# GIMSATUL, ISASAT, KISSAT Entering the SAT Competition 2022

Armin Biere Mathias Fleury
University of Freiburg, Germany

***Abstract*—This system description explains the features of our new multi-threaded SAT solver GIMSATUL submitted to the parallel track of the SAT Competition 2022, as well as updates to our sequential SAT solvers ISASAT, and KISSAT, submitted to the corresponding sequential tracks of the competition.**

## IMPROVED SWEEPING IN KISSAT

Already in version “KISSAT SC2021 SWEEP” submitted to the SAT Competition 2021 we supported SAT based sweeping [1] which relies on the internal embedded SAT solver KITTEN to find backbones and equivalent literals extracted from the *environment* clauses of a candidate variable in which it occurs and some of its neighbouring clauses. The algorithm was improved by eagerly substituting literals determined to be equivalent already during sweeping and more careful scheduling and rescheduling of candidate variables, particularly within the same sweeping phase.

Furthermore we sped up the common case of satisfiable queries to the embedded SAT solver KITTEN, by adding to KITTEN the following *model flipping* API function:

```
bool kitten_flip_literal (kitten *, unsigned lit);
```

It is inspired by non-recursive *model rotation* [2] used in MUS extraction. Our new model-flipping tries to flip the value of the specified literal in the last model returned by KITTEN and succeeds if the resulting new assignment still satisfies the formula. On success the model is updated. Otherwise if flipping falsifies the formula the last model is not touched.

In our application we further require that the model does not change for other literals. Thus the implementation of *model flipping* is straightforward and simply consists of just traversing the clauses watched by the literal to be flipped and checking whether there are “one-satisfied” clauses with only that literal satisfying the watched clause (assuming it was assigned to true in the last model).

We make use of this new API function by trying to flip during sweeping all literals of either the remaining backbone candidates if there are any left or the literals in candidate equivalent literal classes. If all flipped literal attempts failed we have to fall back to a more expensive actual SAT solver call to KITTEN. If flipping succeeds, which happens actually surprisingly often, we refine the backbone candidate list or the candidate equivalent literal class as usual.

In [1] we described how we use randomization of saved phases before KITTEN queries to reduce the number of necessary refinements in the common case that sweeping is mostly unsuccessful for a candidate variable. It turns out that for some benchmarks the old version “KISSAT SC2021 SWEEP” spent a substantial percentage of time during sweeping in just generating random bits for this purposes. By using all 64 bits produced by our random number generator each time instead of just one (while dropping 63 bits) and updating saved phases in a bit-parallel fashion we could remove that bottle-beck.

Supported by Austrian Science Fund (FWF) project W1255-N23 and the Inst. of Formal Methods and Verification, Johannes Kepler University Linz.

## KISSAT SC2022 BULKY

Improved sweeping above is used in all our three versions of KISSAT submitted to the SAT competition 2022. The version “KISSAT SC2022 BULKY” submitted in 2022 inherits most features of version “KISSAT SC2021 SWEEP” [1] submitted in 2021 but includes the following changes:

- added ACIDS [3] branching variable heuristics (disabled)
- added CHB [4] variable branching heuristic (but disabled by default) inspired by the success of ’kissat_mab’ [5]
- faster randomization of phases in the Kitten sub-solver
- literal flipping for faster refinement during sweeping
- disabled priority queue for variable elimination (elimination attempts follow the given fixed variable order)
- disabled by default reusing the trail during restarts
- disabled by default hyper ternary resolution
- initial local search through propagation (similar to ”warmup” runs of Donald Knuth [6] and how local search is initialized in ”ReasonLS” solvers by Shaowei Cai [7])
- actual watch replacement of true literals during unit propagation instead of just updating the blocking literal (as suggested by Norbert Manthey [8])
- fixed clause length and variable occurrences limits during variable elimination instead of dynamically increasing

## KISSAT SC2022 LIGHT AND KISSAT SC2022 HYPER

In order to focus on the most important features of KISSAT, we removed those that did not substantially improve performance on the last three competitions benchmarks. As a result of these experiments we removed the following features:

- autarky reasoning
- eager forward and backward subsumption during variable elimination (global forward subsumption only)
- caching and reusing of minima during local search
- failed literal probing
- transitive reduction of the binary implication graph
- eager subsumption of recently learned clauses
- XOR gate extraction during variable elimination

- delaying of inprocessing functions based on formula size
- vivification of *irredundant* clauses
- keeping untried elimination, backbone and vivification candidates for next inprocessing round (removed options)
- initial focused mode phase limited only by conflicts now (not as before also by ticks)

The light version also removes hyper binary resolution, enabling the use of more variables ($2^{29}-1$ instead of $2^{28}-1$).

## GIMSATUL SC2022

Our new SAT solver GIMSATUL is a parallel multi-threaded SAT solver written from scratch in C in six weeks. Its core engine follows the architecture of "KISSAT SC2022 LIGHT", even though it is missing non-chronological backtracking, on-the-fly subsumption, advanced shrinking, binary implication graph backbones, advanced definition extraction and sweeping.

The main new feature is to aggressively exchange learned clauses by sharing and reference counting instead of copying, reviving an old line of research. The solver is built on top of pthreads, but also uses C11 atomic operations as well as several lock-less fast-paths. For original clauses this already gives substantial memory savings which extends to learned clauses too and allows to generate compact DRUP proofs.

The simplification procedure implements bounded variable elimination, subsumption and equivalent literal substitution and is run up-front as preprocessing in single threaded mode and further in regular intervals after synchronizing all threads and handing over control and clauses to one single simplification thread. During search each solver thread also performs inprocessing in form of vivification and failed literal probing.

References to learned clauses of low glucose level (LBD) are immediately put in thread local pools to be exported. All exporting and importing thread combinations have exactly one pool and each pool has several slots ordered by glucose level. Threads import clauses from the slots of their pool of a randomly chosen thread, prioritized by glucose level. Except for units, which are always eagerly and completely imported, at most one clause is imported before making a decision.

For more details on "GIMSATUL SC2022" and particularly extensive experimental results on scalability and other aspects of our new solver we refer to our presentation at the workshop on *Pragmatics of SAT (POS'22)* [9].

## ISASAT

This is the first submission of the fully verified SAT solver ISASAT to the SAT Competition (and to the best of our knowledge, the first submission of a fully verified SAT solver). Since the submission to the EDA Challenge 2021 [10], we implemented only few new features, namely pure literal detection and resolution and deduplication of binary clauses. The first features is our first non-equivalence preserving transformation.

The main work went into updating the Isabelle version we are using and the version of the LLVM-based library of synthesis [11]. With the update to Isabelle2021-1, synthesis started to take *hours* for even the simplest function, so we had to replace the formalization of the solver state by a proper structure and reorganize our entire development around that.

Unrelated to our verification, we added proof logging to our solver. Remark that there is absolutely no proof of correctness of the generated proofs: The correctness theorem does not mention the proofs (and it happened during development that we forgot to print some of them leading to incorrect proofs–but the result was always correct).

The submitted sources of the SAT solver contain only files generated by Isabelle in the intermediate representation used by clang. For the complete sources (including correctness theorem and comments), refer to the `sc2022` tag in the IsaFOL repository https://bitbucket.org/isafol/isafol/src/sc2022/.

## LICENSE

All our solvers are licensed under an MIT license, with GIMSATUL available at https://github.com/arminbiere/gimsatul KISSAT at https://github.com/arminbiere/kissat and further ISASAT at https://m-fleury.github.io/isasat/isasat-release/.

## REFERENCES

[1] A. Biere, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba entering the SAT Competition 2021," in *Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions*, ser. Dept. of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2021-1. Univ. of Helsinki, 2021, pp. 10–13.

[2] J. P. M. Silva and I. Lynce, "On improving MUS extraction algorithms," in *Theory and Applications of Satisfiability Testing - 14th International Conference, SAT 2011*, ser. LNCS, K. A. Sakallah and L. Simon, Eds., vol. 6695. Springer, 2011, pp. 159–173.

[3] A. Biere and A. Fröhlich, "Evaluating CDCL variable scoring schemes," in *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, SAT 2015*, ser. LNCS, M. Heule and S. A. Weaver, Eds., vol. 9340. Springer, 2015, pp. 405–422.

[4] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Exponential recency weighted average branching heuristic for SAT solvers," in *Proc. 13th AAAI Conf. on Artificial Intelligence, AAAI 2016*, D. Schuurmans and M. P. Wellman, Eds. AAAI Press, 2016, pp. 3434–3440.

[5] M. S. Cherif, D. Habet, and C. Terrioux, "Combining VSIDS and CHB using restarts in SAT," in *27th International Conference on Principles and Practice of Constraint Programming, CP 2021*, ser. LIPIcs, L. D. Michel, Ed., vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 20:1–20:19.

[6] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 6: Satisfiability*, 1st ed. Addison-Wesley Professional, 2015.

[7] S. Cai, C. Luo, X. Zhang, and J. Zhang, "Improving local search for structured SAT formulas via unit propagation based construct and cut initialization (short paper)," in *27th International Conference on Principles and Practice of Constraint Programming, CP 2021*, ser. LIPIcs, L. D. Michel, Ed., vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 5:1–5:10.

[8] N. Manthey, "CaDiCaL modification – Watch Sat," in *Proc. of SAT Competition 2021 – 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-2021-1. University of Helsinki, 2021, pp. 28–29.

[9] M. Fleury and A. Biere, "Scalable proof-producing multi-threaded SAT solving with Gimsatul through sharing instead of copying clauses," in *Pragmatics of SAT 2022*, D. L. Berre and M. Järvisalo, Eds., 2022.

[10] M. Fleury, "CaDiCaL, Kissat, Paracooba entering the EDA Challenge 2021," 2021, submitted to the EDA Challenge 2021.

[11] P. Lammich, "Generating verified LLVM from Isabelle/HOL," in *10th International Conference on Interactive Theorem Proving, ITP 2019*, ser. LIPIcs, J. Harrison, J. O'Leary, and A. Tolmach, Eds., vol. 141. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 22:1–22:19.