*Proceedings of SAT Competition 2024: Solver and Benchmark Descriptions*volume B-2024-1 of Department of Computer Science Series of Publications B, University of Helsinki 2024 .

CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024

Armin Biere*, Tobias Faller*, Katalin Fazekas†, Mathias Fleury*, Nils Froleyks‡, Florian Pollitt*
* University of Freiburg, Germany † TU Wien, Austria ‡ Johannes Kepler University, Linz, Austria

Abstract—In this short system description we describe our three sequential SAT solvers CaDiCaL, IsaSAT and Kissat submitted to the main track of the SAT competition 2024 as well as an update to Gimsatul submitted to the parallel track.

I. CaDiCaL

In the SAT competition 2023, CaDiCaL performed quite well, especially when combined with structured blocked clause addition (SBVA) [1] in SBVA-CaDiCaL [2]. The authors of SBVA-CaDiCaL used CaDiCaL 1.5.3, which was also the basis for the CaDiCaL hack track in 2023. The CaDiCaL hack track was actually won CaDiCaL_vivinst [3]. by integration of instantiation as last resort during vivification. In contrast to 2024, we did not submit an updated CaDiCaL version to the SAT competition 2023, compared to 1.5.3 used in the hack track, as we opted instead to use our main track submission slot for TabularaSAT. We omitted TabularaSAT in 2024. In 2022 we used all slots to submit several variants of Kissat [4].

For the SAT competition 2024, we submitted our major release, CaDiCaL 2.0 [5], which was described in a recent CAV'24 paper. It adds many new features around incremental usage, proof generation [6], interpolation, and the user-propagator [7]. Notably, the stand-alone solver core of CaDiCaL 2.0 is similar to version 1.9.4, which served as the basis for the CaDiCaL hack track. The only difference between the two versions is an unintended performance regression compared to release 1.7.4 from September 2023. However, this regression was only slightly visible on the anniversary track and in particular was not detected before the competition organizers had to freeze the version of the hack track.

This regression was due to a change taking success in shrinking the glue [8] during strengthening redundant clauses [9] in forward subsumption as a sign for such a clause to have been used, which in turn gives those affected redundant clauses a higher chance to survive the next learned clause reduction. This change has been reverted in version 1.9.5 and in the submitted 2.0.0 version too and is expected to give a minor advantage for our submitted version over the hack track base version 1.9.4.

From version 1.5.3 (used in the hack track of the SAT competition 2023 and the winner of the main track SBVA-CaDiCaL in 2023) to 1.9.4 (as used in the hack track in 2024), we have the following algorithmic changes to the core solver on top of features described in [5]: on-the-fly self-subsuming resolution and instantiation during vivification as last resort plus the update of the "used" flag which in turn was reverted in 1.9.5 and kept reverted in the submitted version 2.0.0.

II. Gimsatul

Our multi-threaded shared-memory SAT solver Gimsatul [10] has been extended with glue-promotion (recomputing and reducing the glue of resolved clauses), dynamic computation of the tier 1 glue limit (as described for Kissat below), thread-local eager subsumption of the last learned clauses and we further revisited vivification (again see below as described for Kissat).

III. IsaSAT

Compared to last year [3] there are no major feature changes in IsaSAT, but we only consolidated existing techniques. First, we identified and fixed a minor heuristic issue. Instead of bumping the literals involved in conflict analysis, we also bumped the conflict clause. This minor difference made a small (but noticeable) difference on the SAT Competition 2022 benchmarks, without any real difference on the 2023 benchmarks. We further fixed the computation of the height of the conflict-free part of the trail which is used in the target and best phasing (cf. [11]).

For one family we observed a very low conflict rate in focused mode (with many restarts and VMTF decision heuristics), and therefore introduced ticks (as implemented in Kissat for a long time) and thus limited recursive reason side bumping when the decision rate (number of decisions between two conflicts) is too high, however, without much success on the investigated family. Still it achieves the effect that focused and stable mode are more balanced, which was the motivation to keep it.

Finally, we deactivated our version of pure literal elimination as it is not compatible with DRAT (as discovered last year during the competition when proof checking broke). Our implementation simply learns the unit clause of that literal and is (provably) correct. In our test cases however these clauses were always blocked (our proof output is not verified), which however in general does not have to be the case. The issue is that pure literals redundancy is a criteria that needs only be checked on irredundant clauses [9], i.e., redundant clauses can be

ignored. However, DRAT does not distinguish these sets of clauses, leading to incorrect proofs (even if the answer is correct). Most SAT solvers use variable elimination where all clauses containing the pure literal are eliminated, without learning a unit. However, we found our approach easier to include in our framework. In private communication, Jakob Nordström pointed out that veriPB actually supports adding units for pure literals, but we leave production of veriPB proofs as future work.

IV. Kissat

Clausal congruence closure [12] is a major new addition to the submitted version of Kissat. It combines syntactic extraction of gates and a bit-level version of congruence closure. In contrast to other attempts achieving the same effect in earlier solvers this is our first implementation which can be run to completion on large instances. It solves isomorphic miters instantly (comparing two identical circuits [12]) while other techniques fail including plain CDCL. Congruence closure is run during each round of probing-inprocessing and nicely complements semantic SAT-sweeping technique using our embedded SAT solver Kitten. This clausal equivalence sweeping was available before and is thoroughly described in [13]. Note that our implementation of gate extraction and congruence closure is fast enough to run until completion whenever it is scheduled initially and during inprocessing in contrast to semantic SAT sweeping which needs to be preempted.

We further added an internal version of bounded variable addition [14], a technique which performed very well in SBVA-CaDiCaL [2] as preprocessor for CaDiCaL in the SAT competition 2023. We have implemented an optimized version of the original version [14] and on-top the tie-breaking heuristic of [1], which however is disabled as it only helped on a few benchmarks (high-lighted in [1]) but overall gave worse results.

We also realized that the average glue (LBD) [8] varies dramatically between different formulas and actually also on the same formula between running in an interleaved fashion in stable and focused mode (called SAT/UNSAT in [15]), i.e., differing in restart frequency and variable score decay. Therefore, for (each mode) we count how many clauses of a certain glue where used (resolved and thus bumped) during conflict analysis. Then we use these statistics to dynamically compute limits on the glue for tier 1 (clauses kept unconditionally) and tier 2 clauses (clauses given a second chance if not used since the last redundant clause-data-base reduction). Typical hard coded limits are glue 2 and 6 for tier 1 and tier 2 respectively, which we also use initially.

Our dynamic glue limit for tier 1 is computed as the glue of 50% of the used clauses, i.e., a clause with glue up to that limit has a 50% chance of being used. The limit for tier 2 is the glue where 90% of the used clauses is reached, i.e., clauses with glue above that limit have a chance of less than 10% of being used. We compute those dynamic tier 1 and tier 2 glue limits separately for stable and focused mode. During learned clause data-base reduction they determine which clauses are kept unconditionally (tier 1) and those which are given a second chance (tier 2) as well as on which clauses vivification should focus (as described below). Note that the glue of individual clauses is updated (promoted) regularly during conflict analysis.

We further argue, that those glue limits are mostly important for unsatisfiable formulas, and usually larger in stable mode anyhow. Therefore, we actually only use those limits computed during focused mode, even in stable mode (for vivification and learned clause reduction). A more sensible selection of which limits (stable/focused) should be used in which mode is left to future work.

On hard combinatorial unsatisfiable factoring benchmarks, an intermediate version of Kissat performed much better. In that version certain redundant [9] (thus mostly learned) clauses were eagerly dropped during vivification [16] if shown implied by unit propagation (asymmetric tautologies). More specifically, redundant clauses during vivification were removed (instead of keeping them) if the negation of all but one of the literals in a clause imply exactly the remaining literal and no conflict occurred during propagation nor any other form of subsumption or shrinking applies. On these considered benchmarks eagerly dropping these clauses lead to far less accumulated redundant clauses and thus faster propagation, without increasing the number of required conflicts.

Thus we revisited vivification in Kissat once more, carefully separating all the different forms of possible subsumption and shrinking of candidate clauses and also try variable instantiation if vivifying a candidate otherwise would remain inconclusive [3], i.e., as in CaDiCaL_vivinst.

We vivify redundant tier 1 and tier 2 clauses separately, in this order, as before within a per-tier propagation (ticks) budget limited relative to propagation (ticks) used in the CDCL search loop. We now also spend some time on vivifying tier 3 clauses. At the end we continue to vivify irredundant clauses, again with a relative propagation (ticks) limit. However, we now add any left over budget from vivifying a tier to the budget of vivifying the next tier or even the irredundant clauses. Clauses not vivified due to hitting the limit are marked and tried in the next probing-inprocessing round for vivification.

Binary reason jumping was introduced in Kissat [13] to reduce time spent in conflict analysis, but appears to have negative effects in general (arguably due to skipping first unique implication points in the binary implication graph). We keep it now only enabled on formulas with many clauses (more than 100 000 clauses) with a large fraction of binary clauses (more than 99% of all clauses).

After lucky phase detection [17], ported from CaDiCaL and extended with SLUR [18] look-ahead, we perform an initial round of preprocessing, which consists of one complete run of congruence closure [12], some limited binary backbone extraction [19], and clausal equivalence

sweeping [13], followed by a new separate, simple, fast and more limited bounded variable elimination [20] with integrated forward subsumption but without syntactic nor semantic gate extraction [21]. All these preprocessors are rerun as inprocessors except for lucky phase extraction and fast variable elimination. In inprocessing the latter is replaced by the already existing more sophisticated elimination procedure (including for instance semantic gate extraction [21]).

Acknowledgements

This work was supported in part by the Austrian Science Fund (FWF) under project T-1306, W1255-N23, and S11408-N23, the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract 16ME0132, and by a gift from Intel Corporation.

References

[1] A. Haberlandt, H. Green, and M. J. H. Heule, "Effective auxiliary variables via structured reencoding," in 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy, ser. LIPIcs, M. Mahajan and F. Slivovsky, Eds., vol. 271. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 11:1–11:19.

[2] A. Haberlandt and H. Green, "SBVA-CADICAL and SBVA-KISSAT: Structured bounded variable addition," in Proc. of SAT Competition 2023 – Solver and Benchmark Descriptions, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. J. H. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2023-1. University of Helsinki, 2023, p. 18.

[3] A. Biere, M. Fleury, and F. Pollitt, "CaDiCaL_vivinst, IsaSAT, Gimsatul, Kissat, and TabularaSAT entering the SAT competition 2023," in Proc. of SAT Competition 2023 – Solver and Benchmark Descriptions, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2023-1. University of Helsinki, 2023, pp. 14–15.

[4] A. Biere and M. Fleury, "Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022," in Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2022-1. University of Helsinki, 2022, pp. 10–11.

[5] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," in Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I, ser. Lecture Notes in Computer Science, A. Gurfinkel and V. Ganesh, Eds., vol. 14681. Springer, 2024, pp. 133–152.

[6] K. Fazekas, F. Pollitt, M. Fleury, and A. Biere, "Certifying incremental SAT solving," in LPAR 2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, Port Louis, Mauritius, May 26-31, 2024, ser. EPiC Series in Computing, N. S. Bjørner, M. Heule, and A. Voronkov, Eds., vol. 100. EasyChair, 2024, pp. 321–340.

[7] K. Fazekas, A. Niemetz, M. Preiner, M. Kirchweger, S. Szeider, and A. Biere, "IPASIR-UP: user propagators for CDCL," in 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy, ser. LIPIcs, M. Mahajan and F. Slivovsky, Eds., vol. 271. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 8:1–8:13.

[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. [Online]. Available: http://ijcai.org/Proceedings/09/Papers/074.pdf

[9] 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.

[10] M. Fleury and A. Biere, "Scalable proof producing multi-threaded SAT solving with gimsatul through sharing instead of copying clauses," CoRR, vol. abs/2207.13577, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2207.13577

[11] S. Cai, X. Zhang, M. Fleury, and A. Biere, "Better decision heuristics in CDCL through local search and target phases," J. Artif. Intell. Res., vol. 74, pp. 1515–1563, 2022.

[12] A. Biere, K. Fazekas, M. Fleury, and N. Froleyks, "Clausal congruence closure," in 27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, Pune, India, ser. LIPIcs, S. Chakraborty and J.-H. R. Jian, Eds. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[13] ——, "Clausal equivalence sweeping," in Formal Methods in Computer-Aided Design, FMCAD 2024, Praque, Czech Republic, October 14-18, 2024, N. Narodytska and P. Rümmer, Eds. IEEE, 2024.

[14] N. Manthey, M. Heule, and A. Biere, "Automated reencoding of boolean formulas," in Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers, ser. Lecture Notes in Computer Science, A. Biere, A. Nahir, and T. E. J. Vos, Eds., vol. 7857. Springer, 2012, pp. 102–117.

[15] 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. A. Weaver, Eds., vol. 9340. Springer, 2015, pp. 307–323.

[16] C. Li, F. Xiao, M. Luo, F. Manyà, Z. Lü, and Y. Li, "Clause vivification by unit propagation in CDCL SAT solvers," Artif. Intell., vol. 279, 2020. [Online]. Available: https://doi.org/10.1016/j.artint.2019.103197

[17] A. Biere, "CaDiCaL at the SAT Race 2019," in Proc. of SAT Race 2019 – Solver and Benchmark Descriptions, ser. Department of Computer Science Series of Publications B, M. J. H. Heule, M. Järvisalo, and M. Suda, Eds., vol. B-2019-1. University of Helsinki, 2019, pp. 8–9.

[18] J. S. Schlipf, F. S. Annexstein, J. V. Franco, and R. P. Swaminathan, "On finding solutions for extended Horn formulas," Inf. Process. Lett., vol. 54, no. 3, pp. 133–137, 1995.

[19] N. Froleyks, E. Yu, and A. Biere, "BIG backbones," in Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, October 24-27, 2023, A. Nadel and K. Y. Rozier, Eds. IEEE, 2023, pp. 162–167.

[20] 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.

[21] M. Fleury and A. Biere, "Mining definitions in kissat with kittens," Formal Methods Syst. Des., vol. 60, no. 3, pp. 381–404, 2022.