









CaDiCaL 2.0

Armin Biere^{1,2}^(✉), Tobias Faller¹, Katalin Fazekas³, Mathias Fleury^{1,2},
Nils Froleyks², and Florian Pollitt¹



¹ University Freiburg, Freiburg, Germany
² Johannes Kepler University Linz, Linz, Austria
biere@cs.uni-freiburg.de
³ TU Wien, Vienna, Austria



Abstract. The SAT solver CaDiCaL provides a rich feature set with a clean library interface. It has been adopted by many users, is well documented and easy to extend due to its effective testing and debugging infrastructure. In this tool paper we give a high-level introduction into the solver architecture and then go briefly over implemented techniques. We describe basic features and novel advanced usage scenarios. Experiments confirm that CaDiCaL despite this flexibility has state-of-the-art performance both in a stand-alone as well as incremental setting.

1 Introduction

Progress in SAT solving has a large impact on model checking, SMT, theorem proving, software- and hardware-verification, and automated reasoning in general, and, according to “The SAT Museum” [20], SAT solvers get faster and faster, at least on benchmarks consisting of a single formula. For incremental SAT solving it was less clear, particularly as preprocessing [24] and inprocessing [69] heavily contributing to this improvement were considered incompatible with incremental solving (the winners of the SAT competition main track rely on inprocessing since 2009 except in 2011/2012/2016 and since 2005 all on preprocessing).

A simple and elegant solution to this problem is due to the award winning incremental SAT solving approach [39] first implemented in CADICAL. It reverts clause removal, i.e., restores clauses removed during pre- and inprocessing, restrictively on a case-by-case basis. It allows incremental solving to make full use of pre- and inprocessing techniques, in contrast to less general solutions [87, 89, 90, 112], without reducing their effectiveness nor burden the user to “freeze” and “melt” variables (“*Don’t Touch*” variables in [74]) as necessary with MINISAT [37].

This is the first tool paper on CADICAL, while previous, actually well cited, descriptions appeared only as system description in non-peer-reviewed SAT competition proceedings [14–16, 18, 21, 22]. In general, even though “SAT is considered a killer app for the 21st century” (Donald Knuth), there are few tool papers on SAT solvers, with the prominent exception of MINISAT [37], which appeared in 2003 and was awarded the test-of-time award at SAT’22. The descriptions of

CRYPTOMINISAT [106], GLUCOSE [5] and INTELSAT [86] introduce the corresponding SAT solver and can be considered to be tool papers too though.

Development of CADICAL was triggered by discussions at the “Theoretical Foundations of SAT Solving Workshop” in 2016 at the Fields Institute in Toronto, where it became apparent that both theoreticians and practitioners in SAT have a hard time understanding how practical SAT solving evolved, what key components there are in modern SAT solvers and, most importantly, that it was apparently getting harder and harder to modify state-of-the-art solvers for controlled experiments or to try out new ideas. With CADICAL we tried to change this, thus the main objective was to produce a clean solver, with well-documented source code, which is easy to read, understand, modify, test, and debug, without sacrificing performance too much.

The first goals were achieved from the beginning and performance improved over the years. After its introduction in 2017 CADICAL continued to achieve high rankings in yearly SAT competitions, e.g., in 2019 it solved the largest number of instances in the main track, but scored less than the winner. It never won though except for the most recent SAT competition in 2023 where CADICAL was combined with a strong preprocessor employing bounded variable addition [55, 82]. The competition organizers paraphrased this as “CADICAL strikes back”.

Moreover, with the show-case of our new incremental approach [39] we invested in increasing the feature set supported by CADICAL culminating for now in supporting “user propagators”. This for instance allowed to replace the original but highly modified MINISAT based SAT engine in *cvc5* by CADICAL, as described in a recent well-received SAT’23 paper [40].

The users of CADICAL fall into three categories. A first group applies the solver out of the box on benchmarks where CADICAL turns out to have superior performance. As an example consider solving mathematical problems with the help of SAT solving such as [78, 91, 108, 114]. Second, there is an increasing user base, including [6, 11, 23, 39, 40, 65, 92, 95, 101], which relies on the rich application programmable interface (API) provided by CADICAL, particularly its incremental features. Third, there are research prototypes modifying or extending CADICAL to achieve new features, including [7, 17, 43, 55, 64, 71]. Some of these modifications have been integrated [44, 100] but others remain future work [55].

Finally, CADICAL is used as a blue-print for understanding, porting, and integrating state-of-the-art techniques into other solvers. In this regard we are in contact with companies in cloud services, hardware design, and electronic design automation. It was also consulted in developing ISASAT [45], the only competitive fully verified SAT solver. Furthermore CADICAL was adopted as template solver for the “hack track” of the yearly SAT competition since 2021 as an “easy to hack” state-of-the-art SAT solver.

Related SAT solvers in the SAT competition often lack documentation, are hard to extend and modify, and, most importantly, do not provide such a rich and clean library interface as CADICAL. For instance our SAT solver KISSAT [18] falls into this category. It has been dominating the SAT competition 2020–2022 (in 2022 all top-ten solvers were descendants of KISSAT), is more compact in

memory usage and often faster on individual instances, but is lacking support for even the most basic incremental features such as assumptions.

The majority of the solvers in the SAT competition are restricted in their feature set as they are tuned for stand-alone usage, i.e., running the solver on a single formula stored in a file in DIMACS format [76], even though there is occasionally an incremental track in the SAT competition (last one that really took place was in 2020 as the one announced in 2021 was later cancelled).

Prominent SAT solvers with a richer feature set and particularly supporting incremental solving, beside the rather out-dated MINISAT [37], are newer versions of CRYPTOMINISAT [106], and GLUCOSE [4]. The former is actively developed and in terms of implemented techniques has quite some overlap with CADICAL. In addition it offers special support for XOR reasoning, solution sampling and model counting [105]. The GLUCOSE solver has been improved for incremental solving [3] but is not comparable in terms of implemented techniques nor features.

Unique and non-common features of CADICAL include: literal flipping [23], single clause assumption [46], incremental solving without freezing [39], extensive logging support, record & play of API calls, model-based testing, internal proof and solution (model) checking, termination and clause learner interfaces, various preprocessing techniques, an online proof tracing interface, formula extraction (after simplification), support of many external proof formats (DRAT, LRAT, FRAT, VeriPB) [100], and last but not least the user propagator [40].

This paper is structured by describing in the next section the architecture of CADICAL, which also acts as a summary of integrated techniques and provided features. The rest of the paper consists of highlighting recently added features of the solver or features not presented before, followed by experiments showing that CADICAL has state-of-the-art performance, before concluding.

2 Architecture

CADICAL is a modern SAT solver with many features written in C++. It can be used as stand-alone application through the *command-line interface* (CLI) or as library through its *application programming interface* (API) in C++ (or in limited form in C). Figure 1 depicts a structural overview. The central component, called **Internal**, implements CDCL search [83, 103] and formula simplification techniques [24, 69]. On top of it, the **External** facade hides the internals while maintaining the proofs and solutions (aka *models*) of solved problems.

The heart of the solver is the function `cdcl_loop_with_inprocessing` in **Internal** which interleaves the CDCL loop with formula simplification steps (i.e., with inprocessing [69]). During **Search**, CADICAL supports several techniques, like chronological backtracking [84, 88], rephasing [32], and shrinking [44], which are only some of the important features. See Fig. 1 for more references.

The CDCL loop [83] is scheduled to be preempted in regular intervals to let the solver apply various formula simplification [24] and inprocessing techniques [69]. Each technique is implemented separately (e.g., in file `subsume.cpp`)

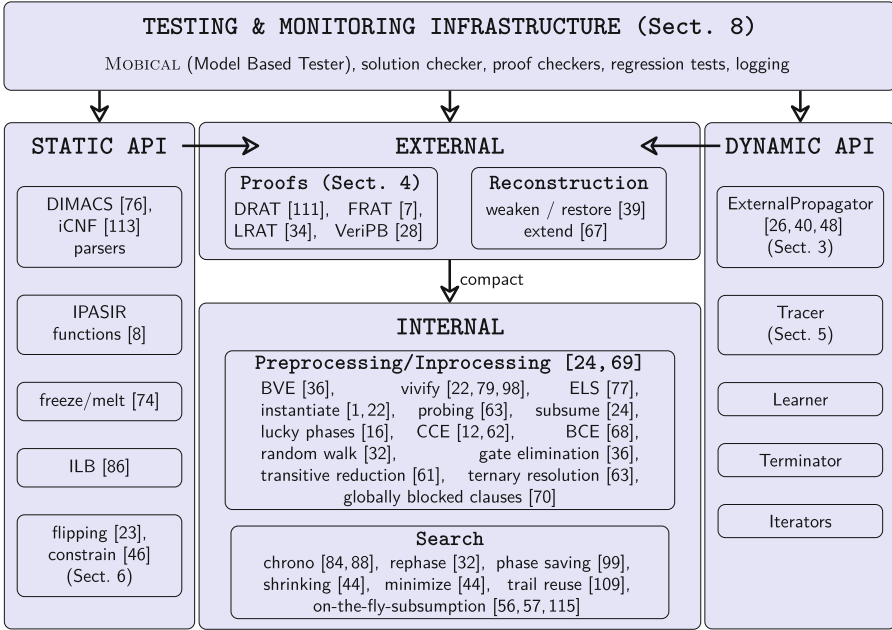


Fig. 1. An overview of the main components of CADICAL.

and has (i) a corresponding function which determines if the solver should preempt CDCL search and apply the technique (e.g., `subsuming()`) and (ii) a function that actually applies the technique (e.g., `subsume()`).

As Fig. 1 shows, CADICAL implements a variety of preprocessing/inprocessing techniques, including bounded variable elimination (BVE) [36], arguably the most effective one. As further examples, CADICAL also supports vivification [79, 98] and instantiation [1]. Combining them [22] won the CADICAL “hack track” 2023.

The **External** component communicates with **Internal** by mapping active variables into a consecutive sequence of integers (*compacting*) and extends internal solutions back to complete solution of the input problem with the help of the reconstruction stack [67]. In incremental use cases **External** also keeps the reconstruction stack *clean* [39] by “undoing” previous inprocessing steps. Beyond that, **External** connects internal and external proof generation (see Sect. 4).

We distinguish two types of API usage in CADICAL: *static* and *dynamic*. The static API provides access to standard solver functionalities *between* SAT solving calls (like IPASIR [8], parsing DIMACS, or iCNF files). With ILB as proposed by INTELSAT [86], we try to keep the trail unchanged between incremental calls.

The dynamic API interacts and controls the solver *during* Search. The solver provides dynamic access to clauses learned during conflict analysis to connected **Learner** instances. The **Terminator** class interface allows users to asynchronously terminate the solving procedure. Through the **Iterator** inter-

face of CADICAL, the user can iterate over the irredundant (simplified) clauses of the problem or can iterate through clauses on the reconstruction stack, supporting simplified formula extraction and external model reconstruction.

3 External Propagator

Applications of CADICAL, for example within the SMT solver `cvc5` [11] (and maybe in the future within other lazy SMT solvers, such as `Z3` [85] or `Yices` [35]), or to support Satisfiability Modulo Symmetries (SMS) [40, 116], require more control over the solver than provided by the standard incremental IPASIR interface [8]. To this purpose CADICAL supports a more fine-grained and tighter integration into larger systems by allowing an external user propagator [26, 48, 49] to be connected to it through the IPASIR-UP interface [40].

This abstract interface is defined in the `ExternalPropagator` class which provides corresponding notification and callback functions. Inheriting from this class allows users to implement dedicated external propagators which for instance import and export learned clauses or suggest decisions to the SAT solver. The full description of functionalities supported by the IPASIR-UP interface is available in [40]. Here we focus on CADICAL-specific implementation details.

First, CADICAL ensures that only external variables appear in the IPASIR-UP interactions, thereby allowing users to ignore the internal (compacted) details. Furthermore, CADICAL employs preprocessing and inprocessing even when an external propagator is connected. To avoid the need to restore clauses during the CDCL loop and to ensure that solution reconstruction [67] does not change assignments of *observed variables* (i.e., relevant to the external propagator), every observed variable is automatically frozen. As a side effect, the external propagator can only set *clean* [39] variables as new observed variables during search. As fresh variables are always clean, this is acceptable and mostly sufficient in practice.

Finally, CADICAL, by default, considers every external clause as irredundant, exactly as the original input clauses of the problem. Thus, during clause database reduction they are not candidates for removal and so can be deleted only when implied by the rest of the formula. In future work we plan to allow users to specify the redundancy of the external clauses and to support incremental inprocessing [39] even for variables observed by the external propagator.

4 Proofs

Unsatisfiability proof certificates are an integral part of SAT solving [59, 60]. Even though clausal proofs were introduced in 2003 [37, 52], checking large proofs only became viable with deletion information [58]. The most prominent format today is DRAT [111] which was mandatory in the SAT competition from 2016 [10] to 2022. In 2023 both DRAT [111] and VeriPB [28] were allowed in the competition [9].

```
void add_derived_clause (uint64_t new_id, bool redundant,
                        const vector<int> & literals_of_clause,
                        const vector<uint64_t> & antecedent_ids);
```

Fig. 2. Tracer virtual callback function to add a derived clause to the proof.

The proof formats GRAT [75] and LRAT [34] were proposed to allow even faster proof checking, i.e., by trading time for space, but also to facilitate formally verified proof checkers (e.g., CAKE_LPR [110]). They require hints for clause additions in form of antecedent *clause identifiers* (ids). External tools like DRAT-TRIM [111] can add such hints in a post-processing step to DRAT proofs.

The proof formats DRAT [111], FRAT [7], LRAT [34], and VeriPB [28] are supported by CADICAL. It is the first solver to support LRAT natively. Without the need for post-processing this reduces proof checking time [100] substantially.

Recent diversification of proof formats in the SAT competition [9] motivated us to add VeriPB. It is a general proof format for various applications [28,50,51]. The tool-chain for checking SAT solver proofs with the verified VeriPB backend [28] is under development and not fast enough yet. Actually, BREAKID-KISSAT [9], one of the top performers in the SAT competition 2023, lost due to multiple timeouts during proof checking. Similarly to FRAT, CADICAL can provide antecedents in VeriPB proofs. We expect this to speed up VeriPB proof checking considerably.

5 Tracer Interface

The dynamic API allows to extract proof information from CADICAL online without files by connecting user-defined tracers as instances of the virtual C++ class `Tracer`. It provides notifications and callbacks for proof-related events, such as addition and deletion of clauses. Proof writers for all formats (Sect. 4) as well as both internal proof checkers (Sect. 8) go through the `Tracer` class. Furthermore, there is ongoing work to produce VeriPB proofs for the MaxSAT solver Pacose [97] using the `Tracer` interface in CADICAL.

We support a large set of event types covering a multitude of use cases. Information provided includes antecedent ids and literals of clauses, separation between original, derived, and restored [39] clauses, and information of clause redundancy, as well as weakening [69] and strengthening [28,69]. For example, Fig. 2 shows the callback function for the proof event of adding a derived clause, where “`derived`” means entailed by the formula (i.e., not original input clause). Additional notifications include reserving ids for original clauses, as used for generating file based proof formats, such as VeriPB and LRAT.

For each solve call, a concluding event gives precise information about the result: a model concludes satisfiable instances, whereas for unsatisfiable instances we provide information about the final conflict clause. We have recently started to explore incremental proof tracing as well [41,42].

6 Constraints and Flipping

SAT solvers are used in a wide range of applications in many different ways. For incremental solving, MINISAT has been the predominant choice. However, in recent years, CADICAL has begun to replace MINISAT in numerous applications, most prominently cvc5. This can be attributed to its overall better performance and various application-specific features unique to CADICAL.

A prime example is the *constraint* feature [46], which allows users to define a temporary clause with the same lifespan as assumptions. It was initially developed to support the SAT based model checking algorithm IC3 [29], which requires often millions of incremental SAT calls during a single run, where each query needs to assume a single clause valid only for that call.

Constraints do not introduce new functionality per se, as temporary clauses can be simulated by activation literals. But they do allow the solver to employ a more efficient implementation, as they particularly avoid to introduce those assumption variables. Beyond IC3, constraints have also proven useful in our backbone extractor CADIBACK [23]. The purpose of using *constraint* in backbone extraction is to find maximally diverging models in order to eliminate backbone candidates fast. CADIBACK uses constraints to ensure that each new model includes at least one literal not observed in previous models. If this is not possible, all unseen literals are immediately determined to be in the backbone.

Once a model is found, we use another feature called *literal flipping* [19] to eliminate further backbone candidates [23]. A literal is *flippable* if toggling its value also results in a model. This concept was employed to speed-up backbone MINIBONES [66] before and also MUS extraction [13]. In these earlier works it was implemented by iterating over all clauses outside the SAT solver, searching for literals that can be flipped in the model provided by the solver. Using clause watching our implementation inside of CADICAL is much more efficient.

7 Interpolation

Software-based test generation targeting RISC-V in the Scale4Edge project [38] relied on interpolation-based model checking and MINICRAIG to generate interpolants. It uses MINISAT as SAT solver and in this application constitutes a performance bottleneck. Therefore we developed a new more scalable solver CADICRAIG based on CADICAL and its proof tracer API (Sect. 5).

The implementation of CADICRAIG is external to CADICAL. It uses the same interpolant construction as in MINICRAIG but is now separated from MINISAT. We are not aware of any other modern open-source SAT solver which allows to build interpolants through a generic API without being forced to write the whole proof to a file, trimming and post-processing it on disk, such as in [53].

The CADICRAIG tracer constructs partial interpolants as usual, e.g., see [73]. Through the proof tracer API the tracer is notified by CADICAL about each new clause and its antecedents needed to derive it by resolution. It then builds a partial interpolant for that clause using previously computed partial antecedent

interpolants. When the solver concludes deriving an empty clause and thus showing unsatisfiability (Sect. 5) the final interpolant is built from the antecedents of the empty clause. It can then be retrieved by via the CADICRAIG API.

8 Testing and Debugging

Such a sophisticated and complex software as CADICAL necessitates rigorous testing to ensure correctness of interactions between its multitude of features. In this section we discuss our arsenal of essential testing and debugging techniques.

First, we primarily rely on logging for debugging purposes. For instance, when enabled, CADICAL will print every single step from its creation to its deletion. From an implementation perspective, logging features are not compiled in by default to avoid performance overhead in release builds. Furthermore, if enabled at run-time, CADICAL prints verbose information about the inprocessing schedule, useful for debugging performance regressions (e.g., inprocessor scheduling).

Further useful debugging tools are the built-in checkers. The LRAT and DRAT checkers are optional and ensure that every learned clause is properly derived. The new LRAT checker [100] was crucial for achieving LRAT support.

Last but not least we want to mention the API fuzzer MOBICAL, which generates random API calls and minimizes failing runs. Internally, MOBICAL implements a state machine issuing API calls. It also performs option fuzzing by varying available options. This approach is extremely useful to produce short failing API call traces focusing on the actual defect, e.g., like picking a low garbage collection limit to trigger a defect in the garbage collector. Combining checkers with MOBICAL greatly increases its strength. During development it is advisable to build MOBICAL and CADICAL with assertions and checkers enabled.

MOBICAL is similar in spirit to the related model-based tester of LINGELING [2] for SAT and BTORMBT [94] and MURXLA [93] targeting SMT. Note that other SMT fuzzers [27,30,96,102] focus on non-incremental usage or only support incremental “push & pop” [80]. For non-incremental SAT solving, there is also CNFFUZZ fuzzer and the CNFDD delta-debugger [2,31].

Accordingly, we have implemented a MOCKPROPAGATOR class in MOBICAL to test the EXTERNALPROPAGATOR API. It fuzzes the IPASIR-UP implementation in combination with all options and features of the solver. It revealed several corner-cases which we believe would have been very hard to trigger otherwise.

MOBICAL targets only incremental SAT problems and could not help when incorrect interpolants showed up in earlier experiments with MINICRAIG and CADICRAIG. Therefore, we have built an external interpolation fuzzer in Python. It checks interpolants and an accompanying delta-debugger minimizes problems by deleting command line options, clauses, and variables.

9 Experiments

The performance of CADICAL 2.0 was evaluated in three experiments. We first follow the *non-incremental* setup of the main track of the SAT competition,

where solvers are run on benchmark files in DIMACS format. The second experiment focuses on *incremental* usage, i.e., following the incremental track of the competition. Finally we show the effectiveness of CADICAL in the context of *interpolation* via its *Tracer* API. All experiments were conducted on our cluster with Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz (turbo-mode disabled).

Non-Incremental. The winner SBVA-CADICAL [54] of the main track of the SAT Competition 2023 combined a novel technique for bounded variable addition [82] with CADICAL 1.5.3. In their implementation preprocessing was limited to 200 s which yields different preprocessed formulas over multiple runs. Therefore, we ran the preprocessor of SBVA-CADICAL separately for 200 s, and then gave the same formulas to CADICAL 1.5.3 and our new version CADICAL 2.0. Running them for 5000 s as in the competition (ignoring preprocessing time in essence) gave very similar results. We provide more details in the artifact. This confirms that CADICAL (also in version 2.0) is state-of-the-art in non-incremental solving.

Incremental. How to assess the incremental performance of a SAT solver is less established. To present an unbiased evaluation, we follow the principles set out by the last incremental track of the SAT competition in 2020 [47]: The solvers are evaluated in six different applications, each featuring 50 benchmarks, with a 2000 s timeout and 24 GB memory limit. Four applications are carried over directly from the 2020 competition: the CEGAR-based QBF solver IJTIHAD, the simple backbone extractor BONES, the longest simple-path search LSP, and the MaxSAT solver MAX. However, we exclude two applications: the essential variable extractor and the classical planner PASAR. Both use features that are not present in all solvers. The former queries `ipasir_learned`, which is missing from CADICAL 1.0, and the latter relies on limiting the number of conflicts. Instead, we include the bounded model checker for bit-level hardware designs CAMICAL [39] and the sophisticated backbone extractor CADIBACK [23].

The benchmarks from the incremental track from the 2020 SAT Competition remain unchanged. For CAMICAL, we randomly select 50 Boolean circuits used in HWMCC’17 [25]. Although CADIBACK solves the same problem as BONES, we opt for a distinct set of benchmarks. In 2020 the “smallest and easiest satisfiable” [47] CNF formulas were selected and even though backbone extraction is harder than mere solving, they were rather easy. Conversely, we compile a non-trivial set of benchmarks by randomly selecting satisfiable formulas from past competitions (2004–2022) [23] that take KISSAT 3.0.0 [19] more than 20 s to solve. We use KISSAT as it is not incremental and hence does not compete.

The artifact has a comparison of CRYPTOMINISAT and CADICAL on 1798 formulas [23] and indicates that our selection does not impact the outcome. As detailed in Sect. 6, CADIBACK utilizes constraints, which are only available in recent versions of CADICAL and are simulated with activation literals otherwise.

Our evaluation includes all solvers that competed in 2020: RISS 7.1.2 [81], CRYPTOMINISAT 5 (CMS) [104, 107], and ABCDSAT i20 [33]. The CADICAL version from that year is referred to as CADICAL 2020. The other two versions are 1.0 from 2019 and our latest release 2.0. We also include MINISAT 2.2 and

the latest version of GLUCOSE 4.2.1. Table 1 presents for each SAT solver and application: the PAR2 score, which is the average runtime in seconds with a penalty of 4000 for unsolved instances; and the number of solved instances.

Table 1. Performance comparison of six incremental solvers, with three versions of CADICAL (2000s timeout). For each solver, we report PAR2 score over 50 benchmarks per application and number of solved instances (“PAR2_{/#solved}”). The four applications to the right have been used in the incremental track of the 2020 SAT competition. The best results per application are marked in **bold**. The last row presents the hypothetical *Virtual Best Solver* which always picks the best performing backend for each instance.

		CaDiBack	CaMiCaL	Bones	LSP	Max	Ijtihad	Total
CADICAL	2.0	3297 ₁₁	2606 ₁₈	494 ₄₅	1898 ₂₇	1976 ₂₆	2980 ₁₃	2209 ₁₄₀
	2020	3409 ₉	2677 ₁₇	622 ₄₃	1955 ₂₆	2015 ₂₅	2986 ₁₃	2277 ₁₃₃
	1.0	3495 ₇	2627 ₁₈	595 ₄₄	2011 ₂₆	2028 ₂₅	2989 ₁₃	2291 ₁₃₃
	CMS	3491 ₈	2701 ₁₇	397 ₄₆	1773 ₂₉	2021 ₂₅	3057 ₁₂	2240 ₁₃₇
	MINISAT	3678 ₅	2807 ₁₆	687 ₄₃	1993 ₂₆	2094 ₂₄	3123 ₁₁	2397 ₁₂₅
	RISS	3665 ₆	2836 ₁₅	892 ₄₀	1835 ₂₈	2017 ₂₅	3140 ₁₁	2398 ₁₂₅
	ABCDSAT	3582 ₇	2966 ₁₃	535 ₄₆	2493 ₂₁	2037 ₂₆	3207 ₁₀	2470 ₁₂₃
	GLUCOSE	3778 ₄	2981 ₁₃	948 ₄₀	2078 ₂₅	2117 ₂₄	3206 ₁₀	2518 ₁₁₆
	VBS	3127 ₁₄	2546 ₁₉	257 ₄₈	1765 ₂₉	1856 ₂₈	2896 ₁₄	2075 ₁₅₂

Our results show that CADICAL 2.0 reaches state-of-the-art performance, demonstrating a distinct improvement over previous versions. Also, differing from the findings in [72], we see a significant advantage of the newer CADICAL and CRYPTOMINISAT, over the older MINISAT, further substantiated below.

Interpolants. To validate CADICRAIG using CADICAL, we converted all 400 benchmarks of the SAT Competition 2023 into interpolation problems split into A and B parts chosen with the goal to assign related clauses to the same part in order to keep the number of global variables limited. The index of the smallest variable of each clause determines the probability of the clause being assigned to A. On our crafted benchmarks (5000s timeout, 7 GB), CADICRAIG significantly outperforms MINICRAIG, solving 117 benchmarks, compared to only 75.

10 Conclusion

In this very first conference paper on CADICAL we reviewed its most important components and features as well as its testing and debugging infrastructure. We highlighted its use as SAT engine in SMT solving via the user propagator interface and how the tracer API can be used to compute interpolants. Our experiments show that CADICAL remains efficient despite this flexibility.

Producing incremental proofs is ongoing work [41, 42]. Further future work consists of producing incremental proofs for all features supported by CADICAL,

avoiding to freeze observed variables by the user propagator, and porting into the main branch features provided by other users.

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. Andersson, G., Bjesse, P., Cook, B., Hanna, Z.: A proof engine approach to solving combinational design automation problems. In: Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, 10–14 June 2002, pp. 725–730. ACM (2002). <https://doi.org/10.1145/513918.514101>
2. Artho, C., Biere, A., Seidl, M.: Model-based testing for verification back-ends. In: Veanes, M., Viganò, L. (eds.) TAP 2013. LNCS, vol. 7942, pp. 39–55. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38916-0_3
3. Audemard, G., Lagniez, J.-M., Simon, L.: Improving glucose for incremental SAT solving with assumptions: application to MUS extraction. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 309–317. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39071-5_23
4. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, 11–17 July 2009, pp. 399–404. Morgan Kaufmann Publishers Inc., San Francisco (2009). <http://ijcai.org/Proceedings/09/Papers/074.pdf>
5. Audemard, G., Simon, L.: On the glucose SAT solver. *Int. J. Artif. Intell. Tools* **27**(1), 1840001:1–1840001:25 (2018). <https://doi.org/10.1142/S0218213018400018>
6. Bacchus, F.: MaxHS in the 2022 MaxSat evaluation. In: Bacchus, F., Berg, J., Järvisalo, M., Martins, R. (eds.) Proceedings of MaxSAT Evaluation 2020 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2022-2, p. 17. University of Helsinki (2022)
7. Baek, S., Carneiro, M., Heule, M.J.H.: A flexible proof format for SAT solver-elaborator communication. In: TACAS 2021. LNCS, vol. 12651, pp. 59–75. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72016-2_4
8. Balyo, T., Biere, A., Iser, M., Sinz, C.: SAT race 2015. *Artif. Intell.* **241**, 45–65 (2016)
9. Balyo, T., Heule, M.J.H., Iser, M., Järvisalo, M., Suda, M. (eds.): 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)
10. Balyo, T., Heule, M.J.H. (eds.): Proceedings of SAT Competition 2016 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2016-1. University of Helsinki (2016)
11. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24

12. Barnett, L.A., Cerna, D., Biere, A.: Covered clauses are not propagation redundant. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12166, pp. 32–47. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51074-9_3
13. Belov, A., Marques-Silva, J.: Accelerating MUS extraction with recursive model rotation. In: Bjesse, P., Slobodová, A. (eds.) International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, 30 October–02 November 2011, pp. 37–40. FMCAD Inc. (2011)
14. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling, YaSAT entering the SAT competition 2017. In: Balyo, T., Heule, M.J.H., Jarvisalo, M. (eds.) Proceedings of SAT Competition 2017 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2017-1, pp. 14–15. University of Helsinki (2017)
15. Biere, A.: CaDiCaL, Lingeling, Plingeling, Treengeling and YaSAT entering the SAT competition 2018. In: Heule, M.J.H., Jarvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2018 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2018-1, pp. 13–14. University of Helsinki (2018)
16. Biere, A.: CaDiCaL at the SAT race 2019. In: Heule, M.J.H., Jarvisalo, M., Suda, M. (eds.) Proceedings of SAT Race 2019 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2019-1, pp. 8–9. University of Helsinki (2019)
17. Biere, A., Chowdhury, M.S., Heule, M.J.H., Kiesl, B., Whalen, M.W.: Migrating solver state. In: SAT. LIPIcs, vol. 236, pp. 27:1–27:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). <https://doi.org/10.4230/LIPICS.SAT.2022.27>
18. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. In: Balyo, T., Froylyks, N., Heule, M.J.H., Iser, M., Jarvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
19. Biere, A., Fleury, M.: Gimsatul, IsaSAT and Kissat entering the SAT competition 2022. In: Balyo, T., Heule, M.J.H., Iser, M., Jarvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2022-1, pp. 10–11. University of Helsinki (2022)
20. Biere, A., Fleury, M., Froylyks, N., Heule, M.J.: The SAT museum. In: Jarvisalo, M., Le Berre, D. (eds.) Proceedings of the 14th International Workshop on Pragmatics of SAT Co-located with the 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), Alghero, Italy, 4 July 2023. CEUR Workshop Proceedings, vol. 3545, pp. 72–87. CEUR-WS.org (2023). <http://ceur-ws.org/Vol-3545/paper6.pdf>
21. Biere, A., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba entering the SAT competition 2021. In: Balyo, T., Froylyks, N., Heule, M.J.H., Iser, M., Jarvisalo, M., Suda, M. (eds.) Proceedings of SAT Competition 2021 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2021-1, pp. 10–13. University of Helsinki (2021)
22. Biere, A., Fleury, M., Pollitt, F.: CaDiCaL_vivinst, IsaSAT, Gimsatul, Kissat, and Tabulara SAT entering the SAT competition 2023. In: Balyo, T., Froylyks, N., Heule, M.J.H., Iser, M., Jarvisalo, M., Suda, M. (eds.) Proceedings of SAT

- Competition 2023 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2023-1, pp. 14–15. University of Helsinki (2023)
23. Biere, A., Froyleys, N., Wang, W.: CadiBack: extracting backbones with CaDiCaL. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, Alghero, Italy, 4–8 July 2023. LIPIcs, vol. 271, pp. 3:1–3:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). <https://doi.org/10.4230/LIPICS.SAT.2023.3>
 24. Biere, A., Järvisalo, M., Kiesl, B.: Preprocessing in SAT solving. In: Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, 2nd edn., vol. 336, pp. 391–435. IOS Press (2021)
 25. Biere, A., van Dijk, T., Heljanko, K.: Hardware model checking competition 2017. In: 2017 Formal Methods in Computer Aided Design (FMCAD), pp. 9–9. IEEE (2017)
 26. Bjørner, N.S., Eisenhofer, C., Kovács, L.: Satisfiability modulo custom theories in Z3. In: Dragoi, C., Emmi, M., Wang, J. (eds.) VMCAI. LNCS, vol. 13881, pp. 91–105. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-24950-1_5
 27. Blotsky, D., Mora, F., Berzish, M., Zheng, Y., Kabir, I., Ganesh, V.: StringFuzz: a fuzzer for string solvers. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10982, pp. 45–51. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96142-2_6
 28. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified symmetry and dominance breaking for combinatorial optimisation. *J. Artif. Intell. Res.* **77**, 1539–1589 (2023)
 29. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18275-4_7
 30. Bringolf, M., Winterer, D., Su, Z.: Finding and understanding incompleteness bugs in SMT solvers. In: ASE, pp. 43:1–43:10. ACM (2022)
 31. Brummayer, R., Lonsing, F., Biere, A.: Automated testing and debugging of SAT and QBF solvers. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 44–57. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14186-7_6
 32. Cai, S., Zhang, X., Fleury, M., Biere, A.: Better decision heuristics in CDCL through local search and target phases. *J. Artif. Intell. Res.* **74**, 1515–1563 (2022). <https://doi.org/10.1613/jair.1.13666>
 33. Chen, J.: optsat, abcdsat and solvers based on simplified data structure and hybrid solving strategies. In: Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions, p. 25 (2020)
 34. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: de Moura, L. (ed.) CADE 2017. LNCS (LNAI), vol. 10395, pp. 220–236. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_14
 35. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_49
 36. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005). https://doi.org/10.1007/11499107_5

37. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
38. Faller, T., Deligiannis, N.I., Schwörer, M., Reorda, M.S., Becker, B.: Constraint-based automatic SBST generation for RISC-V processor families. In: IEEE European Test Symposium, ETS 2023, Venezia, Italy, 22–26 May 2023, pp. 1–6. IEEE (2023). <https://doi.org/10.1109/ETS56758.2023.10174156>
39. Fazekas, K., Biere, A., Scholl, C.: Incremental inprocessing in SAT solving. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 136–154. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_9
40. Fazekas, K., Niemetz, A., Preiner, M., Kirchweger, M., Szeider, S., Biere, A.: IPASIR-UP: user propagators for CDCL. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, Alghero, Italy. LIPICs, vol. 271, pp. 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). <https://doi.org/10.4230/LIPICS.SAT.2023.8>
41. Fazekas, K., Pollitt, F., Fleury, M., Biere, A.: Certifying incremental sat solving. In: Bjorner, N., Heule, M., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 25th International Conference, LPAR-25, Balaclava, Mauritius, 26–31 May 2024. Proceedings (2024)
42. Fazekas, K., Pollitt, F., Fleury, M., Biere, A.: Incremental proofs for bounded model checking. In: Kunz, W., Große, D. (eds.) Workshop on Methods and Description Languages for Modelling and Verification of Circuits and Systems, MBMV 2024, Kaiserslautern, Germany, 14–15 February 2023. ITG Fachberichte, VDE Verlag (2024)
43. Feng, N., Bacchus, F.: Clause size reduction with all-UIP learning. In: Pulina, L., Seidl, M. (eds.) SAT 2020. LNCS, vol. 12178, pp. 28–45. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_3
44. Fleury, M., Biere, A.: Efficient All-UIP learned clause minimization. In: Li, C.-M., Manyà, F. (eds.) SAT 2021. LNCS, vol. 12831, pp. 171–187. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80223-3_12
45. Fleury, M., Lammich, P.: A more pragmatic CDCL for isasat and targetting LLVM (short paper). In: Pientka, B., Tinelli, C. (eds.) Automated Deduction - CADE 29 - 29th International Conference on Automated Deduction, Rome, Italy, 1–4 July 2023, Proceedings. Lecture Notes in Computer Science, vol. 14132, pp. 207–219. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-38499-8_12
46. Froleys, N., Biere, A.: Single clause assumption without activation literals to speed-up IC3. In: Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, 19–22 October 2021, pp. 72–76. IEEE (2021). https://doi.org/10.34727/2021/ISBN.978-3-85448-046-4_15
47. Froleys, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT competition 2020. *Artif. Intell.* **301**, 103572 (2021). <https://doi.org/10.1016/J.ARTINT.2021.103572>
48. Ganesh, V., O'Donnell, C.W., Soos, M., Devadas, S., Rinard, M.C., Solar-Lezama, A.: Lynx: a programmatic SAT solver for the RNA-folding problem. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 143–156. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_12
49. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory solving made easy with clingo 5. In: ICLP (Technical Communications). OASICs, vol. 52, pp. 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)

50. Gocht, S.: Certifying Correctness for Combinatorial Algorithms by Using Pseudo-Boolean Reasoning. Ph.D. thesis, Lund University, Lund, Sweden (2022). <https://portal.research.lu.se/en/publications/certifying-correctness-for-combinatorial-algorithms-by-using-pseu>
51. Gocht, S., Nordström, J.: Certifying parity reasoning efficiently using pseudo-Boolean proofs. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21), pp. 3768–3777 (2021)
52. Goldberg, E., Novikov, Y.: Verification of proofs of unsatisfiability for cnf formulas. In: 2003 Design, Automation and Test in Europe Conference and Exhibition, pp. 886–891 (2003). <https://api.semanticscholar.org/CorpusID:10504432>
53. Gurfinkel, A., Vize, Y.: DRUPing for interpolates. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, 21–24 October 2014, pp. 99–106. IEEE (2014). <https://doi.org/10.1109/FMCAD.2014.6987601>
54. Haberlandt, A., Green, H.: SBVA-CADICAL and SBVA-KISSAT: structured bounded variable addition. In: Balyo, T., Froleyks, N., Heule, M.J.H., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2023 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2023-1, p. 18. University of Helsinki (2023)
55. Haberlandt, A., Green, H., Heule, M.J.H.: Effective auxiliary variables via structured reencoding. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, Alghero, Italy. LIPIcs, 4–8 July 2023, vol. 271, pp. 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). <https://doi.org/10.4230/LIPICS.SAT.2023.11>
56. Hamadi, Y., Jabbour, S., Sais, L.: Learning for dynamic subsumption. In: ICTAI 2009, 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, New Jersey, USA, 2–4 November 2009, pp. 328–335. IEEE Computer Society (2009). <https://doi.org/10.1109/ICTAI.2009.22>
57. Han, H., Somenzi, F.: On-the-fly clause improvement. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 209–222. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_21
58. Heule, M., Jr., W.A.H., Wetzler, N.: Trimming while checking clausal proofs. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, 20–23 October 2013, pp. 181–188. IEEE (2013)
59. Heule, M.J.H.: Proofs of unsatisfiability. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 635–668. IOS Press (2021). <https://doi.org/10.3233/FAIA200998>
60. Heule, M.J.H., Biere, A.: Proofs for satisfiability problems. In: All about Proofs, Proofs for All (APPA), Mathematical, Logic and Foundations, vol. 55. College Publication (2015)
61. Heule, M., Järvisalo, M., Biere, A.: Clause elimination procedures for CNF formulas. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 357–371. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16242-8_26
62. Heule, M.J.H., Järvisalo, M., Biere, A.: Covered clause elimination. In: Voronkov, A., Sutcliffe, G., Baaz, M., Fermüller, C.G. (eds.) Short papers for 17th International Conference on Logic for Programming, Artificial intelligence, and Reasoning, LPAR-17-short, Yogyakarta, Indonesia, 10–15 October 2010. EPiC Series in Computing, vol. 13, pp. 41–46. EasyChair (2010). <https://doi.org/10.29007/CL8S>

63. Heule, M.J.H., Järvisalo, M., Biere, A.: Revisiting hyper binary resolution. In: Gomes, C., Sellmann, M. (eds.) CPAIOR 2013. LNCS, vol. 7874, pp. 77–93. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38171-3_6
64. Hickey, R., Bacchus, F.: Trail saving on backtrack. In: Pulina, L., Seidl, M. (eds.) SAT 2020. LNCS, vol. 12178, pp. 46–61. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51825-7_4
65. Ignatiev, A., Morgado, A., Marques-Silva, J.: PySAT: a python toolkit for prototyping with SAT oracles. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 428–437. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_26
66. Janota, M., Lynce, I., Marques-Silva, J.: Algorithms for computing backbones of propositional formulae. *AI Commun.* **28**(2), 161–177 (2015). <https://doi.org/10.3233/AIC-140640>
67. Järvisalo, M., Biere, A.: Reconstructing solutions after blocked clause elimination. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 340–345. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14186-7_30
68. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_10
69. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 355–370. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_28
70. Kiesel, B., Heule, M.J.H., Biere, A.: Truth assignments as conditional autarkies. In: Chen, Y.-F., Cheng, C.-H., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 48–64. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_3
71. Kiesel-Reiter, B., Whalen, M.W.: Proofs for incremental SAT with inprocessing. In: Nadel, A., Rozier, K.Y. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, 24–27 October 2023, pp. 132–140. IEEE (2023). https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_21
72. Kochemazov, S., Ignatiev, A., Marques-Silva, J.: Assessing progress in SAT solvers through the lens of incremental SAT. In: Li, C.-M., Manyà, F. (eds.) SAT 2021. LNCS, vol. 12831, pp. 280–298. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-80223-3_20
73. Kupferschmid, S.: Über Craigsche Interpolation und deren Anwendung in der formalen Modellprüfung. Ph.D. thesis, University of Freiburg (2013)
74. Kupferschmid, S., Lewis, M., Schubert, T., Becker, B.: Incremental preprocessing methods for use in BMC. *Formal Methods Syst. Des.* **39**(2), 185–204 (2011). <https://doi.org/10.1007/S10703-011-0122-4>
75. Lammich, P.: Efficient verified (UN)SAT certificate checking. *J. Autom. Reason.* **64**(3), 513–532 (2020). <https://doi.org/10.1007/s10817-019-09525-z>
76. Le Berre, D., Roussel, O., Simon, L.: SAT competition 2009: Benchmark submission guidelines. <https://web.archive.org/web/20190325181937/https://www.satcompetition.org/2009/format-benchmarks2009.html>. Accessed 15 Jan 2024
77. Li, C.M.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, Austin, Texas, USA, 30 July–3 August 2000, pp. 291–296. AAAI Press/The MIT Press (2000), <http://www.aaai.org/Library/AAAI/2000/aaai00-045.php>

78. Lohn, E., Lambert, C., Heule, M.J.H.: Compact symmetry breaking for tournaments. In: Griggio, A., Rungta, N. (eds.) 22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, 17–21 October 2022, pp. 179–188. IEEE (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_24
79. Luo, M., Li, C., Xiao, F., Manyà, F., Lü, Z.: An effective learnt clause minimization approach for CDCL SAT solvers. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, 19–25 August 2017, pp. 703–711. *ijcai.org* (2017). <https://doi.org/10.24963/IJCAI.2017/98>
80. Mansur, M.N., Christakis, M., Wüstholtz, V., Zhang, F.: Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. In: ESEC/SIGSOFT FSE, pp. 701–712. ACM (2020)
81. Manthey, N.: Riss 7 in proceedings of SAT competition 2020. In: Proceedings of SAT Competition 2020: Solver and benchmark descriptions (2020)
82. Manthey, N., Heule, M.J.H., Biere, A.: Automated reencoding of Boolean formulas. In: Biere, A., Nahir, A., Vos, T. (eds.) HVC 2012. LNCS, vol. 7857, pp. 102–117. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39611-3_14
83. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 133–182. IOS Press (2021). <https://doi.org/10.3233/FAIA200987>
84. Möhle, S., Biere, A.: Backing backtracking. In: Janota, M., Lynce, I. (eds.) SAT 2019. LNCS, vol. 11628, pp. 250–266. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24258-9_18
85. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
86. Nadel, A.: Introducing Intel(R) SAT Solver. In: Meel, K.S., Strichman, O. (eds.) 25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022). Leibniz International Proceedings in Informatics (LIPIcs), vol. 236, pp. 8:1–8:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2022). <https://doi.org/10.4230/LIPIcs.SAT.2022.8>. <https://drops.dagstuhl.de/opus/volltexte/2022/16682>
87. Nadel, A., Ryvchin, V.: Efficient SAT solving under assumptions. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 242–255. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_19
88. Nadel, A., Ryvchin, V.: Chronological backtracking. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 111–121. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_7
89. Nadel, A., Ryvchin, V., Strichman, O.: Preprocessing in incremental SAT. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 256–269. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_20
90. Nadel, A., Ryvchin, V., Strichman, O.: Ultimately incremental SAT. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 206–218. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_16
91. Neiman, D., Mackey, J., Heule, M.J.H.: Tighter bounds on directed ramsey number $R(7)$. *Graphs Comb.* **38**(5), 156 (2022). <https://doi.org/10.1007/S00373-022-02560-5>
92. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, 17–22 July

- 2023, *Proceedings, Part II. Lecture Notes in Computer Science*, vol. 13965, pp. 3–17. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-37703-7_1
93. Niemetz, A., Preiner, M., Barrett, C.W.: Murkla: a modular and highly extensible API fuzzer for SMT solvers. In: Shoham, S., Vizel, Y. (eds.) *CAV (2)*. *Lecture Notes in Computer Science*, vol. 13372, pp. 92–106. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-13188-2_5
 94. Niemetz, A., Preiner, M., Biere, A.: Model-based API testing for SMT solvers. In: *SMT. CEUR Workshop Proceedings*, vol. 1889, pp. 3–14. CEUR-WS.org (2017)
 95. Niemetz, A., Preiner, M., Biere, A.: Boolector at the SMT competition 2019. In: Hendrix, J., Sharygina, N. (eds.) *Proceedings of the 17th International Workshop on Satisfiability Modulo Theories (SMT 2019)*, affiliated with the 22nd International Conference on Theory and Applications of Satisfiability Testing (SAT 2019), Lisbon, Portugal, 7–8 July 2019, p. 2 (2019)
 96. Park, J., Winterer, D., Zhang, C., Su, Z.: Generative type-aware mutation for testing SMT solvers. *Proc. ACM Program. Lang.* **5**(OOPSLA), 1–19 (2021)
 97. Paxian, T., Reimer, S., Becker, B.: Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) *SAT 2018*. *LNCS*, vol. 10929, pp. 37–53. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_3
 98. Piette, C., Hamadi, Y., Sais, L.: Vivifying propositional clausal formulae. In: Ghalab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N.M. (eds.) *ECAI 2008 - 18th European Conference on Artificial Intelligence*, Patras, Greece, 21–25 July 2008, *Proceedings. Frontiers in Artificial Intelligence and Applications*, vol. 178, pp. 525–529. IOS Press (2008). <https://doi.org/10.3233/978-1-58603-891-5-525>
 99. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) *SAT 2007*. *LNCS*, vol. 4501, pp. 294–299. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72788-0_28
 100. Pollitt, F., Fleury, M., Biere, A.: Faster LRAT checking than solving with CaDiCaL. In: Mahajan, M., Slivovsky, F. (eds.) *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*. *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 271, pp. 21:1–21:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl (2023). <https://doi.org/10.4230/LIPIcs.SAT.2023.21>
 101. Sanders, P., Schreiber, D.: Mallob: scalable SAT solving on demand with decentralized job scheduling. *J. Open Source Softw.* **7**(77), 4591 (2022). <https://doi.org/10.21105/JOSS.04591>
 102. Scott, J., Sudula, T., Rehman, H., Mora, F., Ganesh, V.: BanditFuzz: fuzzing SMT solvers with multi-agent reinforcement learning. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) *FM 2021*. *LNCS*, vol. 13047, pp. 103–121. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_6
 103. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Rutenbar, R.A., Otten, R.H.J.M. (eds.) *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996*, San Jose, CA, USA, 10–14 November 1996, pp. 220–227. IEEE Computer Society/ACM (1996). <https://doi.org/10.1109/ICCAD.1996.569607>
 104. Soos, M., Devriendt, J., Gocht, S., Shaw, A., Meel, K.S.: CryptoMiniSat with ccanr at the SAT competition 2020. In: *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions 2020*, vol. 27 (2020)

105. Soos, M., Gocht, S., Meel, K.S.: Tinted, Detached, and Lazy CNF-XOR solving and its applications to counting and sampling. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 463–484. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_22
106. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 244–257. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02777-2_24
107. Soos, M., Selman, B., Kautz, H., Devriendt, J., Gocht, S.: CryptoMiniSat with WalkSAT at the SAT competition 2020. In: Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions, p. 29 (2020)
108. Subercaseaux, B., Heule, M.J.H.: The packing chromatic number of the infinite square grid is 15. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, 22–27 April 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13993, pp. 389–406. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30823-9_20
109. van der Tak, P., Ramos, A., Heule, M.J.H.: Reusing the assignment trail in CDCL solvers. *J. Satisf. Boolean Model. Comput.* **7**(4), 133–138 (2011). <https://doi.org/10.3233/SAT190082>
110. Tan, Y.K., Heule, M.J.H., Myreen, M.O.: `cake_lpr`: verified propagation redundancy checking in CakeML. In: TACAS 2021. LNCS, vol. 12652, pp. 223–241. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_12
111. Wetzler, N., Heule, M.J.H., Hunt, W.A.: DRAT-trim: efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) SAT 2014. LNCS, vol. 8561, pp. 422–429. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09284-3_31
112. Whittimore, J., Kim, J., Sakallah, K.A.: SATIRE: a new incremental satisfiability engine. In: Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, 18–22 June 2001, pp. 542–545. ACM (2001). <https://doi.org/10.1145/378239.379019>
113. Wieringa, S., Niemenmaa, M., Heljanko, K.: Tarmo: A framework for parallelized bounded model checking. In: Brim, L., van de Pol, J. (eds.) Proceedings 8th International Workshop on Parallel and Distributed Methods in verification, PDMC 2009, Eindhoven, The Netherlands, 4 November 2009. EPTCS, vol. 14, pp. 62–76 (2009). <https://doi.org/10.4204/EPTCS.14.5>
114. Yolcu, E., Aaronson, S., Heule, M.J.H.: An automated approach to the collatz conjecture. *J. Autom. Reason.* **67**(2), 15 (2023). <https://doi.org/10.1007/S10817-022-09658-8>
115. Zhang, L.: On subsumption removal and on-the-fly CNF simplification. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 482–489. Springer, Heidelberg (2005). https://doi.org/10.1007/11499107_42
116. Zhang, T., Szeider, S.: Searching for smallest universal graphs and tournaments with SAT. In: Yap, R.H.C. (ed.) 29th International Conference on Principles and Practice of Constraint Programming, CP 2023, Toronto, Canada, 27–31 August 2023. LIPICs, vol. 280, pp. 39:1–39:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). <https://doi.org/10.4230/LIPICs.CP.2023.39>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

