Forgetting in ASP: The Forgotten Properties

Ricardo Gonçalves, Matthias Knorr, and João Leite

NOVA LINCS & Departamento de Informática, Universidade Nova de Lisboa

Abstract. Many approaches for forgetting in Answer Set Programming (ASP) have been proposed in recent years, in the form of specific operators, or classes of operators, following different principles and obeying different properties. A recently published comprehensive overview of existing operators and properties provided a uniform picture of the landscape, including many novel (even surprising) results on relations between properties and operators. Yet, this overview largely missed an additional set properties for forgetting, proposed by Wong, and this paper aims to close this gap. It turns out that, while some of these properties are closely related to the properties previously studied, four of them are distinct providing novel results and insights, further strengthening established relations between existing operators.

1 Introduction

Forgetting – or variable elimination – is an operation that allows for the removal, from a knowledge base, of *middle* variables no longer deemed relevant, whose importance is witnessed by its application to cognitive robotics [1,2], resolving conflicts [3,4,5], and ontology abstraction and comparison [6,7]. With its early roots in Boolean Algebra, it has been extensively studied within classical logic [3,8].

Only more recently, the operation of forgetting began to receive attention in the context of non-monotonic logic programming, notably of Answer Set Programming (ASP). It turns out that the rule-based nature and non-monotonic semantics of ASP create very unique challenges to the development of forgetting operators – just as with other belief change operators such as those for revision and update, c.f. [9,10,11,12] – making it a special endeavour with unique characteristics distinct from those for classical logic.

Over the years, many have proposed different approaches to forgetting in ASP, through the characterization of the result of forgetting a set of atoms from a given program up to some equivalence class, and/or through the definition of concrete operators that produce a program given an input program and atoms to be forgotten [4,5,13,14,15,16,17]. These approaches were typically proposed to obey some specific set of properties deemed adequate by their authors, some adapted from the literature on *classical* forgetting [18,16], others introduced for the case of ASP [5,13,14,15,17].

The result is a *complex* landscape filled with operators and properties, that is difficult to navigate. This problem was tackled in [19] by presenting a systematic study of *forgetting* in ASP, thoroughly investigating the different approaches found in the literature, their properties and relationships, giving rise to a comprehensive guide aimed at helping users navigate this topic's complex landscape and ultimately assist them in choosing suitable operators for each application.

However, [19] ignores to a large extent the postulates on forgetting in ASP introduced by Wong in [13].[1] In this paper, we close this gap by thoroughly investigating them, their relationships with other properties and existing operators, concluding that, while some of them are straightforwardly implied by one of the previously studied properties, hence ultimately weaker than these and thus of less importance, others turn out to be distinct and provide additional novel results further strengthening the relations between properties and classes of operators as established previously.

2 Preliminaries

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

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

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

Regarding the semantics of propositional formulas, we consider the monotonic logic here-and-there (HT) and equilibrium models [20]. An *HT-interpretation* is a pair $\langle H, T\rangle$ s.t. $H \subseteq T \subseteq \mathcal{A}$. The satisfiability relation in HT, denoted $\models_{\mathsf{HT}}$, is recursively defined as follows for $p \in \mathcal{A}$ and formulas φ and ψ:

- $\langle H, T\rangle \models_{\mathsf{HT}} p$ if $p \in H$; $\quad\quad$ $\langle H, T\rangle \not\models_{\mathsf{HT}} \bot$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \wedge \psi$ if $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$ and $\langle H, T\rangle \models_{\mathsf{HT}} \psi$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \vee \psi$ if $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$ or $\langle H, T\rangle \models_{\mathsf{HT}} \psi$;
- $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \supset \psi$ if (i) $T \models \varphi \supset \psi$,[2] and (ii) $\langle H, T\rangle \models_{\mathsf{HT}} \varphi \Rightarrow \langle H, T\rangle \models_{\mathsf{HT}} \psi$.

An HT-interpretation is an *HT-model* of a formula φ if $\langle H, T\rangle \models_{\mathsf{HT}} \varphi$. We denote by $\mathcal{HT}(\varphi)$ the set of *all HT-models of* φ. In particular, $\langle T, T\rangle \in \mathcal{HT}(\varphi)$ is an *equilibrium model* of φ if there is no $T' \subset T$ s.t. $\langle T', T\rangle \in \mathcal{HT}(\varphi)$. Given two formulas φ and ψ, if $\mathcal{HT}(\varphi) \subseteq \mathcal{HT}(\psi)$, then φ *entails* ψ in HT, written $\varphi \models_{\mathsf{HT}} \psi$. Also, φ and ψ are *HT-equivalent*, written $\varphi \equiv_{\mathsf{HT}} \psi$, if $\mathcal{HT}(\varphi) = \mathcal{HT}(\psi)$.

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

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

where all elements in $A = \{a_1, \ldots, a_k\}$, $B = \{b_1, \ldots, b_l\}$, $C = \{c_1, \ldots, c_m\}$, $D = \{d_1, \ldots, d_n\}$ are atoms.[3]

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

As shown by Cabalar and Ferraris [22], any set of (propositional) formulas is HT-equivalent to an (extended) logic program which is why we can focus solely on these.

[1] We use the term *postulate* to follow [13] and easily distinguish them from the *properties* discussed in [19]. However, their role is the same as the role of other properties.

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

[3] Extended logic programs [21] are actually more expressive, but this form is sufficient here.

The class of logic programs, $\mathcal{C}_e$, i.e., the set of all (extended) logic programs, includes a number of special kinds of rules r: notably if $n = 0$, then we call r *disjunctive*. Then, the class of *disjunctive programs*, $\mathcal{C}_d$ is defined as a finite set of disjunctive rules.

We recall the following from [19]. Given a class of logic programs $\mathcal{C}$ over $\mathcal{A}$, a *forgetting operator* is a partial function $\mathsf{f} : \mathcal{C} \times 2^{\mathcal{A}} \to \mathcal{C}$ s.t. $\mathsf{f}(P, V)$ is a program over $\mathcal{A}(P) \setminus V$, for each $P \in \mathcal{C}$ and $V \in 2^{\mathcal{A}}$. We call $\mathsf{f}(P, V)$ the *result of forgetting about V from P*. Furthermore, f is called *closed* for $\mathcal{C}' \subseteq \mathcal{C}$ if, for every $P \in \mathcal{C}'$ and $V \in 2^{\mathcal{A}}$, we have $\mathsf{f}(P, V) \in \mathcal{C}'$. A *class* F *of forgetting operators* is a set of forgetting operators.

Previous work on forgetting in ASP has introduced a variety of desirable properties and operators satisfy differing subsets of these. For lack of space, we refer to [19].

3 Wong's Properties of Forgetting

With all concepts and notation in place, we can now turn our attention to the postulates introduced by Wong [13]. These postulates were defined in a somewhat different way when compared to the properties presented in [19]. Namely, they only considered forgetting a single atom, were defined for disjunctive programs (the maximal class of programs considered in [13]), and used a generic formulation which allowed different notions of equivalence. Here, we only consider HT-equivalence, i.e., strong equivalence, as, in the literature, this is clearly the more relevant of the two notions considered in [13] (the other one being the non-standard T-equivalence) and in line with previously presented material here and in [19].

We start by recalling these postulates[4] adjusting them to our notation and extending them to the most general class of extended logic programs considered here.

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

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

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

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

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

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

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

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

These postulates represent the following: Forgetting about atom a from HT-equivalent programs preserves HT-equivalence **(F0)**; if a program is an HT-consequence of another program, then forgetting about atom a from both programs preserves this HT-

[4] As mentioned before, we use the term *postulate* to follow [13] and ease readability. Technically, they are treated as every other *property*.

consequence **(F1)**; when forgetting about an atom a, it does not matter whether we add a set of rules over the remaining language before or after forgetting **(F2)**; any consequence of the original program not mentioning atom a is also a consequence of the result of forgetting about a **(F2-)**; the result of forgetting about an atom from a program only contains atoms occurring in the original program **(F3)**; any rule which is a consequence of the result of forgetting about an atom from program P is a consequence of the result of forgetting about that atom from a single rule among the HT-consequences of P **(F4)**; a rule obtained by extending with $not\ a$ the body of a rule which is an HT-consequence of the result of forgetting about an atom a from program P is an HT-consequence of P **(F5)**; and the order is not relevant when sequentially forgetting two atoms **(F6)**.

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

The following proposition relates these postulates and the properties in [19].

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

1. **(F1)** *implies* **(F0)***; [13]*
2. **(F2)** *and* **(F1)** *imply* **(F2-)***; [13]*
3. **(SE)** *implies* **(F0)***;*
4. **(W)** *and* **(PP)** *together imply* **(F1)***;*
5. **(SI)** *implies* **(F2)***;*
6. **(PP)** *implies* **(F2-)***;*
7. **(W)** *implies* **(F5)**.

Postulates **(F0)**, **(F2)**, **(F2-)**, and **(F5)** are implied by existing properties presented in [19], while **(F1)** is implied by a pair of these. This may impact on the question we investigate next, namely which classes of operators from the literature satisfy which of the new postulates. For that purpose, we verified for all classes of operators presented in [19] which of these postulates they satisfy. The results are summarized in the main theorem of our paper, illustrated in one easy-to-read table.

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

It turns out that for three of the four postulates directly implied by existing properties, **(F0)**, **(F2)**, and **(F2-)**, the classes of operators that satisfy them coincide with their existing generalization (see Prop. 1). Also, postulate **(F3)** is always satisfied, which is not surprising given the definition of forgetting operators. Three of the remaining four properties, **(F1)**, **(F4)**, and **(F5)**, are in fact distinct (even though **(F5)** is implied by an existing property), and no other already existing property is satisfied by precisely the same set of classes of forgetting operators in each of these cases (see [19]). Notably, unlike the weaker property **(F0)** and the related **(SE)**, F_{SM} and F_{Sas} do not satisfy **(F1)**, most likely because the premise in the condition for satisfying **(F1)** is weaker than that of **(F0)**. Finally, postulate **(F6)** is not always satisfied, but it seems that this is solely tied to the incompatibility with the crucial property **(SP)**, further discussed in [23].

4 Conclusions

We have studied eight postulates of forgetting in ASP introduced in [13], to fill a gap in a recent comprehensive guide on properties and classes of operators for forgetting

	(F0)	(F1)	(F2)	(F2-)	(F3)	(F4)	(F5)	(F6)
F_{strong}	×	×	✓	×	✓	✓	✓	✓
F_{weak}	×	×	✓	✓	✓	✓	✓	✓
F_{sem}	×	×	×	×	✓	×	×	✓
F_{S}	✓	✓	×	✓	✓	✓	✓	✓
F_{W}	✓	✓	✓	✓	✓	✓	✓	✓
F_{HT}	✓	✓	✓	✓	✓	✓	✓	✓
F_{SM}	✓	×	×	✓	✓	×	×	✓
F_{Sas}	✓	×	✓	✓	✓	×	×	×
F_{SE}	✓	✓	×	✓	✓	✓	✓	✓

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

in ASP, and relations between these [19]. It turns out that four of them can safely be ignored because they either basically coincide with already existing properties or are trivially satisfied by any forgetting operator. The others are in fact distinct, and no other already existing property is satisfied by precisely the same set of classes of forgetting operators in each of these cases.

Left open is the investigation of these postulates for semantics other than ASP, such as [16] based on the FLP-semantics [24], or [25,15] based on the well-founded semantics, as well as forgetting in the context of hybrid theories [26,27,28] and reactive/evolving multi-context systems [29,30], as well as the development of concrete syntactical forgetting operators that can be integrated in reasoning tools such as [31,32,33].

Acknowledgments All authors were partially supported by FCT under strategic project NOVA LINCS (UID/CEC/04516/2013). R. Gonçalves was partially supported by FCT grant SFRH/BPD/100906/2014 and M. Knorr by FCT grant SFRH/BPD/86970/2012.

References

1. Lin, F., Reiter, R.: How to progress a database. Artif. Intell. **92**(1-2) (1997) 131–167
2. Rajaratnam, D., Levesque, H.J., Pagnucco, M., Thielscher, M.: Forgetting in action. In Baral, C., Giacomo, G.D., Eiter, T., eds.: Procs. of KR, AAAI Press (2014)
3. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: Formula-variable independence and forgetting. J. Artif. Intell. Res. (JAIR) **18** (2003) 391–443
4. Zhang, Y., Foo, N.Y.: Solving logic program conflict through strong and weak forgettings. Artif. Intell. **170**(8-9) (2006) 739–778
5. Eiter, T., Wang, K.: Semantic forgetting in answer set programming. Artif. Intell. **172**(14) (2008) 1644–1672
6. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Logic-based ontology comparison and module extraction, with an application to dl-lite. Artif. Intell. **174**(15) (2010) 1093–1141
7. Konev, B., Lutz, C., Walther, D., Wolter, F.: Model-theoretic inseparability and modularity of description logic ontologies. Artif. Intell. **203** (2013) 66–103
8. Larrosa, J., Morancho, E., Niso, D.: On the practical use of variable elimination in constraint optimization problems: 'still-life' as a case study. J. Artif. Intell. Res. (JAIR) **23** (2005) 421–440
9. Alferes, J., Leite, J., Pereira, L.M., Przymusinska, H., Przymusinski, T.: Dynamic updates of non-monotonic knowledge bases. The Journal of Logic Programming **45**(1-3) (2000) 43–70

10. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming (TPLP) **2**(6) (2002) 721–777
11. Leite, J.A.: Evolving Knowledge Bases. Volume 81 of Frontiers of Artificial Intelligence and Applications, xviii + 307 p. Hardcover. IOS Press (2003)
12. Slota, M., Leite, J.: A unifying perspective on knowledge updates. In del Cerro, L.F., Herzig, A., Mengin, J., eds.: Procs. of JELIA. Volume 7519 of LNAI., Springer (2012) 372–384
13. Wong, K.S.: Forgetting in Logic Programs. PhD thesis, The University of New South Wales (2009)
14. Wang, Y., Wang, K., Zhang, M.: Forgetting for answer set programs revisited. In Rossi, F., ed.: Procs. of IJCAI, IJCAI/AAAI (2013)
15. Knorr, M., Alferes, J.J.: Preserving strong equivalence while forgetting. In Fermé, E., Leite, J., eds.: Procs. of JELIA. Volume 8761 of LNCS., Springer (2014) 412–425
16. Wang, Y., Zhang, Y., Zhou, Y., Zhang, M.: Knowledge forgetting in answer set programming. J. Artif. Intell. Res. (JAIR) **50** (2014) 31–70
17. Delgrande, J.P., Wang, K.: A syntax-independent approach to forgetting in disjunctive logic programs. In Bonet, B., Koenig, S., eds.: Procs. of AAAI, AAAI Press (2015) 1482–1488
18. Zhang, Y., Zhou, Y.: Knowledge forgetting: Properties and applications. Artif. Intell. **173**(16-17) (2009) 1525–1537
19. Gonçalves, R., Knorr, M., Leite, J.: The ultimate guide to forgetting in ASP. In Baral, C., Delgrande, J.P., Wolter, F., eds.: Procs. of KR, AAAI Press (2016) 135–144
20. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4) (2001) 526–541
21. Lifschitz, V., Tang, L.R., Turner, H.: Nested expressions in logic programs. Ann. Math. Artif. Intell. **25**(3-4) (1999) 369–389
22. Cabalar, P., Ferraris, P.: Propositional theories are strongly equivalent to logic programs. TPLP **7**(6) (2007) 745–759
23. Gonçalves, R., Knorr, M., Leite, J.: You can't always forget what you want: on the limits of forgetting in answer set programming. In Fox, M.S., Kaminka, G.A., eds.: Procs. of ECAI, IOS Press (2016)
24. Truszczynski, M.: Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. Artif. Intell. **174**(16-17) (2010) 1285–1306
25. Alferes, J.J., Knorr, M., Wang, K.: Forgetting under the well-founded semantics. In Cabalar, P., Son, T.C., eds.: Procs. of LPNMR. Volume 8148 of LNCS., Springer (2013) 36–41
26. Knorr, M., Alferes, J.J., Hitzler, P.: Local closed world reasoning with description logics under the well-founded semantics. Artif. Intell. **175**(9-10) (2011) 1528–1554
27. Gonçalves, R., Alferes, J.J.: Parametrized logic programming. In Janhunen, T., Niemelä, I., eds.: Procs. of JELIA'10. Volume 6341 of LNCS., Springer (2010) 182–194
28. Slota, M., Leite, J., Swift, T.: On updates of hybrid knowledge bases composed of ontologies and rules. Artif. Intell. **229** (2015) 33–104
29. Gonçalves, R., Knorr, M., Leite, J.: Evolving multi-context systems. In Schaub, T., Friedrich, G., O'Sullivan, B., eds.: Procs. of ECAI, IOS Press (2014) 375–380
30. Brewka, G., Ellmauthaler, S., Pührer, J.: Multi-context systems for reactive reasoning in dynamic environments. In Schaub, T., Friedrich, G., O'Sullivan, B., eds.: Procs. of ECAI, IOS Press (2014) 159–164
31. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.T.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2) (2011) 107–124
32. Ivanov, V., Knorr, M., Leite, J.: A query tool for $\mathcal{EL}$ with non-monotonic rules. In Alani, H., et al., eds.: Procs. of ISWC. Volume 8218 of LNCS., Springer (2013) 216–231
33. Costa, N., Knorr, M., Leite, J.: Next step for NoHR: OWL 2 QL. In Arenas, M., et al., eds.: Procs. of ISWC. Volume 9366 of LNCS., Springer (2015) 569–586