Variable Elimination for DLP-Functions

Ricardo Gonçalves[1] and **Tomi Janhunen**[2] and **Matthias Knorr**[1] and **João Leite**[1] and **Stefan Woltran**[3]
[1]Universidade Nova de Lisboa
[2]Aalto University
[3]Vienna University of Technology

Abstract

Forgetting, or the elimination of middle variables no longer deemed relevant, has recently gained considerable interest in the context of Answer Set Programming (ASP), notably due to the formalization of *strong persistence*, a property based on *strong equivalence* between the program and the result of forgetting modulo the atoms being eliminated, which seems to adequately encode the requirements of the forgetting operation. Whereas it has been shown that in general, in ASP, it is not always possible to forget and obey *strong persistence*, the structure of modules in the form of DLP-functions, namely their restricted interface, invites the investigation of a weaker notion of persistence based on *uniform equivalence*.

Forgetting

The operation of *forgetting* aims at eliminating a set of variables from a knowledge base, while preserving all relationships (direct and indirect) between the remaining variables. Not only has forgetting been shown to be *useful*, e.g., as a means to clean up a theory by eliminating all auxiliary variables that have no relevant declarative meaning, but even *necessary*, e.g., as a means to deal with privacy and legal issues such as to eliminate illegally obtained data (Gonçalves, Knorr, and Leite 2016c), or to comply with the recently enacted General Data Protection Regulation, namely its *right to be forgotten*.

Forgetting has been extensively studied in the context of classical logic (Middeldorp, Okui, and Ida 1996; Lang, Liberatore, and Marquis 2003; Moinard 2007), and, more recently, in the context of Answer Set Programming (ASP) (Zhang and Foo 2006; Eiter and Wang 2008; Wong 2009; Wang, Wang, and Zhang 2013; Knorr and Alferes 2014; Wang et al. 2014; Delgrande and Wang 2015; Gonçalves, Knorr, and Leite 2016a; 2016c; 2017).

Strong Persistence

Most operators and classes of operators of forgetting presented in the context of ASP so far share one common feature, namely that they rely, one way or another, on the notion of *strong equivalence*[1] between answer set programs to determine whether the semantics of the program w.r.t. the atoms not forgotten has been preserved. According to (Goncalves, Knorr, and Leite 2016b), this is best captured by the so-called *strong persistence* (Knorr and Alferes 2014), a property, inspired by *strong equivalence*, that requires that there be a correspondence between the answer sets of a program before and after forgetting a set of atoms, and that such correspondence be preserved in the presence of additional rules not containing the atoms to be forgotten. Formally,

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

where F is a class of forgetting operators, $\mathcal{C}$ the class of programs over the signature $\mathcal{A}$ of a given operator $\mathsf{f} \in \mathsf{F}$, V the set of atoms we want to forget about from program P, $\mathcal{AS}(P)$ the set of answer sets for P, and $S_{\|V}$ the omission of any atoms in V from any elements in S.

However, it has been shown that it is not always possible to forget and satisfy *strong persistence* (Gonçalves, Knorr, and Leite 2016c). Given that *not* forgetting is sometimes not an option, e.g., for legal reasons, it is mandatory to address what to do when we have to forget a set of atoms, but cannot do it without violating this property. This was first addressed in (Gonçalves et al. 2017), where three different alternatives to the relaxation of *strong persistence* were investigated, each corresponding to relaxing one of the three properties that it can be decomposed in.

Yet, *strong persistence* may sometimes be too strong in the context of forgetting, just as *strong equivalence* is sometimes too strong in practical uses of ASP. In fact, in ASP, it is often the case that one separately develops a program P that essentially encodes the declarative specification of the problem it aims to solve, to which one adds a set of facts R that encodes the specific instance to be solved. This has been shown to be an adequate programming principle towards a more modular view of ASP, for example to simplify the composition of program parts into larger programs. In such cases, the most adequate notion of equivalence is the so-called *uniform equivalence* (Eiter and Fink 2003), which only requires that two programs, to be uniformly equivalent, have the same answer sets whenever an arbitrary set of *facts* is added to both of them. For example, if we wish to find a simpler encoding of the problem, we would look for some program P' that is uniformly equivalent to P.

[1]Two programs P_1 and P_2 and strongly equivalent iff $P_1 \cup R$ and $P_2 \cup R$ have the same answer sets, for all context programs R.

DLP-Functions

Modularity in ASP has been the focus of a significant amount of research (Dao-Tran et al. 2009; Harrison and Lierler 2016; Janhunen et al. 2009; Oikarinen and Janhunen 2008). In one of the most significant general approaches to modularity – the so-called *programming-in-the-large* – compositional operators are provided for combining separate and independent modules based on standard semantics. This is the case of DLP-functions (Janhunen et al. 2009). DLP-functions (modules) are essentially *disjunctive logic programs* extended with well-defined input/output interfaces, which allow for the composition of complex programs potentially integrating large numbers of these modules. The input of such a module is composed of a set of facts, which is in perfect harmony with the idea of *uniform equivalence*.

Definition 1 (DLP-function) *A* DLP-function*,* Π*, is a quadruple* $\langle R, I, O, H\rangle$*, where* I*,* O*, and* H *are pairwise distinct sets of* input atoms*,* output atoms*, and* hidden atoms*, respectively, and* R *is a disjunctive logic program such that for each disjunctive rule* $A \leftarrow B, not\, C$ *in* R*,*

1. $A \cup B \cup C \subseteq I \cup O \cup H$*, and*
2. *if* $A \neq \emptyset$*, then* $A \cap (O \cup H) \neq \emptyset$*.*

Just as *strong equivalence* is excessive in modular ASP, if we wish to forget about some atoms from the program P of a module, it also seems excessive to require *strong persistence*, given that we know that the actual inputs of modules consist of facts only.

Indeed, this suggests that in modular ASP, the operation of forgetting should be guided by a weaker form of persistence. In this research, we investigate such weaker form of persistence including classes of forgetting operators that satisfy it and its applicability in the context of forgetting in DLP-functions. In particular, we are interested in the different roles played by atoms in DLP-functions and how they may affect the possibilities for elimination.

Acknowledgments R. Gonçalves, M. Knorr, and J. Leite were partially supported by FCT project FORGET (PTDC/CCI-INF/32219/2017) and by FCT project NOVA LINCS (UID/CEC/04516/2013). T. Janhunen was partially supported by the Academy of Finland grant 251170. R. Gonçalves was partially supported by FCT grant SFRH/BPD/100906/2014. S. Woltran was supported by the Austrian Science Fund (FWF): Y698, P25521.

References

Dao-Tran, M.; Eiter, T.; Fink, M.; and Krennwallner, T. 2009. Modular nonmonotonic logic programming revisited. In Hill, P. M., and Warren, D. S., eds., *Procs. of ICLP*, volume 5649 of *LNCS*, 145–159. Springer.

Delgrande, J. P., and Wang, K. 2015. A syntax-independent approach to forgetting in disjunctive logic programs. In Bonet, B., and Koenig, S., eds., *Procs. of AAAI*, 1482–1488. AAAI Press.

Eiter, T., and Fink, M. 2003. Uniform equivalence of logic programs under the stable model semantics. In *Procs. of ICLP*, 224–238. Springer.

Eiter, T., and Wang, K. 2008. Semantic forgetting in answer set programming. *Artif. Intell.* 172(14):1644–1672.

Gonçalves, R.; Knorr, M.; Leite, J.; and Woltran, S. 2017. When you must forget: Beyond strong persistence when forgetting in answer set programming. *TPLP* 17(5-6):837–854.

Gonçalves, R.; Knorr, M.; and Leite, J. 2016a. Forgetting in ASP: the forgotten properties. In Michael, L., and Kakas, A. C., eds., *Procs. of JELIA*, volume 10021 of *LNCS*, 543–550. Springer.

Goncalves, R.; Knorr, M.; and Leite, J. 2016b. The ultimate guide to forgetting in answer set programming. In Baral, C.; Delgrande, J.; and Wolter, F., eds., *Procs. of KR*, 135–144. AAAI Press.

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 Fox, M. S., and Kaminka, G. A., eds., *Procs. of ECAI*. IOS Press.

Gonçalves, R.; Knorr, M.; and Leite, J. 2017. Iterative variable elimination in ASP. In Oliveira, E. C.; Gama, J.; Vale, Z. A.; and Cardoso, H. L., eds., *Procs. of EPIA*, volume 10423 of *LNCS*, 643–656. Springer.

Harrison, A., and Lierler, Y. 2016. First-order modular logic programs and their conservative extensions. *TPLP* 16(5-6):755–770.

Janhunen, T.; Oikarinen, E.; Tompits, H.; and Woltran, S. 2009. Modularity aspects of disjunctive stable models. *J. Artif. Intell. Res. (JAIR)* 35:813–857.

Knorr, M., and Alferes, J. J. 2014. Preserving strong equivalence while forgetting. In Fermé, E., and Leite, J., eds., *Procs. of JELIA*, volume 8761 of *LNCS*, 412–425. Springer.

Lang, J.; Liberatore, P.; and Marquis, P. 2003. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res. (JAIR)* 18:391–443.

Middeldorp, A.; Okui, S.; and Ida, T. 1996. Lazy narrowing: Strong completeness and eager variable elimination. *Theor. Comput. Sci.* 167(1&2):95–130.

Moinard, Y. 2007. Forgetting literals with varying propositional symbols. *J. Log. Comput.* 17(5):955–982.

Oikarinen, E., and Janhunen, T. 2008. Achieving compositionality of the stable model semantics for smodels programs. *TPLP* 8(5-6):717–761.

Wang, Y.; Zhang, Y.; Zhou, Y.; and Zhang, M. 2014. Knowledge forgetting in answer set programming. *J. Artif. Intell. Res. (JAIR)* 50:31–70.

Wang, Y.; Wang, K.; and Zhang, M. 2013. Forgetting for answer set programs revisited. In Rossi, F., ed., *Procs. of IJCAI*. IJCAI/AAAI.

Wong, K.-S. 2009. *Forgetting in Logic Programs*. Ph.D. Dissertation, The University of New South Wales.

Zhang, Y., and Foo, N. Y. 2006. Solving logic program conflict through strong and weak forgettings. *Artif. Intell.* 170(8-9):739–778.