Solver Description of RISS 2.0 and PRISS 2.0

Norbert Manthey
Knowledge Representation and Reasoning Group
Technische Universität Dresden, 01062 Dresden, Germany
norbert@janeway.inf.tu-dresden.de

***Abstract*—The SAT solver RISS 2.0 and its concurrent parallelization PRISS 2.0 are described in the configuration they have been submitted to the SAT Challenge 2012.**

I. The sequential SAT solver RISS 2.0

Based on the CDCL procedure, this solver has been implemented as a module based system. The routines for the decision procedure, the learned clause management, unit propagation, preprocessor and the event heuristics for restart and removal can be exchanged easily. This style of implementation comes to a cost, namely the communication overhead among the components. Whereas plain SAT solver implementations can alter for example the watched list of the unit propagation immediately when a clause should be removed, RISS 2.0 has to store this data first, pass it to the unit propagation module and afterwards this module can execute the wanted operation. Based on this overhead, the implementation is a trade-off between providing as many features as possible and having a good performance on application instances. To still achieve a high performance, RISS 2.0 is equipped with a strong preprocessor COPROCESSOR 2.1 [13], which is also be used during search to simplify the formula and the set of learned clauses.[1]

A. Features of RISS 2.0

The main goal if RISS 2.0 is to solve formulas in CNF. Furthermore, the solver is used as research platform and thus provides many parameters to enable further techniques. These techniques are not present in general SAT solvers:

- Enumeration of all solutions of the input formula
- Loading and storing learned clauses of a run
- Searching for a solution with a set of assumed literals
- Passing an initial model to the solver that should be tested first

Additionally to the named features, RISS 2.0 implements many deduction techniques on top of CDCL that can be enabled. Among them there are *On-the-fly Self-Subsumption* (OTFSS) [7], *Lazy Hyper-Binary-Resolution*(LHBR) [3] and *Dominator Analysis* [6]. To speed up search, most of the techniques that are available in COPROCESSOR 2.1 can be used for simplifying the formula during search. The implementation and handling of data structures and memory accesses is based on the insights that have been published in [10]. The solver furthermore uses *Blocking Literals* introduced in [16] and *Implicit Binary Clauses*(e.g. [15]) to speed up unit propagation and conflict analysis.

The submitted configuration uses the Luby series with a factor 32 as a restart strategy and a geometric series starting with 3000 and an increment factor of 1.1 as removal schedule. The removal is mainly based on the LBD measure [2], but also short clauses are kept. Both OTFSS and LHBR are enabled.

We started to implemented RISS from scratch in 2009 as a teaching system in C++. The binary of the tool we provided for the SAT Challenge has been compiled with the GNU compiler and the optimization -O3. Although plenty of parameters are implemented in both RISS 2.0 and its preprocessor automated parameter setting has not been done yet. This is considered the next step, because parameter setup is not considered to be trivial but has high potential to improve the solvers performance.

B. Features of Coprocessor

The internal preprocessor of RISS 2.0 implements many simplification techniques, that are executed in the specified order. Whenever a technique can reduce the formula, the process is started from the top.

1) Unit propagation
2) Pure literal detection
3) Self-subsuming resolution
4) Equivalence elimination [5]
5) Unhiding [9]
6) Hidden tautology elimination [8]
7) Blocked clause elimination [11]
8) Variable elimination [4]
9) An algorithm based on extended resolution
10) Failed literal probing [12]
11) Clause vivification [14]

Equivalent literal detection is done based on binary clauses and on output literals of gates in the formula. The algorithm based on extended resolution to simplify the formula is unpublished, but submitted for publication. Each technique can be limited so that the consumed run time remains reasonable. After preprocessing, COPROCESSOR allows to shrink the formula so that all assigned or eliminated variables are removed and the resulting formula contains consecutive variables again.

II. The parallel SAT solver PRISS 2.0

The SAT solver PRISS 2.0 is a portfolio SAT solver based on RISS 2.0 and supports up to 64 parallel solver incarnations. After using COPROCESSOR on the input formula, n incarnations

[1]Both the solver and its preprocessor as well as descriptions are available at tools.computational-logic.org.

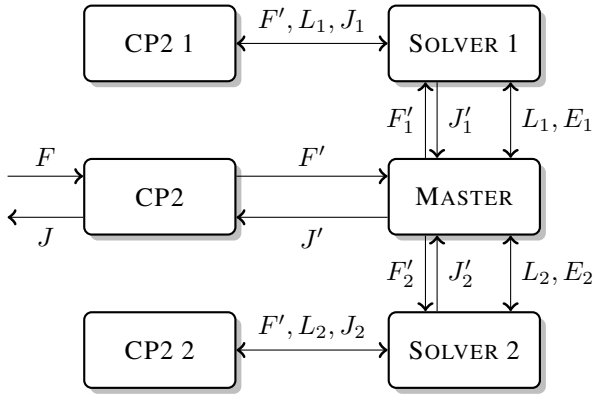

Fig. 1. Components in the PRISS 2.0 framework

of RISS are started concurrently, where each of the incarnations uses its own preprocessor to simplify the formula during search. Learned clauses are shared among the incarnations. The exchange is filtered both on the sender and the receivers side. Submitting clauses is based on the length of the clause and its activity. Whenever the length of a candidate clause is shorter than the average length since the last restart, the clause is a candidate to be submitted to the shared storage. Another criterion is the activity based on the LBD. The reception of clauses from the storage is based on the same criteria again. Furthermore, the PSM [1] is used to reject not useful clauses. In addition to clauses, the RISS incarnations share informations about equivalent literals, which are found during search by the simplification methods. Since the simplification might also add or remove variables from the formula of a certain thread, only information about common variables is shared – a clause that contains an eliminated variable will be rejected by the receiving thread. Based on the current portfolio implementation, this problem cannot be fixed easily. For the future it is wanted to integrate the common preprocessor also as common simplifier, so that all clauses can be shared again.

Figure 1 shows the a pictogram of the components and their communication. After the input formula F is processed by the preprocessor, each solver incarnations is started in a thread with a physical copy of the formula (F_1' and F_2'). For inprocessing each solver has its private preprocessor. Learned clauses and equivalent literals are shared with the master (e.g. L_1 and E_1). When a solver finds a solution, its preprocessor reconstructs eliminated variables, equivalent variables and literals from blocked clauses. The processed model is passed back to the master, which stops all other solver incarnation and also reconstructs the final assignment.

The submitted configuration of the solver uses only 5 cores out of the 8 available cores. Each incarnation has a slightly different configuration. The first incarnation uses the default configuration. The next solver uses *permuted trail* restarts [17]. The third incarnation keeps 50 % of its learned clause data based instead of 25 %. The fourth solver uses the PSM value for removing clauses and bumps variables twice, if they are used during conflict analysis, are assigned at the conflict level and if the activity of their reason clause is comparably high. Finally, the fifth configuration exchanges the VSIDS heuristic by the VMTF heuristic for variable activities. If more cores should be used, the next configuration alters the implementation of the unit propagation by preferring satisfied literals in clauses to be watched. All further configurations are similar to the default configuration except the fact that one percent of their decisions is done randomly to not result in the same search.

REFERENCES

[1] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, and Lakhdar Saïs. On freezing and reactivating learnt clauses. In *Proceedings of the 14th international conference on Theory and application of satisfiability testing*, SAT'11, pages 188–200, Berlin, Heidelberg, 2011. Springer-Verlag.

[2] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[3] Armin Biere. Lazy hyper binary resolution. In *Algorithms and Applications for Next Generation SAT Solvers*, number 09461, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

[4] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *In proc. SAT'05, volume 3569 of LNCS*, pages 61–75. Springer, 2005.

[5] Allen Van Gelder. Toward leaner binary-clause reasoning in a satisfiability solver. *Ann. Math. Artif. Intell.*, 43(1):239–253, 2005.

[6] HyoJung Han, HoonSang Jin, and Fabio Somenzi. Clause simplification through dominator analysis. In *DATE*, pages 143–148. IEEE, 2011.

[7] Hyojung Han and Fabio Somenzi. On-the-fly clause improvement. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 209–222, Berlin, Heidelberg, 2009. 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 *LNCS*, pages 357–371. Springer, 2010.

[9] Marijn Heule, Matti Järvisalo, and Armin Biere. Efficient CNF Simplification based on Binary Implication Graphs. In K.A. Sakallah and L. Simon, editors, *SAT 2011*, volume 6695 of *LNCS*, page 201–215. Springer, 2011.

[10] Steffen Hölldobler, Norbert Manthey, and Ari Saptawijaya. Improving resource-unaware sat solvers. In Christian G. Fermüller and Andrei Voronkov, editors, *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 519–534. Springer, 2010.

[11] 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 *LNCS*, pages 129–144. Springer, 2010.

[12] Inês Lynce and João Marques-Silva. Probing-Based Preprocessing Techniques for Propositional Satisfiability. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '03, pages 105–. IEEE Computer Society, 2003.

[13] Norbert Manthey. Coprocessor 2.0 – A flexible CNF Simplifier (Tool Presentation), 2012. Submitted to SAT 2012.

[14] Cédric Piette, Youssef Hamadi, and Lakhdar Saïs. Vivifying propositional clausal formulae. In *18th European Conference on Artificial Intelligence(ECAI'08)*, pages 525–529, Patras (Greece), jul 2008.

[15] Mate Soos. Cryptominisat 2.5.0. In *SAT Race competitive event booklet*, July 2010.

[16] Niklas Sörensson and Niklas Eén. MiniSat 2.1 and MiniSat++ 1.0 — SAT Race 2008 Editions. Technical report, 2008.

[17] Peter van der Tak, Antonio Ramos, and Marijn J.H. Heule. Reusing the assignment trail in cdcl solvers. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:133–138, 2011. system description.