

CaDiCaL 3.0

Florian Pollitt  

University of Freiburg, Germany

Mathias Fleury  

University of Freiburg, Germany

Katalin Fazekas  

TU Wien, Austria

Nils Froleyks  

Johannes Kepler University Linz, Austria

André Schidler  

University of Freiburg, Germany

Dominik Schreiber  

Karlsruhe Institute of Technology, Germany

Armin Biere  

University of Freiburg, Germany

Abstract

The propositional satisfiability (SAT) solver Kissat supports a relatively narrow feature set in favor of bare-metal performance and targeted improvements to core solving techniques, which helped it dominate the International SAT Competition since 2024. However, many applications rely on advanced SAT solver features such as incremental interaction schemes, finding direct consequences of assumed literals, or expressive proof logging that allows for real-time checking. This system description reports on how we successfully adapted Kissat’s award-winning techniques to the full-featured incremental SAT solver CaDiCaL, including clausal congruence closure, clausal equivalence sweeping, and bounded variable addition. The main challenge was to support efficient linear proof production with hints. We further extended CaDiCaL’s API to extract implied literals under assumptions and applied advanced deterministic scheduling of inprocessing based on the ticks metric for approximating cache line accesses. Experiments confirm the benefits of these efforts.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Incremental SAT, CaDiCaL, SAT Solver

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Category Tool paper

Acknowledgements Supported by Austrian Science Fund (FWF) projects T-1306, W1255-N23, and S11408-N23, the state of Baden-Württemberg through bwHPC, German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, and by a gift from Intel Corporation.

1 Introduction

This system description presents CADICAL 3.0, focusing on the most important additions on top of the widely used CADICAL 2.0 [11]. The source code of the new version 3.0 reached 64 KLOC of C++ compared to 46 KLOC for the old version 2.0. These additions improve performance considerably without compromising on usability. We further introduce the highly sought feature of allowing users to extract implied literals under assumptions.

Our efforts on CADICAL 3.0 are motivated by the success of our bare-metal SAT solver KISSAT [12] in the SAT Competition 2024, where KISSAT took first place in all categories of the main track. We attribute this success to the addition of new inprocessing algorithms,

 © F. Pollitt and M. Fleury and K. Fazekas and N. Froleyks and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:18

 Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 1** Contributions and Feature Comparison with selected SAT solvers [*] = this paper

Solver	Proofs	Algorithms			API	Robustness
	Format	Congruence[13]	Sweeping[15]	BVA[46]	Implied[60]	Scheduling
GLUCOSE [2]	LRUP[22,30]	—	—	—	✓	—
CRYPTOMINISAT [56]	FRAT[4]	XORs[58]	—	✓	—	conflicts
LINGELING [8]	DRUP[40]	simpleprobing[13]	BCE[38]	—	✓	conflicts
KISSAT [12]	DRAT[59]	✓	✓	✓	—	ticks
CADICAL 2.0 [11]	LRUP[22,50] LIDRUP[30]	—	—	—	IPASIR-UP [28,29]	conflicts
CADICAL 3.0 [*]	LRAT[22, *] LIDRUPE[*]	✓[*]	✓[*]	✓[*]	✓[*]	ticks[*]

which include a novel integrated inprocessing version of bounded variable addition (BVA) [12], improved lucky phases (actually ported from CADICAL to KISSAT and extended [12], now ported back), improved vivification [12,51], clausal equivalence sweeping [15], binary backbone extraction [34], fast bounded variable elimination during preprocessing [12], and, last but not least, clausal congruence closure [13].

In addition, KISSAT implements precise scheduling of interleaving inprocessing algorithms and more precise interleaving *stable* and *unstable* mode during search (focusing on *satisfiable* and *unsatisfiable* inputs respectively [49]). This scheduling relies on the “ticks” metric, which deterministically and precisely approximates run-time, while CADICAL 2.0 uses the more fluctuating metrics of conflicts and propagations varying greatly between the two modes.

We succeeded to lift all these distinguishing techniques available in KISSAT to CADICAL. Since bounded variable addition introduces extension variables, we describe a new API contract for incremental SAT solving that allows for extension variables. To the best of our knowledge, this makes CADICAL the first incremental proof-producing SAT solver that goes beyond resolution. Besides maintaining CADICAL’s rich incremental API, the real challenge was the continued support of producing linear resolution (LRAT) proofs [23] with hints in form of antecedents, a unique feature of CADICAL. These proofs can readily be used for interpolation, take almost no overhead to produce, and can be checked much faster than proofs produced by other state-of-the-art solvers, which (among other uses) makes CADICAL a key technology for massively parallel SAT solving with proof-backed results [47,54,55].

Table 1 lists related SAT solvers, GLUCOSE [2], CRYPTOMINISAT [56], and LINGELING [8], with similar incremental features. Below are KISSAT [12] and the previous CADICAL 2.0 [11]. They yield the new version CADICAL 3.0. The table focuses on the new features, split into four categories, with the most challenging feature LRAT **Proofs** first, second the major part on new inprocessing **Algorithms**, third the **API** extension “implied”, and last scheduling for improving **Robustness** of solving. The paper structure follows this categorization.

Besides the algorithmic improvements listed in Table 1 and on top of those already mentioned above, we have lifted in version 3.0 several additional useful techniques from KISSAT to CADICAL, including, for instance, semantic definition mining for variable elimination [33]. As these techniques require no effort on extending proof generation, porting them from KISSAT was straightforward, and we refer to KISSAT’s system descriptions [12] for details.

2 Proofs

Certifying SAT solvers through proofs has a long history [37,39]. The most common proof formats, DRUP [40] (resolution based) and its extension DRAT [59] (resolution with

■ **Table 2** Comparison of different proof formats

<i>id</i>	DIMACS [45]	DRUP [40]	LRUP [22]	LIDRUP [30] (without i-lines)
	p cnf 2 4	-1 0	5 -1 0 4 3 0	q 0
1	1 2 0	d -1 2 0	5 d 3 4 0	1 5 -1 0 4 3 0
2	1 -2 0	d -1 -2 0	6 2 0 5 1 0	d 3 4 0
3	-1 2 0	2 0	7 0 5 6 2 0	1 6 2 0 5 1 0
4	-1 -2 0	0		1 7 0 5 6 2 0
				s UNSATISFIABLE
				u 0 7 0

redundant asymmetric tautologies), date back to 2013 and have been the standard for the SAT Competition until 2023. Proof checking of DRAT proofs, however, is very expensive, even compared to solving. To address this issue, more efficient and trustworthy checking flows were developed, with LRAT [22] and GRAT [44] and with intermediate formats like FRAT [3, 4] and VeriPB [21] (a different format altogether based on pseudo-boolean proofs).

In 2023, CADICAL was extended to support logging proofs in the LRAT format or, more precisely, the LRUP subset [50]. Table 2 illustrates some of the proof formats that CADICAL 3.0 supports today. The first column shows a small formula defined in the DIMACS format [45] and the other columns present the differences between the DRUP [40] and LRUP [22]. While DRUP proofs simply list each learned and deleted clause, LRUP refers to clauses by identifiers (*id*) and provides additional hints on how learned clauses are derived.

The first proof formats for *incremental* SAT solving have been devised only recently, namely IDRUP [31] and LIDRUP [30] (see Table 2, for a truncated example). They are based on DRUP and LRUP respectively, as the soundness argument of unconditional RAT-based clause addition breaks when adding clauses incrementally [27]. LIDRUP extends LRUP (as IDRUP extends DRUP) by including the interactions of the user during the solving process (such as extracting cores). Even if RAT is incompatible, we showed [32] that these formats can be extended by variable addition, yielding LIDRUPE (Sect. 4).

To the best of our knowledge, CADICAL 3.0 is the only solver to support both detailed and incremental proofs with extension variables, requiring substantial implementation effort for clausal congruence closure (Sect. 3) and a new proof format LIDRUPE for bounded variable addition (BVA) (Sect. 4). We also support extracting implied literals (Sect. 5).

3 Clausal Congruence and Equivalence Sweeping

A recently added key feature of KISSAT is *clausal congruence closure* [14]. It detects AND, XOR, and ITE (*if-then-else*) gates encoded in the CNF formula. When two gates of the same type have the same inputs, the output variables are semantically identical and can be merged, i.e., all occurrences of one can be rewritten to the other. This simplifies the extracted gates, potentially leading to more equivalences or even units. The congruence closure algorithm exhaustively detects and merges equivalent variables, learns the corresponding clauses and simplifies the gates through rewriting and unit propagation. The challenge was producing LRAT clauses for various corner cases without updating their reasons in each step.

Clausal equivalence sweeping [15] is another approach for finding equivalences within the formula. It mirrors hybrid SAT sweeping algorithms for the equivalence checking problem of circuits but works directly on the CNF. It employs a second SAT solver, which can focus on a subset of the formula. This “nested SAT solving” is the biggest advantage over prior attempts at adapting circuit-level SAT sweeping, such as blocked clause decomposition (BCE [6, 38]).



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleyks and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:3–23:18

Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

■ **Table 3** Small example with a single BVA simplification, introducing $-5 := (1 \ \& \ 3)$

<i>id</i>	DIMACS [45]	DRAT [59]	LRAT [22]	LIDRUPE [*] (without i-lines)
	p cnf 4 4	1 5 0	5 1 5 0 0	q 0
1	1 2 0	3 5 0	6 3 5 0 0	1 5 1 5 0 0
2	2 3 0	-1 -3 -5 0	7 -1 -3 -5 0 -5 -6 0	1 6 3 5 0 0
3	3 4 0	2 -5 0	8 2 -5 0 1 2 7 0	1 7 -1 -3 -5 0 -5 -6 0
4	4 1 0	4 -5 0	9 4 -5 0 3 4 7 0	1 8 2 -5 0 1 2 7 0
		d -1 -3 -5 0	9 d 7 1 2 3 4 0	1 9 4 -5 0 3 4 7 0
		d 1 2 0		d 7 1 2 3 4 0
		d 2 3 0		s SATISFIABLE
		d 3 4 0		m 1 2 3 4 0
		d 4 1 0		

[*] = this paper

Following KISSAT, we use the lightweight incremental subsolver KITTEN [33] and ported clausal equivalence sweeping [15] accordingly. During the integration of KITTEN in CADICAL, we fixed minor issues that did not occur in KISSAT due to more restrictive usage. The integration also required adjustments to the KITTEN API to produce correct LRAT proofs. As KISSAT only supports DRUP, it can use implicit reasoning for units and equivalences. For CADICAL, we instead have to give fixed variables to KITTEN as unit clauses and further need to explicitly apply equivalence substitutions between rounds of the sweeping algorithm.

4 Bounded Variable Addition

Bounded Variable Addition (BVA), originally devised in 2012 [46], uses extended resolution (or RAT reasoning) to reencode parts of the formula. It had a big resurgence in the SAT Competition 2023 [7]. The winner SBVA-CADICAL sc2023 [35] used a BVA preprocessor in combination with CADICAL 1.5.3, which motivated us to also add BVA to KISSAT for the SAT Competition 2024 [12] as inprocessing. Porting this to CADICAL is challenging.

Unconditional RAT addition is not compatible with incremental reasoning [27]. It is compatible however, if we restrict RAT addition to *latent* witness variables, i.e., *internal* extension variables which do not occur in input nor in temporarily eliminated clauses [32]. Accordingly we extend LIDRUP to LIDRUPE and ensure correct handling of variables with a new API contract: For each new external variable the user has to ask the solver for a new variable index. Thus the solver can maintain a consecutive range of variables where user and latent variables are interleaved arbitrarily. This required various changes to API contracts and tooling, especially to our testing infrastructure with MOBICAL [11] and to proof checkers (LRAT-TRIM [20, 50], LIDRUP-CHECK [30], IDRUP-CHECK [30]).

Consider the example proofs in different formats in Table 3. We have four binary input clauses that we replace via a BVA step – introducing clauses with *ids* 5, 6, 8 and 9. Note that these new clauses resolve on the fresh variable 5 to get back clauses 1–4. As such, we essentially invert a variable elimination step on variable 5. In the proofs, there are three steps that are not plain resolution steps. The first two derived clauses (*ids* 5, 6) are *pure* on the new variable 5, meaning that 5 is not contained negatively in the formula. This pure RAT step is denoted by empty antecedents in the LRAT proof. The third derived helper clause with *id* 7 is *blocked* on the new variable (literal -5), meaning that all possible resolvents on variable 5 are tautologies. To prove entailment for this (blocked clause) RAT addition we need those negated *ids* in the LRAT proof (each one in general requiring a proof entailment by further positive *ids*). The remaining extension clauses (*ids* 8, 9) are derived by resolution.



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleys and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:4–23:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our extension LIDRUPE (last column) adopts the syntax of LRAT but needs to check that the API contract is kept, i.e. that latent variables introduced in lemma clauses (l-lines) never occur in input clauses (i-lines) nor queries (q-lines). Proof checking also ensures that latent variables used as witness literals for RAT steps are not passive (do not occur in weakened clauses, i.e., not yet restored) [26, 30, 32].

5 Implied Literals

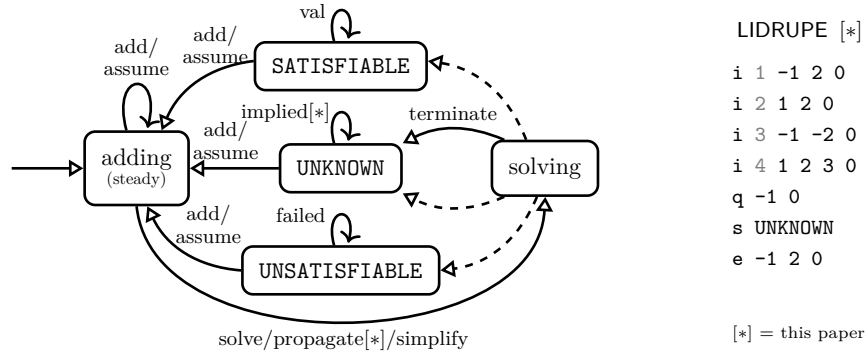


Figure 1 Simplified internal state-transition-graph of a SAT solver (on the left) and a proof produced by a call to `propagate` followed by an `implied` call (on the right).

In many use cases of SAT solvers, e.g., MaxSAT preprocessing, SMT preprocessing, or backbone extraction [18], it is important to compute “immediate” consequences of a set of literals with respect to a formula, rather than searching for a complete model. To this extent incremental SAT solvers like MINISAT [25], GLUCOSE [2], or the PYSAT interface [41, 42], typically provide users with a way to run limited searches (i.e., only *propagate*) on a given set of assumptions. As of yet, CADICAL supported this only through a laborious way through IPASIR-UP [28]. CADICAL 3.0 now features a `propagate` function in its external interface.

Fig. 1 shows the adapted main internal state graph of CADICAL 3.0 with the new propagation functionality. After initialization, users can add clauses (via `add`) and assume literals (via `assume` as in [25]) to define their problem. Then, they can trigger the (potentially restricted) search of the solver by calling `solve`, `propagate`, or `simplify`. Each query can return either **SATISFIABLE**, **UNSATISFIABLE**, or **UNKNOWN** (see dashed edges in Fig. 1).

Based on the returned answer, users can query either the model or the core through the standard `val` resp. `failed` functions [5]. In case of an **UNKNOWN** result, starting from CADICAL 3.0, users can obtain implied literals via a new `implied` call (as in LINGELING [8]). Note, that this “implied” set of literals is in general only a subset of all logically entailed literals (in contrast to backbones [18]), i.e., the solver ignores eliminated variables and clauses.

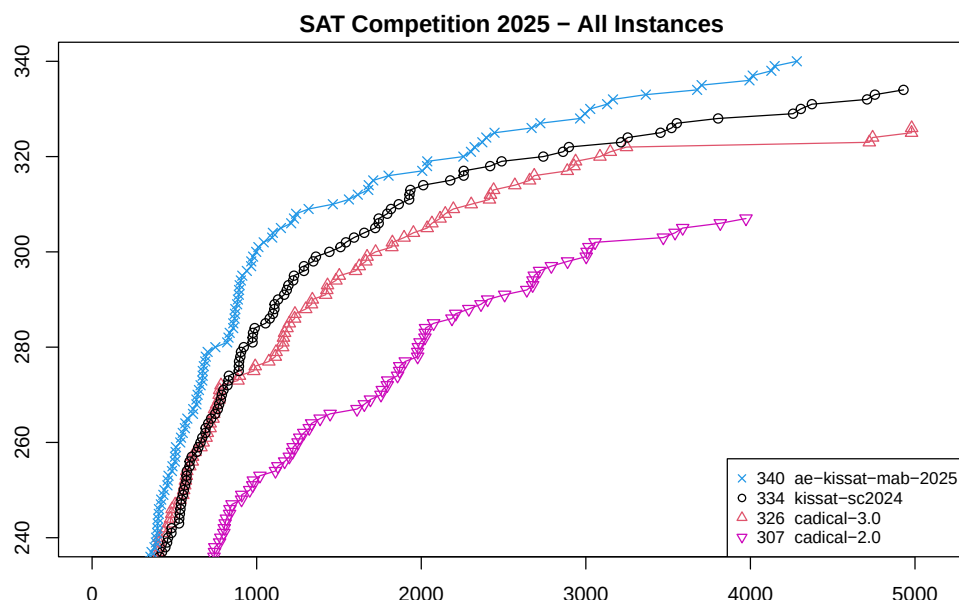
We extended the LIDRUPE proof format accordingly: Instead of restricted tracing to the new propagation calls, we standardized handling of queries returning **UNKNOWN**, which also applies to preprocessing calls and asynchronously terminated search calls. As Fig. 1 illustrates, when neither a model nor a contradiction is found by the solver, LIDRUPE reports an “s UNKNOWN” line and propagated literals start with the prefix “e” (short for *entrained*).

176 Tuning SAT solvers is non-trivial [9]. For instance, CADICAL (and KISSAT) have many
177 different search heuristics, including two different search modes, alternating *stable* and
178 *unstable* search after the ideas of Chanseok Oh [49]. Additionally, they employ a wide range
179 of pre- and inprocessing techniques [19]. Usefulness and run-time cost of these techniques
180 vary wildly between benchmarks and are further hard to predict. To control and ultimately
181 improve robustness, i.e., reduce solving time variance, we want a precise deterministic metric
182 to restrict the time spent on different techniques and in different search modes. Classical
183 metrics in SAT solving are based on the number of conflicts encountered or the number of
184 propagations (number of unit-propagations of assigned variables).

185 However, these metrics do not correlate well with actual time spent. For instance during
186 unstable search with very frequent restarts the propagation rate (per second) can dramatically
187 decrease compared to the stable mode with rare restarts, and accordingly for the conflict rate.
188 Different clause variable ratios or different ranges of LBD levels [1] have similar influences.

189 Like many other solvers (*cf.* Tbl. 1), our previous version CADICAL 2.0 used conflicts
190 to schedule different modes and inprocessing techniques. It further relied on counting
191 propagations to limit the effort spent within individual inprocessing techniques. As argued
192 above (and shown in our experiments) this leads to substantial variations in time allocated
193 to different techniques and makes the split between stable/unstable search less balanced.

194 To counter this effect, KISSAT introduced *ticks* [16], as an approximation to the number
195 of cache line accesses, focusing on the hotspot of execution (usually unit-propagation).
196 Scheduling stable and unstable search based on ticks balances the ratio of time spent in the
197 two modes, bringing it closer to 50%. The ticks metric is also used to limit time spent on
198 inprocessing versus solving. It was also effective in making parallel SAT solvers deterministic
199 in the DPS framework [48]. We further argue that counting cache line accesses instead of



■ **Figure 2** We compare uncertified solving of the winners of the SAT Competitions from 2024 and 2025, i.e., KISSAT SC2024 [12] and AE KISSAT MAB SC2025 [24], against CADICAL 2.0 [11] and the new CADICAL 3.0 on the 400 benchmarks from the SAT Competition 2025 without proofs.

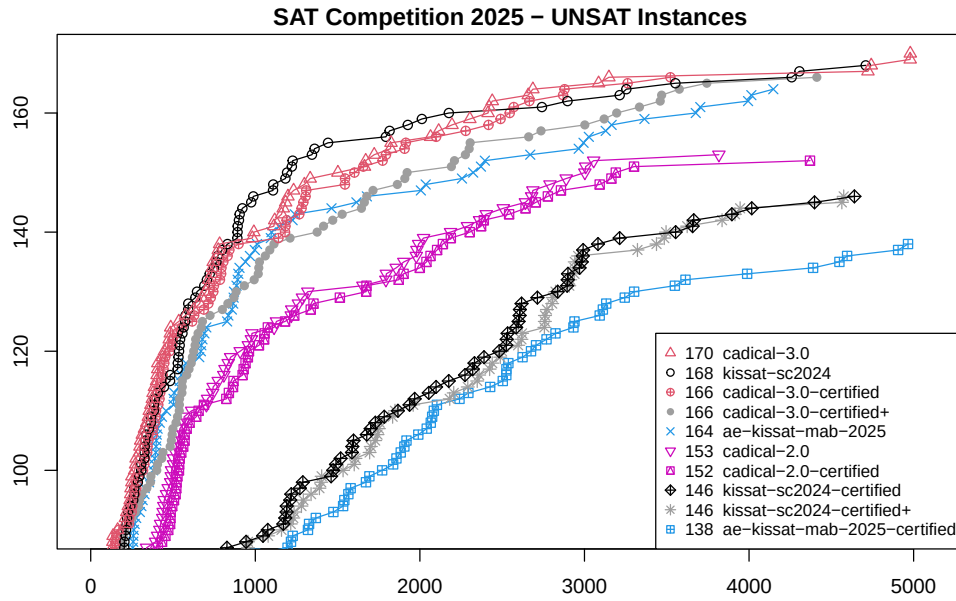


Figure 3 We report total time for (1) solving without proof production, (2) “certified” solving with the unverified proof checkers LRAT-TRIM [20] resp. DPR-TRIM [36], and (3) “certified with verified checker” solving (certified+) with CAKE-LPR [57]. Certification, in both cases, includes solving, proof production and proof checking, and additionally for `kissat-sc2024-certified+` the time needed to elaborate DRAT proofs from KISSAT into LRAT proofs before checking with CAKE-LPR.

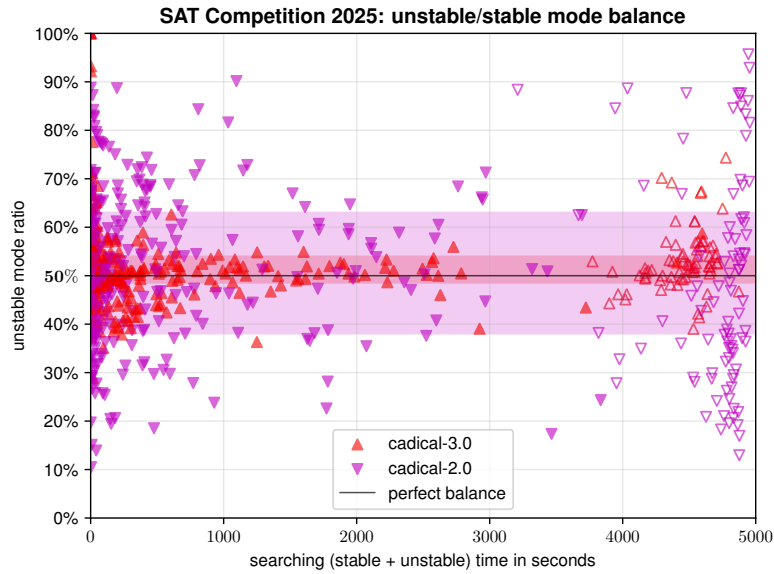
the memory accesses (Knuth’s “mems” [43]) is more accurate. For instance during unit-propagation of binary clauses the other literal is stored in the watch list. Data of consecutively propagated binary clauses resides in the same cache line, and hence can be accessed at no cost, in contrast to dereferencing large clauses, distributed randomly in memory.

The new version of CADICAL 3.0 computes and makes use of ticks during search and most inprocessing techniques. We first added ticks counting to obvious hotspots, then determined outliers and iteratively adapted ticks computation through profiling. As a side effect, we also achieved a better understanding of algorithm costs, e.g., scheduling in vivification turned out to be more expensive than expected and had to be limited for large benchmarks. Ticks are also used to ensure even distribution between stable and unstable modes: the first (unstable) mode phase is limited by conflicts, ticks spent are measured and scaled to limit the time spent in subsequent stable and unstable search mode phases.

7 Experiments

Our experiments were conducted on 16-core AMD EPYC 7343 CPUs running at 3.30 GHz and we used all 400 problems from the SAT Competition 2025. For individual runs we enforced a time limit of 5000 seconds. Experimental data is available on Zenodo [52].

In Fig. 2 we compare CADICAL 3.0 with CADICAL 2.0 and the two variants of KISSAT winning the last two SAT competitions in 2024 and 2025. Indeed CADICAL 3.0 performs much better than CADICAL 2.0, but lacks slightly behind the two considered variants of KISSAT. Focusing on unsatisfiable instances in Fig. 3, however, it even wins. We further show the results of running the considered CADICAL and KISSAT solvers with proof production and then proof checking (certified) using the proof checker LRAT-TRIM [20] or DPR-TRIM [36] and, in addition, with the verified proof checker CAKE-LPR [57] (certified+).



■ **Figure 4** Solving is split into search and simplification through pre- and inprocessing. Furthermore, searching is split into *stable* and *unstable* mode. An important goal of using ticks is to balance the time spent in these two modes. Thus we compare the relative *search-only* time spent in these modes for CADICAL 2.0 (magenta triangles pointing downwards – using conflicts) and CADICAL 3.0 (red triangles pointing upwards – using ticks). Each triangle corresponds to one of the 400 SAT Competition 2025 benchmarks. The x -coordinate gives the total time of the solver spent in both modes and the y -coordinate the fraction spent in unstable mode, i.e., on the 50% line the same time is spent in each mode. Unsolved instances are non-filled triangles. For each solver version, we highlight the region between the 25th and 75th percentiles, containing half of all instances.

223 To use CAKE-LPR for KISSAT we have to elaborate its DRAT proofs into LRAT with
 224 DPR-TRIM first, as only CADICAL supports LRAT directly, while KISSAT can only produce
 225 DRAT proofs. The results confirm, that LRAT proof checking is instant, while checking DRAT
 226 proofs can take more than a factor of 100 the time needed for solving.

227 Furthermore, we evaluate robustness of our new approach using ticks, with respect to
 228 the alternation of *unstable* and *stable* search (Fig. 4). With ticks mode alternation is more
 229 balanced, even though we can observe a slight imbalance towards unstable mode, especially
 230 as CADICAL starts in unstable mode. There are clearly fewer extreme cases.

231 Lastly we run CADICAL 3.0 on incremental benchmarks with our bounded model
 232 checker CAMICAL [27] on all the AIGER benchmarks from the Hardware Model Checking
 233 Competition 2025 [17, 53]. Using CADICAL 2.0 as backend SAT solver to CAMICAL solved
 234 141 507 bounds in 5 000 s, while the new version solves 141 931 (*cf.* Fig. 7 in the Appendix),
 235 underlining that incremental performance improved slightly with CADICAL 3.0 on one of
 236 the most important industrial applications of incremental SAT solving.

237 8 Conclusion

238 We successfully extended CADICAL with clausal congruence closure, clausal equivalence
 239 sweeping, and bounded variable addition, while continuing to support efficient linear proof
 240 production with hints. Scheduling based on ticks makes the solver more robust. Further
 241 improving incremental solving and closing the gap between CADICAL and KISSAT on
 242 *satisfiable* instances (*cf.* Fig. 5 in the Appendix) is left to future work.



© F. Pollitt and M. Fleury and K. Fazekas and N. Froylys and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:8–23:18

Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- 1 Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc. URL: <http://ijcai.org/Proceedings/09/Papers/074.pdf>.
- 2 Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. doi:10.1142/S0218213018400018.
- 3 Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 59–75. Springer, 2021. doi:10.1007/978-3-030-72016-2_4.
- 4 Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/LMCS-18(2:3)2022.
- 5 Tomás Balyo, Armin Biere, Markus Iser, and Carsten Sinz. SAT Race 2015. *Artif. Intell.*, 241:45–65, 2016.
- 6 Tomás Balyo, Andreas Fröhlich, Marijn Heule, and Armin Biere. Everything you always wanted to know about blocked sets (but were afraid to ask). In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *LNCS*, pages 317–332. Springer, 2014. doi:10.1007/978-3-319-09284-3_24.
- 7 Tomas Balyo, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2023: Solver, Benchmark and Proof Checker Descriptions*. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, Finland, 2023.
- 8 Armin Biere. Lingeling, Plingeling, PicoSAT and Precosat at SAT Race 2010. Technical Report 10/1, Johannes Kepler University Linz, FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2010. doi:10.350/fmvtr.2010-1.
- 9 Armin Biere. Lingeling essentials, a tutorial on design and implementation aspects of the SAT solver Lingeling. In Daniel Le Berre, editor, *POS-14. Fifth Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, page 88. EasyChair, 2014. URL: [/publications/paper/xJs](#), doi:10.29007/jhd7.
- 10 Armin Biere. DiMoCheck DIMACS model checker, Last access, February 2025. Source code. URL: <https://github.com/arminbiere/dimocheck>.
- 11 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. Cadical 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.
- 12 Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.
- 13 Armin Biere, Katalin Fazekas, Mathias Fleury, and Nils Froleyks. Clausal congruence closure. In Supratik Chakraborty and Jie-Hong Roland Jiang, editors, *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India,*



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleyks and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:9–23:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- 295 volume 305 of *LIPIcs*, pages 6:1–6:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik,
296 2024. doi:10.4230/LIPICS.SAT.2024.6.
- 297 **14** Armin Biere, Katalin Fazekas, Mathias Fleury, and Nils Froleys. Clausal Congruence Closure.
298 In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT*
299 *2024)*, volume 305 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:25,
300 2024. doi:10.4230/LIPICS.SAT.2024.6.
- 301 **15** Armin Biere, Katalin Fazekas, Mathias Fleury, and Nils Froleys. Clausal equivalence
302 sweeping. In Nina Narodytska and Philipp Rümmer, editors, *Proceedings 24th International*
303 *Conference on Formal Methods in Computer-Aided Design (FMCAD’24)*, pages 236–241. TU
304 Wien Academic Press, 2024. doi:10.34727/2024/isbn.978-3-85448-065-5_29.
- 305 **16** Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximilian Heisinger. CaDiCaL, Kissat,
306 Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo,
307 Nils Froleys, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors,
308 *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of
309 *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.
- 310 **17** Armin Biere, Nils Froleys, and Mathias Preiner. Hardware model checking competition
311 2025. In Ahmed Irfan and Daniela Kaufmann, editors, *Proceedings of the 25th Conference on*
312 *Formal Methods in Computer-Aided Design, FMCAD 2025, Menlo Park, CA, USA, October*
313 *6-10, 2025*. TU Wien Academic Press, 2025. URL: [https://doi.org/10.34727/2025/isbn.](https://doi.org/10.34727/2025/isbn.978-3-85448-084-6_6)
314 [978-3-85448-084-6_6](https://doi.org/10.34727/2025/isbn.978-3-85448-084-6_6), doi:10.34727/2025/ISBN.978-3-85448-084-6_6.
- 315 **18** Armin Biere, Nils Froleys, and Wenxi Wang. CadiBack: Extracting backbones with CaDiCaL.
316 In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory*
317 *and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume
318 271 of *LIPIcs*, pages 3:1–3:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
319 doi:10.4230/LIPICS.SAT.2023.3.
- 320 **19** Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In Armin Biere,
321 Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*,
322 volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391 – 435. IOS Press,
323 2nd edition edition, 2021.
- 324 **20** Armin Biere and Florian Pollitt. LRAT trimmer, Last access, February 2026. Source code.
325 URL: <https://github.com/arminbiere/lrat-trim>.
- 326 **21** Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance
327 and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence*
328 *Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI ’22*.
- 329 **22** Luís Cruz-Filipe, Marijn J. H. Heule, Jr. Hunt, Warren A., Matt Kaufmann, and Peter
330 Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated*
331 *Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg,*
332 *Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *LNCS*, pages 220–236. Springer,
333 2017. doi:10.1007/978-3-319-63046-5_14.
- 334 **23** Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-
335 Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction*
336 *– CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.
- 337 **24** Hang Ding, Mao Luo, Chu-Min Li, Shunwei Li, Runyao Chen, Caiquan Xiong, and Xinyun Wu.
338 A self-optimizing framework for sat solvers via population evolution and large language model
339 collaboration. In Cayden Codel, Katalin Fazekas, Marijn Heule, and Markus Iser, editors,
340 *Proceedings of SAT Competition 2025: Solver and Benchmark Descriptions*, Proceedings of
341 SAT Competitions, pages 15–17. TU Wien Academic Press, 2025. doi:10.34726/10379.
- 342 **25** Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and
343 Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International*
344 *Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*,
345 volume 2919 of *LNCS*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleys and A. Schidler and D. Schreiber and A. Biere;
licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:10–23:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- 346 26 Katalin Fazekas. *On SAT-based Solution Methods for Computational Problems*. PhD thesis,
347 Informatik, Johannes Kepler University Linz, 2020.
- 348 27 Katalin Fazekas, Armin Biere, and Christoph Scholl. Incremental inprocessing in SAT
349 solving. In Mikoláš Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability*
350 *Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12,*
351 *2019, Proceedings*, volume 11628 of *LNCS*, pages 136–154. Springer, 2019. doi:10.1007/
352 978-3-030-24258-9_9.
- 353 28 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and
354 Armin Biere. IPASIR-UP: user propagators for CDCL. In Meena Mahajan and Friedrich
355 Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability*
356 *Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, volume 271 of *LIPICs*, pages 8:1–8:13.
357 Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.8.
- 358 29 Katalin Fazekas, Aina Niemetz, Mathias Preiner, Markus Kirchweger, Stefan Szeider, and
359 Armin Biere. Satisfiability modulo user propagators. *J. Artif. Intell. Res.*, 81:989–1017, 2024.
360 doi:10.1613/JAIR.1.16163.
- 361 30 Katalin Fazekas, Florian Pollitt, Mathias Fleury, and Armin Biere. Certifying incremental
362 SAT solving. In Nikolaj S. Bjørner, Marijn Heule, and Andrei Voronkov, editors, *LPAR*
363 *2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and*
364 *Reasoning, Port Louis, Mauritius, May 26-31, 2024*, volume 100 of *EPiC Series in Computing*,
365 pages 321–340. EasyChair, 2024. doi:10.29007/PDCC.
- 366 31 Katalin Fazekas, Florian Pollitt, Mathias Fleury, and Armin Biere. Incremental proofs for
367 bounded model checking. In Wolfgang Kunz, editor, *27th GMM/ITG/GI Workshop on Methods*
368 *and Description Languages for Modelling and Verification of Circuits and Systems (MBMV’24),*
369 *Kaiserslautern, Germany*, volume 314 of *ITG-Fachberichte*, pages 133–143. VDE Verlag, 2024.
- 370 32 Katalin Fazekas, Florian Pollitt, Mathias Fleury, and Armin Biere. Incremental inprocessing
371 rules beyond resolution. In Jochen Hoenicke, Mikoláš Janota, Aina Niemetz, and Sophie
372 Tourret, editors, *Proceedings 16th Pragmatics of SAT International Workshop (POS’25),*
373 *co-located with the 28th International Conference on Theory and Applications of Satisfiability*
374 *Testing (SAT’23)*, number 4008 in *CEUR Workshop Proceedings*, pages 190–200, 2025. URL:
375 <https://ceur-ws.org/Vol-4008>.
- 376 33 Mathias Fleury and Armin Biere. Mining definitions in Kissat with Kittens. *Formal Methods*
377 *Syst. Des.*, 60(3):381–404, 2022. doi:10.1007/S10703-023-00421-2.
- 378 34 Nils Froleyks, Emily Yu, and Armin Biere. BIG backbones. In Alexander Nadel and
379 Kristin Yvonne Rozier, editors, *Formal Methods in Computer-Aided Design, FMCAD 2023,*
380 *Ames, IA, USA, October 24-27, 2023*, pages 162–167. IEEE, 2023. doi:10.34727/2023/ISBN.
381 978-3-85448-060-0\24.
- 382 35 Andrew Haberlandt and Harrison Green. SBVA-CaDiCaL and SBVA-Kissat: Structured
383 bounded variable addition. In Tomas Balyo, Nils Froleyks, Marijn J. H. Heule, Markus Iser,
384 Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2023 – Solver and*
385 *Benchmark Descriptions*, volume B-2023-1 of *Department of Computer Science Report Series*
386 *B*, page 18. University of Helsinki, 2023.
- 387 36 Marijn Heule. Checkr of DPR proof files, Last access, April 2024. Source code. URL:
388 <https://github.com/marijnheule/dpr-trim>.
- 389 37 Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van
390 Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336
391 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021.
392 doi:10.3233/FAIA200998.
- 393 38 Marijn J. H. Heule and Armin Biere. Blocked clause decomposition. In Kenneth L. McMillan,
394 Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence,*
395 *and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December*
396 *14-19, 2013. Proceedings*, volume 8312 of *LNCS*, pages 423–438. Springer, 2013. doi:10.1007/
397 978-3-642-45221-5_29.



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleyks and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:11–23:18

Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- 398 **39** Marijn J. H. Heule and Armin Biere. Proofs for satisfiability problems. In *All about Proofs,*
399 *Proofs for All (APPA)*, volume 55 of *Math. Logic and Foundations*. College Pub., 2015.
- 400 **40** Marijn J.H. Heule, Warren A. Hunt, and Nathan Wetzler. Trimming while checking clausal
401 proofs. In *2013 Formal Methods in Computer-Aided Design*, pages 181–188, 2013. doi:
402 10.1109/FMCAD.2013.6679408.
- 403 **41** Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A Python toolkit
404 for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger,
405 editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International*
406 *Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford,*
407 *UK, July 9-12, 2018, Proceedings*, volume 10929 of *LNCS*, pages 428–437. Springer, 2018.
408 doi:10.1007/978-3-319-94144-8_26.
- 409 **42** Alexey Ignatiev, Zi Li Tan, and Christos Karamanos. Towards universally accessible SAT
410 technology. In *SAT*, pages 4:1–4:11, 2024. URL: <https://doi.org/10.4230/LIPICS.SAT.2024.16>,
411 doi:10.4230/LIPICS.SAT.2024.16.
- 412 **43** Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 4: Generating All*
413 *Trees—History of Combinatorial Generation (Art of Computer Programming)*. Addison-Wesley
414 Professional, 2006.
- 415 **44** Peter Lammich. The GRAT tool chain - efficient (UN)SAT certificate checking with formal
416 correctness guarantees. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of*
417 *Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia,*
418 *August 28 - September 1, 2017, Proceedings*, volume 10491 of *LNCS*, pages 457–463. Springer,
419 2017. doi:10.1007/978-3-319-66263-3_29.
- 420 **45** Daniel Le Berre, Olivier Roussel, and Laurent Simon. SAT competition 2009: Bench-
421 mark submission guidelines. [https://web.archive.org/web/20190325181937/https://www.](https://web.archive.org/web/20190325181937/https://www.satcompetition.org/2009/format-benchmarks2009.html)
422 [satcompetition.org/2009/format-benchmarks2009.html](https://web.archive.org/web/20190325181937/https://www.satcompetition.org/2009/format-benchmarks2009.html). Accessed: 2024-01-15.
- 423 **46** Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean
424 formulas. In Armin Biere, Amir Nahir, and Tanja Vos, editors, *Hardware and Software:*
425 *Verification and Testing*, pages 102–117, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- 426 **47** Dawn Michaelson, Dominik Schreiber, Marijn JH Heule, Benjamin Kiesl-Reiter, and Michael W
427 Whalen. Producing proofs of unsatisfiability with distributed clause-sharing SAT solvers.
428 *Journal of Automated Reasoning*, 69(2):12, 2025. doi:10.1007/s10817-025-09725-w.
- 429 **48** Hidetomo Nabeshima, Tsubasa Fukiage, Yuto Obitsu, Xiao-Nan Lu, and Katsumi Inoue. Dps:
430 a framework for deterministic parallel sat solvers. EasyChair Preprint 8553, EasyChair, 2022.
- 431 **49** Chanseok Oh. Between SAT and UNSAT: the fundamental difference in CDCL SAT. In
432 *Theory and Applications of Satisfiability Testing—SAT 2015—18th International Conference,*
433 *Austin, TX, USA, September 24-27, 2015, Proceedings*, pages 307–323, 2015. doi:10.1007/
434 978-3-319-24318-4_23.
- 435 **50** Florian Pollitt, Mathias Fleury, and Armin Biere. Faster LRAT checking than solving with
436 cadical. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on*
437 *Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy,*
438 volume 271 of *LIPICs*, pages 21:1–21:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik,
439 2023. doi:10.4230/LIPICS.SAT.2023.21.
- 440 **51** Florian Pollitt, Mathias Fleury, Armin Biere, Karem Sakallah, Marijn Heule, Jiawei Chen,
441 and Yonathan Fisseha. Revisiting clause vivification. In Jochen Hoenicke, Mikoláš Janota,
442 Aina Niemetz, and Sophie Turret, editors, *Proceedings 16th Pragmatics of SAT International*
443 *Workshop (POS’25), co-located with the 28th International Conference on Theory and Applica-*
444 *tions of Satisfiability Testing (SAT’23)*, number 4008 in *CEUR Workshop Proceedings*, pages
445 153–167, 2025. URL: <https://ceur-ws.org/Vol-4008>.
- 446 **52** Florian Pollitt, Mathias Fleury, Katalin Fazekas, Nils Froleyks, André Schidler, Dominik
447 Schreiber, and Armin Biere. Artifact cadical 3.0 paper, March 2026. doi:10.5281/zenodo.
448 18924223.



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleyks and A. Schidler and D. Schreiber and A. Biere;
licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:12–23:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- 449 **53** Mathias Preiner, Nils Froleys, and Armin Biere. HWMCC'25 benchmarks and results,
450 October 2025. doi:10.5281/zenodo.17428464.
- 451 **54** Dominik Schreiber. Trusted scalable SAT solving with on-the-fly LRAT checking. In *Theory*
452 *and Applications of Satisfiability Testing (SAT)*, pages 25:1–25:19. Schloss Dagstuhl – Leibniz-
453 Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SAT.2024.25.
- 454 **55** Dominik Schreiber, Katalin Fazekas, Mathias Fleury, and Armin Biere. Real-time proof
455 checking for distributed incremental SAT solving. In *Tools and Algorithms for the Construction*
456 *and Analysis of Systems (TACAS)*, 2026. To appear. URL: [https://satres.kikit.kit.edu/papers/](https://satres.kikit.kit.edu/papers/2026-tacas-distrincproof.pdf)
457 [2026-tacas-distrincproof.pdf](https://satres.kikit.kit.edu/papers/2026-tacas-distrincproof.pdf).
- 458 **56** Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic
459 problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing*
460 *- SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3,*
461 *2009. Proceedings*, volume 5584 of *LNCS*, pages 244–257. Springer, 2009. doi:10.1007/
462 978-3-642-02777-2_24.
- 463 **57** Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation
464 redundancy checking in CakeML. In Jan Friso Groote and Kim Guldstrand Larsen, editors,
465 *Tools and Algorithms for the Construction and Analysis of Systems - 27th International*
466 *Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and*
467 *Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1,*
468 *2021, Proceedings, Part II*, volume 12652 of *LNCS*, pages 223–241. Springer, 2021. doi:
469 10.1007/978-3-030-72013-1_12.
- 470 **58** Yong Kiam Tan, Jiong Yang, Mate Soos, Magnus O. Myreen, and Kuldeep S. Meel. Formally
471 certified approximate model counting. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer*
472 *Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July*
473 *24-27, 2024, Proceedings, Part I*, volume 14681 of *LNCS*, pages 153–177. Springer, 2024.
474 doi:10.1007/978-3-031-65627-9_8.
- 475 **59** Nathan Wetzler, Marijn J. H. Heule, and Jr. Hunt, Warren A. DRAT-trim: Efficient checking
476 and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory*
477 *and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as*
478 *Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*,
479 volume 8561 of *LNCS*, pages 422–429. Springer, 2014. doi:10.1007/978-3-319-09284-3_31.
- 480 **60** Akihisa Yamada, Armin Biere, Cyrille Artho, Takashi Kitamura, and Eun-Hye Choi. Greedy
481 combinatorial test case generation using unsatisfiable cores. In David Lo, Sven Apel, and
482 Sarfraz Khurshid, editors, *Proceedings of the 31st IEEE/ACM International Conference on*
483 *Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, pages 614–624.
484 ACM, 2016. doi:10.1145/2970276.2970335.



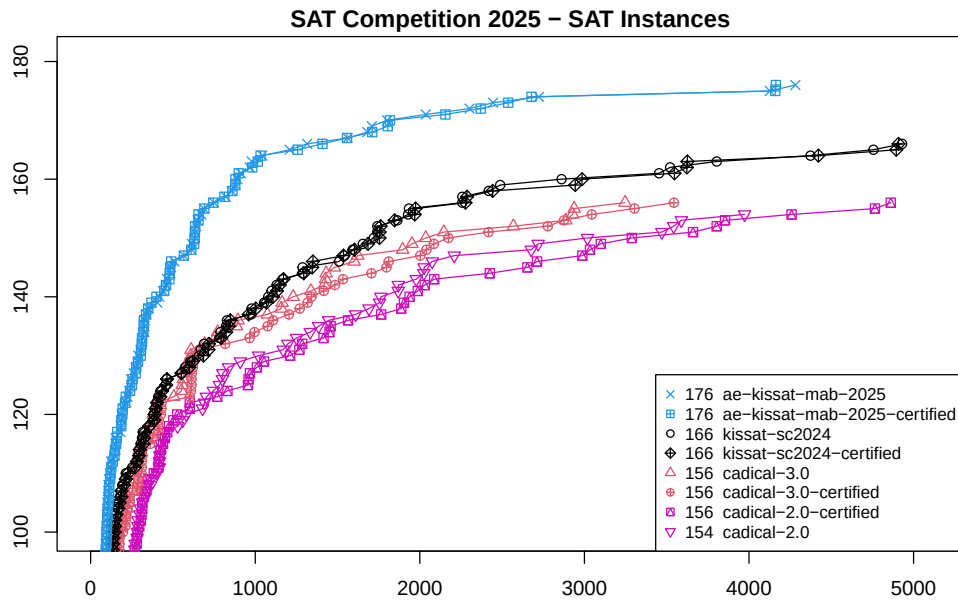


Figure 5 Similar to Fig. 3 focusing on satisfiable instances. This includes both solving and checking time of the produced model using our new model checker DiMoCHECK [10], which takes the original CNF in DIMACS format and the model produced by the SAT solver (in SAT competition output format) as inputs and checks that the model satisfies all clauses in the given CNF. Certification only induces a small overhead. For CADICAL 2.0 however this experiment revealed that enabling LRAT proof tracing changed solving behavior slightly due to certain mark flags unintentionally having been used for both proof production and bumping variables. This resulted in “cadical-2.0-certified” to be slightly faster than the “cadical-2.0”. In CADICAL 3.0 this feature interaction is not present.



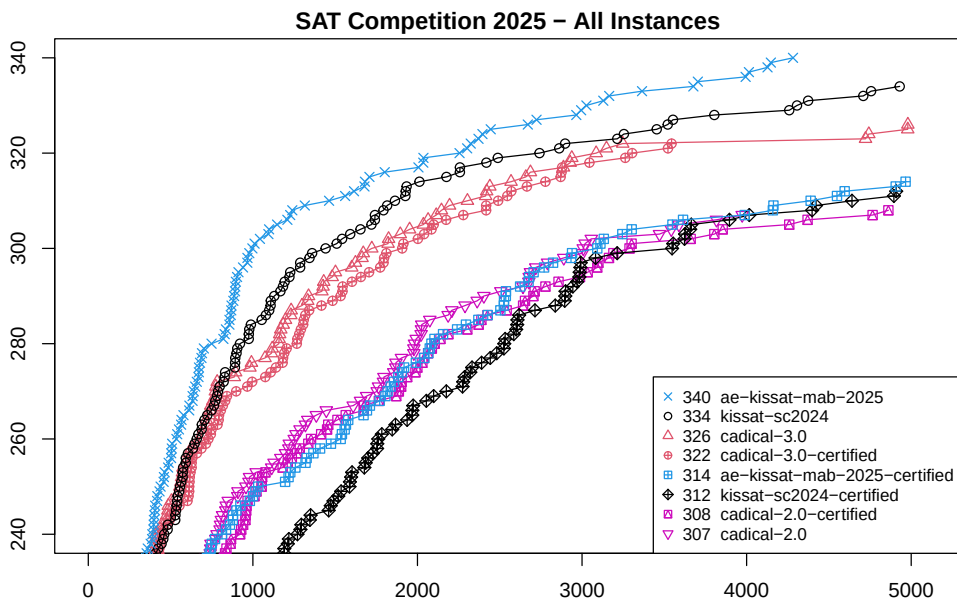
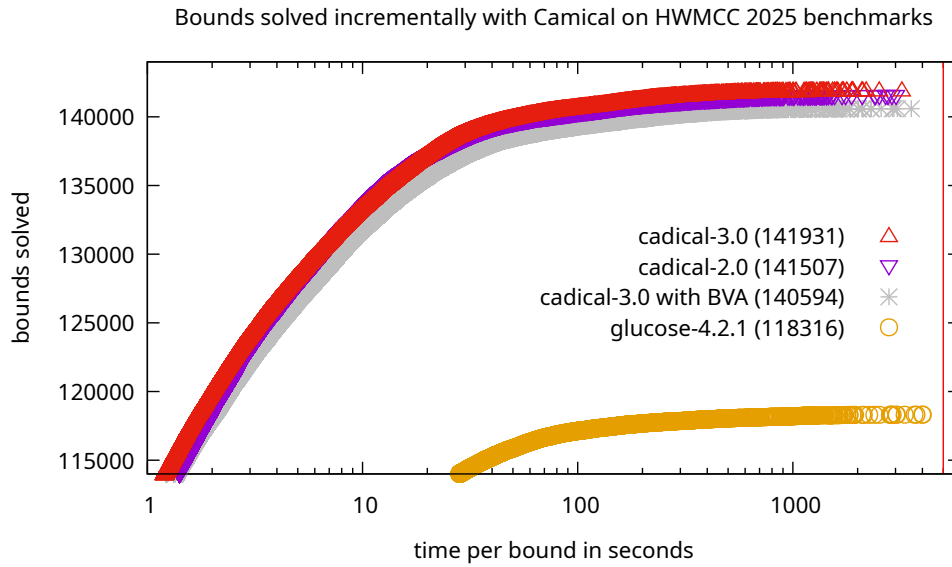


Figure 6 As Fig. 3+5 but with both unsatisfiable and satisfiable benchmarks. This includes proof resp. model production, proof checking for unsatisfiable resp. model checking for satisfiable solver results. For **certified** satisfiable instances we use our model checker DIMOCHECK [10] and for **certified** unsatisfiable instances our new version of LRAT-TRIM [20] for CADICAL and rely on DPR-TRIM [36] for checking DRAT proofs generated by KISSAT. No formally verified model checker was used.



■ **Figure 7** This plot shows the number of solved bounds (y -axis) by CAMiCAL [27] over the time needed for solving a bound incrementally (x -axis) using different versions of CADiCAL on the benchmarks from the latest Hardware Model Checking Competition 2025 [17, 53]. This is the same experimental design as in [27] targeting bounded model checking of hardware circuits, one of the most important industrial application of incremental SAT solving. In this experiment the best backend SAT solver is CADiCAL 3.0 with blocked clause addition (BVA) disabled, a feature which does not seem to be beneficial on hardware problems. We further disabled an experimental new version of incrementally applying lucky phases (which did not exist in CADiCAL 2.0) due to a performance regression we observed. Using the old version CADiCAL 2.0 as SAT solver backend as well as enabling BVA in the new version degrades performance slightly. All CADiCAL versions still solve more bounds per seconds, i.e., go “deeper”, than using the latest version 4.2.1 of GLUCOSE [2] as backend SAT solver, which is another SAT solver often used in incremental SAT solving too.

```

1  #include "cadical.hpp"
2  #include <cassert>
3
4  CaDiCaL::Solver *solver = new CaDiCaL::Solver ();
5
6  int main () {
7      solver->set ("binary", 0);
8      solver->set ("lidrup", 1);
9      solver->trace_proof ("propagate.lidrup");
10     solver->set ("flushproof", 1);
11
12     enum { TIE = 1, SHIRT = 2, HAT = 3 };
13     solver->declare_more_variables (3);
14
15     solver->add (-TIE), solver->add (SHIRT), solver->add (0);
16     solver->add (TIE), solver->add (SHIRT), solver->add (0);
17     solver->add (-TIE), solver->add (-SHIRT), solver->add (0);
18     solver->add (TIE), solver->add (SHIRT), solver->add (HAT), solver->add (0);
19
20     solver->assume (-TIE);
21     int res = solver->propagate ();
22     assert (res == 0);
23
24     std::vector<int> implicants;
25     solver->implied (implicants);
26     assert (implicants.size() > 0 && implicants.size() < 3);
27
28     solver->assume (TIE);
29     res = solver->propagate ();
30     assert (res == 20);
31     assert (solver->failed(TIE));
32
33     solver->assume (SHIRT);
34     solver->assume (HAT);
35     res = solver->propagate ();
36     assert (res == 10);
37     assert (solver->val(TIE) == -1);
38
39     return 0;
40 }

```

```

1  i 1 -1 2 0
2  i 2 1 2 0
3  i 3 -1 -2 0
4  i 4 1 2 3 0
5  q -1 0
6  s UNKNOWN
7  e -1 2 0
8  q 1 0
9  l 5 -1 0 1 3 0
10 l 6 2 0 5 2 0
11 s UNSATISFIABLE
12 u 1 0 5 0
13 q 2 3 0
14 s SATISFIABLE
15 m -1 2 3 0

```

Figure 8 This example illustrates the new incremental propagation interface of CADICAL 3.0, which identifies logical consequences of assumptions without launching a complete CDCL search. Lines 7-10 configure CADICAL to trace a non-binary LIDRUP proof to a file (the output of which is shown in the lower figure). In lines 12-18 a formula over three variables (TIE, SHIRT, and HAT) is constructed in order to encode a dress-code related logic puzzle. Note that variables must be explicitly declared prior to their use (see line 13), due to the new BVA support in CADICAL 3.0. The constructed formula is *propagated* under three different sets of assumptions:

1. Propagate assuming **-TIE** (lines 20-21): This query leads neither to a contradiction nor to a full assignment. However, as line 7 of the produced proof indicates, unit propagation correctly enforces **SHIRT** to be true in that query.
2. Propagate assuming **TIE** (lines 28-29): The solver identifies a contradiction during unit propagation, thus it accordingly returns **UNSATISFIABLE**. The proof confirms **TIE** as a failed assumption of this query (line 12).
3. Propagate assuming **SHIRT** and **HAT**: Unit propagation derives that **TIE** must be false. Consequently, the solver manages to deduce that the formula is satisfiable even without performing a full search. The resulting model is shown at line 15 of the produced proof.



© F. Pollitt and M. Fleury and K. Fazekas and N. Froleyks and A. Schidler and D. Schreiber and A. Biere;

licensed under Creative Commons License CC-BY 4.0

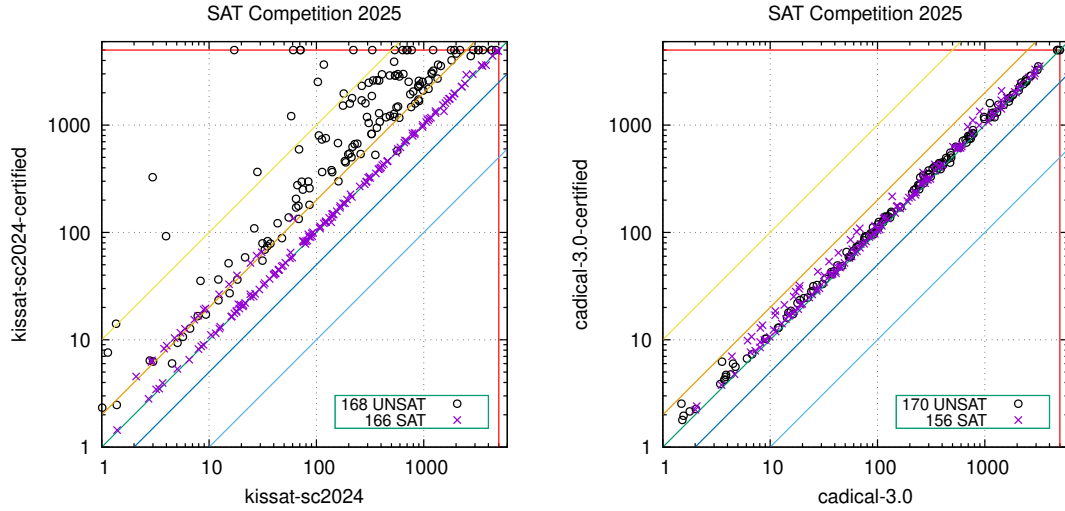
42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:17–23:18

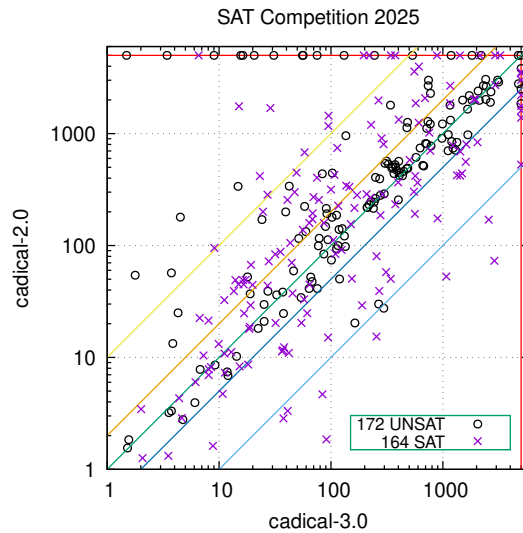


Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 9** The plots show (uncertified) solving time (x -axis) versus total certification time (y -axis) which includes both solving with proof production plus model checking with DIMOCHECK [10] for satisfiable instances and proof checking for unsatisfiable instances with DPR-TRIM [36] for KISSAT respectively LRAT-TRIM [20,50] for CADICAL. The times for the visually separated 19 satisfiable instances above the diagonal where model checking took a substantial amount of time are all solved by our lucky preprocessors and thus total solving and checking time for certification boils down to in essence parsing the instance twice (also note that KISSAT has the fastest parser among the solvers and checkers). The certification overhead for CADICAL (on the right) is almost not existent but for KISSAT (on the left) reaches a factor 10 and beyond.



■ **Figure 10** Scatter plot of solving times of CaDiCaL 3.0 (x -axis) versus CaDiCaL 2.0 (y -axis) showing that many benchmarks are solved much faster (are above the diagonal) for CaDiCaL 3.0.