Appears in A. Balint, A. Belov, D. Diepold, S. Gerber, M. Järvisalo, and C. Sinz (eds.), *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, volume B-2012-2 of Department of Computer Science Series of Publications B, University of Helsinki 2012. ISBN 978-952-10-8106-4

satUZK: Solver Description

Alexander van der Grinten*, Andreas Wotzlaw*, Ewald Speckenmeyer*, Stefan Porschen†
*Institut für Informatik
Universität zu Köln, Pohligstr. 1, D-50969 Köln, Germany
Email: {vandergrinten,wotzlaw,esp}informatik.uni-koeln.de
†Fachgruppe Mathematik, FB4
HTW-Berlin, Treskowallee 8, D-10318 Berlin, Germany
Email: porschen@htw-berlin.de

I. Solver Description

satUZK is a conflict-driven clause learning solver for the boolean satisfiability problem (SAT). It is written in C++ from scratch and aims to be flexible and easily extendable.

In addition to the standard DPLL [1] algorithm with clause learning the solver is able to perform various preprocessing and inprocessing techniques.

A. Preprocessing

We implemented SatELite-like variable elimination and self-subsumption [2], unhiding [3], a distillation technique similar to the one presented in [4], blocked clause elimination [5], and variable probing to detect failed literals, equivalent literals, and literals that must be true in every model.

The preprocessing starts with unhiding, followed by self-subsumption and variable probing in order to fix some variables and increase the number of literals that can be propagated by binary constraint propagation (BCP).

After that the size of the formula is reduced by blocked clause elimination and SatELite-like variable elimination. These techniques can reduce the reasoning power of BCP and that is why they are scheduled after the previous preprocessing steps.

Preprocessing generally tries to eliminate 0.5% of the remaining variables in 1% of the available time. All preprocessing techniques are repeated until the number of variables that are affected by each simplification pass becomes too low or a limit of 10% of the time budget is reached. The available time must be specified to the solver with a command-line parameter `-budget <time in sec>`.

By default, the preprocessing phase is disabled and can be activated with a command-line parameter `-preproc-adaptive`.

B. Search

The data structures required for BCP are implemented in the same way as in MiniSAT 2.2 [6]. Binary clauses are stored in a separate watch list.

We are using the standard 1-UIP [1] learning scheme together with conflict clause minimization and the VSIDS decision heuristic with phase saving.

For learned clause deletion the solver can use a MiniSAT-like learned clause deletion strategy or a more aggressive literal blocks distance based deletion strategy [7]. The first strategy is the default one, whereas the latter one can be enabled with command-line parameters `-clause-red-agile -restart-glucose -learn-minimize-glucose -learn-bump-glue-twice`.

Both Luby restarts and glucose-like dynamic restarts are implemented [7].

C. Inprocessing

The DPLL procedure is interleaved with inprocessing steps that perform unhiding, variable probing and distillation. These techniques do not require literal occurrence lists and thus they can be integrated into the search without great performance overheads.

Variable probing and distillation is only applied to the most active variables and clauses.

At most 10% of the available time is used for inprocessing.

II. SAT Challenge 2012 Specifics

For the challenge in tracks "Hard Combinatorial SAT+UNSAT" and "Application SAT+UNSAT" we have submitted three parametrized versions of our solver, started with the following commands:

- satUZK: `satUZK -budget 900 -preproc-adaptive -show-model <cnf instance>`
- satUZKg: `satUZK -budget 900 -show-model -preproc-adaptive -clause-red-agile -restart-glucose -learn-minimize-glucose -learn-bump-glue-twice <cnf instance>`
- satUZKs: `satUZK_wrapper satUZK -budget 900 -preproc-adaptive <cnf instance>`

The last solver uses first SatELite [2] for preprocessing of the input instance before the satUZK solver is called.

All three solvers have been submitted both as precompiled (with gcc 4.4.3 and -O3) and statically linked 64-bit binaries as well as sources written in C++.

References

[1] A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*, 2009.

[2] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination," in *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, ser. Lecture Notes in Computer Science, vol. 3569, 2005, pp. 61–75.

[3] M. Heule, M. Järvisalo, and A. Biere, "Efficient cnf simplification based on binary implication graphs," in *Proceedings to the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2011)*, ser. Lecture Notes in Computer Science, vol. 6695, 2011, pp. 201–215.

[4] H. Han and F. Somenzi, "Alembic: An efficient algorithm for cnf preprocessing," in *Proceedings of the 44th Design Automation Conference (DAC 2007)*, 2007, pp. 582–587.

[5] M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination," in *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*, ser. Lecture Notes in Computer Science, vol. 6015, 2010, pp. 129–144.

[6] N. Eén and N. Sörensson, "Minisat 2.2." [Online]. Available: http://minisat.se/downloads/minisat-2.2.0.tar.gz

[7] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 2009, pp. 399–404.