Mallob with Cascading Preprocessing in the SAT Competition 2026

Dominik Schreiber Niccolò Rigi-Luperti Anna Görth
Karlsruhe Institute of Technology
Karlsruhe, Germany
dominik.schreiber@kit.edu

Markus Anders
RPTU Kaiserslautern-Landau
Kaiserslautern, Germany
anders@cs.uni-kl.de

Cayden Codel
Carnegie Mellon University
Pittsburgh, PA, United States
ccodel@cs.cmu.edu

***Abstract*—We present the 2026 version of the parallel and distributed SAT solving platform MALLOB and its distributed SAT engine MALLOBSAT to participate in this year's parallel and cloud tracks of the SAT Competition.**

***Index Terms*—SAT solving, HPC, preprocessing**

I. INTRODUCTION

MallobSat is a state-of-the-art distributed SAT solving engine tightly integrated in the distributed automated reasoning platform Mallob [9, 14, 15]. For SAT Competition 2026, we prepared a 2×2 grid of setups: two different configurations – *Quick* and *Safe* – at two different scales: shared-memory (one node, 32 cores) and distributed (100 nodes à 8 cores). Due to the rules only allowing for a single cloud submission, the *Safe* distributed setup does not compete in any official capacity.

Across all configurations, we decided to compromise on the number of solver threads spawned per hardware thread: On a machine with p physical cores and $2p$ hardware threads, we spawn $1.4 \cdot p$ solver threads (except for very large formulas, where fewer threads are spawned to save RAM [15]).

As in our last submissions [10, 12], MallobSat includes custom forks of Kissat [4] (now based on version 4.0.4) and CaDiCaL [3] (based on version 2.1.3), adding some clause import/export logic and, for the CaDiCaL, extended LRUP proof interfaces. Mallob also includes an essentially unmodified library of Lingeling (version 1.0, including YalSAT) and calls Satsuma and ImpCheck programs as external executables (see Sections II and III respectively). Mallob(Sat), Satsuma, and ImpCheck are developed by the authors themselves.

We state that our code has not been AI-tuned, nor does it feature any lines generated by autonomously acting AIs.

II. QUICK

Authors: all of the above. The *Quick* configuration is purely optimized for performance. It features an augmentation of the streamlined preprocessing we introduced last year [12, 13]. Specifically, our implementation now allows it to not only execute any number of preprocessing methods in parallel but also to easily chain them together (i.e., use the result of a preprocessor for subsequent preprocessing). We refer to this framework as *cascading preprocessing*.

Fig. 1 shows the precise configuration we use in the competition. For the most part, the preprocessor setup comes from last year's submission: Lingeling is used purely to determine (un)satisfiability for certain instances (i.e., it does not return a modified instance if no such result is found), whereas Kissat as a preprocessor does return a modified instance. This instance is then processed by a new MallobSat task that gradually replaces the MallobSat task processing the original instance. Beyond this setup, we now additionally run the latest version of the symmetry-breaking preprocessor Satsuma [1, 2, 5].

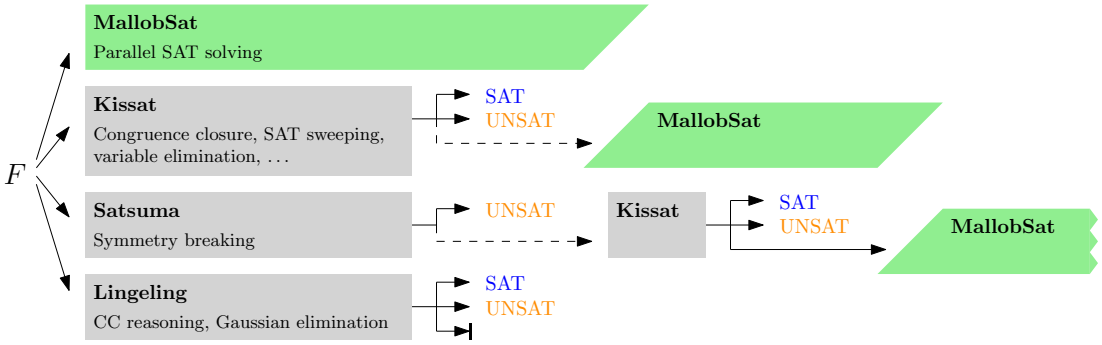

Fig. 1. Cascading preprocessing in the *Quick* setups. Some preprocessors may immediately find (un)satisfiability, which terminates the entire procedure, and/or produce a *preprocessed instance* that is then forwarded to a subsequent preprocessing stage. Dashed arrows indicate that a preprocessing chain only continues if the prior preprocessor actually reports a modified instance. As indicated by the adjusting "MallobSat" shapes, the original MallobSat task retracts its resources in favor of any subsequent MallobSat task, and the MallobSat task using Kissat's preprocessing result retracts in favor of the task using the combined result of Satsuma and Kissat.

Our Satsuma configuration uses iterative symmetry breaking: it adds unit and binary symmetry-breaking clauses using various techniques, applies CNF simplification techniques, and then repeats the procedure by detecting symmetries again on the resulting simplified formula. This process continues until no more symmetries are detected or computational limits are exceeded. Internally, Satsuma also employs the maximum-clique solver Cliquer [7] for additional ordering heuristics [5].

The formula returned by Satsuma is handed to another Kissat preprocessor – applying the same methods as the earlier Kissat preprocessing run for the original instance. We decided to chain the preprocessors in this order since Satsuma's symmetry breaking can be obstructed by Kissat's variable elimination (among other techniques) whereas Kissat's preprocessing methods seem relatively stable with respect to the modifications introduced by Satsuma.

Each MallobSat task runs an array of *search-only* Kissat solvers (with a few YalSAT local search threads in between), i.e., with no explicit portfolio of different configuration options except for trivial randomization (seeds, phases, and magnitudes of clause database reductions). Resources (i.e., processes) are balanced among current MallobSat tasks by gradually shifting resources from the older task(s) to the dominating task, over a time period proportional to the time for which the end-to-end SAT solving procedure has been running. To also adopt this mechanism in the parallel track, we split the single available compute node into two MPI processes.

III. Safe

Authors: Dominik Schreiber. The *Safe* configuration features real-time proof checking [11] in its currently most efficient implementation [16]. As opposed to our "MallobSat-ImpCheck" setup in 2024 [10], we now run an ImpCheck *confirmer* to validate the final result's fingerprint returned by the distributed checkers. As such, the setup offers very high confidence in an attained result even considering unlikely circumstances where, e.g., a logging race condition were to fabricate the string "`s SATISFIABLE`" for an unsatisfiable input.

A number of optimizations in the checking code result in higher efficiency and a smaller memory footprint compared to the checking in 2024 MallobSat-ImpCheck. Due to requiring LRUP proof logging, this setup uses CaDiCaL as a backend solver (see ref. [8]). In addition, at distributed scales, the setup employs a few *Satisfying Assignment Booster* (SAB) threads [6] in the shape of YalSAT local search threads and satisfiability-tuned Kissats (which are ensured to never contribute a clause or an unsatisfiability result to the procedure).

Acknowledgment

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de). Many thanks to the authors of the sequential SAT solvers our system is fundamentally based on.

References

[1] Markus Anders, Sofia Brenner, and Gaurav Rattan. "Satsuma: Structure-based symmetry breaking in SAT". In: *Journal of Artificial Intelligence Research* 85 (2026). DOI: https://doi.org/10.1613/jair.1.18744.

[2] Markus Anders, Cayden R. Codel, and Marijn J. H. Heule. "Orbitopal Fixing in SAT". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2026, pp. 89–109. DOI: 10.1007/978-3-032-22752-2_5.

[3] Armin Biere et al. "CaDiCaL 2.0". In: *Computer Aided Verification (CAV)*. Vol. 14681. 2024, pp. 133–152. DOI: 10.1007/978-3-031-65627-9_7.

[4] Armin Biere et al. "CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2025". In: *SAT Competition 2025: Solver and Benchmark Descriptions*. 2025.

[5] Marijn J.H. Heule Markus Anders Cayden Codel. "Simplify, Order, Break, Repeat". In: *Theory and Applications of Satisfiability Testing (SAT)*. To appear. 2026.

[6] Dawn Michaelson et al. "Producing Proofs of Unsatisfiability with Distributed Clause-Sharing SAT Solvers". In: *Journal of Automated Reasoning* 69 (2025). DOI: 10.1007/s10817-025-09725-w.

[7] Sampo Niskanen and Patric Östergård. *Cliquer user's guide, version 1.0*. Communications Laboratory, Helsinki University of Technology, Espoo, Finland, Tech. Rep. T48. 2003.

[8] Florian Pollitt, Mathias Fleury, and Armin Biere. "Faster LRAT checking than solving with CaDiCaL". In: *Theory and Applications of Satisfiability Testing (SAT)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[9] Peter Sanders and Dominik Schreiber. "Decentralized online scheduling of malleable NP-hard jobs". In: *Proc. Euro-Par*. 2022, pp. 119–135. DOI: 10.1007/978-3-031-12597-3_8.

[10] Dominik Schreiber. "MallobSat and MallobSat-ImpCheck in the SAT Competition 2024". In: *SAT Competition 2024: Solver, Benchmark and Proof Checker Descriptions* (2024), pp. 22–23.

[11] Dominik Schreiber. "Trusted Scalable SAT Solving with on-the-fly LRAT Checking". In: *Theory and Applications of Satisfiability Testing (SAT)*. 2024, 25:1–25:19. DOI: 10.4230/LIPIcs.SAT.2024.25.

[12] Dominik Schreiber, Niccolò Rigi-Luperti, and Armin Biere. "MallobSat naps in the SAT Competition 2025". In: *International SAT Competition 2025: Solver, Benchmark and Proof Checker Descriptions*. 2025.

[13] Dominik Schreiber, Niccolò Rigi-Luperti, and Armin Biere. "Streamlining Distributed SAT Solver Design". In: *Theory and Applications of Satisfiability Testing (SAT)*. 2025, 23:1–23:23. DOI: 10.4230/LIPIcs.SAT.2025.23.

[14] Dominik Schreiber, Niccolò Rigi-Luperti, and Peter Sanders. "Mallob: Scalable Automated Reasoning On Demand". In: *Computer-Aided Verification (CAV)*. To appear. 2026.

[15] Dominik Schreiber and Peter Sanders. "MallobSat: Scalable SAT Solving by Clause Sharing". In: *Journal of Artificial Intelligence Research* 80 (2024), pp. 1437–1495. DOI: 10.1613/jair.1.15827.

[16] Dominik Schreiber et al. "Real-time proof checking for distributed incremental SAT solving". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2026. DOI: 10.1007/978-3-032-22752-2_18.