Appears in A. Balint, A. Belov, M.J.H. Heule, and M. Järvisalo (eds.), *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, volume B-2013-1 of Department of Computer Science Series of Publications B, University of Helsinki 2013. ISBN 978-952-10-8991-6.

Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013

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

***Abstract*—This paper serves as solver description for our SAT solver Lingeling and its two parallel variants Treengeling and Plingeling entering the SAT Competition 2013. We only list important differences to the version of these solvers used in the SAT Challenge 2012. For further information we refer to the solver description [1] of the SAT Challenge 2012 or source code.**

LINGELING

The differences on the search side, that is during the CDCL loop, are as follows. The inner-outer scheme for reduce scheduling is not enabled by default, but only enabled dynamically, if the number of remaining variables drops below 1000. It also turned out that the previous VMTF decision scheduler, though faster to compute, is less robust, and occasionally leads to time-outs on otherwise easy to satisfy instances. Thus we went back to the exponential VSIDS scheme of MiniSAT. Costly recursive clause minimization [2] is only attempted for clauses with small glucose level (LBD) [3]. Local clause minimization is tried for somewhat higher glucose levels. For even higher glucose levels and if in addition the 1^{st}-UIP clause is rather long then a decision-only clause is learned instead of the (minimized) 1^{st}-UIP clause (proposed by Donald Knuth in private communication). The decision-only clause contains the negations of all decisions required to generate the conflict, except for the last decision, which is replaced by the 1^{st}-UIP literal (if different). The variable scores are updated based on the generated but discarded 1^{st}-UIP clause.

Lingeling uses various *inprocessing* algorithms [4], which not only simplify the formula initially and in this case act as *preprocessors*, but also in regular intervals between calls to the CDCL search loop. All inprocessors are running as part of one single simplification phase. The number of conflicts is used as metric to measure the effort spent in search and if a conflict limit is reached, the solver switches to simplification.

In principle, the conflict interval for simplification is increased geometrically. However, depending on the amount of reduction achieved in the last simplification phase, measured in terms of the percentage of removed variables, the increment of the simplification interval is reduced. The more variables are removed the sooner the next simplification phase is scheduled. Furthermore, each inprocessing algorithm monitors its effectiveness individually. If an inprocessor was unsuccessful in the previous simplification phase, the inprocessor is skipped in the next simplification phase. If unsuccessful again, it is skipped twice etc. Various more advanced inprocessors are delayed until either blocked clause elimination or bounded variable elimination is completed at least once.

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

Regarding changes in individual inprocessors we note the following. Tree-based look-ahead already mentioned in [1] has been published [5]. Bounded variable elimination is now much more restricted. It only tries to eliminate variables with few occurrences and thus terminates resp. runs to completion much earlier than in previous versions.

We further realized that during literal probing, which occurs as part of various preprocessors, clauses satisfied during probing which contain the negation of the probed literal are under certain restrictions asymmetric tautologies [6] and can be removed. Actually, in general, short (binary or ternary) clauses determined to be redundant, such as these basic asymmetric tautologies, but also blocked clauses or covered clauses, are "moved", i.e. they are marked as redundant resp. learned clauses to preserve BCP as discussed in [4].

As also discussed in [4] we generate and add binary blocked clauses. To avoid full occurrence lists for large redundant resp. learned clauses, we only add blocked clauses with a blocking literal, which does not occur negated in large redundant clauses at all. Blocked clause addition is disabled in the parallel solvers Plingeling and Treengeling.

Finally, we added a simple form of cardinality constraint reasoning, which is similar to our previously added Gaussian elimination procedure [1]. We first extract trivially encoded at-most-one and at-most-two constraints by a simple and incomplete syntactic procedure. All clauses, which contain a literal occurring in an extracted cardinality constraint are added as cardinality constraint too, e.g. as at-most-k constraint, where $l = k + 1$ is the length of the clause. For this set of cardinality constraints we perform a simple form of variable elimination, as in the Fourier-Motzkin elimination procedure. This technique allows to derive an inconsistent constraint for large pigeon-hole formulas in a fraction of a second. Otherwise the procedure exports derived units and binary clauses.

For the certified UNSAT track we had to disable blocked clause addition, Gaussian elimination and cardinality constraint reasoning, since they can not be simulated by resolution (polynomially). Furthermore, equivalent literal substitution (ELS) turned out to be hard to map to (D)RUP and thus we had to disable all inprocessors which rely on ELS.

PLINGELING

The first version of Plingeling only shared units, while more recent versions also share equivalences. In this new version we also share short clauses with small glucose level, i.e. clauses with at most 40 literals and glucose level of at most 8. In contrast to [7] all exported clauses are imported unless they contain a "melted" literal, such as those eliminated or used as blocking literal in blocked or covered clause elimination during inprocessing. The same restriction was already required for importing equivalences.

Clauses are exported to the master and copied to a global stack. Each slave solver thread imports clauses from this global stack in regular intervals during the CDCL loop, oldest clauses first. Thus this global clause stack actually acts as a queue. The procedure for importing clauses triggers garbage collection of global clauses already imported (consumed) by all solvers in regular intervals.

TREENGELING

Treengeling is a parallel solver based on Cube & Conquer [8], [9], which tries to combine the strengths of look-ahead solving with CDCL solving and in the case of Treengeling also with inprocessing. The basic architecture of Treengeling was already described in [1].

In this new version we essentially added three improvements. First, and most important, we flipped the policy for changing the conflict limit for each search node in order to match the original motivation for Cube & Conquer. If more nodes are closed by a combination of CDCL and inprocessing we double the global conflict limit. Otherwise if the number of closed nodes is smaller than the number of added nodes then the conflict limit is decreased with a rate of 90%. This actually only happens if at least one node was closed and otherwise, if no node was closed, the conflict limit is even decreased by 50%. If the soft memory limit is hit nodes are not split and we end up in the first case (doubling the limit). Further there is a minimum conflict limit of 1,000 conflicts, an initial conflict limit of 10,000 and a maximum conflict limit of 100,000 conflicts. This limit is applied to the search phase of one node in one round (which also includes inprocessing).

The second improvement consists of disabling full tree-based look-ahead [5] if look-ahead alone removes less than 2% of the remaining variables. There is a similar penalty scheme as for inprocessors in Lingeling though. In this situation the next full look-ahead will be skipped and only a cheap to compute static heuristics is used instead.

Finally, Treengeling combines part of the infrastructure of Plingeling with Cube & Conquer by using one core for running an additional solver thread, which exports units to the worker threads in Cube & Conquer. Thus on an 8 core machine, 7 cores are allocated to Cube & Conquer worker threads, which work on 8 times more, thus 56 active nodes. On our 12 core machine with hyper threading, thus 24 virtual cores, the solver will use at most $184 = 8 * 23$ active nodes in parallel.

LICENSE

For the competition version of our solvers we use a new license scheme. It only allows the use of the software for academic and research purposes and 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 and friends entering the SAT Challenge 2012," in *Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, University of Helsinki, vol. B-2012-2, 2012, pp. 33–34.

[2] N. Sörensson and A. Biere, "Minimizing learned clauses," in *Proc. SAT'09*. Springer, 2009, pp. 237–243.

[3] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proc. IJCAI'09*. Morgan Kaufmann, 2009, pp. 399–404.

[4] M. Järvisalo, M. Heule, and A. Biere, "Inprocessing rules," in *Proc. 6th Intl. Joint Conf. on Automated Reasoning (IJCAR'12)*, ser. LNCS, vol. 7364. Springer, 2012, pp. 355–370.

[5] M. Heule, M. Järvisalo, and A. Biere, "Revisiting hyper binary resolution," in *Proc. 10th Intl. Conf. on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'13)*, ser. LNCS, vol. 7874. Springer, 2013, pp. 77–93.

[6] M. J. H. Heule, M. Järvisalo, and A. Biere, "Clause elimination procedures for CNF formulas," in *Proc. LPAR-17*, ser. LNCS, vol. 6397. Springer, 2010, pp. 357–371.

[7] G. Audemard, B. Hoessen, S. Jabbour, J.-M. Lagniez, and C. Piette, "Revisiting clause exchange in parallel SAT solving," in *Proc. SAT'12*, ser. LNCS, vol. 7317. Springer, 2012, pp. 200–213.

[8] P. van der Tak, M. Heule, and A. Biere, "Concurrent cube-and-concur," in *Proc. 3rd Intl. Work. on Pragmatics of SAT (POS'12)*, 2012.

[9] M. Heule, O. Kullmann, S. Wieringa, and A. Biere, "Cube and conquer: Guiding CDCL SAT solvers by lookaheads," in *Proc. HVC'11*, ser. LNCS, vol. 7261. Springer, 2012, pp. 50–65.