SAT-based Termination Analysis for Java Bytecode with AProVE*

Carsten Fuhs

LuFG Informatik 2, RWTH Aachen University, Germany
fuhs@informatik.rwth-aachen.de

Abstract. SAT solvers are nowadays the standard solving engines for the search problems in automated termination analysis. Consequently, the performance of current termination tools heavily relies on the speed of modern SAT solvers on the corresponding SAT encodings. If a model for the SAT instance at hand is found, it can be used to instantiate the parameters for the current proof step to advance the termination proof. This SAT benchmark submission has been created using the automated termination prover AProVE [6]. All instances originate from termination analysis of Java Bytecode programs. This whole benchmark suite only consists of satisfiable instances, and any speed-up for SAT solvers on these instances will directly lead to performance improvements also for automated termination provers.

1 Introduction

Termination is one of the most important properties of programs. Therefore, there is a need for suitable methods and tools to analyze the termination behavior of programs automatically. In particular, there has been intensive research on techniques for termination analysis of *term rewrite systems* (TRSs) [2]. Instead of developing many separate termination techniques for different programming languages, it is a promising approach to transform programs from different languages into TRSs instead. Then termination tools for TRSs can be used for termination analysis of many different programming languages such as Java Bytecode, cf. e.g. [5,8,10,9,3].

The increasing interest in automated termination analysis is also demonstrated by the *International Competition of Termination Tools*,[1] held annually since 2004. Here, each participating tool is applied to the examples from the *Termination Problem Data Base (TPDB)*[2] and gets 60 seconds per termination problem to prove or disprove termination. Thus, in order for a termination prover to be competitive, one needs efficient search techniques for finding termination (dis)proofs automatically.

However, many of the arising search problems in automated termination analysis are NP-complete. Due to the impressive performance of modern SAT

* Description of benchmark instances submitted to the *SAT Competition 2011*.
[1] See http://termination-portal.org/wiki/Termination_Competition.
[2] See http://termination-portal.org/wiki/TPDB.

solvers, in recent years it has become common practice to tackle such problems by encoding them to SAT and by then applying a SAT solver on the resulting CNF to advance the modular termination proof.

2 Benchmark Instances

This SAT benchmark submission has been created using the automated termination prover AProVE [6], which can be used to analyze the termination behavior of term rewriting systems, Java Bytecode programs [9,3], Prolog programs [10], and Haskell 98 programs [5]. More concretely, in this benchmark suite we focus on SAT instances that stem from termination analysis of Java Bytecode.

Recently, automated termination analysis for Java Bytecode programs has become an area of intensive research [11,1,9,3]. This is reflected by Java Bytecode being the most recent addition of input languages to the TPDB, which consists of thousands of termination problems in various formalisms (currently these are term rewrite systems in different flavors, Java Bytecode, Prolog, and Haskell programs) used as benchmarks in the Termination Competition. In AProVE, termination analysis of Java Bytecode is performed in a two-stage process [9,3]:

1. The Java Bytecode program is translated into *Integer Term Rewrite Systems (ITRSs)*. ITRSs are term rewrite systems (TRSs) extended by built-in integers [4] in order to combine the power of TRS techniques on user-defined data types with a powerful treatment of pre-defined integers. In this way, ITRSs are a suitable intermediate representation for termination analysis of programming languages with built-in integers. The translation from Java Bytecode to ITRSs is designed in such a way that termination of the ITRSs implies termination of the original Java Bytecode program.
2. Then dedicated techniques for ITRSs [4] are invoked to complete the termination proof. Here the corresponding constraint-based techniques in AProVE are automated by a reduction to SAT. Here we first encode into a propositional formula with arbitrary junctors. This formula is represented via a directed acyclic graph with sharing of identical subformulas (i.e., structural hashing). Then we convert this formula DAG into an equisatisfiable formula in CNF using SAT4J's [7] implementation of Tseitin's algorithm [12].

Details. The submitted CNFs are named `AProVE11-`n`.dimacs`. For the analyzed Java Bytecode programs from the TPDB, Fig. 1 provides details on the termination problem for each n. Here all paths need to be prefixed by `tpdb-8.0/JBC/`.

3 Conclusion

SAT solving nowadays is a key technology for automated termination analysis of programs written e.g. in Java Bytecode. Therefore, any improvements in efficiency of SAT solvers on the submitted SAT instances will also have a direct impact on efficiency and power of the state of the art in termination proving for Java Bytecode.

Fig. 1. Details on the submitted SAT instances from TPDB problems

n	Termination problem
01	`Aprove_09/Count.jar`
02	`Aprove_09/Count.jar`
03	`Aprove_09/LogAG.jar`
04	`Aprove_09/PastaA10.jar`
05	`Aprove_09/SortCount.jar`
06	`Aprove_09/SortCount.jar`
07	`Costa_Julia_09/KnapsackDP.jar`
08	`Costa_Julia_09/KnapsackDP.jar`
09	`Julia_10_Iterative/Infix2Postfix.jar`
10	`Julia_10_Iterative/Infix2Postfix.jar`
11	`Julia_10_Iterative/Iterations.jar`
12	`Julia_10_Iterative/RSA.jar`
13	`Julia_10_Iterative/Test3.jar`
14	`Julia_10_Iterative/Test5.jar`
15	`Julia_10_Iterative/TriTas.jar`
16	`Julia_10_Iterative/TriTas.jar`

References

1. E. Albert, P. Arenas, M. Codish, S. Genaim, G. Puebla, and D. Zanardini. Termination analysis of Java Bytecode. In *Proc. Formal Methods for Open Object-Based Distributed Systems (FMOODS '08)*, volume 5051 of *LNCS*, pages 2–18, 2008.
2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
3. M. Brockschmidt, C. Otto, C. von Essen, and J. Giesl. Termination graphs for Java Bytecode. In *Verification, Induction, Termination Analysis - Festschrift for Christoph Walther on the Occasion of His 60th Birthday*, volume 6463 of *LNAI*, pages 17–37, 2010.
4. C. Fuhs, J. Giesl, M. Plücker, P. Schneider-Kamp, and S. Falke. Proving termination of integer term rewriting. In *Proc. Rewriting Techniques and Applications (RTA '09)*, volume 5595 of *LNCS*, pages 32–47, 2009.
5. J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM TOPLAS*, 33:1–39, 2011.
6. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. International Joint Conference on Automated Reasoning (IJCAR '06)*, volume 4130 of *LNAI*, pages 281–286, 2006.
7. D. Le Berre and A. Parrain. The SAT4J library, release 2.2. *Journal on Satisfiability, Boolean Modelling and Computation (JSAT)*, 7:59–64, 2010.
8. E. Ohlebusch. Termination of logic programs: Transformational methods revisited. *AAECC*, 12(1-2):73–116, 2001.
9. C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of Java Bytecode by term rewriting. In *Proc. Rewriting Techniques and Applications (RTA '10)*, volume 6 of *LIPIcs*, pages 259–276, 2010.

10. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination proofs for logic programs by term rewriting. *ACM TOPLAS*, 11:1–52, 2009.
11. F. Spoto, F. Mesnard, and É. Payet. A termination analyser for Java Bytecode based on path-length. *ACM TOPLAS*, 32(3), 2010.
12. G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. 1968. Reprinted in *Automation of Reasoning*, volume 2, pages 466–483, Springer, 1983.