Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016

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

***Abstract*—This paper serves as solver description for our new SAT solver Splatz and further documents the versions of our other solvers submitted to the SAT Competition 2016, which are Lingeling, its two parallel variants Treengeling and Plingeling, and our local search solver YalSAT.**

LINGELING

Our sequential solver Lingeling version *bbc* was submitted to all but the RandomSAT track. For the Main track we linked it with our Druplig library to print proof traces. For the other tracks the library was not included. This results in two different solvers according to the competition rules. In essence the same effect could have been achieved by changing command line options, but would result in a small overhead for those checks that skip trace generation.

This version *bbc* of the submission is in essence the same as version *bal* of Lingeling submitted to the SAT Race 2015 last year. It incorporates insights from [1], [2] and is described in the corresponding SAT Race 2015 solver description [3]. There is a small improvement in cardinality reasoning. Another new technique, which resets the reduce interval is disabled. The same applies to restart blocking and parameter selection through bucket classification, using a k-means classifier, which all do not seem to pay off, and are disabled.

As Lingeling has many preprocessing and inprocessing algorithms implemented we expect certain benchmarks to be uniquely solved by it. Because of this feature the amount of time Lingeling is allowed to spend in preprocessing and inprocessing is high. As a consequence this high effort parameter setting used in the submission may not work well for the Agile track even though it could be tuned to do so. It should be beneficial for long runs in the other tracks though.

Since proof trace generation through Druplig is still not completely implemented for all preprocessing and inprocessing techniques yet, only a subset of techniques is enabled in the Main track. For instance cardinality and XOR reasoning are disabled in the Main track.

PLINGELING, TREENGELING

The parallel solvers Plingeling and Treengeling are based on Lingeling and use exactly the same version *bbc* as the submitted sequential version. The front-ends have not changed.

Further, as before, the submitted Treengeling solver links to YalSAT version *03r*, which is run during inprocessing in a small fraction of parallel Lingeling instances. This is expected to be beneficial for crafted instances used in past competitions.

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

YALSAT

To the RandomSAT track we submitted our sequential local search solver YalSAT version *03r*. Even though we experimented with focusing on eagerly flipping break zero variables, the submitted version *03r* does not incorporate algorithmic nor heuristic changes and should behave as the previous version *03l* used in 2014 [4].

Since YalSAT solves some hard satisfiable crafted but also application instances used in past competitions, we also submitted it to the other tracks (Agile,Main,NoLimit). As YalSAT does not use any preprocessing nor any portfolio style combination with a CDCL solver, we do not expect top-class performance in the RandomSAT track, compared to other participating solvers.

The main purpose of submitting YalSAT is to see whether a local search solver can solve some interesting non-random benchmarks, for which other solvers have a hard time to solve them and on the other hand determine its base performance for the random track.

SPLATZ

First, it is pretty challenging to make changes to Lingeling and thus new ideas are hard to add and explore. Further, as in SAT solving, we argue that restarts are valuable and might trigger new ideas. Finally, it is important to consolidate already existing ideas in order to understand their effectiveness.

This lead to the development of our new SAT solver Splatz, with version *03v* submitted to the competition. This solver is developed from scratch in C++. It is a sequential stand-alone SAT solver, with static data-structures, non-reentrant and without API nor incremental usage. However, it is much better documented than say Lingeling and contains many cross-references and explanations.

More specifically, we wanted to implement a new solver, which first has a slightly less optimized, but easier to change clause storage and watching mechanism than Lingeling. This enabled us to implement an inprocessing version of blocked clause decomposition and SAT sweeping, which was left as future work in [5].

The decision heuristics uses stamping based VMTF instead of VSIDS as proposed in [1]. Restart scheduling follows [2].

As in the submitted version of Lingeling we further replaced Glucose style restart blocking by delaying restarts, which works as follows. Assume a restart is supposed to occur. This is called "forced" in Glucose terminology, and according to [2] is triggered if the fast moving glucose level average is above the slow moving average. If in this situation the current decision level is smaller than 50% of the (exponential moving) backjump level average, then restarting is delayed.

We also made the subsumption phase, usually interleaved with bounded variable elimination [6], more efficient by incorporating ideas from [7]. This allows to apply subsumption and shrinking to learned clauses as well. The major benefit is that this in turn allows to remove subsumed (and shrink) "sticky" clauses. As in Glucose [8] these sticky clauses, or low glucose level (LBD) clauses, are clauses which are never removed during learned clause cleaning (also called "reduction"), for instance clauses of glucose level 2. In MiniSAT [9] subsumed learned clauses become inactive and thus by activity based clause cleaning get removed automatically.

On the other side we realized that using a static limit on the glucose level to determine which clauses are sticky, can be replaced by a dynamic limit, by measuring the average glucose level and size of clauses resolved during conflict analysis. Clauses are considered important and become sticky if their glucose level and size is below these measured averages.

This new solver is lacking preprocessing and inprocessing techniques implemented in Lingeling, which we consider to be useful and eventually should be added. Thus the performance of Splatz is not expected to match the performance of Lingeling yet.

A (probably partial) list of implemented data-structures and algorithms is provided here:

- arena based memory allocation for clauses and watchers
- blocking literals (BLIT)
- special handling of binary clause watches
- literal-move-to-front watch replacement (LMTF)
- learned clause minimization
- on-the-fly hyper-binary resolution (HBR)
- learning additional units and binary clauses
- on-the-fly self-subsuming resolution (OTFS)
- decision only clauses (DECO)
- failed literal probing on binary implication graph roots
- eager recent learned clause subsumption
- stamping based VMTF instead of VSIDS
- subsumption for both irredundant and learned clauses
- blocked clause decomposition (BCD) enabling ...
- ... SAT sweeping for backbones and equivalences
- equivalent literal substitution (ELS)
- bounded variable elimination (BVE)
- blocked clause elimination (BCE)
- dynamic sticky clause reduction
- exponential moving average based restart scheduling
- delaying restarts
- trail reuse

For details about these and other ideas implemented in the solver, which due to space constraints can not all be discussed nor referenced here, we recommend to consult the source code and its documentation with more references and explanations.

LICENSE

REFERENCES

[1] A. Biere and A. Fröhlich, "Evaluating CDCL variable scoring schemes," 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. 405–422.

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

[3] A. Biere, "Lingeling and friends entering the SAT Race 2015," Johannes Kepler University, Linz, Austria, FMV Report Series Technical Report 15/2, April 2015.

[4] ——, "Yet another local search solver and Lingeling and friends entering the SAT Competition 2014," in *Proc. of SAT Competition 2014*, ser. Department of Computer Science Series of Publications B, A. Belov, M. J. H. Heule, and M. Järvisalo, Eds., vol. B-2014-2. University of Helsinki, 2014, pp. 39–40.

[5] M. Heule and A. Biere, "Blocked clause decomposition," in *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, ser. Lecture Notes in Computer Science, K. L. McMillan, A. Middeldorp, and A. Voronkov, Eds., vol. 8312. Springer, 2013, pp. 423–438.

[6] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer, 2005, pp. 61–75.

[7] R. J. Bayardo and B. Panda, "Fast algorithms for finding extremal sets," in *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*. SIAM / Omnipress, 2011, pp. 25–34.

[8] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, C. Boutilier, Ed., 2009, pp. 399–404.

[9] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518.