Clausal Congruence Closure

Armin Biere
University Freiburg, Germany

Katalin Fazekas
TU Wien, Vienna, Austria

Mathias Fleury
University Freiburg, Germany

Nils Froleyks
Johannes Kepler University, Linz, Austria

Abstract

Many practical applications of satisfiability solving employ multiple steps to encode an original problem formulation into conjunctive normal form. Often circuits are used as intermediate representation before encoding those circuits into clausal form. These circuits however might contain redundant isomorphic sub-circuits. If blindly translated into clausal form, this redundancy is retained and increases solving time unless specific preprocessing algorithms are used. Furthermore, such redundant sub-formula structure might only emerge during solving and needs to be addressed by inprocessing. This paper presents a new approach which extracts gate information from the formula and applies congruence closure to match and eliminate redundant gates. Besides new algorithms for gate extraction, we also describe previous unpublished attempts to tackle this problem. Experiments focus on the important problem of combinational equivalence checking for hardware designs and show that our new approach yields a substantial gain in CNF solver performance.

2012 ACM Subject Classification Theory of Computation → Automated Reasoning

Keywords and phrases Satisfiability Solving, Congruence Closure, Structural Hashing, SAT Sweeping, Conjunctive Normal Form, Combinational Equivalence Checking, Hardware Equivalence Checking

Digital Object Identifier 10.4230/LIPIcs.SAT.2024.N

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.11652423`
Dataset (Experimental Data: Log files, Plots, Tables): `https://doi.org/10.5281/zenodo.11658133`
Dataset (SAT Competition 2022 Benchmarks): `https://doi.org/10.5281/zenodo.11428010`
Dataset (SAT Competition 2023 Benchmarks): `https://doi.org/10.5281/zenodo.11426992`
Dataset (CNF encoded HWMCC'12 Miters): `https://doi.org/10.5281/zenodo.10823128`
Dataset (CNF encoded IWLS'22 Miters): `https://doi.org/10.5281/zenodo.10823099`

Funding This work was supported by the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, the Austrian Science Fund (FWF) under project No. T-1306, and by a gift from Intel Corporation.

1 Introduction

One of our motivations is to improve SAT solving for combinational equivalence checking of hardware circuits [30, 54, 63]. For decades combinational equivalence checking was considered the most successful application of formal verification in industry, actually before the SAT revolution started. Earlier approaches in the last century relied on binary decision diagram (BDD) technology, i.e., BDD sweeping [53], which however has been combined (if not replaced) with SAT sweeping [54] in this century. There are various commercial providers of equivalence checkers, including major electronic design automation (EDA) vendors such as Synopsys, Cadence, and Siemens, with widespread use in chip design.

27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024).
Editors: Supratik Chakraborty and Jie-Hong Roland Jiang; Article No. N; pp. N:1–N:26
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Even though details about the inner workings of these EDA commercial equivalence checkers are not publicly available, simply encoding large equivalence checking problems into a monolithic SAT formula in conjunctive normal form (CNF) and then using a stand-alone solver to solve them does not scale. Therefore, we submitted monolithic equivalence checking benchmarks to the SAT Competition already in 2013 [24]. These benchmarks are regularly used in SAT competitions (for instance two of them in 2022) and some are still challenging.

It is fair to assume that commercial equivalence checkers use a hybrid approach, where the circuit structure guides incremental SAT queries to establish correspondence between internal sub-circuits, as a recursive process following the topological order of the circuit. These hybrid approaches to combinational equivalence checking have their own challenges [1, 70–72] and, in our view, are not a solved problem.

Furthermore, improving plain CNF-level SAT solving on such instances will be beneficial for hybrid approaches as well. Techniques useful for equivalence checking can have a positive impact on other applications of SAT too.

The question remains why state-of-the-art SAT solvers working on CNF need that guidance and are not able to efficiently find proofs for large equivalence checking problems, actually also called *miters* [30], even though, at the end, also those hybrid approaches just rely on the resolution proof system. While short proofs exist in theory, even for the simplest equivalence checking task of comparing two identical circuits, current state-of-the-art solvers based on the conflict-driven clause learning (CDCL) paradigm [26] fail to find short resolution proofs, as we have shown in previous work [45].

Equivalence checking of arithmetic circuits [12, 52] has similar applications and issues. In principle, algebraic techniques [34, 51, 61] can solve them, but they remain extremely challenging if given in CNF. Therefore, we consider arithmetic circuit verification out-of-scope for this study. We further focus on combinational equivalence checking leaving sequential equivalence checking, which relates to hardware model checking, to future work. Our goal is to improve CNF SAT solving for combinational (non-arithmetic) equivalence checking.

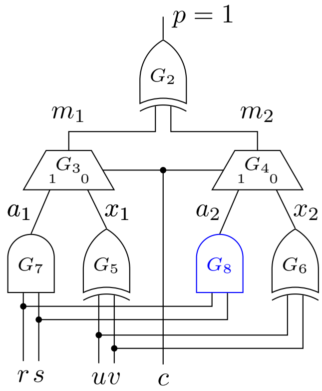
We consider *isomorphic miters*, the problem that encodes equivalence checking of two identical copies of a circuit, but also will take a look at the comparison of non-isomorphic circuits. The latter are actually the main target in industrial applications of equivalence checking, where a synthesized and optimized circuit and the original unsimplified circuit are compared. These *optimized miters* are much harder to solve.

The real cause for this failure of CDCL to solve isomorphic miters encoded into CNF is unclear, but proven empirically, as our experiments confirm. We can offer two explanation attempts though. First Yakau Novikau suggested at the Dagstuhl seminar on "The Theory and Practice of SAT Solving" in 2015 that, due to the recursive nature of equivalence checking, to learn an internal equivalence (two binary clauses) the SAT solver must fully restart in-between learning the two clauses. As a consequence, which again only empirically has been confirmed, solving miters in CNF greatly benefits from rapid restarts, i.e., restarting after each conflict. The second observation is that SAT solvers on miters even for isomorphic circuits learn rather long clauses, followed by shorter and shorter clauses until they learn some binary clauses. But then the whole process repeats, while a guided approach can focus on learning the necessary binary clauses directly.

While when working on circuits directly the gates are explicitly present, new gates can appear during solving. Our experiments on the SAT Competition 2022 shows that many SAT problems have gates, partially due to Tseitin encoding and redundant isomorphic structures. Therefore, it makes sense to have our technique on the CNF side directly: our implementation in Kissat identifies more than 180 million congruent variables.

$p \underset{1}{=} 1$
$p \underset{2}{=} m_1 \oplus m_2$
$m_1 \underset{3}{=} c \; ? \; a_1 : x_1$
$m_2 \underset{4}{=} c \; ? \; a_2 : x_2$
$x_1 \underset{5}{=} u \oplus v$
$x_2 \underset{6}{=} u \oplus v$
$a_1 \underset{7}{=} r \wedge s$
$a_2 \underset{8}{=} r \wedge s$

(a) gates $G_1, \ldots, G_8$



(b) miter circuit

$(p)_1$
$(\overline{p}\, m_1 m_2)_2\, (\overline{p}\, \overline{m}_1 \overline{m}_2)_3\, (p\, \overline{m}_1 m_2)_4\, (p\, m_1 \overline{m}_2)_5$
$(\overline{m}_1 \overline{c}\, a_1)_6\, (\overline{m}_1 c\, x_1)_7\, (m_1 \overline{c}\, \overline{a}_1)_8\, (m_1 c\, \overline{x}_1)_9$
$(\overline{m}_2 \overline{c}\, a_2)_{10}\, (\overline{m}_2 c\, x_2)_{11}\, (m_2 \overline{c}\, \overline{a}_2)_{12}\, (m_2 c\, \overline{x}_2)_{13}$
$(\overline{x}_1 \overline{u}\, \overline{v})_{14}\, (\overline{x}_1 u\, v)_{15}\, (x_1 \overline{u}\, v)_{16}\, (x_1 u\, \overline{v})_{17}$
$(\overline{x}_2 \overline{u}\, \overline{v})_{18}\, (\overline{x}_2 u\, v)_{19}\, (x_2 \overline{u}\, v)_{20}\, (x_2 u\, \overline{v})_{21}$
$(\overline{a}_1 r)_{22}\, (\overline{a}_1 s)_{23}\, (a_1 \overline{r}\, \overline{s})_{24}$
$(\overline{a}_2 r)_{25}\, (\overline{a}_2 s)_{26}\, (a_2 \overline{r}\, \overline{s})_{27}$

(c) CNF with clauses $C_1, \ldots, C_{27}$

Figure 1 Example of an equivalence checking problem for two identical (isomorphic) circuits consisting each of one AND, XOR, and ITE (multiplexer/if-then-else) gate. The miter circuit in the middle (b) compares the output of the two circuits and assumes they are different by feeding them into another XOR gate which in turn is assumed to produce the output value 1. The equational semantics (a) is shown on the left which after Tseitin encoding [67] gives the CNF (c), e.g., the last AND gate G_8 in the second circuit is encoded by the last three clauses C_{25}, C_{26} and C_{27}.

2 Preliminaries

We assume that the reader is familiar with propositional satisfiability (SAT) and otherwise refer to [25]. In order to save space we abbreviate formulas in conjunctive normal form (CNF) by omitting operators if they are clear from the context. For instance we use $(\overline{a}r)(\overline{a}s)(a\overline{r}\,\overline{s})$ to denote the CNF $(\overline{a} \vee r) \wedge (\overline{a} \vee s) \wedge (a \vee \overline{r} \vee \overline{s})$. We identify a double negated literal with itself and denote with $|l|$ the variable v of a positive literal $l = v$ or negative literal $l = \bar{v}$.

In Fig. 1, we present an example of a combinational equivalence checking problem (miter). This is an isomorphic miter as the two circuits compared are identical. In the experiments we also consider the case where one of the circuits is an optimized version of the other, since these optimized miters are the main target in industrial applications of equivalence checking.

Hybrid approaches to equivalence checking (starting from [54] and most recently [72]) keep the two circuits alongside the CNF encoding in the SAT solver. During parsing such an isomorphic miter from a file, they will already detect all equivalences and simplify both circuits to one representation by applying "structural hashing".

This technique is also called "hash consing" in implementations of functional programming languages or "common sub-expression elimination" in compiler optimization. It is also implemented in libraries for the manipulation of binary decision diagrams (BDDs) [33] or and-inverter graphs (AIGs) [54] in the form of a "unique-table".

The basic idea of our approach is to simulate structural hashing by deriving from the CNF through resolution binary clauses of the equivalence of literals representing outputs of equivalent gates: For the two AND gates G_7 and G_8 in Fig. 1 we first derive $a_1 = a_2$, i.e., the binary clauses $(\overline{a}_1 \vee a_2)$ and $(a_1 \vee \overline{a}_2)$. Then we derive $x_1 = x_2$ for the two XOR gates G_5 and G_6. This allows us two replace the inputs a_2 and x_2 of the second ITE gate G_4 by a_1 and x_1 which in turn yields $m_1 = m_2$. Substituting m_2 with m_1 in the right hand side (RHS) of gate G_2 simplifies to 0, which contradicts the assumption that the outputs of the two compared circuits are different ($p = 1$).

We show first how such a simulation is feasible starting from a CNF encoding and second how our new congruence closure approach solves isomorphic miters instantly. In the second scenario, when checking optimized miters, it is further expected that during solving often identical sub-circuits emerge. Our approach then allows to simplify the problem through inprocessing, which reduces over-all solving time, as confirmed in our experiments.

Related to clausal congruence closure is SAT sweeping. It has only been described in our solver description [21, 22] and uses the "small" SAT solver Kitten within Kissat to prove the equivalence of two literals. It can simulate congruence closure (if the variables are scheduled in the right order), but it is more expensive as it relies on Kitten as SAT oracle. However, it is also stronger, because it is not limited to matching gates syntactically.

Our new implementation in Kissat with efficient algorithms for gate extraction runs congruence closure until completion during both pre- and inprocessing, even for the largest CNFs in the SAT competition. We enable it by default without limit, in contrast to our earlier attempts to solve isomorphic miters including "lazy hyper binary resolution" [9], "tree-based look ahead" [45], "simple probing" (see next Sect. 3), "blocked-clause decomposition" [44], and "internal SAT sweeping" [21, 22], which all need to be limited or preempted.

3 Simple Probing

Simple probing is available in Lingeling since 2012 [10] motivated by the observation [45] that though hyper binary resolution (HBR) [3, 4, 42] combined with equivalent literal substitution (ELS) [2, 36, 56, 68] in theory can solve identical miters, in practice it fails to do so.

The problem with existing HBR implementations [3, 4, 42, 45] is that they are "global" and rely on complete failed literal probing, followed or interleaved with a global form of ELS. This means that all literals are probed and all binary clauses are taken into account in finding and substituting equivalent literals. For isomorphic miters, the fix-point of this process is only reached after many rounds of HBR and ELS. The main idea behind "simple probing" is to apply HBR and ELS steps only locally to avoid some unnecessary work.

Continuing with the example in Fig. 1, we resolve the 6 clauses $C_{22}, \dots C_{27}$ of the two AND gates G_7 and G_8 through two hyper-binary resolution steps:

$$\frac{(a_1\overline{r}\,\overline{s})_{24} \quad (\overline{a}_2 r)_{25} \quad (\overline{a}_2 s)_{26}}{(a_1\overline{a}_2)_{28}}\ \text{HBR}_1 \qquad \frac{(a_2\overline{r}\,\overline{s})_{27} \quad (\overline{a}_1 r)_{22} \quad (\overline{a}_1 s)_{23}}{(a_2\overline{a}_1)_{29}}\ \text{HBR}_2$$

These two hyper binary resolution steps yield the equivalence $a_1 = a_2$, represented by the two resolvents, and correspond to the following two linear chains of resolution (RES) steps:

$$\frac{\dfrac{(a_1\overline{r}\,\overline{s})_{24} \quad (\overline{a}_2 r)_{25}}{(a_1\overline{a}_2\overline{s})}\ \text{RES} \qquad (\overline{a}_2 s)_{26}}{(a_1\overline{a}_2)_{28}}\ \text{RES} \qquad \frac{\dfrac{(a_2\overline{r}\,\overline{s})_{27} \quad (\overline{a}_1 r)_{22}}{(a_2\overline{a}_1\overline{s})}\ \text{RES} \qquad (\overline{a}_1 s)_{23}}{(a_2\overline{a}_1)_{29}}\ \text{RES}$$

Note that such linear resolution chains correspond to reverse-unit propagation (Rup) [41] in clausal proofs [46, 47]. Next we have to substitute (w.l.o.g.) a_2 by a_1 in the formula:

$$\frac{(\overline{m}_2\overline{c}\,a_2)_{10} \quad (a_1\overline{a}_2)_{28}}{(\overline{m}_2\overline{c}\,a_1)_{30}}\ \text{RES} \qquad \frac{(m_2\overline{c}\,\overline{a}_2)_{12} \quad (a_2\overline{a}_1)_{29}}{(m_2\overline{c}\,\overline{a}_1)_{31}}\ \text{RES}$$

This again boils down to resolution, which also explains why simple probing can produce Rup proofs [41] easily. Also C_{25}, C_{26}, and C_{27} of the AND gate G_8 of the circuit on the right should be substituted, but the result would be identical to the already existing clauses C_{22}, C_{23} and C_{24} of the equivalent gate G_7 of the circuit on the left, and should be avoided. Instead, they should just be deleted, the main feature in Drup which extends the Rup proof system by including "deletion" information [69] to speed-up proof checking.

Algorithm 2 Pseudo code of "simple probing" from Lingeling through local hyper binary resolution (HBR) and eager equivalent literal substitution (ELS): We interpret the given CNF F as a set of clauses, which in turn are sets of literals, with no duplicates. With $|r| \neq |l|$ in Line 10 we assume that the variables of r and l are different. Line 13 performs the actual ELS by replacing all occurrences of l with the representative literal r (resp. $\bar{l}$ by $\bar{r}$). In the actual implementation, we consider additional cases, e.g., we check for hyper binary resolved units when $\gamma(r) = |C|$ in Line 10.

simple-probing (CNF F) // by reference, i.e., F updated in place

1 literals L = all literals in F
2 candidates $\Lambda = L$
3 **while** $\Lambda \neq \emptyset$
4 pick and remove $l \in \Lambda$
5 **for all** "base" clauses $C \in F$ with $|C| > 2$ and $l \in C$
6 **for all** literals $k \in C$
7 counts $\gamma \colon L \to \mathbb{N}$ initialized to $\gamma \equiv 0$
8 **for all** binary clauses $(o \vee \bar{k}) \in F$
9 $\gamma(o)$++ // increment count of other literal o by one
10 **for all** r with $\gamma(\bar{r}) + 1 = |C|$ and $|r| \neq |l|$ and $(\bar{r} \vee l) \notin F$
11 add $(\bar{r} \vee l)$ to F // HBR
12 **if** $(r \vee \bar{l}) \in F$ // checking for dual clause - ELS
13 substitute $l = r$ in all clauses $D \in F$ with l or $\bar{l}$ in D
14 reschedule literals in resulting clauses by adding them to Λ
15 continue with outer **while** loop at Line 3

This forms the core of simple probing. In the implementation we use a counting argument: we find "immediate" hyper binary resolvents by counting how often a literal occurs in binary clauses which can be resolved with a given non-binary *base clause*. For the base clause C_{24}, we only consider the two binary clauses C_{25} and C_{26} as resolution candidates because we can ignore the blocked clauses C_{22} and C_{23} (as they both contain $\bar{a}_1$). The literal $\bar{a}_2$ occurs twice in them, and, since the base clause has one literal more than the occurrence count, this yields C_{28} through HBR_1. Similarly, we get C_{29} using C_{27} as base clause.

Whenever we find a new hyper-resolvent this way, without adding duplicates, we check whether its dual clause with both literals negated already exists. For instance, assume that in our example applying HBR_1 first. Then, when clause C_{29} is derived through HBR_2, as its dual (C_{28}) already exists, the equivalence $a_2 = a_1$ is derived. To substitute one literal with the other, we traverse all clauses containing the literal to substitute, apply the substitution, and delete the original clause. While checking for dual clauses only requires finding all binary clauses in which a literal occurs, the substitution step requires full occurrence lists.

The complete preprocessing algorithm in Alg. 2 needs to determine which and when clauses are (re)considered as base clauses. As clauses are eagerly removed and added in this approach, we do not want to use base clauses as scheduling objects in a working queue. Instead, we opted for our implementation in Lingeling to have literals occurring in base clause candidates as scheduling objects. Initially, all literals are candidate literals for simple probing. For each candidate, we go through all its non-binary clauses (requiring occurrence lists) and then apply the two-step procedure described above. After finding and substituting equivalence, we reschedule literals occurring in the resulting clauses.

Simple probing will solve isomorphic miters of circuits with only AND gates. Actually, after substituting the equivalence of outputs of the compared circuits, the comparison in clauses of the miter XOR gates will yield a unit clause. We would need to propagate those units to derive unsatisfiability (unless each compared circuit has only one output).

However, even though simple probing implicitly treats OR as AND gates, it does not handle other more complex gates, particularly neither XOR nor ITE gates. Actually, HBR+ELS alone cannot solve such miters with XORs and ITEs, including our example, as already observed by Heule at al. [45]. They proposed to interleave probing based HBR+ELS with saturating *ternary resolution* (TRN) [28] to simulate structural hashing for XOR and ITE gates, i.e., add all resolvents of at most length three between ternary gates.

Such ternary resolution is rather costly, particularly if run until completition. Thus it needs to be localized in combination with simple probing and also does not work for larger XOR gates with more than two inputs. Nevertheless, CaDiCaL [23] and Lingeling [11] both implement (non-localized) TRN but not eagerly and in a limited way.

4 Gate Extraction

Previous attempts (including simple probing) to solve CNF-encoded isomorphic miters through HBR (with ELS and TRN) essentially failed. They are orders of magnitude slower than circuit-based techniques, as already pointed out in the conclusion of [45] and again confirmed in our experiments. The key to obtaining a scalable algorithm is to extract "gates" from the CNF instead, also called "macros" and "(functional) definitions" in related work. This takes us halfway to the reconstruction of the original circuit, except that we do not care about the topological order, nor do we try to find global (primary) inputs or outputs.

Gate extraction goes back to [40, 58, 60] and we refer to the preprocessing chapter of the SAT handbook [26, Sect. 9.6.2] for details. These works were either limited in scope or had as goal to recover an actual circuit, including inputs and outputs as well as topologically ordering extracted gates. This is actually a difficult problem in general, as for instance XOR constraints (and inverters) are not directed, i.e., the Tseitin encoding of an XOR gate of arity n is symmetric in all variables and allows to actually extract $n + 1$ gates. Even for Tseitin-encoded AIGs, which are circuits with only AND gates (and inverters), there are problems. First, constant inputs might turn binary AND gates into unary AND gates (buffer/equivalences/inverters), which have to be ordered. Second, the same clause can be used for extracting multiple gates, which requires selecting a gate.

Recent work stays on the CNF level and uses blocked clause decomposition (BCD) instead [5, 44, 49, 50]. Note, however, that this approach does not support either in-processing or the production of proofs. A basic XOR-constraint extraction algorithm is described in [64] with the goal to enable algebraic reasoning. Gate detection has also been used extensively during SAT preprocessing to filter out resolvents in bounded variable elimination [37, 39]. In that context, it is local to the candidate variable for elimination and thus other algorithms apply. Similar gate-extraction approaches exist in richer logics (#SAT and QBF) too [55, 62].

We only syntactically extract "gates", trying to reverse the CNF encoding, e.g., from the clauses C_{22}, C_{23}, and C_{24} in the CNF of Fig. 1 we extract the "gate" (equation) $a_1 = r \wedge s$. Semantic extraction (such as [39, 62]) is much more powerful, but also much more expensive.

5 AND-Gate Extraction

Our *basic-and-gate-extraction* algorithm is shown in Alg. 3. For each non-binary base clause, it first marks the negation of all its literals. Then, for each literal in the clause, we traverse all binary clauses in which it occurs negatively. If the number of other marked literals in

Algorithm 3 Basic algorithm for extracting AND gates. As in Alg. 2 correctness hinges on the assumption that F is without trivial clauses and all its clauses, as well as F, are interpreted as sets without duplicates. Thus, in the implementation, one must remove duplicated binary clauses first. Only binary clauses need to be watched, assuming base clauses can be traversed in some other way.

basic-and-gate-extraction (CNF F)

1 resulting AND gates $A = \emptyset$
2 literals L = all literals in F
3 **for all** clauses $C \in F$ with $|C| > 2$
4   marks $\mu\colon L \to \mathbb{B}$ initialized to $\mu \equiv \bot$ // implemented as bit-map
5   **for all** literals r with $\overline{r} \in C$
6     $\mu(r) = \top$
7   **for all** literals $l \in C$
8     $n = 0$
9   **for all** binary clauses $(\bar{l} \vee r) \in F$
10   **if** $\mu(r)$ **then** n++
11   **if** $n = |C| - 1$
12   let $(l \vee \overline{r}_1 \vee \ldots \vee \overline{r}_n) = C$ // structured binding
13   add AND gate $(l = r_1 \wedge \cdots \wedge r_n)$ to A
14 **return** A

those binary clauses is one less than the size of the base clause, we have found an AND gate. However, for large formulas with millions of variables, millions of binary, and candidate clauses,[1] this algorithm is too slow to run until completion in order to solve miters.

In a failed improvement attempt, we added all binary clauses to a hash table, such that we can directly search for $(\bar{l} \vee r)$ when considering l as the left-hand side literal for all other r with $\overline{r} \in C$ instead of marking (Lines 5–10). However, it turns out that for large formulas, filling the hash-table took the same amount of time as the marking variant in Alg. 3.

Our first successful improvement counts the number of occurrences of literals in binary clauses and drops candidate clauses C where no literal has enough negative occurrences in binary clauses. Actually, iterations of the loop in Lines 7–13 can always be skipped in Alg. 3 for literals l where $\bar{l}$ occurs less than $|C| - 1$ times in binary clauses. Our second improvement uses the observation that, while considering the left-hand-side (LHS) candidate l in Line 7 of that loop and traversing binary clauses $(\bar{l} \vee r)$ in Line 9, all remaining LHS candidates $l' \in C$ not yet tried still need to occur negated as one of these r.

For example, let $C = (l_1 \vee l_2 \vee l_3)$ in Line 3. Assume $\bar{l}_1$ occurs only once in binary clauses, and thus is skipped. Further, let $(\bar{l}_2 \vee r_1)$ and $(\bar{l}_2 \vee r_2)$ be the only binary clauses with $\bar{l}_2$ when iterating over $l = l_2$ in Line 7. If neither $l_3 = \overline{r}_1$ nor $l_3 = \overline{r}_2$ then l_3 is no LHS candidate as $(\bar{l}_2 \vee \bar{l}_3)$ is missing. To implement this optimization, we use two mark bits for the negation of literals in C. The first mark plays the same role as μ in Alg. 3 while the second is used to mark the negation of remaining LHS candidates. When counting occurrences of marked literals in Lines 9–10 we update the second mark bit and later only consider LHS literals which have the second bit still set.

[1] See e.g., `SAT_MS_sat_nurikabe_p16.pddl_166` from the main track of the SAT Competition 2022 with 19 million variables, 199 million binary clauses and 14 million candidate base clauses.

Algorithm 4 This is a basic algorithm for XOR-gate extraction. It uses the bit-extraction function β to determine if the bit at a given bit position is set and π to compute its parity.

basic-xor-gate-extraction (CNF F)
1 resulting XOR gates $X = \emptyset$
2 let $\beta \colon \mathbb{N} \times \mathbb{N} \to \{0,1\}$ with $\beta(i,s) = (s/2^i) \bmod 2$ // extract i^{th} bit from s
3 let $\pi \colon \mathbb{N} \to \{0,1\}$ with $\pi(s) = |\{i \mid \beta(i,s) = 1\}| \bmod 2$ // parity of all "bits" in s
4 **for all** clauses $C = (l_0 \vee \ldots \vee l_{m-1}) \in F$ with $|C| > 2$
5   **for** $s = 2$ **to** $2^m - 1$ with $\pi(s) = 0$ // flip an even number of sign bits
6     $D = \{l_i \mid \beta(i,s) = 0\} \cup \{\bar{l}_i \mid \beta(i,s) = 1\}$ // negate l_i if i^{th} bit set
7   **if** $D \notin F$ continue with outer loop at Line 4 // clause missing
8   **for** $i = 0$ **to** $m - 1$ // add m XOR gates of arity $m - 1$
9   let $(l_i \vee k_1 \vee \ldots \vee k_{m-1}) = C$ and $l = \bar{l}_i$
10   add XOR gate $(l = k_1 \oplus \cdots \oplus k_{m-1})$ to X
11 **return** X

6 XOR-Gate Extraction

As with AND-gate extraction, there is little published work on XOR extraction. It is briefly mentioned in [10] to support Gaussian elimination and a preliminary form of congruence closure in LINGELING for the SAT Challenge 2012. Both CADICAL since 2019 [13] and KISSAT since 2020 [20] use XOR-gate extraction to make bounded variable elimination more effective, as originally proposed in [37] for AND gates. Our basic algorithm in Alg. 4 follows these implementations and corresponds to a similar algorithm presented in [64].

In Lines 5–7, we check that all clauses D are present in the CNF which differ from the base clause C by negating exactly an even number of literals. If this is the case, we have found the XOR *constraint* $1 = l_0 \oplus \cdots \oplus l_{m-1}$, falsified by the same assignment which falsifies C (assigning all literals of C to false). This constraint can now be rewritten into those m XOR gates added on Line 10, by removing l_i from the right-hand-side (RHS) of the constraint and replacing its LHS with the negation l of l_i ("1" on the LHS above acts as negation).

The reason for adding all m gates is that we cannot (and do not want to) order symmetric gates, where input and output can be exchanged. Consequently, the functional dependency graph between inputs and outputs of our extracted gates becomes cyclic as soon as a single XOR constraint is extracted and covers all the gates. Being able to handle such cyclic dependencies is an important feature of congruence closure in our approach, which is not possible when gate extraction is used to reconstruct the structure of circuits [40, 60].

Note that for each XOR constraint found for a base clause C with m literals, the CNF actually needs to contain $2^{m-1} - 1$ matching D clauses, but we only extract m gates from it. So even for $m = 3$, we extract only three gates covering four clauses. Nevertheless, the basic algorithm performs redundant work, since Line 4 does not detect when C was already used as a matching D clause in a successful extraction before.

We can avoid this redundant work by considering in Line 4 only one of the clauses that encodes an XOR gate. Assume we have a strict order over variables, for instance, by using the integer encoding of variables in the DIMACS format. Then, C can be skipped in Line 4 unless either all literals of it are positive or only the largest one is negative. This amounts to the condition $l_0 = |l_0| < l_1 = |l_1| < \cdots < l_{m-2} = |l_{m-2}| < |l_{m-1}|$ on C in Line 4.

Note that the number of clauses needed to encode an XOR gate of arity n is 2^n, i.e., grows exponentially. As clauses in the encoding have size $m = n + 1$, we can therefore further

Algorithm 5 This is a basic algorithm for ITE-gate extraction. To find ITE gates with a *given* LHS literal l, as in variable elimination, the outer loop at Line 2 would only go over clauses with l.

basic-ite-gate-extraction (CNF F)
1 resulting ITE gates $I = \emptyset$
2 **for all** ternary clauses $C = (l_1 \vee l_2 \vee l_3) \in F$
3   **for** $i = 1 \ldots 3$
4   let $(\bar{c} \vee \bar{l} \vee t) = C$ with $c = \bar{l}_i$
5   **if** $(\bar{c} \vee l \vee \bar{t}) \notin F$ continue with next i at Line 3
6   **for all** ternary clauses $(c \vee \bar{l} \vee e) \in F$
7   **if** $(c \vee l \vee \bar{e}) \in F$
8   add ITE gate $(l = c\,?\,t : e)$ to I
9 **return** I

limit the size of the base clauses in Line 4 in Alg. 4. In practice, we did not see any need to search for XOR gates of arity larger than the run-time parameter $N_{\text{XOR}} = 4$.

Furthermore, as with AND gates, the XOR-extraction algorithm can be improved by counting occurrences of literals in clauses that can be part of the encoding of an XOR gate. Base clauses of size $m = |C|$ considered for extracting an XOR gate of arity $n = m - 1$ can be skipped if C contains a literal that has less than 2^{n-1} occurrences.

Finally, we realized that after counting the number of occurrences of literals in all clauses, some clauses end up having literals with too few occurrences in the reduced set of considered clauses and thus should not be considered anymore. Therefore, recounting might find additional clauses to skip. This process can be repeated until fix-point, but most of the reduction is achieved after two rounds of counting (the run-time parameter we are using).

For checking $D \notin F$ in Line 7, we connect all remaining clauses that can potentially be part of an XOR gate encoding through full occurrence lists. Searching for D can then be restricted to traverse the occurrence list of the literal in D with the minimum number of occurrences, as in backward-subsumption checks [37]. Using hashing instead (still a compile-time parameter) has similar negative results as for AND-gate extraction.

7 ITE-Gate Extraction

The most common type of encoded gates are AND gates, followed by XOR gates. Except for a few applications where they are frequent, such as describing BDDs, ITE gates occur much less often. However, occasionally it can be crucial to handle ITE gates efficiently. For example, for one of the hard synthesized miters that we considered in our experiments (`test02` from [72]) it gave a 1000 × improvement in solving time: 1.79 seconds when extracting vs. 2023.41 seconds when not extracting ITE gates (*cf.* Tab. 13, and Fig. 8 and 9).

As with AND and XOR gates we have been using a simple algorithm for ITE-gate extraction in the context of variable elimination for many years, i.e., where the variable of the LHS literal is fixed. A potential variant to extract all ITE gates in a given formula is shown in Alg. 5. To encode an ITE gate $(l = c\,?\,t : e)$ exactly the following four ternary clauses are needed $(\bar{c} \vee \bar{l} \vee t)$, $(\bar{c} \vee l \vee \bar{t})$, $(c \vee \bar{l} \vee e)$, and $(c \vee l \vee \bar{e})$ ignoring two potential additional redundant clauses $(\bar{l} \vee t \vee e)$ and $(l \vee \bar{t} \vee \bar{e})$, which might be used to improve arc-consistency of the encoding. Observe that the first two clauses encode the conditional equivalence $c \rightarrow l = t$ and the third and fourth the conditional equivalence $\bar{c} \rightarrow l = e$.

Algorithm 6 Fast ITE-gate extraction based on matching conditional equivalences.

find-conditional-equivalences (CNF F, literal c)
1 resulting conditional equivalences $E = \emptyset$
2 **for all** ternary clauses $C = (\bar{c} \vee \bar{l} \vee t) \in F$
3 **if** $(\bar{c} \vee l \vee \bar{t}) \in F$
4 add $l = t$ to E
5 **return** E

merge-conditional-equivalences (literal c, equivalences E^+, equivalences E^-)
6 resulting ITE gates $I = 0$
7 **for all** equivalences $l = t$ in E^+
8 **for all** equivalences $l = e$ in E^-
9 add ITE gate $(l = c\ ?\ t : e)$ to I
10 **return** I

fast-ite-gate-extraction (CNF F)
11 resulting ITE gates $I = 0$
12 **for all** variables v in F
13 $E^+ =$ *find-conditional-equivalences* (F, v)
14 $E^- =$ *find-conditional-equivalences* $(F, \bar{v})$
15 add *merge-conditional-equivalences* (v, E^+, E^-) to I
16 **return** I

The inner loop at Line 6 gives quadratic complexity in the number of literal occurrences, and with the check at Line 7 it looks even cubic. However, the actual goal of this algorithm is to find for a candidate condition c both a positive $(c \to l = t)$ and a matching negative conditional equality $(\bar{c} \to l = e)$, and thus to extract an ITE gate. This observation leads to the optimized algorithm in Alg. 6. It iterates over all variables, instead of clauses, and looks for positive and negative conditional equivalences E^+ and E^- for each of them. Equivalences of both sets with the same LHS are then merged to form ITE gates.

Further implementation details are as follows. Lines 2–3 of *find-conditional-equivalences* are implemented by extracting pairs of all the other literals in ternary clauses with $\bar{c}$, sorting the literals in the pair (smaller literal first), and then sorting all these conditional pairs lexicographically (positive literal smaller than negative). Those sorted pairs are split into "ranges" of positive and negative occurrences of the same variable as first literal in a pair.

Then we try for each pair of the smaller range to find the dual pair (with both literals negated) in the other range by binary search. Thus the complexity of *find-conditional-equivalences* is bounded by $\mathcal{O}(n \cdot \log n)$ where n is the number of ternary clauses with $\bar{c}$.

The nested loop in *merge-conditional-equivalences* can be implemented by first sorting the two conditional equivalence sets and following a merge-sort-style strategy, passing over both of them in increasing order of literals. It is still quadratic in the number of generated ITE gates, which is the worst-case complexity of the problem anyhow.

Finally, we can filter out (and do not watch) clauses which have literals that do not occur often enough: two literals (the condition and the LHS literal) have to occur twice positively and twice negatively, while the third literal must occur at least once in each polarity.

8 Congruence Closure

In SMT solvers [7] the congruence closure algorithm has found several applications, for example in ground theory solvers [57], or during quantifier instantiation [6]. It uses the congruence axiom to propagate and derive further equalities from a given set of equalities over first-order ground terms. For instance, given the equalities $x = y$, $u = f(x)$ and $v = f(y)$, the congruence axiom allows us to deduce $u = v$ too. This idea can be extended to functions and predicates of arbitrary arity. In contrast to structural hashing, it does not require any topological order of the variables, and thus can also be applied to cyclic functional definitions.

Extracted or rewritten gates need to be *normalized* to increase chances of matching other gates. For AND gates, the only form of normalization that can be achieved is to sort the RHS literals, assuming once again a fixed order on variables, e.g., induced by the variable order in the DIMACS file. The same idea can be applied to XOR gates, but besides sorting we can further force all the RHS literals of an XOR gate to be positive: if the number of negated RHS literals is even, their negations cancel, and we can simply drop them; if the number is odd, we also drop the negations and negate the LHS literal instead.

For an ITE gate ($l{=}c\ ?\ t : e$), a normalization strategy known from the BDD literature [29] applies. First, we ensure that the *condition literal* c is positive by using the equation $\bar{c}\ ?\ t : e \equiv c\ ?\ e : t$, if necessary. Then, we also make sure that the *then literal* t is positive, using $c\ ?\ \bar{t} : e \equiv \overline{c\ ?\ t : \bar{e}}$ and negating the LHS literal l if necessary.

After normalizing a gate, we check whether there is already an existing gate with the same operator (AND, ITE, XOR) and the same RHS literals. This check is implemented with a *hash table* using the operator and RHS literals as a key. If a gate is found with the same operator and RHS, we have derived an equivalence between the two LHS literals of the gates. This equivalence is recorded in a *union-find data-structure* [66], where every literal points to its (smaller) representative or itself.

Whenever a literal is assigned a new representative literal, we put that literal into a *queue*. Once all gates have been extracted, the propagation of these queued equivalences can be started in the main congruence closure loop (lines 13–18 in Alg. 7). In each iteration, a literal l of the queue is processed by iterating through all the gates that have l in their RHS. Each such gate is *rewritten* by replacing l (resp. $\bar{l}$) in them with its representative.

If a rewriting step results in a trivial gate, it is marked as garbage and skipped in later checks. For example, assume that literal b is dequeued in Line 13, and it is equivalent to its representative a. Then, the rewriting of the AND gate ($l = a \land b$) based on this equivalence results in the equivalence $l = a$. This we record and then mark the gate as garbage, without removing it from the RHS occurrence list of a.

Recording or *merging* an equivalence $l_1 = l_2$ consists of determining the representatives r_1 of l_1 and r_2 of l_2 (could be the literal itself). Assuming w.l.o.g. that $|r_1| < |r_2|$, we use r_1 as the new representative for both literals and push l_2 (the literal that is assigned a new representative) on the equivalence queue. As a last step, for proof logging, we augment the CNF with two binary clauses to capture that $l_2 \leftrightarrow r_1$ (this step is not shown in Alg. 7). Once the loop terminates, this augmented CNF is passed to a global equivalent literal substitution (ELS) procedure, which substitutes all equivalent literals in one pass over the formula.

Besides those (actually rather complex) ways of rewriting gates, another complication exists. It has to be taken into account when rewriting actually leads to a unit: for instance, if b in the discussed example with ($l = a \land b$) has $\bar{a}$ as representative instead of a, we can derive the unit clause $\bar{l}$. In this situation, we not only propagate this new assignment through the original CNF clauses, using the existing BCP mechanism of the SAT solver, but also need

Algorithm 7 An abstract version of our congruence closure algorithm. In the actual implementation we use a hash table to search gates in G by their RHS (in Lines 11 and 17) and interleave the loop in Lines 11–12 with gate extraction in Line 7. We further need to have fast access in Line 14 to all gates with the dequeued literal in their RHS, for which we use occurrence lists. We also do not show how derived unit clauses on this level of abstraction are handled which in our implementation are first propagated over the CNF and then used to simplify gates.

merge-literals (CNF F, queue Q, representatives ρ, literals l_1, l_2) // F, Q, ρ by reference

1 $r_1 = \rho(l_1)$, $r_2 = \rho(l_2)$

2 **if** $r_1 = \bar{r}_2$ **then** $F = \bot$ **and return** // inconsistent equivalence thus F unsatisfiable

3 select $r \in \{r_1, r_2\}$ with $|r| = \mathsf{min}(|r_1|, |r_2|)$ // pick representative with smaller variable

4 update $\rho(l_1) = \rho(l_2) = r$ and $\rho(\bar{l}_1) = \rho(\bar{l}_2) = \bar{r}$

5 **if** $r \neq r_1$ **then** enqueue l_1 to Q

6 **if** $r \neq r_2$ **then** enqueue l_2 to Q

clausal-congruence-closure (CNF F) // by reference, i.e., F updated in place

7 $G = $ *extract-gates* (F)

8 literals $L =$ all literals in F

9 representatives $\rho\colon L \to L$ initialized to $\rho(l) = l$

10 $Q =$ empty literal queue

11 **for all** $(l_1 = rhs_1), (l_2 = rhs_2) \in G$ with $rhs_1 = rhs_2$

12   *merge-literals* (F, Q, ρ, l_1, l_2)

13 **while** $F \neq \bot$ and Q not empty dequeue l from Q

14   **for** all gates $(k = rhs) \in G$ where l or $\bar{l}$ occurs in rhs

15   use ρ to rewrite $(k = rhs)$ to $(k' = rhs')$

16   remove gate $(k = rhs)$ from G

17   **if** G contains $(k'' = rhs'')$ with $rhs' = rhs''$ **then** *merge-literals* (F, Q, ρ, k', k'')

18   **else** add gate $(k' = rhs')$ to G

19 remove clauses C from F with $C \neq \rho(C) \wedge \rho(C) \in F$

20 replace F with $\rho(F)$

to simplify all gates in which l or $\bar{l}$ occurs. Thus our loop actually consists of propagating with higher priority all literals root-level assigned to a constant through gates in which they occur on the RHS, *simplifying* them accordingly, and then with lower priority propagating equivalent literals and rewriting their gates as discussed above.

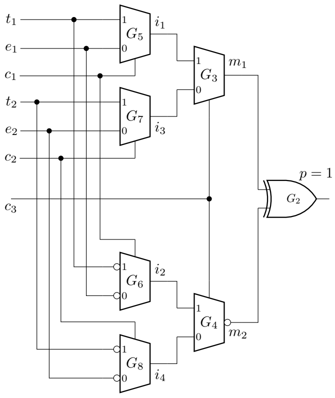
During this procedure (*cf.* Alg. 7), it might happen that an inconsistency is detected. For instance, if in the last example where $l = \bot$ is derived, the LHS l is already assigned to $\top$. Then the loop aborts and claims unsatisfiability of the formula immediately. This will in particular be the outcome when congruence closure is applied to isomorphic miters.

As already pointed out in Sect. 3, matching two isomorphic gates and substituting one LHS literal by its representative in all clauses where it occurs, necessarily results in duplicating the clauses of the representative gate. This occurs, for instance, in isomorphic miters where half of the variables vanish, but the number of clauses does not change.

Therefore, we originally tried to eagerly delete clauses used to extract a gate as soon as it became garbage or was removed. This risks turning unsatisfiable formulas satisfiable, as clauses can be used multiple times to extract gates. Instead, we implemented a dedicated global forward subsumption algorithm (hinted at in Line 19), which targets removing identical clauses modulo equivalent literals as recorded in the union-find data structure.

$$p \underset{1}{=} 1$$
$$p \underset{2}{=} m_1 \oplus \overline{m_2}$$
$$m_1 \underset{3}{=} c_3 \mathbin{?} i_1 : i_3$$
$$m_2 \underset{4}{=} c_3 \mathbin{?} i_2 : i_4$$
$$i_1 \underset{5}{=} c_1 \mathbin{?} t_1 : e_1$$
$$i_2 \underset{6}{=} c_2 \mathbin{?} t_2 : e_2$$
$$i_3 \underset{7}{=} c_1 \mathbin{?} \overline{t_1} : \overline{e_1}$$
$$i_4 \underset{8}{=} c_2 \mathbin{?} \overline{t_2} : \overline{e_2}$$

(a) gates $G_1, \ldots, G_8$



(b) miter circuit

$$(p)_1$$
$$(\overline{p}\, m_1 \overline{m}_2)_2\ (\overline{p}\, \overline{m}_1 m_2)_3\ (p\, \overline{m}_1 \overline{m}_2)_4\ (p\, m_1 m_2)_5$$
$$(\overline{c}_3\, \overline{m}_1\, i_1)_6\ (\overline{c}_3\, m_1\, \overline{i}_1)_7\ (c_3\, \overline{m}_1\, i_3)_8\ (c_3\, m_1\, \overline{i}_3)_9$$
$$(\overline{c}_3\, \overline{m}_2\, i_2)_{10}\ (\overline{c}_3\, m_2\, \overline{i}_2)_{11}\ (c_3\, \overline{m}_2\, i_4)_{12}\ (c_3\, m_2\, \overline{i}_4)_{13}$$
$$(\overline{c}_1\, \overline{i}_1\, t_1)_{14}\ (\overline{c}_1\, i_1\, \overline{t}_1)_{15}\ (c_1\, \overline{i}_1\, e_1)_{16}\ (c_1\, i_1\, \overline{e}_1)_{17}$$
$$(\overline{c}_1\, \overline{i}_2\, \overline{t}_1)_{18}\ (\overline{c}_1\, i_2\, t_1)_{19}\ (c_1\, \overline{i}_2\, \overline{e}_1)_{20}\ (c_1\, i_2\, e_1)_{21}$$
$$(\overline{c}_2\, \overline{i}_3\, t_2)_{22}\ (\overline{c}_2\, i_3\, \overline{t}_2)_{23}\ (c_2\, \overline{i}_3\, e_2)_{24}\ (c_2\, i_3\, \overline{e}_2)_{25}$$
$$(\overline{c}_2\, \overline{i}_4\, \overline{t}_2)_{26}\ (\overline{c}_2\, i_4\, t_2)_{27}\ (c_2\, \overline{i}_4\, \overline{e}_2)_{28}\ (c_2\, i_4\, e_2)_{29}$$

(c) CNF with clauses $C_1, \ldots, C_{29}$

Figure 8 An example of an optimized miter. It is comprised of an unsimplified circuit on the bottom part and its optimized variant above it. The optimized circuit simply omits the unnecessary inverters. This example also illustrates why `test02` from [72] is considerably more challenging without ITE-gate extraction. For example, to recognize easily that the output of G_5 (resp. G_7) is the negation of the output of gate G_6 (resp. G_8), the CNF encoding must maintain parts of the structure of the circuits. Extracting and normalizing ITE gates allows the congruence closure approach to realize the equivalence between the two circuits efficiently (*cf.* Fig. 9).

9 Proofs

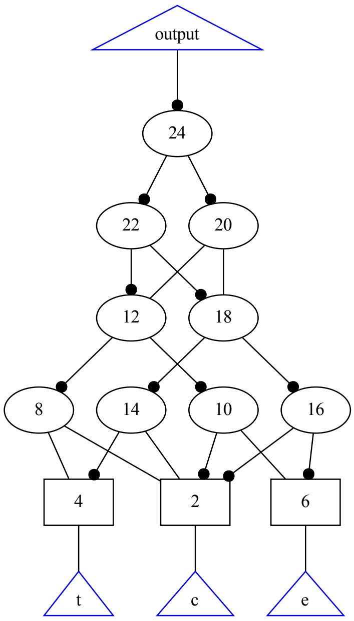
The algorithms for extraction and congruence closure as well as for rewriting and simplifying gates are rather involved. Therefore, we rely on generating and checking clausal proofs for correctness (i) with our internal proof checker during development and testing, as well as (ii) by producing DRUP proofs and external checking in production [48, 65, 69].

In principle, we just have to derive the two binary clauses for each detected equivalence. While equivalences from matched AND gates are easy to handle as they can be simulated by HBR and thus yield RUP steps as discussed in Sect. 3, equivalences from matched XOR and ITE gates require more intermediate DRUP steps as proposed in [59] for XORs or how ternary resolution is used in [45] for ITE gates and binary XORs.

Note that eager ELS during congruence closure does not need to be modelled in DRUP proofs, i.e., substituting an equivalent literal by its representative (either in clauses or in the RHS of a gate) as this is captured by propagation semantics in RUP. Proofs with hints/antecedents, such as LRAT proofs [35], would require much more effort. The internal proof checker receives the same information as the DRUP proof, but in addition we check that the clauses of the Tseitin encoding of all extracted or rewritten gates are RUP.

10 Benchmarks

Our first HWMCC'12 benchmark set contains CNF encoded miters where HBR has difficulties and which had already been submitted to the SAT Competition 2013 [24]. These are miters for 341 AIGER [8] models used in the Hardware Model Checking Competition 2012. The original models are sequential and to obtain combinational miters, we simply treat latches as inputs and their next-state functions as outputs. We further used ABC [31] as synthesis tool to optimize the models (using the '`&dc2`' command). These are passed through AIGMITER (from the AIGER [8] tools) to construct optimized miters, tagged `opt`. Isomorphic miters,



(a) A miter of two ITE gates in AIGER [8] format.

$$
\begin{aligned}
&(\overline{x}_4\, x_1), (\overline{x}_4\, x_2), (x_4\, \overline{x}_1\, \overline{x}_2),\\
&(\overline{x}_5\, \overline{x}_1), (\overline{x}_5\, x_3), (x_5\, x_1\, \overline{x}_3),\\
&(\overline{x}_6\, \overline{x}_4), (\overline{x}_6\, \overline{x}_5), (x_6\, x_4\, x_5),\\
&(\overline{x}_7\, x_1), (\overline{x}_7\, \overline{x}_2), (x_7\, \overline{x}_1\, x_2),\\
&(\overline{x}_8\, \overline{x}_1), (\overline{x}_8\, \overline{x}_3), (x_8\, x_1\, x_3),\\
&(\overline{x}_9\, \overline{x}_7), (\overline{x}_9\, \overline{x}_8), (x_9\, x_7\, x_8),\\
&(\overline{x}_{10}\, x_6), (\overline{x}_{10}\, x_9), (x_{10}\, \overline{x}_6\, \overline{x}_9),\\
&(\overline{x}_{11}\, \overline{x}_6), (\overline{x}_{11}\, \overline{x}_9), (x_{11}\, x_6\, x_9),\\
&(x_{12}\, x_{10}\, x_{11}), (\overline{x}_{12}).
\end{aligned}
$$

(b) The ands CNF encoding of the AIG.

$$
\begin{aligned}
&(\overline{x}_4\, \overline{x}_1\, x_3), (\overline{x}_4\, x_1\, x_2),\\
&(x_4\, \overline{x}_1\, \overline{x}_3), (x_4\, x_1\, \overline{x}_2),\\
&(\overline{x}_5\, \overline{x}_1\, \overline{x}_3), (\overline{x}_5\, x_1\, \overline{x}_2),\\
&(x_5\, \overline{x}_1\, x_3), (x_5\, x_1\, x_2),\\
&(x_6\, \overline{x}_5\, x_4), (x_6\, x_5\, \overline{x}_4), (\overline{x}_6).
\end{aligned}
$$

(c) The xits CNF encoding of the AIG.

Figure 9 An illustration of the difference between xits and ands CNF encodings of a given AIG. The miter applies an XOR (described by the three AND nodes A_{20}, A_{22}, and A_{24}) to compare $c\ ?\ t : e$ (AND nodes A_8, A_{10}, and A_{12}) to $\overline{c\ ?\ \bar{t} : \bar{e}}$ (AND nodes A_{14}, A_{16}, and A_{18}). The ands encoding (Fig. 9b) translates all 9 AND nodes of the AIG independently of each other, resulting in 26 clauses over 12 Boolean variables. The xits encoding (Fig. 9c), on the other hand, recognizes the ITE and XOR gates in the AIG and encodes the corresponding nodes *together* into a CNF with 11 clauses over 6 variables. While the ands encoding destroys the original ITE and XOR structures of the formula, the xits encoding maintains them. That allows our approach to recognize, extract and normalize the ITE gates efficiently and thereby the congruence closure algorithm can quickly conclude that the two ITE expressions are equivalent. This explains the efficiency of our algorithm on the `test02` miter from the IWLS'22 benchmark set (*cf.* Sect. 7 and Fig. 12).

tagged iso, are generated in the same way, except that optimization through Abc is skipped.

These miter circuits are then translated to CNF with a new version of AigToCnf (available in the AIGER GitHub repository and in the source code artifact [15]) which has been extended to detect XOR and ITE gates in AIGER circuits. During Tseitin encoding, we check whether an AND gate has two negated AND gates as children and actually implements an XOR or ITE gate. In this case we use a direct CNF encoding of four clauses, as for the XOR and ITE gates in Fig. 1, instead of 9 clauses for three AND gates, skipping the two child AND gates of the top AND gate. This reduces not only the number of clauses and variables but also has positive effects on running time as our experiments will show (except for the simple identical miters where there is little difference).

Therefore we have extended the original HWMCC'12 benchmark set by using this new version of AigToCnf with XOR and ITE matching too, which results in four variants of the 341 AIGER models: ands-iso, ands-opt, xits-iso, and xits-opt. We give in Fig. 8+9 an example of an optimized miter. These benchmarks are available at [17].

Our second IWLS'22 set comes from the IWLS'22 paper [72] by He-Teng Zhang, Jie-Hong R. Jiang, Alan Mishchenko, and Luca Amarù. It is an update on their DAC'21 paper [70], focusing on a hybrid approach to SAT-sweeping, i.e., using a SAT solver incrementally, taking circuit structure into account. Experiments in [72] used a subset of the benchmarks from [70]. These includes the five miters `n01`, `n04`, `n06`, and `test01` and `test02` in AIGER [8] format, provided by Alan Mishchenko. These benchmarks were considered hard for SAT sweeping, particularly for monolithic CNF-level SAT solving. Thus we consider this set of benchmarks as a litmus test for our usecase. As for HWMCC'12, our IWLS'22 CNF benchmarks come in two flavors: xits with special treatment of XOR and ITE gates during Tseitin encoding and ands without. These benchmarks are available at [16].

It turned out, confirmed by Alan Mishchenko, that the outputs of `test01` and `test02` were flipped in the generation process. This does not invalidate the SAT sweeping experiments in [70, 72] at all. However, it needs to be taken care of when encoding them into CNF with AigToCnf, by simply first negating the outputs with AigFlip. Furthermore, the other three AIGs, `n01`, `n04`, and `n06`, are not negated but have multiple outputs. Thus, we joined them by disjunction with AigOr. These tools are part of the AIGER library `https://github.com/arminbiere/aiger` and included in the source code artifact [15].

Continuing the discussion of Section 9, we not only empirically checked via fuzzing [32] that our implementation of congruence closure is sound but also that it is complete, i.e., it really solves isomorphic miters with AND, XOR, and ITE gates. To that end, we generated combinational AIGER models with our AigFuzz fuzzer, used AigMiter to produce an isomorphic miter, and then encoded it to CNF with our new version of AigToCnf, which detects XOR and ITE gates. The resulting CNF is given to Kissat using options that make sure that only congruence closure is run (to completion as always) without using any other preprocessing and not even entering the CDCL loop. Thus the CNF remains unsolved unless congruence closure alone can solve it.

11 Experiments

We follow the set-up of the main track of the SAT Competition, where each solver configuration is run on one benchmark instance in single-threaded mode. As compute platform we used the bwForCluster Helix with AMD Milan EPYC 7513 CPUs and for all experiments enforced a memory limit of 15 GB and a time limit of 5000 seconds with Runlim.

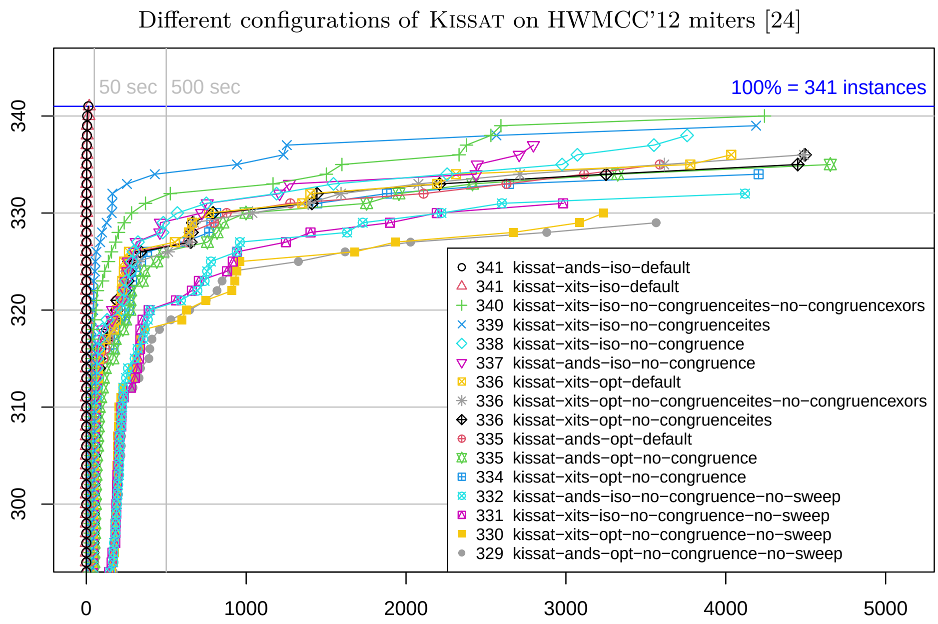


Figure 10 Comparison of variants of Kissat with more and more relevant features disabled. The default configuration employs all of the methods described here. First, only the extraction of ITE gates is disabled (no-congruenceites), then also the extraction of XOR gates (no-congruenceites-no-congruencexors), then congruence closure is completely disabled (no-congruence), and finally even internal SAT sweeping [21, 22] is disabled (no-congruence-no-sweep). Note that for the ands encoding, no ITE nor XOR gate can be extracted anyhow and therefore disabling their extraction gives the same result as enabling them. Thus the plot shows only 6 ands variants but 10 xits variants. On this and all the following plots, the results are shown in the same way as in the annual SAT competition, e.g., a point with coordinates (1407, 333) means that 333 problems were solved in 1407 seconds.

We compare our implementation of congruence closure, enabled by default in our new version of Kissat with the latest version 1.0.0 of Lingeling implementing simple probing, blocked clause decomposition (using the tools `sblitter`, followed by `mequick`, and finally using the same SAT solver Lingeling 1.0.0) [44], the winner Sbva-CaDiCaL [43] of the SAT Competition 2023, the latest version 1.9.5 of CaDiCaL [23] and MiniSat 2.2.0 [38]. We further compare against Abc [31, 72] on miter circuits. It represents the state-of-the-art [72] in hybrid SAT sweeping, but "per se" is not a solver, even though it uses SAT solvers.

Our results are presented as a cumulative distribution function (CDF), as in the SAT Competition since 2021, giving the number of solved problems (y-axis) within the amount of time (x-axis), i.e., the higher and the more to the left, the better. We include a horizontal line for all instances (100%). The x-axis shows time up-to the time-limit of 5000 seconds.

While adding congruence closure to Kissat we introduced a dedicated preprocessing round, during which, after unit propagation, the first complete round of congruence closure is applied. Later, during solving, whenever probing based inprocessing is scheduled—which includes vivification, equivalent literal substitution, and other procedures—we always schedule again congruence closure elimination, but only on irredundant and binary clauses. It is also run until completion. This allows us to find additional congruent literals, as gate structure emerges after learning units, shrinking clauses, vivification, and variable elimination.

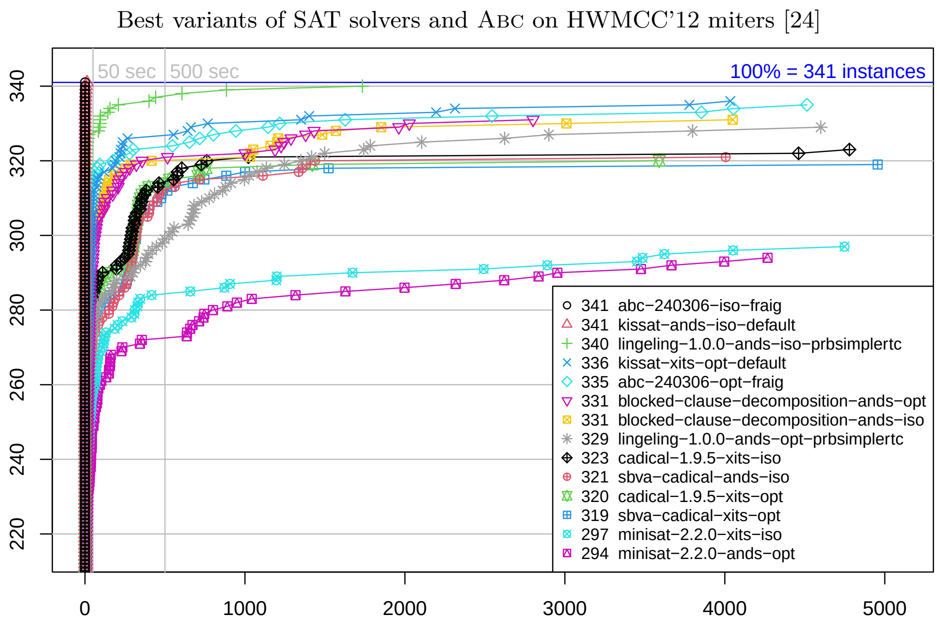


Figure 11 Number of solved HWMCC'12 miter instances at each point of time. To improve clarity and save space, we show only the best encoding variant (ands or xits) for each SAT solver. For instance our experiments revealed that for KISSAT on optimized miters (opts) the xits encoding, i.e., kissat-xits-opt-default was superior to kissat-and-opt-default with the ands encoding while for MINISAT it was the opposite (and therefore we only show minisat-2.2.0-ands-opt and not minisat-2.2.0-xits-opt).

Our primary results on HWMCC'12 miters in Fig. 10 show that isomorphic miters (iso) can be solved by our new congruence closure approach kissat-⋆-default instantly, both if we encode only AND gates directly (ands) or match them to XOR and ITE gates and then use a more elaborate Tseitin encoding (xits). Internal SAT sweeping [21, 22] implemented in KISSAT is in principle also able to find equivalences of gates. It is, however, scheduled after our faster congruence closure algorithm. For xits encoding, congruence closure takes 0.79 s on average (0 s–44.21 s) and the average percentage of total solving time is 5.4% (0.02%–26.03%). Isomorphic miters are solved by our new algorithm instantly (the vertical lines in Fig. 10).

Comparison with other solvers is shown in Fig. 11. Optimized miters (opt) are in general harder to solve, but clausal congruence closure, as enabled by default in kissat-xits-opt-default, even surpasses abc-240306-opt-frag, which represents the state-of-the-art in hybrid SAT sweeping [72], as implemented in ABC (command '`&fraig -y`'). Running simple probing in LINGELING until completion (lingeling-1.0.0-ands-iso-prbsimplertc) is the only CNF-level approach that can compete on isomorphic miters (iso), but is not competitive on optimized (opt) ones (Fig. 12, Tab. 13). Note that simple probing can not handle XOR nor ITE gates.

Fig. 12 shows the results on IWLSS'22 benchmarks. Our implementation in KISSAT (in contrast to ABC) can provide DRAT proofs [69] as standard in the SAT Competition. Actually only DRUP proofs are relevant for congruence closure, as described in Sect. 9. The results demonstrate that the overhead for proof production (proof) for KISSAT is low and proof checking (check) has comparable run-time to solving. On these 5 benchmarks our new algorithm gives substantial improvements in solving time, i.e., kissat-xits-default vs. kissat-xits-no-congruence in Tab. 13. On four of these benchmarks ABC still wins (running on the AIGER circuit model while KISSAT only gets CNF) except for `test02` where KISSAT is faster for the xits encoding. See Fig. 8+9 for an explanation.

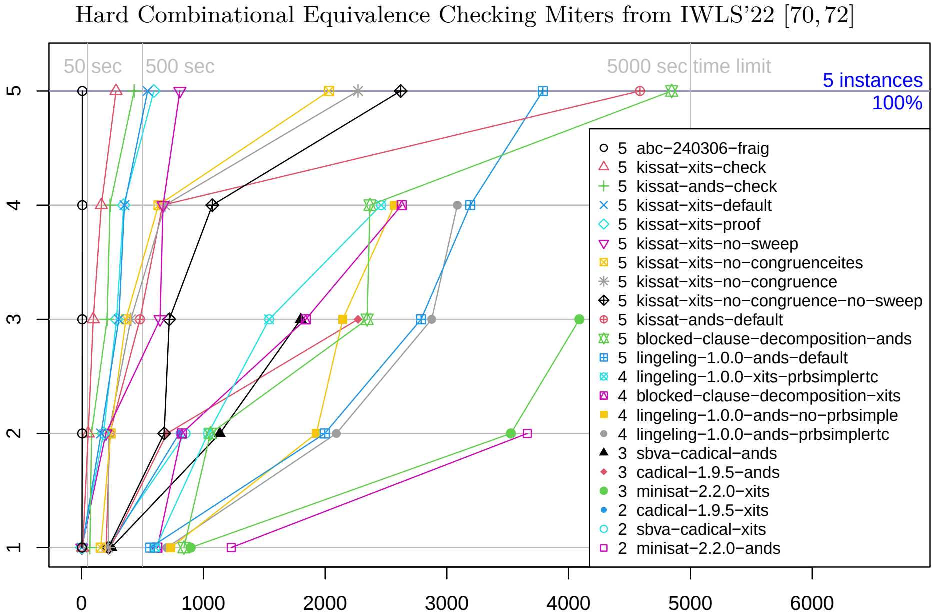


Figure 12 These miters from [70] were target of optimizations reported in [72]. They are indeed hard for monolithic SAT solving starting after Tseitin encoding. Results on benchmark `test02` are particularly interesting as kissat-xits-default took 1.79 s. to solve it, Abc 5.75 s, while disabling extraction of ITE gates in kissat-xits-no-congruenceites already needs 2032.41 s and plain AND-only Tseitin encoding in kissat-ands-default even 4585.76 s (*cf.* Tab. 13 for more detailed results).

Table 13 The actual run-time on the IWLS'22 miters from [70, 72] (*cf.* CDF in Fig. 12).

	n01	n04	n06	test01	test02
abc-240306-fraig	5.96	5.38	4.86	2.89	5.75
kissat-xits-check	95.45	162.38	282.61	54.28	9.06
kissat-ands-check	81.57	209.54	233.75	67.95	431.21
kissat-xits-default	305.60	160.01	542.18	352.15	1.79
kissat-xits-proof	287.21	179.72	593.54	345.60	2.38
kissat-xits-no-sweep	199.17	807.07	644.22	669.11	1.79
kissat-xits-no-congruenceites	238.32	157.06	631.46	363.93	2032.41
kissat-xits-no-congruence	222.25	218.73	684.94	404.48	2270.00
kissat-xits-no-congruence-no-sweep	221.25	678.17	720.29	1073.75	2620.65
kissat-ands-default	231.87	201.45	664.81	479.28	4585.76
blocked-clause-decomposition-ands	840.19	1058.28	2345.20	2368.54	4846.14
lingeling-1.0.0-ands-default	563.03	3192.09	1997.28	2788.51	3788.10
lingeling-1.0.0-xits-prbsimplertc	607.82	1039.04	1540.55	2459.75	—
blocked-clause-decomposition-xits	622.46	822.68	1841.48	2628.96	—
lingeling-1.0.0-ands-no-prbsimple	733.61	1928.03	2144.69	2568.83	—
lingeling-1.0.0-ands-prbsimplertc	700.58	3085.86	2092.79	2875.45	—
sbva-cadical-ands	244.94	1800.21	1135.28	—	—
cadical-1.9.5-ands	236.14	2270.17	701.13	—	—
minisat-2.2.0-xits	895.77	4088.40	3525.61	—	—
cadical-1.9.5-xits	227.21	—	801.69	—	—
sbva-cadical-xits	205.70	—	853.77	—	—
minisat-2.2.0-ands	1229.07	—	3660.71	—	—

KISSAT and SBVA-CADICAL on all 400 SAT Competition 2022 main track benchmarks

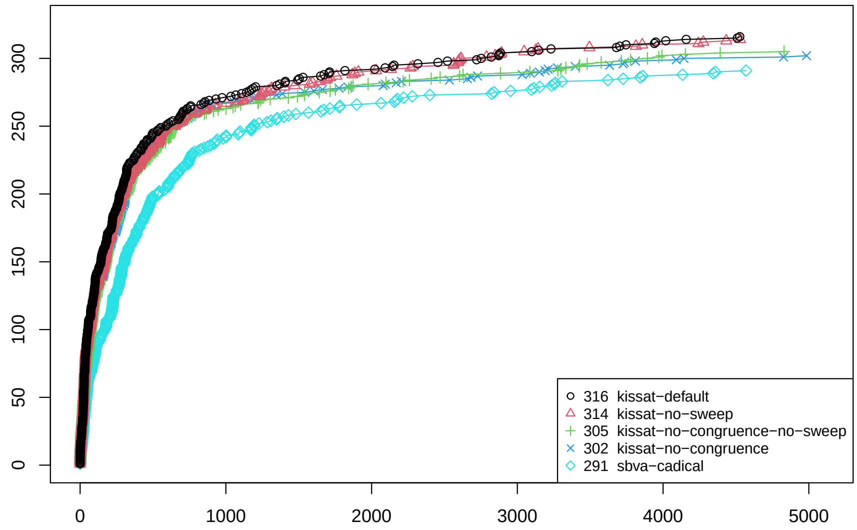


Figure 14 On the problems of the main track of the SAT Competition 2022 [18] the congruence closure algorithm is successful. In fact, all versions of KISSAT are faster than SBVA-CADICAL. Two benchmarks `6133-sc2014` and `6s184`, reused from our HWMCC'12 isomorphic miter benchmarks submitted to the SAT Competition 2013 [24], were solved immediately by congruence closure (in 0.07 s and 0.04 s), but were also solved without congruence closure (in 37.51 s and 25.99 s). The default configuration of KISSAT eliminated a total of 108 272 236 equivalent literals found by congruence closure among all the 400 benchmarks of the main track.

KISSAT and SBVA-CADICAL on all 400 SAT Competition 2023 main track benchmarks

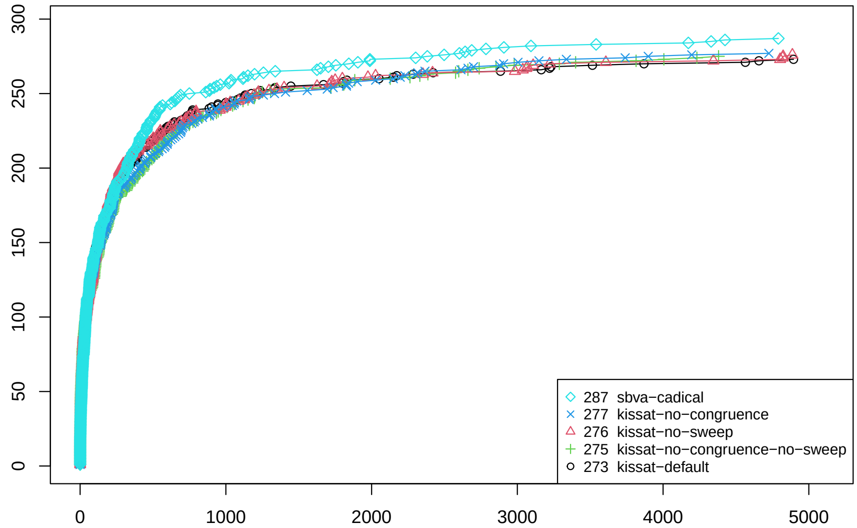


Figure 15 In the main track of the SAT Competition 2023 [19] with many hard combinatorial problems, Structured Bounded Variable Addition (SBVA) [43] in SBVA-CADICAL, the winner of this track, has an advantage over KISSAT, because SBVA and congruence closure are orthogonal. The different variants of congruence closure are very similar here, although the default version spent on average 4.41% of the running time in congruence closure.

To assess the effectiveness of congruence closure on a more general set of problems, we also evaluated our new version of Kissat on problem instances [18, 19] from the SAT Competition 2022 and 2023. The results show that our implementation is fast enough to run to completion even when there are few or no gates to extract. On the 2022 problems our method solves 14 more instances (Fig. 14). The effect of congruence closure on the 2023 problem set is small (*cf.* Fig. 15), probably due to a large fraction of combinatorial benchmarks. Following the SAT practitioner manifesto [27] we also compare against the 2023 winner Sbva-CaDiCaL.

Finally, we want to investigate the average learned clause length, related to the observation in the introduction on CDCL not being able to produce short proofs. Therefore, we have rerun without congruence closure (no-congruence) but with more statistics all the isomorphic HWMCC'12 miters again (see the metrics directories in the experimental data artifact [14]) and computed the average learned clause lengths over all miters, which is 43.6 literals per learned clause for ands-iso-no-congruence, and 46.7 for xits-iso-no-congruence. Our default version of Kissat with congruence closure solves these miters instantly through preprocessing, without the need to learn any clause, and thus we computed instead the average added clause length in the Rup proofs which is 1.88 literals for ands-iso and 2.12 for xits-iso.

Source code is available on Zenodo [15]. The HWMCC'12 benchmarks [17] and ILWS'22 benchmarks [16] are available on Zenodo too, as well as all experimental data [14].

12 Conclusion

We explored the idea of applying congruence closure to gates extracted from CNF using an inverse of the Tseitin encoding. Our new optimized extraction algorithms for AND, XOR, and ITE gates are able to run until completion within seconds on large combinational equivalence checking miters and benchmarks from the SAT competition. These gates are then used in a congruence closure algorithm to match equivalent gates and deduce equivalent literals, which can also run to completion on standard benchmarks from the SAT competition and is now enabled by default in our new version of the SAT solver Kissat.

Our experiments show that this is the first approach in the literature to instantly solve large isomorphic CNF encoded miters. Further, it gives substantial improvements on industrially relevant optimized miters, where our CNF level approach reaches the performance or even is better than a dedicated circuit level SAT sweeping technique.

Acknowledgements

Finally, we want to thank the anonymous reviewers for valuable comments and useful suggestions, which definitely helped us to improve the paper considerably, particularly in the exposition of the experimental part.

References

1 Luca G. Amarù, Felipe S. Marranghello, Eleonora Testa, Christopher Casares, Vinicius N. Possani, Jiong Luo, Patrick Vuillod, Alan Mishchenko, and Giovanni De Micheli. Sat-sweeping enhanced for logic synthesis. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020. doi:10.1109/DAC18072.2020.9218691.

2 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

3 Fahiem Bacchus. Enhancing Davis Putnam with extended binary clause reasoning. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pages 613–619. AAAI Press / The MIT Press, 2002.

4 Fahiem Bacchus and Jonathan Winter. Effective preprocessing with hyper-resolution and equality reduction. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2003. doi:10.1007/978-3-540-24605-3_26.

5 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 *Lecture Notes in Computer Science*, pages 317–332. Springer, 2014. doi:10.1007/978-3-319-09284-3_24.

6 Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Congruence closure with free variables. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II*, volume 10206 of *Lecture Notes in Computer Science*, pages 214–230, 2017. doi:10.1007/978-3-662-54580-5_13.

7 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021. doi:10.3233/FAIA201017.

8 Armin Biere. The AIGER And-Inverter Graph (AIG) format version 20071012. Technical Report 07/1, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2007.

9 Armin Biere. P{re,i}coSAT@SC'09. In *SAT 2009 Competitive Event Booklet*, 2009. URL: http://www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.

10 Armin Biere. Lingeling and friends entering the SAT Challenge 2012. In Adrian Balint, Anton Belov, Daniel Diepold, Simon Gerber, Matti Järvisalo, and Carsten Sinz, editors, *Proc. of SAT Challenge 2012: Solver and Benchmark Descriptions*, volume B-2012-2 of *Department of Computer Science Series of Publications B*, pages 33–34. Univ.Helsinki, 2012.

11 Armin Biere. Yet another local search solver and Lingeling and friends entering the SAT Competition 2014. In Adrian Balint, Andon Belov, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2014 – Solver and Benchmark Descriptions*, volume B-2014-2 of *Department of Computer Science Series of Publications B*, pages 39–40. Univ.Helsinki, 2014.

12 Armin Biere. Collection of combinational arithmetic miters submitted to the SAT Competition 2016. In Tomáš Balyo, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, volume B-2016-1 of *Department of Computer Science Series of Publications B*, pages 65–66. Univ.Helsinki, 2016.

13 Armin Biere. CaDiCaL at the SAT Race 2019. In Marijn Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, pages 8–9. Univ.Helsinki, 2019.

14 Armin Biere. Clausal congruence closure paper logs, plots and tables, June 2024. doi:10.5281/zenodo.11658133.

15 Armin Biere. Clausal congruence closure paper source code, June 2024. doi:10.5281/zenodo.11652423.

16 Armin Biere. CNF encoded hard miters from IWLS'22 paper, March 2024. doi:10.5281/zenodo.10823099.

17 Armin Biere. CNF encoded isomorphic and optimized miters from Hardware Model Checking Competition 2012 models, March 2024. doi:10.5281/zenodo.10823128.

18 Armin Biere. SAT Competition 2022 main track benchmarks, June 2024. doi:10.5281/zenodo.11428010.

19 Armin Biere. SAT Competition 2023 main track benchmarks, June 2024. doi:10.5281/zenodo.11426992.

20 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. Univ.Helsinki, 2020.

21 Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pages 10–11. Univ.Helsinki, 2022.

22 Armin Biere, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba entering the SAT Competition 2021. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions*, volume B-2021-1 of *Department of Computer Science Report Series B*, pages 10–13. Univ.Helsinki, 2021.

23 Armin Biere, Mathias Fleury, and Florian Pollitt. CaDiCaL_vivinst, IsaSAT, Gimsatul, Kissat, and TabularaSAT entering the SAT competition 2023. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2023 – Solver and Benchmark Descriptions*, volume B-2023-1 of *Department of Computer Science Report Series B*, pages 14–15. Univ.Helsinki, 2023.

24 Armin Biere, Marijn Heule, Matti Järvisalo, and Norbert Manthey. Equivalence checking of HWMCC 2012 circuits. In Adrian Balint, Andon Belov, Marijn Heule, and Matti Järvisalo, editors, *Proc. of SAT Competition 2013 – Solver and Benchmark Descriptions*, volume B-2013-1 of *Department of Computer Science Series of Publications B*, page 104. Univ.Helsinki, 2013.

25 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. doi:10.3233/FAIA336.

26 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 391–435. IOS Press, 2021. doi:10.3233/FAIA200992.

27 Armin Biere, Matti Järvisalo, Daniel Le Berre, Kuldeep S. Meel, and Stefan Mengel. The SAT Practitioner's Manifesto, September 2020. doi:10.5281/zenodo.4500928.

28 Alain Billionnet and Alain Sutter. An efficient algorithm for the 3-satisfiability problem. *Oper. Res. Lett.*, 12(1):29–36, 1992. doi:10.1016/0167-6377(92)90019-Y.

29 Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a BDD package. In Richard C. Smith, editor, *Proceedings of the 27th ACM/IEEE Design Automation Conference. Orlando, Florida, USA, June 24-28, 1990*, pages 40–45. IEEE Computer Society Press, 1990. doi:10.1145/123186.123222.

30 Daniel Brand. Verification of large synthesized designs. In Michael R. Lightner and Jochen A. G. Jess, editors, *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993*, pages 534–537. IEEE Computer Society / ACM, 1993. doi:10.1109/ICCAD.1993.580110.

31 Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. Springer, 2010. doi:10.1007/978-3-642-14295-6_5.

32 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In Ofer Strichman and Stefan Szeider, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2010. doi:10.1007/978-3-642-14186-7_6.

33 Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

34 Maciej J. Ciesielski, Cunxi Yu, Walter Brown, Duo Liu, and André Rossi. Verification of gate-level arithmetic circuits by function extraction. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 52:1–52:6. ACM, 2015. doi:10.1145/2744769.2744925.

35 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017. doi:10.1007/978-3-319-63046-5_14.

36 Alvaro del Val. Simplifying binary propositional theories into connected components twice as fast. In Robert Nieuwenhuis and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings*, volume 2250 of *Lecture Notes in Computer Science*, pages 392–406. Springer, 2001. doi:10.1007/3-540-45653-8_27.

37 Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005.

38 Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003. doi:10.1007/978-3-540-24605-3_37.

39 Mathias Fleury and Armin Biere. Mining definitions in Kissat with Kittens. *Formal Methods Syst. Des.*, 60(3):381–404, 2022. doi:10.1007/S10703-023-00421-2.

40 Zhaohui Fu and Sharad Malik. Extracting logic circuit structure from conjunctive normal form descriptions. In *20th International Conference on VLSI Design (VLSI Design 2007), Sixth International Conference on Embedded Systems (ICES 2007), 6-10 January 2007, Bangalore, India*, pages 37–42. IEEE Computer Society, 2007. doi:10.1109/VLSID.2007.81.

41 Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2008, Fort Lauderdale, Florida, USA, January 2-4, 2008*, 2008. URL: http://isaim2008.unl.edu/PAPERS/TechnicalProgram/ISAIM2008_0008_60a1f9b2fd607a61ec9e0feac3f438f8.pdf.

42 Roman Gershman and Ofer Strichman. Cost-effective hyper-resolution for preprocessing CNF formulas. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 423–429. Springer, 2005. doi:10.1007/11499107_34.

43 Andrew Haberlandt, Harrison Green, and Marijn J. H. Heule. Effective auxiliary variables via structured reencoding. In Meena Mahajan and Friedrich Slivovsky, editors, *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023,*

Alghero, Italy, volume 271 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.SAT.2023.11.

44 Marijn Heule and Armin Biere. Blocked clause decomposition. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 423–438. Springer, 2013. doi:10.1007/978-3-642-45221-5_29.

45 Marijn Heule, Matti Järvisalo, and Armin Biere. Revisiting hyper binary resolution. In Carla P. Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*, volume 7874 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 2013. doi:10.1007/978-3-642-38171-3_6.

46 Marijn J. H. Heule. Proofs of unsatisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 635–668. IOS Press, 2021. doi:10.3233/FAIA200998.

47 Marijn J. H. Heule and Armin Biere. Proofs for satisfiability problems. In *All about Proofs, Proofs for All (APPA)*, volume 55 of *Math. Logic and Foundations*. College Pub., 2015.

48 Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Strong extension-free proof systems. *J. Autom. Reason.*, 64(3):533–554, 2020. doi:10.1007/S10817-019-09516-0.

49 Markus Iser. *Recognition and Exploitation of Gate Structure in SAT Solving*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020. URL: https://nbn-resolving.org/urn:nbn:de:101:1-2020042904595660732648.

50 Markus Iser, Felix Kutzner, and Carsten Sinz. Using gate recognition and random simulation for under-approximation and optimized branching in SAT solvers. In *29th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2017, Boston, MA, USA, November 6-8, 2017*, pages 1029–1036. IEEE Computer Society, 2017. doi:10.1109/ICTAI.2017.00158.

51 Daniela Kaufmann and Armin Biere. Improving AMulet2 for verifying multiplier circuits using SAT solving and computer algebra. *Int. J. Softw. Tools Technol. Transf.*, 25(2):133–144, 2023. doi:10.1007/s10009-022-00688-6.

52 Daniela Kaufmann, Manuel Kauers, Armin Biere, and David Cok. Arithmetic verification problems submitted to the SAT Race 2019. In Marijn Heule, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Race 2019 – Solver and Benchmark Descriptions*, volume B-2019-1 of *Department of Computer Science Series of Publications B*, page 49. Univ.Helsinki, 2019.

53 Andreas Kuehlmann and Florian Krohm. Equivalence checking using cuts and heaps. In Ellen J. Yoffa, Giovanni De Micheli, and Jan M. Rabaey, editors, *Proceedings of the 34st Conference on Design Automation, Anaheim, California, USA, Anaheim Convention Center, June 9-13, 1997*, pages 263–268. ACM Press, 1997. doi:10.1145/266021.266090.

54 Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 21(12):1377–1394, 2002. doi:10.1109/TCAD.2002.804386.

55 Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Definability for model counting. *Artif. Intell.*, 281:103229, 2020. doi:10.1016/j.artint.2019.103229.

56 Chu Min Li. Integrating equivalency reasoning into Davis-Putnam procedure. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA.*, pages 291–296. AAAI Press / The MIT Press, 2000.

57 Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Inf. Comput.*, 205(4):557–580, 2007. doi:10.1016/J.IC.2006.08.009.

58 Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Sais. Recovering and exploiting structural knowledge from CNF formulas. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2002. `doi:10.1007/3-540-46135-3_13`.

59 Tobias Philipp and Adrian Rebola-Pardo. DRAT proofs for XOR reasoning. In Loizos Michael and Antonis C. Kakas, editors, *Logics in Artificial Intelligence - 15th European Conference, JELIA 2016, Larnaca, Cyprus, November 9-11, 2016, Proceedings*, volume 10021 of *Lecture Notes in Computer Science*, pages 415–429, 2016. `doi:10.1007/978-3-319-48758-8_27`.

60 Jarrod A. Roy, Igor L. Markov, and Valeria Bertacco. Restoring circuit structure from SAT instances. In *Proceedings of International Workshop on Logic and Synthesis (IWLS)*, pages 663–678, 2004.

61 Amr A. R. Sayed-Ahmed, Daniel Große, Ulrich Kühne, Mathias Soeken, and Rolf Drechsler. Formal verification of integer multipliers by combining gröbner basis with logic reduction. In Luca Fanucci and Jürgen Teich, editors, *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, pages 1048–1053. IEEE, 2016. URL: `https://ieeexplore.ieee.org/document/7459464/`.

62 Friedrich Slivovsky. Interpolation-based semantic gate extraction and its applications to QBF preprocessing. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pages 508–528. Springer, 2020. `doi:10.1007/978-3-030-53288-8_24`.

63 Gordon L. Smith, Ralph J. Bahnsen, and Harry Halliwell. Boolean comparison of hardware and flowcharts. *IBM J. Res. Dev.*, 26(1):106–116, 1982. `doi:10.1147/RD.261.0106`.

64 Mate Soos and Kuldeep S. Meel. BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1592–1599. AAAI Press, 2019. `doi:10.1609/AAAI.V33I01.33011592`.

65 Yong Kiam Tan, Marijn J. H. Heule, and Magnus O. Myreen. cake_lpr: Verified propagation redundancy checking in cakeml. 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 II*, volume 12652 of *Lecture Notes in Computer Science*, pages 223–241. Springer, 2021. `doi:10.1007/978-3-030-72013-1_12`.

66 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979. `doi:10.1016/0022-0000(79)90042-4`.

67 Grigorii Samuilovich Tseitin. On the complexity of derivation in propositional calculus. *Studies in Mathematics and Mathematical Logic*, 2:115–125, 1968.

68 Allen Van Gelder and Yumi K. Tsuji. Satisfiability testing with more reasoning and less guessing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 559–586. DIMACS/AMS, 1993.

69 Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. 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 *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014. `doi:10.1007/978-3-319-09284-3_31`.

70 He-Teng Zhang, Jie-Hong R. Jiang, Luca G. Amarù, Alan Mishchenko, and Robert K. Brayton. Deep integration of circuit simulator and SAT solver. In *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*, pages 877–882. IEEE, 2021. doi:10.1109/DAC18074.2021.9586331.

71 He-Teng Zhang, Jie-Hong R. Jiang, and Alan Mishchenko. A circuit-based SAT solver for logic synthesis. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, November 1-4, 2021*, pages 1–6. IEEE, 2021. doi:10.1109/ICCAD51958.2021.9643505.

72 He-Teng Zhang, Jie-Hong R. Jiang, Alan Mishchenko, and Luca G. Amarù. Improved large-scale SAT sweeping. In *Proc. 31st International Workshop on Logic & Synthesis*, 2022.