TPLP **23** (1): 111–156, 2023.

doi:10.1017/S1471068421000570 First published online 27 December 2021

Forgetting in Answer Set Programming – A Survey

RICARDO GONÇALVES, MATTHIAS KNORR and JOÃO LEITE
NOVA LINCS, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Portugal
(*e-mails:* rjrg@fct.unl.pt, mkn@fct.unl.pt, jleite@fct.unl.pt)

submitted 1 August 2020; revised 7 November 2021; accepted 1 December 2021

Abstract

Forgetting – or variable elimination – is an operation that allows the removal, from a knowledge base, of *middle* variables no longer deemed relevant. In recent years, many different approaches for forgetting in Answer Set Programming have been proposed, in the form of specific operators, or classes of such operators, commonly following different principles and obeying different properties. Each such approach was developed to address some particular view on forgetting, aimed at obeying a specific set of properties deemed desirable in such view, but a comprehensive and uniform overview of all the existing operators and properties is missing. In this article, we thoroughly examine existing properties and (classes of) operators for forgetting in Answer Set Programming, drawing a complete picture of the landscape of these classes of forgetting operators, which includes many novel results on relations between properties and operators, including considerations on concrete operators to compute results of forgetting and computational complexity. Our goal is to provide guidance to help users in choosing the operator most adequate for their application requirements.

KEYWORDS: forgetting, answer set programming, logic programs

1 Introduction

Forgetting is part of the human nature, often with negative connotations, and investigation for its causes started in Cognitive Psychology in the late 19th century (Ebbinghaus 1885). Studies show that forgetting can be caused, among others, by the interference in between new and previously learned information, by decay over time, or because certain links, called cues, used to retrieve a specific memory are not (any longer) available (Shrestha 2017). In fact, recent experiments in Neuroscience reveal that part of this forgetting appears to be done actively, in the sense that the human brain seems to be able to remove information no longer deemed necessary (Davis and Zhong 2017). This enables humans to distinguish important memories among the huge amount of memories they acquire, while abstracting irrelevant details, which allows for making better decisions under varying conditions (Richards and Frankland 2017).

In organizations, *intentional forgetting* plays a similar role (Kluge and Gronau 2018). As increasing amounts of information may become difficult to process, forgetting allows one to simplify the interpretation of the acquired knowledge which contributes to future success of the organization. The methods of forgetting employed build on ideas of

forgetting in Cognitive Psychology, for example, rather than unlearning organizational routines, the retrieval cues are forgotten (Kluge and Gronau 2018).

In the same vein, huge problems appear in Computer Science, where the vastly increasing amounts of data challenge the limits of space in terms of physical storage and processing speed, in particular in situations where the exponential worst-case complexity becomes prohibitive. At the same time, forgetting has become increasingly important to properly deal with legal and privacy issues, such as, for example, the elimination of illegally acquired information to implement a court order, or enforcing the *right to be forgotten*, that is, the right of the eliminination of private data on a person's request, following the EU General Data Protection Regulation (European Parliament 2016).

Thus, forgetting has received increasing attention in Computer Science aiming at the elimination of irrelevant information for improved decision processes and tools (Beierle and Timm 2019). In Artificial Intelligence and Knowledge Representation and Reasoning in particular, the ideas of forgetting can be traced back to Boole's variable marginalization (1854), also called the elimination of middle terms. These notions raised interest when Lin and Reiter (1994) considered forgetting about a fact or a relation in a first-order logic for Cognitive Robotics. This subsequently triggered research into forgetting and closely related notion such as uniform interpolation (Visser 1996), variable elimination (Lang *et al*. 2003), or ignorance (Baral and Zhang 2005), being extended for example to Description Logics (Ghilardi *et al*. 2006; Wang *et al*. 2010; Lutz and Wolter 2011), Planning (Erdem and Ferraris 2007), and Modal Logic (Zhang and Zhou 2009). Commonly two kinds of forgetting can be found in the literature (Eiter and Kern-Isberner 2019). One of them eliminates some formula from a given knowledge base, in a way that closely resembles the operation of contraction in belief revision. The other, more common one, views forgetting as an operation that omits part of the vocabulary of the knowledge base, including possible adjustments on formulas to accommodate some notion of preservation of information (for the remaining vocabulary). This is the kind of forgetting we consider here and particularly useful when we want to eliminate elements representing auxiliary concepts, with the aim to simplify a knowledge base or improve its declarativity, or related to data protection issues. The importance of forgetting in Knowledge Representation and Reasoning is also witnessed by applications in cognitive robotics (Lin and Reiter 1997; Liu and Wen 2011; Rajaratnam *et al*. 2014), for resolving conflicts (Lang *et al*. 2003; Zhang and Foo 2006; Eiter and Wang 2008; Lang and Marquis 2010; Delgrande and Wang 2015), and ontology abstraction and comparison (Wang *et al*. 2010; Kontchakov *et al*. 2010; Konev *et al*. 2012; 2013).

While forgetting has been extensively studied in the context of classical logic (Bledsoe and Hines 1980; Lang *et al*. 2003; Larrosa 2000; Larrosa *et al*. 2005; Middeldorp *et al*. 1996; Moinard 2007; Weber 1986; Gabbay *et al*. 2008), it only recently gathered wider attention in Answer Set Programming (cf. the overview by Eiter and Kern-Isberner (2019)). Answer Set Programming (ASP) (Gelfond and Lifschitz 1988; 1991) provides a declarative rule-based language for knowledge representation and reasoning accompanied with efficient implementations such as CLASP (Gebser *et al*. 2011), DLV (Leone *et al*. 2006; Alviano *et al*. 2017), and Smodels (Simons *et al*. 2002). Its non-monotonic rule-based nature required the development of specific methods and techniques – similar to what happened with other belief change operations such as revision and update (Alferes *et al*. 2000; Eiter *et al*. 2002; Sakama and Inoue 2003; Slota and Leite 2012a;b; Delgrande *et al*.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

2013; Slota and Leite 2014). The result is a significant number of different approaches on forgetting (Zhang and Foo 2006; Eiter and Wang 2008; Wong 2009; Wang *et al*. 2012; 2013; Knorr and Alferes 2014; Wang *et al*. 2014; Delgrande and Wang 2015; Gonçalves *et al*. 2016c; 2017; 2019; Berthold *et al*. 2019b;a; Gonçalves *et al*. 2021), some presenting single operators, others semantic characterizations of classes of operators, usually with different sets of properties deemed desirable, some adapted from the literature on *classical* forgetting (Zhang and Zhou 2009; Wang *et al*. 2012; 2014), others introduced for the case of ASP (Eiter and Wang 2008; Wong 2009; Wang *et al*. 2012; 2013; Knorr and Alferes 2014; Delgrande and Wang 2015; Gonçalves *et al*. 2017; 2019), and often defined for different classes of answer set programs (details on the individual motivations for introducing these approaches and their distinct characteristics can be found in Section 5).

The result is a *complex* landscape filled with classes of operators and properties, with very little effort put into drawing a map that could help us to better understand the relationships between properties and operators. Whereas, in principle, having an operator that obeys *all* of the properties would be desirable, it turns out that any such operator cannot be defined for any class that includes the standard class of normal logic programs (Ji *et al*. 2015). In fact, one of the properties arguably best captures the essence of forgetting, however, there cannot exist an operator that always satisfies it (Gonçalves *et al*. 2016c; Gonçalves *et al*. 2020). This strengthens the idea that there cannot be a one-size-fits-all forgetting operator for ASP, but rather a variety of approaches, each obeying a specific set of properties. The choice of operator will then depend on which properties are deemed more important for the specific application in hand, for which it is important to understand: (a) the relationships between different properties, (b) which properties are obeyed by which (classes of) operators, and even (c) whether some sets of properties make more sense than others. To this end, we present a systematic, comprehensive and thorough survey of *forgetting in ASP*, including many novel results and insights that help answering the raised questions and provide guidelines to users which operators are most suited in which applications.

After a brief section with preliminaries and a section where the common notion of forgetting in ASP is defined, the article is divided into three main sections. The first one contains a comprehensive account of the properties found in the literature, together with an investigation into the relationships between them, including several novel results. The subsequent section is devoted to describing the operators defined in the literature, and establishing some results on their relationships, including an equivalence between two of these operators, as well as considerations on proposed concrete operators, and the computational complexity of problems related to forgetting. Then, we devote our final major section to present a comprehensive account of which properties are satisfied by which operators, some of the results to be found scattered in the literature, but more than half being novel. This section also provides a detailed discussion of the suitability of these classes with respect to their characteristics, and establishes precisely the relationship to uniform interpolation. Finally, the suitability of the classes of operators w.r.t. several applications considered in the context of forgetting is discussed.

This paper is a considerable extension of a previous conference publication (Gonçalves *et al*. 2016b). The material has been extended to incorporate novel approaches in the literature and revised to accommodate these new results and provide a more complete

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

picture of the existing properties and classes of forgetting operators, and their relations. The new detailed material on concrete forgetting operators, computational complexity, the relation to uniform interpolation, and suitability of classes of operators for certain applications help shed further light on these relations, and provide more guidance for the reader. This is complemented with an appendix, available as supplementary material to this paper, that contains all the proofs of the novel results, as well as those in the previous publication, and pointers for all those results spread out over the literature.

2 Answer set programming

In this section, we recall the necessary notions and notation on logic programs under the answer set semantics.

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

$$a_1 \vee \ldots \vee a_k \leftarrow b_1, ..., b_l, not\, c_1, ..., not\, c_m, not\, not\, d_1, ..., not\, not\, d_n\ , \tag{1}$$

where all $a_1, \ldots, a_k, b_1, \ldots, b_l, c_1, \ldots, c_m$, and $d_1, \ldots, d_n$ are atoms of $\mathcal{A}$. Such rules r are also commonly written in a more succinct way as

$$A \leftarrow B, not\, C, not\, not\, D\ , \tag{2}$$

where $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_l\}$, $C = \{c_1, \ldots, c_m\}$, and $D = \{d_1, \ldots, d_n\}$, and we use both forms interchangeably. For each rule r, we distinguish the *head*, $head(r) = A$, and its *body*, $body(r) = B \cup not\ C \cup not\, not\ D$, where $not\, C$ and $not\, not\, D$ represent $\{not\, p \mid p \in C\}$ and $\{not\, not\, p \mid p \in D\}$, respectively. We write the set of atoms appearing in P as $\mathcal{A}(P)$ and the class of (extended) logic programs as $\mathcal{C}_e$. Such extended logic programs are actually equivalent to an even more expressive syntax (Lifschitz *et al.* 1999), but the more concise syntax used here suffices. Also note that double negation is commonly required in the context of forgetting in ASP and commonly supported by answer set solvers such as clingo (Gebser *et al.* 2018).

The class of extended programs includes a number of special kinds of rules r: if $n = 0$, then we call r *disjunctive*; if, in addition, $k \leq 1$, then r is *normal*; if on top of that $m = 0$, then we call r *Horn*; if moreover $l = 0$, then we call r a *fact*. We also admit *constraints*, which are (extended) rules where $k = 0$. The classes of *disjunctive*, *normal*, and *Horn programs*, $\mathcal{C}_d$, $\mathcal{C}_n$, and $\mathcal{C}_H$, are defined as a finite set of disjunctive, normal, and Horn rules, respectively. We have $\mathcal{C}_H \subset \mathcal{C}_n \subset \mathcal{C}_d \subset \mathcal{C}_e$.

We now define the *answer sets* (Gelfond and Lifschitz 1991) of a program, that is, its models, and we recall this notion based on HT-models as defined in the context of the logic of here-and-there (Heyting 1930), the monotonic logic underpinning ASP (Pearce 1999). We start with the notion of *reduct* (Gelfond and Lifschitz 1991) of a program P with respect to a set I of atoms:

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

An *HT-interpretation* is a pair $\langle X, Y\rangle$ s.t. $X \subseteq Y \subseteq \mathcal{A}$. Note that we follow a common convention and usually abbreviate sets in HT-interpretations such as $\{a, b\}$ with the sequence of its elements, ab. Given a program P, an HT-interpretation $\langle X, Y\rangle$ is an

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

HT-model of P if $Y \models P$ and $X \models P^Y$, where $\models$ is the standard consequence relation for classical logic and where programs are interpreted as conjunctions of classical implications

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

corresponding to its rules r of the form (1) where $\neg$, $\wedge$, and $\vee$ denote classical negation, conjunction and disjunction, respectively. We occasionally admit for the sake of readability that the HT-models of a program P are restricted to $\mathcal{A}(P)$ even if $\mathcal{A}(P) \subset \mathcal{A}$. The set of *all HT-models of* P is written $\mathcal{HT}(P)$. Given two programs P_1 and P_2, if $\mathcal{HT}(P_1) \subseteq \mathcal{HT}(P_2)$, then P_1 *entails* P_2 in HT, written $P_1 \models_{\mathsf{HT}} P_2$. Also, P_1 and P_2 are *HT-equivalent*, written $P_1 \equiv_{\mathsf{HT}} P_2$, if $\mathcal{HT}(P_1) = \mathcal{HT}(P_2)$.

A set of atoms Y is an *answer set* of P if $\langle Y, Y\rangle \in \mathcal{HT}(P)$, and there is no $X \subset Y$ such that $\langle X, Y\rangle \in \mathcal{HT}(P)$. We denote by $\mathcal{AS}(P)$ the set of all answer sets of P. For $\mathcal{C}_d$ and its subclasses, all $I \in \mathcal{AS}(P)$ are pairwise incomparable. A program P is *consistent* if $\mathcal{AS}(P)$ is not empty.

Example 1
Consider the following program P:

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

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

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

It is then easy to see that $\{b, d, e\}$ is not an answer set of P (given that $\langle b, bde\rangle$ is an HT-model). Similarly, $\{b, d\}$ is not an answer set of P because $\{b, d\} \not\models P$. In fact, we can verify that $\{b\}$ is the only answer set of P, and that P is therefore consistent.

Different notions of equivalence between programs have been established (Eiter *et al.* 2007), essentially based on comparing their answer sets in different ways. Namely, we say that two programs P_1, P_2 are *equivalent*, written $P_1 \equiv P_2$, if $\mathcal{AS}(P_1) = \mathcal{AS}(P_2)$, that is, their answer sets coincide. Uniform equivalence strengthens this condition by imposing that the equality of answer sets should hold even if a set of facts is added to both programs. Formally, two programs P_1, P_2 are *uniformly equivalent*, written $P_1 \equiv_u P_2$, if $\mathcal{AS}(P_1 \cup R) = \mathcal{AS}(P_2 \cup R)$ for every set R of facts. Strong equivalence further strengthens the condition by imposing equality of answer sets under the addition of any set of rules. Formally, two programs P_1, P_2 are *strongly equivalent* if $\mathcal{AS}(P_1 \cup R) = \mathcal{AS}(P_2 \cup R)$ for every $R \in \mathcal{C}_e$. It is well-known that two programs P_1, P_2 are strongly equivalent exactly when $P_1 \equiv_{\mathsf{HT}} P_2$ (Lifschitz *et al.* 2001), and we therefore use $\equiv_{\mathsf{HT}}$ to denote both equivalences. Relativized equivalence relaxes the condition of strong equivalence by allowing to vary the language of the additional programs. Formally, two programs P_1, P_2 are *relativized equivalent* w.r.t. $V \subseteq \mathcal{A}$, written $P_1 \equiv_V P_2$, if $\mathcal{AS}(P_1 \cup R) = \mathcal{AS}(P_2 \cup R)$ for every $R \in \mathcal{C}_e$ s.t. $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$. Thus, strong equivalence and equivalence can be considered special cases of relativized equivalence, where the considered set V is empty and identical to $\mathcal{A}$, respectively.

Example 2
Consider $P_1 = \{b \leftarrow\}$. Then P_1 and P from Example 1 are equivalent, but neither strongly nor uniformly equivalent, for example, because adding $R = \{c \leftarrow\}$ to both

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

programs yields different answer sets, namely $\{b, c\}$ and $\{a, c, d, e\}$, respectively. Also, neither is an HT-consequence of the other, since $\langle \emptyset, bc\rangle \in \mathcal{HT}(P)$, but not in $\mathcal{HT}(P_1)$, and $\langle b, bd\rangle \in \mathcal{HT}(P_1)$, but not in $\mathcal{HT}(P)$.

On the other hand, $P_2 = \{a \vee b\}$ and $P_3 = \{a \leftarrow not\, b; b \leftarrow not\, a\}$, are well-known to be uniformly equivalent (Eiter and Fink 2003), but not strongly equivalent, for example, for $R = \{a \leftarrow b; b \leftarrow a\}$. We have that $P_2 \models_{\mathsf{HT}} P_3$, but not $P_3 \models_{\mathsf{HT}} P_2$, because of, for example, $\langle \emptyset, ab\rangle$.

Occasionally, we want to omit certain elements of the signature from sets of interpretations and HT-models. Given a set of atoms V, the *V-exclusion* of a set of answer sets (resp. a set of HT-interpretations) $\mathcal{M}$, written $\mathcal{M}_{\|V}$, is $\{X\backslash V \mid X \in \mathcal{M}\}$ (resp. $\{\langle X\backslash V, Y\backslash V\rangle \mid \langle X, Y\rangle \in \mathcal{M}\}$). Also, given two sets of atoms $X, X' \subseteq \mathcal{A}$, we write $X \sim_V X'$ whenever $X\backslash V = X'\backslash V$. For *HT*-interpretations $\langle H, T\rangle$ and $\langle X, Y\rangle$, $\langle H, T\rangle \sim_V \langle X, Y\rangle$ denotes that $H \sim_V X$ and $T \sim_V Y$. Then, for a set $\mathcal{M}$ of *HT*-interpretations, $\mathcal{M}_{\dagger V}$ denotes the set $\{\langle X, Y\rangle \mid \langle H, T\rangle \in \mathcal{M}$ and $\langle X, Y\rangle \sim_V \langle H, T\rangle\}$.

3 Forgetting

In this section, we formally introduce the notion of forgetting in answer set programming. More precisely, we define operators of forgetting and classes of these in a general way that aligns with all the different approaches presented in the literature. We note that our account is on forgetting propositional atoms, just as all the literature on forgetting in answer set programming. This also allows us to capture forgetting atoms from ground programs obtained from programs built over predicate symbols, constants, and variables, simply by considering a one-to-one mapping of such ground atoms to a propositional alphabet. To keep the exposition simple, we here consider forgetting in the propositional setting.

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

Example 3
Consider program P from Example 1:

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

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

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

In addition, if the atom to be forgotten does not appear at the same time in some rule body and some rule head, usually no dependencies need to be preserved. Alternatively, consider forgetting about c from P. Then, since c only appears negated in the body of the rule with head b, c is false. Thus, b becomes unconditionally true when forgetting, and the expected result would be:

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

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Of course, with a fact for b present, the body of the first rule can never be true and, alternatively, we may consider the following program as result of forgetting:

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

It can be verified that both programs are in fact strongly equivalent, that is, both equally preserve the semantics for the remaining atoms.

Thus, forgetting can be viewed as returning a set of programs, which are equivalent in some way, for example, according to one of the notions presented in the previous section, that only mention the remaining atoms and preserve the semantics of the given program over these remaining atoms. In the literature, concrete operators have been defined that, similar to a function, provide one unique such representative for each program P and set of atoms V to be forgotten. We formalize this central idea with the notion of a forgetting operator.

Definition 1
Given a class of logic programs $\mathcal{C}$ over $\mathcal{A}$, a *forgetting operator (over $\mathcal{C}$)* is defined as a function $\mathsf{f} : \mathcal{C} \times 2^{\mathcal{A}} \to \mathcal{C}$ where, for each $P \in \mathcal{C}$ and $V \subseteq \mathcal{A}$, $\mathsf{f}(P, V)$, the *result of forgetting about V from P*,

- is a program over $\mathcal{A}(P) \backslash V$; and
- preserves the semantic relations between atoms in $\mathcal{A}(P) \backslash V$ from P.

We denote the domain of f by $\mathcal{C}(\mathsf{f})$. A forgetting operator f is called *closed* for $\mathcal{C}' \subseteq \mathcal{C}(\mathsf{f})$ if, for every $P \in \mathcal{C}'$ and $V \subseteq \mathcal{A}$, we have $\mathsf{f}(P, V) \in \mathcal{C}'$.

Our definition establishes that forgetting can be understood as a reduction of the language preserving the semantic relations for the remaining atoms in P. The latter is in line with the argument in Example 3 and requires that these semantic relations be established based on some semantic notion such as answer sets or one of the established equivalence notions, though, for the sake of generality, no concrete notion is fixed. This condition allows us to exclude nonsensical functions, such as, for example simply always deleting the entire program or replacing it by arbitrary rules over the remaining atoms, but we do not specify precisely how these semantic relations are established. In the literature, more precise notions for such semantic relations have been defined. For example, Delgrande (2017) defined for arbitrary logics that the result of forgetting corresponds to all the consequences of the given formulas over the remaining language. As we will see in Section 5, this aligns with some of the approaches in the literature of forgetting in ASP, but many others rather rely on for example preserving some notion of models (answer sets or HT-models) in some way, which in fact often turns out to be closely connected to the properties of forgetting in ASP we present in Section 4. For this reason, we abstain here from specifying how these semantic relations are determined specifically and refer to Section 5 for the details for each of the existing approaches.

We point out that the notion of closed operators allows us to indicate whether an operator, when applied to a (sub-)class of programs for which it is defined, does return a program in that same class. This is important, since based on this we are able to answer the question whether such operator can be iterated on such (sub-)class. Naturally, by Definition 1, any forgetting operator is closed for the most general class for which it is defined.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

It is worth noting that some notions of forgetting do not explicitly require that atoms to be forgotten be absent from the result of forgetting, but instead that they be *irrelevant*, that is, the result of forgetting is strongly equivalent to a program that does not mention the atoms to be forgotten. In our view, this is not aligned with the conceptual idea of forgetting itself. However, since such (irrelevant) occurrences of atoms in a result of forgetting are commonly assumed to be not occurring in the result, the required strong equivalence naturally holds. Hence, requiring that forgetting operators yield programs without the atoms to be forgotten does not prevent coverage of these particular approaches.

Now, as Example 3 indicates, preserving the semantics for the remaining atoms is not necessarily tied to one unique program. In fact, in the literature, usually, a representative up to some notion of equivalence between programs is considered, which is also why, in this case, we often refer to *a* result of forgetting (in indefinite terms) as opposed to *the* result. In this sense, many notions of forgetting for logic programs are defined semantically, that is, they introduce a class of operators that satisfy a certain semantic characterization. To capture this, we introduce the notion of a class of forgetting operators.

Definition 2
A *class* F *of forgetting operators (over* $\mathcal{C}$*)* is a set of forgetting operators f, with $\mathcal{C}(\mathsf{f}) \subseteq \mathcal{C}$, that satisfy the semantic characterization of that class.

In this sense, the notion of a class of operators is used as an easy way of referring to all concrete operators that satisfy its semantic characterization, which is also useful in the cases in the literature where only the semantic characterization is presented and no concrete operator is defined. To remain as general and uniform as possible, in this paper, 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.

The subset inclusion for the domain of operators in the previous definition is justified by the fact that there are classes of operators in the literature that include concrete operators only defined for a subclass of the programs considered by the class of operators.

Finally, with respect to uniform interpolation, we note that it is indeed closely connected to the concept of forgetting (Gabbay *et al.* 2011). However, it does not exactly correspond to the notion of forgetting in ASP in the broad sense as considered here. We will discuss this in more detail in Section 6.4 and establish a precise relationship after we have properly presented properties of forgetting, classes of forgetting operators and their relations.

4 Properties of forgetting

In the literature of forgetting in answer set programming, commonly one central focus has been the investigation of guiding principles that would provide desirable characteristics of classes of operators of forgetting, often called properties of forgetting. In this section, we recall these properties found in the literature and investigate existing relations between them.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

In the course of this presentation, we opt for following a historical order without any considerations on their importance or preference among each other. In terms of technical notation, unless otherwise stated, F represents a class of forgetting operators, $\mathcal{C}(\mathsf{f})$ the class of programs over $\mathcal{A}$ of a given $\mathsf{f} \in \mathsf{F}$, and 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.

The first three properties, named Strengthened Consequence, Weak Equivalence, and Strong Equivalence, were proposed by Eiter and Wang (2008), though not formally introduced as such. The first two were in fact guiding principles for defining their notion of forgetting, and formalized later (Gonçalves *et al.* 2016b), while the third was frequently considered in the literature in terms of formal results, but only formalized as a property by Wang *et al.* (2013).

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

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

In other words, forgetting does not admit the introduction of new answer sets, it may only remove some in the course of forgetting.

The other two properties focus on the preservation of some notion of equivalence during forgetting, that is, if two programs are equivalent (w.r.t. some notion of equivalence of programs), then the respective results are as well.

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

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

Weak Equivalence and Strong Equivalence require that forgetting preserves equivalence and strong equivalence of programs, respectively. For other notions of equivalence, no corresponding property has been considered in the literature.

The next four properties, called Irrelevance, Weakening, Positive Persistence, and Negative Persistence, were introduced by Zhang and Zhou (2009) as postulates of knowledge forgetting in the context of modal logics, and later adopted by Wang *et al.* (2012; 2014) for forgetting in ASP.

Irrelevance requires that a result of forgetting be strongly equivalent to a program that does not mention the atoms to be forgotten.

(IR) F satisfies *Irrelevance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, $\mathsf{f}(P, V) \equiv_{\mathsf{HT}} P'$ for some P' not containing any $v \in V$.

This property corresponds to the concept of being irrelevant discussed in the end of the previous section. Thus, satisfaction of this property is an integral part of the definitions of forgetting operators and classes (cf. Definitions 1 and 2).

The other three properties focus on HT-consequences. Namely, Weakening requires that the HT-models of P also be HT-models of a result of forgetting.

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

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

This means that a result of forgetting, $\mathsf{f}(P, V)$, has at most the same consequences as the program P itself.

Positive and negative persistence concern preserving the HT-consequences of P and not introducing new ones, respectively.

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

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

Thus, Positive Persistence requires that the HT-consequences of P not containing atoms to be forgotten be preserved in a result of forgetting, while Negative Persistence requires that a program not containing atoms to be forgotten not be an HT-consequence of $\mathsf{f}(P, V)$, unless it was already a HT-consequence of P.

Essentially in parallel to the appearance of the previous set of properties, Wong introduced a set of properties in his PhD dissertation (2009). They were defined for forgetting a single atom from a given disjunctive program, and did not gather much attention in the literature, possibly due to the form of publication. These properties were generalized to extended programs and to forgetting sets of atoms and assigned a more descriptive name (as Wong simply used alphanumeric identifiers) (Gonçalves *et al*. 2016b;a; 2017). In the course of this generalization, it turned out that two of the resulting properties would precisely coincide with **(SE)** and **(PP)**. Thus, in the following, we will present those generalizations of Wong's properties that are distinct from the previous ones, using the descriptive names for the ease of readability. Namely, we recall, Strong Invariance, Strong Consequence, Rule Consequence, Non-contradictory Consequence, and Permutation Invariance.

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

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

In particular, this means that when computing a forgetting result using an operator from a class that satisfies this property, we can ignore the rules that do not mention the atoms to be forgotten while forgetting, and only add them in the end to this result.

The next three properties are related to HT-consequences. Namely, Strong Consequence requires that HT-consequences be preserved when forgetting.

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

This is similar in spirit to properties **(wE)** and **(SE)**, only here the HT-consequence is preserved while forgetting.

Rule Consequence requires that any rule which is a consequence of a result of forgetting about V from P be a consequence of a result of forgetting about V from a single rule among the HT-consequences of P.

(RC) F satisfies *Rule Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ 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 rule r' such that $\{r'\} \in \mathcal{C}(\mathsf{f})$, and $P \models_{\mathsf{HT}} r'$.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

In particular, since any rule r in a result of forgetting is an HT-consequence of it, for each such rule, there is a rule r' in the original program that gives rise to r (in terms of HT-consequence).

Non-contradictory Consequence requires that whenever a rule r is an HT-consequence of a forgetting result $\mathsf{f}(P, V)$, then the rule obtained by adding the default negation of the atoms of V to the body of r should be an HT-consequence of P.

(NC) F satisfies *Non-contradictory Consequence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, if $\mathsf{f}(P, V) \models_{\mathsf{HT}} A \leftarrow B \cup not\ C \cup not\ not\ D$, then $P \models_{\mathsf{HT}} A \leftarrow B \cup not\ C \cup not\ V \cup not\ not\ D$.

Again, since any rule r in a result of forgetting is an HT-consequence of it, the original program P has such a non-contradictory HT-consequence.

The final property by Wong, Permutation Invariance, requires that the order not be relevant when sequentially forgetting atoms.

(PI) F satisfies *Permutation Invariance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$, and $V \subseteq \mathcal{A}$, we have that $\mathsf{f}(P, V) \equiv_{\mathsf{HT}} \mathsf{f}(...\mathsf{f}(P, V_1), ..., V_n)$ for every partition $\{V_1, ..., V_n\}$ of V.

In fact, such iteration of forgetting had been considered before in a similar manner in a result by Eiter and Wang (2008), but based on an already given sequence of atoms to be forgotten instead of a set. Thus, **(PI)** is slightly more general (Gonçalves *et al.* 2017). Also, independently, a variant of **(PI)** was introduced by Wang *et al.* (2013), but shown to be equivalent (Gonçalves *et al.* 2017). Therefore, in the following, we only use **(PI)** as a representative of these properties of invariance for different orders of forgetting a set of atoms.

The next property, called Existence, was first discussed by Wang *et al.* (2012) and formalized by Wang *et al.* (2013). It requires that a result of forgetting for P in $\mathcal{C}$ be again in the class $\mathcal{C}$. This is important to determine if class of forgetting operators is suitably well-defined for the class of programs it is intended for, as well for iteration on subclasses of programs for which it is defined. We follow the notation introduced in (Gonçalves *et al.* 2016b) which formalizes this property s.t. it be explicitly tied to a class $\mathcal{C}$, thus allowing to speak about a class of forgetting operators F being closed for different classes $\mathcal{C}$.

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

Thus, class F being closed for some $\mathcal{C}$ requires that there exist some "witness in favor of it". This also means that if a class F of operators does not satisfy this property for some class $\mathcal{C}$ of programs, no operator of F can be found that is closed for that class of programs.

The next property, called Consequence Persistence, was introduced by Wang *et al.* (2013) building on the ideas behind **(sC)** by Eiter and Wang (2008). Consequence persistence requires that the answer sets of a result of forgetting correspond exactly to the answer sets of the original program, ignoring the atoms to be forgotten.

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

In other words, forgetting cannot introduce new answer sets nor remove existing ones.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

The following property, called Strong Persistence, was introduced by Knorr and Alferes (2014) with the aim of imposing the preservation of all dependencies contained in the original program building on ideas of strong equivalence between the original program and a result of forgetting (modulo the atoms to be forgotten).

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

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

A further property, Weakened Consequence, is due to results by Delgrande and Wang (2015). It requires that the answer sets of the original program be preserved while forgetting, ignoring the atoms to be forgotten.

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

This property can thus also be understood as being the counterpart to **(sC)**: one does prevent the introduction of new answer sets, the other the loss of existing ones. One can also observe that they correspond to the two inclusions of **(CP)**.

The following two properties, weakened and strengthened Strong Persistence, are similiar in spirit, as they are generalizations of **(wC)** and **(sC)** that correspond to the two inclusions of **(SP)** (Gonçalves *et al.* 2017).

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

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

Weakened Strong Persistence guarantees that all answer sets of P are preserved when forgetting, no matter which rules R over $\mathcal{A}\backslash V$ are added to P, while strengthened Strong Persistence ensures that all answer sets of a result of forgetting indeed correspond to answer sets of P, independently of the added set of rules R.

Finally, Uniform Persistence was introduced by Gonçalves *et al.* (2019) in the context of forgetting in modular answer set programming. Uniform Persistence requires that the correspondence between answer sets of a result of forgetting and those of the original program be preserved in the presence of any additional set of facts not containing the atoms to be forgotten.

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

Hence, **(UP)** can be seen as a variant of **(SP)** under uniform equivalence. Since this is the only property in the literature related to uniform equivalence, we complement with one further property to complete the picture in that regard. It is a variant of a property already presented, namely **(SI)**, but directed towards uniform equivalence.

(UI) F satisfies *Uniform (addition) Invariance* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathsf{f}(P,V) \cup R \equiv_{\mathsf{HT}} \mathsf{f}(P \cup R, V)$ for all sets of facts R with $\mathcal{A}(R) \subseteq \mathcal{A}\backslash V$.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Thus, Uniform Invariance requires that it be (strongly) equivalent to add a set of facts without the atoms to be forgotten before or after forgetting.

Recently, a different relaxation has of **(SI)** has been introduced by Gonçalves *et al.* (2021) when investigating syntactic operators under uniform equivalence.

(**SI$_\mathbf{u}$**) F satisfies *Strong Invariance with respect to uniform equivalence* if, for each $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$ and $V \subseteq \mathcal{A}$, we have $\mathsf{f}(P,V) \cup R \equiv_u \mathsf{f}(P \cup R, V)$, for all programs $R \in \mathcal{C}(\mathsf{f})$ with $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$.

This property relaxes strong invariance by allowing that rules not mentioning the atoms to be forgotten can be ignored and be added to the result still preserving uniform equivalence (and not strong equivalence as **(SI)**). The difference to **(UI)** is that **(UI)** allows us to ignore facts over the remaining language while forgetting (under strong equivalence) whereas (**SI$_\mathbf{u}$**) allows us to ignore rules over the remaining language under uniform equivalence. The latter is arguably more useful when forgetting from a program that contains general rules not mentioning the atoms to be forgotten.

All these properties are not orthogonal to one another, and in what follows we join the results established in the literature on the relations that exist between them. In particular, we opt for presenting these results in a concise way, trying to avoid repetition of results that are implicitly obtained from others.

To ease the reading, for a property **(P)**, we represent with "**(P)**" that "F satisfies **(P)**".

Theorem 1
The following relations hold for all F:

1. **(W)** is equivalent to **(NP)**;
2. **(SP)** implies **(SE)**;
3. **(CP)** and **(SI)** together are equivalent to **(SP)**;
4. **(sC)** and **(wC)** together are equivalent to **(CP)**;
5. **(CP)** implies **(wE)**;
6. **(SE)** and **(SI)** together imply **(PP)**;
7. **(wSP)** and **(sSP)** together are equivalent to **(SP)**;
8. **(sC)** and **(SI)** together imply **(sSP)**;
9. **(wC)** and **(SI)** together imply **(wSP)**;
10. **(W)** and **(PP)** together imply **(SC)**;
11. **(SC)** implies **(SE)**;
12. **(W)** implies **(NC)**;
13. **(wC)** is incompatible with **(W)** for F over $\mathcal{C}$ such that $\mathcal{C}_n \subseteq \mathcal{C}$;
14. **(wC)** and **(UI)** together are incompatible with **(RC)** for F over $\mathcal{C}$ such that $\mathcal{C}_n \subseteq \mathcal{C}$.
15. **(SI)** implies **(UI)**;
16. **(CP)** and **(UI)** together are equivalent to **(UP)**;
17. **(SI)** implies (**SI$_\mathbf{u}$**);
18. **(UP)** is incompatible with (**SI$_\mathbf{u}$**).

Note first, that 1., as proven in by Ji *et al.* (2015), also relies on **(IR)** in its original formulation, in particular, **(W)** is equivalent to **(NP)** and **(IR)**. However, as **(IR)** is an integral part of our definition of forgetting operators, this reliance is ensured implicitly. This means that, by 1., the four properties, originally proposed by Zhang and Zhou

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

(2009), in the context of forgetting in ASP actually reduce to two distinct ones, namely **(W)** and **(PP)**.

The following results, 2.–9., show that **(SP)** is an expressive property, where 2. was shown by Knorr and Alferes (2014) 3.–6. by Gonçalves *et al.* (2016b), and 7.–9. are new. In fact, 3. provides a non-trivial decomposition of **(SP)** into **(SI)** and **(CP)**. These two are themselves expressive, as witnessed by other results. Namely, 4. shows that **(CP)** in turn is the combination of **(sC)** and **(wC)**, and 5. that it implies preservation of equivalence, while 6. provides the non-trivial result that Strong Equivalence and Strong Invariance imply Positive Persistence. Then, 7. provides another decomposition of **(SP)** into **(wSP)** and **(sSP)** which in turn, by 8. and 9., are implied by properties used in the decompositions 3. and 4., respectively. Thus, **(SP)** implies **(SE)**, **(CP)**, **(SI)**, **(sC)**, **(wC)**, **(wE)**, **(PP)**, **(wSP)** and **(sSP)**, where the result for **(PP)** has been shown directly Ji *et al.* (2015).

The next three results, 10.–12., were shown by Gonçalves *et al.* (2017) and clarify the relation between the properties originally proposed by Zhang and Zhou (2009) and those by Wong (2009). Namely, the two distinct properties **(W)** and **(PP)** introduced by Zhang and Zhou (2009) together imply **(SC)** which in turn implies **(SE)**, where the combination of these two results, that is, that **(W)** and **(PP)** together imply **(SE)**, has been shown directly by Gonçalves *et al.* (2016b). In addition, **(W)** alone implies **(NC)**, which indicates that among Wong's properties, **(SC)** is stronger than **(NC)**.

The following two results establish incompatibility results for classes of operators defined over (at least) normal programs. Both results are new, though 13. is a revised result by Wang *et al.* (2013) where incompatibility of **(W)** with **(CP)** is established. In this sense, this new result makes the incompatibility more precise. In comparison to 14., we can also observe that, though no formal relation exists between **(W)** and **(RC)**, **(W)** is stronger than **(RC)** in the sense that it is incompatible with **(wC)**, whereas, in the case of **(RC)**, an additional property is necessary, namely **(UI)**, to establish incompatibility with **(wC)**.

The two new results 15.–16., establish relations between the new properties introduced here w.r.t. uniform equivalence and their correspondents based on strong equivalence. In fact, 15. establishes that Strong Invariance implies Uniform Invariance. The second result provides a decomposition of **(UP)**, and it is interesting as it, together with 3., allows us to trace the difference between **(UP)** and **(SP)** to the different considered form of invariance. This strengthens previous results by Gonçalves *et al.* (2019) that positioned **(UP)** in between **(CP)** and **(SP)**. However, unlike **(SP)**, **(UP)** is incompatible with any property that allows to ignore the remaining arbitrary rules (beyond facts). This is naturally the case for **(SI)**, as **(SI)** relies on strong equivalence whereas **(UP)** relies on uniform equivalence, but, by 18., even the relaxation of **(SI)** to uniform equivalence, ($\mathbf{SI_u}$) (17.), is incompatible with **(UP)** (Gonçalves *et al.* 2021).

To gain a better overview on the results in Theorem 1, Figure 1 summarizes the positive results (all but the incompatibility results) in graphical form, representing equivalences and implications between (compositions of) properties. We can observe that **(SP)** is indeed important as the vast majority of properties is implied by it, further strengthening our view that this property is central as it arguably best captures preserving all the relations between the remaining atoms. Though **(SP)** can in general not be satisfied for classes of programs containing normal programs (Gonçalves *et al.*

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

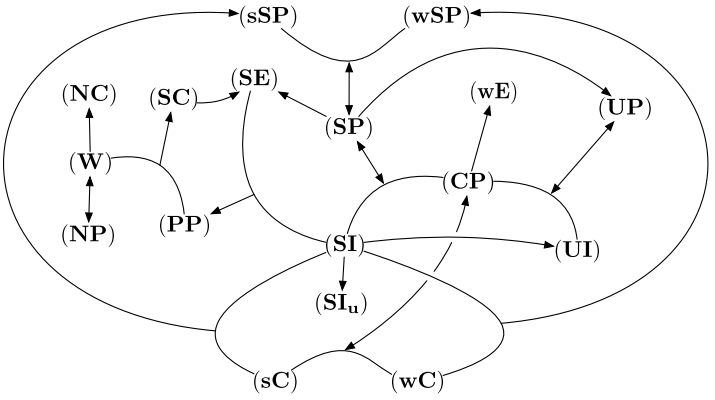

Fig. 1. Relations between properties according to Theorem1: curved lines without arrows join properties and arcs indicate directions of implication.

2016c; Gonçalves *et al*. 2020), in our view, it ideally corresponds to what forgetting in ASP should amount to whenever possible. Now, among the pairs of properties equivalent to **(SP)**, **(SI)** and **(CP)** are arguably more important: the former allows one to focus on the rules containing the atoms to be forgotten when forgetting, which is beneficial for computation and amenable to syntactic forgetting, and the latter captures (the baseline in comparison to **(SP)**) that the answer sets be preserved. While **(SP)** is in general not satisfiable, its relaxation to uniform equivalence, **(UP)**, is (Gonçalves *et al*. 2019). In fact, **(UP)** is the strongest relaxation of **(SP)** w.r.t. programs R such that there is a forgetting operator over a class including normal programs that satisfies it (Gonçalves *et al*. 2021). This makes this property an important alternative, well-suited in the setting of ASP, where problem solutions are often encoded as rules with varying sets of facts. There are also two properties not present in Figure 1 that we deem important, **(PI)** and **(E$_C$)**. The former allows forgetting atoms in any order, which facilitates the usage of forgetting and its implementation in concrete operators that may be defined forgetting one atom at a time. The latter is crucial for guaranteeing that operators (and classes of these) are indeed well-defined as well as permitting the iteration of these. Finally, we note that among properties that are variants of each other with respect to equivalence and strong equivalence, such as **(wE)** and **(SE)**, we consider those based on strong equivalence more important as it is well-known that equivalence does not preserve the structure of rules, and this applies also in the context of forgetting. We revisit these observations in more detail taking into consideration also the existing approaches in the literature and which properties these satisfy (as presented in the following sections).

5 Operators of forgetting

The different properties presented in the literature have often been the driving motivation for the definition of a variety of different approaches on forgetting in answer set programming. We now turn our attention to these classes of operators of forgetting, by first reviewing the approaches found in the literature and then establishing non-trivial relations between them. In the course of the exhibition, we follow a chronological order, similar in spirit to the presentation of the properties in the previous section.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Strong and Weak Forgetting. The first proposals are due to Zhang and Foo (2005); Zhang *et al.* (2005); Zhang and Foo (2006) with the objective to apply forgetting for conflict resolution in the case of inconsistencies as well as to capture and characterize updates of answer set programs. The authors introduced two syntactic operators for normal logic programs, termed Strong and Weak Forgetting. Both start with computing a reduction corresponding to the well-known weak partial evaluation (WGPPE) (Brass and Dix 1999), defined as follows: for a normal logic program P and $a \in \mathcal{A}$, $R(P, a)$ is the set of all rules of the form $head(r_1) \leftarrow body(r_1) \setminus \{a\} \cup body(r_2)$ such that there are $r_1, r_2 \in P$ with $a \in body(r_1)$ and $head(r_2) = a$. Then, the two operators differ on how they subsequently remove rules containing a, the atom to be forgotten. In Strong Forgetting, all rules containing a are simply removed:

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

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

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

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

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

No semantic characterization of the operators was provided, but a simplified representative was created which has the same answer sets as the corresponding forgetting results. The authors also introduced a framework for conflict-solving in answer set programs based on strong and weak forgetting and showed that different approaches of logic program updates can be represented. It was also shown that forgetting does not increase the complexity of determining inferences of logic programs under answer set semantics.

Semantic Forgetting. Building on ideas first exposed by Wang *et al.* (2005), Eiter and Wang (2008) proposed Semantic Forgetting to improve on some of the shortcomings of the two purely syntax-based operators f_{strong} and f_{weak}. Semantic Forgetting introduces a class of operators for consistent disjunctive programs[1] defined as follows:

$$\mathsf{F}_{sem} = \{\mathsf{f} \mid \mathcal{AS}(\mathsf{f}(P, V)) = \mathcal{MIN}(\mathcal{AS}(P)_{\|V}) \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}.$$

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. Thus, preserving the semantic relations between the remaining atoms in the sense of Definition 1 is based on preserving answer sets, that is, certain atoms occur in the same answer sets. The authors showed that their approach can be characterized by forgetting in classical logic (using the minimal models). Three concrete algorithms were presented, two based

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

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

on semantic considerations and one syntactic. Unlike the former two, the latter is not closed for classes $\mathcal{C}_d^+$ and $\mathcal{C}_n^+$ (where the superscript $^+$ denotes the restriction to consistent programs), since double negation is required in general. Hence, it is not a forgetting operator according to our definition. A detailed complexity analysis was provided discussing model checking as well as credulous and skeptical reasoning under forgetting. The authors also presented a framework for resolving conflicts in multi-agent systems which is similar in spirit to that defined by Zhang and Foo (2006) though arguably more general in the sense that it can be adapted more easily than the former. The authors also characterized inheritance logic programs (Buccafurri *et al.* 2002) and update logic programs (Eiter *et al.* 2002) using their forgetting approach.

Semantic Strong and Weak Forgetting. Wong (2009) argued that forgetting in ASP should be characterized by a set of properties, similar as proposed by Eiter and Wang (2008), but rather rely on strong equivalence, as answer sets alone do not contain all the information present in a program nor do they preserve all the information that is unrelated to the forgotten atoms. He defined two classes of forgetting operators for disjunctive programs, building on HT-models.[2] First, given a program P, we define $Cn(P) = \{r \mid r \text{ disjunctive}, P \models_{\mathsf{HT}} r,\ \mathcal{A}(r) \subseteq \mathcal{A}(P)\}$, the set of all consequences of P. We obtain $P_S(P,a)$ and $P_W(P,a)$, the results of strongly and weakly forgetting a single atom a from P, as follows:

1. Consider $P_1 = Cn(P)$.
2. Obtain P_2 by removing from P_1: (i) r with $a \in \mathit{body}(r)$, (ii) a from the head of each r with $\mathit{not}\, a \in \mathit{body}(r)$.
3. Given P_2, obtain $P_S(P,a)$ and $P_W(P,a)$ by transforming certain rules r in P_2 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, that is, $P_S(P,V)$ and $P_W(P,V)$, can be obtained by simply sequentially forgetting each $a \in V$, yielding the following classes of operators.

$$\mathsf{F}_S = \{\mathsf{f} \mid \mathsf{f}(P,V) \equiv_{\mathsf{HT}} P_S(P,V) \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}$$
$$\mathsf{F}_W = \{\mathsf{f} \mid \mathsf{f}(P,V) \equiv_{\mathsf{HT}} P_W(P,V) \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}$$

While steps 2. and 3. are syntactic, different strongly equivalent representations of $Cn(P)$ exist, thus providing different instances. Wong (2009) defined one construction based on inference rules for HT-equivalence, closed for $\mathcal{C}_d$. He also introduced T-equivalence which can be characterized as a weaker form of equivalence by considering only a subset of the inference rules for HT-equivalence, and showed that these correspond to strong and weak forgetting. Thus, F_S and F_W as well as strong and weak forgetting rely on preserving the consequences from P over the remaining atoms, in the spirit of Delgrande's general approach (2017), though for a varying notion of equivalence/consequence. Finally, computational complexity of F_S and F_W was not considered by Wong.

[2] Wong (2009) considered SE-models (Turner 2003). Without loss of generality, we consider the more general HT-models.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

HT-Forgetting. Wang *et al.* (2012; 2014) introduced HT-Forgetting, also termed Knowledge Forgetting (Wang *et al.* 2014), building on work by Zhang and Zhou (2009) in the context of modal logics that proposed certain desirable characteristics of forgetting, which were shown to precisely characterize forgetting in classical propositional logic and modal logic S5. The authors showed that no existing notion would correspond to these ideas, and proposed HT-Forgetting which was defined for extended programs and used representations of sets of HT-models directly.

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

Thus, in this case, the semantic relations between the remaining atoms are preserved based on the equivalence of HT-models. A concrete operator was presented (Wang *et al.* 2014) that was shown to be closed for $\mathcal{C}_e$ and $\mathcal{C}_H$, and it was also shown that no operator exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$. In addition, FLP-forgetting (Wang *et al.* 2014) was considered under the FLP-stable model semantics (Truszczynski 2010), but as this semantics differs from the answer set semantics and the results are essentially identical for both variants considered, we only focus on the answer set semantics. The authors also established results on computational complexity and discussed conflict solving using HT-forgetting based on a framework similar to that used by Eiter and Wang (2008).

SM-Forgetting. Wang *et al.* (2013) defined a modification of HT-Forgetting, SM-Forgetting, for extended programs, with the objective of preserving the answer sets of the original program (modulo the forgotten atoms).

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

Hence, in this case, though the definition is a variation of the one for F_{HT}, the semantic relations between the remaining atoms are preserved based on the equivalence of answer sets (utilizing the correspondence of HT-models). A concrete operator was provided that builds on the notion of countermodels in HT-logic (Cabalar and Ferraris 2007), a technique, in fact, also used by Wang *et al.* (2014) for HT-forgetting, and subsequently in the literature. This class, similar to F_{HT}, was shown to be closed for $\mathcal{C}_e$ and $\mathcal{C}_H$, and it was also shown that no operator exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$. The authors also discussed relations to forgetting in propositional logic and uniform interpolation and considered the computational complexity of model checking, credulous and skeptical inference.

Strong AS-Forgetting. Knorr and Alferes (2014) introduced Strong AS-Forgetting with the aim of preserving not only the answer sets of P itself but also those of $P \cup R$ for any R over the signature without the atoms to be forgotten. The notion was defined abstractly for classes of programs $\mathcal{C}$.

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

The definition of this class is indeed closely aligned with property **(SP)** and naturally, the semantic relations between the remaining atoms are preserved based on the equivalence of answer sets (for varying programs). A concrete operator was presented for a non-standard

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

class of programs (extended programs without disjunction), but not closed for $\mathcal{C}_n$ and only defined for certain programs with double negation. It is therefore not an operator for the class of normal programs according to Definition 1. In parallel, Strong WF-Forgetting for normal programs under the well-founded semantics was considered with favorable results (closed and defined for the whole class), but is not further considered here. The computational complexity was studied, but only in terms of computing a model.

SE-Forgetting. Delgrande and Wang (2015) introduced SE-Forgetting based on the idea that forgetting an atom from program P is characterized by the set of those SE-consequences, that is, HT-consequences, of P that do not mention the atoms to be forgotten. The notion was defined for disjunctive programs building on an inference system by Wong (2008) that preserves strong equivalence. Given that $\vdash_s$ is the consequence relation of this system, $Cn_{\mathcal{A}}(P)$ is $\{r \in \mathcal{L}_{\mathcal{A}} \mid r \text{ disjunctive}, P \vdash_s r\}$. The class is defined by:

$$\mathsf{F}_{\mathsf{SE}} = \{\mathsf{f} \mid \mathsf{f}(P,V) \equiv_{\mathsf{HT}} Cn_{\mathcal{A}}(P) \cap \mathcal{L}_{\mathcal{A}(P)\setminus V} \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}.$$

This notion is clearly aligned with the general notion of forgetting by Delgrande (2017) relying on preserving the logical consequences over the remaining language. An operator was provided, which is closed for $\mathcal{C}_d$, based on computations using resolution, and a prototype implementation was made available. It was observed that, though forgetting a single atom results only in a quadratic blow-up in the size of the program, forgetting several atoms yields an exponential blow-up of the resulting program (in the worst case). Conflict solving was also revisited based on the framework presented by Eiter and Wang (2008), aiming to provide more intuitive/better solutions, mainly due to the fact that HT-consequences were used in opposite to Semantic Forgetting that merely relies on preserving answer sets.

The following three classes are all based on the same idea introduced by Knorr and Alferes (2014), that is, they aim at preserving all the dependencies between atoms not being forgotten in the sense of property **(SP)**, but taking into consideration that it is not always possible to forget and satisfy **(SP)** (Gonçalves *et al.* 2016c). All three approaches rely on the manipulation of HT-models to ensure that the semantic relations between the remaining atoms are preserved, oriented by the idea to maintain all answer sets from the original program whenever this is possible.

To ease the reading and to keep the material self-contained, we recall here the necessary notions, adapted from criterion Ω (Gonçalves *et al.* 2016c), which allows us to determine whether it is possible to forget a set of atoms while satisfying **(SP)**.

Let P be a program over $\mathcal{A}$, $V \subseteq \mathcal{A}$, and $Y \subseteq \mathcal{A}\setminus V$. Consider the following.

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

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

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

The set $Rel^{Y}_{\langle P,V\rangle}$ identifies those $A \subseteq V$ such that $Y \cup A$ is a potential answer set of P, which are therefore relevant for Y being an answer set of the result of forgetting. For

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

each such A, the set $R^{Y,A}_{\langle P,V\rangle}$ collects the V-reduct of the left component of the HT-models of P with right component $Y \cup A$. For each Y, the set $\mathcal{R}^{Y}_{\langle P,V\rangle}$ collects all such sets.

SP-Forgetting. Gonçalves *et al.* (2016c); Gonçalves *et al.* (2020) introduced SP-Forgetting with the aim to satisfy **(SP)** whenever that is possible. The notion was defined for extended logic programs, and, following criterion Ω, tied to the existence of a least element in the set $\mathcal{R}^{Y}_{\langle P,V\rangle}$, which, if it exists, coincides with the intersections over $\mathcal{R}^{Y}_{\langle P,V\rangle}$.

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

A concrete (semantic) operator was defined by Gonçalves *et al.* (2020) based on countermodels, as well as a syntactic operator (Berthold *et al.* 2019b). Both operators and the entire class are closed for $\mathcal{C}_e$ and $\mathcal{C}_H$, and it was shown that no operator exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$.

Relativized forgetting. Gonçalves *et al.* (2017); Gonçalves *et al.* (2020) introduced Relativized Forgetting as a solution to the fact that, in general, the result of forgetting according to SP-Forgetting may have answer sets that do not correspond to answer sets in the original program P. Relativized Forgetting was originally defined using V-HT-models, an extension of HT-models closed related with relativized equivalence (Eiter *et al.* 2007). An alternative characterization (Gonçalves *et al.* 2020), which helps clarify its relation with SP-Forgetting, is used here.

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

Thus, in comparison to SP-Forgetting, a union of the elements in the sets $\mathcal{R}^{Y}_{\langle P,V\rangle}$ is used.

F_M-Forgetting. Gonçalves *et al.* (2017); Gonçalves *et al.* (2020) introduced F_M-Forgetting as an alternative to both SP-Forgetting and Relativized Forgetting, based on the fact that SP-Forgetting may introduce answer sets that do not correspond to those of the original program, while Relativized Forgetting may remove answer sets of the original program, even in cases where it would be possible to forget and satisfy **(SP)**. The difference between F_{SP} and F_{R} lies in the usage of intersection and union in their respective definitions. Whenever $\mathcal{R}^{Y}_{\langle P,V\rangle}$ has more than one element union and intersection will not coincide. Based on this, F_M-Forgetting was formally defined as follows based on a case distinction.

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

For both of the latter classes, a concrete operator was defined based on countermodels, and, in both cases, the operator and the class are closed for $\mathcal{C}_e$ and $\mathcal{C}_H$. Moreover, no operator of the class exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$ (Gonçalves *et al.* 2020). In addition, a comparison between the three previous classes is presented, discussing when

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

to prefer which class of operators, as well as an analysis of the computational complexity (Gonçalves *et al.* 2020).

Uniform forgetting. Uniform Forgetting was introduced by Gonçalves *et al.* (2019) in the context of forgetting in modular answer set programming. In this setting the programs are fixed, and only the input, that is, sets of facts, varies, which is closely related with the notion of uniform equivalence. Uniform Forgetting aims at preserving the answer sets of P no matter what input not containing the atoms to be forgotten is added to P. This implies a careful choice of the HT-models of a result of forgetting, and the formal definition requires additional technical notions which we recall here for self-containedness (in a resumed manner).

$$\begin{aligned} Sel^{Y}_{\langle P,V\rangle} &= \{A \subseteq V \mid \langle Y \cup A, Y \cup A\rangle \in \mathcal{HT}(P)\} \\ T_{\langle P,V\rangle} &= \{Y \subseteq \mathcal{A}\backslash V \mid \text{there exists } A \in Sel^{Y}_{\langle P,V\rangle} \text{ s.t. } \langle Y \cup A', Y \cup A\rangle \notin HT(P) \\ &\qquad\qquad \text{for every } A' \subset A\} \\ N^{Y,A}_{\langle P,V\rangle} &= \{X \setminus V \mid \langle X, Y \cup A\rangle \in \mathcal{HT}(P) \text{ and } X \neq Y \cup A\}. \end{aligned}$$

The sets $Sel^{Y}_{\langle P,V\rangle}$ characterize all the different total HT-models of P for each $Y \subseteq \mathcal{A}\backslash V$. Among these, the ones that give rise to total HT-models of the result of forgetting are given by the set $T_{\langle P,V\rangle}$. For the non-total HT-models of the result of forgetting, we consider the set $N^{Y,A}_{\langle P,V\rangle}$ and the indexed family of such sets $\mathcal{S}^{Y}_{\langle P,V\rangle} = \{N^{Y,i}\}_{i\in I}$ where $I = Sel^{Y}_{\langle P,V\rangle}$. For each tuple $(X_i)_{i\in I}$ such that $X_i \in N^{Y,i}$, the intersection of its sets is denoted as $\bigcap_{i\in I} X_i$, and $SInt^{Y}_{\langle P,V\rangle}$ is the set of all such intersections. Formally, Uniform Forgetting combines the total models from $T_{\langle P,V\rangle}$ and the non-total from $SInt^{Y}_{\langle P,V\rangle}$ as folows:

$$\begin{aligned} \mathsf{F_{UP}} = \{\mathsf{f} \mid \mathcal{HT}(\mathsf{f}(P,V)) = (\{\langle Y,Y\rangle \mid Y \in T_{\langle P,V\rangle}\} \cup \{\langle X,Y\rangle \mid Y \in T_{\langle P,V\rangle} \text{ and} \\ X \in SInt^{Y}_{\langle P,V\rangle}\}), \text{ for all } P \in \mathcal{C}(\mathsf{f}) \text{ and } V \subseteq \mathcal{A}\}. \end{aligned}$$

Again, a concrete operator was defined based on countermodels, which, just like $\mathsf{F_{UP}}$ itself, is closed for $\mathcal{C}_e$ and $\mathcal{C}_H$. A further operator was defined which combines the former with a syntactic one whenever this is possible (Gonçalves *et al.* 2021). Moreover, no operator of the class exists that is closed for either $\mathcal{C}_d$ or $\mathcal{C}_n$. In addition, the effects of applying forgetting to answer set programming modules were studied, as well as the computational complexity.

While all these classes were introduced with differing motivations, they coincide under certain conditions, for example, when restricted to specific classes of programs. This is the case for Horn programs, where the result of forgetting according to most of the classes of operators presented above are strongly equivalent.

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

Notably, of the classes of operators presented in this section, F_{sem} and F_W are the only ones that do not coincide with all others when restricted to Horn programs.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Example 4
Consider the following Horn program $P = \{a \leftarrow e,\ e \leftarrow b,\ b \leftarrow,\ c \leftarrow d\}$. Then, for any f in any of the classes mentioned in Theorem 2, we have that $\mathsf{f}(P, \{e\})$ is strongly equivalent to the program $\{a \leftarrow b,\ b \leftarrow,\ c \leftarrow d\}$. None of the three concrete operators defined for F_{sem} actually satisfies this condition, because the two semantic operators do not consider $c \leftarrow d$ as it is not relevant for the answer sets, while the syntactic one discards this rule in its pre-processing. However, a modification of the syntactic operator is possible such that it coincides with the result (by omitting part of the pre-processing), that is, though F_{sem} does not align with Theorem 2, at least one corresponding operator exists. For F_W, the result completely differs since any operator in F_W must include $\leftarrow b$ in its result.

Interestingly, Wang *et al.* (2012; 2014) additionally showed that, for $\mathcal{C}_H$, the result of F_{HT} is strongly equivalent to that of classical forgetting. We thus obtain as a corollary that this holds for all classes of forgetting operators mentioned in Theorem 2.

Besides the coincidence when restricted to the class of Horn programs, there are two classes of operators that turn out to coincide.

Theorem 3
Consider the class of disjunctive programs. Then, F_S and F_{SE} coincide.

This coincidence can be traced back to the fact that the inference system used for F_{SE} is the same as that used to define the example operator for F_S. Since the two classes coincide, in what follows, we often use F_S to refer to both. This correspondence can be extended to F_{HT} when the result of forgetting is still in the class of disjunctive programs.

Theorem 4
Let P be a disjunctive program, $V \subseteq \mathcal{A}(P)$, $\mathsf{f}_S \in \mathsf{F}_S$, $\mathsf{f}_{\mathsf{HT}} \in \mathsf{F}_{\mathsf{HT}}$, and $\mathsf{f}_{\mathsf{SE}} \in \mathsf{F}_{\mathsf{SE}}$. Then, $\mathsf{f}_S(P, V) \equiv_{\mathsf{HT}} \mathsf{f}_{\mathsf{HT}}(P, V) \equiv_{\mathsf{HT}} \mathsf{f}_{\mathsf{SE}}(P, V)$ whenever $\mathsf{f}_{\mathsf{HT}}(P, V)$ is strongly equivalent to a disjunctive program.

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

Example 5
Given $P = \{a \leftarrow \mathit{not}\, b,\ b \leftarrow \mathit{not}\, a,\ \leftarrow a, b\}$, consider forgetting about b from P. For any $\mathsf{f}_{\mathsf{HT}} \in \mathsf{F}_{\mathsf{HT}}$, it is easy to see that $\mathsf{f}_{\mathsf{HT}}(P, \{b\})$ is strongly equivalent to $a \leftarrow \mathit{not}\, \mathit{not}\, a$, which is not strongly equivalent to any disjunctive program. In the case of F_{SE} and F_S the result of forgetting is, by definition, always a disjunctive program.

This also means that item 1. in Proposition 2 (Delgrande and Wang 2015), which semantically characterizes F_{SE} by asserting that it coincides with the set of HT-models restricted to the remaining atoms, that is, that it coincides with F_{HT} on the class of disjunctive programs, actually does not hold.

Forgetting operators. Concrete forgetting operators have been considered in all the presented approaches, either in the form of a syntax-based operator, namely for F_{strong}, F_{weak}, F_{sem}, F_{Sas}, and F_{SP}, or based on a semantic characterization, namely for F_{sem}, F_{HT}, F_{SM}, F_{SP}, F_{R}, F_{M}, and F_{UP}, or one combining the latter two principles, namely for F_{UP}, or based on a consequence relation, namely F_S, F_W, and F_{SE}.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

On closer inspection, the syntactic operators share a common basis in the form of the principle of weak partial evaluation (WGPPE) (Brass and Dix 1999), which is present in the construction of each of them, even in the realization of the consequence relation used in the case of F_{SE}. Hence, positive occurrences in the rule bodies of atoms to be forgotten are generally treated in the same way. The difference lies in the treatment of negation and of the rules not mentioning the atoms to be forgotten, as well as the applicability of the operator in question. In the case of F_{strong} and F_{weak}, the treatment of negation is not founded on a semantic principle, which causes problems as argued by Eiter and Wang (2008). The other three operators defined for the classes F_{sem}, F_{Sas}, and F_{SP} as well as the syntactic part of the operator defined for F_{UP}, even share the basic ideas for treatment of negation, inspired by the first such approach for F_{sem}. As argued by Berthold *et al.* (2019b), they then differ in how rules not mentioning the atoms to be forgotten are treated. Namely, the operator in F_{sem} often simplifies these away, as its semantic notion is only relying on answer sets. On the other hand the operator defined for F_{Sas} is only applicable in a very restricted setting.

For the semantic operators, two major approaches exist. The operators defined for F_{sem} rely on computing the answer sets, applying the definition of forgetting and finding a canonical program that represents the resulting set of answer sets. All the other approaches rely on the construction based on countermodels (Cabalar and Ferraris 2007), as first used by Wang *et al.* (2013). The benefit is that one can simply determine the set of HT-models of a result of forgetting, and then provide a corresponding program based on countermodels. The difference between these operators resides then only in the characterization of the desired HT-models themselves. While this is an elegant way to obtain a result of forgetting, that can in fact be applied to any approach based on HT-models, the resulting program is often not in a minimal form, containing many unnecessary rules, which requires further non-trivial considerations on minimal programs (Cabalar *et al.* 2007). This also impacts on the similarity between the original program and a result of forgetting, that is, while the semantic characterization is precisely matched, syntactically they may be completely different, even for rules that do not mention the atoms to be forgotten. The latter can be avoided in certain cases, as argued by Gonçalves *et al.* (2020), namely if property **(SI)** is satisfied, as this allows us to exclude the rules not mentioning atoms to be forgotten from the forgetting process, and simply pass them to the result. Still, for the rules involving the atoms to be forgotten, the result may bear no resemblance. The approaches based on a consequence relation do not suffer from this problem, that is, the rules of the program that are not removed while forgetting do persist, they are, however, accompanied by a huge number of additional rules that do not add anything to the program which would require additional processing and simplification. This is what makes providing syntactic operators for classes with a well-defined semantic characterization an important approach. It is not an easy one though.

Computational complexity. We finish the section with considerations on the computational complexity of forgetting. To begin with, all approaches showed or mentioned that computing a concrete result of forgetting with one particular operator is in general in EXP. In addition, many approaches provided arguments and results that show that forgetting is a computationally expensive task, but also argued that this is not surprising given the computational complexity of problems such as model existence in answer set

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Table 1. *Known complexity results for skeptical reasoning under forgetting. All results are completeness results. For class* F *(of operators) and class* $\mathcal{C}$ *(of programs), '×' represents that* $\mathcal{C}$ *has not been considered for this problem for* F*, and '-' that* F *is not defined for* $\mathcal{C}$*.*

	$\mathcal{C}_n$	$\mathcal{C}_d$	$\mathcal{C}_e$
F_{strong}	coNP	–	–
F_{weak}	coNP	–	–
F_{sem}	coNP	Π_2^P	–
F_{HT}	×	×	coNP
F_{SM}	×	×	Π_2^P

programming (Dantsin *et al*. 2001). A general comparison is however not straightforward as several approaches are quite distinct, which impacts on the kind of complexity results that are presented. In the following, we therefore focus on the results presented in the literature, in particular on two problems whose complexity is considered in several approaches, as this makes them suitable for such a comparison. In what follows, we assume a basic understanding of standard complexity classes as well as the polynomial hierarchy.

The first problem is skeptical reasoning under forgetting, that is, is some atom true in all answer sets of a forgetting result $\mathsf{f}(P, V)$. Formally, given $\mathsf{f} \in \mathsf{F}$, $P \in \mathcal{C}(\mathsf{f})$, $V \subseteq \mathcal{A}$, and $a \in \mathcal{A} \setminus V$, we determine $\mathsf{f}(P, V) \models_s a$, where $\models_s$ denotes skeptical inference, that is, truth in all answer sets. This problem has been considered for F_{strong}, F_{weak}, F_{sem}, F_{HT} and F_{SM} for different classes of programs. In fact, for F_{sem}, a can be a literal, and for F_{HT} a formula, but we simplified this aspect here for the sake of the comparison. Figure 1 summarizes the results. We can observe that, for normal programs, F_{strong}, F_{weak}, which are only defined for this class of programs, and F_{sem} coincide. In fact, though it has not been considered explicitly for normal programs, the result for F_{HT} does also coincide (for the larger class of programs) due to hardness of skeptical reasoning for ASP even without forgetting. For F_{SM}, this is not likely due to the maximization check. In general, for classes $\mathcal{C}_d$ and $\mathcal{C}_e$ for which skeptical reasoning without forgetting is Π_2^P-complete, we observe that F_{sem} and F_{SM} are computationally more expensive than the other classes due the additional minimization/maximization which is part of their respective definitions. We note that only F_{sem} and F_{SM} do also consider credulous reasoning. In the former case, the result raises to Σ_3^P, while, in the latter, it remains on the same level of the hierarchy, that is, Σ_2^P.

Thus, requiring that answer sets be preserved comes at a cost in terms of computational complexity. When comparing F_{sem} with both F_{strong} and F_{weak} with their lack of semantic grounds (for part of their construction), arguably the additional cost is preferrable. When comparing F_{HT} and F_{SM}, this is less straightforward, as F_{HT} clearly is based on semantic grounds. Such considerations therefore depend on the intended usage and whether preserving answer sets justifies the additional cost.

The second problem considered is determining whether a given program is indeed a result of forgetting. Formally, given $\mathsf{f} \in \mathsf{F}$, $P, P' \in \mathcal{C}(\mathsf{f})$, $V \subseteq \mathcal{A}$, we determine $P' \equiv_{\mathsf{HT}} \mathsf{f}(P, V)$. This problem has been considered for F_{HT}, F_{SM}, F_{SP}, F_{R}, F_{M}, and F_{UP} and Figure 2 presents the results. It can be observed that the results for F_{HT} and F_{SM} do coincide (unlike the first problem), and that there is an increase in terms of complexity for the classes that relate to property **(SP)**. This is probably not surprising, as F_{SP}, F_{R}, and F_{M} all relate to criterion Ω that establishes whether it is possible to forget and

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Table 2. *Known complexity results for determining whether a given program is a result of forgetting. All results are completeness results but* $\mathsf{F_{UP}}$*, which is only an inclusion.*

$\mathsf{F_{HT}}$	$\mathsf{F_{SM}}$	$\mathsf{F_{SP}}$	$\mathsf{F_{R}}$	$\mathsf{F_{M}}$	$\mathsf{F_{UP}}$
Π_2^P	Π_2^P	Π_3^P	Π_3^P	Π_3^P	Π_3^P

preserve (**SP**) for the concrete combination of program and atoms to be forgotten, which has been shown to be Σ_3^P-complete (Gonçalves *et al*. 2020). The result for class $\mathsf{F_{UP}}$ also points into this direction, but hardness and completeness remain to be shown. Still, the known complexity results on uniform equivalence (Eiter *et al*. 2007) do indicate that it is not likely that an improvement can be achieved in comparison to the other three classes.

Here, the main conclusion is that trying to preserve the answer sets for arbitrary programs (or sets of facts) to be added increases the computational complexity. In our view, this is preferable unless the intended usage does not require it. We will revisit this question in the following section.

6 On the properties of existing operators

Having presented the properties and classes of operators introduced in the literature for forgetting in ASP, in this section, we provide a detailed comparison of these classes with respect to their characteristics. In more detail, we draw a precise picture on the relations between classes of operators and the properties they satisfy. We then discuss the suitability of these classes based on their characteristics, establish concisely the relationship to uniform interpolation, and discuss their suitability w.r.t. several applications considered in the context of forgetting.

6.1 Specific properties

Putting aside for a moment the considerations on the suitability of properties at the end of Section 4, the *desirability* of these properties is, to some extent, in the eye of the beholder. Often, a particular novel approach to forgetting is justified by the fact that previous approaches did not obey some new property deemed crucial, neglecting however that this novel approach actually ended up failing to satisfy other properties, themselves deemed crucial by those who introduced them. Whereas the introduction of most known approaches to forgetting was accompanied by a study of some properties they each enjoyed, there are many missing gaps, some because some properties were only introduced later, others because they were simply neglected. Despite the discussion and potential controversy around the adequacy of the properties, which may ultimately depend on the application at hand, the first and perhaps most important step is to draw an exhaustive picture regarding which properties are obeyed by which classes of operators. This takes us to the central theorem of our paper, illustrated in one easy-to-read table.

Theorem 5
All results in Table 3 hold.

One first observation is that every class of operators obeys a different set of properties (apart from F_S and $\mathsf{F_{SE}}$, which coincide, cf. Theorem 3). This is a strong indication

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Table 3. *Satisfaction of properties for known classes of forgetting operators. For class* F *and property* $\mathbf{P}$*, '✓' represents that* F *satisfies* $\mathbf{P}$*, '×' that* F *does not satisfy* $\mathbf{P}$*, and '-' that* F *is not defined for the class* $\mathcal{C}$ *in consideration.*

	F_{strong}	F_{weak}	F_{sem}	F_{S}	F_{W}	F_{HT}	F_{SM}	F_{Sas}	F_{SE}	F_{SP}	F_{R}	F_{M}	F_{UP}
sC	×	×	✓	×	✓	×	✓	✓	×	×	✓	✓	✓
wE	×	×	✓	×	✓	×	✓	✓	×	×	×	✓	✓
SE	×	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
W	✓	×	×	✓	×	✓	×	✓	✓	×	×	×	×
PP	×	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
SI	✓	✓	×	×	✓	✓	×	✓	×	✓	✓	×	×
SC	×	×	×	✓	✓	✓	×	✓	✓	×	×	×	×
RC	✓	✓	×	✓	✓	✓	×	✓	✓	×	✓	×	×
NC	✓	✓	×	✓	✓	✓	×	✓	✓	×	✓	×	×
PI	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	(✓)
CP	×	×	×	×	×	×	✓	✓	×	×	×	✓	✓
SP	×	×	×	×	×	×	×	✓	×	×	×	×	×
wC	×	×	×	×	×	×	✓	✓	×	✓	×	✓	✓
wSP	×	×	×	×	×	×	×	✓	×	✓	×	×	×
sSP	×	×	×	×	✓	×	×	✓	×	×	✓	✓	×
UP	×	×	×	×	×	×	×	✓	×	×	×	×	✓
UI	✓	✓	×	×	✓	✓	×	✓	×	✓	✓	×	✓
SI$_u$	✓	✓	×	×	✓	✓	×	✓	×	✓	✓	×	×
E$_{\mathcal{C}_H}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E$_{\mathcal{C}_n}$	✓	✓	✓	×	✓	×	×	–	×	×	×	×	×
E$_{\mathcal{C}_d}$	–	–	✓	✓	✓	×	×	–	✓	×	×	×	×
E$_{\mathcal{C}_e}$	–	–	–	–	–	✓	✓	–	–	✓	✓	✓	✓

that these properties play a role in characterizing the classes of operators. In fact, a precise characterization of some classes of operators in terms of the properties they satisfy sometimes exists (Wang *et al*. 2012; 2014; Delgrande and Wang 2015), although this not the case in general.

We now focus on analyzing specific properties and how they relate to the known classes of operators, following commonly the presentation of the properties from left to right according to the historical order established in Section 4.

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

The properties **(SE)**, **(W)**, and **(PP)** have received more attention in the literature, although focussing more on the properties not satisfied by previous approaches to moti-

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

vate the introduction of a new one. As a result, several novel positive results are included in the table. It is perhaps worth pointing out that despite Wang *et al.* (2014) having discussed that F_S and F_W do not satisfy **(PP)**, they did so using a counterexample – Example 5 in this paper – that is not really part of the language for which F_S and F_W are defined, since it relies on rules with double negation as missing consequences, which in our view seems to be an unfair argument. According to our uniformized notions, for the language for which they are defined, they satisfy **(PP)**. In any case, we can observe that **(SE)** and **(PP)** are satisfied by most of the classes, basically only some of the early approaches do not satisfy these, and we can arguably conclude that these are vastly consensual properties of forgetting. The same does not hold for **(W)** where the incompatibility result with **(wC)** (and thus **(CP)**), item 13 in Theorem 1, affects the results.

Example 6
Consider $P = \{a \leftarrow \mathit{not}\, b,\ b \leftarrow \mathit{not}\, c\}$ whose only answer set is $\{b\}$. Thus, according to **(wC)**, a forgetting result must contain at least this answer set modulo the forgotten atoms. For example, for $\mathsf{f} \in \mathsf{F}_{\mathsf{SP}}$, $\mathsf{f}(P, b)$ contains $a \leftarrow \mathit{not}\, \mathit{not}\, c$, whose only answer set is $\{\}$. However, this rule is not an HT-consequence of P, hence **(W)** is not satisfied.

(SI) has received less attention, yet often this non-trivial property is satisfied. Wong (2009) showed **(SI)** for strong and weak forgetting, but using t-equivalence instead of HT-equivalence, whose semantics differs. This explains the negative result in the case of F_S. The negative result for F_{SE} follows by correspondence to F_S, and for F_{SM} from forgetting about b from P as in Example 6: $\mathsf{f}(P, b) \equiv_{\mathsf{HT}} \emptyset$ for $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}}$, so adding $c \leftarrow$ results precisely in a program containing this fact. If we add $c \leftarrow$ before forgetting, then the HT-models of the result of forgetting, ignoring all occurrences of b, correspond precisely to $\langle c, c\rangle$, $\langle c, ac\rangle$, and $\langle ac, ac\rangle$. To preserve the answer sets of this modified program (there is only one – $\{a, c\}$), only the last of these three HT-models can be considered. Hence, $a \leftarrow$ and $c \leftarrow$ (or strongly equivalent rules) occur in the result of forgetting for any $\mathsf{f} \in \mathsf{F}_{\mathsf{SM}}$, and **(SI)** does not hold. Still, as pointed out in Sec. 4, **(SI)** is an important property as it allows one to focus forgetting on the rules that contain the atoms to be forgotten.

For the other three HT-related properties originally proposed by Wong (2009), **(SC)**, **(RC)**, and **(NC)**, the consensus is less obvious. Considering first **(SC)**, we recall from 10. and 11. of Theorem 1 that it is implied by **(PP)** and **(W)** together, and in turn implies **(SE)**. In fact, four of the five classes that satisfy **(SC)**, also satisfy **(PP)** and **(W)**. Only F_W does not satisfy **(W)**, and still satisfy **(SC)** (according to an first example presented by Wang *et al.* (2014)): Consider $P = \{p \leftarrow \mathit{not}\, q; q \leftarrow \mathit{not}\, p\}$. For any $\mathsf{f} \in \mathsf{F}_W$, forgetting about p is strongly equivalent to $q \leftarrow$, which is not a consequence of P itself. Regarding the relation to **(SE)**, we can observe that there are several cases where **(SE)** holds, while **(SC)** does not, showing that both properties are relevant and indicating that **(SC)** can be seen as a stronger version of **(SE)**. Still, given the broad consensus for **(SE)**, arguably **(SE)** seems preferable among the two.

Regarding **(RC)** and **(NC)**, we note that both are always satisfied for the same cases. This could be based on some correlation, but none is known. Arguably, this coincidence could be based on the fact that both are closely tied to the concrete definitions of F_S and F_W along which they were introduced. Actually, inspecting the definitions of both properties, one could consider that **(NC)** implies **(RC)**, that is, that the rule which is an HT consequence in **(NC)** is somehow the rule r' in **(RC)**, but this is not the case in general.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Example 7
Consider F_{strong} which satisfies **(NC)** and $P = \{a \leftarrow b\}$. We have that $\mathsf{f}_{strong}(P, \{p\}) = P$. Since $\mathsf{f}_{strong}(P, \{p\}) \models_{\mathsf{HT}} a \leftarrow b$, we have, by **(NC)**, that $P \models_{\mathsf{HT}} a \leftarrow b, not\, p$. But we also know that $\mathsf{f}_{strong}(\{a \leftarrow b, not\, p\}, \{p\}) = \emptyset$, and, therefore $\mathsf{f}_{strong}(\{a \leftarrow b, not\, p\}, \{p\}) \models_{\mathsf{HT}} a \leftarrow b$ does not hold. This shows that there are cases where r' of **(NC)** is not the rule for property **(RC)**.

Nevertheless, item 12. of Theorem 1 indicates that **(W)** implies **(NC)**, and there are several cases where the latter is satisfied and the former is not, indicating that **(W)** has a more restrictive condition than **(NC)**. This is also corroborated by the two incompatibility results in Theorem 1 and the apparent coincidence of satisfaction for **(RC)** and **(NC)** in Table 3, in the sense that **(W)** also has a more restrictive condition than **(RC)**.

Property **(PI)** is also widely accepted, that is, a set of atoms can be forgotten in any order. The only exceptions are the three classes closely tied to **(SP)**, F_{SP}, F_{R}, and F_{M}, essentially, because it is not always possible to forget and preserve **(SP)** (Gonçalves *et al.* 2016c). In such situations, the three classes then provide approximations of forgetting while preserving **(SP)**, and, since the order in which atoms are forgotten may affect whether forgetting is possible (while preserving **(SP)**) (Gonçalves *et al.* 2017), the property does not hold for any of them. Regarding F_{UP}, we note that a weaker version of **(PI)** was proven (Gonçalves *et al.* 2019), showing that it is possible to iterate the operators of the class when applied in the context of modular logic programming. Thus strictly speaking the exact result is open. On the other hand, in the context of modular answer set programming, directed towards uniform equivalence, this result suffices, which explains the particular notation of the result in Table 3.

The new negative results for **(CP)** and **(SP)** can be illustrated with forgetting about b from $P = \{a \leftarrow not\, b,\ b \leftarrow not\, a\}$, that is, the first two rules of Example 5. Since $\mathcal{AS}(P) = \{\{a\}, \{b\}\}$, the result must have two answer sets $\{a\}$ and $\emptyset$, which is not possible for disjunctive programs obtained from operators in F_S, F_W, and F_{SE}. The same example serves as counterexample for all negative results of **(wC)**, while positive results follow for classes satisfying **(CP)** from Theorem 1. Notably, the counterexample also applies to F_{SE}, thus invalidating Theorem 2 in the paper of Delgrande and Wang (2015).

Regarding **(SP)**, only F_{Sas} satisfies the property due to the way the class is defined. However, an important note is in order. Namely, unlike previously stated by Gonçalves *et al.* (2016b; 2017; 2019), in Table 3, all results are positive for F_{Sas}. The following consideration reveals the cause for this discrepancy. Many of the negative results previously stated for F_{Sas} are based on counterexamples using the concrete operator defined in (Knorr and Alferes 2014), among them, for example, that F_{Sas} does not satisfy **(UP)** (Gonçalves *et al.* 2019). However, items 3., 15., and 16. of Theorem 1 allow us to show that **(SP)** implies **(UP)**, seemingly a contradiction. The reason why this is not a problem is revealed by a close inspection of the definition of class F_{Sas}. The operator defined by Knorr and Alferes (2014) is actually not defined for a standard class of programs, nor for all programs in this class which is in conflict with the quantification applied in the definition. This is complemented by the general result that forgetting and satisfying **(SP)** is not always possible for programs that include normal programs. These observations lead

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

to the following corollary, strongly restricting the class of programs to which operators in F_{Sas} can be applied.

Corollary 1
There is no forgetting operator $\mathsf{f} \in \mathsf{F}_{Sas}$ with $\mathcal{C}_n \subseteq \mathcal{C}(\mathsf{f})$.

Thus, the operator defined by Knorr and Alferes (2014) does not belong to F_{Sas} and all previously negative results for F_{Sas} are invalidated, allowing that this class indeed satisfies all the properties in the literature considered here, but only within the scope of Horn programs.

Regarding **(wSP)** and **(sSP)**, these properties are essentially exclusive to the **(SP)**-related classes of forgetting, which is not surprising given item 7. of Theorem 1, with the exception of **(sSP)** for F_W, as this class satisfies both **(sC)** and **(SI)**, which implies satisfaction of **(sSP)** by item 8. of Theorem 1.

Concerning **(UP)** this is indeed also a strong property, as it is essentially only satisfied by the class for which it was defined, besides F_{Sas}, of course. In fact, **(UP)** can be seen as an adaptation of **(SP)** to uniform equivalence. Thus, despite the reduced number of classes that satisfy it, it is an important property as it shows that the conceptual idea of **(SP)** can be satisfied in the scope of general programs (not restricted to Horn), if one only varies facts for the sake of preserving the dependencies for the atoms not to be forgotten. This is also well-aligned with a central idea of answer set programming, where the general specification of a problem is encoded as an answer set program which is combined with different set of facts, representing an instance of the problem one wants to solve.

With respect to **(UI)**, we observe that satisfaction is vastly sanctioned by the fact that it is implied by **(SI)** (cf. 15 of Theorem 1), with the exception of F_{UP} for which the positive result is a result of 16. of Theorem 1. We exemplify the negative result for the case of F_S. Consider forgetting about p from $P = \{a \leftarrow not\, p, b;\ p \leftarrow not\, a;\ \leftarrow p, b\}$. This program has $a \leftarrow not\, not\, a, b$ as an HT-consequence, which is however not a disjunctive rule and thus not captured in the construction. So adding $b \leftarrow$ to the program makes a true, and forgetting preserves that. If we forget first, that connection is lost and we cannot conclude a by adding $b \leftarrow$ to the result of forgetting.

Although $(\mathbf{SI_u})$ is a weakening of **(SI)**, the results for $(\mathbf{SI_u})$ are often a consequence of those for **(SI)**. The positive ones follow immediately from the fact that $(\mathbf{SI_u})$ is implied by **(SI)** (cf. 17 of Theorem 1), and most of the counterexamples for the negative results are similar to those of **(SI)**. An interesting exception is the case of F_{M}, for which the negative result for **(SI)** can be easily justified by the satisfaction of **(CP)**, as these properties together are equivalent to **(SP)**. In the case of $(\mathbf{SI_u})$ the same argument cannot be used, since this property together with **(CP)** is not enough to imply **(SP)**. Rather an advanced counterexample can be found based on HT-models in which the result is different from that of F_{SP} and F_{R}, which both satisfy $(\mathbf{SI_u})$ (for the details we refer to the appendix).

Finally, the results on $(\mathbf{E}_{\mathcal{C}})$ for different classes of programs $\mathcal{C}$ reveal that all classes of operators are closed for $\mathcal{C}_H$, and, in addition, each F is closed for the maximal class of programs considered, but often not for intermediate ones, with the exception of F_{sem} and F_W. Interestingly, the two known semantic operators in F_{sem} are not closed for $\mathcal{C}_H$, while the syntactic one is, despite not being closed in general and thus not an operator

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

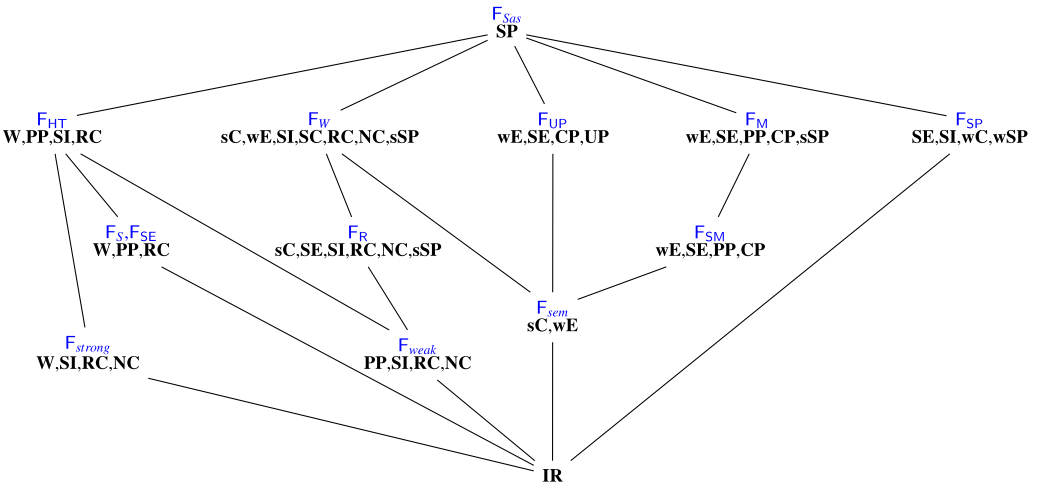

Fig. 2. Sets of properties satisfied by known classes of forgetting operators, where each connection represents that the above set of properties implies the one below (with (**SP**) and (**W**) restricted to Horn programs).

according to 1. This can be remedied by defining an additional operator specific to Horn programs, which is a simplification of forget_1 (Eiter and Wang 2008) taking advantage of the fact that only one answer set exists. Note that '-' was used w.r.t. to the definitions of each F: the singleton classes F_{strong} and F_{weak} are precisely defined for normal programs; the intuition behind minimization embedded in F_{sem}'s definition does not combine well with double negation; and, for F_S, F_W, and F_{SE}, the consequence relation that is applied is defined for disjunctive rules.

6.2 Classes of operators

The results in Table 3 provide us with valuable information to compare classes of operators, as well as some guidelines regarding the choice of a forgetting operator.

The first concern is perhaps the required class of programs $\mathcal{C}$. If some class of operators is not closed or even not defined for $\mathcal{C}$, then it is certainly not a good choice. Of course, as already mentioned, nowadays, existing ASP solvers have no problem with accepting the full syntax of extended programs here considered, hence, there is no impediment in that regard. Rather, if the application in question requires a certain class of programs, then this may discard certain classes of forgetting operators as possible choices. Hence, many classes defined for extended programs may face difficulties if normal or disjunctive programs are required (cf. the occurrences of "×" in Table 3). At the same time, F_{Sas} is clearly not suitable with its restriction to Horn programs, while F_{strong} and F_{weak} present considerable limitations if disjunctions are required, since it is well-known that these cannot be represented in a strongly equivalent normal program; and F_{sem}, F_S, and F_W may require additional effort to represent double negation such as $a \leftarrow not\ not\ a$ by $a \leftarrow not\ aux$ and $aux \leftarrow not\ a$ using an additional auxiliar atom aux, where introducing new atoms to be able to forget others seems counterintuitive.

With these considerations on $(\mathbf{E}_{\mathcal{C}})$ in mind, we can analyze the remaining properties. To that end, Figure 2 presents a lattice of inclusions between the sets of properties satisfied by each known class of forgetting operators, taking into consideration the following:

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

- Properties on existence, that is, **(E$_C$)**, as well as **(PI)** are not considered in this lattice. The reasons are that existence can be easily handled orthogonally, and the same is true for the few cases where **(PI)** does not hold. Moreover, this allows obtaining more interesting observations that would otherwise be obfuscated.
- For the sake of readability, not all properties are made visible in each case. Rather, taking advantage of the results obtained in Theorem 1, only the necessary ones are presented, those that are implied are left implicit. For example, for F_W, which satisfies both **(SC)** and **(SE)**, only **(SC)** is shown, while **(SE)** is left implicit.
- As F_{Sas} satisfies all the properties, it represents the top element of the lattice, in a certain sense the ideal case (even though this turned out to only be possible for Horn programs). To complement this, property **(IR)** which is true for any notion of forgetting, has been chosen as the bottom element.

Figure 2 makes it apparent that there is one kind of property that divides the classes into two groups, namely whether some kind of relation between the answer sets of the original program and those of its result of forgetting holds or not. The classes for which this is the case are F_{sem}, F_W, F_{SM}, F_{Sas}, F_{SP}, F_{R}, F_{M}, and F_{UP} by either satisfying **(sC)** or **(wC)** or a property that implies these.

Among them, putting F_{Sas} aside, as it satisfies all the properties which turns out to not provide considerable insights, the two classes that satisfy **(CP)** and **(PP)** turn out to be separated by only one property as follows (using semi-formal notation):[3]

$$\mathsf{F}_{\mathsf{SM}} + (\mathbf{sSP}) \rightsquigarrow \mathsf{F}_{\mathsf{M}}.$$

Hence, the difference between these two classes resides in the preservation of answer sets, no matter which rules are added (over the remaining atoms).

Another such close relationship in terms of satisfied properties can be established between F_W and F_{R} as follows:

$$\mathsf{F}_{\mathsf{R}} + (\mathbf{SC}) + (\mathbf{wE}) \rightsquigarrow \mathsf{F}_W.$$

Thus, the difference between these two classes lies in strengthening property **(SE)** (using **(SC)**) and adding preserving equivalence while forgetting.

We can even take this one step further and state the following:

$$\mathsf{F}_{\mathsf{R}} + \mathsf{F}_{sem} + (\mathbf{SC}) \rightsquigarrow \mathsf{F}_W.$$

However, note that picking an operator of F_{R} and somehow enforcing **(SC)** and **(wE)** will not provide an operator of F_W. It can be shown that the effect of forgetting V from P for $\mathsf{f} \in \mathsf{F}_W$ yields a result that replaces all $v \in V$ as if they were false, independently of the actual rules in P. For example, forgetting about p from $p \leftarrow$ would yield $\bot$, which is not aligned with the original idea of forgetting, that is, removing all $v \in V$ without affecting other derivations, nor with the idea of class F_{R} in particular, and neither **(SC)** nor **(wE)** will change that. In general, an operator of a superclass does not necessarily belong to a subclass in the hierarchy of Figure 2. Still, there are cases where this is true, for example, $\mathsf{f} \in \mathsf{F}_{\mathsf{UP}}$ naturally satisfies the defining condition of class F_{sem}, and thus belongs to the class.

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

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Also note that the three classes related to **(SP)** F_{SP}, F_{R}, F_{M}, place themselves quite differently in this picture. Recall that these three classes, in the face of the result that it is not always possible to forget while satisfying **(SP)** (Gonçalves *et al.* 2016c), are considered relaxations of the equivalence, stating that **(SP)** is composed of **(sC)**, **(wC)**, and **(SI)** (Gonçalves *et al.* 2020). Each of the three classes then basically is relaxed by dropping one of these three properties. Given that these three properties imply different other properties (cf. Theorem 1), this relation becomes less apparent in our lattice representation. Still, some observations can be made. F_{M}, which satisfies both **(sC)** and **(wC)**, aligns well with approaches that preserve answer sets. F_{R}, which satisfies both **(sC)** and **(SI)**, fits within the classes that satisfy **(SI)**. Finally, F_{SP}, which satisfies both **(wC)** and **(SI)**, does not relate to any class other than F_{Sas}, essentially because a) it is the only class that satisfies **(wC)** and not **(sC)**, and b) it satisfies a set of properties distinct from any of the classes that satisfy **(CP)**.

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

For the third node without this preservation support, $\mathsf{F}_S/\mathsf{F}_{\mathsf{SE}}$, there is also a close proximity to F_W based on their definition, yet, their characterizations differ substantially. Both satisfy **(SE)** and **(PP)**, but differ on six other properties, which means that in this case the variation in the definition has a much more profound effect on the set of satisfied properties. At the same time, we already know that there is a close relation to the remaining class F_{HT} as witnessed in Theorem 4. This is matched by a close correspondence in terms of satisfied properties.

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

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

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

Now, in the spirit of describing classes of forgetting operators, for several classes, a defining characterization in terms of satisfied properties has been introduced in the literature. But providing operators that precisely satisfy only a certain set of properties can also be used to show that some sets of properties do not suffice to characterize a class:

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

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

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

We mention examples of further operators that can be defined in a similar style, providing evidence that certain sets of properties alone are probably of little interest as they are satisfied by absurd operators. In fact, several of them do not even correspond to the definition of a forgetting operator, as no semantic relations between the remaining atoms are preserved.

- **(IR)** : delete all rules; then add some arbitrary rules over the remaining alphabet (after forgetting);[4]
- **(W)** : delete all rules with atoms to be forgotten and an arbitrary 50% of the remaining rules;
- **(PP)** : perform weak forgetting; in the resulting program, pick an arbitrary set of rules and turn them into facts by removing their body;
- **(SE)** : compute all answer sets; remove all atoms to be forgotten from them; create a set of facts that represents the intersection of all such reduced answer sets;
- **(SI)** : perform the WGPPE replacement step of strong and weak forgetting; in the resulting program, arbitrarily delete rules with negative occurrences (in the body) of atoms to be forgotten, or just remove their bodies, and delete all rules with positive occurrences (in the body or head) of atoms to be forgotten;
- **(SE)**,**(W)** : delete the entire program;
- **(SE)**,**(PP)** : add, as facts, to the result of performing SE-forgetting, the atoms that belong to some answer set of the original program and for which there is some rule of the original program that contains some negative and no positive occurrence of forgotten atoms.

Thus, often a meaningful choice of a class of forgetting operators requires looking at more than one property in combination with the rationale behind their definition, in particular for cases such as F_{sem} where the number of satisfied properties is comparably small.

Finally, while the results in Theorem 5 for **(E$_{\mathcal{C}}$)** naturally differentiate between the classes of programs $\mathcal{C}$ considered, the remaining properties are stated for the most general class of programs for which the class of operators is defined. This begs the question whether restricting $\mathcal{C}$ would affect the results shown in Table 3, which we will now address to gain further insights.

We first consider Horn programs, for which we already know that all classes of operators are closed. From Theorem 2, we know that several classes that satisfy different sets of properties in the general case actually coincide. As expected, these classes all satisfy the same set of properties when restricted to Horn programs. The reason can be traced back to the incompatibility result 13. in Theorem 1, only stated for program classes above $\mathcal{C}_H$, that is, it does not apply here.

Theorem 6
For Horn programs, the following holds:

- F_{strong}, F_{weak}, F_S, F_{HT}, F_{SM}, F_{Sas}, F_{SE}, F_{SP}, F_{R}, F_{M}, and F_{UP} satisfy **(W)**, **(RC)**, **(SP)**, and **(PI)**;

[4] The term "arbitrary" is used freely to represent some deterministic set, for example, the first elements of some specific ordering.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

- F_{sem} satisfies **(CP)** and **(PI)**;
- F_{W} satisfies the same properties as in the general case.

Actually, only the minimally necessary properties are mentioned, the remaining can be obtained from Theorem 1. This means, in particular, that the first group of classes satisfies all the presented properties (with $\mathbf{(E_C)}$ limited to $\mathbf{(E_H)}$, of course), and, that reducing to Horn programs does not change the set of properties satisfied by F_{W}.

For normal programs, it turns out that the introduction of negation in the body immediately makes the result coincide with the general one for all the classes (except for F_{Sas} which is not defined for such classes). This is witnessed by the fact that all counterexamples are normal programs, in particular those mentioned in the previous section.

6.3 On desirability of classes of operators

The plethora of results presented so far spread out over the previous sections, and the many differentiating details discussed with respect to these make it difficult to assess which classes of operators are in the end more important. To aid the reader in that regard, we present some summarizing considerations, guided mainly by the definition of forgetting operators, that is, the idea that the semantic relations between the remaining atoms be preserved.

We start by identifying several classes of forgetting operators which in our view are less important, and can be dismissed for the remaining discussion. We note that in listing these, we follow the chronological order only, without any preference associated to this order.

- F_{strong} and F_{weak} can be dismissed, as their semantic consequence relation is non-standard (T-equivalence), resulting in forgetting results that do not even align with answer sets. Moreover, they are only defined for normal programs which severely limits the application.
- F_{sem} can be dismissed, as the characterization based on answer sets only is semantically too weak, not even aligning with expected results for Horn programs in general. It is true that one can find concrete operators that overcome part of these problems, but not in general.
- F_{W} can be dismissed because, even though the class satisfies many properties, it cannot coincide with intended results even for Horn programs, that is, the semantic relations preserved over the remaining atoms are not aligned with our expectations.
- F_{Sas} can be dismissed, as it is only defined for Horn programs, which does not fit the intention of forgetting in ASP, where default negation is fundamental. The interest in the approach lies rather in the introduction of the idea of **(SP)** itself.

The remaining seven classes, F_{S}, F_{HT}, F_{SM}, F_{SP}, F_{R}, F_{M}, and F_{UP}, all coincide for Horn programs, so further considerations are needed. If we look for a consensus among the satisfied properties, then we note that **(PP)** and **(SE)** are basically satisfied by all of them (with the exception of F_{UP} which is justified by the fact that this approach focusses on uniform equivalence). However, these two properties alone do not suffice because, as we have seen, corresponding absurd operators can be defined. To facilitate a comparison, we can separate them according to how semantic relations between the remaining atoms are preserved.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Preserving strong equivalence. This is the strongest form of preservation of semantic relations, as it aims to approximate satisfying **(SP)**, and the classes F_{SP}, F_{R}, and F_{M} correspond to this. A preference among these three strongly depends on which of the individual distinguishing characteristics one prefers. This relies mainly on the satisfied properties, as the computational complexity and the classes of programs to which these can be applied do coincide. These differences have been spelled out with much detail by Gonçalves *et al.* (2020) and we refer the reader to this paper.

Preserving uniform equivalence. This is a weaker form of preserving the dependencies compared to the previous one, and aligned with **(UP)**. The class F_{UP} belongs to this category. The main benefit over the previous ones is that **(UP)** is indeed satisfied and that individual atoms can be forgotten in any order without leading to a different result. It is also well-aligned with ASP when we only want to vary the instance data, that is, the facts.

Preserving equivalence. This is again weaker than the previous ones, and aligned with **(CP)**. Only F_{SM} fits this category among the remaining. While the preservation of relations over the remaining atoms is weaker, as noted in the previous section in particular w.r.t. F_{M}, it comes in exchange with a lower computational complexity.

Preserving consequences. These approaches preserve the semantic consequences over the remaining atoms, including the classes F_S and F_{HT}. While a comparison with the previous ones w.r.t. the strength of preserving semantic relations from the original program is not straightforward, we argue that, since the definition of F_{SM} imposes a further restriction on that of F_{HT}, preserving consequences is weaker. Among the two classes, as shown by several technical results in this paper, F_{HT} is preferable.

While this exposition indicates an order of preference among these classes, it is not strict, and still depends on the concrete intended usage. To aid in that regard, we next discuss with more detail the relation to uniform interpolation, before we consider applications.

6.4 Forgetting and interpolation

A strong notion of interpolation, called *uniform interpolation*, is well-known to be closely related to forgetting (Zhang and Zhou 2009; Lutz and Wolter 2011; Gabbay *et al.* 2011). Gabbay *et al.* (2011) introduced the notion for the case of logic programs, which we here adapt to make it more concise.

Definition 3
A class of logic programs $\mathcal{C}$ is said to have the uniform interpolation property with respect to a consequence relation $\vdash$, if for every program $P \in \mathcal{C}$ and set $V \subseteq \mathcal{A}$ of atoms, there exists a program $P^* \in \mathcal{C}$ with $\mathcal{A}(P^*) \subseteq \mathcal{A} \backslash V$, such that:

(*i*) $P \vdash P^*$;
(*ii*) $P^* \vdash R$, for every $R \in \mathcal{C}$ with $\mathcal{A}(R) \subseteq \mathcal{A} \backslash V$ such that $P \vdash R$.

In this case, P^* is said to be a *uniform interpolant* of P with respect to V.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

We say that a forgetting operator f over a class $\mathcal{C}$ of logic programs can be used to obtain uniform interpolants, if $\mathsf{f}(P, V)$ is a uniform interpolant of P with respect to V, for every $P \in \mathcal{C}$ and $V \subseteq \mathcal{A} \backslash V$. Gabbay *et al.* (2011) considered the non-monotonic skeptical consequence relation $\sim$ for logic programs, defined as $P \sim P'$ if $M \models P'$, for every $M \in \mathcal{AS}(P)$. Using a forgetting operator in F_{sem}, as defined by Eiter and Wang (2008), Gabbay *et al.* showed that, with respect to the consequence relation $\sim$, the class of disjunctive programs satisfies uniform interpolation for programs R composed of facts. Later, Wang *et al.* (2013) extended this result by showing that, with respect to the consequence relation $\sim$, the class of all logic programs satisfies uniform interpolation. Besides knowing that the class $\mathcal{C}_e$ of all logic programs satisfies uniform interpolation, it is also worth determining which approaches of forgetting can be used to obtain such uniform interpolants. It turns out that satisfying **(CP)** is a sufficient condition to obtain uniform interpolants with respect to $\sim$, and in fact the two inclusions of **(CP)**, that is, **(wC)** and **(sC)**, imply, respectively, conditions (i) and (ii) of the definition of uniform interpolation.

Theorem 7
If a class F of operators over a class $\mathcal{C}$ of logic programs satisfies property **(CP)**, then every operator of that class can be used to obtain uniform interpolants w.r.t. $\sim$.

Given the above result, we know that every forgetting operator satisfying **(CP)** can be used to obtain uniform interpolants, namely the classes $\mathsf{F_{SM}}$, $\mathsf{F_M}$, and $\mathsf{F_{UP}}$. Class F_{Sas} also satisfies **(CP)**, but this case is not worth considering, because the class is only defined for Horn programs. From the remaining classes that do not satisfy **(CP)**, only $\mathsf{F_R}$ satisfies the conditions of uniform interpolation.

Theorem 8
Every forgetting operator of the classes $\mathsf{F_{SM}}$, $\mathsf{F_M}$, $\mathsf{F_{UP}}$, and $\mathsf{F_R}$ can be used to obtain uniform interpolants with respect to $\sim$.

None of the other classes of forgetting operators can be used to obtain uniform interpolants with respect to $\sim$, and we now present, for each of these classes, a counterexample showing that one of the conditions for uniform interpolation is not satisfied. As shown by Gabbay *et al.* (2011), the class F_{sem} satisfies the conditions of uniform interpolation when R is a set of facts. Although condition (ii) is satisfied in general, we can see that this is not the case for condition (i). Consider the program $P = \{a \leftarrow p, a \leftarrow b, p \leftarrow not\, b, b \leftarrow not\, p\}$ over $\mathcal{A} = \{a, b, c, p\}$, where $\mathcal{AS}(P) = \{\{a, p\}, \{a, b\}\}$. Then, since the only restriction that any $\mathsf{f} \in \mathsf{F}_{sem}$ has to satisfy is $\mathcal{AS}(\mathsf{f}(P, p)) = \{\{a\}\}$, we can consider $\mathsf{f}(P, p) = \{a \leftarrow, c \leftarrow b\}$. In this case $P \not\sim \mathsf{f}(P, p)$, showing that condition (i) of uniform interpolation is not satisfied.

In the case of F_{weak}, consider the program $P = \{a \leftarrow not\, b, b \leftarrow not\, a\}$, for which $\mathcal{AS}(P) = \{\{a\}, \{b\}\}$. In this case, $\mathsf{f}_{weak}(P, a) = \{b \leftarrow\}$. But since $\{a\} \not\models \mathsf{f}_{weak}(P, a)$, we have that $P \not\sim \mathsf{f}_{weak}(P, a)$, and therefore condition (i) of uniform interpolation is not satisfied. For the classes F_{strong} and F_S, consider the program $P = \{b \leftarrow not\, a\}$, where $\mathcal{AS}(P) = \{\{b\}\}$. In this case, for $\mathsf{f} \in (\mathsf{F}_{strong} \cup \mathsf{F}_S)$, we have that $\mathsf{f}(P, a) \equiv_{\mathsf{HT}} \{\}$, and therefore $\mathcal{AS}(\mathsf{f}(P, a)) = \{\{\}\}$. If we take $R = \{b \leftarrow\}$, then $P \sim R$, but $\mathsf{f}(P, a) \not\sim R$, which means that condition (ii) of uniform interpolation is not satisfied.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

For F_W, consider the program $P = \{b \leftarrow, a \leftarrow b\}$, where $\mathcal{AS}(P) = \{\{a, b\}\}$. In this case, for $\mathsf{f} \in \mathsf{F}_W$, we have $\mathsf{f}(P, a) = \{b \leftarrow, \bot \leftarrow b\}$, and therefore $\mathcal{AS}(\mathsf{f}(P, a)) = \{\}$. But since $\{a, p\} \not\models \mathsf{f}(P, a)$, we have that $P \not\sim \mathsf{f}(P, a)$, and therefore condition (i) of uniform interpolation is not satisfied.

If we now consider F_{HT}, we can take a program P such that $\mathcal{HT}(P) = \{\langle ab, ab\rangle, \langle b, ab\rangle\}$. Therefore, $\mathcal{AS}(P) = \{\}$. In this case, for $\mathsf{f} \in \mathsf{F}_{\mathsf{HT}}$, we have $\mathcal{HT}(\mathsf{f}(P, a)) = \{\langle b, b\rangle\}$, and therefore $\mathcal{AS}(\mathsf{f}(P, a)) = \{\{b\}\}$. If we take $R = \{\bot \leftarrow not\, b\}$, then $P \sim R$, but $\mathsf{f}(P, a) \not\sim R$, which means that condition (ii) of uniform interpolation is not satisfied.

Finally, in the case of F_{SP}, consider the program $P = \{a \leftarrow p, b \leftarrow not\, p, p \leftarrow not\, not\, p\}$, for which $\mathcal{AS}(P) = \{\{a, p\}, \{b\}\}$. In this case, for $\mathsf{f} \in \mathsf{F}_{\mathsf{SP}}$, we have $\mathcal{AS}(\mathsf{f}(P, p)) = \{\{a\}, \{b\}, \{a, b\}\}$. If we take $R = \{\bot \leftarrow a, b\}$, then $P \sim R$, but $\mathsf{f}(P, a) \not\sim R$, which means that condition (ii) of uniform interpolation is not satisfied.

All these classes of operators that cannot be used to obtain uniform interpolants share the fact that property **(CP)** is not satisfied. Although this may indicate that **(CP)** could be a necessary condition for uniform interpolation with respect to $\sim$, a simple counterexample shows that this is not the case, and that, therefore, **(CP)** is in fact strictly stronger than uniform interpolation.

Example 8
Consider the program $P = \{a \leftarrow not\, b, b \leftarrow not\, a, c \leftarrow a, c \leftarrow b\}$, where $\mathcal{AS}(P) = \{\{a, c\}, \{b, c\}\}$, and a forgetting operator f such that $\mathsf{f}(P, a) = \{c \leftarrow\}$. Then, it is clear that f cannot satisfy **(CP)**. Nevertheless, it satisfies the two conditions of uniform interpolation, and it could therefore, be taken as a uniform interpolant for P.

This also reveals that uniform interpolation with respect to the skeptical consequence $\sim$ is not well-aligned with forgetting, as it does not impose a strong connection between the answer sets of the original program and those of the result of forgetting. This is in part due to the fact that the consequence $\sim$ only imposes the preservation of skeptical consequence.

Hence, instead of using the weak skeptical consequence, $\sim$, as originally considered by Gabbay *et al.* (2011), HT-consequence, $\models_{\mathsf{HT}}$, is arguably a more suitable consequence relation for uniform interpolation with respect to logic programs. In this case, conditions (i) and (ii) of uniform interpolation with respect to HT-consequence match two of the properties considered for forgetting, namely **(W)** and **(PP)**, respectively.

Theorem 9
A class F of forgetting operators can be used to obtain uniform interpolants w.r.t. $\models_{\mathsf{HT}}$ iff F satisfies both **(W)** and **(PP)**.

The above characterization, together with the results of Theorem 5, allow us to conclude exactly what classes of operators can be used to obtain uniform interpolants with respect to $\models_{\mathsf{HT}}$.

Theorem 10
Every forgetting operator of the classes F_{HT} and F_S can be used to obtain uniform interpolants with respect to $\models_{\mathsf{HT}}$.

The classes F_{HT} and F_S are therefore closely aligned with uniform interpolation, as both satisfy **(W)** and **(PP)**. In fact, as stated in Theorem (10) of (Wang *et al.* 2014),

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

the class F_{HT} is precisely characterized by **(W)** and **(PP)**, which also means that it precisely characterizes uniform interpolation with respect to $\models_{\mathsf{HT}}$. As pointed out by Wang *et al.*, this result also shows that the class of all logic programs has the uniform interpolation property with respect to HT-consequence. Note that, when considering the class of all logic programs, we should consider F_{HT} to obtain uniform interpolants, since F_S is only defined for disjunctive programs.

The above results and discussion also show that, contrarily to forgetting in the realm of monotonic logics, for example classical logic or description logics, where uniform interpolation can serve as a guideline for what forgetting should be, in our view, in non-monotonic frameworks such as ASP this cannot be the case: the relation $\sim$ seems too weak to capture forgetting and $\models_{\mathsf{HT}}$ imposes **(W)**, a monotonic property which is incompatible with the preservation of answer sets when forgetting.

6.5 Applications

In this section, we will discuss some of the applications of forgetting mentioned in the literature and provide indications as to what classes of operators are, from our point of view, most suitable for each of them.

Conflict resolution. Forgetting has been considered as a means of conflict resolution for inconsistencies by weakening pieces of information to restore consistency in propositional logic, also termed recovery of preferences (Lang and Marquis 2002; 2010). This has been adapted to a multi-agent setting using answer set programming (Zhang and Foo 2006; Eiter and Wang 2008; Delgrande and Wang 2015), where a set of programs represent the knowledge or preferences of individual agents. If these programs together are inconsistent, then a compromise is a sequence of sets of atoms – one set per program indicating that these are to be forgotten from the corresponding program – such that the resulting programs together are consistent, admitting an agreement, that is, one or several answer sets. Zhang and Foo (2006) defined compromises (termed preferred solutions) that are minimal w.r.t. the amount of atoms forgotten, and Eiter and Wang (2008) argued that, in such a setting, answer sets indeed result in minimal agreements (unlike in propositional logic where non-minimal agreements may be obtained). Finally, Delgrande and Wang (2015) emphasized that such forgetting results should clearly keep as much of the program as is while forgetting, in particular not affecting rules over the remaining atoms. Thus, in our view, an approach that preserves as much as possible the semantics over the remaining atoms is recommended according to our considerations in Section 6.3, in particular, one that preserves these semantic relations when other rules are added from the programs of the other agents, that is, when using one of the classes F_{SP}, F_{R}, and F_{M}.

Query answering. As argued by Delgrande and Wang (2015), in query answering, if one can determine what is relevant to a query, then the irrelevant part of the knowledge base/logic program can be forgotten, allowing for more efficient querying. Similar ideas have been expressed in the context of forgetting in Description Logics for using only a small fraction of the ontology in an application (Konev *et al.* 2009). Conceptually common to these ideas is that one wants to use a simplified version of the knowledge base in question "as is", that is, without having to incorporate additional information

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

during the reasoning process. This is indeed closely related to uniform interpolation. Therefore, in our view, the most suitable solutions can be found among the approaches that can be used to obtain uniform interpolants with respect to $\models_{\mathsf{HT}}$, that is, F_{HT} and F_S, among which, according to our considerations in Section 6.4, we prefer to recommend F_{HT}, since it is more general and satisfies **(SI)** which allows us to simplify the process of computing forgetting results.

Modular reuse. While similar in spirit to the ideas presented with respect to query answering, a crucial difference for modular reuse is that a simplified program is created via forgetting that, unlike the former, is still subject to interaction with rules from other programs. The answer to the question what approach is preferred then largely depends on how this interaction is realized. If arbitrary rules can be added or interact with the program, as, for example, outlined for the case of conflict resolution where a joint program is composed, then the same recommendations apply as outlined for conflict resolution. If however the interaction follows a truly modular approach with well-defined input-output interfaces between modules (Janhunen *et al.* 2009), where interaction is limited to atoms, then a particular preference can be given to F_{UP} whose characteristics are well-suited for this setting.

Hiding/Privacy. The principal idea of hiding is that certain parts/terms of a program are not intended to be public. This usage is also aligned with the notion of privacy and the data protection regulations such as the GDPR (European Parliament 2016). Here, the recommendation on what class of forgetting operators to use depends on the intended usage. If we aim to merely create a publicly viewable version of a program, then any approach that can be used to obtain uniform interpolants is suitable. Thus, as spelled out for Query Answering, F_{HT} would be most suitable. However, since this class does not come with a known syntactic operator, among the two, F_S is preferable in this context. Alternatively, if the resulting program is meant to be a public version that is being used, possibly together with other programs, then the recommendations from Modular Reuse are more suitable.

Embeddings in forgetting. Forgetting has been considered in the literature as a way to capture other problems with the aim to facilitate their study and comparison, as well as being able to re-use existing algorithms. For example, Zhang and Foo (2006) have studied how different approaches to updating logic programs can be embedded into their framework of solving conflicts which is built on forgetting, allowing then to analyze these different frameworks. Similarly, Eiter and Wang (2008) showed how inheritance logic programs (Buccafurri *et al.* 2002) can be captured using forgetting, which in turn could be used to capture updates in logic programs. In these cases, picking a suitable approach depends to a considerable extent on the target, so a more detailed inspection on the approaches and what properties they satisfy is recommended (cf. Section 6.1).

Summing up, in our view, those classes of operators that are closely related with **(SP)** and **(UP)**, that is, F_{SP}, F_{R}, F_{M}, and F_{UP}, are in general the most useful, except for those applications that are closer related to uniform interpolation, in which case F_{HT} would be more suitable.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

7 Conclusions

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

This paper aimed at addressing this problem, presenting a systematic study of forgetting in ASP, including a thorough investigation of both properties and existing classes of forgetting operators, going well beyond a survey of the state-of-the-art as many novel results were included, thus achieving a truly comprehensive picture of the landscape.

In more detail, after providing a uniform definition of forgetting, we presented a detailed account on properties of forgetting found in the literature, that allowed us to establish existing relations between these properties independently of specific classes of forgetting operators.

We then recalled the classes of forgetting operators proposed in the literature in a systematic and uniform way, which allowed us to obtain several relations between them, including that some classes of forgetting operators coincide for restricted classes of programs, and that two of them even coincide in general. This was strengthened by complementary considerations on existing concrete forgetting operators and results on computational complexity so far spread out over the literature.

We also provided a complete study showing which properties are satisfied by which class of operators. This allowed a thorough discussion and comparison of the existing properties, their impact on comparisons between classes of forgetting operators, as well as considerations on unsuitable operators of forgetting, all contributing to guidelines for the choice of a concrete forgetting operator.

Regarding such choice, it clearly depends on the application in mind, and we discussed several applications and indicated options of forgetting operators to consider. As a brief summary, the following criteria are important in our view:

- Preservation of semantic relations: Arguably, **(SP)** best captures the notion of forgetting, but as this is in general impossible for classes of programs beyond Horn, approximations are in order, which requires a more detailed look into the satisfied properties.
- Concrete Operators: Preferably operators should provide results that are as similar as possible to the given program with the aim to preserve the declarative nature of ASP.
- Complexity: Even though forgetting is known to be computationally expensive, a lower computational complexity is preferable.
- Program Class: Depending on the application, certain operators may not be suitable.

As our study shows, optimizing all criteria simultaneously, is not suitable. For example, the classes that provide closer approximations of **(SP)** are computationally more expensive, and simple concrete operators may only be defined for a restricted class of programs. Still, we believe that our results together with the considerations on suitable options for certain use cases provide substantial material to help making a choice that balances these criteria.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

One important open issue is establishing further concrete operators that are efficient (within the theoretical limitations imposed) and preferably provide forgetting results that are as similar as possible to the original program to preserve declarative nature of the programs, in the line of work by Berthold *et al.* (2019b). Other avenues for future research include investigating the potential connections between forgetting and the work on abstraction in the context of Answer Set Programs (Saribatur and Eiter 2018; Saribatur *et al.* 2019; Saribatur and Eiter 2021; Saribatur *et al.* 2021). Alternatively, we may further pursue forgetting over extensions of the syntax of logic programs, in the line of the work by Aguado *et al.* (2019), where an extension of the syntax of programs by a new connective, called fork, allows Strong Persistence to always hold (cf. (Berthold *et al.* 2019a)). Also interesting is to study forgetting with semantics other than ASP, such as the FLP-semantics (Truszczynski 2010) following the work by Wang *et al.* (2014), or the well-founded semantics (Van Gelder *et al.* 1991) as considered by Alferes *et al.* (2013); Knorr and Alferes (2014). Finally, one ambitious open problem is the generalization to non-ground programs with variables to be able to forget from ASP programs expressed in first-order terms allowing then to forget predicates or constants.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work was partially supported by FCT project FORGET (PTDC/CCI-INF/32219/2017) and by FCT project NOVA LINCS (UIDB/04516/2020).

Supplementary material

To view supplementary material for this article, please visit http://dx.doi.org/10.1017/S1471068421000570.

References

Aguado, F., Cabalar, P., Fandinno, J., Pearce, D., Pérez, G. and Vidal, C. 2019. Forgetting auxiliary atoms in forks. *Artificial Intelligence 275*, 575–601.

Alferes, J. J., Knorr, M. and Wang, K. 2013. Forgetting under the Well-Founded Semantics. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Proceedings*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 36–41.

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

Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P. and Zangari, J. 2017. The ASP system DLV2. In *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Proceedings*, M. Balduccini and T. Janhunen, Eds. Lecture Notes in Computer Science, vol. 10377. Springer, 215–221.

Baral, C. and Zhang, Y. 2005. Knowledge updates: Semantics and complexity issues. *Artificial Intelligence 164,* 1–2, 209–243.

Beierle, C. and Timm, I. J. 2019. Intentional forgetting: An emerging field in AI and beyond. *Künstliche Intelligenz 33,* 1, 5–8.

Berthold, M., Gonçalves, R., Knorr, M. and Leite, J. 2019a. Forgetting in Answer Set Programming with anonymous cycles. In *Progress in Artificial Intelligence, 19th EPIA Conference on Artificial Intelligence, EPIA 2019, Proceedings, Part II*, P. M. Oliveira, P. Novais and L. P. Reis, Eds. Lecture Notes in Computer Science, vol. 11805. Springer, 552–565.

Berthold, M., Gonçalves, R., Knorr, M. and Leite, J. 2019b. A syntactic operator for forgetting that satisfies strong persistence. *Theory and Practice of Logic Programming 19,* 5–6, 1038–1055.

Bledsoe, W. W. and Hines, L. M. 1980. Variable elimination and chaining in a resolution-based prover for inequalities. In *5th Conference on Automated Deduction, Proceedings*, W. Bibel and R. A. Kowalski, Eds. Lecture Notes in Computer Science, vol. 87. Springer, 70–87.

Boole, G. 1854. *An Investigation of The Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Macmillian, London.

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

Buccafurri, F., Faber, W. and Leone, N. 2002. Disjuctive logic programs with inheritance. *Theory and Practice of Logic Programming 2,* 3, 293–321.

Cabalar, P. and Ferraris, P. 2007. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming 7,* 6, 745–759.

Cabalar, P., Pearce, D. and Valverde, A. 2007. Minimal logic programs. In *Logic Programming, 23rd International Conference, ICLP 2007, Proceedings*, V. Dahl and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 4670. Springer, 104–118.

Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33,* 3, 374–425.

Davis, R. L. and Zhong, Y. 2017. The biology of forgetting - A perspective. *Neuron 95,* 3, 490–503.

Delgrande, J. P. 2017. A knowledge level account of forgetting. *Journal of Artificial Intelligence Research 60*, 1165–1213.

Delgrande, J. P., Schaub, T., Tompits, H. and Woltran, S. 2013. A model-theoretic approach to belief change in Answer Set Programming. *ACM Transactions on Computational Logic 14,* 2, 14:1–14:46.

Delgrande, J. P. and Wang, K. 2015. A syntax-independent approach to forgetting in disjuctive logic programs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, B. Bonet and S. Koenig, Eds. AAAI Press, 1482–1488.

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

Eiter, T. and Fink, M. 2003. Uniform equivalence of logic programs under the Stable Model Semantics. In *Logic Programming, 19th International Conference, ICLP 2003, Proceedings*, C. Palamidessi, Ed. Lecture Notes in Computer Science, vol. 2916. Springer, 224–238.

Eiter, T., Fink, M., Sabbatini, G. and Tompits, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming 2,* 6, 711–767.

Eiter, T., Fink, M. and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in Answer Set Programming. *ACM Transactions on Computational Logic 8,* 3, 17.

Eiter, T. and Kern-Isberner, G. 2019. A brief survey on forgetting from a knowledge representation and reasoning perspective. *Künstliche Intelligenz 33,* 1, 9–33.

Eiter, T. and Wang, K. 2008. Semantic forgetting in Answer Set Programming. *Artificial Intelligence 172,* 14, 1644–1672.

Erdem, E. and Ferraris, P. 2007. Forgetting actions in domain descriptions. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 409–414.

European Parliament. 2016. General data protection regulation. *Official Journal of the European Union L119/59*.

Gabbay, D. M., Pearce, D. and Valverde, A. 2011. Interpolable formulas in Equilibrium Logic and Answer Set Programming. *Journal of Artificial Intelligence Research 42*, 917–943.

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

Gebser, M., Kaminski, R., Kaufmann, B., Lühne, P., Obermeier, P., Ostrowski, M., Romero, J., Schaub, T., Schellhorn, S. and Wanko, P. 2018. The Potsdam answer set solving collection 5.0. *Künstliche Intelligenz 32,* 2–3, 181–182.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T. and Schneider, M. T. 2011. Potassco: The Potsdam answer set solving collection. *AI Communications 24,* 2, 107–124.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium, (2 Volumes)*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.

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

Ghilardi, S., Lutz, C. and Wolter, F. 2006. Did I damage my ontology? A case for conservative extensions in Description Logics. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, P. Doherty, J. Mylopoulos and C. A. Welty, Eds. AAAI Press, 187–197.

Gonçalves, R., Janhunen, T., Knorr, M. and Leite, J. 2021. On syntactic forgetting under uniform equivalence. In *Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Proceedings*, W. Faber, G. Friedrich, M. Gebser and M. Morak, Eds. Lecture Notes in Computer Science, vol. 12678. Springer, 297–312.

Gonçalves, R., Janhunen, T., Knorr, M., Leite, J. and Woltran, S. 2019. Forgetting in Modular Answer Set Programming. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*. AAAI Press, 2843–2850.

Gonçalves, R., Knorr, M., ao Leite, J. and Woltran, S. 2020. On the limits of forgetting in Answer Set Programming. *Artificial Intelligence 286*, 103–307.

Gonçalves, R., Knorr, M. and Leite, J. 2016a. Forgetting in ASP: the forgotten properties. In *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Proceedings*, L. Michael and A. C. Kakas, Eds. Lecture Notes in Computer Science, vol. 10021. Springer, 543–550.

Gonçalves, R., Knorr, M. and Leite, J. 2016b. The ultimate guide to forgetting in Answer Set Programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016*, C. Baral, J. P. Delgrande and F. Wolter, Eds. AAAI Press, 135–144.

Gonçalves, R., Knorr, M. and Leite, J. 2016c. You can't always forget what you want: On the limits of forgetting in Answer Set Programming. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. van Harmelen, Eds. Frontiers in Artificial Intelligence and Applications, vol. 285. IOS Press, 957–965.

Gonçalves, R., Knorr, M. and Leite, J. 2017. Iterative variable elimination in ASP. In *Progress in Artificial Intelligence - 18th EPIA Conference on Artificial Intelligence, EPIA 2017, Proceedings*, E. C. Oliveira, J. Gama, Z. A. Vale and H. L. Cardoso, Eds. Lecture Notes in Computer Science, vol. 10423. Springer, 643–656.

Gonçalves, R., Knorr, M., Leite, J. and Woltran, S. 2017. When you must forget: Beyond strong persistence when forgetting in Answer Set Programming. *Theory and Practice of Logic Programming 17,* 5–6, 837–854.

Heyting, A. 1930. Die formalen Regeln der intuitionistischen Logik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften. Physikalisch-mathematische Klasse*, 42–56.

Janhunen, T., Oikarinen, E., Tompits, H. and Woltran, S. 2009. Modularity aspects of disjunctive stable models. *Journal of Artificial Intelligence Research 35*, 813–857.

Ji, J., You, J. and Wang, Y. 2015. On forgetting postulates in Answer Set Programming. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, Q. Yang and M. J. Wooldridge, Eds. AAAI Press, 3076–3083.

Kluge, A. and Gronau, N. 2018. Intentional forgetting in organizations: The importance of eliminating retrieval cues for implementing new routines. *Frontiers in Psychology 9,* 51, 1–17.

Knorr, M. and Alferes, J. J. 2014. Preserving strong equivalence while forgetting. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014. Proceedings*, E. Fermé and J. Leite, Eds. Lecture Notes in Computer Science, vol. 8761. Springer, 412–425.

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

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

Konev, B., Walther, D. and Wolter, F. 2009. Forgetting and uniform interpolation in large-scale description logic terminologies. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, C. Boutilier, Ed. IJCAI/AAAI.

Kontchakov, R., Wolter, F. and Zakharyaschev, M. 2010. Logic-based ontology comparison and module extraction, with an application to DL-lite. *Artificial Intelligence 174,* 15, 1093–1141.

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

Lang, J. and Marquis, P. 2002. Resolving inconsistencies by variable forgetting. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, D. Fensel, F. Giunchiglia, D. L. McGuinness and M. Williams, Eds. Morgan Kaufmann, 239–250.

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

Larrosa, J. 2000. Boosting search with variable elimination. In *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Proceedings*, R. Dechter, Ed. Lecture Notes in Computer Science, vol. 1894. Springer, 291–305.

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

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

Lifschitz, V., Pearce, D. and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2,* 4, 526–541.

Lifschitz, V., Tang, L. R. and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 369–389.

Lin, F. and Reiter, R. 1994. Forget it! In *AAAI Fall Symposium on Relevance.* AAAI Press, 154–159.

Lin, F. and Reiter, R. 1997. How to progress a database. *Artificial Intelligence 92,* 1–2, 131–167.

Liu, Y. and Wen, X. 2011. On the progression of knowledge in the Situation Calculus. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, T. Walsh, Ed. IJCAI/AAAI, 976–982.

Lutz, C. and Wolter, F. 2011. Foundations for uniform interpolation and forgetting in expressive description logics. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, T. Walsh, Ed. IJCAI/AAAI, 989–995.

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

Moinard, Y. 2007. Forgetting literals with varying propositional symbols. *Journal of Logic and Computation 17,* 5, 955–982.

Pearce, D. 1999. Stable inference as intuitionistic validity. *Journal of Logic Programming 38,* 1, 79–91.

Rajaratnam, D., Levesque, H. J., Pagnucco, M. and Thielscher, M. 2014. Forgetting in action. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014*, C. Baral, G. D. Giacomo and T. Eiter, Eds. AAAI Press.

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

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

Saribatur, Z. and Eiter, T. 2021. Omission-based abstraction for answer set programs. *Theory and Practice of Logic Programming 21,* 2, 145–195.

Saribatur, Z. G. and Eiter, T. 2018. Omission-based abstraction for answer set programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018*, M. Thielscher, F. Toni and F. Wolter, Eds. AAAI Press, 42–51.

Saribatur, Z. G., Eiter, T. and Schüller, P. 2021. Abstraction for non-ground answer set programs. *Artificial Intelligence 300*, 103563.

Saribatur, Z. G., Schüller, P. and Eiter, T. 2019. Abstraction for non-ground answer set programs. In *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Proceedings*, F. Calimeri, N. Leone and M. Manna, Eds. Lecture Notes in Computer Science, vol. 11468. Springer, 576–592.

Shrestha, P. 2017. Meaning and causes of forgetting. Psychstudy, November 17, 2017. https://www.psychestudy.com/cognitive/memory/meaning-causes-forgetting.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the Stable Model Semantics. *Artificial Intelligence 138,* 1–2, 181–234.

Slota, M. and Leite, J. 2012a. Robust equivalence models for semantic updates of answer-set programs. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012*, G. Brewka, T. Eiter and S. A. McIlraith, Eds. AAAI Press.

Slota, M. and Leite, J. 2012b. A unifying perspective on knowledge updates. In *Logics in Artificial Intelligence - 13th European Conference, JELIA 2012. Proceedings*, L. F. del Cerro, A. Herzig and J. Mengin, Eds. Lecture Notes in Computer Science, vol. 7519. Springer, 372–384.

Slota, M. and Leite, J. 2014. The rise and fall of semantic rule updates based on SE-models. *Theory and Practice of Logic Programming 14,* 6, 869–907.

Truszczynski, M. 2010. Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artificial Intelligence 174,* 16-17, 1285–1306.

Turner, H. 2003. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming 3,* 4–5, 609–622.

https://doi.org/10.1017/S1471068421000570 Published online by Cambridge University Press

Van Gelder, A., Ross, K. A. and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38,* 3, 620–650.

Visser, A. 1996. Uniform interpolation and layered bisimulation. In *Gödel 1996*. Lecture Notes in Logic, vol. 6. Springer Verlag, 139–164.

Wang, K., Sattar, A. and Su, K. 2005. A theory of forgetting in logic programming. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press/The MIT Press, 682–688.

Wang, Y., Wang, K. and Zhang, M. 2013. Forgetting for answer set programs revisited. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, F. Rossi, Ed. IJCAI/AAAI, 1162–1168.

Wang, Y., Zhang, Y., Zhou, Y. and Zhang, M. 2012. Forgetting in logic programs under strong equivalence. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012*, G. Brewka, T. Eiter and S. A. McIlraith, Eds. AAAI Press.

Wang, Y., Zhang, Y., Zhou, Y. and Zhang, M. 2014. Knowledge forgetting in Answer Set Programming. *Journal of Artificial Intelligence Research 50*, 31–70.

Wang, Z., Wang, K., Topor, R. W. and Pan, J. Z. 2010. Forgetting for knowledge bases in DL-lite. *Annals of Mathematics and Artificial Intelligence 58,* 1–2, 117–151.

Weber, A. 1986. Updating propositional formulas. In *Expert Database Systems, Proceedings From the First International Conference*, L. Kerschberg, Ed. Benjamin/Cummings, 487–500.

Wong, K.-S. 2008. Sound and complete inference rules for SE-consequence. *Journal of Artificial Intelligence Research 31*, 205–216.

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

Zhang, Y. and Foo, N. Y. 2005. A unified framework for representing logic program updates. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press/The MIT Press, 707–713.

Zhang, Y. and Foo, N. Y. 2006. Solving logic program conflict through strong and weak forgettings. *Artificial Intelligence 170,* 8–9, 739–778.

Zhang, Y., Foo, N. Y. and Wang, K. 2005. Solving logic program conflict through strong and weak forgettings. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, L. P. Kaelbling and A. Saffiotti, Eds. Professional Book Center, 627–634.

Zhang, Y. and Zhou, Y. 2009. Knowledge forgetting: Properties and applications. *Artificial Intelligence 173,* 16–17, 1525–1537.