CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT Entering the SAT Competition 2017

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

***Abstract*—This paper serves as a first solver description for our new SAT solver CaDiCaL and documents the versions of our other solvers submitted to the SAT Competition 2017, which are Lingeling, its two parallel variants Treengeling and Plingeling, and our local search solver YalSAT.**

Lingeling,Plingeling, Treengeling,YalSAT

Our focus in the SAT Competition 2016 was on our new SAT solver Splatz [1]. It tried to simplify the Lingeling design, and further implemented a first inprocessing [2] version of blocked clause decomposition for SAT sweeping to detect equivalences. In the same spirit, we focus on a new solver called CaDiCaL in the SAT Competition 2017.

We submitted to the *agile*, *main*, and *no-limit* tracks of the SAT Competition 2017 the Lingeling version *bbe*, which except for some minor bug fixes in the code for picking random decisions is the same as the version entering the SAT Competition 2016 [1]. Its parallel extensions Plingeling and Treengeling submitted to the *parallel* track have the same version *bbe* accordingly. They were were marking the state-of-the-art in the parallel track of the SAT Competition 2016 [3] and have not changed at all.

The same applies to our local search solver YalSAT which also did not really change and was submitted as version *03s* to the *random* track only.

CaDiCaL

The goal of the development of CaDiCaL was to obtain an inprocessing solver [2], which is easy to understand and change, while at the same time not being much slower than other state-of-the-art solvers. Originally we also wanted to radically simplify the design and internal data structures. But that goal was only achieved partially, for instance compared to Lingeling. On the other hand, after adding, what we believe, are essential ingredients of a state-of-the-art solver, the solver *did* become competitive with other state-of-the-art solvers, for instance surpassing Lingeling on many instances. The name of the solver has its roots in "radical(ly)" and "CDCL" [4].

The main search loop interleaves inprocessing [2] and CDCL [4] search. The inprocessing part consist of three individually scheduled inprocessing methods: probing, subsumption, and (bounded) variable elimination.

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

During (failed literal) **probing** only roots of the binary implication graph are probed and binary clauses are learned through hyper binary resolution [5]. These are used to eliminate equivalent literals after decomposing the binary implication graph into strongly connected components, which is scheduled right before and after probing. Hyper binary clauses tend to be generated many and thus will only survive at most one clause reduction. We also explicitly remove duplicated binary clauses before probing.

As in Splatz we also remove subsumed learned clauses during **subsumption** in regular intervals. Due to a new much faster subsumption algorithm than in previous solvers [6] we can afford to apply subsumption checking to redundant learned clauses with small glucose level [7] too, which might otherwise be kept forever. Of course, we also perform self-subsuming resolution [6] to strengthen clauses. We also have a second propagation based subsumption check, similar to vivification [8], in each subsumption phase, which however is restricted to irredundant clauses only. For binary clauses there is a specialized transitive reduction algorithm for the binary implication graph at the end of each subsumption phase.

As in other solvers (bounded) **variable elimination** [6] is one of the most effective inprocessing techniques and is carefully scheduled as in Lingeling [9], except that we wait for an initial interval (of 1000) conflicts before being triggered. Variable elimination is interleaved with subsumption for a bounded number of rounds and as long something changes.

More precisely, the solver carefully monitors variables which occurred in removed irredundant clauses or in added (arbitrary redundant or irredundant) clauses. Removed variables trigger variable elimination attempts, while added variables trigger subsumption checks. This information is kept persistent across CDCL search and inprocessing phases and allows the solver to restrict the effort in subsumption checking and variable eliminations to those part of the formula which changed.

On the **search** side, we incorporated the idea of saving the position of the last replaced watch in large clauses [10], use VMTF instead of VSIDS as explained in [11], schedule restarts based on exponential moving averages [12] and alternate and reset the default phase of decisions, starting with picking true as initial phase of yet unassigned decision variables.

The version *sc17* of CADICAL, which in essence is identical to our internal version *058*, was submitted to the *agile*, *main* and *no-limit* track. In contrast to LINGELING generating proofs for the *main* track should not change how the solver works and in our experiments adds negligible overhead. Currently only DRUP proofs are generated.

We have also made some minor effort to come up with parameter settings, which improve CADICAL on the *agile* track compared to its otherwise used default configuration. From the benchmarks of the *agile* track of the SAT Competition 2016 substantially more could be solved, if the base restart interval is increased from 6 to 400 conflicts and flipping and resetting the default decision phase is disabled. We have also submitted this "agile" version to the *agile*, *main* and *no-limit* tracks.

The solver is implemented from scratch in a modular way in C++. There is also an API interface for C, but the core library is not ready for incremental usage yet, since it is lacking assumption handling. The source code strives to be carefully documented and consists of roughly 10 000 lines of code.

LICENSE

The default license of YALSAT, LINGELING, PLINGELING and TREENGELING did not change in the last two 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. For the new solver CADICAL we use an MIT style license which is far less restrictive.

REFERENCES

[1] 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.

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

[3] T. Balyo, M. J. H. Heule, and M. Järvisalo, "SAT competition 2016: Recent developments," in *AAAI*. AAAI Press, 2017, pp. 5061–5063.

[4] J. P. M. Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning SAT solvers," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, vol. 185, pp. 131–153.

[5] M. Heule, M. Järvisalo, and A. Biere, "Revisiting hyper binary resolution," in *CPAIOR*, ser. Lecture Notes in Computer Science, vol. 7874. Springer, 2013, pp. 77–93.

[6] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *SAT*, ser. Lecture Notes in Computer Science, vol. 3569. Springer, 2005, pp. 61–75.

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

[8] 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.

[9] A. Biere, "Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling," in *POS@SAT*, ser. EPiC Series in Computing, vol. 27. EasyChair, 2014, p. 88.

[10] I. P. Gent, "Optimal implementation of watched literals and more general techniques," *J. Artif. Intell. Res. (JAIR)*, vol. 48, pp. 231–251, 2013.

[11] 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.

[12] ——, "Evaluating CDCL restart schemes," in *Proceedings POS-15. Sixth Pragmatics of SAT workshop*, 2015, to be published.