Riss Solver Framework v5.05

Lucas Kahlert, Franziska Krüger, Norbert Manthey and Aaron Stephan
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. Finally, the search space partitioning SAT solver PCASSO is built on top of PRISS, allowing to solve a formula with a portfolio solver, and additionally solve this formula with iterative partitioning where each partition is solved with the portfolio solver PRISS, including clause sharing over partitions.**

I. INTRODUCTION

The CDCL solver RISS was first build on the MINISAT search engine [1], and next incorporated the improvements that have been proposed for GLUCOSE 2.2 [2], [3]. Afterwards, more search algorithm extensions have been added, and RISS is equipped with the preprocessor COPROCESSOR [4], that implements most of the recently published formula simplification techniques, ready to be used as inprocessing as well by taking care of learned clauses. The parallel simplification techniques of COPROCESSOR are disables in the used configurations. The aim of the solver is to provide a huge set of options on the one hand, to be able to adapt to new SAT applications, and an efficient implementation on the other hand. Due to the huge configuration space, to increase robustness an algorithm for checking the reported satisfying assignment, as well as an on-the-fly DRAT proof verification are added to the sequential solver – as the combination of all parameter configurations cannot be tested extensively.

Similarly, we build the parallel solver PRISS and PCASSO, which again are highly configurable. In PRISS, each incarnation of RISS can be configured, and furthermore the properties of information sharing can be set. PRISS first runs a global formula simplification step, before search is started in parallel. Each solver incarnation can simplify the formula again, where currently only equivalence preserving techniques are enabled. An incarnation can also work on the original formula.

Finally, PCASSO can RISS or PRISS as base solver. PCASSO solves a formula based on iterative partitioning. Furthermore, an additional PRISS incarnation is added to solve the given formula. Similarly to the other two solvers, PCASSO first simplifies the formula with a COPROCESSOR incarnation before starting parallel search.

The incremental solver interface is currently only supported by RISS. PRISS supports a similar interface, and can be used to use a parallel portfolio solver as incremental solver.

The remainder of the document describes only the changes that have been applied to the tools compared to their previous versions of 2014 [5], [6], [7].

II. MODIFICATIONS OF RISS AND COPROCESSOR

RISS 5.05 is an extension of RISS 4.27, where a few techniques have been removed to clean the code base, which has been executed during search. On the other hand, many modification have been added, among others ideas that have been proposed in recent MINISAT-Hack-Tracks. The considered extensions come from the solvers RESTARTSAT, CIRMINISAT, MINITSAT, MIPISAT and MINISAT_HACK_999ED. All novel features are turned off by default, such that the default behavior of the solver does not change too much when modifying the implementation. The initialization of the solver has been improved: the activities for decision variables are initialized with decreasing values.

A. *Conflict Analysis and Minimization*

Compared to GLUCOSE 2.2, the LBD of the currently constructed learnt clause is only re-calculated if a minimization technique modified the clause. Finally, RISS implements a modified variables on *restricted extended-resolution* [8], [9]. This variant also allows to structurally extract binary AND-gates from the formula, to later use this information to reduce learned clauses by replacing both input literals with the output literal of such a gate. Finally, another unpublished additional clause minimization technique is added to conflict analysis. Removing learnt clauses can be done as in GLUCOSE 2.2 based on the LBD of the clause, and an dynamic schedule. However, the clause activity based version and the more static schedule of MINISAT 2.2 are also supported, including the ideas of the randomly assigned activity from [10].

B. *Unit Propagation*

The implementation of unit propagation has also been improved: Instead of storing binary clauses in an extra watch list, as implemented in GLUCOSE 2.2, we store all clauses in a common watch list. However, each list entry stores a Boolean flag which indicated whether the current clause is binary, such that we just have to work with the *blocking literal*, and do not need to access the actual clause. By using only one watch list, the used memory can be reduced compared to GLUCOSE 2.2.

C. *Coprocessor*

COPROCESSOR received only little attention. Small changes have been applied to the execution order of the simplification techniques, which now all process least occurring literals first (except *bounded variable addition* [11]). Furthermore, COPROCESSOR now runs all activated simplification techniques twice in the specified order.

We added an implementation of an elimination routine to remove *resolution asymmetric tautologies* (RAT) [12]. Based on this routine, and the idea of *covered literal elimination* [13][1], we implemented routines to add a redundant clause, if it subsumes another clause that is present in the formula. Hence, COPROCESSOR allows to perform *vivification* by adding *asymmetric tautologies* that subsume other clauses. Similarly, COPROCESSOR can add *blocked clauses*, *covered clauses*, or *RATs*.

The implemented extraction of XORs in the formula does not use a detection based on subsumption due to its high time consumption. On the other hand, the XOR reasoning does not only use information about retrieved unit clauses and equivalent literals, but furthermore encodes ternary XORs back into CNF.

III. THE PARALLEL PORTFOLIO SOLVER PRISS

The parallel portfolio solver PRISS can simplify the formula with COPROCESSOR for all its incarnations in parallel. Afterwards, each incarnation can work on this formula, such that information sharing is easier – all clauses can be shared as long as no further simplification that do not preserve equivalence are applied. An incarnation can choose to to work on the original formula instead, but then (currently) knowledge sharing with this incarnation is disabled.

A. Information Sharing

Incarnations of RISS in PRISS can share learned clauses, as well as information about equivalent literals. PRISS has two storages that are implemented as ring buffer. The first buffer stored unit clauses and equivalent literal classes. The other buffer stores shared learned clauses. Information is received whenever an incarnation should do a search decision for decision level 1. Due to the ring buffer implementation, incarnations might miss shared clauses. On the other hand, this implementation ensures that the size of the shared buffer is limited.

Received clauses can be minimized by *vivification* and reduced clauses can be shared immediately, as proposed in [14]. Learned clauses are shared after (i) they have been learned, (ii) they have been used in unit propagation [15], or (iii) after they have been used for conflict analysis again. This way, only "relevant" learned clauses are shared.

Before a clause is shared, it has to pass a size filter and an LBD filter. These filters can be set to static thresholds. However, the thresholds can also be increased dynamically, if not enough clauses are sent. Finally, in case not all clauses are selected for sharing (the above (ii) and (iii)), then the filters might not be used, as the clauses turned out to be relevant for the current incarnation already.

[1]Developped independly also by Heule et al.

The activity of received clauses is set to the value of the most recently learnt clause. The LBD of the received clause is estimated based on the size to LBD ratio of the current incarnation, as on search level 0 there are not variable assignments and levels that can be used to determine the actual LBD of the clause. Furthermore, received clauses are ignored in the next removal iteration. All these options can be specified for each incarnation – in the submitted version the parameters are fixed as described above.

B. Configuration Selection

For each incarnation in the portfolio, the configuration can be specified on the command line. In the the used preset, the first three configurations are the most robust configurations that have been while working with application formulas. The simplification of the first incarnation is used for the full portfolio and the other two incarnations do not participate in information sharing. The first incarnation only sends information, but does not receive information. The remaining configurations are based on configurations that showed good performance and are extended with randomization, such that the portfolio works better especially on satisfiable formulas. Furthermore, *inprocessing* with equivalence preserving techniques is enabled, including replacing equivalent literals.

IV. THE SEARCH SPACE PARTITIONING SOLVER PCASSO

PCASSO implements solving formulas in parallel based on iterative search space partitioning [16]. The partitioning is the same as in last years version.

A. Solving In Parallel

Besides the iterative partitioning approach, a PRISS incarnation is executed in parallel. This incarnation is also used for the initial formula simplification. Furthermore, shared information is forwarded to the partitioning approach.

B. Solving Partitions

Compared to last years version the solver that solved the partitions is exchanged. Instead of GLUCOSE 2.2, we now use either a RISS or a PRISS incarnation. Due to the possibility of inprocessing, upward clause sharing in the partition tree is currently disabled. Learned information are still shared downward in the tree. With the PRISS incarnation, a partition can also be solved with a parallel solver. Then, the incarnations in PRISS share their learned information with the other internal incarnations, as well as with their child partitions.

V. INCREMENTAL SAT SOLVING WITH RISS

RISS implements the ipasir interface. RISS furthermore implements a similar C interface, where a configuration can be selected. The ipasir interface is a wrapper for the internal interface.

COPROCESSOR also provides an incremental interface, where formula simplification can be applied to a formula.

VI. Special Algorithms, Data Structures, and Other Features

Due to the high number of parameters Riss implements a configuration prediction.

A. Black Box

The machine learning front end is based on the feature extraction routines implemented in Riss [17]. Compared to last years version we improved the implementation of the extraction to reduce the memory consumption considerably by consuming slightly more run time.

The Knowledge base for prediction is now integrated into the solver, such that there are no dependencies to external tools any longer. From a formula we extract 382 features, which are projected to 40 values by *principal component analysis* (PCA). We replaced the selection based on *information gain ratio*, as in the latter case the projected values still correlate. After PCA there is no more correlation among the values. The data base is created based on 14 configurations and 2200 application formulas. The best prediction we found for this set uses a k-nearest neighbor with only a single neighbor. Riss 4.27 BlackBox used a random decision forest. The found values might perform well due to the small number of formulas and the comparably high number of configurations. We currently experiment on how to improve this prediction, especially for working better on unknown formulas.

The implementation of PCA is based on two external libraries: *Libpca*[2] and *armadillo*[3].

VII. SAT Race 2015 Specifics

Riss and Coprocessor are implemented in C++. All simplification techniques inside Coprocessor are implemented in separate classes to increase the structure of the code. Priss and Pcasso implement the parallel code for multi-core architectures with the pthreads library.

The solvers are submitted to all tracks that are offered. The submitted configurations are described below.

A. Sequential SAT Solving

The submitted default configuration of Riss 5.05 uses the following techniques: bounded variable elimination, cardinality extraction and Fourier-Motzkin, five iterations of reasoning on the binary implication graph, covered literal elimination, detecting and replacing equivalent literals with structural hashing of AND-gates and with strongly connected components on the binary implication graph, XOR reasoning, and variable renaming to compact the representation of the formula during search. Furthermore, the unpublished learned clause minimization is used, and the activities for decision variables is initialized as described above. Finally, during clause removal we keep 1 percent of the heuristically "worst" learned clauses. The second sequential configuration uses the *black box* prediction.

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

[3]http://arma.sourceforge.net/

B. Incremental SAT Solving

The incremental versions do not use formula simplification, but instead use methods to reduce the overhead of solver calls. Riss 5.04 removes satisfied clauses only if one of the first two literals is satisfied, similarly to Glucose 3.0. The simplification to remove satisfied clauses is executed only every 16 attempts. The variables, which are used in the assumptions are not considered to calculate the LBD of learned clauses. The activity of variables is bumped more compared to the usual configuration. Restarts are based on the matching trail restarts of RestartSAT. During a restart, Riss does not jump beyond assumptions (except every 1024th attempt). Finally, the counters of the solver are reset every 3rd call to solve an updated formula.

The incremental version of Riss 5.05 uses the same configuration and furthermore adds initializing the variable activities as in the usual configuration. Furthermore, after 50000 conflicts (dependent on counter resets), a formula simplification is triggered, which executes the same techniques as in the sequential solver configurations, except variable elimination, covered literal elimination and variable renaming. Instead, subsumption and self-subsuming resolution are executed, such that only equivalence preserving techniques are used and no variables are eliminated or renamed. After this simplification, the next simplification is allowed after another 1000000 conflicts.

C. Parallel SAT Solving

In Pcasso, we use different configurations depending on the number of available computational resources. The configuration of Pcasso uses a global Priss incarnation with 3 workers. Another 5 workers are used for solving in the partitioning approach. The values are increased to 8 workers in Priss and 24 in partitioning. The full number of cores is not used, as the slowdown based on resource sharing is too high with the currently implemented data structures, as explained in [18]. Each node in the partition tree is allowed to store 100 learned clauses, as well as 100 groups of unit clauses or equivalent literals. We do not utilize hyper threads.

The second configuration of Pcasso (BlackBox) uses the same number of workers. However, one configuration of Priss uses the predicted configuration. Furthermore, clause sharing inside the partition tree is enhanced by allowing larger ring buffers to store more clauses, namely 16000 per partition. The incarnations of Riss use the same inprocessing setup as the solver that is used in version 5.05 in the incremental track (see Section VII-B).

VIII. Availability

All tools in the solver collection are available for research. Besides the above mentioned tools, we furthermore provide a preliminary parallel DRAT proof verifier, as well as the model checker ShiftBMC in this package. The framework can be downloaded from http://tools.computational-logic.org.

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 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] ——, "Riss 4.27," ser. Department of Computer Science Series of Publications B, A. Belov, D. Diepold, M. J. Heule, and M. Järvisalo, Eds., vol. B-2014-2. University of Helsinki, Helsinki, Finland, 2014, pp. 65–67.

[6] E. M. Alfonso and N. Manthey, "Riss 4.27 BlackBox," ser. Department of Computer Science Series of Publications B, A. Belov, D. Diepold, M. J. Heule, and M. Järvisalo, Eds., vol. B-2014-2. University of Helsinki, Helsinki, Finland, 2014, pp. 68–69.

[7] A. Irfan, D. Lanti, and N. Manthey, "PCASSO – a parallel cooperative SAT solver," 2014.

[8] G. Audemard, G. Katsirelos, and L. Simon, "A restriction of extended resolution for clause learning SAT solvers," in *AAAI*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[9] N. Manthey, "Extended resolution in modern SAT solving," in *Joint Automated Reasoning Workshop and Deduktionstreffen*, 2014.

[10] S. Jabbour, J. Lonlac, L. Sais, and Y. Salhi, "Revisiting the learned clauses database reduction strategies," *CoRR*, vol. abs/1402.1956, 2014. [Online]. Available: http://arxiv.org/abs/1402.1956

[11] N. Manthey, M. J. Heule, and A. Biere, "Automated reencoding of Boolean formulas," in *Hardware and Software: Verification and Testing*, ser. Lecture Notes in Computer Science, A. Biere, A. Nahir, and T. Vos, Eds., vol. 7857. Springer Berlin Heidelberg, 2013, pp. 102–117. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39611-3_14

[12] M. Järvisalo, M. J. Heule, and A. Biere, "Inprocessing rules," in *Automated Reasoning*, ser. Lecture Notes in Computer Science, B. Gramlich, D. Miller, and U. Sattler, Eds., vol. 7364. Springer Berlin Heidelberg, 2012, pp. 355–370. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31365-3_28

[13] N. Manthey and T. Philipp, "Formula simplifications as DRAT derivations," in *KI 2014: Advances in Artificial Intelligence*, ser. Lecture Notes in Computer Science, C. Lutz and M. Tielscher, Eds., vol. 8736. Springer Berlin Heidelberg, 2014, pp. 111–122.

[14] S. Wieringa and K. Heljanko, "Concurrent clause strengthening," in *SAT*, ser. LNCS, vol. 7962, 2013, pp. 116–132.

[15] G. Audemard and L. Simon, "Lazy clause exchange policy for parallel sat solvers," in *Theory and Applications of Satisfiability Testing – SAT 2014*, ser. Lecture Notes in Computer Science, C. Sinz and U. Egly, Eds. Springer International Publishing, 2014, vol. 8561, pp. 197–205. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-09284-3_15

[16] A. E. Hyvärinen and N. Manthey, "Designing scalable parallel SAT solvers," in *Theory and Applications of Satisfiability Testing – SAT 2012*, ser. Lecture Notes in Computer Science, A. Cimatti and R. Sebastiani, Eds., vol. 7317. Springer Berlin Heidelberg, 2012, pp. 214–227. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31612-8_17

[17] 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.

[18] M. Aigner, A. Biere, C. Kirsch, A. Niemetz, and M. Preiner, "Analysis of portfolio-style parallel SAT solving on current multi-core architectures," in *POS-13*, ser. EPiC Series, D. L. Berre, Ed., vol. 29. EasyChair, 2014, pp. 28–40.

[19] A. Belov, D. Diepold, M. J. Heule, and M. Järvisalo, Eds., *Proceedings of SAT Competition 2014*, ser. Department of Computer Science Series of Publications B. University of Helsinki, Helsinki, Finland, 2014, vol. B-2014-2.