EPiC Series in Computing

Volume 38, 2016, Pages 29–32

Proceedings of the 1st and 2nd Vampire Workshops

SAT Solving Experiments in Vampire*

Armin Biere[1], Ioan Dragan[2], Laura Kovács[2,3], and Andrei Voronkov[4]

[1] Johannes Kepler University, Linz, Austria
biere@jku.at
[2] Vienna University of Technology, Vienna, Austria
ioan.dragan@tuwien.ac.at
[3] Chalmers University of Technology, Gothenburg, Sweden
laura.kovacs@chalmers.se
[4] University of Manchester, Manchester, U.K.
andrei@voronkov.com

Abstract

In order to better understand how well a state of the art SAT solver would behave in the framework of a first-order automated theorem prover we have decided to integrate Lingeling [2], best performing SAT solver, inside Vampire's AVATAR [5] framework. In this paper we propose two ways of integrating a SAT solver inside of Vampire and evaluate overall performance of this combination. Our experiments show that by using a state of the art SAT solver in Vampire we manage to solve more problems. Surprisingly though, there are cases where combination of the two solvers does not always prove to generate best results.

Extended Abstract

Automated theorem proving attracted a lot of attention both from academia and industry in recent years. Today automated theorem proving is used in many industry branches. Most of its applications are related to verification of correctness for software. This approach has proven to be essential in safety-critical systems, like airplane software verification and validation. Another emerging market for theorem provers is the automotive industry, where program verification is also a must. But as in most of the verification fields, theorem proving alone cannot solve all the problems arising in industry. For this purpose a lot of effort was spent on integrating different technologies into a single solver. Recently a new architecture for theorem provers, called AVATAR (Advanced Vampire Architecture for Theories And Resolution), was proposed. This architecture facilitates the integration of different solvers, e.g. SAT, SMT, QBF solvers, inside of a theorem prover.

Within the AVATAR framework a SAT solver is mainly used for helping the theorem prover in splitting and asserting clauses. The base observation for splitting a first-order clause comes from the following idea. Assume that we have a set $\mathcal{S}$ of first-order clauses and a clause $C_1 \vee C_2$ such that all the variables in C_1 and C_2 are disjoint. Then the clause $\forall\ (C_1 \vee C_2)$ is equivalent

*This work was partially supported by the Wallenberg Academy Fellowship 2014, the Swedish VR grant D0497701 and the Austrian research projects FWF S11408-N23, S11409-N23, and S11410-N23.

to $\forall\ (C_1) \vee \forall\ (C_2)$, which implies that the set $\mathcal{S} \cup \{C_1, C_2\}$ is unsatisfiable if and only if both $\mathcal{S} \cup \{C_1\}$ and $\mathcal{S} \cup \{C_2\}$ are unsatisfiable. By doing this operation the search space is split into two search spaces that operate on shorter and light-weighter clauses and each of them can be saturated separately.

In a nutshell the AVATAR framework works as follows: the first-order reasoning part works as usual, using a saturation algorithm. Whereas the main difference comes from the use of a SAT solver for deciding splittings. Assume that we have a clause $C_1 \vee C_2 \vee \cdots \vee C_n$ that is splittable into components $C_1, C_2, \ldots, C_n$ and the clause passes the retention test, this clause is not added to the set of *passive* clauses but rather a new propositional clause $[C_1] \vee [C_2] \vee \cdots \vee [C_n]$ is added to the SAT solver and the solver is asked for a solution. In case the problem is proven to be unsatisfiable, then we report the result to the first-order reasoning part, which in turn returns unsatisfiability of the original problem. If the problem is proven to be satisfiable then a model for the propositional part is generated and passed to the first-order part. This model acts as an component interpretation and guides the assertion of components in the first order reasoning part. More details about how the mapping from first-order clauses to propositional clauses is done and how components are used can be found in [5, 3].

As the majority of SAT solvers, Lingeling allows the users to add clauses in an incremental manner, in the case of AVATAR this kind of behavior is desired. We will now briefly describe how Lingeling was integrated inside Vampire's AVATAR architecture and how we can add clauses to the solver. We devised two ways of adding the propositional clauses to the SAT solver. First, we call it *"almost" incremental*, takes the set of propositional clauses generated by the first-order part and adds them to the SAT solver. After adding all of the clauses it checks whether the problem is satisfiable or not. If the propositional problem proves to be satisfiable it returns the model generated by the SAT solver. Whereas if the problem proves to be unsatisfiable then unsatisfiability result is sent to the first-order reasoning part. The other proposed way of adding clauses to the SAT solver, called *incremental*, iterates over the set of generated propositional clauses and after adding each of the clauses to the SAT solver calls for satisfiability check and decides whether to report satisfiability (return a model) or to return unsatisfiability of the propositional problem. When integrating the SAT solver in Vampire we added some options to Vampire so that we can easily control the underlying SAT solver behavior. Details about these options and how they can be combined can be found in [3].

All together there are five new options that control the SAT solver behavior implemented in Vampie. Using combinations of these options and default Vampire options we evaluated the integration on a subset of problems coming from the TPTP library [4]. In order to evaluate performance we considered 300 problems from the CASC 2013 first-order division and another set of 6637 problems selected from the TPTP library. The 6637 problems where selected based on their ranking. That is, we selected only problems that had rank greater than .2 and less or equal than 1. Ranking .2 means that 80% of state of the art theorem provers can solve this problem, while ranking 1 means that no state of the art solver can solve this problem with its default strategy (it could happen that by using a solver in portfolio mode the problem can be solved). On these sets of problems we ran Vampire with all the implemented options both with default values for built-in options of Vampire and in competition mode. Vampire in competition mode is a special mode of running Vampire which attempts to solve a problem by launching a cocktail of strategies with very little time limit. Our experiments where conducted on the Infragrid [1] infrastructure and each run was limited to 60 seconds and a 2GB memory limit. A summary of the results obtained by running these strategies on the 300 CASC problems and running Vampire in competition mode can be found in Table 1.

The table is structured as follows, on first row the abbreviations for all the used strategies

Strategy	Vamp	L	L S	L I	L I S
Average Time	3.4679	3.0615	4.2701	2.8139	3.7852
# of solved instances	230	233	240	232	232
# different	1	8	13	8	7

Table 1: Results of running Vampire using a cocktail of strategies on the 300 CASC problems.

are presented. Second row presents the average time used by each of the strategies in order to solve the problems. Here we take into account only the time spent on the problems that can be solved using a particular strategy. Third row presents the total number of problems solved by each strategy where last row presents the number of different problems. By different problems we mean problems that could be solved either by Vampire with the default SAT solver and not solved by any of the strategies involving Lingeling and those problems that can be solved only by at least one strategy that involves Lingeling but cannot be solved by Vampire using the default SAT solver.

The abbreviations that appear in the header of Table 1 stand for the following: *vamp* stands for Vampire using the default SAT solver, L stands for Vampire using Lingeling as background SAT solver, in an "almost" incremental way, $L\ S$ similar to L but turning the generation of similar models on the SAT solver side on, $L\ I$ stands for Vampire using Lingeling as background SAT solver in pure incremental way and $L\ I\ S$ is similar to $L\ I$ but with the change that it turns similar model generation on the SAT solver side. Each of these options is better documented in [3].

Our experiments show that in general using a state of the art SAT solver for guiding the splitting process in AVATAR proves to perform better with regard to number of solved problems. Although this is the trend there are still situations where using the default SAT solver proves to behave best. We believe that further refinements on the SAT solver part and better fine tuning of the SAT solver alongside with better collaboration with the first-order reasoning part will produce even better results. We are currently investigating different ways of combining the Vampire built-in SAT solver with external ones such that we can obtain better performance by using the default solver for small problems and deploy the external solver on harder instances. Besides splitting, Vampire uses a SAT solver also for instance generation and indexing. We are therefore interested in finding out whether the use of a state of the art SAT solver improves the overall performance of Vampire.

References

[1] Infragrid. http://hpc.uvt.ro/infrastructure/infragrid/.

[2] Armin Biere. Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling. In *POS-14. Fifth Pragmatics of SAT workshop, a workshop of the SAT 2014 conference, part of FLoC 2014 during the Vienna Summer of Logic, July 13, 2014, Vienna, Austria*, page 88, 2014.

[3] Armin Biere, Ioan Dragan, Laura Kovács, and Andrei Voronkov. Experimenting with SAT solvers in vampire. In *Human-Inspired Computing and Its Applications - 13th Mexican International Conference on Artificial Intelligence, MICAI 2014, Tuxtla Gutiérrez, Mexico, November 16-22, 2014. Proceedings, Part I*, pages 431–442, 2014.

[4] G. Sutcliffe. The tptp problem library and associated infrastructure: The fof and cnf parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[5] Andrei Voronkov. AVATAR: the architecture for first-order theorem provers. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 696–710, 2014.