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 identical 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 identical 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 Besides new algorithms for gate extraction we also describe previous unpublished attempts to tackle this problem.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 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

1 Introduction

One of our motivations is to improve SAT solving for combinational equivalence checking of hardware circuits [27, 51, 61]. 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 diagrams (BDD) technology, i.e., BDD sweeping [50], which however was combined (if not replaced) with SAT sweeping [51] 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.

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 [21]. These benchmarks are regularly used in SAT competitions (for instance two 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, 68–70] 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* [27], 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 [23] fail to find short resolution proofs, as we have shown in previous work [42].

Equivalence checking of arithmetic circuits [12, 49] has similar applications and issues. In principle algebraic techniques [30, 48, 59] 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 the recursive nature of equivalence checking, requiring for each internal equivalence to learn two binary clauses can not proceed after learning one of them without a complete restart. 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.

2 Preliminaries

We assume that the reader is familiar with propositional satisfiability (SAT) and otherwise refer to [22]. 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 $l = \bar{v}$.

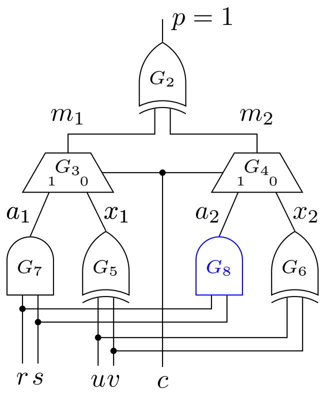
In Fig. 1 we present an example of a combinational equivalence checking problem (miter). This is an isomorphic miter as the two circuits compared are identical. In the experiments we also consider the case where one of the circuits is an optimized version of the other, since these optimized miters are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

Hybrid approaches to equivalence checking (starting from [51] and most recently [70]) 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

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

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



(b) miter circuit

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

(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 [65] 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}.

languages or “common sub-expression elimination” in compiler optimization. It is also implemented in libraries for the manipulation of binary decision diagrams (BDDs) [29] or and-inverter graphs (AIGs) [51] 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 the 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.

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” [42], “blocked-clause decomposition” [41], “simple probing” in Sect. 3, and “internal SAT sweeping” [18, 19], which all need to be limited or preempted.

3 Simple Probing

Simple probing is available in Lingeling since 2012 [10] motivated by the observation [42] that though hyper binary resolution (HBR) [3, 4, 39] combined with equivalent literal substitution (ELS) [2, 33, 53, 66] in theory can solve identical miters, in practice it fails to do so.

The problem with these existing HBR implementations [3, 4, 39, 42] is, that they are "global", and rely on complete failed literal probing followed or interleaved with a global form of ELS, i.e., 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 and thereby 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:

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

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:

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

Note, that such linear resolution chains correspond to reverse-unit propagation (Rup) [38] in clausal proofs [43, 44]. Next we have to substitute (w.l.o.g.) a_2 by a_1 in the formula:

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

This again boils down to resolution, which also explains why simple probing can produce Rup proofs [38] 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 [67] to speed-up proof checking.

This forms the core of simple probing. In the implementation we use a counting argument: first 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 as 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 (while avoiding to add duplicates), we check whether its dual clause with both literals negated already exists. For instance, assume that in our example applying HBR_1 is the first step. Then, when clause C_{29} is derived through HBR_2, as its dual (C_{28}) already exists, the equivalence $a_2 = a_1$ is immediately derived. To substitute one of the two literals by the other one, we traverse all clauses with the literal to substitute, apply the substitution and delete the original clause. While checking for dual clauses only needs to find all binary clauses in which a literal occurs, the substitution step requires full occurrence lists.

The complete preprocessing algorithm in Fig. 2 needs to determine which and when clauses are (re)considered as base clause. 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

```
    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 $(o \vee \overline{k}) \in F$
9             $\gamma(o)$++    // increment count of other literal $o$ by one
10          for all $r$ with $\gamma(\overline{r}) + 1 = |C|$ and $|r| \neq |l|$ and $(\overline{r} \vee l) \notin F$
11            add $(\overline{r} \vee l)$ to $F$    // HBR
12            if $(r \vee \overline{l}) \in F$        // checking for dual clause - ELS
13              substitute $l = r$ in all clauses $D \in F$ with $l$ or $\overline{l}$ in $D$
14              reschedule literals in resulting clauses by adding them to $\Lambda$
15              continue with outer while loop at line 3
```

Figure 2 Pseudo code of "simple probing" through local hyper binary resolution (HBR) and eager equivalent literal substitution (ELS): We interpret the given CNF F as sets of clauses, which in turn are sets of literals, thus both without 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 by the representative literal r (resp. $\overline{l}$ by $\overline{r}$). In the actual implementation we care for additional cases. For instance we also check for hyper binary resolved units, i.e., when $\gamma(r) = |C|$ in line 10.

clause candidates as scheduling objects. Initially all literals are candidate literals for simple probing and 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.

The simple probing algorithm will solve isomorphic miters of circuits with only AND gates. Actually after substituting equivalence of outputs of the compared circuits, the comparison in clauses of the miter XOR gates will yield a unit clause and 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. [42]. They proposed to interleave probing based HBR+ELS with saturating *ternary resolution* (TRN) [25] to simulate structural hashing for XOR and ITE gates, i.e., add all resolvents of at most length three between ternary gates. This is not only costly and needs to be localized in combination with simple probing but does not work for larger XOR gates with more than two inputs. Nevertheless, CaDiCaL and Lingeling 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) in essence failed. They are orders of magnitude slower than circuit-based techniques as already pointed out in the conclusion of [42] and again confirmed in our experiments. The key to obtain 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 [37, 56, 58] and we refer to the preprocessing chapter of the SAT handbook [23, Sect.9.6.2] for details. These works were either limited in scope or had as goal to recover an actual circuit, including the 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, and second the same clause can be used for extracting multiple gates, which requires to select a gate.

Recent work stays on the CNF level and uses blocked decomposition instead [5, 41, 46, 47]. A basic XOR-constraint extraction algorithm is described in [62] 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 [34, 36]. 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 [52, 60].

We only syntactically extract “gates”, trying to reverse the CNF encoding, e.g., from the clauses C_{22}, C_{23} in and C_{24} in the CNF of Fig. 1 we extract the “gate” (equation) $a_1 = r \wedge s$. Semantic extraction (such as [36, 60]) is much more powerful, but also much more expensive.

5 AND-Gate Extraction

The *basic-and-gate-extraction* algorithm is shown in Fig. 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 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, binary clauses and candidate clauses[1] this algorithm is too slow to run it 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 left-hand side literal for all the other r with $\bar{r} \in C$ instead of marking (line 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 Fig. 3.

Our first successful improvement counts the number of occurrences of literals in binary clauses and drops those candidate clauses C where no literal has enough negative occurrences in binary clauses. Actually, iterations of the loop in line 7-13 can always be skipped in Fig. 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 the 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 $l_3 = \bar{r}_1$ nor $l_3 = \bar{r}_2$ then l_3 is no LHS candidate

[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.

```
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 $\mu \equiv \bot$
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** $A$
```

Figure 3 Basic algorithm for extracting AND gates. As in Fig. 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 has to remove duplicated binary clauses first. Only binary clauses need to be watched, assuming base clauses can be traversed in some other way.

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 Fig. 3 while the second is used to mark the negation of remaining LHS candidates. When counting occurrences of marked literals in line 9-10 we update the second mark bit and later only consider LHS literals which have the second bit still set (*cf.* Fig. 11 in the Appendix).

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 [17] use XOR-gate extraction to make bounded variable elimination more effective, as originally proposed in [34] for AND gates. Our basic algorithm in Fig. 4 follows these implementations and corresponds to a similar algorithm presented in [62].

In line 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 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 by the negation l of l_i ("1" on the LHS 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. As a consequence the functional dependency graph between inputs and outputs of our extracted gates becomes cyclic as soon as a single XOR constraint is extracted (and all the gates it covers). 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 [37, 58].

Note that for each XOR constraint found for a base clause C with m literals, the CNF

```
   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 $i^{\text{th}}$ 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 $i^{\text{th}}$ 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 $X$
```

Figure 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.

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 as 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 encode 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 limit the size of the base clauses in line 4 in Fig. 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 which can be part of the encoding of an XOR gate. Base clauses of size $m = |C|$ considered to extract 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 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 which is also 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 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 [34]. We also explored using hashing instead (still a compile time parameter) with 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

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

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

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 [70]) it gave a $1000\times$ improvement in solving time: 1.79 s when extracting vs. 2023.41 s when not extracting ITE gates (*cf.* Fig. 23+24+25 in the appendix).

As with AND and XOR gates we have been using a simple algorithm for ITE-gates 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 Fig. 5. To encode an ITE gate $(l = c\ ?\ t : e)$ exactly the following four ternary clauses are needed $(\bar{c} \lor \bar{l} \lor t)$, $(\bar{c} \lor l \lor \bar{t})$, $(c \lor \bar{l} \lor e)$, and $(c \lor l \lor \bar{e})$ ignoring two potential additional redundant clauses $(\bar{l} \lor t \lor e)$ and $(l \lor \bar{t} \lor \bar{e})$, which might be used to improve arc-consistency of the encoding. Observe that the first two clauses encode the conditional equivalence $c \to l = t$ and the third and fourth the conditional equivalence $\bar{c} \to l = e$.

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 Fig. 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.

There are further implementation details though. 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.

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

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

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

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

8 Congruence Closure

In SMT solvers [7] the congruence closure algorithm found already several applications, for example in ground theory solvers [55], or during quantifier instantiation [6]. It makes use of 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 and, in contrast to structural hashing, it does not require any topological order of the variables, thus can also be applied to cyclic functional definitions.

Extracted or rewritten gates have to be *normalized* to increase chances of matching other gates. For AND gates the only form of normalization which can be achieved is to sort RHS literals, assuming 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 parity of the number of negated RHS literals is even, their negations cancel, and we can simply drop them; if the parity is odd, we also drop the negations, but negate the LHS literal instead.

For an ITE gate $(l = c \,?\, t : e)$ a normalization strategy known from the BDD literature [26] 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 those two parts as a key (operator and RHS literals). If a gate is found with the same operator and RHS we derived an equivalence between the two LHS side literals of the gates. This equivalence is recorded in a *union-find data-structure* [64], where

```
   merge-literals (CNF $F$, queue $Q$, representatives $\rho$, literals $l_1$, $l_2$)    // $F$, $Q$, $\rho$ 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 ($F$, $Q$, $\rho$, $l_1$, $l_2$)
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 $\rho$ 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 ($F$, $Q$, $\rho$, $k'$, $k''$)
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 $\rho(F)$
```

Figure 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 line 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.

every literal points to its representative. Here we use again the variable order and ensure that each literal points to either itself or to a smaller representative.

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 (line 13-18 in Fig. 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 (resp. with the negation of 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 $(l = a \wedge b)$ 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* the equivalence $l_1 = l_2$ consists of first 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 Fig. 7). Once the loop terminates, this augmented CNF is passed to a global ELS procedure, which substitutes all equivalent literals in one pass over the formula.

Besides those (actually rather complex) ways of rewriting gates another last 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 $(l = a \wedge b)$ 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 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.* Fig. 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 for instance be the outcome, when congruence closure is applied to isomorphic miters.

As already pointed out in Sect. 3, after matching two isomorphic gates and substituting one LHS literal by its representative in all clauses in which it occurs, this 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 that gate became garbage or is removed. This risks to turn 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 to remove identical clauses modulo equivalent literals as recorded in the union-find data-structure.

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) producing DRUP proofs and external checking in production [45, 63, 67].

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 [57] for XORs or how ternary resolution is used in [42] 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 [32], 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.

9 Benchmarks

The IWLS'22 paper [70] by He-Teng Zhang, Jie-Hong R. Jiang, Alan Mishchenko, and Luca Amarù, which can be considered as an update on their DAC'21 paper [68] was focusing on a hybrid approach to SAT-sweeping, i.e., using a SAT solver incrementally, taking circuit structure into account. Experiments in [70] used a subset of the benchmarks from [68] which also contains five miters `n01`, `n04`, `n06`, and `test01` and `test02` in AIGER [8] format provided to us by Alan Mishchenko through private communication and which were considered hard for SAT sweeping, particularly for monolithic CNF-level SAT solving.

It turned out, confirmed by Alan Mishchenko, that the outputs of `test01` and `test02`, were flipped in the process they were generated. This does not invalidate the SAT sweeping experiments in [68, 70] at all, but needs to be taken care off, when encoding them into CNF with AigToCnf, by simply first negating the outputs with AigFlip. Further note, that the other three AIGs `n01`, `n04`, and `n06` are not negated but have multiple outputs and thus we joined them by disjunction with AigOr. These tools are part of the AIGER library `https://github.com/arminbiere/aiger` (and also included in the artifact).

On the development branch of the AIGER repository we have extended AigToCnf to detect XOR and ITE gates in AIGER circuits, i.e., 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 has positive effects on running time too as our experiments will show. Thus our IWLS'22 CNF benchmarks, available at [15], come in two flavors tagged xits with special treatment of XOR and ITE gates during Tseitin encoding and ands without.

Continuing the discussion of the last section regarding testing, we not only checked 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 extent we generated simple AIGER models with our AigFuzz fuzzer, used AigMiter to produce a miter and then encoded it to CNF with this new version of AigToCnf that detects XOR and ITE gates.

Our second set of HWMCC'12 benchmarks submitted to the SAT Competition 2013 [21] provides benchmarks were HBR has difficulties. It consists of miters for 341 AIGER 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 [28] to optimize the models (using the '`&dc2`' command) which beside the isomorphic miters, tagged iso, also gives optimized miters, tagged opt. We have further extended this benchmark set by using our new version of AigToCnf with XOR and ITE matching, which results in four variants of the 341 AIGER models: ands-iso, ands-opt, xits-iso and xits-opt. They are all available [16].

10 Experiments

In our experiments we used the bwForCluster Helix with AMD Milan EPYC 7513 CPUs and enforced a memory limit of 15 GB and a time limit of 5000 seconds with Runlim. 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 (`sblitter | mequick | lingeling`) [41], the winner Sbva-CaDiCaL [40] of the SAT Competition 2023, the latest version 1.9.5 of CaDiCaL [20] (for details on 1.5.3 as used in Sbva-CaDiCaL see the appendix) and MiniSat 2.2.0 [35].

Our primary results on the HWMCC'12 miters in Fig. 8 show that indeed 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). Optimized miters (opt) are, as expected, harder to solve, but on these our new approach even surpasses the hybrid SAT sweeping technique [70] implemented in Abc (command '&fraig -y') with the AIGER model as input.

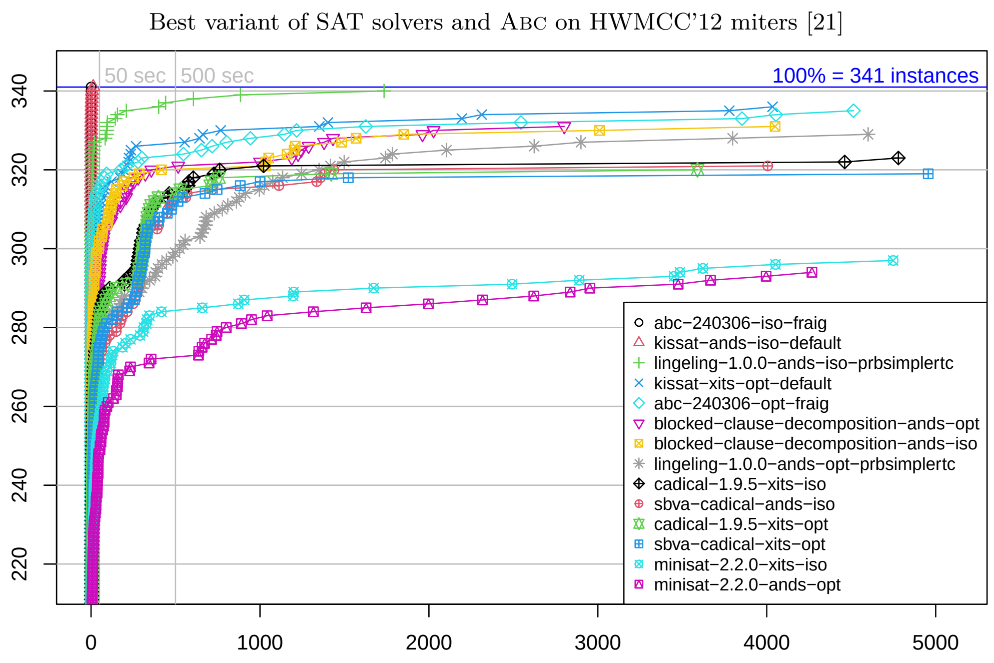


Figure 8 For each SAT solver we only show the best encoding variant (ands or xits).

Running simple probing in LINGELING until completion (prbsimplertc) is the only CNF level approach that can compete on isomorphic miters, but is not competitive on optimized ones.

Figure 9 allows a closer look at the IWLSS'22 benchmarks and also for KISSAT shows that the overhead for proof production (proof) is low and that proof checking (check) has comparable run-time as solving. Variants of KISSAT are compared in Fig. 10.

Source code is available as supplementary material in EasyChair and will be published open source on acceptance of the paper. The HWMCC'12 benchmarks [16] and ILWS'22 benchmarks [15] are available on Zenodo as well as all the log files [14].

11 Conclusion

We revisited the idea of congruence closure applied to gates extracted from CNF by 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 accordingly 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 which can instantly solve simple but large isomorphic CNF encoded miters. Further, it gives substantial improvements on industrial relevant optimized miters, where our CNF level approach actually showed superior performance over a dedicated circuit level SAT sweeping technique.

Acknowledgment. This work was supported by the state of Baden-Württemberg through bwHPC, the German Research Foundation (DFG) through grant INST 35/1597-1 FUGG, and by a gift from Intel Corporation.

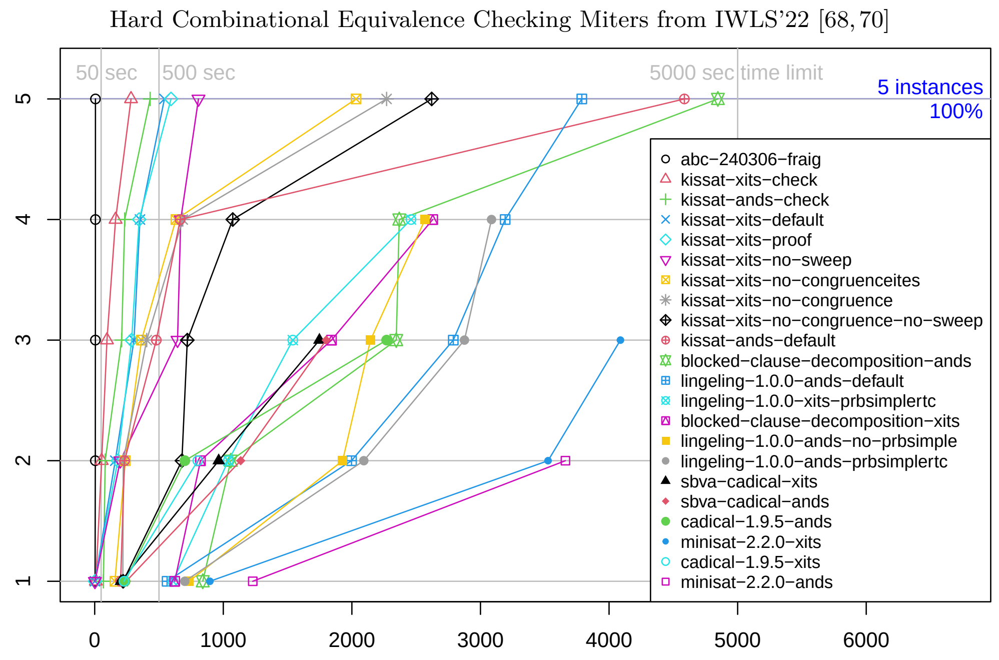


Figure 9 These miters from [68] were target of optimizations reported in [70]. They are indeed hard for monolithic SAT solving starting after Tseitin encoding. Results on benchmark `test02` are particularly interesting as kissat-xits-default took 1.79 sec. to solve it, Abc 5.75 sec., while disabling extraction of ITE gates in kissat-xits-no-congruenceites already needs 2032.41 sec. and plain AND-only Tseitin encoding in kissat-ands-default even 4585.76 sec. (*cf.* Fig. 23 in the appendix).

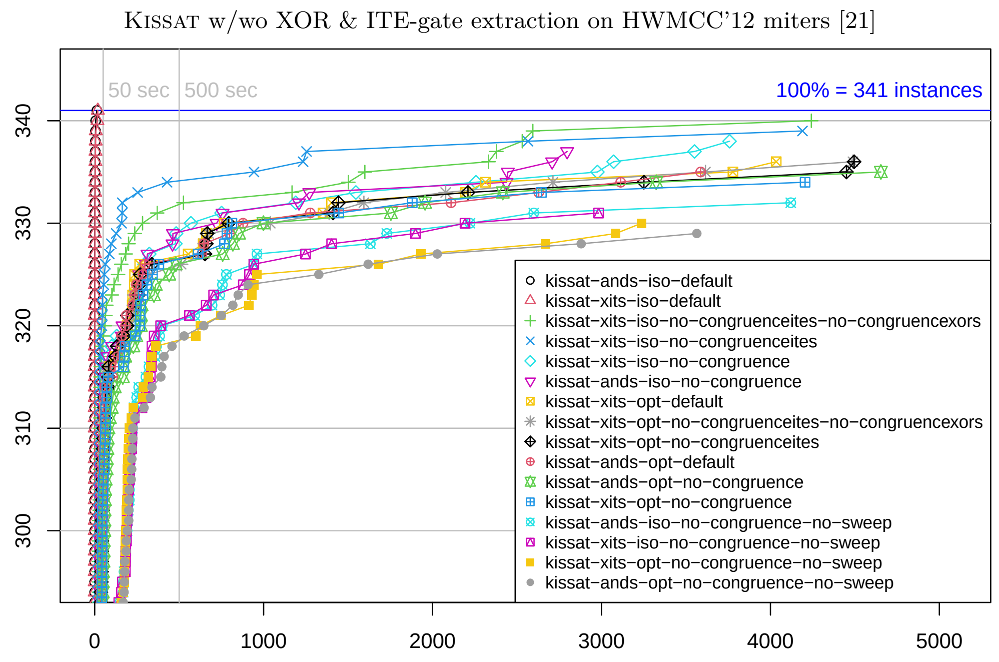


Figure 10 Comparing different variants of Kissat where no-congruenceites disables extraction of ITE gates, no-congruencexors of XOR gates and no-sweep disables internal SAT sweeping [18, 19].

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. University of 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. University of 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. University of 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. University of Helsinki, 2019.

14 Armin Biere. Clausal Congruence Closure Paper Logs, Plots and Tables, March 2024. `doi:10.5281/zenodo.10829559`.

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

16 Armin Biere. CNF Encoded Isomorphic and Optimized Miters from Hardware Model Checking Competition 2012 Models, March 2024. `doi:10.5281/zenodo.10823128`.

17 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. University of Helsinki, 2020.

18 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. University of Helsinki, 2022.

19 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. University of Helsinki, 2021.

20 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. University of Helsinki, 2023.

21 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. University of Helsinki, 2013.

22 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`.

23 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`.

24 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`.

25 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`.

26 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`.

27 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`.

28 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`.

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

30 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`.

31 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In Leonardo de Moura, editor, *Automated Deduction – CADE 26*, pages 220–236, Cham, 2017. Springer International Publishing.

32 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`.

33 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`.

34 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.

35 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`.

36 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`.

37 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`.

38 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`.

39 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.

40 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.

41 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.

42 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.

43 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.

44 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.

45 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.

46 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.

47 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.

48 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.

49 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. University of Helsinki, 2019.

50 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.

51 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.

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

53 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.

54 Sid Mijnders, Boris de Wilde, and Marijn Heule. Symbiosis of search and heuristics for random 3-SAT. In D. Mitchell and E. Ternovska, editors, *Third International Workshop on Logic and Search (LaSh 2010)*, 2010.

55 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.

56 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.

57 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.

58 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.

59 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/.

60 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.

61 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.

62 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.

63 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.

64 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.

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

66 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.

67 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.

68 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.

69 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.

70 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.

Appendix

In this appendix we give in Fig. 11 pseudo-code for the fast AND gate extraction algorithm. It contains the optimizations to the basic version in Fig. 3 discussed briefly in Sect. 5.

As optimizations described in Sect. 6 to improve the XOR-gate extraction algorithm in Fig. 4 are on the one-hand easy to add but regarding occurrence lists on the other hand hard to describe on an abstract level, we do not have additional pseudocode for this optimized version, but beside the textual description in the main part of the paper in Sect. 6 refer to the provided source code of Kissat instead (included in the EasyChair supplement).

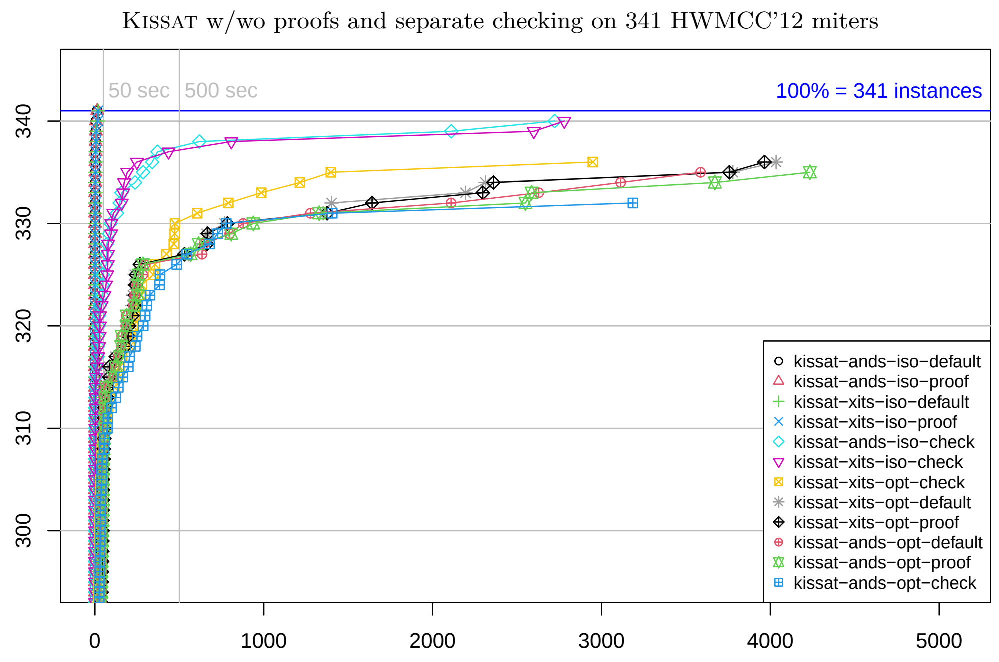
We also have in the appendix additional experimental results which uses this additional space to high-light some aspects for clarity that are harder to grasp from the dense plots in experimental Sect. 10 alone. These plots are contained in the artefact with all log files of the experiments and will also be made available in an extended version of the paper.

We then give in Fig. 24+25 an example of an optimized miter, which also shows why ITE normalization is important and also explains the performance of kissat-xits-opt-default vs. kissat-ands-opt-default on the IWLS'22 benchmark `test02`.

Finally we want to elaborate on the average learned clause length, related to the comment in the introduction on CDCL not being able to produce short proofs. Therefore we have rerun without congruence closure (no-congruence) but with more verbose statistics all the isomorphic HWMCC'12 miters again (these are in the metrics directories in the artifact) and computed the average learned clause lengths over all miters, which is 43.6 literals per learned clause for ands-iso-no-congruence, 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.

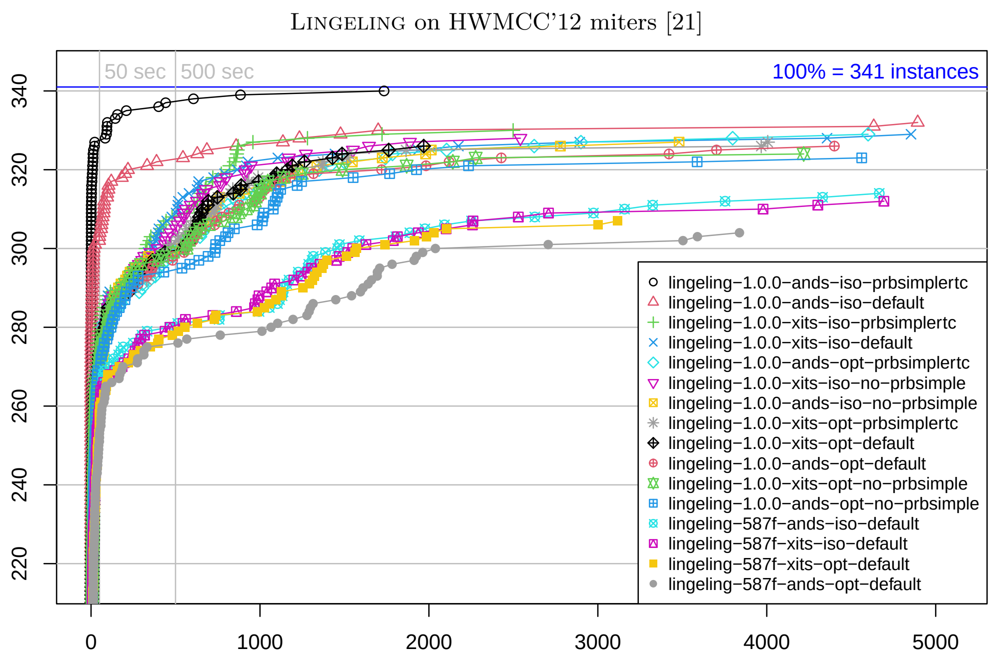
```
   *fast-and-gate-extraction* (CNF $F$)
1    resulting AND gates $A = \emptyset$
2    literals $L$ = all literals in $F$
3    negative binary occurrence count $\gamma\colon L \to \mathbb{N}$ initialized to $\gamma \equiv 0$
4    **for all** binary clauses $C = \{l_1, l_2\} \in F$
5      $\gamma(\bar{l}_1)$++,  $\gamma(\bar{l}_2)$++
6    **for all** clauses $C \in F$ with $m = |C| > 2$
7      let $(l_1 \vee \ldots \vee l_m) = C$ with $\gamma(l_i) \leq \gamma(l_{i+1})$ for $1 \leq i < m$     // sort
8      **if** $\gamma(l_m) < m - 1$ **then** continue outer loop at line 6        // (i)
9      marks $\mu_1\colon L \to \mathbb{B}$ initialized to $\mu_1 \equiv \bot$
10     **for all** literals $r$ with $\bar{r} \in C$
11       $\mu_1(r) = \top$
12     let $i = \min\,\{j \mid \gamma(l_j) \geq m - 1\}$        // (i) as it skips $l_1, \ldots, l_{i-1}$
13     $n = 0$
14     marks $\mu_2\colon L \to \mathbb{B}$ initialized to $\mu_2 \equiv \bot$
15     **for all** binary clauses $(\overline{l_i} \vee r) \in F$
16       **if** $\mu_1(r)$ **then** $n$++, $\mu_2(r) = \top$
17     **if** $n = m - 1$
18       let $(l_i \vee \bar{r}_1 \vee \ldots \vee \bar{r}_n) = C$
19       add AND gate  $(l_i = r_1 \wedge \ldots \wedge r_n)$  to $A$
20     **for** $l = l_{i+1}\ \ldots\ l_m$        // thus in increasing $\gamma$ order
21       **if** $\neg\mu_2(\bar{l})$ **then** continue loop at line 20        // (ii)
22       $n = 0$
23       marks $\mu_3\colon L \to \mathbb{B}$ initialized to $\mu_3 \equiv \bot$
24       **for all** binary clauses $(\bar{l} \vee r) \in F$
25         **if** $\mu_1(r)$ **then** $n$++, $\mu_3(r) = \top$
26       **if** $n = m - 1$
27         let $(l \vee \bar{r}_1 \vee \ldots \vee \bar{r}_n) = C$
28         add AND gate  $(l = r_1 \wedge \ldots \wedge r_n)$  to $A$
29       **for all** $l$ with $\mu_2(l)$
30         **if** $\neg\mu_3(l)$ **then** $\mu_2(l) = \bot$
31   **return** $A$
```

Figure 11 Improved algorithm for extracting AND gates with two optimizations: (i) count occurrences in binary clauses to filter LHS literals; (ii) mark, update and check remaining LHS candidates. The first mark bit μ_1 corresponds to μ in Fig. 3. Negation of remaining candidate LHS literals are marked with μ_2, filtered by which occurred negated with μ_3.



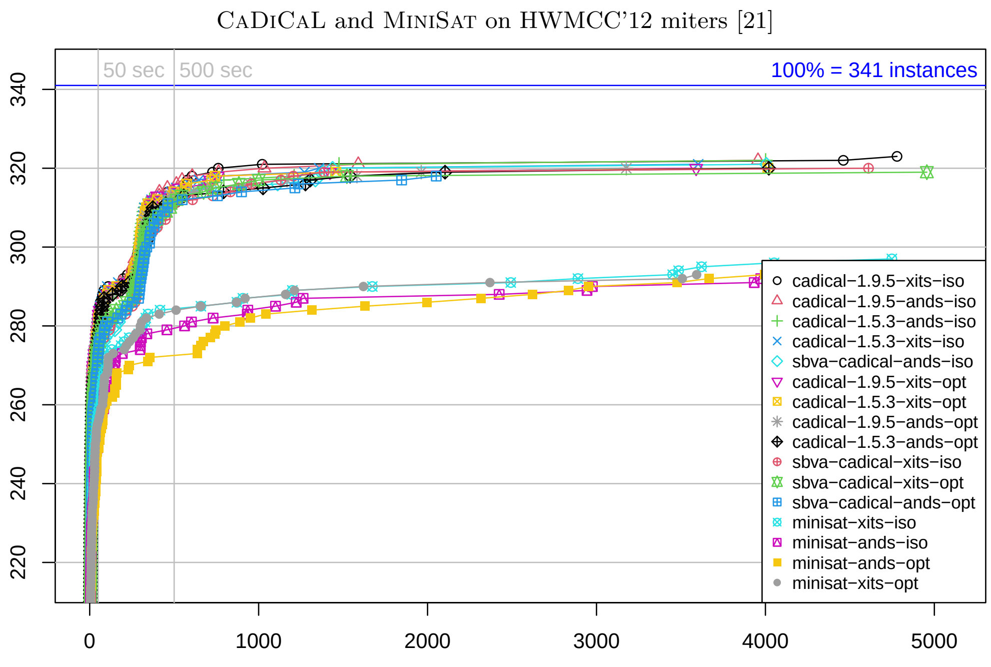
	cnt	ok	uns	fld	to	mo	unk	real	time	space	max	best
kissat-ands-iso-default	341	341	341	0	0	0	0	68	56	13026	2757	319
kissat-ands-iso-proof	341	341	341	0	0	0	0	286	61	14426	2783	49
kissat-xits-iso-default	341	341	341	0	0	0	0	136	90	15203	3064	167
kissat-xits-iso-proof	341	341	341	0	0	0	0	200	94	15514	3086	68
kissat-ands-iso-check	341	340	340	1	1	0	0	7968	7843	22990	1484	0
kissat-xits-iso-check	341	340	340	1	1	0	0	8653	8556	22853	1641	0
kissat-xits-opt-check	341	336	336	5	0	0	5	14246	14124	76523	8496	0
kissat-xits-opt-default	341	336	336	5	5	0	0	22358	22184	12078	840	4
kissat-xits-opt-proof	341	336	336	5	5	0	0	22915	22682	12493	848	0
kissat-ands-opt-default	341	335	335	6	6	0	0	20491	20381	12436	811	4
kissat-ands-opt-proof	341	335	335	6	6	0	0	22364	22131	12730	834	2
kissat-ands-opt-check	341	332	332	9	0	1	8	12739	12639	70009	11891	0

Figure 12 This shows the run-time of KISSAT solving the four sets of benchmarks in the default configuration with congruence closure and sweeping enabled. The same configuration is then also run with DRUP proof production (proof) and afterwards the proofs are checked (check) with DPR-TRIM. Few optimized miters could not be solved and accordingly proof checking failed for them too. Proof checking of one instance failed with DPR-TRIM, due to DPR-TRIM handling duplicated literals in a different way than DRAT-TRIM, but manually checking with DRAT-TRIM succeeded. We ran out of resources three times and then reproduced and checked proofs separately. Thus these experiments confirm that both congruence closure and sweeping produce correct proofs. Clearly solving with and without proof production of isomorphic miters is very fast (first two vertical lines) but checking has quite some overhead (up to two orders of magnitude). For optimized miters checking is faster with XOR and ITE Tseitin encodings than solving (kissat-xits-opt-check vs. kissat-xits-opt-default) but in the other variants run-times are similar.



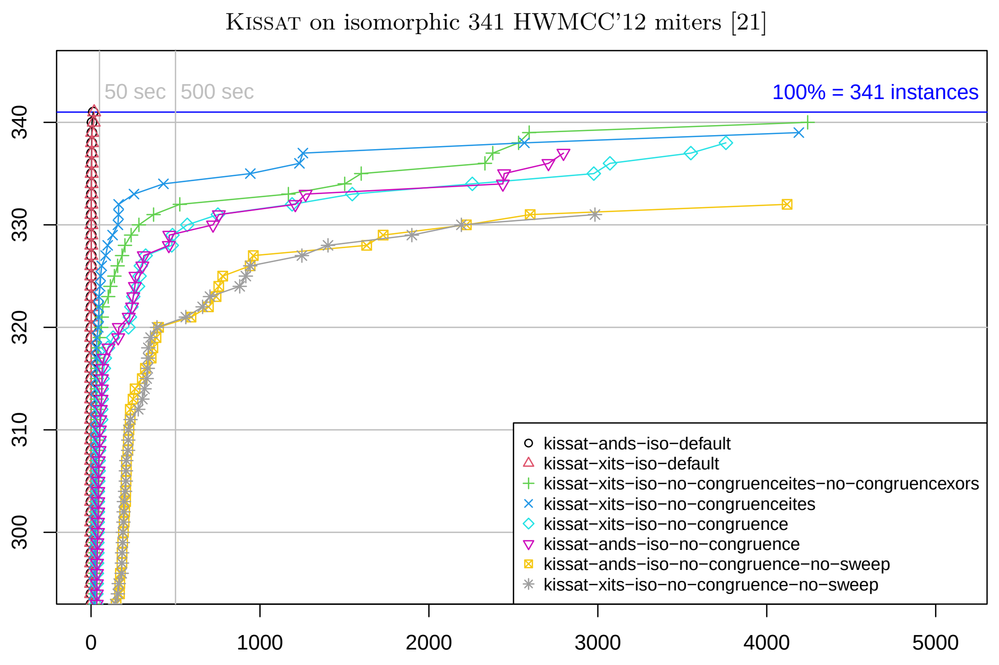
```
                                      cnt  ok uns fld to unk  real  time space  max best uniq
lingeling-1.0.0-ands-iso-prbsimplertc 341 340 340   1  1   0  5252  5216 16808 2702  298    8
     lingeling-1.0.0-ands-iso-default 341 332 332   9  9   0 20681 20571  8615  913  203    0
lingeling-1.0.0-xits-iso-prbsimplertc 341 330 330  11 11   0 27782 27638 11946 1015    2    0
     lingeling-1.0.0-xits-iso-default 341 329 329  12 12   0 36941 36766 11708  912    7    0
lingeling-1.0.0-ands-opt-prbsimplertc 341 329 329  12 12   0 47208 46982 11488  633    3    0
lingeling-1.0.0-xits-iso-no-prbsimple 341 328 328  13 13   0 30479 30338 11063  701    4    0
lingeling-1.0.0-ands-iso-no-prbsimple 341 327 327  14 14   0 37530 37348 11564  709    4    0
lingeling-1.0.0-xits-opt-prbsimplertc 341 327 327  14 14   0 37709 37531 10647  916    2    0
     lingeling-1.0.0-xits-opt-default 341 326 326  15 15   0 31107 30960 10303  626    0    0
     lingeling-1.0.0-ands-opt-default 341 326 326  15 15   0 43816 43603 10557  600    3    0
lingeling-1.0.0-xits-opt-no-prbsimple 341 324 324  17 17   0 35213 35046 10906  810    6    0
lingeling-1.0.0-ands-opt-no-prbsimple 341 323 323  18 18   0 41896 41689 11200  741    1    0
      lingeling-587f-ands-iso-default 341 314 314  27 25   2 65441 65068 25784 1958    0    0
      lingeling-587f-xits-iso-default 341 312 312  29 28   1 59764 59459 21083 1148    4    0
      lingeling-587f-xits-opt-default 341 307 307  34 33   1 46307 46079 17281  560    4    0
      lingeling-587f-ands-opt-default 341 304 304  37 35   2 54291 53887 21067  835    2    0
```

Figure 13 For LINGELING we use in our experiments the latest version 1.0.0, which however did not change much since 2018, also in the runs with blocked clause decomposition (BCD) (`sblitter | mequick | lingeling`) [41]. The other version 587f is the version which in our paper on revisiting HBR [42] was the fastest CDCL solver, on par with MARCHRW [54], on an earlier set of isomorphic miter benchmarks, while here we use the actually harder HWMCC'12 benchmark set with both isomorphic and optimized miters, as well as with two different CNF encodings. On these benchmarks running simple probing until completion (prbsimplertc) is doable and solves almost all isomorphic miters, but of course only when using the only-AND gate encoding (ands). Also the default version of LINGELING works reasonably well on these, even though it is preempted after a fixed number of resolution steps and thus can not take full advantage of isomorphic gates.



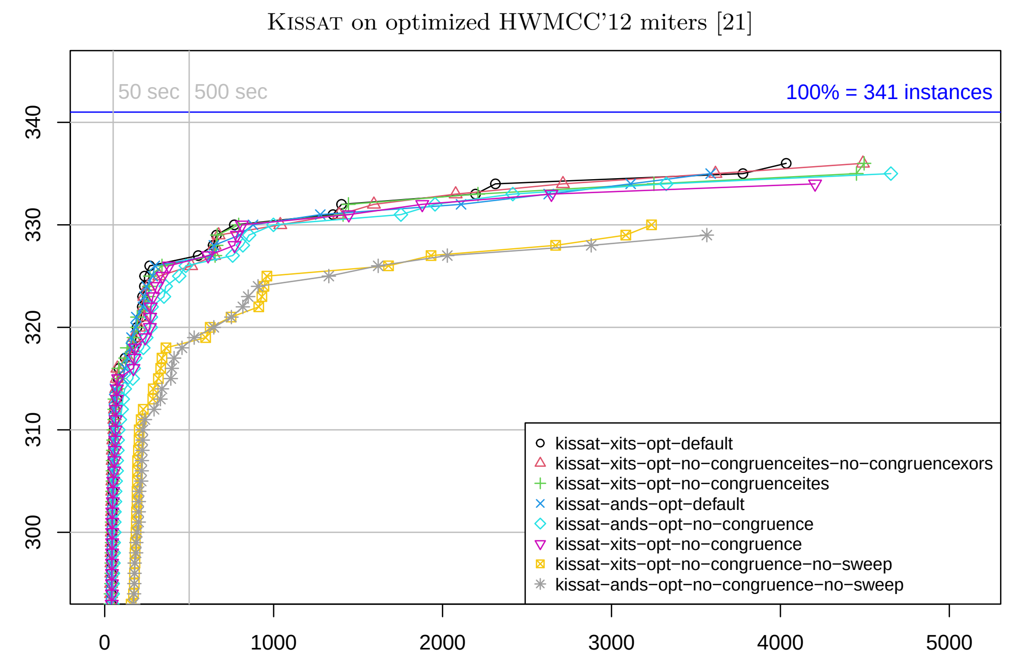
	cnt	ok	uns	fld	to	mo	real	time	space	max	best	uniq
cadical-1.9.5-xits-iso	341	323	323	18	18	0	23289	23111	20586	1783	33	1
cadical-1.9.5-ands-iso	341	322	322	19	19	0	18950	18799	19873	1747	25	0
cadical-1.5.3-ands-iso	341	322	322	19	19	0	21399	21260	16966	1472	15	0
cadical-1.5.3-xits-iso	341	321	321	20	20	0	17870	17764	17210	1522	30	0
sbva-cadical-ands-iso	341	321	321	20	19	1	24523	24272	53561	10248	9	0
cadical-1.9.5-xits-opt	341	320	320	21	21	0	17592	17407	18212	1626	14	0
cadical-1.5.3-xits-opt	341	320	320	21	21	0	17835	17582	15452	1393	15	0
cadical-1.9.5-ands-opt	341	320	320	21	21	0	20406	20128	18784	1572	8	0
cadical-1.5.3-ands-opt	341	320	320	21	21	0	22682	22412	15440	1328	8	0
sbva-cadical-xits-iso	341	320	320	21	19	2	24088	23763	44657	10152	7	0
sbva-cadical-xits-opt	341	319	319	22	21	1	21814	21461	42204	11383	8	0
sbva-cadical-ands-opt	341	318	318	23	21	2	20333	20029	33358	10066	2	0
minisat-xits-iso	341	297	297	44	44	0	36797	36605	17175	1967	116	0
minisat-ands-iso	341	294	294	47	47	0	37495	37295	16134	1226	56	0
minisat-ands-opt	341	294	294	47	47	0	43983	43746	16501	913	28	0
minisat-xits-opt	341	293	293	48	48	0	21960	21844	14386	1889	52	0

Figure 14 Here we compare the other considered CDCL solvers CaDiCaL, Sbva-CaDiCaL and MiniSat on our HWMCC'12 benchmark set with both encodings (xits and ands) and for both isomorphic and optimized miters. Version 1.9.5 is the latest release of CaDiCaL, while version 1.5.3 is the identical version used in Sbva-CaDiCaL, the winner of the SAT Competition 2023 combined with structural bounded variable addition. For MiniSat we use the well known version 2.2.0.



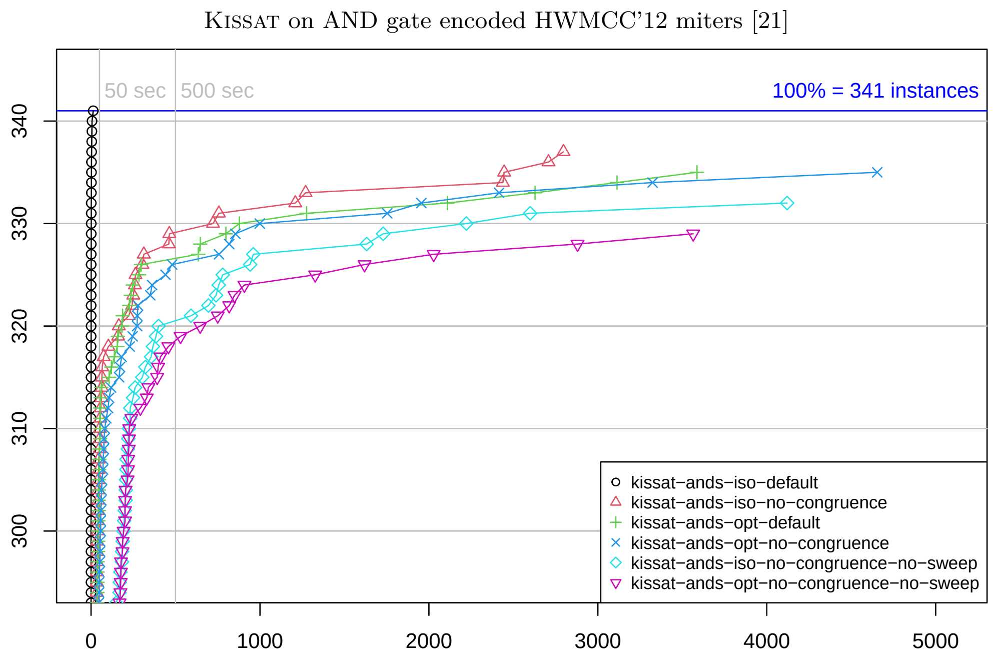
	cnt	ok	uns	fld	to	real	time	space	max	best
kissat-ands-iso-default	341	341	341	0	0	68	56	13026	2757	337
kissat-xits-iso-default	341	341	341	0	0	136	90	15203	3064	168
kissat-xits-iso-no-congruenceites-no-congruencexors	341	340	340	1	1	22558	22407	17647	2628	14
kissat-xits-iso-no-congruenceites	341	339	339	2	2	13312	13156	16370	3126	18
kissat-xits-iso-no-congruence	341	338	338	3	3	25502	25333	14445	1378	4
kissat-ands-iso-no-congruence	341	337	337	4	4	20005	19893	13408	923	5
kissat-ands-iso-no-congruence-no-sweep	341	332	332	9	9	26648	26494	13035	914	6
kissat-xits-iso-no-congruence-no-sweep	341	331	331	10	10	23352	23200	11575	860	7

Figure 15 More variants of KISSAT than in Fig. 12 but focused on isomorphic miters (iso) only.



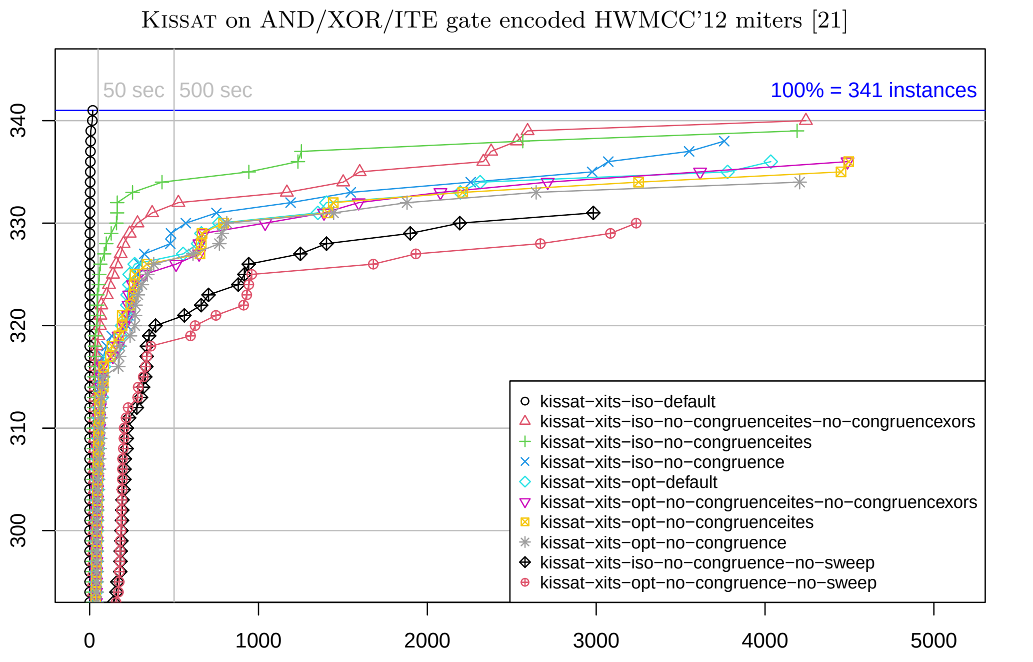
	cnt	ok	uns	fld	to	real	time	space	max	best
kissat-xits-opt-default	341	336	336	5	5	22358	22184	12078	840	125
kissat-xits-opt-no-congruenceites-no-congruencexors	341	336	336	5	5	23888	23758	12581	844	54
kissat-xits-opt-no-congruenceites	341	336	336	5	5	24833	24677	12317	787	41
kissat-ands-opt-default	341	335	335	6	6	20491	20381	12436	811	66
kissat-ands-opt-no-congruence	341	335	335	6	6	24186	24043	12028	769	14
kissat-xits-opt-no-congruence	341	334	334	7	7	18866	18740	10976	770	20
kissat-xits-opt-no-congruence-no-sweep	341	330	330	11	11	26354	26215	10495	765	49
kissat-ands-opt-no-congruence-no-sweep	341	329	329	12	12	24754	24613	10301	812	35

Figure 16 More variants of KISSAT than in Fig. 12 but focused on optimized (opt) miters only.



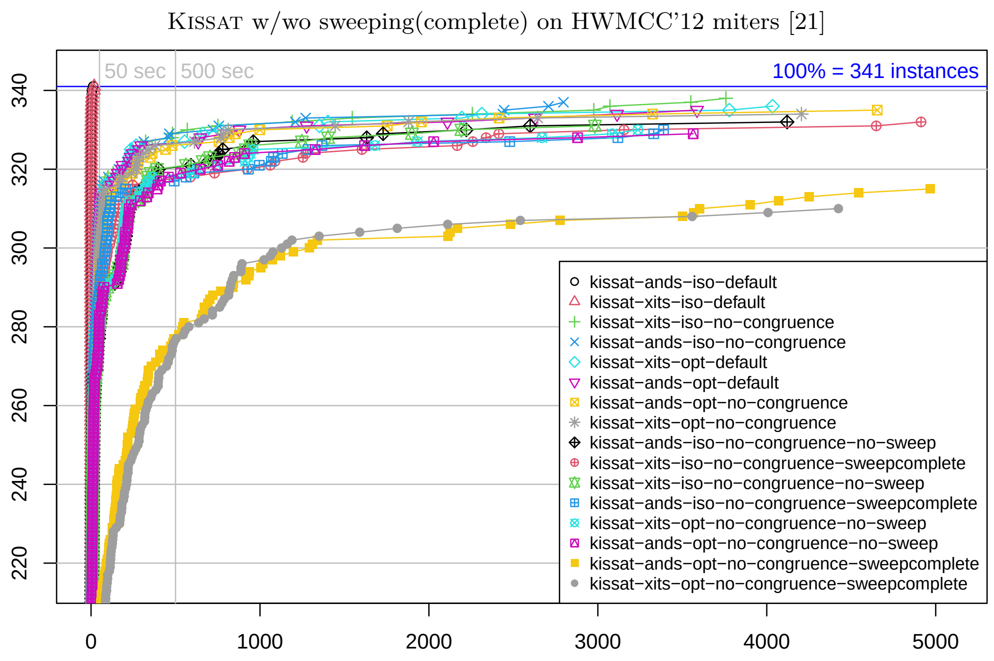
```
                                      cnt  ok uns fld to  real  time space  max best uniq
            kissat-ands-iso-default 341 341 341   0  0    68    56 13026 2757  341    4
        kissat-ands-iso-no-congruence 341 337 337   4  4 20005 19893 13408  923    5    0
            kissat-ands-opt-default 341 335 335   6  6 20491 20381 12436  811    4    0
        kissat-ands-opt-no-congruence 341 335 335   6  6 24186 24043 12028  769    4    0
kissat-ands-iso-no-congruence-no-sweep 341 332 332   9  9 26648 26494 13035  914    6    0
kissat-ands-opt-no-congruence-no-sweep 341 329 329  12 12 24754 24613 10301  812    5    0
```

Figure 17 More variants of Kissat than in Fig. 12 but focused on AND (ands) encoded miters.



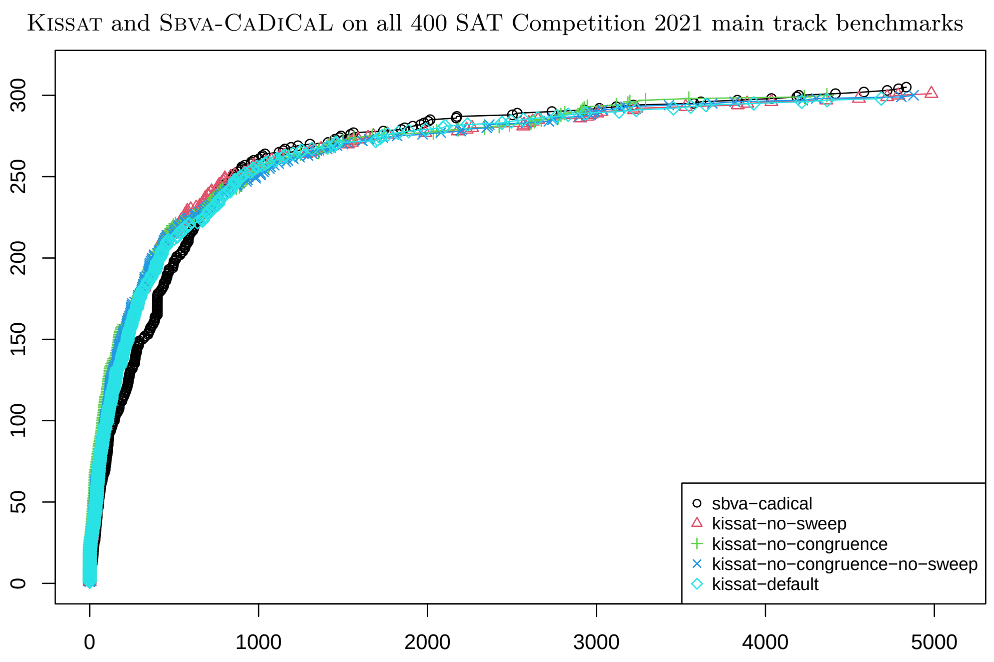
	cnt	ok	uns	fld	to	real	time	space	max	best	uniq
kissat-xits-iso-default	341	341	341	0	0	136	90	15203	3064	329	1
kissat-xits-iso-no-congruenceites-no-congruencexors	341	340	340	1	1	22558	22407	17647	2628	16	0
kissat-xits-iso-no-congruenceites	341	339	339	2	2	13312	13156	16370	3126	25	0
kissat-xits-iso-no-congruence	341	338	338	3	3	25502	25333	14445	1378	4	0
kissat-xits-opt-default	341	336	336	5	5	22358	22184	12078	840	4	0
kissat-xits-opt-no-congruenceites-no-congruencexors	341	336	336	5	5	23888	23758	12581	844	3	0
kissat-xits-opt-no-congruenceites	341	336	336	5	5	24833	24677	12317	787	3	0
kissat-xits-opt-no-congruence	341	334	334	7	7	18866	18740	10976	770	3	0
kissat-xits-iso-no-congruence-no-sweep	341	331	331	10	10	23352	23200	11575	860	7	0
kissat-xits-opt-no-congruence-no-sweep	341	330	330	11	11	26354	26215	10495	765	6	0

Figure 18 More variants of Kissat as in Fig. 12 on AND/XOR/ITE (xits) encoded miters.



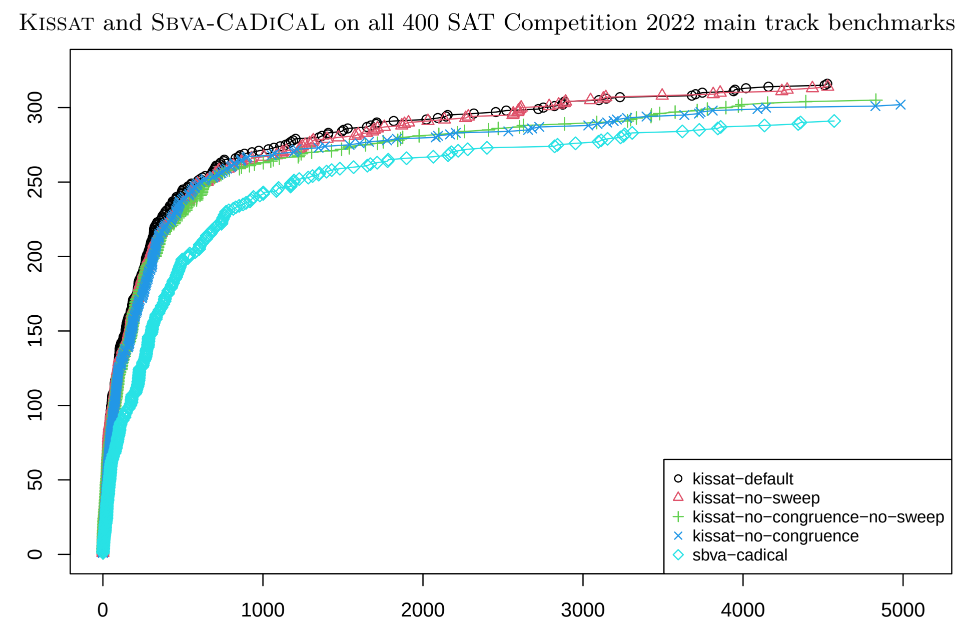
```
                                            cnt  ok uns fld to  real  time space  max best
                    kissat-ands-iso-default 341 341 341   0  0    68    56 13026 2757  337
                    kissat-xits-iso-default 341 341 341   0  0   136    90 15203 3064  168
               kissat-xits-iso-no-congruence 341 338 338   3  3 25502 25333 14445 1378    4
               kissat-ands-iso-no-congruence 341 337 337   4  4 20005 19893 13408  923    5
                    kissat-xits-opt-default 341 336 336   5  5 22358 22184 12078  840    4
                    kissat-ands-opt-default 341 335 335   6  6 20491 20381 12436  811    4
               kissat-ands-opt-no-congruence 341 335 335   6  6 24186 24043 12028  769    4
               kissat-xits-opt-no-congruence 341 334 334   7  7 18866 18740 10976  770    3
      kissat-ands-iso-no-congruence-no-sweep 341 332 332   9  9 26648 26494 13035  914    6
  kissat-xits-iso-no-congruence-sweepcomplete 341 332 332   9  9 36145 35970  7471  566    3
      kissat-xits-iso-no-congruence-no-sweep 341 331 331  10 10 23352 23200 11575  860    7
  kissat-ands-iso-no-congruence-sweepcomplete 341 330 330  11 11 26173 26051  6132  312    4
      kissat-xits-opt-no-congruence-no-sweep 341 330 330  11 11 26354 26215 10495  765    6
      kissat-ands-opt-no-congruence-no-sweep 341 329 329  12 12 24754 24613 10301  812    5
  kissat-ands-opt-no-congruence-sweepcomplete 341 315 315  26 26 81804 81439 19306 2335    2
  kissat-xits-opt-no-congruence-sweepcomplete 341 310 310  31 31 61775 61498 11215 1016    1
```

Figure 19 Running internal SAT sweeping [18, 19] until completion (sweepcomplete) is too costly.



	cnt	ok	sat	uns	fld	to	real	time	space	max	best	uniq
sbva-cadical	400	305	144	161	95	95	189159	188293	250406	13150	85	9
kissat-no-sweep	400	301	146	155	99	99	168641	167797	73433	2070	72	1
kissat-no-congruence	400	300	145	155	100	100	156601	155898	91761	5155	43	1
kissat-no-congruence-no-sweep	400	300	147	153	100	100	164641	163881	96961	4805	83	0
kissat-default	400	298	145	153	102	102	157570	156753	72616	2048	34	0

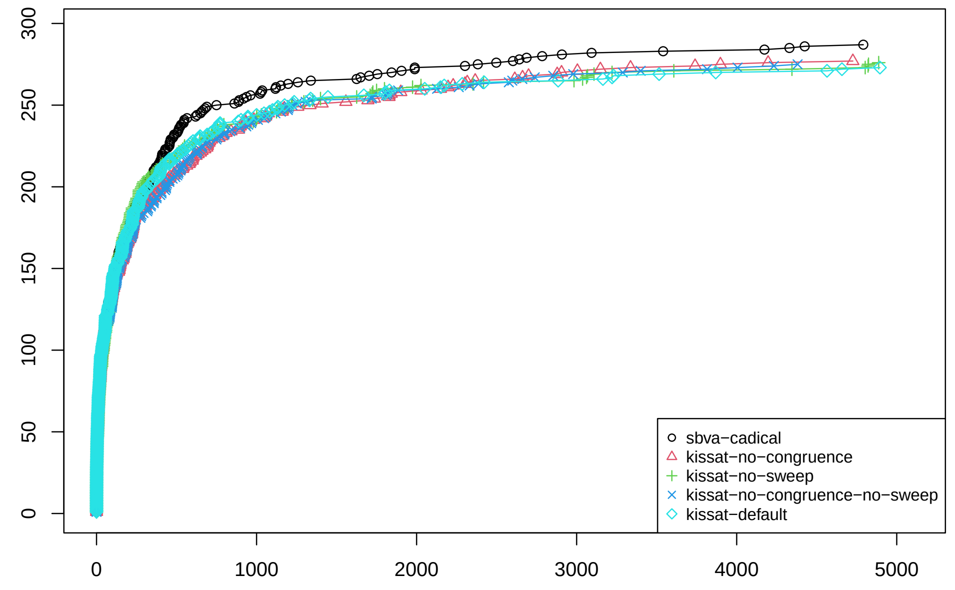
Figure 20 On the benchmarks from the main track of the SAT Competition 2021, all variants of KISSAT (with and without congruence closure) as well as the winner SBVA-CADICAL of the SAT Competition 2023 give very similar results.



	cnt	ok	sat	uns	fld	to	mo	real	time	space	max	best	uniq
kissat-default	400	316	159	157	84	84	0	158324	157382	225342	8159	60	0
kissat-no-sweep	400	314	157	157	86	86	0	161325	160489	222705	8167	82	0
kissat-no-congruence-no-sweep	400	305	148	157	95	95	0	160483	159635	194297	13444	73	0
kissat-no-congruence	400	302	147	155	98	97	1	147200	146454	183412	9138	61	0
sbva-cadical	400	291	143	148	109	102	7	180840	179882	591532	13925	48	4

Figure 21 In the main track of the SAT Competition 2022 congruence closure is successful. Actually 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 were solved instantly by congruence closure (in 0.07 sec. and 0.04 sec.) but were also solved without congruence closure (in 37.51 sec. and 25.99 sec.). The `default` version found 108 272 236 equivalent literals through congruence closure in all 400 benchmarks of the main track.

Kissat and Sbva-CaDiCaL on all 400 SAT Competition 2023 main track benchmarks



	cnt	ok	sat	uns	fld	to	mo	real	time	space	max	best	uniq
sbva-cadical	400	287	124	163	113	102	11	114132	113577	246598	14866	82	24
kissat-no-congruence	400	277	126	151	123	123	0	123847	123009	125809	8540	44	0
kissat-no-sweep	400	276	127	149	124	123	1	117499	116970	199575	14347	80	1
kissat-no-congruence-no-sweep	400	275	126	149	125	125	0	118995	118411	134948	7934	56	0
kissat-default	400	273	122	151	127	127	0	105853	105289	201410	14992	59	0

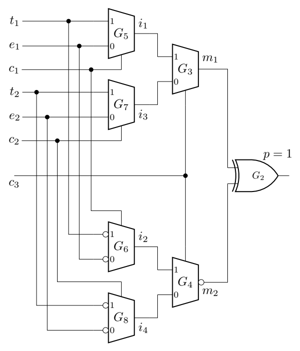
Figure 22 In the main track of the SAT Competition 2023 with many hard combinatorial problems Structured Bounded Variable Addition (SBVA) in Sbva-CaDiCaL, the winner of that track, has an edge over Kissat. The different variants of congruence closure are very similar even though the default version spent on average 4.41% of the running time in congruence closure.

	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-xits	204.17	1747.53	964.30	—	—
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	—	—
minisat-2.2.0-ands	1229.07	—	3660.71	—	—

Figure 23 The actual run-time on the IWLS'22 miters from [68, 70] (*cf.* CDF in Fig. 9).

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

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

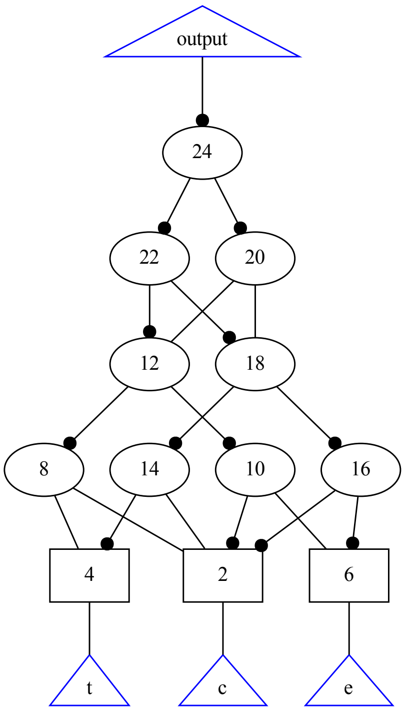


(b) miter circuit

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

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

Figure 24 An example of optimized miters. 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 [70] 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 (see Fig. 25). Recognizing and normalizing ITE gates allows the congruence closure approach to realize the equivalence between the two circuits efficiently (*cf.* Fig. 25)



(a) A miter of two if-then-elses in the AIGER format.

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

(b) The `ands` CNF encoding of the AIG.

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

(c) The `xits` CNF encoding of the AIG.

Figure 25 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. 25b) translates all 9 AND-nodes of the AIG independently from each other, resulting in 26 clauses over 12 Boolean variables. The `xits` encoding (Fig. 25c), 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. 9+23+24).