



# Stratified Certification for $k$ -Induction

Emily Yu\*   
zhengqi.yu@jku.at

Nils Froleyks\*   
nils.froleyks@jku.at

Armin Biere†   
biere@cs.uni-freiburg.de

Keijo Heljanko‡§   
keijo.heljanko@helsinki.fi

\*Johannes Kepler University, Linz, Austria

‡Helsinki Institute for Information Technology and

†Albert–Ludwigs–University, Freiburg, Germany

§University of Helsinki, Helsinki, Finland

**Abstract**—Our recently proposed certification framework for bit-level  $k$ -induction-based model checking has been shown to be quite effective in increasing the trust of verification results even though it partially involved quantifier reasoning. In this paper we show how to simplify the approach by assuming reset functions to be stratified. This way it can be lifted to word-level and in principle to other theories where quantifier reasoning is difficult. Our new method requires six simple SAT checks and one polynomial-time check, allowing certification to remain in co-NP while the previous approach required five SAT checks and one QBF check. Experimental results show a substantial performance gain for our new approach. Finally we present and evaluate our new tool CERTIFAIGER-WL which is able to certify  $k$ -induction-based word-level model checking.

## I. INTRODUCTION

Over the past several years, there has been growing interest in system verification using word-level reasoning. Satisfiability Modulo Theories (SMT) solvers for the theory of fixed-size bit-vectors are widely used for word-level reasoning [1], [2]. For example, word-level model checking has been an important part of the hardware model checking competitions since 2019. Given the theoretical and practical importance of word-level verification, a generic certification framework for it is necessary. As quantifiers in combination with bit-vectors are challenging for SMT solvers and various works have focused on eliminating quantifiers in SMT [2]–[4], a main goal of this paper is to generate certificates without quantification.

Temporal induction (also known as  $k$ -induction) [5] is a well-known model checking technique for verifying software and hardware systems. An attractive feature of  $k$ -induction is that it is natural to integrate it with modern SAT/SMT solvers, making it popular in both bit-level model checking and beyond [6]–[8], including word-level model checking.

Certification helps gaining confidence in model checking results, which is important for both safety- and business-critical applications. There have been several contributions focusing on generating proofs for SAT-based model checking [9]–[15]. For example [16] and [14] proposed an approach to certify LTL properties and a few preprocessing techniques by generating deductive proofs. In this paper, we focus on finding an inductive invariant for  $k$ -induction. Unlike other SAT/SMT-based techniques such as IC3 [17] and interpolation [18], [19],  $k$ -induction does not automatically generate an inductive

invariant that can be used as a certificate [20]. In previous research [21], certification of  $k$ -induction can be achieved via five SAT checks together with a one-alternation QBF check, redirecting the certification problem to verifying an inductive invariant in an extended model that simulates the original one.

At the heart of the present contribution is the idea of reducing the certification method of  $k$ -induction to pure SAT checks, *i.e.*, eliminating the quantifiers. This enables us to complete the certification procedure at a lower complexity, and to directly apply the framework to word-level certification. We introduce the notion of stratified simulation which allows us to reason about the simulation relation between two systems.

This stratified simulation relation can be verified by three SAT and a polynomial-time check. The latter checks whether the reset function is indeed stratified. In addition, we present a witness circuit construction which simulates the original under the stratified simulation relation thus creating a simpler and more elegant certification construction for  $k$ -induction.

While the previous work only focused on bit-level model checking, we also lift our method to word-level by implementing a complete toolsuite CERTIFAIGER-WL, where the experiments show the practicality and effectiveness of our certification method for word-level models.

## II. BACKGROUND

This paper extends previous work in certification for  $k$ -induction-based bit-level model checking [21]. In this section, we present essential concepts and notations.

For the sake of simplicity we work with *functions* represented as interpreted terms and formulas over fixed but arbitrary theories which include an equality predicate. We further assume a finite sorted set of variables  $L$  where each variable  $l \in L$  is associated with a finite domain of possible values. We also include Boolean variables as variables with a domain of  $\{\top, \perp\}$ , for which we keep standard notations.

For two sets of variables  $I$  and  $L$ , we also write  $I, L$  to denote their union. Given two functions  $f(V), g(V')$  where  $V \subseteq V'$  (represented as interpreted terms over our fixed but arbitrary theories) we call them *equivalent*, written  $f(V) \equiv g(V')$ , if for every assignment to variables in  $V$  and  $V'$  that matches on the shared set of variables  $V$ , the functions  $f(V), g(V')$  have the same values. Additionally, we use “ $\simeq$ ” for syntactic equivalence [22], “ $\rightarrow$ ” for syntactic

Funded by FWF project W1255-N23, the LIT AI Lab funded by the State of Upper Austria, and Academy of Finland project 336092.

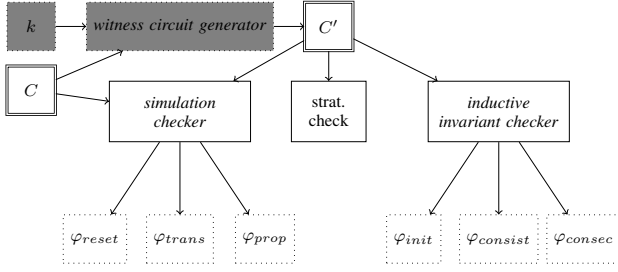


Fig. 1: An outline of the certification approach. Given some value of  $k$  and a model  $C$ ,  $C'$  is the resulting witness circuit. The coloured area is specific to our approach for  $k$ -induction, and the rest corresponds to the general certification flow.

implication, and “ $\Rightarrow$ ” for semantic implication. To define semantical concepts or abbreviations we stick to equality “ $=$ ”. We use  $vars(f)$  to denote the set of variables occurring in the syntactic representation of a function  $f$ .

In word-level model checking operations are applied to fixed-size bit-vectors. We introduce the notion of word-level circuits where we model inputs and latches as finite-domain variables.

**Definition 1** (Circuit). *A circuit is a tuple  $C = (I, L, R, F, P)$  such that:*

- $I$  is a finite set of input variables.
- $L$  is a finite set of latch variables.
- $R = \{r_l(L) \mid l \in L\}$  is a set of reset functions.
- $F = \{f_l(I, L) \mid l \in L\}$  is a set of transition functions.
- $P(I, L)$  is a function that evaluates to a Boolean output, encoding the (good states) property.

By Def. 1 a circuit represents a hardware system in a fully symbolic form. In order to talk about the reset functions of a subset of latches  $L'' \subseteq L$ , we also write

$$R(L'') = \bigwedge_{l \in L''} (l \simeq r_l(L)).$$

The following four definitions are adapted from our previous work [21] for completeness of exposition.

**Definition 2** (Unrolling). *For an unrolling depth  $m \in \mathbb{N}$ , the unrolling of a circuit  $C = (I, L, R, F, P)$  of length  $m$  is defined as  $U_m = \bigwedge_{i \in [0, m]} (L_{i+1} \simeq F(I_i, L_i))$ .*

**Definition 3** (Inductive invariant). *Given a circuit  $C$  with a property  $P$ ,  $\phi(I, L)$  is an inductive invariant in  $C$  if and only if the following conditions hold:*

- 1)  $R(L) \Rightarrow \phi(I, L)$ , “initiation”
- 2)  $\phi(I, L) \Rightarrow P(I, L)$ , and “consistency”
- 3)  $U_1 \wedge \phi(I_0, L_0) \Rightarrow \phi(I_1, L_1)$ . “consecution”

As a generalisation of the notion of an inductive invariant,  $k$ -induction checks  $k$  steps of unrolling instead of 1. In the following, to verify that a property is an inductive invariant, we consider it as the special case of  $k$ -induction with  $k = 1$  and  $\phi(I, L) = P(I, L)$ .

**Definition 4** ( $k$ -induction). *Given a circuit  $C$  with a property  $P$ ,  $P$  is called  $k$ -inductive in  $C$  if and only if the following two conditions hold:*

- 1)  $U_{k-1} \wedge R(L_0) \Rightarrow \bigwedge_{i \in [0, k)} P(I_i, L_i)$ , and “BMC”
- 2)  $U_k \wedge \bigwedge_{i \in [0, k)} P(I_i, L_i) \Rightarrow P(I_k, L_k)$ . “consecution”

**Definition 5** (Combinational extension).

*A circuit  $C' = (I', L', R', F', P')$  combinationaly extends a circuit  $C = (I, L, R, F, P)$  if  $I = I'$  and  $L \subseteq L'$ .*

### III. CERTIFICATION

In this section we introduce and formalise our certification approach which reduces the certification problem to six SAT checks and one polynomial stratification check.

The certification approach is outlined in Fig. 1. Intuitively, a *witness circuit* is generated from a given value of  $k$  (provided by the model checker) and a model (either bit-level or word-level). The witness circuit simulates the original circuit while allowing more behaviours (we formally define it as the *stratified simulation relation*). In practice, the witness circuit would be required to be provided by model checkers as the certificate in hardware model checking competitions.

We also perform a polynomial-time stratification check on the witness circuit. The check requires that the definition of the reset function is stratified, *i.e.*, no cyclic dependencies between the reset definitions of the variables exist. This is the case for all hardware model checking competition benchmarks. Even though cyclic definitions have been the subject of study in several papers [23]–[25], they are usually avoided due to the complexity of their analysis and subtle effects on semantics.

The approach in [21] can handle cyclic resets but at the cost of QBF quantification, and thus [21] not being able to be efficiently adapted to the context of word-level verification. Furthermore, the witness circuit includes an inductive invariant which serves as a proof certificate, which is verified by another three SAT checks as defined in Def. 3.

We begin by defining stratified reset functions.

**Definition 6.** (*Dependency graph.*) *Given a set of latches  $L$  and a set of reset functions  $R = \{r_l \mid l \in L\}$ , the dependency graph  $G_R$  has latch variables  $L$  as nodes and contains a directed edge  $(a, b)$  from  $a$  to  $b$  iff  $a \in vars(r_b)$  and  $r_b \neq b$ .*

Latches with undefined reset value are common in applications. We simply set  $r_b = b$  for some uninitialised latch  $b$  in such a case (as in AIGER and BTOR) to avoid being required to reason about ternary logic or partial functions. Thus the syntactic condition “ $r_b \neq b$ ” in the last definition simply avoids spurious self-loops in the dependency graph for latches with undefined reset values.

**Definition 7.** (*Stratified resets.*) *Given a set of latches  $L$ , and a set of reset functions  $R = \{r_l \mid l \in L\}$ .  $R$  is said to be stratified iff  $G_R$  is acyclic.*

TABLE I: Summary of certification results for the bit-level TIP suite.

Benchmarks	$\varphi_{init}$		$\varphi_{consist}$		$\varphi_{consec}$		$\varphi_{trans}$		$\varphi_{prop}$		$\varphi_{reset}$	
	$t_1$	$t_2$	$t_1$	$t_2$	$t_1$	$t_2$	$t_1$	$t_2$	$t_1$	$t_2$	$t_1$	$t_2$
c.periodic	7.78	0.06	0.06	0.06	56.82	56.29	0.15	0.14	0.05	0.05	84.04	0.00
n.guidance <sub>1</sub>	0.19	0.01	0.01	0.01	3.73	3.79	0.12	0.12	0.01	0.01	1.21	0.00
n.guidance <sub>7</sub>	4.09	0.02	0.02	0.02	18.40	18.17	0.12	0.12	0.02	0.02	25.22	0.00
n.tcasp <sub>2</sub>	0.17	0.01	0.01	0.01	2.64	2.68	0.23	0.23	0.01	0.02	1.79	0.00
n.tcasp <sub>3</sub>	0.11	0.01	0.01	0.01	1.82	1.70	0.23	0.26	0.02	0.02	1.01	0.00
v.prodcell <sub>12</sub>	2.35	0.03	0.03	0.03	59.05	59.22	0.12	0.12	0.03	0.03	8.48	0.00
v.prodcell <sub>13</sub>	0.22	0.01	0.01	0.01	2.99	2.99	0.12	0.12	0.01	0.01	0.20	0.00
v.prodcell <sub>14</sub>	0.64	0.02	0.02	0.02	13.69	13.69	0.12	0.12	0.02	0.02	1.45	0.00
v.prodcell <sub>15</sub>	2.22	0.02	0.03	0.03	32.66	32.28	0.12	0.12	0.02	0.02	2.26	0.00
v.prodcell <sub>16</sub>	0.01	0.01	0.01	0.01	1.19	1.20	0.12	0.12	0.01	0.01	0.06	0.00
v.prodcell <sub>17</sub>	2.34	0.03	0.03	0.03	48.51	48.17	0.12	0.12	0.03	0.03	6.86	0.00
v.prodcell <sub>18</sub>	0.67	0.01	0.01	0.01	8.67	8.78	0.12	0.12	0.02	0.02	0.79	0.00
v.prodcell <sub>19</sub>	1.66	0.02	0.02	0.03	31.98	31.78	0.12	0.12	0.03	0.03	3.73	0.00
v.prodcell <sub>24</sub>	3.32	0.04	0.04	0.04	112.12	115.18	0.12	0.12	0.04	0.04	17.64	0.00

Columns report the benchmark names, and the time (in seconds) used for each SAT check by CERTIFAIGER ( $t_1$ ) and CERTIFAIGER++ ( $t_2$ ) respectively.

Interestingly, the SAT solving time for the new reset check is close to zero, which checks the equality of the reset functions between the shared set of latches and the latches in the original circuit. This is because all latches in the benchmark set are initialized to  $\perp$ , thus making the SAT checks rather trivial.

**Definition 8.** (Stratified circuit.) A circuit  $C = (I, L, R, F, P)$  is said to be stratified iff  $R$  is stratified.

The stratification check can be done in polynomial time using Def. 7 and it is enforced syntactically in the two hardware description formats AIGER and BTOR2.

**Definition 9.** (Stratified simulation.) Given two stratified circuits  $C$  and  $C'$ , where  $C'$  combinationally extends  $C$ . There is a stratified simulation between  $C'$  and  $C$  iff,

- 1)  $r_l(L) \equiv r'_l(L')$  for  $l \in L$ , “reset”
- 2)  $f_l(I, L) \equiv f'_l(I, L')$  for  $l \in L$ , and “transition”
- 3)  $P'(I, L') \Rightarrow P(I, L)$ . “property”

In essence, the crucial change here compared to the combinational simulation definition in [21] is the reset condition, whose simplification was possible under the stratification assumption. The above three conditions are encoded into SAT/SMT formulas ( $\varphi_{reset}, \varphi_{trans}, \varphi_{prop}$  in Fig. 1) which are then checked by a solver for validity. In the rest of the paper, we simply refer to the stratified simulation relation as simulation relation. Proofs of the presented theoretical results can be found in an extended version of this paper [26].

**Theorem 1.** Given two circuits  $C$  and  $C'$ , where  $C'$  simulates  $C$ . If  $C'$  is safe, then  $C$  is also safe.

Next, we introduce the witness circuit construction. This is similar to the construction in [21] but differs in several details, e.g., the reset function definition is stratified and significantly simplified compared to [21].

**Definition 10.** (Witness circuit.) Given a circuit  $C = (I, L, R, F, P)$  and an integer  $k \in \mathbb{N}^+$ , its witness circuit  $C' = (I', L', R', F', P')$  is defined as follows:

- 1)  $I' = I$  (also referred to as  $X^{k-1}$ ),
- 2)  $L' = L^{k-1} \cup \dots \cup L^0 \cup X^{k-2} \cup \dots \cup X^0 \cup B$  where,
  - $L^{k-1} = L$ , the other variables sets are copies of  $I$  and  $L$  respectively with the same variable domains.
  - $B = \{b^{k-1}, \dots, b^0\}$  are Booleans.

3)  $R'$  :

- for  $l \in L^{k-1}$ ,  $r'_l = r_l(L^{k-1})$ .
- for  $l \in L^0 \cup \dots \cup L^{k-2} \cup X^0 \cup \dots \cup X^{k-2}$ ,  $r'_l = l$ .
- $r'_{b^{k-1}} = \top$ .
- for  $i \in [0, k-1)$ ,  $r'_{b^i} = \perp$ .

4)  $F'$  :

- for  $l \in L^{k-1}$ ,  $f'_l = f_l(I', L^{k-1})$ .
- $f'_{b^{k-1}} = b^{k-1}$ .
- for  $i \in [0, k-1)$ ,  $l^i \in (L^i \cup X^i \cup \{b^i\})$ ,  $f'_{l^i} = l^{i+1}$ .

5)  $P' = \bigwedge_{i \in [0, 4]} p_i(I', L')$  where

- $p_0(I', L') = \bigwedge_{i \in [0, k-1)} (b^i \rightarrow b^{i+1})$ .
- $p_1(I', L') = \bigwedge_{i \in [0, k-1)} (b^i \rightarrow (L^{i+1} \simeq F(X^i, L^i)))$ .
- $p_2(I', L') = \bigwedge_{i \in [0, k)} (b^i \rightarrow P(X^i, L^i))$ .
- $p_3(I', L') = \bigwedge_{i \in [1, k)} ((\neg b^{i-1} \wedge b^i) \rightarrow R(L^i))$ .
- $p_4(I', L') = b^{k-1}$ .

Here we extend a given circuit to a witness circuit, which has  $k$  copies of the original latches and inputs, and additional  $k$  latches of  $B$  that we refer to as the initialisation bits. We refer to the  $\{k-1\}th$  as the most recent, and the 0th as the oldest. Intuitively the most recent copy unrolls in the same way as the original circuit, with the older copies copying the previous values of the younger copies. When all initialisation bits are  $\top$ , we say the machine has reached a “full initialisation” state.

**Lemma 1.** Given a circuit  $C$  with reset function  $R$  and its witness circuit  $C'$  with reset function  $R'$ . If  $R$  is stratified, then  $R'$  is also stratified.

**Theorem 2.** Given a circuit  $C$  and its witness circuit  $C'$ .  $C'$  simulates  $C$ .

We now present the main theorem of this paper.

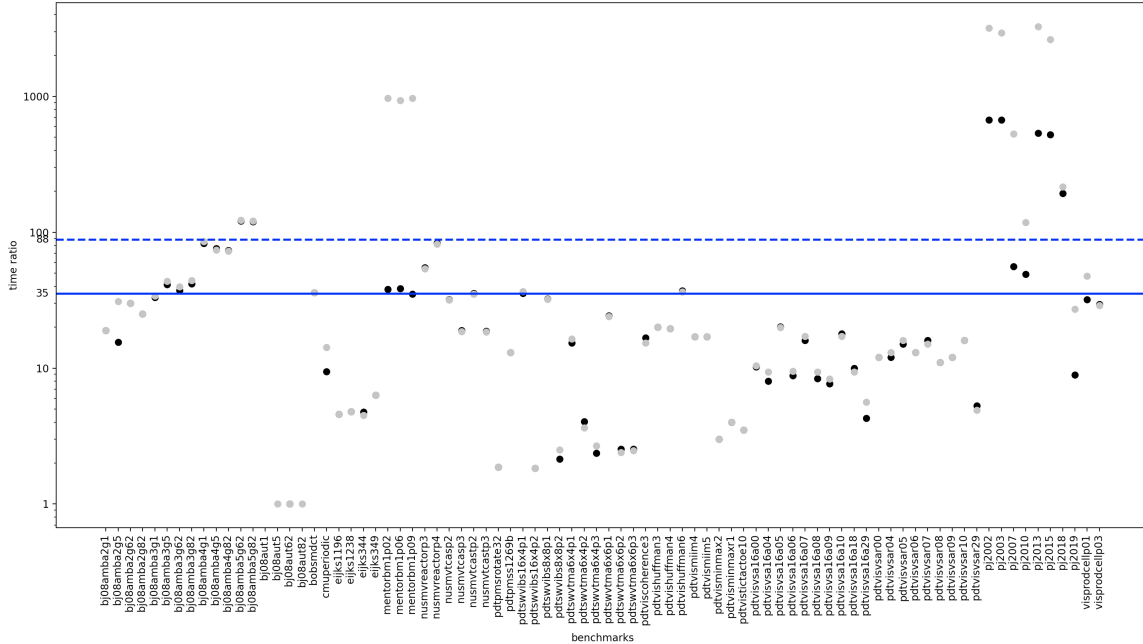


Fig. 2: Bit-level: the experimental results of the HWMCC 2010. The benchmark names are shown on the x-axis. The time ratio on the y-axis is calculated by computing certification time divided by model checking time (ran on the model checker McAiger [27]). The black dots in the graph are the results obtained from CERTIFAIGER++ and the grey dots are from CERTIFAIGER. The straight line and the dashed line are the calculated means for CERTIFAIGER++ and CERTIFAIGER respectively. As we can see from the plot, especially for the instances with certification time greater than 500 seconds, the new implementation significantly improved the certification performance.

**Theorem 3.** Given a circuit  $C = (I, L, R, F, P)$  and its witness circuit  $C' = (I', L', R', F', P')$ .  $P$  is  $k$ -inductive in  $C$  iff  $P'$  is 1-inductive in  $C'$ .

#### IV. IMPLEMENTATION AND EXPERIMENTAL EVALUATION

We implemented the proposed certification approach into two complete toolkits [28]: CERTIFAIGER++ for bit-level, and CERTIFAIGER-WL for word-level. We evaluate the performance of our tools against several benchmark sets from previous literature and the model checking competitions.

##### A. Bit-level

Our toolkit CERTIFAIGER++ extends the certification toolkit CERTIFAIGER [21]. Note that the AIGER format only allows stratified resets by default. All experiments were performed on a workstation with an Intel® Core™ i9-9900 CPU 3.60GHz computer with 32GB RAM running Manjaro with Linux kernel 5.4.72-1.

To determine the speedups of the new implementation proposed in this paper, we performed experiments on the same sets of the benchmarks used in [21]. The results are reported in Table I. There are significant overall gains in the initiation checks ( $\varphi_{init}$ ) as well as the reset checks ( $\varphi_{reset}$ ). For the initiation check which checks the invariant holds in all initial

states, the performance improvement is largely due to the simplification of the reset functions in the new witness circuit construction.

The results in Fig. 2 demonstrate that CERTIFAIGER++ in general is much faster than CERTIFAIGER during the overall certification process. Compared to CERTIFAIGER, CERTIFAIGER++ achieved overall speedups of 2.46 times. We observe performance gains in most benchmarks, as the previous performance bottleneck for certain benchmarks is the QBF solving time for the reset check. For other instances, the bottleneck is the SAT solving time for the consecution check, which is also improved due to a simpler reset construction (as part of the inductive invariant).

##### B. Word-level

We further lifted the method to certifying word-level model checking by implementing an experimental toolkit called CERTIFAIGER-WL. CERTIFAIGER-WL follows the same architecture design as CERTIFAIGER++ and uses Boolector [29] as the underlying SMT solver. All models and SMT encodings are in BTOR2 [29] format, which is the standard word-level model checking format used in hardware model checking competitions.

TABLE II: Summary of certification results word-level benchmarks from the HWMCC20

Benchmarks	k	#model	#witness	ModelCh.	Certifi.	Consec.	Ratio
paper_v3	256	35	12801	10.25	1.14	0.90	0.11
VexRiscv-regch0-15-p0	17	2149	43077	10.31	4.04	3.29	0.39
zipcpu-pfcache-p02	37	1818	105874	13.95	4.40	2.73	0.32
zipcpu-pfcache-p24	37	1818	105874	14.35	4.49	2.83	0.31
zipcpu-busdelay-p43	101	950	145466	15.29	6.14	3.86	0.40
dspfilters_fastfir_second-p42	15	6732	115388	16.11	14.80	12.96	0.92
zipcpu-pfcache-p01	41	1818	117434	18.33	6.34	4.47	0.35
dspfilters_fastfir_second-p10	11	6732	84348	24.56	9.76	8.44	0.40
zipcpu-busdelay-p15	101	950	145466	58.17	8.18	5.89	0.14
qspiflash_dualfexpress_divfive-p120	97	3100	394412	63.58	22.07	14.58	0.35
zipcpu-pfcache-p22	93	1818	267714	166.07	23.66	19.06	0.14
VexRiscv-regch0-20-p0	22	2149	55862	240.50	16.76	15.76	0.07
dspfilters_fastfir_second-p14	15	6732	115388	354.01	21.27	19.44	0.06
dspfilters_fastfir_second-p11	21	6732	161948	627.69	46.88	44.30	0.07
dspfilters_fastfir_second-p45	17	6732	130908	1094.11	30.14	28.06	0.03
VexRiscv-regch0-30-p1	32	2150	81464	1444.47	83.38	81.95	0.06
dspfilters_fastfir_second-p43	19	6732	146428	2813.61	58.02	55.69	0.02

To select the benchmarks presented, we first ran AVR with a timeout of 5000 seconds. We display the results here that are of particular interest with a running time of more than ten seconds (there are 7 instances with  $k = 1$  which were certified and solved under 0.2s). Columns report the benchmark names, the value of  $k$ , the size of the model (measured in number of instructions) and the generated witness, the model checking time, and certification time (in seconds). Additionally we list the time Boolector took to solve the consecution check, as well as the ratio of model checking vs. certification time. We only list the consecution check (Consec.) here as it takes up the majority of the certification time.

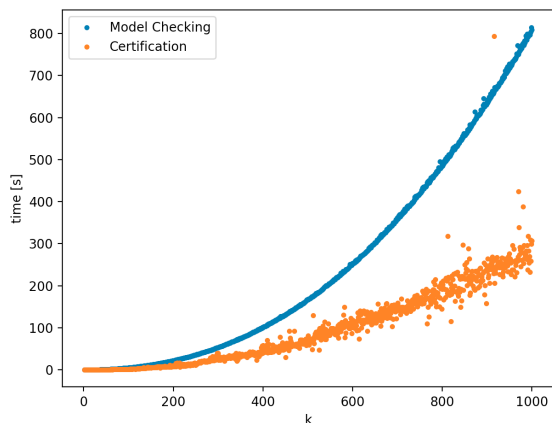


Fig. 3: Word-level: model checking vs. certification time for the Counter example (with 500 bits) with increasing values of  $k$ . For the experiments, we fixed the modulo bound at 32 and scaled the inductive depth up to 1000. The certification time is significantly smaller than the model checking time. As the value of  $k$  increases, on average the certification time is proportionally lower.

We ran benchmarks of the Counter example [21] on AVR [30] to get the values of  $k$ . Fig. 3 shows the experimental results obtained with CERTIFAIGER-WL under the same setting as Section IV-A. Interestingly, the certification time is much lower than the model checking time as can be seen in the diagram, meaning certification is at a low cost.

In Table II we report the experimental results obtained on a superset of the hardware model checking competition 2020 [31] benchmarks. We observe that the certification time is much lower than model checking time. Including certification would increase the runtime of AVR on the model checking benchmarks by less than 6%.

## V. CONCLUSION AND FUTURE WORK

We have presented a new certification framework which allows certification for  $k$ -induction to be done by six SAT checks and a polynomial-time check. We further lifted our approach to word-level, and implemented our method in both contexts. Experimental results demonstrate the effectiveness and computational efficiency of our toolkits. The removal of the QBF quantifiers has reduced the theoretical complexity of the problem compared to [21] and also reduced the overall runtime overhead of the certification. Additionally, in future work we plan to obtain formally verified certificate checkers by using theorem proving. Finally, how to certify liveness properties is another important avenue of further research.

## REFERENCES

- [1] A. Niemetz, M. Preiner, and A. Biere, “Precise and complete propagation based local search for satisfiability modulo theories,” in *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, S. Chaudhuri and A. Farzan, Eds., vol. 9779. Springer, 2016, pp. 199–217. [Online]. Available: [https://doi.org/10.1007/978-3-319-41528-4\\_11](https://doi.org/10.1007/978-3-319-41528-4_11)
- [2] —, “Propagation based local search for bit-precise reasoning,” *Formal Methods Syst. Des.*, vol. 51, no. 3, pp. 608–636, 2017. [Online]. Available: <https://doi.org/10.1007/s10703-017-0295-6>
- [3] A. Niemetz, M. Preiner, A. Reynolds, Y. Zohar, C. W. Barrett, and C. Tinelli, “Towards satisfiability modulo parametric bit-vectors,” *J. Autom. Reason.*, vol. 65, no. 7, pp. 1001–1025, 2021.
- [4] A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, and C. Tinelli, “Solving quantified bit-vectors using invertibility conditions,” in *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, H. Chockler and G. Weissenbacher, Eds., vol. 10982. Springer, 2018, pp. 236–255. [Online]. Available: [https://doi.org/10.1007/978-3-319-96142-2\\_16](https://doi.org/10.1007/978-3-319-96142-2_16)
- [5] M. Sheeran, S. Singh, and G. Stålmarck, “Checking safety properties using induction and a SAT-solver,” in *FMCAD*, ser. Lecture Notes in Computer Science, vol. 1954. Springer, 2000, pp. 108–125.
- [6] A. Champion, A. Mebsout, C. Stickel, and C. Tinelli, “The Kind 2 model checker,” in *CAV (2)*, ser. Lecture Notes in Computer Science, vol. 9780. Springer, 2016, pp. 510–517.
- [7] L. M. de Moura, S. Owre, H. Rueß, J. M. Rushby, N. Shankar, M. Sorea, and A. Tiwari, “SAL 2,” in *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings*, ser. Lecture Notes in Computer Science, R. Alur and D. A. Peled, Eds., vol. 3114. Springer, 2004, pp. 496–500. [Online]. Available: [https://doi.org/10.1007/978-3-540-27813-9\\_45](https://doi.org/10.1007/978-3-540-27813-9_45)
- [8] D. Jovanovic and B. Dutertre, “Property-directed k-induction,” in *FMCAD*. IEEE, 2016, pp. 85–92.
- [9] S. Conchon, A. Mebsout, and F. Zaïdi, “Certificates for parameterized model checking,” in *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*, ser. Lecture Notes in Computer Science, N. Bjørner and F. S. de Boer, Eds., vol. 9109. Springer, 2015, pp. 126–142.
- [10] A. Gurfinkel and A. Ivrii, “K-induction without unrolling,” in *FMCAD*. IEEE, 2017, pp. 148–155.
- [11] T. Kuismin and K. Heljanko, “Increasing confidence in liveness model checking results with proofs,” in *HaiFa Verification Conference*, ser. Lecture Notes in Computer Science, vol. 8244. Springer, 2013, pp. 32–43.
- [12] K. S. Namjoshi, “Certifying model checkers,” in *CAV*, ser. Lecture Notes in Computer Science, vol. 2102. Springer, 2001, pp. 2–13.
- [13] L. G. Wagner, A. Mebsout, C. Tinelli, D. D. Cofer, and K. Slind, “Qualification of a model checker for avionics software verification,” in *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, ser. Lecture Notes in Computer Science, C. W. Barrett, M. Davies, and T. Kahsai, Eds., vol. 10227, 2017, pp. 404–419.
- [14] A. Griggio, M. Roveri, and S. Tonetta, “Certifying proofs for LTL model checking,” in *FMCAD*. IEEE, 2018, pp. 1–9.
- [15] Z. Yu, A. Biere, and K. Heljanko, “Certifying hardware model checking results,” in *ICFEM*, ser. Lecture Notes in Computer Science, vol. 11852. Springer, 2019, pp. 498–502.
- [16] A. Griggio, M. Roveri, and S. Tonetta, “Certifying proofs for SAT-based model checking,” *Formal Methods Syst. Des.*, vol. 57, no. 2, pp. 178–210, 2021.
- [17] A. R. Bradley, “SAT-based model checking without unrolling,” in *VMCAI*, ser. Lecture Notes in Computer Science, vol. 6538. Springer, 2011, pp. 70–87.
- [18] K. L. McMillan, “Interpolation and SAT-based model checking,” in *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, ser. Lecture Notes in Computer Science, W. A. H. Jr. and F. Somenzi, Eds., vol. 2725. Springer, 2003, pp. 1–13. [Online]. Available: [https://doi.org/10.1007/978-3-540-45069-6\\_1](https://doi.org/10.1007/978-3-540-45069-6_1)
- [19] —, “An interpolating theorem prover,” *Theor. Comput. Sci.*, vol. 345, no. 1, pp. 101–121, 2005. [Online]. Available: <https://doi.org/10.1016/j.tcs.2005.07.003>
- [20] Z. Manna and A. Pnueli, *Temporal verification of reactive systems - safety*. Springer, 1995.
- [21] E. Yu, A. Biere, and K. Heljanko, “Progress in certifying hardware model checking results,” in *CAV (2)*, ser. Lecture Notes in Computer Science, vol. 12760. Springer, 2021, pp. 363–386.
- [22] A. Degtyarev and A. Voronkov, “Equality reasoning in sequent-based calculi,” in *Handbook of Automated Reasoning (in 2 volumes)*, J. A. Robinson and A. Voronkov, Eds. Elsevier and MIT Press, 2001, pp. 611–706.
- [23] S. Malik, “Analysis of cyclic combinational circuits,” *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 13, no. 7, pp. 950–956, 1994. [Online]. Available: <https://doi.org/10.1109/43.293952>
- [24] J. R. Jiang, A. Mishchenko, and R. K. Brayton, “On breakable cyclic definitions,” in *2004 International Conference on Computer-Aided Design, ICCAD 2004, San Jose, CA, USA, November 7-11, 2004*. IEEE Computer Society / ACM, 2004, pp. 411–418. [Online]. Available: <https://doi.org/10.1109/ICCAD.2004.1382610>
- [25] M. D. Riedel, *Cyclic combinational circuits*. California Institute of Technology, 2004.
- [26] E. Yu, N. Froleyks, A. Biere, and K. Heljanko, “Stratified certification for k-induction,” 2022. [Online]. Available: <https://arxiv.org/abs/2208.01443>
- [27] A. Biere and R. Brummayer, “Consistency checking of all different constraints over bit-vectors within a SAT solver,” in *FMCAD*. IEEE, 2008, pp. 1–4.
- [28] Certifaiger++ and Certifaiger-wl, “Certifaiger++ and Certifaiger-wl,” 2022, <http://fmv.jku.at/certifaiger>.
- [29] A. Niemetz, M. Preiner, C. Wolf, and A. Biere, “Btor2 , btormc and boolector 3.0,” in *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Chockler and G. Weissenbacher, Eds., vol. 10981. Springer, 2018, pp. 587–595. [Online]. Available: [https://doi.org/10.1007/978-3-319-96145-3\\_32](https://doi.org/10.1007/978-3-319-96145-3_32)
- [30] A. Goel and K. A. Sakallah, “AVR: abstractly verifying reachability,” in *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Biere and D. Parker, Eds., vol. 12078. Springer, 2020, pp. 413–422. [Online]. Available: [https://doi.org/10.1007/978-3-030-45190-5\\_23](https://doi.org/10.1007/978-3-030-45190-5_23)
- [31] M. Preiner, A. Biere, and N. Froleyks, “Hardware model checking competition 2020,” 2020, <http://fmv.jku.at/hwmcc20/>.