Tabled Abduction in Logic Programs

ARI SAPTAWIJAYA* and LUÍS MONIZ PEREIRA

Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
(*e-mail:* ar.saptawijaya@campus.fct.unl.pt, lmp@fct.unl.pt)

submitted 10 April 2013; accepted 23 May 2013

Abstract

Abduction has been on the back burner in logic programming, as abduction can be too difficult to implement, and costly to perform, in particular if abductive solutions are not tabled for reuse. On the other hand, current Prolog systems, with their tabling mechanisms, are mature enough to facilitate the specific introduction of tabling abductive solutions (tabled abduction) into them. We conceptualize tabled abduction for abductive normal logic programs, permitting abductive solutions to be reused, from one abductive context to another. The approach relies on a program transformation into tabled logic programs that makes use of the dual transformation, and enables efficiently handling the problem of abduction under negative goals, by introducing dual positive counterparts for them. Tabled abduction is realized in TABDUAL, a system implemented in XSB Prolog, where the dual transformation is refined to permit executing programs with variables and non-ground queries, and to allow performing dualization by-need only. Furthermore, we foster pragmatic approaches in TABDUAL to cater to all varieties of loops in normal logic programs, now complicated by abduction. TABDUAL has been evaluated in practice (with applications in declarative debugging and decision making) by examining its five variants, according to various evaluation objectives. TABDUAL's correctness and complexity are also referred, we identify its features that could migrate to the engine level, in Logic Programming systems wanting to encompass tabled abduction, and we summarize related work.

KEYWORDS: abductive logic programming, tabled abduction, dual transformation, XSB Prolog, applications of abduction.

1 Tabdual: Tabled Abduction in Logic Programs

Abduction has been well studied in the field of computational logic, and logic programming in particular, for a few decades by now (Denecker and de Schreye 1992; Inoue and Sakama 1996; Fung and Kowalski 1997; Eiter et al. 1997; Kakas et al. 1998; Satoh and Iwayama 2000; Alferes et al. 2004). Abduction in logic programs offers a formalism to declaratively express problems in a variety of areas, e.g. in diagnosis, planning, scheduling, reasoning of rational agents, decision making, knowledge assimilation, natural language understanding, security protocols verification, and systems biology (Eshghi 1988; Kakas and Mancarella 1990; Balsa et al. 1995; Gartner et al. 2000; Kakas and Michael 2001; Alberti et al. 2005; Ray et al. 2006; Kowalski and Sadri 2011; Pereira et al. 2013). On the

* Affiliated with Fakultas Ilmu Komputer at Universitas Indonesia, Depok, Indonesia.

other hand, many Prolog systems have become mature and practical, and thus it makes sense to facilitate the use of abduction into such systems, be it two-valued abduction (as adopted in this work) or three-valued, e.g. (Damásio and Pereira 1995).

In abduction, finding some best explanations (i.e. adequate abductive solutions) to the observed evidence, or finding assumptions that can justify a goal, can be very costly. It is often the case that abductive solutions found within one context are also relevant in a different context, and can be reused with little cost. In logic programming, absent of abduction, goal solution reuse is commonly addressed by employing a tabling mechanism. Therefore, tabling appears to be conceptually suitable for abduction, so as to reuse abductive solutions. In practice, abductive solutions reuse is not immediately amenable to tabling, because such solutions go together with an abductive context.

In (Pereira and Saptawijaya 2012), we preliminarily explore the idea of how to benefit from tabling mechanisms in order to reuse priorly obtained abductive solutions, from one abductive context to another. Tabling abductive solutions (tabled abduction) with its prototype TABDUAL, implemented in XSB Prolog (Swift and Warren 2012), consists of a program transformation from abductive normal logic programs into tabled logic programs, plus a library of system predicates. It requires no meta-interpreter, but generates a self-sufficient program transform, on which abduction is subsequently enacted. TABDUAL is available at `http://sourceforge.net/projects/tabdual`.

We simplify the specification of tabled abduction in TABDUAL by introducing the core transformation, which abstracts away from implementation details of its subsequent refinements and more complex constructs, such as loops (i.e. positive loops and loops over negation) in abductive normal logic programs. That is, the core TABDUAL transformation focuses on an innovative re-uptake of prior abductive solution entries in tabled predicates, and the dual transformation (Alferes et al. 2004), on which TABDUAL relies. The dual transformation, initially employed in ABDUAL (Alferes et al. 2004), allows to more efficiently handle the problem of abduction under negative goals, by introducing their positive dual counterparts. It enables us to obtain one abductive solution at a time, just as when we treat abduction under positive goals. The dual transformation defines for each atom A and its set of rules R in a normal program P, a set of dual rules whose head not_A is true if and only if A is false by R in the employed semantics of P. Note that, instead of having a negative goal *not A* as the rules' head, we use its corresponding 'positive' literal, not_A. The specification of the core TABDUAL transformation and further development of TABDUAL are detailed in (Saptawijaya and Pereira 2013b).

Originally, the dual transformation in ABDUAL does not concern itself with programs having variables. We refine the dual transformation in the context of TABDUAL, to allow it dealing with such programs. More precisely, it is refined to help ground (dualized) negative subgoals, and to deal with non-ground negative goals. Regarding the latter, we look just for abductive solutions of such non-ground negative goals, and not for constraints on free variables of its calling arguments, i.e. no constructive negation.

The tabling mechanism in XSB Prolog supports the Well-Founded Semantics (WFS) (van Gelder et al. 1991), which allows dealing with loops in the program and ensuring termination of looping queries. TABDUAL, being implemented in XSB, employs its tabling mechanism as much as possible to deal with loops. TABDUAL detects direct positive loops (of any length), e.g. in $r \leftarrow r$, and fail predicates involved in such loops, as the XSB tabling mechanism does. Nevertheless, tabled abduction introduces a complication

concerning some varieties of loops. For instance, whereas $r \leftarrow r$ fails query r, perforce its dual rule $not\_r \leftarrow not\_r$ succeeds query not_r. In TABDUAL, such positive loops in dualized negation (not_r in the above example) are detected by tracking the ancestors of negative subgoals, whenever they are called from other negative subgoals. The core TABDUAL transformation is therefore adapted, resorting to a pragmatic approach, by maintaining a list of ancestors of negative subgoals. Indeed, this ancestor list implements the co-unfounded set of literals, defined in Definition 3.5 of (Alferes et al. 2004), in order to deal with such loops. Another type of loops that needs to be handled carefully in the transformation is negative loops over negation (of any length), e.g. in

$$p \leftarrow q. \qquad q \leftarrow not\ p.$$

which are taken care by wrapping the positive dual counterparts of negative goals in the bodies of rules (i.e. wrapping not_p for the above example) with the tabled negation predicate ($tnot/1$ in XSB) twice: on the one hand it preserves the semantics of the rule (keeping the truth value by applying *tnot* twice), and on the other hand introducing *tnot* creates the intended negative loops over negation. The reader is referred to (Saptawijaya and Pereira 2013b) for the implementation details of dealing with such loops.

Keeping in mind TABDUAL as a practical tabled abduction system, under the WFS with abduction (Alferes et al. 2004), several pragmatic aspects have been examined from the implementation viewpoint. First, because TABDUAL allows for modular mixes between abductive and non-abductive program parts, one can benefit in the latter part by enacting a simpler translation of predicates in the program comprised just of facts. This simpler treatment distinguishes the transformation between rules in general and predicates defined extensionally by facts alone. It particularly helps avoid superfluous transformation of facts, which would hinder the use of large factual data. Second, we address the issue of potentially heavy transformation load due to producing the *complete* dual rules (i.e. all dual rules regardless of their need), if these are constructed in advance by the transformation. Such a heavy dual transformation makes it a bottleneck of the whole abduction process. A natural solution is instead to perform the dual transformation *by-need*, i.e. dual rules for a predicate are only created as their need is felt during abduction. We detail two approaches to realizing the dual transformation by-need: creating and tabling all dual rules for a predicate on the first invocation of its negation, or, in contrast, lazily generating and storing (instead of tabling) its dual rules in a trie, as new alternatives are required. The former approach leads to an eager dual rules tabling (albeit by-need) transformation (under local table scheduling strategy), whereas the latter permits a by-need driven lazy one (in lieu of batched table scheduling). Third, TABDUAL provides a system predicate that permits accessing ongoing abductive solutions. This is a useful feature and extends TABDUAL's flexibility, as it allows manipulating abductive solutions dynamically, e.g. preferring or filtering ongoing abductive solutions, e.g. checking them explicitly against nogoods at predefined program points. These implementation aspects and others are examined in (Saptawijaya and Pereira 2013a; Saptawijaya and Pereira 2013b).

TABDUAL has been evaluated with various objectives, where five TABDUAL variants (of the same underlying implementation) are examined, by separately factoring out TABDUAL's most important distinguishing features. In the first, we evaluate the benefit of tabling abductive solutions, where we employ an example from declarative debugging to debug incorrect solutions of logic programs, via a process now characterized as abduction

(Saptawijaya and Pereira 2013d), instead of as belief revision (Pereira et al. 1993a; Pereira et al. 1993b). The other case of declarative debugging, that of debugging missing solutions, is used next to evaluate the three dual transformation variants: complete, eager by-need, and lazy by-need. We touch upon tabling so-called *nogoods* of subproblems in the context of abduction (i.e. abductive solution candidates that violate constraints), and show, in the third evaluation, that tabling abductive solutions can be appropriate for tabling nogoods of subproblems. We also evaluate TABDUAL in dealing with programs having loops, in the fourth evaluation, where we also compare its results with ABDUAL, showing that TABDUAL provides more correct and complete results. Finally, we describe how TABDUAL can be applied in action decision making under hypothetical reasoning, and in a real medical diagnosis case (Saptawijaya and Pereira 2013d). The evaluations and their results are detailed in (Saptawijaya and Pereira 2013b; Saptawijaya and Pereira 2013c).

As the core TABDUAL transformation relies on the dual transformation, its correctness stems from that of ABDUAL, shown formally there. It theoretically justifies, supports, and closely reflects the correctness of the core TABDUAL transformation. The details introduced in the core transformation, and its subsequent refinements, including dealing with programs having loops, are just a concrete realization of the more abstract theory of ABDUAL. Its implementation aspects are extra complexities and refinements introduced for TABDUAL to achieve optimizations pertinent to the XSB features, like tabling and tries. Nevertheless, one needs to note that, whereas ABDUAL is restricted to ground programs and queries, TABDUAL caters to programs with variables and non-ground queries. Indeed, its non-groundness does not violate the groundness assumption in the theory of ABDUAL, since one can move the head unifications of a rule to equalities in its body. Thereby, one can think of the ground instances of all the rules, and stick to the dual transformation of ABDUAL with its groundness assumption.

In terms of complexity, the size of the program produced by the core TABDUAL transformation is linear in the size of the input program; the proof is given in (Saptawijaya and Pereira 2013b). This result is similar to that of ABDUAL using the folded dual form, cf. Lemma A.5 in (Alferes et al. 2004). It is known that the problem of query evaluation to abductive frameworks is NP-complete, even for those frameworks in which entailment is based on the WFS (Eiter et al. 1997). In (Alferes et al. 2004), it is shown that the complexity of an ABDUAL query evaluation is proportional to the maximal number of abducibles in any abductive subgoals, and to the number of abducible atoms in the program. In particular, if the set of abducible atoms and ICs are both empty, then the cost of query evaluation is polynomial. The complexity of TABDUAL query evaluation should naturally be based on that of ABDUAL, since TABDUAL also employs the dual transformation. One may observe that the table size, used in tabling abductive solutions, would be proportional to the number of distinct (positive) subgoals in the procedural call-graph, i.e. each first call of the subgoals in a given query will table, as solution entries, the abductive solutions of the called subgoal.

2 Related Work

There have been a plethora of work on abduction in logic programming, cf. (Kakas et al. 1998; Denecker and Kakas 2002) for a survey on this line of work. But, with the exception of ABDUAL (Alferes et al. 2004), we are not aware of any other efforts that have

addressed the use of tabling in abduction for abductive normal logic programs, which may be complicated with loops. Though both ABDUAL and TABDUAL rely on the dual transformation and the same theoretic underpinnings, ABDUAL does not allow variables in rules, which is no restriction in TABDUAL. Differently from ABDUAL, TABDUAL employs no meta-interpreter, but generates a self-sufficient program transform. Moreover, tabling in ABDUAL is employed only to table its meta-interpreter, and it does not address at all the issues raised by the desirable reuse of tabled abductive solutions; the latter being the main concern of TABDUAL.

TABDUAL, being implemented in XSB, is underpinned by WFS, which enjoys the relevance property. Thus, it allows abduction to be performed by need only, induced by the top-down query-oriented procedure, solely for finding the relevant abducibles and their truth value, assuming the ICs are satisfied. This is not the case with the bottom-up approaches for abduction, e.g. (Satoh and Iwayama 1991), where stable models for computing abductive explanations, not necessarily related to an observation, are constructed. Moreover, TABDUAL allows dealing with odd loops in programs because of its 3-valued program semantics, whilst retaining 2-valued abduction by-need and the use of integrity constraints. This is not enjoyed by the bottom-up approach and its 2-valued implementation.

The tabling technique, within the context of statistical abduction, is employed in (Sato and Kameya 2001; Riguzzi and Swift 2011). But they concern themselves with probabilistic logic programs, whereas TABDUAL with abductive normal logic programs. Moreover, they do not employ the dual transformation and other techniques described here. In particular, the tabling technique in (Sato and Kameya 2001) imposes a constraint that does not allow loops in a program, which pose no restrictions at all in TABDUAL.

The reader is referred to (Saptawijaya and Pereira 2013b) for further discussion on related work.

3 Concluding Remarks

The development of TABDUAL thus far hints that several features, which are currently deployed at the object language level, could migrate into an engine-level of Prolog systems that support tabling and, optionally, tries data structure, like XSB. These include migrating operations on tabled abduction entries, such as consistency checking; hiding data structures, like abductive contexts, and attending operations to detect loops in programs; and admixtures of batched and local table scheduling strategies, which may simplify and improve the lazy by-need dual transformation.

Abduction is by now a staple feature of hypothetical reasoning and non-monotonic knowledge representation. It is already mature enough in its concept, deployment, applications, and proof-of-principle, to warrant becoming a run-of-the-mill ingredient in a Logic Programming environment. We hope this work will lead, in particular, to an XSB System that can provide its users with specifically tailored tabled abduction facilities.

Acknowledgements Ari Saptawijaya acknowledges the support of FCT/MEC Portugal, grant SFRH/BD/72795/2010. We thank Terrance Swift and David Warren for their expert advice in dealing with implementation issues in XSB.

References

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., and Torroni, P. 2005. Security protocols verification in abductive logic programming. In *6th Int. Workshop on Engineering Societies in the Agents World (ESAW)*. LNCS, vol. 3963. Springer.

Alferes, J. J., Pereira, L. M., and Swift, T. 2004. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theory and Practice of Logic Programming 4,* 4, 383–428.

Balsa, J., Dahl, V., and Lopes, J. G. P. 1995. Datalog grammars for abductive syntactic error diagnosis and repair. In *Proc. Natural Language Understanding and Logic Programming Workshop*.

Damásio, C. V. and Pereira, L. M. 1995. Abduction over 3-valued extended logic programs. In *Procs. 3rd. Intl. Conf. Logic Programming and Non-Monotonic Reasoning (LPNMR)*. LNAI, vol. 928. Springer, 29–42.

Denecker, M. and de Schreye, D. 1992. SLDNFA: An abductive procedure for normal abductive programs. In *Procs. of the Joint Intl. Conf. and Symp. on Logic Programming*. The MIT Press.

Denecker, M. and Kakas, A. C. 2002. Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond*. Springer Verlag.

Eiter, T., Gottlob, G., and Leone, N. 1997. Abduction from logic programs: semantics and complexity. *Theoretical Computer Science 189,* 1-2, 129–177.

Eshghi, K. 1988. Abductive planning with event calculus. In *Proc. Intl. Conf. on Logic Programming*. The MIT Press.

Fung, T. H. and Kowalski, R. 1997. The IFF procedure for abductive logic programming. *Journal of Logic Programming 33,* 2, 151–165.

Gartner, J., Swift, T., Tien, A., Damásio, C. V., and Pereira, L. M. 2000. Psychiatric diagnosis from the viewpoint of computational logic. In *Procs. 1st Intl. Conf. on Computational Logic (CL 2000)*. LNAI, vol. 1861. Springer, 1362–1376.

Inoue, K. and Sakama, C. 1996. A fixpoint characterization of abductive logic programs. *J. of Logic Programming 27,* 2, 107–136.

Kakas, A., Kowalski, R., and Toni, F. 1998. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay, C. Hogger, and J. Robinson, Eds. Vol. 5. Oxford U. P.

Kakas, A. C. and Mancarella, P. 1990. Knowledge assimilation and abduction. In *Intl. Workshop on Truth Maintenance*. ECAI'90.

Kakas, A. C. and Michael, A. 2001. An abductive-based scheduler for air-crew assignment. *J. of Applied Artificial Intelligence 15,* 1-3, 333–360.

Kowalski, R. and Sadri, F. 2011. Abductive logic programming agents with destructive databases. *Annals of Mathematics and Artificial Intelligence 62,* 1, 129–158.

Pereira, L. M., Damásio, C. V., and Alferes, J. J. 1993a. Debugging by diagnosing assumptions. In *Automatic Algorithmic Debugging*. LNCS, vol. 749. Springer, 58–74.

Pereira, L. M., Damásio, C. V., and Alferes, J. J. 1993b. Diagnosis and debugging as contradiction removal in logic programs. In *Progress in Artificial Intelligence*. LNAI, vol. 727. Springer, 183–197.

Pereira, L. M., Dell'Acqua, P., Pinto, A. M., and Lopes, G. 2013. Inspecting and preferring abductive models. In *The Handbook on Reasoning-Based Intelligent Systems*, K. Nakamatsu and L. C. Jain, Eds. World Scientific Publishers, 243–274.

Pereira, L. M. and Saptawijaya, A. 2012. Abductive logic programming with tabled abduction. In *Procs. 7th Intl. Conf. on Software Engineering Advances (ICSEA)*. ThinkMind, 548–556.

Ray, O., Antoniades, A., Kakas, A., and Demetriades, I. 2006. Abductive logic programming in the clinical management of hiv/aids. In *Proc. 17th. European Conference on Artificial Intelligence*. IOS Press.

Riguzzi, F. and Swift, T. 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming 11,* 4-5, 433–449.

Saptawijaya, A. and Pereira, L. M. 2013a. Implementing tabled abduction in logic programs. In *Procs. 16th Portuguese Intl. Conf. on Artificial Intelligence (EPIA)*. Doctoral Symposium on Artificial Intelligence (SDIA).

Saptawijaya, A. and Pereira, L. M. 2013b. Tabled abduction in logic programs. Tech. rep., CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa. Available at `http://centria.di.fct.unl.pt/~lmp/publications/online-papers/tabdual_lp.pdf`.

Saptawijaya, A. and Pereira, L. M. 2013c. Towards practical tabled abduction in logic programs. In *16th Portuguese Conference on Artificial Intelligence (EPIA)*. LNAI. Springer.

Saptawijaya, A. and Pereira, L. M. 2013d. Towards practical tabled abduction usable in decision making. In *Procs. 5th. KES Intl. Symposium on Intelligent Decision Technologies (KES-IDT)*. Frontiers of Artificial Intelligence and Applications (FAIA). IOS Press.

Sato, T. and Kameya, Y. 2001. Parameter learning of logic programs for symbolic-statistical modeling. *J. of Artificial Intelligence Research (JAIR) 15*, 391–454.

Satoh, K. and Iwayama, N. 1991. Computing abduction by using the TMS. In *Procs. 8th Intl. Conf. on Logic Programming (ICLP)*. The MIT Press, 505–518.

Satoh, K. and Iwayama, N. 2000. Computing abduction by using TMS and top-down expectation. *Journal of Logic Programming 44,* 1-3, 179–206.

Swift, T. and Warren, D. S. 2012. XSB: Extending Prolog with tabled logic programming. *Theory and Practice of Logic Programming 12,* 1-2, 157–187.

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