Proving Termination with (Boolean) Satisfaction

Michael Codish*

Department of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva, Israel mcodish@cs.bgu.ac.il

1 Introduction

At some point there was the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [6]. Forty five years later, research on Boolean satisfiability (SAT) is still ceaselessly generating even better SAT solvers capable of handling even larger SAT instances. Remarkably, the majority of these tools still bear the hallmark of the DPLL algorithm. In sync with the availability of progressively stronger SAT solvers is an accumulating number of applications which demonstrate that real world problems can often be solved by encoding them into SAT. When successful, this circumvents the need to redevelop complex search algorithms from scratch.

This presentation is about the application of Boolean SAT solvers to the problem of determining program termination. Proving termination is all about the search for suitable ranking functions. The key idea in this work is to encode the search for particular forms of ranking functions to Boolean statements which are satisfiable if and only if such ranking functions exist. In this way, proving termination can be performed using a state-of-the-art Boolean satisfaction solver.

2 Encoding Lexicographic Path Orders

In [3] we describe a propositional encoding for lexicographic path orders (LPOs) [15, 8] and the corresponding LPO-termination property of term rewrite systems. In brief, a term rewrite system (TRS) is a set of rules of the form $\ell \to r$ where ℓ and r are terms constructed from given sets of symbols and variables. A lexicographic path order is an order $\succ_{lpo}$ on terms, induced from a partial order $>$ on the symbols occurring in the terms (a so-called *precedence*). A term rewrite system is LPO-terminating if and only if there exists a partial order on the symbols such that the induced LPO orients all of the rules in the system. Namely such that $\ell \succ_{lpo} r$ for each rule $\ell \to r$.

There are two variants of LPO-termination: "strict" and "quasi" depending on if we restrict the precedence to be strict or not. Both imply termination of the corresponding term rewrite system. Quasi-LPO-termination is typically the

* Supported by the Frankel Center for Computer Sciences at Ben-Gurion University

harder problem as the search for a non-strict precedence is more extensive than that for a strict precedence. Both of the corresponding decision problems, strict- and quasi- LPO-termination, are decidable and NP complete [16].

We encode an LPO-termination problem to SAT in two steps: first, a *partial order constraint* on the symbols in the system is derived; then this constraint is solved through an encoding to SAT obtained by viewing each symbol as an integer value corresponding to its index in the partial order. Partial order constraints are propositional formula in which the atoms are statements about a partial order on a finite set of symbols and can be seen as an instance of the more general formulae of separation logic (sometimes called difference logic) described in [23].

Consider an example. To orient a rule $not(or(A, B)) \rightarrow and(not(A), not(B))$, is reduced to solving the following partial order constraint on the symbols $\{or, and, not\}$:

$$((or > and) \wedge (or > not)) \vee (not > and).$$

We encode each of the three symbols as an integer in two bits, and each atom in the partial order constraint as a comparison on a pair of integers in bit representation. For instance, numbering the bits with subscripts on the symbols, the encoding of the atom $(or > and)$ works out to:

$$(((\underbrace{or_{[2]} \wedge \neg and_{[2]}}_{or_{[2]} > and_{[2]}}) \vee (\underbrace{or_{[2]} \leftrightarrow and_{[2]}}_{or_{[2]} = and_{[2]}} \wedge \underbrace{or_{[1]} \wedge \neg and_{[1]}}_{or_{[1]} > and_{[1]}}))$$

The experimental results presented in [3] are unequivocal. Our SAT based implementation of LPO-termination surpasses in orders of magnitude the performance of previous implementations such as those provided at the time by the termination proving tools TTT [13] and AProVE [12].

3 Encoding Argument Filterings

Lexicographic path orders on their own are too weak for many interesting termination problems and hence are typically combined with more sophisticated termination proving techniques. One of the most popular and powerful such techniques is the *dependency pair* (DP) method [1]. A main advantage is that this allows the application of *argument filterings* which specify parts of terms that should be ignored when comparing terms. It can be viewed like this: given a set of pairs of terms to orient with an LPO, first decide which parts of the terms to filter away and then orient the filtered pairs in an LPO. The argument filtering specifies for each function symbol f if subterms of the form $f(s_1, \ldots, s_n)$ should be *collapsed* to their i^{th} argument; or if some of the argument positions should be filtered away. Filtering terms can simplify considerably the partial order constraints that need be solved to find an LPO. However, argument filterings represent also a severe bottleneck for the automation of dependency pairs, as the search space for argument filterings is enormous (exponential in the sum of the arities of the symbols).

In [5] we introduce a propositional encoding which combines the search for an LPO with the search for an argument filtering. The key idea is to introduce a small number of additional Boolean variables: one for each symbol to indicate if it is collapsed, and one for each argument position of a symbol to indicate if it is filtered. Then the encoding of LPO is enhanced to consider these new variables. So, there exist an argument filtering and an LPO which orient a set of inequalities if and only if the encoding of the inequalities is satisfiable. Moreover, each model of the encoding corresponds to a suitable argument filtering and a suitable LPO which orient the inequalities. Once again experimental results [5] indicate speedups in orders of magnitude.

4 Encoding Recursive Path Orders

In [22] we introduce two additional extensions which together lead to an encoding of the so-called recursive path order with status (RPO). In the first extension, the lexicographic path order is extended to consider the lexicographic extension, not just from left-to-right, but rather with respect to any fixed order. It can be viewed like this: given a permutation for each symbol in a term rewrite system, first reorder the arguments of every subterm as prescribed by the permutation for its root symbol. Then check if the resulting system is LPO-terminating. So, now to orient a set of rules we seek a partial order on the symbols as well as permutations for each symbol. For the encoding, we introduce a small number of additional Boolean variables to represent for each symbol the order its arguments are permuted to. Then the encoding of LPO is enhanced to consider this order (in terms of these new variables). In the second extension, we consider an encoding of the the multiset path order (MPO) [7] where term arguments are compared with the multiset ordering. Also, in this case, with a small number of additional Boolean variables we can model the multiset order in the encoding. For RPO, each symbol in the system is associated with a *status* (one more Boolean variable per symbol in the encoding) indicating if its arguments are to be compared with a multiset extension or with a lexicographic extension modulo some permutation. By now the reader will not be surprised that we simply encode all of the components for RPO to SAT to obtain an implementation using a SAT solver. The results presented in [22] again leave no doubt that encoding to SAT is the way to go.

5 Experimental Results

Throughout this work we have found Prolog a convenient language for expressing the various encodings to SAT. Prototype analyzers were written in SWI Prolog [25] applying the MiniSAT solver [20] through its Prolog interface described in [4]. Subsequently the approach has been integrated within the termination analyzer AProVE [11], using the SAT4J solver [21].

In [22] we report the following results for the various analyses described in this paper. We tested the implementation on all 865 TRSs from the TPDB

[24]. The TPDB is the collection of examples used in the annual *International Termination Competition* [19]. The experiments were run under AProVE on a 2.2 GHz AMD Athlon 64 with a time-out of 60 seconds (as in the *International Termination Competition* [19]).

In the table below, the first two rows compare our SAT-based approach for application of the various path orders to the previous dedicated solvers for path orders in AProVE 1.2 which did not use SAT solving. The last two rows give a similar comparison for the path orders in combination with the dependency pairs method and argument filterings. The columns contain the data for LPO with strict and non-strict precedence (denoted *lpo*/*qlpo*), for LPO with permutations (*lpos*/*qlpos*), for MPO (*mpo*/*qmpo*), and for RPO with status (*rpo*/*qrpo*). For each encoding we give the number of TRSs which could be proved terminating (with the number of time-outs in brackets) and the analysis time (in seconds) for the full collection (including time-outs). For the SAT based implementation, checking the full collection of 865 TRSs for strict-RPO termination with argument filterings requires about 100 seconds. Allowing non-strict orders takes about 3 times longer.

	Solver	*lpo*	*qlpo*	*lpos*	*qlpos*	*mpo*	*qmpo*	*rpo*	*qrpo*
1	SAT-based	**123** (0)	**127** (0)	**141** (0)	**155** (0)	**92** (0)	**98** (0)	**146** (0)	**162** (0)
	(direct)	**31.0**	**44.7**	**26.1**	**40.6**	**49.4**	**74.2**	**50.0**	**85.3**
2	dedicated	**123** (5)	**127**(16)	**141** (6)	**154**(45)	**92** (7)	**98**(31)	**145**(10)	**158** (65)
	(direct)	**334.4**	**1426.3**	**460.4**	**3291.7**	**653.2**	**2669.1**	**908.6**	**4708.2**
3	SAT-based	**357** (0)	**389** (0)	**362** (0)	**395** (2)	**369** (0)	**408** (1)	**375** (0)	**416** (2)
	(arg. filt.)	**79.3**	**199.6**	**69.0**	**261.1**	**110.9**	**267.8**	**108.8**	**331.4**
4	dedicated	**350**(55)	**374**(79)	**355**(57)	**380**(92)	**359**(69)	**391**(82)	**364**(74)	**394**(102)
	(arg. filt.)	**4039.6**	**5469.4**	**4522.8**	**6476.5**	**5169.7**	**5839.5**	**5536.6**	**7186.1**

The table shows that with our SAT encodings, performance improves by orders of magnitude over existing solvers both for direct analysis with path orders and for the combination of path orders and argument filterings in the DP framework. Note that without a time-out, this effect would be intensified. By using SAT, the number of time-outs reduces dramatically from up to 102 to at most 2. The two remaining SAT examples with time-out have function symbols of high arity and can only be shown terminating by further sophisticated termination techniques in addition to RPO. Apart from these two, for SAT, there are only 15 examples that take longer than two seconds and only 3 of these take longer than 10 seconds. The table also shows that the use of RPO instead of LPO increases the proving power substantially, while in the SAT-based setting, runtimes increase only mildly.

6 Other SAT Based Termination Analyses

The first encoding of a termination problem into propositional logic is presented in [17]. The encoding is different than the one we consider and adopts a BDD-based representation. It does not provide competitive results. However, it makes

an important step. Another BDD-based encoding, this one for size-change termination [18], is described in [2]. Here, sets of size change graphs are viewed as partial order constraints, similar to those considered in this paper for term rewrite systems.

In the past year, several additional papers [9, 10, 14, 26] have illustrated the huge potential in applying SAT solvers for other types of termination proving techniques for term rewrite systems. A common theme in all of these works is to represent (finite domain) integer variables as binary numbers in bit representation and to encode arithmetic constraints as Boolean functions on these representations. Results indicate uniformly that the SAT based approach to proving termination is very attractive.

7 Summary

Lexicographic- and multiset- path orders are about lifting a base order on terms to consider the arguments of terms as sequences or as multisets with corresponding lexicographic or multiset orders. We have introduced a new kind of propositional encoding for reasoning about termination of term rewrite systems based on variants of these path orders. Our results have had a direct impact on the design of several major termination analyzers for term rewrite systems.

Of particular and general interest are the encoding techniques which enable to refine a search algorithm to consider a property of interest for all subsets of objects, instead of for the full set of objects; or to check if a property holds when considering a sequence of objects in any order, instead of in the fixed left-to-right order. The common theme is to represent with a small number of additional Boolean variables the large number of cases which need be considered. For the extensions of LPO-termination considered in this work, the additional cost in analysis time is minor in comparison to the increase in the size of the search space.

Acknowledgment. The author has been lucky to work on this research with friends, old and new. Thankyou coauthors of [3], [5], and [22]: Elena Annov, Jürgen Giesl, Vitaly Lagoon, Peter Schneider–Kamp, Peter J. Stuckey, and René Thiemann.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
2. M. Codish, V. Lagoon, P. Schachte, and P. J. Stuckey. Size-change termination analysis in k-bits. In P. Sestoft, editor, *Proceedings of the European Symposium on Programming*, volume 3924 of *Lecture Notes in Computer Science*, pages 230–245. Springer, 2006.
3. M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for LPO termination. In *Proc. RTA '06*, volume 4098 of *LNCS*, pages 4–18, 2006.

4. M. Codish, V. Lagoon, and P. J. Stuckey. Logic programming with satisfiability. *Theory and Practice of Logic Programming*, 2007. (To appear) `http://arxiv.org/pdf/cs.PL/0702072`.
5. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. SAT solving for argument filterings. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 30–44. Springer, November 2006.
6. M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
7. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
8. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1/2):69–116, 1987.
9. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In U. Furbach and N. Shankar, editors, *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *Lecture Notes in Computer Science*, pages 574–588. Springer, 2006.
10. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 340–354, 2007.
11. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, volume 4130 of *LNAI*, pages 281–286, 2006.
12. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In v. Oostrom, editor, *Proc. RTA '04*, volume 3091 of *LNCS*, pages 210–220, Aachen, Germany, 2004.
13. N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA)*, volume 3467 of *Lecture Notes in Computer Science*, pages 175–184, Nara, Japan, 2005. Springer.
14. D. Hofbauer and J. Waldmann. Termination of string rewriting with matrix interpretations. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications (RTA)*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342, 2006.
15. S. Kamin and J.-J. Levy. Two generalizations of the recursive path ordering. Department of Computer Science, University of Illinois, Urbana, IL. Available at `http://www.ens-lyon.fr/LIP/REWRITING/OLD_PUBLICATIONS_ON_TERMINATION` (viewed December 2005), 1980.
16. M. Krishnamoorthy and P. Narendran. On recursive path ordering. *Theoretical Computer Science*, 40:323–328, 1985.
17. M. Kurihara and H. Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Innovations in Applied Artificial Intelligence, 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Proceedings*, volume 3029 of *Lecture Notes in Computer Science*, pages 827–837, Ottawa, Canada, 2004. Springer.
18. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. *ACM SIGPLAN Notices*, 36(3):81–92, 2001. Proceedings of POPL'01.

19. C. Marché and H. Zantema. The termination competition. In *Proc. RTA '07*, LNCS, 2007. To appear.
20. MiniSAT solver. `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat`. Viewed December 2005.
21. SAT4J satisfiability library for `Java`. `http://www.sat4j.org`.
22. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In F. Wolter, editor, *Proceedings of 6th International Symposium on Frontiers of Combining Systems (FroCoS '07)*, volume 4720 of *Lecture Notes on Artificial Intelligence*, page (to appear). Springer-Verlag, September 2007.
23. M. Talupur, N. Sinha, O. Strichman, and A. Pnueli. Range allocation for separation logic. In *CAV*, pages 148–161, 2004.
24. The termination problem data base. `http://www.lri.fr/~marche/tpdb/`.
25. J. Wielemaker. An overview of the SWI-Prolog programming environment. In F. Mesnard and A. Serebenik, editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium, Dec. 2003. Katholieke Universiteit Leuven. CW 371.
26. H. Zankl and A. Middeldorp. Satisfying KBO constraints. In F. Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications*, volume 4533 of *Lecture Notes in Computer Science*, pages 389–403, 2007.