# Hardware Model Checking Certification with Certifaiger and Cerbtora

Nils Froleyks[1] and Emily Yu[2]

[1] KU Leuven, Belgium
[2] Leiden University, Netherlands

**Abstract.** Certificates are machine-checkable witnesses that help increase confidence in verification results by providing independently verifiable evidence beyond a simple yes/no answer. In this short paper, we present two certificate checkers for hardware model checking, Certifaiger and Cerbtora, which target bit-level and word-level verification of hardware designs, respectively. Certifaiger has been adopted in recent editions of the Hardware Model Checking Competition, but not described in the literature before. Cerbtora extends the same theoretical framework to the word level, in which certificates are expressed in the same modeling language as the design under test and are validated using efficient automated reasoning engines. We describe the architecture and main components of both tools and evaluate them on competition benchmarks.

## 1 Introduction

Certification has gained increasing attention in automated reasoning and formal verification as a way to provide an additional layer of trust in the correctness of safety-critical systems beyond a simple yes/no answer. The concept of certification spans several areas, such as Boolean satisfiability (SAT) solving [15, 17], satisfiability modulo theories (SMT) [16, 21], and software verification [2]. In hardware model checking, certification has also advanced recently. In particular, the bit-level track of the Hardware Model Checking Competition (HWMCC) now requires participating tools to generate certificates, with encouraging results [14]. In this setting, the certificates take the form of *witness circuits* produced at the end of the verification process. The certificate format is designed to accommodate a wide range of model checking techniques [13, 25–27]. Validating a witness circuit is lightweight and efficient, reducing to a small number of SAT checks.

So far, most effort has focused on certifying safety properties in bit-level model checking (i.e., reasoning over Boolean variables). In contrast, certification support for *word-level* verification (i.e., reasoning over machine words and instructions) remains largely unexplored. Word-level methods model hardware systems at a higher level of abstraction, capturing arithmetic and datapath components such as adders and multipliers rather than individual wires and primitive gates, and are essential for scalable hardware verification [3, 10]. A key goal of the HWMCC is to make word-level certification mandatory, thereby promoting the development of certifying word-level model checkers; achieving this requires

a dedicated word-level certificate checker. At present, the ecosystem of certifying word-level model checkers is still emerging, and there is no publicly available implementation that produces certificates. To exercise the end-to-end workflow, in this short paper, we generate word-level certificates using a prototype certifying $k$-induction engine, one of the essential model checking techniques. By providing a common validation backend, our checker aims to lower the barrier for tool developers and help catalyze broader community adoption, mirroring the progress achieved at the bit level.

We present Certifaiger and Cerbtora (*tora* means "tiger" in Japanese), two certificate checkers that implement a common framework [13,25–27] for validating witness circuits at the bit level and word level, respectively. Certifaiger (available at https://github.com/Froleyks/certifaiger) has been used in the Hardware Model Checking Competition since 2024 [14], but has not previously been described in the literature. Cerbtora (available at https://github.com/Froleyks/cerbtora) is a new C++ tool that uses the SMT solver Bitwuzla [18] as its backend. Both tools are integrated into a complete certification workflow (cf. Sec. 3) and are released under the MIT license on GitHub.

## 2 Model Checking Certificates

In this section, we briefly describe the certificate format. Our certificate checkers accept two circuits as input: (i) the model circuit $M$ under verification and (ii) a witness circuit $M'$ that certifies correctness, produced by a model checker. A circuit $M = (I, L, R, F, P, C)$ has inputs $I$, latches $L$, reset and transition functions $R, F$, safety property $P$, and constraint $C$; it is safe if every state reachable from a $C$-respecting reset through $C$-respecting transitions satisfies $P$. We use the following symbolic formulas over input and latch variables: for any $S \subseteq L$, $R\{S\}$ denotes the reset condition for the latches in $S$ (each latch equals its reset value), and $F_{0,1}\{S\}$ is the one-step transition relation for $S$, using time-stamped copies of $I$ and $L$. For any formula $\varphi$ over $I \cup L$, $\varphi_t$ denotes its time-shifted copy (replace each latch $l \in L$ by $l_t$ and each input $i \in I$ by $i_t$). Reset functions $R$ are *stratified* [27] if their latch-dependency graph is acyclic.

**Definition 1 (Witness circuit).** *Given a circuit $M$, the circuit $M'$, with $M = (I, L, R, F, P, C)$, $M' = (I', L', R', F', P', C')$, and $K = L \cap L'$, is a witness circuit if $R'$ is stratified and the following conditions hold:*

| | |
|---|---|
| *Reset* | $R\{K\} \wedge C \Rightarrow R'\{K\} \wedge C'$ |
| | *An initial state in the model implies an initial state in the witness.* |
| *Transition* | $F_{0,1}\{K\} \wedge C_0 \wedge C_1 \wedge C'_0 \Rightarrow F'_{0,1}\{K\} \wedge C'_1$ |
| | *A transition in the model implies a transition in the witness.* |
| *Property* | $(C \wedge C') \Rightarrow (P' \Rightarrow P)$ |
| | *A violation in the model implies a violation in the witness.* |
| *Base* | $R'\{L'\} \wedge C' \Rightarrow P'$ |
| | *The witness property holds in all initial states.* |
| *Step* | $P'_0 \wedge F'_{0,1}\{L'\} \wedge C'_0 \wedge C'_1 \Rightarrow P'_1$ |
| | *The witness property is inductive.* |

`model.aag`

```
aag 10 0 3 1 7
2 3
4 13
6 19
20
8 4 3
10 5 2
12 11 9
14 4 2
16 6 4
18 17 15
20 16 14
```

`witness.aag`

```
aag 9 1 2 1 6
2
4 13
6 19
16
8 4 3
10 5 2
12 11 9
14 4 2
16 6 4
18 17 15
i0 = 2
l0 = 4
l1 = 6
c
MAPPING 2
2 2
4 4
6 6
```

`model.btor2`

```
1 sort bitvec 1
2 sort bitvec 3
3 constd 2 0
4 constd 2 1
5 constd 2 -3
6 constd 2 4
7 constd 2 5
8 constd 2 7
9 state 2
10 init 2 9 3
11 eq 1 9 6
12 eq 1 9 7
13 eq 1 9 8
14 or 1 11 12
15 or 1 13 14
16 ite 2 15 5 4
17 add 2 9 16
18 next 2 9 17
19 bad 13
```

`witness.btor2`

```
1 sort bitvec 1
2 sort bitvec 3
3 constd 2 0
4 constd 2 1
5 constd 2 -3
6 constd 2 4
7 constd 2 5
8 constd 2 7
9 state 2
10 init 2 9 3
11 eq 1 9 6
12 eq 1 9 7
13 eq 1 9 8
14 or 1 11 12
15 or 1 13 14
16 ite 2 15 5 4
17 add 2 9 16
18 next 2 9 17
19 constd 2 6
20 eq 1 9 19
21 or 1 13 20
22 bad 21
```

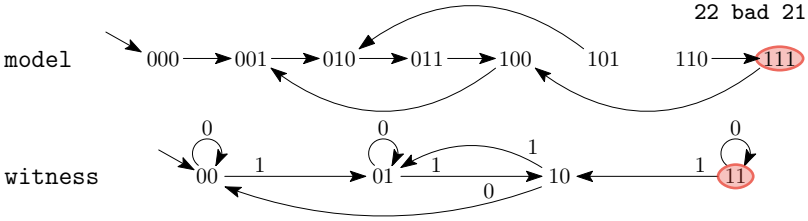


Fig. 1: A binary counter that increments by 1 or subtracts 3. The model contains three latches to encode values from 0 to 7, and no inputs. Its state space is depicted with the bad state 111 marked (the safety property enforces that the bad state is not reachable). The state space of the bit-level witness circuit with one input and two latches is shown at the bottom. We describe the model in both AIGER (`model.aag`) and Btor2 (`model.btor2`). Additionally, two witnesses, one in AIGER (`witness.aag`) and one in Btor2 (`witness.btor2`), are presented. The AIGER witness requires an explicit mapping (cf. Sec. 3) to the inputs and latches in the model. In this example, the mapping is (redundantly) specified in the symbol table and in the comment section.

The five conditions *Reset*, *Transition*, *Property*, *Base*, *Step* are encoded as SAT formulas whose validity certifies the correctness of the model checking result. Additionally, the witness circuit must be *stratified*. This is checked explicitly in Certifaiger, while at the word level it is syntactically enforced by the BTOR2 format [19]. A soundness proof of Def. 1 is provided in [14]. Fig. 1 illustrates the same model and witness circuits in both bit-level and word-level formats.

## 3 Certifaiger and Cerbtora

Certifaiger is a C++ application that takes as input a model and its witness in the AIGER [7] format and generates five CNF files, each corresponding to the negation of one check in Def. 1. These CNF files are solved by a SAT solver (Kissat [12] or CaDiCaL [4]). Our tool uses the AIGER library parser [8], constructs two copies of each circuit, encodes $R_0, C_0, C_1, \ldots$, and builds one output per check. The pipeline then applies cone-of-influence reduction and a Tseitin transformation [24] via `aigsplit` and `aigtocnf` to obtain the five SAT formulas. If all checks are proved unsatisfiable, the witness is successfully validated.

Cerbtora follows a similar workflow and is also implemented in C++. At the word level, models are given in the Btor2 [19] format; Cerbtora uses the Btor2Tools parser [9] and generates the five checks as combinational Btor2 files. These files are then passed to the SMT solver Bitwuzla [18]. If all five checks are unsatisfiable, the witness is validated and the model property is established.

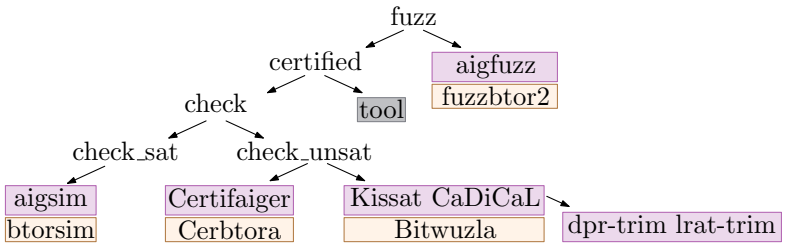


Fig. 2: Simulators, SAT/SMT solvers, SAT proof checkers, and fuzzers included.

*Mapping.* In Def. 1, the sets of inputs $I$ and latches $L$ of the model and the witness circuit have some intersection to connect them. In principle, we could rely on naming literals identically in both circuits; however, that causes gaps in literal numbering, which is incompatible with the binary format of AIGER [7]. We therefore allow the user to explicitly specify a literal *mapping*. The witness can define such a mapping in multiple ways, in decreasing order of precedence:

1. A comment `MAPPING <n>` followed by $n$ lines of the form `<witness literal> <model literal>`;
2. symbol-table entries [7] of the form `=<model literal>`;
3. if no mapping is given, Certifaiger assumes a mapping between the first $n$ inputs and first $m$ latches in the model and witness, respectively.

*Integration.* Both Certifaiger and Cerbtora include applications that integrate a number of external tools. These applications and tools are automatically downloaded and built with the `-DTOOL` option. Fig. 2 gives an overview.

**check** `<model> <witness>` identifies whether the `witness` describes a simulation trace (SAT) or a witness circuit (UNSAT) and checks its correctness with the appropriate tool: `aigsim` [8] or `btorsim` [9] to simulate a trace, and Certifaiger

Table 1: Comparison of formula generation and SAT solver configurations.

| | Verified | PAR10[sec] | Max[sec] |
|---|---|---|---|
| **Kissat** | | | |
| **Certifaiger (parallel + unsat bias)** | 1059 | 180.77 | 3577.93 |
| parallel | 1056 | 276.35 | 3155.87 |
| split | 1056 | 277.18 | 2935.26 |
| merged | 1022 | 1530.25 | 2469.53 |
| **CaDiCaL** | | | |
| CaDiCaL | 1057 | 189.15 | 1957.92 |
| **Kissat Features** | | | |
| no-sweep | 1052 | 412.88 | 3529.78 |
| no-congruence | 1027 | 1374.67 | 3484.84 |
| no-congruence-no-sweep | 992 | 2515.47 | 3321.83 |
| **UNSAT Proofs** | | | |
| CaDiCaL + LRAT | 1053 | 320.44 | 1097.20 |
| Kissat + DRAT | 974 | 3298.02 | 3528.02 |

or Cerbtora to generate five formulas, which are checked for unsatisfiability. If Certifaiger is configured with `-DPROOF`, proofs of unsatisfiability are produced by the SAT solver and checked using `dpr-trim` [23] or `lrat-trim` [20].

**certified** `<tool> <model>` wraps the model checker provided by the user to generate and check a produced simulation trace or witness circuit. The tool must follow the HWMCC interface: `<tool> <model> <witness.sat> <witness.unsat>` where the latter are paths for the violation trace and witness circuits respectively.

**fuzz** `<tool>` repeatedly uses `aigfuzz` (or `fuzzbtor2`) to generate random input circuits, runs the provided model checker, and verifies the resulting certificates. The AIGER version additionally integrates the delta debugger `aigdd` [28], used to automatically shrink failure-inducing inputs. If the provided tool supports the `--range` option, which lists all integer options as `<name> <default> <min> <max>`, option fuzzing [29] and delta debugging are performed.

*Base of trust.* Certifaiger can reduce the number of trused components by checking UNSAT proofs. At the word level, Bitwuzla does not support proofs, and while CVC5 [1] can generate Alethe proofs [22], QF_BV is only partially covered, so the SMT solver remains in the trusted computing base. A pragmatic route to stronger assurance is to bit-blast each QF_BV check to CNF and validate LRAT proofs, together with proofs of the bit-blasting step. Alternatively, one can use a verified translation, such as `bv_decide` in Lean or CoqQFBV.

## 4 Evaluation

In this section, we empirically evaluate our certificate checkers. We use an HPC cluster with Intel Xeon Platinum 8360Y CPUs at 2.4 GHz, 14 GB of memory and one hour of wall time per certificate validation. For the bit-level experiments, we

Table 2: Comparison of bit-level and word-level certificate checks (in seconds), and inductive depth ($k$). The last row presents results over all 47 benchmarks.

| | k | | Reset | Transition | Base | Step | Property | Total |
|---|---|---|---|---|---|---|---|---|
| Problem-11 | 8 | bit | 0.01 | 1.30 | 0.63 | 180.81 | 0.62 | 183.38 |
| | | word | 0.20 | 0.43 | 0.39 | 103.23 | 0.40 | **104.66** |
| A19-p16 | 21 | bit | 0.02 | 0.18 | 0.29 | 310.07 | 0.23 | 310.80 |
| | | word | 0.33 | 0.70 | 0.86 | 23.33 | 2.93 | **28.15** |
| VexRiscv-p20-1 | 22 | bit | 0.01 | 0.05 | 0.13 | 143.51 | 0.10 | **143.80** |
| | | word | 0.18 | 0.34 | 0.38 | 610.68 | 1.85 | 613.43 |
| Dijkstra-1 | 34 | bit | 0.01 | 0.43 | 1.05 | 304.19 | 7.31 | 313.00 |
| | | word | 0.06 | 0.11 | 0.13 | 203.45 | 15.55 | **219.30** |
| ZipCPU-p10 | 97 | bit | 0.01 | 0.02 | 0.15 | 227.78 | 0.10 | 228.06 |
| | | word | 0.32 | 0.72 | 1.03 | 22.29 | 2.96 | **27.33** |
| ByteAdd-1 | 260 | bit | 0.01 | 0.02 | 0.41 | 655.08 | 0.28 | 655.81 |
| | | word | 0.34 | 0.68 | 0.90 | 44.88 | 6.03 | **52.83** |
| **All (47)** | | bit | **1.37** | 77.23 | **4.44** | 2009.81 | **10.34** | 2103.19 |
| | | word | 15.54 | **41.72** | 20.43 | **1488.18** | 50.61 | **1616.48** |

use the 1062 valid witness circuits produced during the HWMCC'24 [14]. Word-level certification is not yet required by the HWMCC, so we developed our own prototype $k$-induction engine to produce certificates for the same benchmark set. We expect the number of certifying word-level model checkers to grow substantially after the release of Cerbtora.

Table 1 presents several configurations of Certifaiger with different SAT backends. PAR10 presents the average runtime, where each timeout is counted as 10 hours. Across all configurations, we observe that formula generation time is negligible (mean 6 ms) compared to SAT solving. The configuration *split* validates Def. 1 via five sequential SAT checks. *merged* combines them into a single SAT instance, while *parallel* runs five instances of Kissat in parallel. *Certifaiger*, additionally configures the SAT solver to bias the search towards UNSAT. This configuration is used as the basis for the rest of the experiments (and the development of Cerbtora). Although CaDiCaL verified two fewer certificates, it was approximately 60% faster than Kissat on the remaining instances. Clausal congruence closure [5] and clausal equivalence sweeping [6] are both essential for witness circuit validation due to their ability to efficiently establish equivalences between the model and the witness. If we do not want to trust the SAT solver, Certifaiger can be configured to produce and check proofs of unsatisfiability for each SAT check. This is done using `dpr-trim` for Kissat's DRAT proofs and `lrat-trim` for CaDiCaL's LRAT proofs.

Certifaiger can handle multiple properties by considering their conjunction. To evaluate the performance implications of this, we implemented `aigmerge`, a tool that merges (witness) circuits by taking the union of inputs and latches, and concatenating their combinational structure. Table 3 considers the benchmark `cv32e40x` from YosysHQ. It originally featured 768 properties, of which 86 were

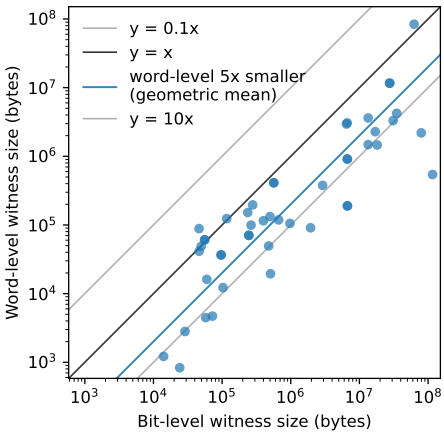


Fig. 3: Btor2 vs AIGER witnesses.

Table 3: Split vs multi-property. Times are measured in minutes. Merged compares verification time for merged circuits with reduandancy (plain) to the result of structural hashing (hash).

| **Verified** | | **Split**[m] | | **Merged**[m] | | |
|---|---|---|---|---|---|---|
| Model | Checker $n$ | total | mean | plain | hash | S |
| ric3 | 52 | 334 | 6.4 | **4.1** | 4.2 | 82 |
| ncip-minicraig | 26 | 134 | 5.0 | 4.7 | **4.0** | 34 |
| ncip-cadicraig | 26 | 158 | 5.9 | **5.2** | 5.4 | 30 |
| ncip-portfolio | 26 | 167 | 6.4 | 5.2 | **4.0** | 41 |
| supercar | 21 | 110 | 4.9 | **5.4** | 9.9 | 20 |

used in HWMCC'24. The table presents the number of properties verified by each model checker, the *total* time to validate all, and the *mean* time, and compares these with the single *merged* witness circuit that verifies all properties at once. We also removed redundancy using structural *hashing* in ABC [11]. While structural hashing is ineffective, the speedup $S$, which compares the total time for individual checks to the best merged check, shows vast gains in efficiency when witnesses are merged prior to checking.

*Word-level Reasoning.* Even though bit-blasting is a common approach to word-level model checking, for model checkers that operate directly on the word level, bit-level certification is not sufficient, and it leaves the bit-blasting process itself uncertified. In fact, during the development of Cerbtora and our prototype $k$-induction model checker, fuzzing immediately revealed bugs in `btor2aiger`, the bit blaster in the Btor2Tools library. Figure 3 compares certificate sizes obtained by bit-blasting the models for the 47 benchmarks solved by $k$-induction against certificates generated directly at the word level. This shows that the word-level certificates are five times more compact in the geometric mean.

Table 2 reports certificate checking times for both word-level and bit-blasted approaches. We display all benchmarks where the inductive depth $k$ exceeds 3. *Step* is typically the most expensive check. In general, we observe certificate checking to be slightly more efficient at the word level.

## 5 Conclusion

We presented Certifaiger, a certificate checker for bit-level model checking, and its word-level friend Cerbtora. As ongoing work, we are extending both tools to support liveness properties. For a fully trusted certification pipeline, we are implementing our certification approach in a theorem proving environment, including verified bit blasting and on-the-fly verification of LRAT proofs.

**Acknowledgments.** Partially funded by the European Union project (ERC, CertiFOX, 101122653). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Further funded by a gift from Intel Corporation.

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

**Data-Availability Statement** All experiments presented in this paper can be reproduced: https://doi.org/10.5281/zenodo.18629636

## References

1. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: TACAS (1). Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24
2. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: SIGSOFT FSE. pp. 326–337. ACM (2016). https://doi.org/10.1145/2950290.2950351
3. Biere, A.: Tutorial on world-level model checking. In: FMCAD. p. 1. IEEE (2020). https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_3
4. Biere, A., Faller, T., Fazekas, K., Fleury, M., Froleyks, N., Pollitt, F.: CaDiCaL 2.0. In: CAV (1). Lecture Notes in Computer Science, vol. 14681, pp. 133–152. Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_7
5. Biere, A., Fazekas, K., Fleury, M., Froleyks, N.: Clausal congruence closure. In: Chakraborty, S., Jiang, J.R. (eds.) 27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India. LIPIcs, vol. 305, pp. 6:1–6:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.SAT.2024.6
6. Biere, A., Fazekas, K., Fleury, M., Froleyks, N.: Clausal equivalence sweeping. In: Narodytska, N., Rümmer, P. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024. pp. 1–6. IEEE (2024). https://doi.org/10.34727/2024/ISBN.978-3-85448-065-5_29
7. Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University (2011). https://doi.org/10.35011/fmvtr.2011-2
8. Biere, A., Herbstritt, M., Berre, D.L., Wieringa, S., Niemetz, A.: Aiger: And-inverter graph (aig) format and tools. https://github.com/arminbiere/aiger (2006)
9. Biere, A., Niemetz, A., Preiner, M.: Btor2tools: Btor2 utilities and libraries. https://github.com/hwmcc/btor2tools (2016)
10. Bjesse, P.: A practical approach to word level model checking of industrial netlists. In: International Conference on Computer Aided Verification. pp. 446–458. Springer (2008). https://doi.org/10.1007/978-3-540-70545-1_43
11. Brayton, R.K., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: CAV. Lecture Notes in Computer Science, vol. 6174, pp. 24–40. Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_5

12. Fleury, A., Heisinger, M.: Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. Sat Competition **2020**, 50 (2020)
13. Froleyks, N., Yu, E., Biere, A., Heljanko, K.: Certifying phase abstraction. In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14739, pp. 284–303. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_17
14. Froleyks, N., Yu, E., Preiner, M., Biere, A., Heljanko, K.: Introducing certificates to the hardware model checking competition. In: Piskac, R., Rakamaric, Z. (eds.) Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15931, pp. 281–295. Springer (2025). https://doi.org/10.1007/978-3-031-98668-0_14
15. Heule, M.J.H.: Proofs of unsatisfiability. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 635–668. IOS Press (2021). https://doi.org/10.3233/FAIA200998
16. Hoenicke, J., Schindler, T.: A simple proof format for smt. In: SMT. pp. 54–70 (2022)
17. Lammich, P.: Fast and verified unsat certificate checking. In: International Joint Conference on Automated Reasoning. pp. 439–457. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_26
18. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13965, pp. 3–17. Springer (2023). https://doi.org/10.1007/978-3-031-37703-7_1
19. Niemetz, A., Preiner, M., Wolf, C., Biere, A.: Btor2 , btormc and boolector 3.0. In: CAV (1). Lecture Notes in Computer Science, vol. 10981, pp. 587–595. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_32
20. Pollitt, F., Fleury, M., Biere, A.: Faster LRAT checking than solving with CaDiCaL. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 271, pp. 21:1–21:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2023). https://doi.org/10.4230/LIPIcs.SAT.2023.21
21. Schurr, H.J., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: Towards a generic smt proof format. arXiv preprint arXiv:2107.02354 (2021). https://doi.org/10.48550/arXiv.2107.02354
22. Schurr, H., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: Towards a generic SMT proof format (extended abstract). In: Keller, C., Fleury, M. (eds.) Proceedings Seventh Workshop on Proof eXchange for Theorem Proving, PxTP 2021, Pittsburg, PA, USA, July 11, 2021. EPTCS, vol. 336, pp. 49–54 (2021). https://doi.org/10.4204/EPTCS.336.6
23. Tan, Y.K., Heule, M.J.H., Myreen, M.O.: cake_lpr: Verified propagation redundancy checking in cakeml. In: TACAS (2). Lecture Notes in Computer Science, vol. 12652, pp. 223–241. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_12
24. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Mathematics and Mathematical Logic **2**, 115–125 (1968)

25. Yu, E., Biere, A., Heljanko, K.: Progress in certifying hardware model checking results. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12760, pp. 363–386. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_17
26. Yu, E., Froleyks, N., Biere, A., Heljanko, K.: Stratified certification for k-Induction. In: Griggio, A., Rungta, N. (eds.) 22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022. pp. 59–64. IEEE (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_11
27. Yu, E., Froleyks, N., Biere, A., Heljanko, K.: Towards compositional hardware model checking certification. In: Nadel, A., Rozier, K.Y. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, October 24-27, 2023. pp. 1–11. IEEE (2023). https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_12
28. Zeller, A., Hildebrandt, R.: Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering **28**(2), 183–200 (2002). https://doi.org/10.1109/32.988498
29. Zhang, Z., Klees, G., Wang, E., Hicks, M., Wei, S.: Fuzzing configurations of program options. ACM Trans. Softw. Eng. Methodol. **32**(2), 53:1–53:21 (2023). https://doi.org/10.1145/3580597