CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2018

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

***Abstract*—This note documents the versions of our SAT solvers submitted to the SAT Competition 2018, which are CaDiCaL, Lingeling, its two parallel variants Treengeling and Plingeling, and our local search solver YalSAT.**

Lingeling,Plingeling, Treengeling,YalSAT

Compared to the version of Lingeling submitted last year [1] we added *Satisfication Driven Clause Learning* (SDCL) [2], which however due to its experimental nature is disabled (option "`--prune=0`"). We further disabled blocked clause removal (option "`--block=0`") [3], binary blocked clause addition (option "`--bca=0`") [4], as well as on-the-fly subsumption (option "`--otfs=0`") [5], since all of them can not be combined with SDCL style pruning.

As in our new version of CaDiCaL we also experimented with bumping reason side literals too, as suggested in [6]. See below for the motivation to include this feature. There is also a slight change in the order how literals are bumped: previously they were bumped in trail order and are now bumped in variable score order.

Since already last year's version of Lingeling [1] was almost identical to that from the SAT 2016 Competition [7], it is fair to say that Lingeling and also its parallel extensions Plingeling and Treengeling essentially did not change since 2016. This applies even more to the submitted version of YalSAT, which surprisingly won the random track in 2017, even though it did not change since 2016.

CaDiCaL

As explained in our last year's solver description [1] the goal of developing CaDiCaL was to produce a radically simplified CDCL solver, which is easy to understand and change. This was only partially achieved, at least compared to Lingeling. On the other hand the solver became competitive with other state-of-the-art solvers, actually surpassing Lingeling in performance in the last competition, while being more modular, as well as easier to understand and change.

We also gained various important new insights starting to develop a SAT solver (again) from scratch [1], particularly how inprocessing attempts for variable elimination and subsumption should be scheduled, and how subsumption algorithms can be improved (see again [1] for more details).

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

On the feature side not much changed, since CaDiCaL still does not have a complete incremental API (assumptions are missing). However, the non-incremental version was used as back-end of Boolector in the SMT 2017 Competition [8] in the quantifier-free bit-vector track (QF_BV), where it contributed to the top performance of Boolector (particularly compared to the version with Lingeling as back-end).

Our analysis of the SAT 2017 Competition [9] results revealed that the technique of *bumping reason side literals* [6] of MapleSAT [6] and successors [10], [11] has an extremely positive effect on the selected benchmarks. It consists of going over the literals in learned (minimized 1st UIP) clauses and "bumping" [12] all other literals in their reason clauses too. Even though MapleSAT actually only uses this technique with the new variable scoring scheme proposed in [6], it is already effective in combination with the VMTF scheme [12] used in CaDiCaL (and probably for VSIDS too).

Last year's success of the MapleLCM solver [11], which is an extension of MapleSAT by a different set of authors, also showed that vivification [13] of *learned* clauses as described by the authors of MapleLCM in their IJCAI paper [14] can be quite useful. In last year's version of CaDiCaL we already had a fast implementation of vivification [1], but only applied it to *irredundant* clauses. During inprocessing [15] our new version of CaDiCaL has two vivification phases, the first phase working on all including *redundant* clauses and the second phase works as before only on irredundant clauses.

Furthermore, all the top performing configurations of MapleSAT and MapleLCM made use of the observation of Chanseok Oh [16], that a CDCL solver should alternate between "quiet" no-restart phases and the usual fast restart scheduling [17]. This also turns out to be quite beneficial for last year's selection of benchmarks and we added such "stabilizing" phases scheduled in geometrically increasing conflict intervals.

Then, we experimented with "rephasing", which in arithmetically increasing conflict intervals overwrites all saved phases [18] and either (i) restores the initial phase (default *true* in CaDiCaL), (ii) flips the current saved phase, (iii) switches to the inverted initial phase (thus *false*), or (iv) picks a completely random phase. This technique gives another (but smaller) boost to the performance of CaDiCaL on last year's benchmarks compared to the other new techniques above.

Finally, we observed, that for very long running instances (taking much longer than the 5000 seconds time limit used in the competition), the standard arithmetic increase [19] of the limit on kept learned clauses increases memory usage over time substantially and slows down propagation. Therefore we flush all redundant clauses (including low glucose level clauses) in geometrically increasing conflict intervals too. This should happen less than a dozen of times during each competition run though.

LICENSE

The default license of YALSAT, LINGELING, PLINGELING and TREENGELING did not change in the last three years. It allows the use of these solvers for research and evaluation but not in a commercial setting nor as part of a competition submission without explicit permission by the copyright holder. However, as part of our new open source release of BOOLECTOR 3.0 [20] we also plan to release LINGELING under an open source MIT style license, which for CADICAL continues to be the case.

REFERENCES

[1] A. Biere, "Deep Bound Hardware Model Checking Instances, Quadratic Propagation Benchmarks and Reencoded Factorization Problems Submitted to the SAT Competition 2017," in *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2017-1. University of Helsinki, 2017, pp. 40–41.

[2] M. J. H. Heule, B. Kiesl, M. Seidl, and A. Biere, "Pruning through satisfaction," in *Haifa Verification Conference*, ser. Lecture Notes in Computer Science, vol. 10629. Springer, 2017, pp. 179–194.

[3] M. Järvisalo, A. Biere, and M. Heule, "Blocked clause elimination," in *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, ser. Lecture Notes in Computer Science, J. Esparza and R. Majumdar, Eds., vol. 6015. Springer, 2010, pp. 129–144.

[4] M. Järvisalo, M. Heule, and A. Biere, "Inprocessing rules," in *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, ser. Lecture Notes in Computer Science, B. Gramlich, D. Miller, and U. Sattler, Eds., vol. 7364. Springer, 2012, pp. 355–370.

[5] H. Han and F. Somenzi, "On-the-fly clause improvement," in *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed., vol. 5584. Springer, 2009, pp. 209–222.

[6] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for SAT solvers," in *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, ser. Lecture Notes in Computer Science, N. Creignou and D. L. Berre, Eds., vol. 9710. Springer, 2016, pp. 123–140.

[7] A. Biere, "Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016," in *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2016-1. University of Helsinki, 2016, pp. 44–45.

[8] "Boolector at the SMT competition 2017," FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep., 2017.

[9] T. Balyo, M. Heule, and M. Järvisalo, Eds., *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, vol. B-2017-1. University of Helsinki, 2017.

[10] J. H. Liang, C. Oh, V. Ganesh, K. Czarnecki, and P. Poupart, "MapleCOMSPS_LRB_VSIDS and MapleCOMSPS_CHB_VSIDS," in *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2017-1. University of Helsinki, 2017, pp. 20–21.

[11] F. Xiao, M. Luo, C.-M. Li, F. Manyà, and Z. Lü, "MapleLRB_LCM, Maple_LCM, Maple_LCM_Dist, MapleLRB_LCMoccRestart and Glucose-3.0+width in SAT Competition 2017," in *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2017-1. University of Helsinki, 2017, pp. 22–23.

[12] A. Biere and A. Fröhlich, "Evaluating CDCL variable scoring schemes," in *SAT*, ser. Lecture Notes in Computer Science, vol. 9340. Springer, 2015, pp. 405–422.

[13] C. Piette, Y. Hamadi, and L. Sais, "Vivifying propositional clausal formulae," in *ECAI*, ser. Frontiers in Artificial Intelligence and Applications, vol. 178. IOS Press, 2008, pp. 525–529.

[14] M. Luo, C. Li, F. Xiao, F. Manyà, and Z. Lü, "An effective learnt clause minimization approach for CDCL SAT solvers," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 2017, pp. 703–711.

[15] M. Järvisalo, M. Heule, and A. Biere, "Inprocessing rules," in *IJCAR*, ser. Lecture Notes in Computer Science, vol. 7364. Springer, 2012, pp. 355–370.

[16] C. Oh, "Between SAT and UNSAT: the fundamental difference in CDCL SAT," in *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, ser. Lecture Notes in Computer Science, M. Heule and S. Weaver, Eds., vol. 9340. Springer, 2015, pp. 307–323.

[17] A. Biere and A. Fröhlich, "Evaluating CDCL restart schemes," in *Proceedings POS-15. Sixth Pragmatics of SAT workshop*, 2015, to be published.

[18] K. Pipatsrisawat and A. Darwiche, "A lightweight component caching scheme for satisfiability solvers," in *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, ser. Lecture Notes in Computer Science, J. Marques-Silva and K. A. Sakallah, Eds., vol. 4501. Springer, 2007, pp. 294–299.

[19] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI*, 2009, pp. 399–404.

[20] A. Niemetz, M. Preiner, C. Wolf, and A. Biere, "Btor2, BtorMC and Boolector 3.0," in *Computer Aided Verification - 30th International Conference, CAV 2018*, ser. Lecture Notes in Computer Science. Springer, 2018, to appear.