, M., eds., *Procs. of IJCAI*, 3076–3083. AAAI Press.

Knorr, M., and Alferes, J. J. 2014. Preserving strong equivalence while forgetting. In Fermé, E., and Leite, J., eds., *Procs. of JELIA*, volume 8761 of *LNCS*, 412–425. Springer.

Konev, B.; Ludwig, M.; Walther, D.; and Wolter, F. 2012. The logical difference for the lightweight description logic EL. *J. Artif. Intell. Res. (JAIR)* 44:633–708.

Konev, B.; Lutz, C.; Walther, D.; and Wolter, F. 2013. Model-theoretic inseparability and modularity of description logic ontologies. *Artif. Intell.* 203:66–103.

Kontchakov, R.; Wolter, F.; and Zakharyaschev, M. 2010. Logic-based ontology comparison and module extraction, with an application to dl-lite. *Artif. Intell.* 174(15):1093–1141.

Lang, J., and Marquis, P. 2010. Reasoning under inconsistency: A forgetting-based approach. *Artif. Intell.* 174(12-13):799–823.

Lang, J.; Liberatore, P.; and Marquis, P. 2003. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res. (JAIR)* 18:391–443.

Larrosa, J.; Morancho, E.; and Niso, D. 2005. 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.

Larrosa, J. 2000. Boosting search with variable elimination. In Dechter, R., ed., *Procs. of CP*, volume 1894 of *LNCS*, 291–305. Springer.

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

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Trans. Comput. Log.* 2(4):526–541.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Ann. Math. Artif. Intell.* 25(3-4):369–389.

Lin, F., and Reiter, R. 1997. How to progress a database. *Artif. Intell.* 92(1-2):131–167.

Liu, Y., and Wen, X. 2011. On the progression of knowledge in the situation calculus. In Walsh, T., ed., *Procs. of IJCAI*, 976–982. IJCAI/AAAI.

Middeldorp, A.; Okui, S.; and Ida, T. 1996. Lazy narrowing: Strong completeness and eager variable elimination. *Theor. Comput. Sci.* 167(1&2):95–130.

Moinard, Y. 2007. Forgetting literals with varying propositional symbols. *J. Log. Comput.* 17(5):955–982.

Rajaratnam, D.; Levesque, H. J.; Pagnucco, M.; and Thielscher, M. 2014. Forgetting in action. In Baral, C.; Giacomo, G. D.; and Eiter, T., eds., *Procs. of KR*. AAAI Press.

Slota, M., and Leite, J. 2014. The rise and fall of semantic rule updates based on se-models. *TPLP* 14(6):869–907.

Truszczynski, M. 2010. Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artif. Intell.* 174(16-17):1285–1306.

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

Wang, Z.; Wang, K.; Topor, R. W.; and Pan, J. Z. 2010. Forgetting for knowledge bases in DL-Lite. *Ann. Math. Artif. Intell.* 58(1-2):117–151.

Wang, Y.; Zhang, Y.; Zhou, Y.; and Zhang, M. 2012. Forgetting in logic programs under strong equivalence. In Brewka, G.; Eiter, T.; and McIlraith, S. A., eds., *Procs. of KR*, 643–647. AAAI Press.

Wang, Y.; Zhang, Y.; Zhou, Y.; and Zhang, M. 2014. Knowledge forgetting in answer set programming. *J. Artif. Intell. Res. (JAIR)* 50:31–70.

Wang, Y.; Wang, K.; and Zhang, M. 2013. Forgetting for answer set programs revisited. In Rossi, F., ed., *Procs. of IJCAI*. IJCAI/AAAI.

Weber, A. 1986. Updating propositional formulas. In *Expert Database Conf.*, 487–500.

Wong, K.-S. 2008. Sound and complete inference rules for SE-consequence. *J. Artif. Intell. Res. (JAIR)* 31:205–216.

Wong, K.-S. 2009. *Forgetting in Logic Programs*. Ph.D. Dissertation, The University of New South Wales.

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

Zhang, Y., and Zhou, Y. 2009. Knowledge forgetting: Properties and applications. *Artif. Intell.* 173(16-17):1525–1537.