TECHNISCHE
UNIVERSITÄT
DRESDEN

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

riss 2010 Solver Description

Norbert Manthey

KRR Report 10-02

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

riss 2010 Solver Description

Norbert Manthey

International Center for Computational Logic,
TU Dresden, 01062 Dresden, Germany
Norbert@janeway.inf.tu-dresden.de

Abstract. The SAT solver riss has been developed for analyzing the resource utilization of state of the art SAT solvers. It introduces new techniques like the Slab Allocator and Prefetching.

1 The SAT Solver riss

The SAT solver riss is CDCL based. It has been implemented during a student project to analyze the memory hierarchy utilization. It uses a cache aware implementation and data structure improvements described in [9].

The 32bit C++implementation is based on HydraSAT [2], which has been submitted to the SAT Competition 2009. Since riss is a re-implemented (except conflict analysis and preprocessor) of HydraSAT, which is close to the MiniSAT 1.4 [6] implementation, the basic riss version is also comparable to MiniSAT. The components are exchangeable at runtime and can by loaded from libraries. The component system uses C++templates.

2 Data Structures

The basic data structures are taken from the Standard Template Library. Clauses are implemented according to the clause packing schema in [4] using 5 local literals. They are allocated using a Slab Allocator [3]. The watched list of the two-watched-literal schema are implemented using a vector. Removing elements from them is done lazily.

3 Solver Components

Riss implements a conflict driven decision learning search. Conflict analysis is done using the first UIP scheme. Afterwards self subsumption is applied to minimize the resulting clause (learnt clause) even further. Before the search a preprocessor,similar to the Satellite preprocessor [5], is applied.

Unit propagation treats binary clauses specially. First the assignment is applied to binary clauses and all their implications, next longer clauses are considered.

Propagating longer clauses is improved by prefetching the clause headers. Additionally, blocking literals [11] are used. Like SApperloT [7], a probing [8] step is applied, if a literal is propagated on the first level of the search tree. The obtained variable equivalence informations is not used at the moment, but found units are propagated.

Decisions are done using phase saving as it has been introduced in [10]. The phases are stored during backjumping in the search tree for all assignments that are undone. The phase information is not reset. Random decisions are made with a probability of 2%.

Scheduling restarts uses the luby schema with 32 as factor. Scheduling clause removal is done using a geometric series with base 1000 and 4/3 as increase factor. The the clause removal is activity based. Clauses of size two and clauses with an activity higher than a certain threshold are kept. The activity of learnt clauses is inversely proportional to the number of tree levels that occur in this clause [1]. Learnt clauses that are added exactly before a restart are never removed.

Acknowledgment I want to thank Max Seelemann and Friedrich Gräter, the programmer of the first HydraSAT version, for implementing the conflict analysis and the conflict minimization and Christoph Baldow, who implemented the HydraSAT preprocessor.

References

1. G. Audemard and L. Simon. GLUCOSE: a solver that predicts learnt clauses quality. SAT 2009 Competitive Event Booklet, http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.
2. C. Baldow, F. Gräter, S. Hölldobler, N. Manthey, M. Seelemann, P. Steinke, C. Wernhard, K. Winkler, and E. Zenker. HydraSAT 2009.3 solver description. SAT 2009 Competitive Event Booklet, http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.
3. J. Bonwick. The slab allocator: an object-caching kernel memory allocator. In *Proceedings of the USENIX Summer 1994 Technical Conference*, 1994.
4. G. Chu, A. Harwood, and P. J. Stuckey. Cache conscious data structures for Boolean satisfiability solvers. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:99–120, 2009.
5. N. Eeen and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005*, 2005.
6. N. Eén and N. Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
7. S. Kottler. SApperloT,Description of two solver versions submitted for the SAT-competition 2009. SAT 2009 Competitive Event Booklet, http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.
8. I. Lynce and J. Marques-Silva. Probing-based preprocessing techniques for propositional satisfiability. In *ICTAI'03*, 2003.
9. N. Manthey and A. Saptawijaya. Towards improving the resource usage of sat-solvers. In *submitted to the Pragmatics of SAT Workshop 2010*, 2010.
10. K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In J. Marques-Silva and K. A. Sakallah, editors, *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.
11. N. Sorensson and N. Een. MINISAT 2.1 and MINISAT++ 1.0 SAT Race 2008 Editions. SAT 2009 Competitive Event Booklet, http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.