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, whose importance is witnessed by its application to cognitive robotics [35, 36, 39], resolving conflicts [26, 54, 11, 27], and ontology abstraction and comparison [50, 25, 23, 24]. With its early roots in Boolean Algebra [32], it has been extensively studied within classical logic [3, 26, 28, 29, 37, 38, 51].

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 it happened with the development of other belief change operators such as those for revision and update, cf. [31, 2, 10, 30, 40, 41, 42, 8, 43, 44] – 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 program given an input program and atoms to be forgotten [54, 11, 53, 48, 47, 21, 49, 9].

All these approaches were typically proposed to obey some specific set of properties deemed adequate by their authors, some adapted from the literature on *classical* forgetting [55, 48, 49], others specifically introduced for the case of ASP [11, 53, 48, 47, 21, 9]. Examples of 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, of difficult navigation. This problem was tackled in [17] by presenting a systematic study of *forgetting* in ASP, thoroughly investigating the different approaches found in the literature, their properties and relationships, 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, [17] ignores to a large extent the postulates on forgetting in ASP introduced by Wong in [53].[2] In this paper, we close this gap by thoroughly investigating them, their relationships with other properties and existing operators, concluding that, while some of them are straightforwardly implied by one of the previously studied properties, hence ultimately weaker than these and thus of less importance, others turn out to be distinct and provide additional novel results further strengthening the relations between properties and classes of operators as established previously.

Besides space considerations, the main reason why these postulates were left out of [17] was the fact that, thus far, they had not played a significant role in the literature on forgetting. Whereas completing the picture presented in [17] would be sufficient reason to thoroughly investigate these postulates, recent findings in [18] made it even more relevant. It was shown in [18] that it is not always possible to forget while preserving so-called *strong persistence* – an essential property for forgetting in ASP that encodes the required preservation, under forgetting, of all relations between non-forgotten atoms – shifting the attention to the question of when (and how) it is possible to forget, which is to some extent related to some of Wong's postulates. In particular, investigating Wong's postulates led us to prove that it may be impossible to step-wise iteratively forget a set of atoms that can be forgotten as a whole, while preserving *strong persistence*.

To make the presentation self-contained, we first adapt part of the material presented in [17]. Namely, we present general notation on HT-models, logic programs, answer sets, and on forgetting in ASP, recall existing properties of forgetting, as discussed in [17], the classes of operators existing in the literature, and results on relations of properties and classes of operators. Subsequently, we introduce the postulates from [53] and present our results on relations w.r.t. previously established properties and on which classes of operators satisfy which postulates. We then investigate possible generalisations of Wong's postulates, and the novel impossibility result concerning step-wise iterative forgetting, before concluding.

[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 [53] and easily distinguish them from the *properties* discussed in [17]. However, their role is the same as the role of other properties.

2 Preliminaries

We assume a propositional language $\mathcal{L}_\mathcal{A}$ over a *signature* $\mathcal{A}$, a finite set of propositional atoms. 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 $\{\neg\neg\varphi \mid \varphi \in S\}$. 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 [33]. 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) $T \models \varphi \supset \psi$,[3] 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 $T \sim_V Y$. 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 *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.[4] 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 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 [6], any set of (propositional) formulas is HT-equivalent to an (extended) logic program which is why we can focus solely on these.

[3] $\models$ is the standard consequence relation from classical logic.
[4] Extended logic programs [34] are actually more expressive, but this form is sufficient here.

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* [14] for logic programs. Given a program P and a set I of atoms, the *reduct* P^I is $P^I = \{A \leftarrow B : r$ 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. $I' \subset I$. 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*. 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 [33].

We also recall notions on forgetting from [17]. 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.

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. [17].

Example 1 *Consider the following program $P = \{d \leftarrow not\, c; a \leftarrow e; e \leftarrow b; b \leftarrow\}$. 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.

4 Properties of Forgetting

Previous work on forgetting in ASP has introduced a variety of desirable properties which we recall next. Unless stated otherwise, F is a class of forgetting operators, and $\mathcal{C}$ the class of programs over $\mathcal{A}$ of a given $\mathsf{f} \in \mathsf{F}$.

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

(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 $\mathcal{AS}(P) = \mathcal{AS}(P')$.

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

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

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

(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 $\mathsf{f}(P,V) \not\models_{\mathsf{HT}} P'$.

(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 $\mathcal{A}(R) \subseteq \mathcal{A} \setminus V$.

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

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

(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 $\mathcal{A}(R) \subseteq \mathcal{A} \setminus V$.

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

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

Some notions of forgetting do only require that atoms to be forgotten be *irrelevant*:

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

However, this is not a restriction, as argued in [17], and, implicitly, any F satisfies **(IR)**.

The following proposition establishes all known relevant relations between them.

Proposition 1 *The following relations hold for all* F*:*[5]

1. **(CP)** *is incompatible with* **(W)** *as well as with* **(NP)** *(for* F *closed for* $\mathcal{C}$*, where* $\mathcal{C}$ *contains normal logic programs); [47]*
2. **(W)** *is equivalent to* **(NP)***; [20]*
3. **(SP)** *implies* **(PP)***; [20]*
4. **(SP)** *implies* **(SE)***; [21]*
5. **(W)** *and* **(PP)** *together imply* **(SE)***; [17]*
6. **(CP)** *and* **(SI)** *together are equivalent to* **(SP)***; [17]*
7. **(sC)** *and* **(wC)** *together are equivalent to* **(CP)***; [17]*
8. **(CP)** *implies* **(wE)***; [17]*
9. **(SE)** *and* **(SI)** *together imply* **(PP)***. [17]*

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

5 Operators of Forgetting

We now review existing approaches to operators of forgetting in ASP following [17].

Strong and Weak Forgetting The first proposals are due to Zhang and Foo [54] introducing two syntactic operators for normal logic programs, termed Strong and Weak Forgetting. Both start by computing a reduction corresponding to the well-known weak partial evaluation (WGPPE) [4], 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 containing $not\, a$ in their bodies are kept, without the $not\, a$.

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

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 [11] proposed Semantic Forgetting to address some shortcomings of the two purely syntax-based operators f_{strong} and f_{weak}. Semantic Forgetting introduces the following class of operators for consistent disjunctive programs:[6]

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

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[7] $\mathcal{C}_d^+$ and $\mathcal{C}_n^+$, since double negation is required in general.

Semantic Strong and Weak Forgetting Wong [53] argued that semantic forgetting should not focus on answer sets only, as they do not contain all the information present in a program, and defined two classes of forgetting operators for disjunctive programs, building on

[6] Actually, classical negation can occur in scope of not, but due to the restriction to consistent programs, this difference is of no effect [14], so we ignore it here.

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

HT-models.[8] For program P and atom a, the set of consequences of P is $Cn(P,a) = \{r \mid r$ 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 atom a from P, as follows:

1. Obtain P_1 by removing from $Cn(P,a)$: (i) r with $a \in \mathit{body}(r)$, (ii) a from the head of each r with $\mathit{not}\, a \in \mathit{body}(r)$.
2. Obtain $P_S(P,a)$ and $P_W(P,a)$ from P_1 by replacing/removing rules r 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 both steps are syntactic, different strongly equivalent representations of $Cn(P,a)$ exist, thus providing different instances. Wong [53] defined one construction based on inference rules for HT-consequence, closed for $\mathcal{C}_d$.

HT-Forgetting Wang et al. [48, 49] introduced HT-Forgetting, building on properties introduced by Zhang and Zhou [55] 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 [49] 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. [47] introduced SM-Forgetting for extended programs, aiming at preserving the answer sets of the original program (modulo 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 [21] 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 [9] 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 atoms to be forgotten. The notion is defined for disjunctive programs building on an inference system by Wong [52] that preserves strong equivalence. Given that $\vdash_s$ is the consequence relation of this system, $Cn_{\mathcal{A}}(P)$ is $\{r \in \mathcal{L}_{\mathcal{A}} \mid r$ 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 $\mathcal{C}_d$.

To ease later comparisons, we also include in Fig. 1 the results on satisfaction of properties for known classes of forgetting operators obtained in [17].

6 Wongs Properties of Forgetting

With all concepts and notation in place regarding forgetting in ASP, the properties commonly considered, and the existing classes of forgetting operators, we can now turn our attention to the postulates introduced by Wong [53]. These postulates were defined in a somewhat different way when compared to the properties presented in Sec. 4. Namely, they only considered forgetting a single atom, were defined for disjunctive programs (the maximal class of programs considered in [53]), and used a generic formulation which allowed different notions of equivalence. 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 [53] (the other one being the non-standard T-equivalence) and in line with previously presented material here and in [17].

We start by recalling these postulates[9] adjusting them to our notation and extending them to the most general class of extended logic programs considered here, but maintaining, for now, the restriction to forgetting only single atoms.

(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 $\mathsf{f}(\{r'\},\{a\}) \models_{\mathsf{HT}} r$ for some $r' \in Cn_{\mathcal{A}}(P)$.

(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 postulates represent the following: Forgetting about atom a from HT-equivalent programs preserves HT-equivalence **(F0)**; if a program is an HT-consequence of another program, then forgetting about atom a from both programs preserves this HT-consequence **(F1)**; when forgetting about an atom a, it does not matter whether 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-)**;

[8] Without loss of generality, we consider HT-models instead of SE-models [46] as in [53].

[9] As mentioned before, we use the term *postulate* to follow [53] and ease readability. Technically, they are treated as every other *property*.

	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.

the result of forgetting about an atom from a program only contains 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 a consequence of the result of forgetting about that atom from a single rule among the HT-consequences of P **(F4)**; a rule obtained by extending with $not\, a$ the body of a rule which is an HT-consequence of the result of forgetting about an atom a from program P is an HT-consequence of P **(F5)**; and the order is not relevant when sequentially forgetting two atoms **(F6)**.

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

The following proposition relates these postulates and the properties in Sec. 4.

Proposition 2 *The following relations hold for all* F*:*

1. **(F1)** *implies* **(F0)***; [53]*
2. **(F2)** *and* **(F1)** *imply* **(F2-)***; [53]*
3. **(SE)** *implies* **(F0)***;*
4. **(W)** *and* **(PP)** *together imply* **(F1)***;*
5. **(SI)** *implies* **(F2)***;*
6. **(PP)** *implies* **(F2-)***;*
7. **(W)** *implies* **(F5)***.*

Postulates **(F0)**, **(F2)**, **(F2-)**, and **(F5)** are implied by existing properties presented in [17], while **(F1)** is implied by a pair of these. We discuss this in more detail next, while investigating which operators from Sec. 5 satisfy which of the new postulates.

We start with **(F0)**, which can readily be seen as a special case of **(SE)**, obtained by only considering 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 3 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. 2 and Fig. 1, since they all satisfy **(SE)**. In [53], F_{strong} and F_{weak} are shown to not satisfy **(F0)**. For F_{sem}, the argument given in [11] 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 considered classes of operators coincide with those for **(SE)** (see Fig.1).

As per **(F1)**, forgetting the same atom(s) should preserve HT-consequence between two programs. As argued in [53], this postulate can be seen as a strengthening of **(F0)**.

Proposition 4 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 [53]. For F_{HT} and F_{SE}, this result follows from Prop. 2 and Fig. 1 and because 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 distinct per se, as it provides a unique set of classes of operators of forgetting for which it is satisfied. In particular, unlike the weaker property **(F0)** and the related **(SE)**, F_{SM} and F_{Sas} do not satisfy **(F1)**, most likely because the premise in the condition for satisfying **(F1)** is weaker than that of **(F0)**.

As argued in [53], it should not matter whether we add new rules before or after forgetting, as long as these rules do not refer to the forgotten atom(s). Similar to **(F0)**, postulate **(F2)** is a special case of one of the properties considered in Sec. 4.

Proposition 5 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 [53] that F_W satisfies **(F2)**. The classes F_{strong}, F_{weak}, F_{HT} and F_{Sas} do satisfy **(F2)**, since they satisfy **(SI)** and by Prop. 2 and Fig. 1. Regarding the negative results, it was proved in [53] that F_S and F_{sem} do not satisfy **(F2)**. For F_{SM} and F_{SE}, the counterexample given in [17] for **(SI)** also applies for **(F2)**. Thus, all results coincide with those of **(SI)**.

In [53], **(F2-)** was introduced as a weakening of **(F2)**. Surprisingly, it turns out to be a special case of **(PP)** by definition of both these properties.

Proposition 6 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. 2 and Fig. 1. Regarding the two negative results, the counterexamples given in [49] for **(PP)** also apply for **(F2-)**. Thus, all results coincide with those of **(PP)**.

In [53], two variations of **(F2)** are considered. One, **(F2')**, is discarded right away as being insufficient to solve the incompatibility between F_S and **(F2)**. The other, **(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.

Postulate **(F3)** encodes that forgetting is meant to simplify the language of a program by removing unwanted atoms. This is reasonable, otherwise, if atoms not occurring in a program were allowed in the result of forgetting, a trivial solution for forgetting would be to simply rename the atoms to be forgotten using such extra atoms.

Proposition 7 *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)**.

Our definition of (classes of) forgetting operators ensures satisfaction of **(F3)**. Hence, similar to **(IR)** (see Sec. 4), it can be omitted from further considerations.

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

Proposition 8 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 [53]. For F_{HT}, this follows directly from the alternative definition of HT-forgetting in [49]. 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 [53], and since the additional derivation rules distinguishing this notion of equivalence and HT-equivalence do not affect the result.

The negative results for F_{Sas} and F_{SM} can be shown with a counterexample based on program $P = \{a \leftarrow p; p \leftarrow not\, not\, p\}$. For any operator in either class of forgetting operators, the result of forgetting about p from P is strongly equivalent to $a \leftarrow not\, not\, a$. However, neither this nor any other rule over $\{a\}$, which has this rule as an 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.

Therefore, this postulate turns out to be of interest as no previously studied property is satisfied by precisely the same set of classes of forgetting operators.

The intuition of **(F5)**, according to [53], is that any rule which is an HT-consequence of the result of forgetting must be an HT-consequence of the program itself in the situations where the atom to be forgotten is not known.

Proposition 9 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 [53]. A similar argument can be used for F_{weak}. For F_{strong} and F_{HT}, the result follows from Prop. 2 and the fact that these classes satisfy **(W)** (cf. Fig. 1). The negative result for F_{sem} was shown in [53]. For F_{SM} and F_{Sas}, consider the program $P = \{a \leftarrow p; p \leftarrow not\, not\, p\}$. Then, for $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}}$ or $\mathsf{f} \in \mathsf{F}_{Sas}$, we have that $\mathsf{f}(P, \{p\}) \equiv_{\mathsf{HT}} \{a \leftarrow not\, not\, a\}$. Therefore, $\mathsf{f}(P, \{p\}) \models_{\mathsf{HT}} a \leftarrow not\, not\, a$, but it is not the case that $P \models_{\mathsf{HT}} a \leftarrow not\, not\, a, not\, p$.

Thus, surprisingly, even though the postulate is implied by the existing property **(W)**, the set of classes of forgetting operators that satisfy it 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. Also, notably, while the properties **(F4)** and **(F5)** are different, they turn out to be satisfied by the same set of known operators. We conjecture that this is so because both are rather closely tied to the concrete definitions of F_S and F_W along which they were introduced.

Finally, **(F6)** encodes the irrelevance of the order in which two atoms are forgotten.

Proposition 10 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 paper where the operator was defined (cf. Sec. 5). The negative result for F_{Sas} follows from the fact that F_{Sas} satisfies **(SP)** which, as shown in [18], implies that in certain cases it is not possible to forget certain atoms. Take $P = \{p \leftarrow not\, not\, p; a \leftarrow p; b \leftarrow not\, p\}$. Forgetting about b from P first is strongly equivalent to removing the third rule, and subsequently forgetting about p is strongly equivalent to $\{a \leftarrow not\, not\, a\}$. 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 [17] which is satisfied by all classes but F_{Sas}. In fact, we will see in the next section that **(F6)** and its generalizations are of interest for open questions related to the property **(SP)** recently investigated in detail in [18], where it was shown that forgetting is not always possible in a meaningful way, shifting the focus to investigating what can be forgotten.

7 Conclusions

We have studied eight postulates of forgetting in ASP introduced in [53], to fill a gap in a recent comprehensive guide on properties and classes of operators for forgetting in ASP, and relations between these [17].

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 properties precisely coincide. This suggests that these three, **(F0)**, **(F2)**, and **(F2-)** can safely be ignored. Postulate **(F3)** can also be safely ignored as it is always satisfied by definition of forgetting operators.

Three of the remaining four 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. They are worth being considered for inclusion in the set of relevant properties as not only they would provide further distinguishing criteria for existing classes of operators, as they would help further clarify the relation between properties **(SE)**, **(W)**, and **(PP)** considered before, and even provide additional means to *axiomatically* characterieze many classes of forgetting operators.

Finally, postulate **(F6)** is not always satisfied, but it seems that this is solely tied to the incompatibility with the crucial property, **(SP)**. Though not fundamental to distinguish known classes of operators, it helped establishing one of the fundamental results of this paper: that even if it is possible to forget a set of atoms, it may be impossible to step-wise iteratively forget its subsets.

Left open, for future work, is the investigation of these postulates for forgetting for semantics other than ASP, such as [49] based on the FLP-semantics [45], or [1, 21] based on the well-founded semantics [13], as well as forgetting in the context of hybrid theories such as [22, 15, 44] and reactive/evolving multi-context systems [16, 5], as well as the development of concrete syntactical forgetting operators that can be integrated in reasoning tools such as [12, 19, 7].

ACKNOWLEDGEMENTS

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 by FCT grant SFRH/BPD/86970/2012.

REFERENCES

[1] José Júlio Alferes, Matthias Knorr, and Kewen Wang, 'Forgetting under the well-founded semantics', in *Procs. of LPNMR*, eds., Pedro Cabalar and Tran Cao Son, volume 8148 of *LNCS*, pp. 36–41. Springer, (2013).

[2] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski, 'Dynamic updates of non-monotonic knowledge bases', *The Journal of Logic Programming*, **45**(1-3), 43–70, (September/October 2000).

[3] 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).

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

[5] Gerhard Brewka, Stefan Ellmauthaler, and Jörg Pührer, 'Multi-context systems for reactive reasoning in dynamic environments', in *Procs. of ECAI*, eds., Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 159–164. IOS Press, (2014).

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

[7] Nuno Costa, Matthias Knorr, and João Leite, 'Next step for nohr: OWL 2 QL', in *Procs. of ISWC*, eds., Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul T. Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, volume 9366 of *LNCS*, pp. 569–586. Springer, (2015).

[8] James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran, 'A model-theoretic approach to belief change in answer set programming', *ACM Trans. Comput. Log.*, **14**(2), 14, (2013).

[9] 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).

[10] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits, 'On properties of update sequences based on causal rejection', *Theory and Practice of Logic Programming (TPLP)*, **2**(6), 721–777, (2002).

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

[12] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider, 'Potassco: The potsdam answer set solving collection', *AI Commun.*, **24**(2), 107–124, (2011).

[13] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf, 'The well-founded semantics for general logic programs', *J. ACM*, **38**(3), 620–650, (1991).

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

[15] Ricardo Gonçalves and José Júlio Alferes, 'Parametrized logic programming', in *Procs. of JELIA'10*, eds., Tomi Janhunen and Ilkka Niemelä, volume 6341 of *LNCS*, pp. 182–194. Springer, (2010).

[16] Ricardo Gonçalves, Matthias Knorr, and João Leite, 'Evolving multi-context systems', in *Procs. of ECAI*, eds., Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pp. 375–380. IOS Press, (2014).

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

[18] Ricardo Gonçalves, Matthias Knorr, and João Leite, 'You can't always forget what you want: on the limits of forgetting in answer set programming', in *Procs. of ECAI*, eds., Maria S. Fox and Gal A. Kaminka. IOS Press, (2016).

[19] Vadim Ivanov, Matthias Knorr, and João Leite, 'A query tool for *EL* with non-monotonic rules', in *Procs. of ISWC*, eds., Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, volume 8218 of *LNCS*, pp. 216–231. Springer, (2013).

[20] 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).

[21] 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).

[22] Matthias Knorr, José Júlio Alferes, and Pascal Hitzler, 'Local closed world reasoning with description logics under the well-founded semantics', *Artif. Intell.*, **175**(9-10), 1528–1554, (2011).

[23] 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).

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

[25] 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).

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

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

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

[29] 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).

[30] João Alexandre Leite, *Evolving Knowledge Bases*, volume 81 of *Frontiers of Artificial Intelligence and Applications, xviii + 307 p. Hardcover*, IOS Press, 2003.

[31] João Alexandre Leite and Luís Moniz Pereira, 'Generalizing updates: From models to programs', in *Procs. LPKR*, eds., Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski, volume 1471 of *LNCS*, pp. 224–246. Springer, (1997).

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

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

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

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

[36] 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).

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

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

[39] 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).

[40] Chiaki Sakama and Katsumi Inoue, 'An abductive framework for computing knowledge base updates', *Theory and Practice of Logic Programming (TPLP)*, **3**(6), 671–713, (2003).

[41] Martin Slota and João Leite, 'Robust equivalence models for semantic updates of answer-set programs', in *Procs. of KR*, eds., Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, pp. 158–168. AAAI Press, (2012).

[42] Martin Slota and João Leite, 'A unifying perspective on knowledge updates', in *Procs. of JELIA*, eds., Luis Fariñas del Cerro, Andreas Herzig,

and Jérôme Mengin, volume 7519 of *LNAI*, pp. 372–384. Springer, (2012).
[43] Martin Slota and João Leite, 'The rise and fall of semantic rule updates based on se-models', *TPLP*, **14**(6), 869–907, (2014).
[44] Martin Slota, João Leite, and Theresa Swift, 'On updates of hybrid knowledge bases composed of ontologies and rules', *Artif. Intell.*, **229**, 33–104, (2015).
[45] Miroslaw Truszczynski, 'Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs', *Artif. Intell.*, **174**(16-17), 1285–1306, (2010).
[46] Hudson Turner, 'Strong equivalence made easy: nested expressions and weight constraints', *TPLP*, **3**(4-5), 609–622, (2003).
[47] Yisong Wang, Kewen Wang, and Mingyi Zhang, 'Forgetting for answer set programs revisited', in *Procs. of IJCAI*, ed., Francesca Rossi. IJCAI/AAAI, (2013).
[48] 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).
[49] Yisong Wang, Yan Zhang, Yi Zhou, and Mingyi Zhang, 'Knowledge forgetting in answer set programming', *J. Artif. Intell. Res. (JAIR)*, **50**, 31–70, (2014).
[50] 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).
[51] Andreas Weber, 'Updating propositional formulas', in *Expert Database Conf.*, pp. 487–500, (1986).
[52] Ka-Shu Wong, 'Sound and complete inference rules for SE-consequence', *J. Artif. Intell. Res. (JAIR)*, **31**, 205–216, (2008).
[53] Ka-Shu Wong, *Forgetting in Logic Programs*, Ph.D. dissertation, The University of New South Wales, 2009.
[54] Yan Zhang and Norman Y. Foo, 'Solving logic program conflict through strong and weak forgettings', *Artif. Intell.*, **170**(8-9), 739–778, (2006).
[55] Yan Zhang and Yi Zhou, 'Knowledge forgetting: Properties and applications', *Artif. Intell.*, **173**(16-17), 1525–1537, (2009).