Iterative Variable Elimination in ASP

Ricardo Gonçalves, Matthias Knorr(✉), and João Leite

NOVA LINCS, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
mkn@fct.unl.pt

Abstract. In recent years, a large variety of approaches for forgetting in Answer Set Programming (ASP) have been proposed, in the form of specific operators, or classes of operators, following different principles and obeying different properties. A recent comprehensive overview of existing operators and properties provides a uniform picture of the landscape, including many novel results on relations between properties and operators. In this paper, we introduce four new properties not considered previously and show that these are indeed succinct and relevant additions providing novel results and insights, further strengthening established relations between existing operators. Most notably among these, the invariance to permutations of the order of forgetting a set of atoms iteratively raises interesting questions with surprising results.

1 Introduction

Forgetting – or variable elimination – is an operation that allows for the removal, from a knowledge base, of so-called *middle* variables no longer deemed relevant, whose importance is witnessed by its application to cognitive robotics [1–3], resolving conflicts [4–7], and ontology abstraction and comparison [8–11]. With its early roots in Boolean Algebra [12], it has been extensively studied within classical logic [4,13–18].

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) [19]. 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. [20–26] – 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 [5,6,27–32]. 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 [28,31,33], others specifically introduced for the case of ASP [6,27–30,32].

E. Oliveira et al. (Eds.): EPIA 2017, LNAI 10423, pp. 643–656, 2017.
DOI: 10.1007/978-3-319-65340-2_53

The result is a *complex* landscape filled with operators and properties, that is difficult to navigate. This problem was tackled in [34] 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, [34] ignores to a large extent a set of postulates on forgetting in ASP introduced by Wong in [27]. This can be justified by the fact that these are limited to forgetting a single atom from a program and often their generalization has been considered independently, and also because, otherwise, they had not played a significant role in the literature on forgetting.

Whereas completing the picture presented in [34] would be sufficient reason to thoroughly investigate these postulates, recent findings in [35] made it even more relevant. It was shown in [35] 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. This also relates to Wong's postulates, notably the one which deals with order in which atoms (or more generally sets of atoms) can be forgotten from a logic program. The possibility to change the order of atoms to be forgotten as well as the step-wise iteration of forgetting a set of atoms comes with a number of benefits. First, designing algorithms for forgetting one atom can often be easier, and forgetting a set of atoms would then simply amount to iteratively apply the resulting simpler algorithm. Second, forgetting one atom can be done without having to worry that this may impose restrictions on the possibility to forget other atom in the future. However, maybe surprisingly, investigating Wong's postulates allowed us to prove that if we want to preserve the property *strong persistence*, then a) the order of forgetting (sets of) atoms matters, and b) it may be impossible to step-wise iteratively forget a set of atoms that can be forgotten as a whole.

In this paper, we introduce new properties, which generalize four of Wong's postulates that are not trivially covered by existing properties in [34]. We show that these new properties turn out to be distinct and provide additional novel results further strengthening the relations between properties and classes of operators of forgetting as previously established. Notably, we clarify the limits of iteratively forgetting a set of atoms and permuting the order of atoms to be forgotten.

After the recalling the necessary notions, we introduce the four new properties and present our results on relations w.r.t. previously established properties and on which classes of operators satisfy which properties. We then investigate one of them with more detail, and establish the novel impossibility result concerning step-wise iterative forgetting, before concluding.

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 shortcuts for $\varphi \supset \bot$ and $\bot \supset \bot$, resp. 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.

Regarding the semantics of propositional formulas, we consider the monotonic logic here-and-there (HT) and equilibrium models [36]. An *HT-interem pretation* 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 $p \in H$; $\quad\quad \langle H, T\rangle \not\models_{\mathsf{HT}} \bot$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \wedge \psi$ 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 (i) $T \models \varphi \supset \psi$,[1] and (ii) $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \Rightarrow \langle H, T\rangle \models_{\mathsf{HT}} \psi$.

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 $\langle X, Y\rangle \sim_V \langle H, T\rangle\}$. 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.[2] Such rules r are also commonly written as

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

and we use both forms interchangeably. We denote the *head* of r by $\mathit{head}(r) = A$ and its *body* by $\mathit{body}(r) = B \cup \neg C \cup \neg\neg D$, a disjunction and a conjunction, respectively.

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

[2] Extended logic programs [37] are actually more expressive, but this form is sufficient here.

As shown by Cabalar and Ferraris [38], 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 have $\mathcal{C}_H \subset \mathcal{C}_n \subset \mathcal{C}_d \subset \mathcal{C}_e$.

We now recall the *answer set semantics* [39] for logic programs. Given a program P and a set I of atoms, the *reduct* P^I is $P^I = \{A \leftarrow B : r \text{ of the form (3) in } P, C \cap I = \emptyset, D \subseteq I\}$. A set I' of atoms is a model of P^I if, for each $r \in P^I$, $I' \models B$ implies $I' \models A$. I is minimal in a set S, denoted by $I \in \mathcal{MIN}(S)$, if there is no $I' \in S$ s.t. $I' \subset I$. I is an *answer set* of P iff I is a minimal model of P^I. Note that, for $\mathcal{C}_n$ 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 [36].

We also recall notions on forgetting from [34]. 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 $\mathsf{f}(P, V) \in \mathcal{C}'$. Often, a set of forgetting operators is conjoined by some characteristic condition/definition. We also call such sets *classes* F *of forgetting operators*.

3 Properties of Forgetting

Previous work on forgetting in ASP has introduced a variety of desirable properties which we recall next [34]. 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 [34], and, implicitly, any F satisfies **(IR)**.

4 Operators of Forgetting

We now review existing approaches to operators of forgetting in ASP following [34].

Strong and Weak Forgetting [5] are based on syntactic operators for normal programs. Both start by computing a reduction corresponding to weak partial evaluation (WGPPE) [40]: for a normal 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, they differ on how to 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 [6] aimed at addressing shortcomings of the previous two syntactic operators, introducing the class F_{sem} for consistent disjunctive programs:[3]

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

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 semantic ones and one syntactic. Unlike the former, the latter is not closed for classes[4] $\mathcal{C}_d^+$ and $\mathcal{C}_n^+$, since double negation is required.

Semantic Strong and Weak Forgetting [27][5] were introduced for disjunctive programs to focus on HT-models,[6] instead of answer sets. For program P and atom a, the set of consequences of P is $Cn(P,a) = \{r \mid r \text{ disjunctive}, P \models_{\mathsf{HT}} r, \mathcal{A}(r) \subseteq \mathcal{A}(P)\}$. $P_S(P,a)$ and $P_W(P,a)$, the results of strongly/weakly forgetting atom a from P, are:

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

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

While both steps are syntactic, different strongly equivalent representations of $Cn(P,a)$ exist, including one based on inference rules for HT-consequence, closed for $\mathcal{C}_d$.

HT-Forgetting [28,31] builds on properties [33] introduced in the context of modal logics, with the aim of overcoming problems with Wongs notions of forgetting [27]. It is defined for extended programs and uses representations of sets of HT-models directly.

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

[3] Actually, classical negation can occur in scope of *not*, but due to the restriction to consistent programs, this difference is of no effect [39], so we ignore it here.

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

[5] It has been shown in [34] that SE-Forgetting [32] coincides with Semantic Strong Forgetting.

[6] Without loss of generality, we consider HT-models instead of SE-models [41] as in [27].

A concrete operator is presented [31] 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 [29] was introduced 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 [30] was introduced 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 $\mathcal{C}$.

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

An operator is defined for programs without disjunction, but not closed for $\mathcal{C}_n$ or $\mathcal{C}_e$.

5 Beyond Wong's Properties

The postulates introduced by Wong [27] were defined in a somewhat different way when compared to the properties presented in Sect. 3. Namely, they only consider forgetting a single atom, were defined for disjunctive programs (the maximal class of programs considered in [27]), 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 [27] (the other one being the non-standard T-equivalence) and in line with previously presented material here and in [34]. We recall these postulates[7] adjusting them to our notation and extending them to the most general class of extended logic programs considered here.

(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 $\mathcal{C}$.

[7] We use the term *postulate* to follow [27] and ease readability. Technically, they are treated as every other *property*.

(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-)**; 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.

Lifting of some of the postulates to the case of forgetting sets of atoms precisely coincides with existing properties, namely **(F0)**, **(F2)**, and **(F2-)** with **(SE)**, **(SI)**, and **(PP)**, resp., hence we will not further consider them. In addition, **(F3)** can also be safely ignored, since the postulate is satisfied already by definition for any class of operators and an extension to forgetting sets of atoms would not affect this.

The remaining are distinct, and thus worth further investigation. We therefore generalize them next, associating each of them with a distinctive name.

(SC) F satisfies *Strong Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}$ and $V \subseteq \mathcal{A}$: if $P \models_{\mathsf{HT}} P'$, then $\mathsf{f}(P, V) \models_{\mathsf{HT}} \mathsf{f}(P', V)$.
(RC) F satisfies *Rule Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$: if $\mathsf{f}(P, V) \models_{\mathsf{HT}} r$, then $\mathsf{f}(\{r'\}, V) \models_{\mathsf{HT}} r$ for some $r' \in Cn_{\mathcal{A}}(P)$.
(NC) F satisfies *Non-contradictory Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$: if $\mathsf{f}(P, V) \models_{\mathsf{HT}} A \leftarrow B \cup \neg C \cup \neg\neg D$, then $P \models_{\mathsf{HT}} A \leftarrow B \cup \neg C \cup \neg V \cup \neg\neg D$.
(PI) F satisfies *Permutation Invariance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$, and $V \subseteq \mathcal{A}$: $\mathsf{f}(P, V) \equiv_{\mathsf{HT}} \mathsf{f}(\ldots \mathsf{f}(P, V_1), \ldots, V_n)$ for every partition $\{V_1, \ldots, V_n\}$ of V.

These new properties indeed generalize their corresponding postulates. Furthermore, some are related to properties previously considered.

Proposition 1. *The following relations hold for all* F*:*

1. **(SC)** *implies* **(F1)**;
2. **(RC)** *implies* **(F4)**;
3. **(NC)** *implies* **(F5)**;
4. **(PI)** *implies* **(F6)**;
5. **(W)** *and* **(PP)** *together imply* **(SC)**;
6. **(SC)** *implies* **(SE)**;
7. **(W)** *implies* **(NC)**.

Notably, 5. and 6. of Proposition 1 generalize 5. of Proposition 1 in [34], and **(NC)** is intuitively weaker than **(SC)** by 5. and 7. of Proposition 1.

One might wonder whether the original postulates also imply these novel generalizations, which turns out not to be the case.

Proposition 2. *The converse of each of 1.–4. in Proposition 1 does not hold.*

Thus, these new properties are indeed more general than the original postulates, which, together with the new results established in Proposition 1 on relations between these and the properties studied in [34], already allows us to conclude that they are indeed interesting additions to the overall landscape of properties of forgetting in ASP.

To complete the picture, we show which operators satisfy which of the new properties, which also allows us to clarify that these properties are indeed distinct.

	sC	**wE**	**SE**	**W**	**PP**	**NP**	**SI**	**CP**	**SP**	**wC**	**SC**	**RC**	**NC**	**PI**
F_{strong}	×	×	×	✓	×	✓	✓	×	×	×	×	✓	✓	✓
F_{weak}	×	×	×	×	✓	×	✓	×	×	×	×	✓	✓	✓
F_{sem}	✓	✓	×	×	×	×	×	×	×	×	×	×	×	✓
F_{S}	×	×	✓	✓	✓	✓	×	×	×	×	✓	✓	✓	✓
F_{W}	✓	✓	✓	×	✓	×	✓	×	×	×	✓	✓	✓	✓
F_{HT}	×	×	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	✓
F_{SM}	✓	✓	✓	×	✓	×	×	✓	×	✓	×	×	×	✓
F_{Sas}	✓	✓	✓	×	✓	×	✓	✓	✓	✓	×	×	×	×

Fig. 1. Satisfaction of properties for known classes of forgetting operators. For class F and property **P**, '✓' represents that F satisfies **P**, and '×' that F does not satisfy **P**.

Proposition 3. *All results for properties* **(SC)**, **(RC)**, **(NC)** *and* **(PI)** *in Fig. 1 hold.*

Thus, **(SC)** 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 **(SE)**, F_{SM} and F_{Sas} do not satisfy **(SC)**, most likely because the premise in the condition for satisfying **(SC)** is weaker than that of **(SE)**. Also, **(RC)** turns out to be of interest as no previously studied property is satisfied by precisely the same set of classes of forgetting operators. Moreover, maybe surprisingly, even though

(NC) 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 **(NC)** also a property of interest in the context of distinguishing existing classes of forgetting operators. Also, notably, while the properties **(RC)** and **(NC)** are different, they turn out to be satisfied by the same set of known operators. We conjecture that this is so because both are closely tied to the concrete definitions of F_S and F_W along which they were introduced. Finally, **(PI)** is distinct and there is no property considered in [34] which is satisfied by all classes but F_{Sas}.

On closer inspection of which classes of operators satisfy **(PI)**, there seems to exist an inherent incompatibility between this property and F_{Sas}, a class of operators that satisfies **(SP)**. The property **(SP)** has been argued to be essential for forgetting in ASP inasmuch as it is the one that adequately encodes the required preservation, under forgetting, of all relations between non-forgotten atoms. However, it was shown in [35] that it is not always possible to forget while preserving **(SP)**, shifting the attention to the question of when it is possible to forget.

One consequence of the negative result for F_{Sas} w.r.t. **(SP)** is that even if it is possible to forget $V \cup V'$, it may not be possible to iteratively forget V and V' in any arbitrary order, which is also why classes such as F_{Sas} cannot satisfy **(PI)**.

Example 1. 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 (as shown in [35]). Hence, the order of forgetting matters for F_{Sas}.

However, this raises another important question, which hasn't been addressed before: *If a set of atoms can be forgotten as a whole, is there at least one partition that permits iterative forgetting?* Previous arguments, such as in Example 1, might suggest that there is, but it turns out not to be the case. Before we state it formally, we recall a piece of notation from [35] denoting the restriction of **(SP)** to a concrete instance: a forgetting operator f over $\mathcal{C}$ satisfies $\textbf{(SP)}_{\langle P,V\rangle}$ if $\mathcal{AS}(\mathsf{f}(P,V) \cup R) = \mathcal{AS}(P \cup R)_{\|V}$, for all programs $R \in \mathcal{C}$ with $\mathcal{A}(R) \subseteq \mathcal{A} \setminus V$.

Theorem 1. *Let P be a logic program, $V \subseteq \mathcal{A}$, and f a forgetting operator that satisfies $\textbf{(SP)}_{\langle P,V\rangle}$. There may not exist any V' with $\emptyset \subset V' \subset V$ such that f satisfies $\textbf{(SP)}_{\langle P,V'\rangle}$.*

Hence, even if it is possible to forget a set of atoms, it may be impossible to forget arbitrary proper subsets of it, in particular, there may not exist a partition of V for which it is possible to step-wise iteratively forget the elements of V.

Corollary 1. *Let P be a logic program, $V \subseteq \mathcal{A}$, and f a forgetting operator that satisfies $\textbf{(SP)}_{\langle P,V\rangle}$. There may not exist any partition of V, $\{V_1, \ldots, V_n\}$, $n > 1$, such that f satisfies $\textbf{(SP)}_{\langle \mathsf{f}(\ldots \mathsf{f}(P,V_1),\ldots,V_{i-1}),V_i\rangle}$, for every $1 < i \leq n$.*

One final note to mention two variants of **(PI)** previously considered in the literature.

The first variant, here named **(PIa)**, was first discussed in [6].

(PIa) F satisfies *Permutation Invariance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$, and $\{p_1, \ldots, p_n\} = V \subseteq \mathcal{A}$: $\mathsf{f}(P, V) \equiv_{\mathsf{HT}} \mathsf{f}(\ldots \mathsf{f}(P, \{p_1\}), \ldots, \{p_n\})$.

Maybe not surprisingly, this variant is implied by **(PI)** though not vice-versa, although, together with **(SE)**, even the converse direction holds.

Proposition 4. **(PI)** *implies* **(PIa)**, *and* **(PIa)** *and* **(SE)** *together imply* **(PI)**.

The second variant, here named **(PIb)**, was first discussed in [29].

(PIb) F satisfies *Permutation Invariance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}$, and $V, V' \subseteq \mathcal{A}$: $\mathsf{f}(P, V \cup V') \equiv_{\mathsf{HT}} \mathsf{f}(\mathsf{f}(P, V), V')$.

Despite appearing less general, it turns out that this variant is equivalent to **(PI)**.

Proposition 5. **(PI)** *is equivalent to* **(PIb)**.

This also shows that our formalization of permutation invariance is indeed well done, as it covers the alternative notions/formalizations existing in the literature.

6 Conclusions

We have studied four new properties of forgetting in ASP (as generalizations of postulates introduced in [27]), to fill a gap in a recent comprehensive guide on properties and classes of operators for forgetting in ASP, and relations between these [34].

Each of the four properties is in fact distinct (even though **(NC)** 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 since 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* characterize many classes of forgetting operators.

Finally, **(PI)** 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 is the investigation of these properties for semantics other than ASP, such as [31] based on the FLP-semantics [42], or [30,43] based on the well-founded semantics, as well as forgetting in the context of hybrid theories

[44–46] and dynamic multi-context systems [47,48], as well as the development of concrete syntactical forgetting operators that can be integrated in reasoning tools such as [49–52].

Acknowledgments. This work was partially supported by Fundação para a Ciência e a Tecnologia (FCT) under UID/CEC/04516/2013, and grants SFRH/BPD/100906/2014 (R. Gonçalves) and SFRH/BPD/86970/2012 (M. Knorr).

References

1. Lin, F., Reiter, R.: How to progress a database. Artif. Intell. **92**(1–2), 131–167 (1997)
2. Liu, Y., Wen, X.: On the progression of knowledge in the situation calculus. In: Walsh, T. (ed.) Proceedings of IJCAI, pp. 976–982. IJCAI/AAAI (2011)
3. Rajaratnam, D., Levesque, H.J., Pagnucco, M., Thielscher, M.: Forgetting in action. In: Baral, C., Giacomo, G.D., Eiter, T. (eds.) Proceedings of KR. AAAI Press (2014)
4. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: formula-variable independence and forgetting. J. Artif. Intell. Res. (JAIR) **18**, 391–443 (2003)
5. Zhang, Y., Foo, N.Y.: Solving logic program conflict through strong and weak forgettings. Artif. Intell. **170**(8–9), 739–778 (2006)
6. Eiter, T., Wang, K.: Semantic forgetting in answer set programming. Artif. Intell. **172**(14), 1644–1672 (2008)
7. Lang, J., Marquis, P.: Reasoning under inconsistency: a forgetting-based approach. Artif. Intell. **174**(12–13), 799–823 (2010)
8. Wang, Z., Wang, K., Topor, R.W., Pan, J.Z.: Forgetting for knowledge bases in DL-lite. Ann. Math. Artif. Intell. **58**(1–2), 117–151 (2010)
9. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Logic-based ontology comparison and module extraction, with an application to DL-lite. Artif. Intell. **174**(15), 1093–1141 (2010)
10. Konev, B., Ludwig, M., Walther, D., Wolter, F.: The logical difference for the lightweight description logic EL. J. Artif. Intell. Res. (JAIR) **44**, 633–708 (2012)
11. Konev, B., Lutz, C., Walther, D., Wolter, F.: Model-theoretic inseparability and modularity of description logic ontologies. Artif. Intell. **203**, 66–103 (2013)
12. Lewis, C.I.: A survey of symbolic logic. University of California Press (1918). Republished by Dover (1960)
13. Bledsoe, W.W., Hines, L.M.: Variable elimination and chaining in a resolution-based prover for inequalities. In: Bibel, W., Kowalski, R. (eds.) CADE 1980. LNCS, vol. 87, pp. 70–87. Springer, Heidelberg (1980). doi:10.1007/3-540-10009-1_7
14. Larrosa, J.: Boosting search with variable elimination. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 291–305. Springer, Heidelberg (2000). doi:10.1007/3-540-45349-0_22
15. Larrosa, J., Morancho, E., Niso, D.: 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)
16. Middeldorp, A., Okui, S., Ida, T.: Lazy narrowing: strong completeness and eager variable elimination. Theor. Comput. Sci. **167**(1&2), 95–130 (1996)
17. Moinard, Y.: Forgetting literals with varying propositional symbols. J. Log. Comput. **17**(5), 955–982 (2007)

18. Weber, A.: Updating propositional formulas. In: Expert Database Conference, pp. 487–500 (1986)
19. Leite, J.: A bird's-eye view of forgetting in answer-set programming. In: Balduccini, M., Janhunen, T. (eds.) LPNMR 2017. LNCS, vol. 10377, pp. 10–22. Springer, Cham (2017). doi:10.1007/978-3-319-61660-5_2
20. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic updates of non-monotonic knowledge bases. J. Logic Program. **45**(1–3), 43–70 (2000)
21. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theory Pract. Logic Program. (TPLP) **2**(6), 721–777 (2002)
22. Sakama, C., Inoue, K.: An abductive framework for computing knowledge base updates. Theory Pract. Logic Program. (TPLP) **3**(6), 671–713 (2003)
23. Slota, M., Leite, J.: Robust equivalence models for semantic updates of answer-set programs. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Proceedings of KR, pp. 158–168. AAAI Press (2012)
24. Slota, M., Leite, J.: A unifying perspective on knowledge updates. In: Cerro, L.F., Herzig, A., Mengin, J. (eds.) JELIA 2012. LNCS, vol. 7519, pp. 372–384. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33353-8_29
25. Delgrande, J.P., Schaub, T., Tompits, H., Woltran, S.: A model-theoretic approach to belief change in answer set programming. ACM Trans. Comput. Log. **14**(2), 14 (2013)
26. Slota, M., Leite, J.: The rise and fall of semantic rule updates based on se-models. TPLP **14**(6), 869–907 (2014)
27. Wong, K.S.: Forgetting in logic programs. Ph.D. thesis, The University of New South Wales (2009)
28. Wang, Y., Zhang, Y., Zhou, Y., Zhang, M.: Forgetting in logic programs under strong equivalence. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Proceedings of KR, pp. 643–647. AAAI Press (2012)
29. Wang, Y., Wang, K., Zhang, M.: Forgetting for answer set programs revisited. In: Rossi, F. (ed.) Proceedings of IJCAI. IJCAI/AAAI (2013)
30. Knorr, M., Alferes, J.J.: Preserving strong equivalence while forgetting. In: Fermé, E., Leite, J. (eds.) JELIA 2014. LNCS, vol. 8761, pp. 412–425. Springer, Cham (2014). doi:10.1007/978-3-319-11558-0_29
31. Wang, Y., Zhang, Y., Zhou, Y., Zhang, M.: Knowledge forgetting in answer set programming. J. Artif. Intell. Res. (JAIR) **50**, 31–70 (2014)
32. Delgrande, J.P., Wang, K.: A syntax-independent approach to forgetting in disjunctive logic programs. In: Bonet, B., Koenig, S. (eds.) Proceedings of AAAI, pp. 1482–1488. AAAI Press (2015)
33. Zhang, Y., Zhou, Y.: Knowledge forgetting: properties and applications. Artif. Intell. **173**(16–17), 1525–1537 (2009)
34. Gonçalves, R., Knorr, M., Leite, J.: The ultimate guide to forgetting in answer set programming. In: Baral, C., Delgrande, J., Wolter, F. (eds.) Proceedings of KR, pp. 135–144. AAAI Press (2016)
35. Gonçalves, R., Knorr, M., Leite, J.: You can't always forget what you want: on the limits of forgetting in answer set programming. In: Fox, M.S., Kaminka, G.A. (eds.) Proceedings of ECAI. IOS Press (2016)
36. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4), 526–541 (2001)
37. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Ann. Math. Artif. Intell. **25**(3–4), 369–389 (1999)

38. Cabalar, P., Ferraris, P.: Propositional theories are strongly equivalent to logic programs. TPLP **7**(6), 745–759 (2007)
39. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3–4), 365–385 (1991)
40. Brass, S., Dix, J.: Semantics of (disjunctive) logic programs based on partial evaluation. J. Log. Program. **40**(1), 1–46 (1999)
41. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. TPLP **3**(4–5), 609–622 (2003)
42. Truszczynski, M.: Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. Artif. Intell. **174**(16–17), 1285–1306 (2010)
43. Alferes, J.J., Knorr, M., Wang, K.: Forgetting under the well-founded semantics. In: Cabalar, P., Son, T.C. (eds.) LPNMR 2013. LNCS, vol. 8148, pp. 36–41. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40564-8_4
44. Knorr, M., Alferes, J.J., Hitzler, P.: Local closed world reasoning with description logics under the well-founded semantics. Artif. Intell. **175**(9–10), 1528–1554 (2011)
45. Gonçalves, R., Alferes, J.J.: Parametrized logic programming. In: Janhunen, T., Niemelä, I. (eds.) JELIA 2010. LNCS, vol. 6341, pp. 182–194. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15675-5_17
46. Slota, M., Leite, J., Swift, T.: On updates of hybrid knowledge bases composed of ontologies and rules. Artif. Intell. **229**, 33–104 (2015)
47. Gonçalves, R., Knorr, M., Leite, J.: Evolving multi-context systems. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) Proceedings of ECAI, pp. 375–380. IOS Press (2014)
48. Brewka, G., Ellmauthaler, S., Pührer, J.: Multi-context systems for reactive reasoning in dynamic environments. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) Proceedings of ECAI, pp. 159–164. IOS Press (2014)
49. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.T.: Potassco: the Potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011)
50. Ivanov, V., Knorr, M., Leite, J.: A query tool for $\mathcal{EL}$ with non-monotonic rules. In: Alani, H., et al. (eds.) ISWC 2013. LNCS, vol. 8218, pp. 216–231. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41335-3_14
51. Costa, N., Knorr, M., Leite, J.: Next step for NoHR: OWL 2 QL. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 569–586. Springer, Cham (2015). doi:10.1007/978-3-319-25007-6_33
52. Lopes, C., Knorr, M., Leite, J.: NoHR: integrating XSB Prolog with the OWL 2 profiles and beyond. In: Balduccini, M., Janhunen, T. (eds.) LPNMR 2017. LNCS, vol. 10377, pp. 236–249. Springer, Cham (2017). doi:10.1007/978-3-319-61660-5_22