Clausal Equivalence Sweeping

Armin Biere
University Freiburg
Freiburg, Germany
biere@cs.uni-freiburg.de

Katalin Fazekas
TU Wien
Vienna, Austria
katalin.fazekas@tuwien.ac.at

Mathias Fleury
University Freiburg
Freiburg, Germany
fleury@cs.uni-freiburg.de

Nils Froleyks
Johannes Kepler University
Linz, Austria
nils.froleyks@jku.at

***Abstract*—The state-of-the-art to combinational equivalence checking is based on SAT sweeping. It recursively establishes equivalence of internal nodes of two circuits to prove equivalence of their outputs. The approach follows the topological order from inputs to outputs and makes use of simulation to refine the set of potentially equivalent nodes to reduce the number of SAT solver queries. This non-uniform hybrid reasoning, using both the circuit structure and a clausal encoding, is complex to orchestrate. In earlier work, clausal encoding was avoided using a dedicated circuit-aware SAT solver. Instead, we propose to perform SAT sweeping directly on the clausal encoding of the complete equivalence checking problem within the SAT solver, but again relying on a second, dedicated, internal SAT solver. Both SAT solvers work on a clausal representation. This allows to transparently make use of all the advanced reasoning capabilities of the SAT solver, particularly pre- and inprocessing techniques.**

***Index Terms*—Equivalence Checking, SAT Sweeping, Miters, Equivalence Reasoning, Conjunctive Normal Form, Backbones.**

I. Introduction

Hardware equivalence checking is considered one of the oldest and most successful industrial formal verification techniques. Its purpose is to formally prove that a synthesized circuit matches its golden model. While early approaches [1] relied on binary decision diagrams (BDDs), more recent approaches rely on SAT sweeping [2]. It is fair to assume that SAT sweeping is important in commercial equivalence checking tools too.

The state-of-the-art [3]–[8] uses a *hybrid approach* to detect equivalent literals through SAT sweeping. It follows the topological structure of the two compared circuits and uses incremental queries to the SAT solver as well as dedicated SAT solving engines which can be made aware of the circuit structure too [4]–[6]. It can also focus the SAT solving effort on small parts of the circuit, which avoids the overhead in having the SAT solver deal with the full combined problem.

This hybrid approach is in contrast to a *monolithic approach*, advertised in this paper, in which the equivalence checking problem (the miter [9]) is translated once as a whole into a clausal representation in conjunctive normal form (CNF) and then simply given to a SAT solver. The advantage of the monolithic approach is that it can easily make use of sophisticated preprocessing [10] and inprocessing [11] techniques implemented in modern SAT solvers.

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

In this paper we combine the benefits of both approaches by using within the main SAT solver (KISSAT) a second embedded simple SAT solver (KITTEN) for SAT sweeping directly on CNF. This not only improves monolithic solving of miters substantially but also reduces solving time on other formulas in CNF for which no circuit structure is available.

Hybrid SAT sweeping relies on *structural hashing* to remove isomorphic parts of the miter. For instance, when comparing two identical copies of the same circuit, structural hashing alone can prove equivalence. In the monolithic approach this is much harder, at least for plain CDCL solving [10], which empirically fails on such isomorphic miters [12], [13].

Our recent work [14] on *clausal congruence closure* allows to simulate structural hashing on the CNF level. It relies on gate extraction and succeeds in solving such simple isomorphic miters instantly. Alone it falls far behind hybrid approaches on more practically relevant miters checking equivalence of optimized (synthesized) and original (golden) circuits, unless it is combined with *clausal equivalence sweeping*, presented in this paper. This monolithic sweeping approach has not been described nor evaluated in the literature before, except briefly being mentioned in system descriptions of KISSAT in SAT competition proceedings [15], [16]. For more related work from the SAT and CP community see [14], [17], [18].

II. Preliminaries

We assume the reader is familiar with standard notations in SAT, i.e., we work with formulas F in conjunctive normal form (CNF), usually denoted as a set $F = \{C_1, \ldots, C_m\}$. Each clause C is a set of literals $C = \{l_1, \ldots, l_n\}$, with each literal l being a variable v or its negation $\bar{v}$. We also use logical notation $F = C_1 \wedge \cdots \wedge C_m$ and $C = l_1 \vee \cdots \vee l_n$ as well as logical negation $\neg v = \bar{v}$ and assume $\neg\neg l = l$. Besides variables we also use the Boolean constants $\mathbb{B} = \{0, 1\}$.

The variables are taken from a fixed set $\mathcal{V}$, which we assume to be totally ordered via $\leq$. The variable v of a literal l is obtained as $|l| = v$, meaning $l = v$ or $l = \neg v$. The variable order yields a preorder on the set of all literals, denoted as $\mathcal{L}$, and the Boolean constants as follows: $l \leq l'$ iff $|l| \leq |l'|$ and $0, 1 \leq l$ for all $l, l' \in \mathcal{L}$ (note $l \leq \neg l$ and $\neg l \leq l$). We also use the irreflexive version $<$, where additionally $|l| \neq |l'|$.

With $\mathcal{V}(C) = \{|l| \mid l \in C\}$ and $\mathcal{V}(F) = \{\mathcal{V}(C) \mid C \in F\}$ we denote the set of variables of a clause and a formula and similarly for $\mathcal{L}(C)$ and $\mathcal{L}(F)$ for its literals. Let $|S|$ refer to the number of elements of a set S.

https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_29

full-sweeping (CNF G)

1 literal representative $\rho\colon \mathcal{L} \to \mathcal{L} \cup \mathbb{B}$ with $\rho(l) = l$
2 **if** G is *unsatisfiable* **return** $\lambda l.(l \neq |l|)$ // map $v \mapsto 0$
3 pick initial assignment σ with $\sigma(G) = 1$
4 backbone candidates $B \leftarrow \{l \in \mathcal{L}(G) \mid \sigma(l) = 1\}$
5 equivalent literals partition $P \leftarrow \{B\}$
6 **while** $B \neq \emptyset$
7 pick $l \in B$ and set $B \leftarrow B \backslash \{l\}$
8 **if** exists model σ with $\sigma(G \wedge \neg l) = 1$ // SAT call
9 $B \leftarrow \{l \in B \mid \sigma(l) = 1\}$ // refine backbone
10 $P \leftarrow$ *refine* (P, σ) // refine partition
11 **else**
12 $\rho \leftarrow$ *propagate* $(G, \rho \circ \{l \mapsto 1\} \circ \{\neg l \mapsto 0\})$
13 remove from $L \in P$ all $l \in L$ with $\rho(l) \in \mathbb{B}$
14 $G \leftarrow \rho(G)$
15 **while** exists literal class $L \in P$ with $|L| > 1$
16 pick $k, l \in L$ with $k < l$
17 **if** exists σ with $\sigma(G \wedge l \wedge \neg k) = 1$ or // SAT call
18 $\sigma(G \wedge \neg l \wedge k) = 1$ // SAT call
19 $P \leftarrow$ *refine* (P, σ) // refine partition
20 **else**
21 $\rho \leftarrow \rho \circ \{l \mapsto k\} \circ \{\neg l \mapsto \neg k\}$
22 remove l from L in P
23 **return** ρ

Fig. 1: Pseudo code of our full SAT sweeping routine, which in practice is only applied to variable environments (*cf.* Fig.4/5). We use the '$\circ$' operator to denote function composition.

An assignment $\sigma\colon \mathcal{V} \to \mathbb{B}$ is extended to literals, clauses and formulas by applying Boolean simplification. A formula F is *satisfiable* if there is an assignment σ with $\sigma(F) = 1$, also called *satisfying assignment* or *model*. Let $\bot = \{\emptyset\}$ denote the unsatisfiable CNF consisting of the empty clause $\emptyset$. Given a satisfiable formula F, a literal l is a *backbone* of F if $\sigma(l) = 1$ for all models σ of F. This can be checked by showing that $F \wedge \neg l$ is unsatisfiable. Two literals k and l are *equivalent* if $\sigma(k) = \sigma(l)$ in all models σ of F. This can be checked by showing that $F \wedge l \wedge \neg k$ is unsatisfiable as well as $F \wedge \neg l \wedge k$.

III. Algorithm

Our unbounded algorithm for *full-sweeping* is shown in Fig. 1. It returns a *mapping* ρ of the literals L of the given formula to literals or to Boolean constants $\mathbb{B} = \{0, 1\}$. We further assume $\rho(\neg l) = \neg\rho(l)$ and $\rho(l) \leq l$ as it is common in this kind of union-find data-structure.

The entire sweeping algorithm has three phases. First, lines 1–5, it tries to find a satisfying assignment σ. If none exists, an arbitrary constant mapping ρ is returned, i.e., $\rho(v) = 0$ for all $v \in \mathcal{V}$, guaranteed to falsify at least one clause. Otherwise, σ is used to determine the backbone candidates B (literals set to true) and an initial equivalence class of literals also all set to true. Note that thereby negations of these literals are also considered potentially equivalent with each other.

refine (P, σ)

1 $R \leftarrow \emptyset$
2 **for** all $L \in P$
3 $L_i \leftarrow \{l \in L \mid \sigma(l) = i\}$ for $i \in \mathbb{B}$
4 **if** $L_0 = \emptyset$ or $L_1 = \emptyset$ **then** $R \leftarrow R \cup \{L\}$
5 **else** $R \leftarrow R \cup \{L_0\} \cup \{L_1\}$
6 **return** R

Fig. 2: Refinement of equivalent literal partition.

propagate (CNF F, literal mapping $\rho\colon \mathcal{L} \to \mathcal{L} \cup \mathbb{B}$)

1 $F \leftarrow \rho(F)$ // pre-simplify
2 **while** $\emptyset \notin F$ and there is a unit clause $\{l\} \in F$
3 $\rho \leftarrow \rho \circ \{l \mapsto 1\} \circ \{\neg l \mapsto 0\}$
4 $F \leftarrow \rho(F)$ // simplify
5 **return** ρ

Fig. 3: Unit propagation on a literal mapping.

In the second phase, lines 6–14, each remaining l in the backbone candidate set B is checked to have a model of the formula falsifying l. If such a model exists, we remove all falsified literals from B in that model and refine the equivalence classes in the partitioning according to Fig. 2 by splitting classes inconsistent with that model into one class of literals assigned to 0 and one class of literals assigned to 1. Otherwise, there is no model of the formula which sets the considered backbone candidate l to 0. So we update ρ by setting l permanently to 1 and $\neg l$ to 0 and *propagate* this information over the formula, as shown in Fig. 3, which might deduce additional constant assignments. Afterwards, the formula is simplified (line 14) by applying the updated mapping.

In the third phase, lines 15–22, after backbone extraction, we check for each pair of remaining equivalent literals candidates, within the same equivalence class, whether there exists a model of the formula with the two literals assigned to different values. If this is the case we split their equivalence class as well as all other equivalence classes which are inconsistent with the model. Otherwise, we have shown that these two literals are equivalent and record that information by mapping the larger literal to the smaller (and accordingly their negations).

This procedure calls a SAT oracle in three places and as such is not really useful to simplify a formula for which we only want to know whether it is satisfiable. Thus in order to use this sweeping procedure in the context of SAT solving, we have to limit the effort put into these SAT calls. There are two obvious ways to achieve this. Either we replace the oracle calls by some cheaper procedure to limit the run-time of the oracle or we apply full sweeping only to a subset of literals.

We have explored the first option before in LINGELING [19] and SPLATZ [20] without much success though. Therefore, KISSAT uses the second approach, shown in Fig. 4. As in hybrid approaches [4]–[6], we focus each full-sweeping only on a small part of the formula, assuming that the cheap-to-detect equivalences are between literals close to each other.

bounded-sweeping (CNF F, bound $k \in \mathbb{N}$)

```
1  working set Γ ← V(F)     // all variables in F
2  while Γ ≠ ∅
3      pick v ∈ Γ and set Γ ← Γ\{v}
4      G ← environment ({v}, ∅, F, k)
5      ρ ← full-sweeping (G)     // sweep environment clauses
6      ρ ← propagate (F, ρ)      // propagate ρ on whole F
7      F' ← ρ(F)                 // simplify F with ρ
8      if ∅ ∈ F' return ⊥  // return CNF with empty clause
9      Γ ← Γ ∪ V(F'\F)  // add variables in changed clauses
10     F ← F'
11 return F
```

Fig. 4: Pseudo code of our bounded SAT sweeping routine.

environment (variables V, CNF G, CNF F, bound $k \in \mathbb{N}$)

```
1  if k = 0 return G
2  G' ← {C ∈ F | V ∩ V(C) ≠ ∅}     // clauses with V
3  V' ← ⋃ V(G')                    // variables in clauses with V
4  return environment (V', G', F, k − 1)
```

Fig. 5: Compute bounded environment of a variable.

To this extent we consider two variables (and thus their literals) to be "very close" if they occur in the same clause and extend this notion recursively in a breadth-first manner limited by some bound k, i.e., the distance between two variables. We further decided to restrict the part of a formula to which full sweeping is applied to consist of all clauses containing variables a maximum distance away from a given variable v, i.e., the *environment* of v as shown in Fig. 5.

The *bounded-sweeping* algorithm goes over all variables v of the formula and performs full sweeping on the environment of v. The whole formula is then simplified by applying the mapping ρ obtained from the full sweeping of the environment (line 7). All variables in clauses that changed during that simplification are reconsidered (line 9). This approach is sound as both local backbones and equivalences of a sub-formula are of course also backbones and equivalences of the whole formula. It is obviously not complete but gives substantial improvements in practice, as our experiments will show.

IV. Implementation

The use of a dedicated light-weight SAT solver in hybrid approaches (*cf.* [6]) provided the motivation to explore using a separate light-weight *little* SAT solver (KITTEN) within our full-blown state-of-the-art *big* SAT solver (KISSAT). This allows to (i) solve only parts of a big formula by copying it, and (ii) avoids any other interaction of solving the small problems such as keeping statistics, variable scores etc. untouched, and (iii) allows to record proofs in memory in case it becomes necessary to export proofs from the small to the big solver, without the need to support this feature in the big solver.

Although clausal equivalence sweeping, presented in this paper, was the first application of KITTEN inside KISSAT, it was also used to mine definitions [21], [22], and to improve bounded variable elimination [23]. The article on definition-mining [21] contains implementation details about KITTEN.

Most of the time SAT calls during sweeping in Fig. 1 are of course satisfiable as otherwise the formula would radically simplify. Even though often trivial to solve (few or no conflicts), these satisfiable queries are relatively expensive, as a full model of the environment has to be constructed, i.e., at least linear in the number of variables in the environment. Motivated by the usefulness of model rotation in MUS extraction [24], we added an API call "`kitten_flip_literal`" to KITTEN, which checks after a model has been found, whether the value of a single literal can be flipped, without falsifying any clause.

Flipping can be implemented efficiently by traversing only the clauses watched by that literal, an insight which helped to improve stand-alone backbone extraction [25] (after porting it to CADICAL). Without using watches, model rotation appears to be too costly [26]. In our implementation of clausal equivalence sweeping, we aggressively use literal flipping whenever we find a model (at line 9 and 19 in Fig. 1) to refine both backbone candidates and the equivalent literal classes, i.e., any backbone candidate and any literal in an equivalence literal class can be removed if it can be flipped. This technique reduces the number of full satisfiable queries substantially.

Despite a small bound of distance $k = 3$ (which actually starts at $k = 2$ and only is increased to $k = 3$ in the next inprocessing round after successful completion of sweeping), we also limit the number of variables ($2^{13} = 8192$) and clauses ($2^{15} = 32768$) allowed in an environment. Still, also in contrast to clausal congruence closure [14], clausal equivalence sweeping is too costly to run until completion on larger instances. Therefore we limit the effort (time spent in KITTEN measured in "ticks" – an approximation of cache line accesses) relative to the time spent during CDCL, as with other inprocessing procedures, preempt sweeping and continue later with the remaining variables in the next inprocessing round.

V. Benchmarks

We evaluated our approach on problems of the hardware model checking competition (HWMCC) from 2012 [27] and 2020 [28] and from the SAT competition 2022 and 2023. The AIGER [29] problems from HWMCC are encoded into CNF based on detecting and encoding XOR and ITE gates in an optimized way instead of the default AND gate encoding.

Moreover, each miter comes in two versions: iso and opt. The former (iso) compares each circuit with an *isomorphic* copy of itself, while the latter set (opt) uses the command `dc2` of ABC to *optimize* the original circuit and then compares this optimized circuit with the original circuit [30]–[32].

Evaluating our technique on SAT competition benchmarks allows us to assess the potential overhead and benefits of our approach on more general SAT instances that may have less underlying structure which can be exploited by our technique.

VI. Experiments

We implemented our approach in KISSAT and evaluated its performance on the bwForCluster Helix, utilizing AMD Milan

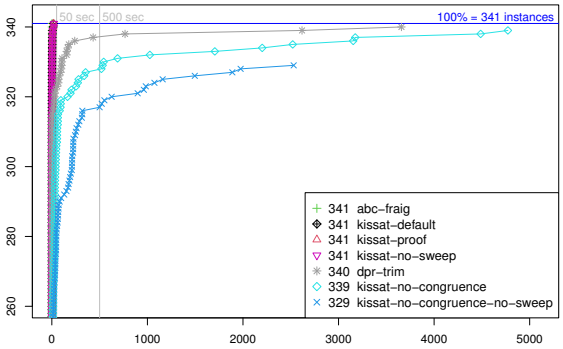

Fig. 6: Number of solved **isomorphic** miters (y-axis) from 341 HWMCC'12 benchmarks in the given time (x-axis in seconds). The legend displays the number of solved instances per solver.

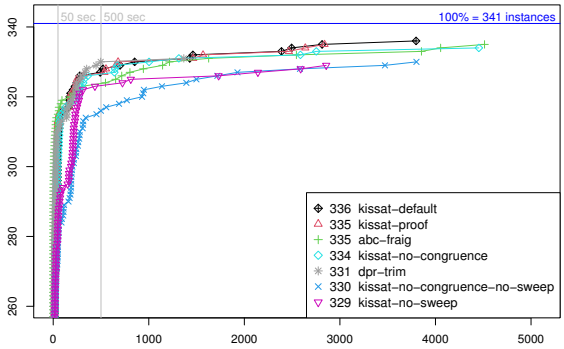

Fig. 7: Number of solved **optimized** miters (y-axis) from 341 HWMCC'12 benchmarks in the given time (x-axis in seconds). The legend displays the number of solved instances per solver.

EPYC 7513 CPUs, with 15 GB of memory and 5000 second time limit. All plots follow the SAT competition set-up [33] showing the number of solved instances (y-axis) over the time it took to solve them (on the x-axis in seconds). Source code is available at [32] and experimental data at [32], [34].

In our experiments we compare the default configuration of KISSAT (kissat-default), where both SAT sweeping and clausal congruence closure [14] is enabled, to activating only one or none of these techniques. Additionally, we consider runs of the default configuration with proof production enabled (kissat-proof), and present here the required time to check these produced proofs using DPR-TRIM as well (dpr-trim).

Figures 6-9 depict the results of the experiments on the HWMCC problems. Here we compare our approach to the state-of-the-art SAT sweeping technique [6] implemented in ABC, available as `fraig -y` in ABC superseding `fraig -x` used in the `cec` command (according to personal communication with the author of ABC). The results show that our pure SAT-based approach, that sees only the clausal representation of the circuits is able to perform comparable to the hybrid approach specialized in reasoning about circuits. Regarding SAT competition problems, we follow [35] and include SBVA-CADICAL, winner of the SAT Competition 2023. The results in Fig. 10-14 demonstrate that *sweeping* and *congruence closure* both contribute to better performance on these more generic competition problems too. Fig. 13/14 also compare solving times versus checking times with DPR-TRIM.

In Fig. 12 we show results on 5 challenging miters from the IWLS'22 paper [6] (originating from [5]) which introduced the advanced SAT sweeping technique implemented in ABC (through the command "`fraig -y`") as also used in our experiments. Again sweeping gives a substantial improvement in our monolithic approach. Note that one miter "test02" can actually be solved by congruence closure instantly (faster than ABC) and does not need sweeping (*cf.* [14] for details).

We further extracted from the log files [32], [34] the following numbers. The time spent in clausal equivalence sweeping with kissat-default is for hwmcc12/opt on average 13.72 sec (0.00 sec - 636.67 sec) and 21.77% (3.26% - 80.98%),

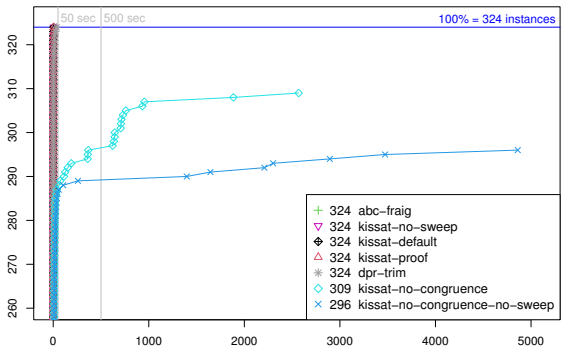

Fig. 8: Number of solved **isomorphic** miters (y-axis) from 324 HWMCC'20 benchmarks in the given time (x-axis in seconds). The legend displays the number of solved instances per solver.

for hwmcc20/opt on average 31.42 sec (0.00 sec - 636.67 sec) and 17.89% (3.26% - 80.98%), for iwls22 on average 64.35 sec (0.00 sec - 104.51 sec) and 9.81% (0.00% - 10.92%), for sc2022 on average 34.75 sec (0.04 sec - 681.70 sec) and 4.52% (0.13% - 29.70%), for sc2023 on average 29.65 sec (0.00 sec - 437.05 sec) and 5.87% (0.06% - 93.72%).

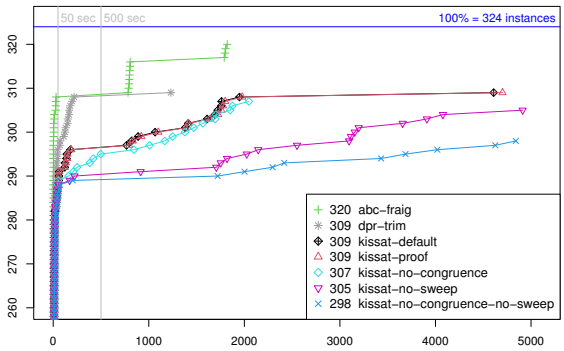

Fig. 9: Number of solved **optimized** miters (y-axis) from 324 HWMCC'20 benchmarks in the given time (x-axis in seconds). The legend displays the number of solved instances per solver.

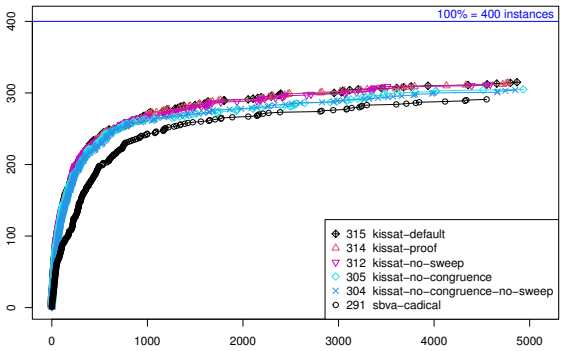

Fig. 10: Number of solved SAT Competition 2022 main track benchmarks (y-axis) in the given time (x-axis in seconds). The legend displays the number of solved instances per solver.

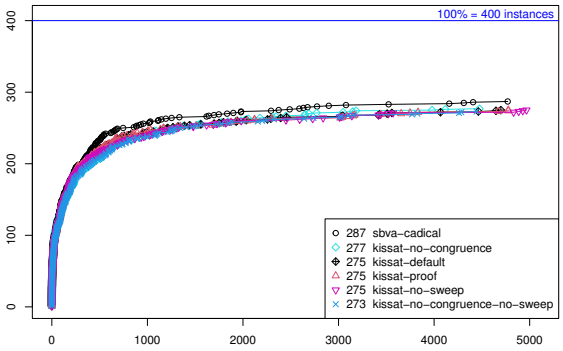

Fig. 11: Number of solved SAT Competition 2023 main track benchmarks (y-axis) in the given time (x-axis in seconds). The legend displays the number of solved instances per solver.

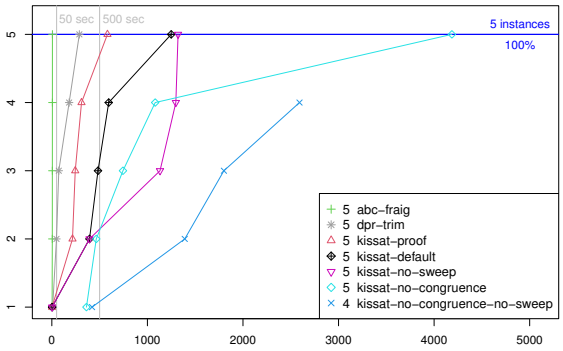

Fig. 12: Number of solved miters (on the y-axis) of the 5 IWLS'22 benchmarks in the given time (on the x-axis in seconds). The legend displays the number of solved instances per solver. One of the miters, i.e., test02, is instantly solved by KISSAT with clausal congruence closure even though it was considered challenging in [6]. This is due normalization during ITE gates extraction. See [14] for a more detailed explanation.

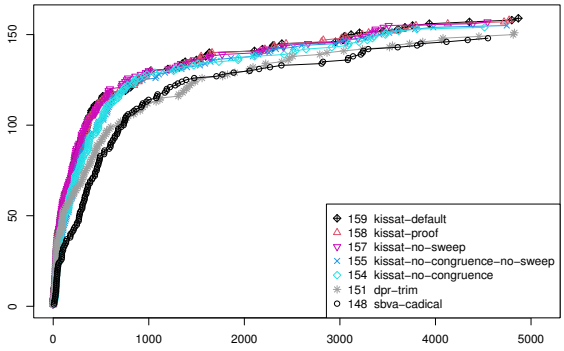

Fig. 13: Number of solved **unsatisfiable** SAT Competition 2022 main track benchmarks (y-axis) in the given time (x-axis in seconds). The total number of unsatisfiable instances is unknown though. The legend displays the number of solved instances per solver.

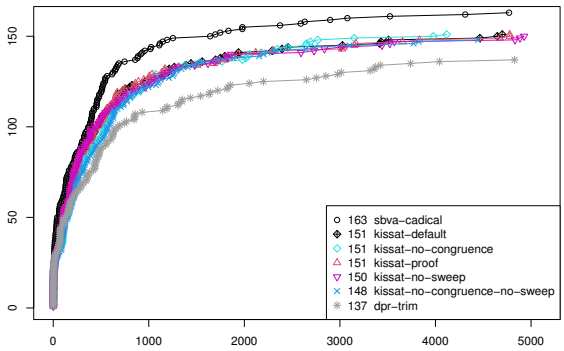

Fig. 14: Number of solved **unsatisfiable** SAT Competition 2023 benchmarks (y-axis) in the given time (x-axis in seconds). The total number of unsatisfiable instances is unknown though. The legend displays the number of solved instances per solver.

The number of backbones and equivalences found were for hwmcc12/opt 70 780 backbones and 446 784 equivalences, for hwmcc20/opt 12 976 backbones and 162 427 equivalences, for iwls22 2 052 backbones and 58 792 equivalences, for sc2022 298 065 backbones and 1 590 810 equivalences, for sc2023 838 120 backbones and 4 019 861 equivalences.

The percentage of satisfiable queries was for hwmcc12/opt 91.45%, for hwmcc20/opt 95.27%, for iwls22 94.25%, for sc2022 95.64%, for sc2023 96.00% and the ratio of successfully flipped literals over the number of satisfiable queries was for hwmcc12/opt 12.77, for hwmcc20/opt 32.34, for iwls22 21.03, for sc2022 64.05, for sc2023 23.34.

VII. CONCLUSION

We presented a "big-little" approach to clausal equivalence sweeping directly on CNF using an embedded SAT solver KITTEN inside of KISSAT and show that it improves solving hard equivalence checking problems substantially as well as being useful on plain CNF problems from the SAT competition.

REFERENCES

[1] A. Kuehlmann and F. Krohm, "Equivalence checking using cuts and heaps," in *Proceedings of the 34st Conference on Design Automation, Anaheim, California, USA, Anaheim Convention Center, June 9-13, 1997*, E. J. Yoffa, G. D. Micheli, and J. M. Rabaey, Eds. ACM Press, 1997, pp. 263–268.

[2] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, 2002.

[3] V. N. Possani, A. Mishchenko, R. P. Ribas, and A. I. Reis, "Parallel combinational equivalence checking," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 3081–3092, 2020.

[4] L. G. Amarù, F. S. Marranghello, E. Testa, C. Casares, V. N. Possani, J. Luo, P. Vuillod, A. Mishchenko, and G. D. Micheli, "SAT-sweeping enhanced for logic synthesis," in *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*. IEEE, 2020, pp. 1–6.

[5] H. Zhang, J. R. Jiang, L. G. Amarù, A. Mishchenko, and R. 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*. IEEE, 2021, pp. 877–882.

[6] H. Zhang, J. R. Jiang, A. Mishchenko, and L. G. Amarù, "Improved large-scale SAT sweeping," in *Proc. 31st International Workshop on Logic & Synthesis*, 2022.

[7] Z. Chen, X. Zhang, Y. Qian, Q. Xu, and S. Cai, "Integrating exact simulation into sweeping for datapath combinational equivalence checking," in *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*. IEEE, 2023, pp. 1–9.

[8] H. Pan, R. Zhang, Y. Xia, L. Wang, F. Yang, X. Zeng, and Z. Chu, "A semi-tensor product based circuit simulation for sat-sweeping," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2024, Valencia, Spain, March 25-27, 2024*. IEEE, 2024, pp. 1–6.

[9] D. Brand, "Verification of large synthesized designs," in *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, 1993, Santa Clara, California, USA, November 7-11, 1993*, M. R. Lightner and J. A. G. Jess, Eds. IEEE Computer Society / ACM, 1993, pp. 534–537.

[10] A. Biere, M. Järvisalo, and B. Kiesl, "Preprocessing in SAT solving," in *Handbook of Satisfiability - Second Edition*, ser. Frontiers in Artificial Intelligence and Applications, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2021, vol. 336, pp. 391–435.

[11] M. Järvisalo, M. Heule, and A. Biere, "Inprocessing rules," in *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, ser. Lecture Notes in Computer Science, B. Gramlich, D. Miller, and U. Sattler, Eds., vol. 7364. Springer, 2012, pp. 355–370.

[12] A. Biere, M. Heule, M. Järvisalo, and N. Manthey, "Equivalence checking of HWMCC 2012 circuits," in *Proc. of SAT Competition 2013 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, A. Balint, A. Belov, M. Heule, and M. Järvisalo, Eds., vol. B-2013-1. University of Helsinki, 2013, p. 104.

[13] M. Heule, M. Järvisalo, and A. Biere, "Revisiting hyper binary resolution," in *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*, ser. Lecture Notes in Computer Science, C. P. Gomes and M. Sellmann, Eds., vol. 7874. Springer, 2013, pp. 77–93.

[14] A. Biere, K. Fazekas, M. Fleury, and N. Froleyks, "Clausal Congruence Closure," in *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*, ser. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[15] A. Biere, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba entering the SAT Competition 2021," in *Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2021-1. University of Helsinki, 2021, pp. 10–13.

[16] A. Biere and M. Fleury, "Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022," in *Proc. of SAT Competition 2022 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2022-1. University of Helsinki, 2022, pp. 10–11.

[17] M. Heule and A. Biere, "Blocked clause decomposition," in *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, ser. Lecture Notes in Computer Science, K. L. McMillan, A. Middeldorp, and A. Voronkov, Eds., vol. 8312. Springer, 2013, pp. 423–438.

[18] M. Codish, Y. Fekete, and A. Metodi, "Backbones for equality," in *Hardware and Software: Verification and Testing - 9th International Haifa Verification Conference, HVC 2013, Haifa, Israel, November 5-7, 2013, Proceedings*, ser. Lecture Notes in Computer Science, V. Bertacco and A. Legay, Eds., vol. 8244. Springer, 2013, pp. 1–14.

[19] A. Biere, "Lingeling and friends entering the SAT Race 2015," Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 15/2, 2015.

[20] ——, "Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016," in *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, ser. Department of Computer Science Series of Publications B, T. Balyo, M. Heule, and M. Järvisalo, Eds., vol. B-2016-1. University of Helsinki, 2016, pp. 44–45.

[21] M. Fleury and A. Biere, "Mining definitions in Kissat with Kittens," *Formal Methods Syst. Des.*, vol. 60, no. 3, pp. 381–404, 2022.

[22] J. E. Reeves, M. J. H. Heule, and R. E. Bryant, "Moving definition variables in quantified boolean formulas," in *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, D. Fisman and G. Rosu, Eds., vol. 13243. Springer, 2022, pp. 462–479.

[23] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," in *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, ser. Lecture Notes in Computer Science, F. Bacchus and T. Walsh, Eds., vol. 3569. Springer, 2005, pp. 61–75.

[24] A. Belov and J. Marques-Silva, "Accelerating MUS extraction with recursive model rotation," in *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, P. Bjesse and A. Slobodová, Eds. FMCAD Inc., 2011, pp. 37–40.

[25] A. Biere, N. Froleyks, and W. Wang, "Cadiback: Extracting backbones with cadical," in *26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy*, ser. LIPIcs, M. Mahajan and F. Slivovsky, Eds., vol. 271. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 3:1–3:12.

[26] M. Janota, I. Lynce, and J. Marques-Silva, "Algorithms for computing backbones of propositional formulae," *AI Commun.*, vol. 28, no. 2, pp. 161–177, 2015.

[27] A. Biere, K. Heljanko, M. Seidl, and S. Wieringa, "Hardware model checking competition (hwmcc'12)," 2012. [Online]. Available: https://fmv.jku.at/hwmcc12

[28] A. Biere, N. Froleyks, and M. Preiner, "Hardware model checking competition (hwmcc'20)," 2020. [Online]. Available: https://hwmcc.github.io/2020

[29] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 11/2, 2011.

[30] A. Biere, K. Fazekas, M. Fleury, and N. Froleyks, "CNF Encoded Isomorphic and Optimized Miters from Hardware Model Checking Competition 2020 Models," May 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11202461

[31] A. Biere, "CNF Encoded Isomorphic and Optimized Miters from Hardware Model Checking Competition 2012 Models," Mar. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.10823128

[32] [Online]. Available: https://cca.informatik.uni-freiburg.de/ces

[33] N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, "SAT Competition 2020," *Artif. Intell.*, vol. 301, p. 103572, 2021.

[34] A. Biere, K. Fazekas, M. Fleury, and N. Froleyks, "Clausal equivalence sweeping paper logs," May 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11203283

[35] A. Biere, M. Järvisalo, D. Le Berre, K. S. Meel, and S. Mengel, "The SAT practitioner's manifesto," Sep. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4500928