Riss 6 Solver and Derivatives

Norbert Manthey, Aaron Stephan and Elias Werner
Knowledge Representation and Reasoning Group
TU Dresden, Germany

***Abstract*—The sequential SAT solver RISS combines the Minisat-style solving engine of GLUCOSE 2.2 with a state-of-the-art preprocessor COPROCESSOR and adds many modifications to the search process. RISS allows to use inprocessing based on COPROCESSOR. Based on this RISS, we create a parallel portfolio solver PRISS, which allows clause sharing among the incarnations, as well as sharing information about equivalent literals.**

I. INTRODUCTION

The CDCL solver RISS is a highly configurable SAT solver based on MINISAT [1] and GLUCOSE 2.2 [2], [3]. Many search algorithm extensions have been added, and RISS is equipped with the preprocessor COPROCESSOR [4]. Finally, RISS supports automated configuration selection based on CNF formulas features, emitting DRAT proofs for many techniques, and incremental solving. The parallel solver PRISS is a portfolio solver that allows to use COPROCESSOR for formula simplification commonly for all solver incarnations, provides inprocessing for all solver incarnations, and can handle sharing of sets of equivalent literals.

This document mentions only the differences to RISS 5.05 that has been submitted to SAT Race 2015. Most differences come from bug fixes, where the bugs have been found with SPYBUG [5], and from disabling techniques that cannot print (short) DRAT proofs for their reasoning.

II. MODIFICATIONS OF THE SEARCH

RISS 5.05 used a vivification based learned clause minimization. As it turned out to be ineffective for formulas of more recent years, this minimization is disabled by default. Furthermore, the input formula is scanned for being easily solvable by assigning all variables $\top$ or assigning all variables $\bot$. Otherwise, both the activity and the polarity information to perform search decisions are pre-initialized: the activity per variable decreases linearly, and the initial polarity is set according to the Jeroslow-Wang score.

III. MODIFICATIONS OF COPROCESSOR

To be able to emit DRAT proofs many simplification techniques of Coprocessor had to be disabled, among them reasoning with XORs and cardinality constraints [6]. However, we added lazy hyper binary resolution, as used in ABCDSAT, to the portfolio of simplification techniques.

IV. CONFIGURATION SELECTION

The machine learning front end is based on the feature extraction routines implemented in RISS [7]. Compared to RISS 5.05 we included only configurations that allow to print DRAT proofs. The used configurations have been picked by hand, or have been produced by running SMAC [8] on sets of formulas that are known to be challenging for RISS.

The knowledge base for prediction is integrated into the solver. From the features and measured times that are available, we use the *information gain ratio* to select improtant features and remove redundancy. Next, with *principal component analysis* we further reduce the dimension of the feature space, where during training the information loss has to be small. Additionally more redundancy in form of correlation between the features is removed. Based on the features, we perform a k-nearest neighbor search.

The implementation of PCA is based on two external libraries: *Libpca*[1] and *armadillo*[2].

V. INCREMENTAL SAT SOLVING WITH RISS

The two configurations that have been submitted for the incremental track: RISS 5.21 uses formula simplification only during the first incremental call, while RISS 6 uses inprocessing with very long intermediate intervals, starting after the first 5000 conflicts.

VI. SAT COMPETITION SPECIFICS

RISS and COPROCESSOR are implemented in C++. The DS version of RISS implements watch lists based on an own allocator – as also done in LINGELING – so that garbage collection on watch lists is possible. PRISS implements the parallel code for multi-core architectures with the pthreads library.

The solvers are submitted to all tracks that are offered, except the *random SAT* track.

VII. AVAILABILITY

All tools in the solver collection are available for research. The source of the solver will be made publicly available under the LGPL v2 license after the competition at http://tools.computational-logic.org.

[1] http://sourceforge.net/projects/libpca/
[2] http://arma.sourceforge.net/

ACKNOWLEDGMENT

The author would like to thank the developers of GLUCOSE 2.2 and MINISAT 2.2. The computational resources to develop, evaluate and configure the SAT solver have been provided by the ZIH of TU Dresden. This project is supported by the DFG grant HO 1294/11-1.

REFERENCES

[1] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT 2003*, ser. LNCS, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Heidelberg: Springer, 2004, pp. 502–518.

[2] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI 2009*, C. Boutilier, Ed. Pasadena: Morgan Kaufmann Publishers Inc., 2009, pp. 399–404.

[3] ——, "Refining restarts strategies for sat and unsat," in *CP'12*, 2012, pp. 118–126.

[4] N. Manthey, "Coprocessor 2.0 – a flexible CNF simplifier," in *SAT 2012*, ser. LNCS, A. Cimatti and R. Sebastiani, Eds., vol. 7317. Heidelberg: Springer, 2012, pp. 436–441.

[5] N. Manthey and M. Lindauer, "Spybug: Automated bug detection in the configuration space of SAT solvers," in *SAT*, 2014, accepted.

[6] A. Biere, D. Le Berre, E. Lonca, and N. Manthey, "Detecting cardinality constraints in CNF," in *Theory and Applications of Satisfiability Testing – SAT 2014*, ser. Lecture Notes in Computer Science, C. Sinz and U. Egly, Eds., vol. 8561. Springer International Publishing, 2014, pp. 285–301. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-09284-3_22

[7] E. Alfonso and N. Manthey, "New CNF features and formula classification," in *POS-14*, ser. EPiC Series, D. L. Berre, Ed., vol. 27. EasyChair, 2014, pp. 57–71.

[8] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *LION*, ser. LNCS, 2011, vol. 6683, pp. 507–523.