Towards a Knowledge Level Analysis of Forgetting

James P. Delgrande
School of Computing Science
Simon Fraser University
Burnaby, B.C.
V5A 1S6 Canada
jim@cs.sfu.ca

Abstract

Forgetting has been addressed in various areas in KR, including classical logic, logic programming, modal logic, and description logics. Here, we view forgetting as an abstract operator, independent of the underlying logic. We argue that forgetting amounts to a reduction in the signature of a language of a logic, and that the result of forgetting elements of a signature in a theory is the set of logical consequences over the reduced language. This definition offers several advantages. It provides a uniform approach to forgetting, applicable to any logic with a well-defined consequence relation. Obtained results are thus applicable to all subsumed formal systems, and typically are obtained much more straightforwardly. The approach also leads to insights with respect to specific logics: forgetting in first-order logic is somewhat different from the accepted approach; and the definition applied to logic programs yields a new syntax-independent notion of forgetting.

Introduction

Forgetting has been studied in the context of various logical systems, including classical propositional logic (PC), first-order logic (FOL), logic programming, modal logics, and description logics. While there is no generally-agreed upon definition, a common intuition is that in forgetting an agent becomes ignorant of, or unaware of, some part of its language.

Forgetting potentially has several pragmatic uses. For example, forgetting the part of a knowledge base (KB) that is irrelevant to a query may yield more efficient query-answering. Forgetting may provide a formal account of predicate hiding. As well, forgetting may be used in summarising a KB or reusing part of a KB or in clarifying relations between predicates.

A typical approach to addressing forgetting is to specify a host logic, give intuitions, and then provide a formal definition. Consequently, forgetting has been addressed with respect to various logics and with differing intuitions, but with no consensus as to what it means as a general concept.

In this paper, forgetting is addressed as an abstract belief change operator, independent of any specific formal system. The goal is to investigate forgetting at the *knowledge level* and thus independent of syntax. Our thesis is that forgetting amounts to a reduction in the *signature* of a language of a logic. This approach to forgetting is not new; in fact it is, implicitly or explicitly, the most common definition for the term.

This means that, for a given logic over a language with signature σ, forgetting $\sigma' \subseteq \sigma$ from a KB K is the set of consequences of K expressible over $\sigma \setminus \sigma'$. This definition is very simple but offers several advantages. Foremost, it provides a uniform approach, applicable to any logic with a well-defined consequence relation. By taking a general view, various issues are clarified, unified, and simplified. Obtained results are obviously applicable to all subsumed formal systems, and many results follow immediately. An inference-based perspective leads to simpler techniques for computing forgetting in some cases. Since forgetting results in a theory, the main computational challenge is to, if possible, determine a finite representation of forgetting, or a *uniform interpolant*. Notably, a uniform interpolant may not exist (e.g., in FOL) whereas forgetting remains well-defined.

The next section reviews related work while the third section presents notation and definitions. The fourth section presents the general approach. The next section briefly discusses applying the approach to specific logics. This is followed by a concluding section.

Background

The best-know definition of forgetting is due to George Boole, wherein forgetting an atom p from formula ϕ is given by $\phi[p/\top] \vee \phi[p/\bot]$. (Lin and Reiter 1994) addresses forgetting in FOL. In logic programming, work has focussed on computing forgetting via program transformations. E.g.,[1] (Eiter and Wang 2008) give an algorithm for computing forgetting in disjunctive logic programming. These approaches are syntactic, in that forgetting may produce different results for two programs that are strongly equivalent.

In modal logic, most work on forgetting *per se* has dealt with epistemic logic, although some work (e.g. (Ghilardi 1995)) addresses uniform interpolation. (van Ditmarsch *et al.* 2009) present an approach wherein "becoming ignorant" is modelled as an event in a dynamic epistemic logic. According to (van Ditmarsch *et al.* 2009), there is no hard notion of what it means to forget something, whereas in the

[1] The set of references is limited due to space contraints.

present approach, forgetting is uniquely defined.

In description logic, work has also focussed on uniform interpolation. E.g. (Konev *et al.* 2009) addresses uniform interpolation in $\mathcal{EL}$. Characterizations of uniform interpolants are given via bisimulations between models. Work specifically on forgetting includes (Wang *et al.* 2010), which addresses the problem in $\mathcal{ALC}$.

Formal Preliminaries

For an understood signature, a logic for a language is given in terms of a consequence relation $\vdash$. The relation $\vdash$ is assumed to be a Tarskian consequence relation; that is, it satisfies reflexivity, cut, and monotony. We allow sets of formulas on the right hand side of $\vdash$. $\Gamma \leftrightarrow \Psi$ abbreviates $\Gamma \vdash \Psi$ and $\Psi \vdash \Gamma$. As well, $\mathcal{C}n(\Gamma) = \{\phi \mid \Gamma \vdash \phi\}$.

The set of interpretations is denoted by Ω, where the logic and signature are understood from the context. $w \models \phi$ denotes that ϕ is true in interpretation w. $Mod(\phi)$ is the set of models of ϕ. $\Gamma \models \phi$ just if $Mod(\Gamma) \subseteq Mod(\phi)$. For the logics we consider, we have that $\Gamma \models \phi$ iff $\Gamma \vdash \phi$.

σ (σ', etc.) denotes a signature for a (given) logic. $\mathcal{L}_\sigma$ is the language defined over σ, where the language and consequence relation are clear from the context. $Sig(\Gamma)$ denotes the signature of Γ, i.e. the nonlogical symbols mentioned in formulas in Γ. We also use $Sig(w)$ to denote the signature associated with interpretation w.

The above notions may where appropriate be subscripted by a signature. E.g. for an understood logic, Ω_σ denotes the set of interpretations over, or restricted to, σ. In particular, $\mathcal{C}n_\sigma(\Gamma) = \{\phi \mid \Gamma \vdash \phi \text{ where } \phi \in \mathcal{L}_\sigma\}$

Let σ and σ' be signatures where $\sigma \subset \sigma'$. Then σ is a *reduct*[2] of σ', and σ' is an *expansion* of σ. If also $w \in \Omega_\sigma$, $w' \in \Omega_{\sigma'}$, and w and w' agree on the interpretation of symbols in σ then w is the *σ-reduct* of w', and w' is a *σ'-expansion* of w. $w'_{|\sigma}$ is the reduct of w' with respect to σ whereas $w_{\uparrow\sigma'}$ is the set of expansions of w with respect to σ'. This notation extends to sets of models and formulas in the obvious way. E.g. for $\sigma \subseteq \sigma'$, $\Gamma_{|\sigma}$ is the set of formulas in Γ with signature in σ.

The Approach

Our central definition is the following.

Definition 1 *Assume some fixed logic with language $\mathcal{L}$ and signature σ. Let $\Gamma \subseteq \mathcal{L}_\sigma$. The result of* forgetting *σ' in Γ, denoted $\mathcal{F}(\Gamma, \sigma')$, is given by:*

$$\mathcal{F}(\Gamma, \sigma') = \mathcal{C}n_\sigma(\Gamma) \cap \mathcal{L}_{\sigma \setminus \sigma'}.$$

While this definition is very simple, it specifies a unique, unambiguous operator, applicable to any "reasonable" logic. Notably, it applies to logics for which an interpolation theorem does not hold, such as classical FOL, S4, and various relevance logics and description logics. The characterization in Definition 1 is at the *knowledge level* and yields a theory. A key question then is to determine those instances of $\mathcal{F}$ that may be finitely characterized, i.e. for which a uniform interpolant exists.

$\mathcal{F}(\Gamma, \sigma')$ results in a reduced language (unless $\sigma \cap \sigma' = \emptyset$). We can re-express forgetting in the original language (as many approaches do) via:

$$\mathcal{F}_O(\Gamma, \sigma') = \mathcal{C}n_\sigma(\mathcal{F}(\Gamma, \sigma'))$$

The following results are elementary, but show that the definition of forgetting has the "right" properties.

Proposition 1 *Let Γ and Γ' be sets of sentences.*

1. $\Gamma \vdash \mathcal{F}(\Gamma, \sigma)$
2. *If* $\Gamma \leftrightarrow \Gamma'$ *then* $\mathcal{F}(\Gamma, \sigma) \leftrightarrow \mathcal{F}(\Gamma', \sigma)$
3. $\mathcal{F}(\Gamma, \sigma) = \mathcal{C}n_{\sigma'}(\mathcal{F}(\Gamma, \sigma))$ *where* $\sigma' = Sig(\Gamma) \setminus \sigma$.
4. $\mathcal{F}(\Gamma, \sigma_1 \cup \sigma_2) = \mathcal{F}(\mathcal{F}(\Gamma, \sigma_1), \sigma_2))$
5. Γ *is a* conservative extension *of* $\mathcal{F}(\Gamma, \sigma)$.

Thus, forgetting yields no new consequences; it is independent of syntax and results in a deductively-closed theory. Part 4 shows that forgetting is decomposable with respect to a signature; hence it is commutative with respect to its second argument. Last, in forgetting, no results over the reduced signature are lost.

The next results give a model-based characterisation.

Proposition 2 *Let $\sigma' \subseteq \sigma$, and let $\Gamma \subseteq \mathcal{L}_\sigma$.*

1. $Mod_{\sigma \setminus \sigma'}(\mathcal{F}(\Gamma, \sigma')) = Mod_\sigma(\Gamma)_{|(\sigma \setminus \sigma')}$
2. $Mod_\sigma(\mathcal{F}(\Gamma, \sigma')) = (Mod_\sigma(\Gamma)_{|(\sigma \setminus \sigma')})_{\uparrow\sigma}$

Thus the models of $\mathcal{F}(\Gamma, \sigma')$ are exactly the models of Γ restricted to the signature $\sigma \setminus \sigma'$. The second part expresses forgetting with respect to the original signature. The form in Proposition 2.2 has frequently appeared in the literature for characterising forgetting.

(Zhang and Zhou 2009) give four postulates characterising their approach to forgetting in S5. An analogous result follows here.

Definition 2 *Signature σ is* irrelevant *to Γ, $IR(\Gamma, \sigma)$, iff there is Γ' such that $\Gamma \leftrightarrow \Gamma'$ and $Sig(\Gamma') \cap \sigma = \emptyset$.*

Proposition 3 $\Gamma' = \mathcal{F}_O(\Gamma, \sigma')$ *iff*

1. $\Gamma \vdash \Gamma'$
2. *If* $IR(\phi, \sigma')$ *and* $\Gamma \vdash \phi$ *then* $\Gamma' \vdash \phi$
3. *If* $IR(\phi, \sigma')$ *and* $\Gamma \nvdash \phi$ *then* $\Gamma' \nvdash \phi$
4. $IR(\Gamma', \sigma')$

Applying the Framework

We next instantiate the approach to specific logical systems, including PC, FOL, a relevance logic, and disjunctive logic programs. Given space constraints, some familiarity with these approaches is assumed.

Propositional Logic For this part, $\mathcal{L}$ is the language of PC, and $\mathcal{P}$, a signature, is a set of propositional atoms. For set of formulas Γ, Γ_{CNF} is the clause form of Γ, expressed as a set of sets of literals. $Res(S, p)$ is the set of resolvants of clauses in S with respect to p; e.g. for $S = \{\{p, q\}, \{\neg p, r\}, \{\neg p, s\}, \{t\}\}$, $Res(S, p) = \{\{q, r\}, \{q, s\}\}$. We obtain:

[2] This term is standard in model theory. It should not be confused with the term *reduct* in answer set programming.

Theorem 1 *Let $\Gamma \subseteq \mathcal{L}$ and $p \in \mathcal{P}$.*

1. $\mathcal{F}(\Gamma, p) \leftrightarrow \Gamma[p/\top] \vee \Gamma[p/\bot]$ *for finite* Γ.
2. $\mathcal{F}(\Gamma, p) \leftrightarrow \Gamma_{CNF|(\mathcal{P}\setminus\{p\})} \cup Res(\Gamma_{CNF}, p)$.

The first part shows that, restricted to PC, the result of forgetting according to Definition 1 coincides with the Boole definition. The second part gives an alternative means of computing forgetting that offers several advantages over the Boole definition. The Boole definition, while easily computable, only works for a finite set of formulas and results in a disjunction as the top-level connective in a KB (which in general may be awkward to work with). It does not readily extend to other logics including, as we will argue, FOL.

For the second means of computing forgetting, there is no need to require a finite KB, and the resulting KB is in a more useful form. However, it may result in a quadratic (in the number of clauses mentioning p) blowup of the KB. Other properties are inherited from Proposition 1 and 3. The model-theoretic characterisation of forgetting in PC is a corollary to Proposition 2.

First-Order Logic In FOL, nonlogical symbols are of two sorts: predicate and function symbols. (Lin and Reiter 1994) define the forgetting of a predicate symbol via interpretations, using a definition analogous to Proposition 2. Forgetting a constant symbol doesn't appear to have been considered in the literature (nor for that matter, arbitrary function symbols). For forgetting a constant symbol we have the following.

Proposition 4 *Let $\Gamma \subset \mathcal{L}_\sigma$ be finite, c a constant, and $\Gamma_c = \{\phi \in \Gamma \mid c \in \sigma(\phi)\}$. For x not appearing in Γ we have:*

$$\mathcal{F}(\Gamma, c) \leftrightarrow (\Gamma \setminus \Gamma_c) \cup \{\exists x \wedge_{\mu \in \Gamma_c} \mu[c/x]\}.$$

Lin and Reiter also discuss forgetting a ground atomic formula. Our definition of forgetting is inapplicable in this case. Moreover, if Definition 1 were modified to admit ground atomic formulas, it would produce unacceptable results. E.g. consider forgetting that John is a student, $st(J)$, in Γ by taking the set of consequences $\Gamma' \subseteq \Gamma$ that don't mention $st(J)$. Then Γ' would still contain $\exists x(x = J \wedge st(x))$, since this is a consequence of Γ that does not mention $st(J)$. So not only does Γ' still entail $st(J)$, but Γ' is also not a (deductively-closed) theory.

We suggest that $st(J)$, or any well-formed formula for that matter, is not an appropriate object for forgetting: $st(J)$ is an *assertion* and the removal of this fact is in fact the removal of a *proposition*. In such a case the appropriate belief change operation is *contraction*. See the full paper for details.

Relevance Logic We next consider the relevance logic of *first degree entailment* $\mathbf{E}_{fde}$ (Anderson and Belnap Jr. 1975), a logic weaker than classical PC. $\mathbf{E}_{fde}$ provides an example where the standard Boole definition of forgetting is unsuitable. A formula of $\mathbf{E}_{fde}$ is of the form $\phi \rightarrow \psi$ where ϕ, ψ are propositional formulas formed using the standard connectives $\wedge$, $\vee$, $\neg$ but not mentioning $\rightarrow$. In $\mathbf{E}_{fde}$, the so-called "paradoxes of implication" such as $p \wedge \neg p \rightarrow \phi$ and $\phi \rightarrow p \vee \neg p$ are missing. See (Anderson and Belnap Jr. 1975) for an axiomatisation and semantic theory.

If we write $\phi \rightleftarrows \psi$ for $\phi \rightarrow \psi$ and $\psi \rightarrow \phi$, then it can be shown that $\phi \rightleftarrows \phi'$ holds where ϕ' is ϕ in CNF. Moreover, for ϕ, ψ in CNF, determining whether $\phi \rightarrow \psi$ holds can be easily determined: the entailment holds just when for each clause (disjunction) C in ψ there is a clause in ϕ whose literals are are among those in C.

With these facts, it is straightforward to define forgetting in $\mathbf{E}_{fde}$. For Γ a set of propositional formulas formed using connectives $\wedge$, $\vee$, $\neg$, define $\mathcal{C}n(\Gamma)$ by:

$$\mathcal{C}n(\Gamma) = \{\phi \mid \Gamma \rightarrow \phi\}.$$

Then Definition 1 can be applied. We obtain:

Theorem 2 *Let Γ be a set of propositional formulas formed using $\wedge$, $\vee$, $\neg$. Then*

$$\mathcal{F}(\Gamma, p) \rightleftarrows \{C \in \Gamma_{CNF} \mid \textit{ not: } p \rightarrow C\}.$$

Thus forgetting p from Γ is characterised by just those clauses in Γ, in CNF, that don't mention p. So forgetting in this case is a trivial operation once a KB is in CNF.

Logic Programming We turn next to answer set programming (ASP) (Gelfond and Lifschitz 1988). A (*disjunctive*) *logic program* (DLP) over a set of atoms $\mathcal{A}$ is a finite set of rules of the form

$$a_1; \ldots; a_m \leftarrow b_1, \ldots, b_n, {\sim}c_1, \ldots, {\sim}c_p$$

where $a_i, b_j, c_k \in \mathcal{A}$ and $0 \leq m, n, p$.

Operators ';' and ',' express disjunction and conjunction respectively. For atom a, ${\sim}a$ is (default) negation. A program induces 0, 1, or more *answer sets*, roughly, minimal sets of atoms that satisfy the rules. The difficulty is that a program is nonmonotonic with respect to its answer sets. E.g., the program $\{q \leftarrow {\sim}p\}$ has answer set $\{q\}$ while $\{q \leftarrow {\sim}p,\ p\}$ has answer set $\{p\}$.

More recently ASP has been put on a monotonic footing with a notion called *strong equivalence* (Lifschitz *et al.* 2001) that provides an account of logical equivalence for programs. Subsequently, (Wong 2008) has provided an inferential system for disjunctive rules that preserves strong equivalence. The set of inference rules may be found in the aforecited reference; the key point is that these rules allow a consequence relation $\mathcal{C}n(\cdot)$ to be defined, and consequently Definition 1 can be used.

Moreover, we can define a notion analogous to Res in Theorem 1 for computing *forget* in a DLP. That is, for a DLP P and atom a, we can define $ResLP(P, a)$ to be a set of rules such that

1. each rule in $ResLP(P, a)$ is obtained by a single application of an inference rule to rules in P; and
2. no rule in $ResLP(P, a)$ mentions a.

Then we obtain the result:

Theorem 3 *Let P be a DLP and $a \in \mathcal{A}$. Then:*

$$\mathcal{F}(P, a) \leftrightarrow P_{|(\mathcal{A}\setminus\{a\})} \cup ResLP(P, a).$$

The relation $\leftrightarrow$ is that of strong equivalence. Since we inherit the results of Proposition 1 and 3, the results of forgetting are independent of syntax, even though the expression

on the right hand side of Theorem 3 is a set of rules obtained by transforming and selecting rules in P. As in PC, forgetting an atom results in at worst a quadratic blowup in the size of the program. This then results in an interesting operator: As indicated in the Background section, there has been substantial effort in addressing forgetting in ASP. Such previous work is *syntactic*, in that the result of forgetting depends on the form of the underlying logic program, whereas the current approach is independent of how a program is expressed.

Other Logics Clearly the approach can be extended to other classes of logics. Thus, there is no obstacle in principle to providing a syntactic definition (analogous to Theorem 3) for forgetting in S5, for example by exploiting the inferential system RS5 of (Enjalbert and Fariñas del Cerro 1989). Further, the central definitions are applicable to description logics. The current approach then provides a definition of forgetting regardless of whether a particular system has the uniform interpolation property or not.

Conclusion

We have presented an account of forgetting as an abstract, knowledge level, belief change operator, in which forgetting constitutes a reduction in the signature of a KB. By taking a general view, various issues are clarified, unified, and simplified. To this end, the central definition is applicable to any logic with a well-defined consequence relation, and the results obtained are applicable to all subsumed logics. For a given logic and signature, we obtain a unique, syntax-independent operation of forgetting, independent also of a domain of application. Thus, to this last point, forgetting doesn't involve losing information about a domain *per se* but rather involves losing the ability to express, or represent, information about the domain.

We argue that this view leads to insights with respect to specific logics. In the literature, there is no universally-agreed on use of the term *forgetting*, and it has been used for conceptually different phenomena. Some approaches describe forgetting as "becoming ignorant" or base intuitions on forgetting knowledge. Such approaches would appear to address some form of *contraction* in an agent's knowledge.

We have considered "instantiations" of forgetting in PC, FOL, a relevance logic, and in disjunctive logic programs. Forgetting in FOL is somewhat different from the accepted approach; forgetting is readily definable in at least some relevance logics; and the definition applied to disjunctive logic programs yields a new syntax-independent definition of forgetting. Forgetting has most often been identified with uniform interpolation. Indeed there is a close relation: a uniform interpolant is a syntactic object, which may or may not exist for a given logic. Forgetting, in contrast, is well-defined for any "reasonable" logic. If a uniform interpolant exists, it corresponds to a (finite) representation of the result of forgetting. In fact, the key computational question for forgetting is whether a uniform interpolant exists and, if so, how it is determined.

References

A.R. Anderson and N.D. Belnap Jr. *Entailment: The Logic of Relevance and Necessity, Vol. I.* Princeton University Press, 1975.

Thomas Eiter and Kewen Wang. Forgetting in answer set programming. *Artificial Intelligence*, 172(14):1644–1672, 2008.

Patrice Enjalbert and Luis Fariñas del Cerro. Modal resolution in clausal form. *Theoretical Computer Science*, 65(1):1–33, 1989.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. The MIT Press, 1988.

Silvio Ghilardi. An algebraic theory of normal forms. In *Annals of Pure and Applied Logic*, volume 71, pages 189–245, 1995.

Boris Konev, Dirk Walther, and Frank Wolter. Forgetting and uniform interpolation in large-scale description logic terminologies. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 830–835, 2009.

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

F. Lin and R. Reiter. Forget it! In *AAAI Fall Symposium on Relevance*, New Orleans, November 1994.

Hans P. van Ditmarsch, Andreas Herzig, Jérôme Lang, and Pierre Marquis. Introspective forgetting. *Synthese*, 169(2):405–423, 2009.

Zhe Wang, Kewen Wang, Rodney W. Topor, and Xiaowang Zhang. Tableau-based forgetting in $\mathcal{ALC}$ ontologies. In *ECAI'10*, pages 47–52, 2010.

Ka-Shu Wong. Sound and complete inference rules for SE-consequence. *Journal of Artificial Intelligence Research*, 31(1):205–216, January 2008.

Yan Zhang and Yi Zhou. Knowledge forgetting: Properties and applications. *Artificial Intelligence*, 173(16-17):1525–1537, November 2009.