TECHNISCHE
UNIVERSITÄT
DRESDEN

Faculty of Computer Science Institute of Artificial Intelligence Knowledge Representation and Reasoning

Solver submission of riss 1.0 to the SAT Competition 2011

Norbert Manthey

KRR Report 11-01

Mail to
Technische Universität Dresden
01062 Dresden

Bulk mail to
Technische Universität Dresden
Helmholtzstr. 10
01069 Dresden

Office
Room 2006
Nöthnitzer Straße 46
01187 Dresden

Internet
http://www.wv.inf.tu-dresden.de

DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

Solver submission of *riss 1.0* to the SAT Competition 2011

Norbert Manthey

ICCL – International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
`norbert@janeway.inf.tu-dresden.de`

Abstract. In this note the configurations of *riss 1.0* that have been submitted to the SAT Competition 2011 are described. The SAT solver is component based and is able to enable most of the recently developed techniques in SAT solving and preprocessing a formula. Furthermore, two parallelizations of the algorithm can be used.

1 The SAT solver *riss 1.0*

Originally, *riss 1.0* has been implemented to analyze the resource utilization of SAT solvers. Furthermore, it should be possible to extend the solver easily. Therefore, a component based scheme has been chosen that separates the CDCL algorithm [16] into the components: unit propagation, decision heuristic, conflict analysis, restart scheduling and removal. Additionally, a preprocessor has been integrated to be able to simplify the formula after top level units have been found. Each of this components has lots of parameters so that it is hard to find the best combination. It can been shown that different configurations perform differently well for several timeouts. Most of the recent parallel solvers execute several configurations of a SAT solver in parallel and share learned clauses among these configurations [7]. This parallelization is also available in *riss 1.0*. The solver is submitted in sequential and parallel configurations to the competition.

Basic Configuration The standard version of *riss 1.0* implements the Two-Watched-Literal scheme without blocking literals, but with prefetching [9] the clauses of the current watch list. Each new implied literal is propagated on binary clauses before it is propagated on longer clauses, since the binary clauses are represented implicitly in their own watch list and are also stored as reason clauses in this way. Thus, the conflict analysis does not need to access the binary clause explicitly. After the conflict clause has been analyzed and a 1st UIP clause [19] has been learned, this clause is minimized further [18]. Decision are based on the VSIDS heuristic and polarity caching [15] is used to determine the polarity for the decision variable.

Besides the major components, *riss 1.0* implements various event schedulers. For scheduling restarts it uses a nested geometric series starting with 150 and using an increment factor of 1.3. Learned clause removal is scheduled using a geometric series starting with 1000 and with an increase value of 1.5. The removal uses clause activities that are based on the LBD [3]. This activity is assigned only once the clause is learned and stays static. Binary clauses are not removed. Half of the remaining clauses are deleted.

The preprocessor of *riss 1.0* implements a non increasing variable elimination (VE) [6] together with blocked clause elimination (BCE) [10], equivalence elimination (EE), hidden tautology elimination (HTE) [8], vivification (VI) [14] and extended resolution (ER) [1]. These techniques are applied in the following order: EE + HTE + BCE + VE + VI + ER + EE + HTE + BCE. Furthermore, the solver is able to run any of these techniques again during a restart. Currently, this option is not enabled since it introduces some overhead due to the component based implementation of the solver.

1.1 Sequential Configurations

As for the usual use case, a static configuration has been submitted, that is the currently best known configuration of *riss 1.0*. This configuration uses the basic configuration. The second sequential configuration *rissAuto* sets up a configuration according to the timeout for an instance. Therefore, it enables the best known configuration for the specified time. The third sequential solver *rissExp* uses an experimental configuration.

rissAuto The specified timeouts for the two stages of the competition are given in advance. Therefore, *rissAuto* chooses its configuration based on the timeout parameter. For 5000 seconds the basic configuration is chosen. For the first stage timeout (1200 seconds) the solver applies a probing step [11] whenever it decides a variable on the root of the search tree. During this step, the decision literal l is propagated in both polarities. If propagating one of the polarities fails, a new unit has been found. Furthermore, the implied literals of both polarities are compared. If both polarities imply literal *l'*, *l'* can also be set as a top level unit.

rissExp During the SAT Race 2010 the winner CryptoMiniSAT [17] was able to extract encoded XOR-functions from the formula. The configuration *rissExp* extracts XOR clauses and uses special structures instead of these clauses. The extracted XOR is also used during conflict analysis, such that only a single clause needs to be stored that represents an XOR. Furthermore, the propagation of longer clauses uses the blocking literal scheme [13]. Additionally, if a binary conflict is found, the propagation is continued until a longer conflict clause is recognized. If there is no such a longer conflict clause, the last binary conflict is selected as in PrecoSAT [4]. Restarts are scheduled dynamically following [2].

1.2 Parallel Configurations

Since modern CPUs will get more cores, the parallelization of SAT solvers is important [5]. Modern SAT solvers run several configurations and share learned clauses among these configurations [7]. The submitted version of *riss 1.0* is furthermore able to parallelize unit propagation [12] of a sequential solver.

The parallelization of *riss 1.0* is quite experimental, such that the chosen combination of configurations is not well studied yet. However, big series of experiments have been used to discover well performing sequential configurations that are able to solve as many instances as possible within a certain timeout if they are executed in parallel without sharing clauses. Based on this knowledge, the first two parallel configurations have been chosen. The third parallel configuration uses the parallelized unit propagation.

Finding a good clause sharing algorithm is hard. Usually, the length of learned clauses increases during solving an instance. To prevent sharing too long clauses, a thread sends only clauses that are shorter than the minimum of its learned clauses since the last restart. Additionally, only clauses are received that are shorter than the average of the learned clause length since the last restart of the receiving thread. Using this mechanism, shared clauses help to reduce the average size of learned clauses.

PrissFix The parallel configuration *PrissFix* uses a fixed setting that uses 4 cores. It follows the portfolio approach as for example ManySAT [7]. Differently to ManySAT, the basic configuration of *riss 1.0* is part of the parallel version and is run without receiving clauses from the other configurations. Thus, *PrissFix* should be at least as powerful as the basic configuration, because the search path of this configuration is not changed, if the overhead of communication and sharing the memory architecture is neglected. Due to clause sharing among the remaining configurations, the solvers performance might be even higher.

PrissAuto A drawback of the fixed configuration *PrissFix* is that it is not able to use an arbitrary number of cores. Thus, *PrissAuto* is able to enable up to 8 configurations that are executed in parallel and share learned clauses among each other. Again, the first configuration is the basic configuration and it also receives clauses. The remaining configurations have been chosen out of many experiments on the last SAT competitions application benchmark such that the combination of the configurations is able to solve as many instances as possible in a specified timeout.

PrissUP The parallelization of the unit propagation is able to use an arbitrary number of threads. Differently to the basic configuration, *PrissUP* does not distinguish between binary clauses and other clauses. The formula is regarded as a set of clauses. This set is separated among the n used threads. During propagation, each thread T propagates the literal of its propagation queue using its partition of the formula. In case of a conflict, the propagation of all threads is stopped. Otherwise, if T reaches a fix point, it synchronizes its propagation queue with the queues of the other threads. The algorithm is implemented with a single lock, which is only needed for sharing the information about a conflict because multiple threads might find a conflict. Thus, the submitted version of *PrissUP* uses 2 threads because the scalability has not been analyzed yet.

Acknowledgment The author would like to thank Peter Steinke for many discussions concerning the configuration of the solvers. Additional thanks go to Marijn Heule because of many fruitful discussions about HTE and other preprocessing techniques.

References

1. Gilles Audemard, George Katsirelos, and Laurent Simon. A restriction of extended resolution for clause learning sat solvers. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.
2. Gilles Audemard and Laurent Simon. Glucose: a solver that predicts learnt clauses quality. SAT 2009 Competitive Event Booklet, `http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf`, 2009.
3. Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solver. In *Twenty-first International Joint Conference on Artificial Intelligence(IJCAI'09)*, pages 399–404, jul 2009.
4. Armin Biere. PrecoSAT system description. `http://fmv.jku.at/precosat/preicosat-sc09.pdf`, 2009.
5. Intel Corporation. Intels Teraflops Research Chip. `http://download.intel.com/pressroom/kits/Teraflops/Teraflops_Research_Chip_Overview.pdf`, 2010.
6. Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *In proc. SAT05, volume 3569 of LNCS*, pages 61–75. Springer, 2005.
7. Long Guo, Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Diversification and intensification in parallel sat solving. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, pages 252–265, Berlin, Heidelberg, 2010. Springer-Verlag.
8. Marijn Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for cnf formulas. In Christian Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 357–371. Springer Berlin / Heidelberg, 2010.
9. Steffen Hölldobler, Norbert Manthey, and Ari Saptawijaya. Improving resource-unaware sat solvers. In Christian Fermüller and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *Lecture Notes in Computer Science*, pages 357–371. Springer Berlin / Heidelberg, 2010.
10. Matti Järvisalo, Armin Biere, and Marijn Heule. Blocked clause elimination. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin / Heidelberg, 2010.
11. I. Lynce and J. P. Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. *IEEE International Conference on Tools with Artificial Intelligence*, 2003.
12. Norbert Manthey. Parallelizing the heart of a sat solver, 2010. Submitted to SAT 2011.
13. Niklas Sörensson. Minisat 2.2 and minisat++ 1.1. `http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_25+26.pdf`, 2010.
14. Cédric Piette, Youssef Hamadi, and Lakhdar Sas. Vivifying propositional clausal formulae.

15. Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of 10th International Conference on Theory and Applications of Satisfiability Testing(SAT)*, pages 294–299, 2007.
16. João P. Marques Silva and Karem A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, ICCAD '96, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.
17. Mate Soos. Cryptominisat 2.5.0. In *SAT Race competitive event booklet*, July 2010.
18. Niklas Sörensson and Armin Biere. Minimizing learned clauses. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 237–243, Berlin, Heidelberg, 2009. Springer-Verlag.
19. Lintao Zhang, Conor F. Madigan, and Matthew H. Moskewicz. Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.