Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions, volume B-2014-2 of Department of Computer Science Series of Publications B, University of Helsinki 2014. ISBN 978-951-51-0043-6.

Yet another Local Search Solver and Lingeling and Friends Entering the SAT Competition 2014

Armin Biere
Institute for Formal Models and Verification
Johannes Kepler University Linz

***Abstract*—This paper serves as solver description for the SAT solvers Lingeling and its two parallel variants Treengeling and Plingeling, as well as for our new local search solver YalSAT entering the Competition 2014. For Lingeling and its variants we only list important differences to earlier version of these solvers as used in the SAT Competition 2013. For further information we refer to the solver description [1] of the SAT Competition 2013 or source code.**

YALSAT

Recent SAT competitions witnessed a new generation of several efficient local search solvers including Sparrow [2] and ProbSAT [3], which surprisingly were able to solve some non-random instances. Fascinated by this success and the simplicity of the ProbSAT solver [3] we started to implement *Yet Another Local Search SAT Solver* (YalSAT). At its core it implements several variants of ProbSAT's algorithm and recent extensions [4]. These variants are selected randomly at restarts, scheduled by a reluctant doubling scheme (Luby).

Beside initializing the assignment at each restart with a randomly picked assignment among previously saved best assignments within one restart round, it is also possible to assign all variables to false, to true or to a random value. At each restart the submitted version further varies the base cb_k for the exponential score distribution between either using the original base values of ProbSAT, where k is determined by the maximum length of the clause in the instance, or using the default of $cb_k = 2$. Then clause weights are either chosen to be the same for all clauses or are chosen as linear function depending on the clause length (with either larger clauses having larger weight or alternatively smaller clauses). The maximum weight is also picked randomly.

With this set-up we were able to solve a surprisingly large number of satisfiable crafted instances from last year's competition. For uniform random instances YalSAT is supposed to work almost identical to last year's version of ProbSAT.

LINGELING

This year's version *ayv* of Lingeling is slightly improved in several ways. Compared to other solvers in the last competition, last year's version *aqw* of Lingeling performed not well on certain unsatisfiable instances (particularly on miter instances). Our analysis showed that this was simply due to our agility based restart heuristic, which skipped too many restarts on these instances. As first measure we increased the agility limit slightly, but then, inspired by restart policies in Glucose [5], incorporated a new restart policy for skipping restarts, called "saturating". It compares the average LBD (glucose level) versus the average decision height. If the latter is relatively small (70% higher at most) restarts are skipped. Using this new restart heuristic allowed us to completely disable agility based restarts without much penalty on satisfiable instance, but with a substantial postive effect on unsatisfiable instances. As compromise the default version still uses agility based restarts, but we submitted an additional version "Lingeling (no agile)" without agility controlled restarts.

A new technique among the enabled techniques is called "tabula rasa". It monitors the number of remaining variables and clauses. If these numbers drop dramatically (below 25% or 50% respectively) all learned clauses are flushed. Finally, the covered clause elimination (CCE) inprocessor has substantially been improved by for instance trying to eliminate large clauses first, focusing on fast clause elimination procedures in early inprocessing rounds, like asymmetric tautology elimination, and then turning to ABCE and full CCE in later rounds.

PLINGELING

Based on the average number of occurrences per literal and its standard deviation Plingeling tries to figure out whether the instance is actually a uniform random instance. If this seems to be the case it uses the integration of YalSAT into Lingeling and in essence runs a local search as sub-routine until completion. This is enabled for several worker threads, even all but one if clauses all have the same length.

The soft memory limit is set to one third of the physically available memory, while last year, using half the memory still resulted in last year's version of Plingeling to occasionally run out of memory. This was due to excessive memory defragmentation when using many threads, e.g. resident set size being more than twice as large as the actual allocated memory.

As described above, last year's version *aqw* of Lingeling turned out not to work well for several unsatisfiable instances due to skipping too many restarts. Thus the third worker thread disables agility based restart skipping.

Supported by FWF, NFN Grant S11408-N23 (RiSE).

TREENGELING

As discussed above, local search solvers can be quite competetive on a subset of crafted instances. Thus we integrated YalSAT into Lingeling, which by default is disabled but in Treengeling enabled in the already previously existing single parallel top-level worker thread. It is not run until completition though, but simply scheduled as another inprocessor, limited by the number of memory operations. In this set-up the local search sub-routine exports the best solution found to the CDCL part, by setting default phases accordingly.

Another important change in Treengeling was to use an internal cloning function after simplifying the top-level node for several rounds (10 rounds of inprocessing). This is based on the observation, that heavy preprocessing is useful for crafted instances, even though it requires some warm-up time.

After initial preprocessing the number of variables and clauses is usually substantially reduced. Thus much memory is wasted during cloning the full solver. This for instance includes the clauses on the reconstruction stack as well as the mapping of all original variables to those in reduced instances. Our current solution is to clone the initially simplified formula only internally. Then the root solver can not be reused, but will be needed to reconstruct a solution of the original instance. Further solver instances are cloned from this first internal clone. As a consequence, we have three solver instance after the first look-ahead: the root solver, the first internal clone for the first branch, and its dual for the second branch, both with a unit literal added. In last year's version there would be only two solver instances at this point.

Finally, if the available cores are not utilized, Treengeling will split more eagerly, to produce more workers, by simply keeping the conflict limit for CDCL small.

LINGELING DRUPLIG

A substantial amount of work went into improving DRUP tracing for version *azd* of Lingeling, as submitted to the UNSAT tracks. In Lingeling many clauses are implicitly added, deleted or strengthened at various places. In order to find all these places, we implemented a library Druplig, which can be used to dump DRUP traces, but also, in debugging and testing mode, contains a forward online DRUP checker.

This allowed us to reuse our model-based testing and debugging frame work for the incremental API of Lingeling [6] for developping more complete DRUP support. This approach is in our experience much more effective in finding bugs and debugging them, compared to file based fuzzing and delta-debugging [7].

Compared to previous year's DRUP tracing version, we now also trace clause deletion and further were able to enable many more inprocessing algorithms. Most of the probing based inprocessors can produce traces now. Equivalent literal reasoning is enabled too.

However, no form of extended resolution is traced, e.g. only plain DRUP, no DRAT proofs are supported yet. This means blocked clause addition, cardinality reasoning, gaussian elimination, and variable elimination based on irredundant covers all had to be disabled. Further disabled inprocessors are double look-head based equivalence extraction (lifting), congruence closure for equivalences, as well as unhiding. Even though these resolution based inprocessors can all in principle be mapped to DRUP, the effort for adding trace support is much higher and left as future work.

LICENSE

For the competition version of our solvers we use the same license scheme as introduced last year for our solvers. It allows the use of the software for academic, research and evaluation purposes. It further prohibits the use of the software in other competitions or similar events without explicit written permission. Please refer to the actual license, which comes with the source code, for more details.

REFERENCES

[1] A. Biere, "Lingeling, Plingeling and Treengeling entering the SAT Competition 2013," in *Proc. of SAT Competition 2013*, ser. Department of Computer Science Series of Publications B, University of Helsinki, A. Belov, M. Heule, and M. Järvisalo, Eds., vol. B-2013-1, 2013, pp. 51–52.

[2] A. Balint and A. Fröhlich, "Improving stochastic local search for SAT with a new probability distribution," in *SAT*, ser. Lecture Notes in Computer Science, O. Strichman and S. Szeider, Eds., vol. 6175. Springer, 2010, pp. 10–15.

[3] A. Balint and U. Schöning, "Choosing probability distributions for stochastic local search and the role of make versus break," in *SAT*, ser. Lecture Notes in Computer Science, A. Cimatti and R. Sebastiani, Eds., vol. 7317. Springer, 2012, pp. 16–29.

[4] A. Balint, A. Biere, A. Fröhlich, and U. Schöning, "Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses," in *SAT*, ser. Lecture Notes in Computer Science. Springer, 2014.

[5] G. Audemard and L. Simon, "Refining restarts strategies for SAT and UNSAT," in *CP*, ser. Lecture Notes in Computer Science, M. Milano, Ed., vol. 7514. Springer, 2012, pp. 118–126.

[6] C. Artho, A. Biere, and M. Seidl, "Model-based testing for verification back-ends," in *TAP*, ser. Lecture Notes in Computer Science, M. Veanes and L. Viganò, Eds., vol. 7942. Springer, 2013, pp. 39–55.

[7] R. Brummayer, F. Lonsing, and A. Biere, "Automated testing and debugging of SAT and QBF solvers," in *SAT*, ser. Lecture Notes in Computer Science, O. Strichman and S. Szeider, Eds., vol. 6175. Springer, 2010, pp. 44–57.

[8] O. Strichman and S. Szeider, Eds., *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6175. Springer, 2010.