Certifying Constraints in Hardware Model Checking

Nils Froleyks[1], Emily Yu[2], Armin Biere[3], and Keijo Heljanko[45]

[1] KU Leuven, Belgium
[2] Leiden University, Netherlands
[3] University of Freiburg, Freiburg, Germany
[4] University of Helsinki, Helsinki, Finland
[5] Helsinki Institute for Information Technology, Helsinki, Finland

Abstract. Model checking is a powerful automated reasoning technique for verifying hardware designs, ensuring that they function correctly before deployment. However, modern model checkers are complex software systems with hundreds of thousands of lines of code, making them prone to errors. To increase confidence in verification results, recent efforts in hardware verification focus on requiring model checkers to produce machine-checkable proofs according to a standardized format that can be independently validated. Yet, implementing proof generation across different verification algorithms presents a unique challenge. In hardware model checking, constraints play an essential role, as they encode assumptions about the environment and help simplify analysis. This paper addresses the challenge by developing a certification approach that ensures verification results remain trustworthy when constraints are present. We introduce certificate generation methods for three classes of constraints that can be extracted from the models. Furthermore, to support a broader range of constraints and more complex reset logic for industrial use, we also provide alternative Quantified Boolean Formula checks in the proof format with a single quantifier alternation. Lastly, we present a certificate generation method for k-induction with uniqueness constraints, an important model checking technique. We implement these in a certification toolkit, and provide empirical evaluation on competition benchmarks, demonstrating their effectiveness.

1 Introduction

Model checking is a widely used technique in hardware verification to ensure that hardware designs, such as processors or embedded circuits, operate correctly before they are manufactured. This automated process checks whether a design in the form of a gate-level netlist satisfies specific requirements by exploring all possible executions of the system. However, the automated reasoning tools that perform model checking, i.e., the model checkers, are complex, often consisting of hundreds of thousands of lines of code. This complexity introduces the risk of errors within the tools themselves, raising questions about the reliability of their results. To address this, recent developments in hardware verification emphasize

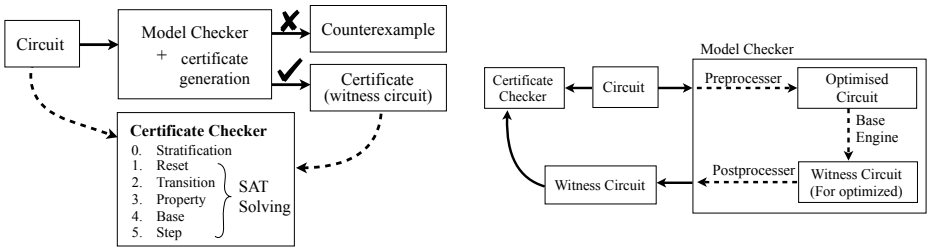

Fig. 1: An overview of the model checking certification process (left), and the certification flow when preprocessing is employed in a model checker (right).

the generation of certificates, which are formal proofs that can be independently verified by another program to confirm the verdicts [1–5].

As in most industrial verification languages [6, 7], following the assume-guarantee methodology [8], an essential feature of hardware model checking is the use of *constraints*, which represent assumptions about the environment in which the hardware operates. These assumptions simplify the verification task, improve computational efficiency, and allow model checking to tackle more complex designs. The AIGER format [9], a standard in hardware model checking competitions (HWMCCs) [10], explicitly incorporates constraints as part of its design description language. In the latest iteration, HWMCC'24, certification became mandatory [11], where all participating model checkers had to produce a machine-checkable certificate when confirming a design's correctness. The verification pipeline is illustrated in Fig. 1.

The certificate format [11], in the form of a witness circuit, relies on five simple satisfiability (SAT) checks and a polynomial-time check for reset function acyclicity (called stratification [12]). This approach is efficient and compatible with most bit-level model checking algorithms. However, challenges arise when designs include complex reset logic, such as cyclic dependencies, which complicates certificate generation but are common in industrial practice [13–15]. To overcome this, we propose alternative checks using Quantified Boolean Formula (QBF) in the same format that are less restrictive. This relaxed format is easily adapted by the certificate checker CERTIFAIGER used in the HWMCC'24, and offers more flexibility for model checker developers while maintaining trust in the results.

Since commercial designs often require hundreds of constraints to accurately model their environments, verification tools must go beyond the explicit constraints provided in the design description. *Hidden constraints* can often be inferred from the model itself [16, 17]. In fact, such constraint extraction is also implemented in the state-of-the-art model checker ABC [18], as documented in its official manual. Complementary efforts have also focused on the efficient synthesis of such constraints [19, 20]. These constraints, identified through preprocessing, refine the verification problem by focusing only on parts of system states, while preserving the validity of the original property. While this sim-

plification enhances efficiency, the resulting certificate from the model checker applies only to the reduced model, not the original. This thus adds additional challenges to certification.

Our contribution. We address this challenge by presenting a certification approach that proves the correctness of the original design while accounting for extracted constraints. Furthermore, we consider k-induction [21], a powerful model checking technique based on inductive reasoning over sequences of states. For completeness, k-induction requires *uniqueness constraints*, which exclude cyclic transitions from consideration. Producing certificates when using such constraints is challenging [22], as soundness implicitly relies on a reachability analysis over the entire state space. In this paper we also demonstrate certificate generation for it. We summarize our main contributions as follows.

1. *Certification of extracted constraints.* We introduce a method for generating certificates that account for three classes of extracted constraints: *model constraints*, *inductive constraints*, and *property constraints*. The resulting certificates validate that the verification result holds for the original model, prior to constraint extraction.
2. *Relaxed certificate format.* To accommodate more complex encoding logic in modern model checking techniques, we propose alternative QBF-based certificate checks. These relaxations extend the certificate format used in HWMCCs and remain verifiable by an independent checker.
3. *Certification of uniqueness constraints in k-induction.* We present a certificate generation method for k-induction with uniqueness constraints.
4. *Empirical evaluation.* We evaluate our approach on a set of HWMCC benchmarks, demonstrating its practicality and effectiveness.

2 Background

In the rest of the paper, we employ the following notation: let $\mathcal{V}$ be a set of Boolean variables, we consider formulas over $\mathcal{V}$ with the Boolean operators $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$. The last denotes equivalence and can have an infix negation $\not\leftrightarrow$. A (partial) assignment gives each variable in (a subset of) $\mathcal{V}$ the value *true*($\top$) or *false*($\bot$). Applying a function f to an assignment s, denoted as $f(s)$, follows the usual semantics. If s is a total assignment, $f(s)$ yields a truth value; if s is partial, $f(s)$ results in a formula not dependent on any variables in s. By *dependent* we mean syntactically dependent, i.e. no variable v in s appears in $f(s)$. For a partial assignment, we use *extension* to refer to an assignment that assigns more variables and is otherwise the same. For sets of Boolean variables $\mathcal{U}$ and $\mathcal{V}$, we denote union as $\mathcal{U}, \mathcal{V}$ and as $\mathcal{U} \dot{\cup} \mathcal{V}$ when we want to emphasize that $\mathcal{U}$ and $\mathcal{V}$ are disjoint. For next-state variables, we write $\mathcal{V}_1$ to denote a copy of $\mathcal{V}$, and for symmetry we then refer to the original copy $\mathcal{V}$ as $\mathcal{V}_0$. We extend this notation to any number of transitions.

We consider hardware designs modeled as finite logical circuits [3]. Each circuit is associated with a safety property and an environment constraint encoded

as Boolean formulas over input and latch variables. A state of the system is an assignment to inputs and latches satisfying the constraint. The values of the latches are initialized using their reset function and evolve according to the transition functions. At every timestamp, the values of latches are determined by the values of inputs and the previous latch values. The inputs can have arbitrary values and represent non-determinism.

Definition 1 (Circuit). *A circuit $M = (I, L, R, F, P, C)$ is defined as a tuple consisting of the following attributes:*

1. *I: a finite ordered set of Boolean input variables;*
2. *L: a finite ordered set of Boolean latch variables;*
3. *$R = \{r_l(I, L) \mid l \in L\}$, where $r_l(I, L)$ is a reset function for latch l;*
4. *$F = \{f_l(I, L) \mid l \in L\}$, where $f_l(I, L)$ is a transition function for latch l;*
5. *$P(I, L)$ represents the set of good states; and*
6. *$C(I, L)$ encodes the set of constrained states.*

In this paper, we focus on safety properties, which should hold in all reachable states. For circuits that do not come with explicitly given constraints, we simply consider $C(I, L) = \top$. For multiple constraints $c_0, c_1, \ldots$, we take their conjunction to form a single one $C(I, L) = \bigwedge_i c_i$. The set of initial states is defined by $R\{L\} = \bigwedge_{l \in L}(l \leftrightarrow r_l(I, L))$. The same notation is used for transitions $F_{0,1}\{L\} = \bigwedge_{\ell \in L}(\ell_1 \leftrightarrow f_\ell(I_0, L_0))$. Both are also used for subsets of L. The convention to use indices on formulas, while omitting explicit variable references, extends to other circuit components, e.g., C_0 stands for $C(I_0, L_0)$.

When designing a certificate format, one of the key objectives is to eliminate the need for quantifiers in certificate checking, allowing the process to fall within the co-NP complexity class and admits the use of SAT solvers. For this purpose, the authors of [12] introduced the concept of *stratified* reset functions, i.e., the reset functions of a circuit do not have cyclic dependencies. This entails that the formula $R\{L\}$ is always satisfiable.

A circuit is said to be safe if the property P holds in all states reachable from the initial states without violating the constraint. Therefore a counterexample is a path $s_0, s_1, \ldots, s_n$ from a reset state s_0 to a bad state s_n where all s_i satisfy the constraint:

$$R_0\{L\} \;\wedge \bigwedge_{i \in [0,n)} F_{i,i+1}\{L\} \;\wedge \bigwedge_{i \in [0,n]} C_i \;\wedge\; \neg P_n.$$

Example 1. A common problem in hardware design is to arbitrate the usage of shared resources. In the following, we illustrate with such an example, described in Fig. 2. Each user i can send a request for the shared resources by setting the input signal req_i and gains access from the acknowledge signal ack_i. The crucial property is that no two users get acknowledged at the same time. The internal design of the arbiter can be quite complex when additional properties, such as fairness, are considered. To make the model checking task easier, we specify the constraint as $C = \bigwedge_{i \neq j} \neg(ack_i \wedge ack_j)$. This constraint can either be extracted

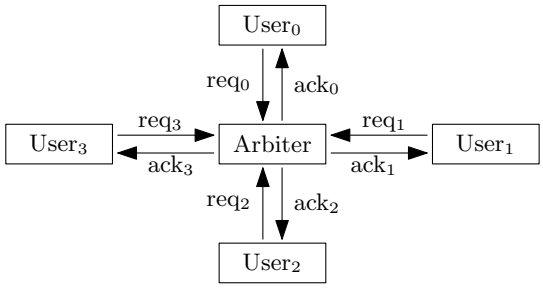

Fig. 2: An arbiter for four users.

by the model checker in a preprocessing step (we later discuss this in Sec. 3), or it can be specified explicitly by the hardware designer. In both cases, the implementation details of the arbiter may be abstracted away from the design.

Counterexamples are witnesses for validating model checking failures. On the other hand, when model checking succeeds, we rely on a certificate, i.e., a witness circuit, which is the format used in the hardware model checking competition.

Definition 2 (Witness Circuit). *$W = (I', L', R', F', P', C')$ is a witness circuit of circuit $M = (I, L, R, F, P, C)$, both with constraints, if R' is stratified and for $K = L \cap L'$:*

1. *Reset:* $R\{K\} \wedge C \rightarrow R'\{K\} \wedge C'$;
2. *Transition:* $F_{0,1}\{K\} \wedge C_0 \wedge C_1 \wedge C'_0 \rightarrow F'_{0,1}\{K\} \wedge C'_1$;
3. *Property:* $(C \wedge C') \rightarrow (P' \rightarrow P)$;
4. *Base:* $R'\{L'\} \wedge C' \rightarrow P'$;
5. *Step:* $P'_0 \wedge F'_{0,1}\{L'\} \wedge C'_0 \wedge C'_1 \rightarrow P'_1$.

Each condition defined above is encoded as a SAT formula. Additionally, a polynomial-time check that R' is stratified [12] is needed, such that there are no cyclic dependencies among the reset functions.

3 Certifying Constraint Extraction

Constraint extraction is a effective preprocessing step that simplifies verification for any base model checking engine [16, 17], such as k-induction. In this section, we address the challenge of certifying the hidden constraints uncovered by this process, as illustrated in Fig. 1. Because a witness circuit generated by the base engine only proves correctness on the preprocessed circuit, it does not immediately satisfy Def. 2. Consequently, postprocessing the witness circuit is necessary to produce a valid certificate for the original problem [23].

3.1 Model Constraints

The first type of constraints we consider is *model constraints*. Intuitively, they are those that hold in all reachable states, like a safety property. It is therefore evident that adding them does not change the set of reachable states in the model. In industrial practice, it is common to verify hundreds of properties for the same design, and it is beneficial for verification performance to add already proven properties as constraints [24, 25].

Definition 3 (Model constraint). *Given a circuit $M = (I, L, R, F, P, C)$, the formula $D(I, L)$ is said to be a model constraint if it holds in all reachable states, i.e., the following formula is unsatisfiable for any n:*

$$R_0\{L\} \wedge \bigwedge_{i\in[0,n)} F_{i,i+1}\{L\} \wedge \bigwedge_{i\in[0,n]} C_i \wedge \neg D_n.$$

By Def. 3, during model checking, the extracted model constraints can simply be appended to strengthen C to form a new constraint $C \wedge D$. A base model checker that uses for instance IC3/PDR [26] or k-induction can then be employed. Once verification is completed, the base engine provides a witness circuit. Next, we are going to show how to postprocess such a witness circuit to obtain one that certifies the safety of the original circuit. For that we are going to compose it with another witness circuit for the model constraint D.

In the following we assume inputs and latches not shared with the model to be unique to the respective witness circuits. Otherwise renaming is necessary.

Theorem 1. *Let $M = (I, L, R, F, P, C)$ be a model with constraint D, and let $M_P = (I, L, R, F, P, C \wedge D)$ be the constrained model with witness $W_P = (I^P, L^P, R^P, F^P, P^P, C^P)$. Let $W_D = (I^D, L^D, R^D, F^D, D^D, C^D)$ be a witness certifying that D is a model constraint, i.e., a witness for $M_D = (I, L, R, F, D, C)$. If the inputs and latches of W_P and W_D are disjoint except for those shared with M, then the combined witness circuit W defined as follows is a witness for M.*

$$W = (I^P \cup I^D,\ L^P \cup L^D,\ R^P \cup R^D,\ F^P \cup F^D,\ P^P \wedge P^D,\ C^P \wedge C^D \wedge D)$$

Proof. Before we show that W passes every check of Def. 2, we note that W_P and W_D not sharing any additional inputs or latches implies that $R^P \cup R^D$ is still stratified. Further, given the reset check for both W_P and W_D the reset states in W are simply: $(R^P \cup R^D)\{L^P \cup L^D\} = R^P\{L^P\} \wedge R^D\{L^D\}$. The same is true for encoding transitions. We begin with the reset check and show that the premise implies D by the witness relation between M_D and W_D:

$$R\{K\} \wedge C \Rightarrow R^D\{K\} \wedge C^D \Rightarrow R^D\{L^D\} \wedge C^D \Rightarrow P^D \Rightarrow D$$

Now the premise for the reset check for both W_P and W_D are met, and the conclusion of the reset check for W follows. The premise of the transition check of W gives us D_0, which by the transition check of W_D gives $F^D \wedge C_1^D$ and further D_1 using the step check. Again the premises for the step check of both W_P and

W_D are fulfilled. For the remaining three checks, no additional arguments are needed as the conclusions are the same as in the witness checks for W_P and W_D and the premises have only been strengthened.

Another interesting application of the described construction is the efficient certification of multi-property circuits. Consider a model $M = (I, L, R, F, P, C)$, where $P = \bigwedge_{i \in [0,n]} P^i$, and each P^i typically concerns only a sub-circuit of the original. A model checker may verify these properties individually and later compose the corresponding witness circuits using the same construction as above. In this case both witnesses are produced without the addition of an explicit model constraint and the D in the constraint definition of W is simply true. Any number of witness circuits may be composed through repeated application of the construction, corresponding to a union over all witnesses at each component.

3.2 Inductive Constraints

Inductive constraints [16] can be used to strengthen existing constraints in a model, partially motivated by their use in backward reachability [27].

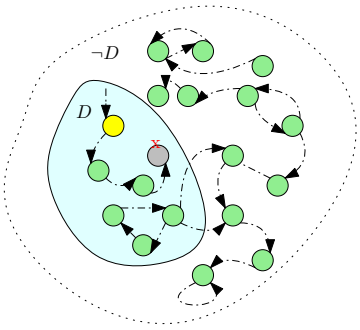

Fig. 3: An illustration of the state space for inductive constraints. The bad states are marked gray; and the initial state is marked yellow. The blue region is the set of states satisfying the constraint D. Note that D is of course not a subset of $\neg D$.

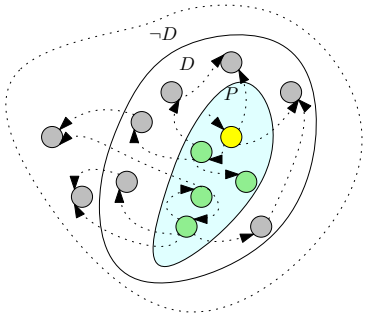

Fig. 4: An illustration of property constraints. Since the property constraints act as conditions in the transition function, outgoing transitions only take place in states that satisfy D. There is no transition from states in $\neg D$ to D.

Definition 4 (Inductive constraint). *An inductive constraint $D(I, L)$ of a circuit $M = (I, L, R, F, P, C)$ satisfies:*

(1) $F_{0,1}\{L\} \wedge \neg D_0 \rightarrow \neg D_1$; *and* *(2)* $\neg D \rightarrow P$.

We illustrate inductive constraints in Fig. 3. The soundness of adding inductive constraints follows from the fact that these constraints do not eliminate any bad traces of the original model; therefore, the safety property is preserved. Once a trace leaves D it cannot cross back, since $\neg D$ is inductive, and no bad state can be reached with $\neg D \rightarrow P$. Therefore the model checker only needs to

ensure P holds in the states that satisfy D (i.e., the area marked blue). Note the slightly confusing naming of *inductive* constraints taken from [16]; it is the negation of D that is inductive, and moreover it is not an inductive invariant, as it does not necessarily hold in the reset states.

We now show how to produce witness circuits for models with added inductive constraints. The following theorem requires the constraint to also be inductive with respect to the transition function of the witness under constraint W_D. This does not impose a practical limitation, since D only depends on inputs and latches from the original circuit and witness circuits typically do not introduce additional transitions for original latches. If this does happen to be the case, F' can be restricted to the original transitions without changing the success of the transition or step check in Def. 2.

To simplify the presentation, we denote a modified set of reset and transition functions like this $R' \vee \neg D$ and $F' \vee \neg D_1$. This construction can be easily achieved by introducing a fresh input i_l for each latch l, and encoding the formula D over these inputs to obtain D_I. With this setup, the new reset function for each latch l is defined as: $\texttt{ite}(D_I, r'_l, i_l)$, which selects r'_l when D_I holds, and otherwise takes the value of i_l. The transition function $\texttt{ite}(D_I, f'_l, i_l)$, similarly allows to transition to any state violating D. Since R' is stratified and D_I depends only on inputs, the resulting reset function remains stratified.

We prove the correctness of our construction in the following theorem.

Theorem 2. *Given model $M = (I, L, R, F, P, C)$, the constrained model $M_D = (I, L, R, F, P, C \wedge D)$ with witness $W_D = (I', L', R', F', P', C')$, where D is an inductive constraint in M and $F' \wedge \neg D_0 \rightarrow \neg D_1$, the unconstrained witness $W = (I', L', R' \vee \neg D, F' \vee \neg D_1, P' \vee \neg D, C' \vee \neg D)$ is a witness for M.*

Proof. We consider an arbitrary assignment to the formula $L_0 \cup L'_0 \cup I \cup I' \cup L_1 \cup L'_1$. If it satisfies $\neg D$, then it also satisfies P and the property, reset and base check are trivially fulfilled. Similarly, with the assumption $\neg D_1$ the transition and step check hold. If D holds, any violation of the property, reset or base check would also be a violation for the witness relation of M_D and W_D. Since $\neg D$ is inductive under F and F', D_1 implies D_0 if the assignment violates either the transition or step check. In both cases, it would also contradict the witness relation of M_D and W_D.

3.3 Property Constraints

We now take a look at another class of constraints, namely property constraints, that can be generated from a model by a dedicated constraint mining algorithm. Intuitively, property constraints are implied by the property itself.

Definition 5 (Property constraint). *A property constraint $D(I, L)$ of a circuit $M = (I, L, R, F, P, C)$ satisfies: $P \rightarrow D$.*

Once property constraints are extracted, their use differs from the previous two constraint types, as both model constraints and inductive constraints are

directly appended to the explicit constraint, i.e., $C \wedge D$. Property constraints, on the other hand, need to be integrated into the transition functions.

Model checking under property constraints. Given a circuit $M = (I, L, R, F, P, C)$ with property constraint D. The circuit under the property constraint is $M_D = (I, L, R, F_D, P, C)$ where: $F_D = \{\texttt{ite}(D_0, f_l, l) \mid l \in L\}$.

This suggests that at every step, if the property constraint holds in the current state, then it follows the same transition as before, otherwise it stays in the current state. We illustrate in Fig. 4 that adding property constraints does not affect the model checking result. In other words, if the circuit under property constraints is proven to be safe, then this implies the original circuit is also safe. This can be validated by a certificate, constructed as follows.

Theorem 3. *Given model $M = (I, L, R, F, P, C)$, the model under property constraint $M_D = (I, L, R, F_D, P, C)$ with witness $W_D = (I', L', R', F'_D, P', C'_D)$, the witness circuit for property constraints $W' = (I', L', R', F', P', C')$, where $F' = \{ite(D_0, f_l'^D, f_l) \mid l \in K\} \cup \{f_l'^D \mid l \in L' \setminus K\}$ and $C' = C'_D \wedge D$, is a witness for M.*

Proof. Given the witness relation between M_D and W_D, we need to prove that W is a witness for M. First we conclude that D holds in any reset of M valid under the constraint:

$$R\{K\} \wedge C \Rightarrow R'\{K'\} \wedge C'_D \Rightarrow R'\{L'\} \wedge C'_D \Rightarrow P' \Rightarrow P \Rightarrow D$$

Now, we assume an assignment s to $L_0 \cup L'_0 \cup I \cup I' \cup L_1 \cup L'_1$ violating any of the witness checks outlined in Def. 2. If D_0 does not hold, the reset check is fulfilled by the above argument. The other checks are trivially fulfilled as C'_0 is false. If D_0 holds in s, the assignment will also violate the same check for the witness relation between M_D and W_D.

4 Relaxed simulation

According to Def. 2, in order for the Reset and Transition checks to pass, the constraint C' needs to hold for all possible extensions generated by the reset and transition functions. One solution is to ensure that the constraint only contains variables shared between the two circuits, which is a natural assumption. For many model checking algorithms such as IC3/PDR, it is often the case that the certificate W is simply $W = (I, L, R, F, P', C)$ [28], where P' is the inductive strengthening generated by the algorithm itself, encoding an over-approximation of the reachable states, with the rest of the circuit unchanged from M.

However, for other algorithms where it is necessary to constrain new variables in the witness circuit as well as to allow cyclic reset definitions, the format in Def. 2 has to be relaxed to include quantifiers in the checks. We therefore present, in the following, a more general certificate format. While this generalized format is inherently more expensive to check due to the presence of quantifiers, we expect the certificate checker to fall back to the quantifier-free checks of Def. 2 whenever possible, in order to retain efficient certificate validation.

Definition 6 (Quantified Reset and Transition). *Consider the given circuit* $M = (I, L, R, F, P, C)$*, then the circuit* W *with* $W = (I', L', R', F', P', C')$ *is a witness circuit for the local witness variables* $X' = (I' \cup L') \setminus (I \cup L)$ *if it satisfies Def. 2, with the* Reset *and* Transition *conditions relaxed as follows:*

- *Reset*$^{\exists}$*:* $\quad R\{L\} \wedge C \rightarrow \exists X' \, [R'\{L'\} \wedge C']$*;*
- *Transition*$^{\exists}$*:* $\quad F_{0,1}\{L\} \wedge C_0 \wedge C_1 \wedge C'_0 \rightarrow \exists X'_1 \, [F'_{0,1}\{L'\} \wedge C'_1]$.

If Reset$^{\exists}$ *is not used,* R' *has to be stratified.*

If the reset functions of W are cyclic, it is necessary to use the Reset$^{\exists}$ check. Without this existential quantification, it is not sufficient to prove the soundness theorem of Def. 2: *if* W *is a witness circuit of* M*, then* M *is safe.* Proving this theorem requires establishing a simulation relation between M and W, which in turn requires every reset state in M to correspond to a reset state in W.

Since the Reset$^{\exists}$ condition directly guarantees that any reset state in M aligns with a reset state in W, we no longer need to impose the stratification requirement on W.

In the following, we formally prove the soundness of this format.

Theorem 4. *Given a circuit* M *and its witness circuit* W *that satisfies Def. 6, then* M *is safe.*

Proof. We assume that W is a witness circuit for M, and do a proof by contradiction. Suppose M is not safe. Then there is a bad trace of some finite length n in the form of an assignment to $n + 1$ copies of $I \cup L$ satisfying:

$$R_0\{L\} \wedge C_0 \wedge F_{0,1}\{L\} \wedge C_1 \ldots C_{n-1} \wedge F_{n-1,n}\{L\} \wedge C_n \wedge \neg P_n.$$

We extend this assignment to each copy of the variables in X' that satisfies:

$$R'_0\{L'\} \wedge C'_0 \wedge F'_{0,1}\{L'\} \wedge C'_1 \ldots C'_{n-1} \wedge F'_{n-1,n}\{L'\} \wedge C'_n \wedge \neg P'_n.$$

The Reset$^{\exists}$ and Transition$^{\exists}$ checks directly imply the existence of a reset state in W, respectively a successor state. Since the Transition$^{\exists}$ check only quantifies over the next-state version of the extension variables X'_1 the trace in W can be constructed iteratively from reset to bad state. Applying the same argument n times yields an assignment to $(X')^n$ satisfying $F'_{i,i+1}\{L\}$ for $i \in [0, n)$ and C_i for $i \in [0, n]$. Lastly, the property check guarantees $\neg P'_n$, giving us the desired assignment. However, the Base and Step check together ensure that the property P' holds on all reachable states of W, thus contradicting the initial assumption that a bad trace exists in M.

5 Extending Circuits with Oracles

The model checking technique k-induction checks (1) k consecutive states from any initial state are safe; and (2) if any k consecutive states are safe, then the subsequent $(k+1)$-th state is also safe. Even though certifying k-induction has been

studied before [3, 29–31], none of these works address *uniqueness constraints*, which requires in the second check that the sequence of k consecutive states to be unique. The main difficulty of certifying uniqueness constraints comes from the requirement of accounting for multiple possible execution paths for the inductiveness. It became neccessary for us to further relax the proof format definition. We therefore in the following define witness circuits with *oracles*. Oracles are just like inputs and can be assigned arbitrary values. However, we distinguish oracles from inputs so that they can be explicitly identified and correctly handled in the certificate checks.

Definition 7 (Step check with oracles). *Consider the circuit M with $M = (I, L, R, F, P, C)$, then the circuit W with $W = (I' \dot\cup O', L', R', F', P', C')$ where O' are the oracle inputs. W is a witness if R' and F' are independent of O' and the circuits satisfy the conditions in Def. 2, where the step check is relaxed to:*

$$\mathit{Step}^{\exists} : \forall O'_0 \, [P'_0 \wedge F'_{0,1}\{L'\} \wedge C'_0] \wedge C'_1 \rightarrow P'_1$$

If R' or F' are dependent on O', we need additional quantifiers in the respective checks similar to Def. 6.

Definition 8 (Reset and Transition with oracles). *Given a circuit M, the circuit W, with $M = (I, L, R, F, P, C)$, $W = (I' \dot\cup O', L', R', F', P', C')$, and $X' = (I' \cup L') \setminus (I \cup L)$, is a witness circuit if it meets Def. 2, where the step check has been relaxed to $\mathit{Step}^{\exists}$ from Def. 7 and either or both reset and transition checks have been relaxed to:*

- $\mathit{Reset}^{\exists\forall}$: $\quad R\{L\} \wedge C \rightarrow \exists X' \forall O' \, [R'\{L'\} \wedge C']$;
- $\mathit{Transition}^{\exists\forall}$: $\quad F_{0,1}\{L\} \wedge C_0 \wedge C_1 \wedge C'_0 \rightarrow \exists X'_1 \forall O'_1 \, [F'_{0,1}\{L'\} \wedge C'_1]$.

Theorem 5. *A witness circuit that satisfies the certificate format with oracles is a valid certificate.*

Proof. Since the oracles are only in the witness, the Reset$^{\exists\forall}$ and Transition$^{\exists\forall}$ conditions still allow us to construct the trace in W the same way as in the proof of Theorem 4. The same is true if R' or F' are syntactically independent of O' and the quantifier-free checks from Def. 2 are used. What is left to show is that no bad state is reachable in W. We will be considering the equivalence classes induced by O' on the state space of W, i.e., states only differing in O' are in the same equivalence class. Assume there exists a bad trace $s_0, \ldots, s_n$ in W. By the reset check, all states in the equivalence class of s_0 also satisfy R', and by the base check are guaranteed to be good. By applying the transition check to all states in the equivalence class of s_0, we know they each have a transition to all states in the equivalence class of s_1. This allows us to apply the step condition to all states in the equivalence class of s_1, concluding that they are all good. This argument can be applied n times to show that all states in the equivalence class of s_n are good, including s_n itself.

This more complex certificate is, in fact, hard for the second level of the polynomial hierarchy.

Theorem 6. *A closed formula $\forall A \exists E f$ is true exactly if the witness circuit $W = (I', L', R', F', P', C')$ passes the Step$^\exists$ check, where $I' = A$, $O' = E$, $L' = \{\ell\}$, $r'_\ell = \top$, $f'_\ell = \top$, $C' = \top$, and $P' = \neg f \wedge \neg \ell$.*

Proof. In the Step$^\exists$ check, F' is simply ℓ_1, which implies $\neg P'_1$. With C constant true, O'_0 quantifies over $\neg f_0 \wedge \neg \ell_0$. Since ℓ_0 is independent of O'_0, we can leave it behind when moving the rest to the other side of the implication. Omitting the universal quantifiers for unused variables, we are left with: $\forall \ell_0 \forall \ell_1 \forall A_0 \, [\neg \ell_0 \wedge \ell_1 \rightarrow \exists E_0 \, [f_0]]$, which is trivially true for all valuations of the first two quantifiers, except for one where it simplifies to $\forall A_0 \exists E_0 f_0$.

5.1 Uniqueness Constraints

In the following, we present certificate generation for uniqueness constraints. We begin by providing the definition of k-induction under uniqueness constraint.

Definition 9. *A property P of a given circuit $M = (I, L, R, F, P, C)$ is said to be k-inductive under uniqueness constraints iff the following holds:*

$$R_0\{L\} \wedge \Big(\bigwedge_{i \in [0,k-1)} F_{i,i+1}\{L\} \Big) \rightarrow \Big(\bigwedge_{i \in [0,k)} \big(\bigwedge_{j \in [0,i]} C_j \big) \rightarrow P_i \Big) \text{ and}$$

$$\Big(\bigwedge_{i \in [0,k)} F_{i,i+1}\{L\} \Big) \wedge \Big(\bigwedge_{i \in [0,k)} C_i \wedge P_i \Big) \wedge \mathit{unique}_k \rightarrow (C_k \rightarrow P_k),$$

$$\text{where } \mathit{unique}_k = \bigwedge_{0 \le i < j < k} (I_i \not\leftrightarrows I_j) \vee (L_i \not\leftrightarrows L_j).$$

The first formula, called the initiation check, specifies that the property holds for k steps from the initial states. The second formula is an inductive check such that if the property holds for k steps then it also holds at the $(k+1)$-th step. The uniqueness constraint ensures that the inductive step only considers k consecutive states that are all distinct from one another.

Next we present our certificate construction for k-induction under uniqueness constraints. An example for an intuitive understanding of our certificate construction is in Fig. 5. The construction relies on the following idea. A k-inductive property can be strengthened to be inductive: for every bad state, all predecessors up to k steps back are labelled as bad states too.

Definition 10 (k-witness circuit under uniqueness constraints). *Given a circuit $M = (I, L, R, F, P, C)$ with $k \in \mathbb{N}^+$. The k-witness circuit under uniqueness constraints is defined as $W = (I \dot\cup O', L, R, F, P', C)$, where $O' = \bigcup_{i \in [0,k)} I^i$ and $P' = \bigwedge_{i \in [0,k)} [v^i \rightarrow P(I^i, L^i)]$, with $v^0 = C(I, L)$, $v^i = v^{i-1} \wedge C(I^i, L^i)$, $L^0 = L$ and $L^{i+1} = \{f_l(I^i, L^i) \mid l \in L\}$.*

In the above definition, $L^1, \ldots, L^{k-1}$ are not necessarily latches: we directly use the output of the transition functions, which are simply AND-gates in practice.

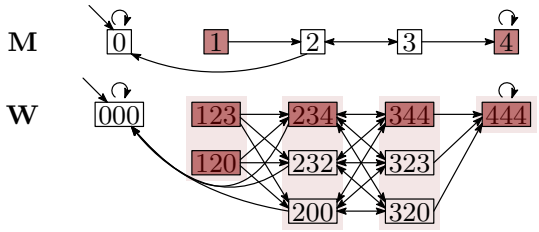

Fig. 5: Example of the k-witness construction for certifying k-induction under uniqueness constraints. The figure displays the state space of a circuit M, which is 3-inductive under uniqueness constraints, alongside its corresponding k-witness circuit W (Def. 10). Bad states are marked in red, state 0 is the only initial state. Witness states that differ only by the value of O' are grouped vertically. If at least one state in a group violates the property, the entire group is highlighted with a light red background.

The variables v^i keep track of the depth up to which the path along the L^i remains valid; that is, only traverses states satisfying the constraint. The latches are the same in the original and the k-witness circuit, however, the k-witness circuit has more bad states, as it requires all states reachable in $k-1$ steps to be good. Note that by construction the reset and transition functions are independent of the oracles, thus the Reset$^{\exists\forall}$ and Transition$^{\exists\forall}$ checks are not necessary and the quantifier-free versions should be used.

Even though Def. 10 is also a valid construction for k-induction *without* uniqueness constraints, it is expensive for the certificate checker to perform QBF checks. Hence in the case where uniqueness constraints are not required, the witness circuit construction should still follow the construction outlined in [12]. Their approach produces inductive witnesses by recording a history of k states rather than precomputing a possible future. While this method avoids the quantification, it would violate the transition check, since the witness is not allowed to transition to a state already in the history while the correctness of the construction relies on the model being inductive under uniqueness constraints.

Theorem 7. *Given a circuit $M = (I, L, R, F, P, C)$ with some $k \in \mathbb{N}^+$, and a k-witness circuit $W = (I \dot{\cup} O', L, R, F, P', C)$. If M is k-inductive under uniqueness constraints, then W is a valid witness circuit for M.*

Proof. Since L, R, F and C remain unchanged, the reset and transition conditions hold trivially. Given C, the new property P' clearly implies P, thus the property check is satisfied. For the base check we note that if an initial state of the witness violates P', a bad state in the original circuit can be reached in k steps or less. It is left to show that if the Step$^{\exists}$ check fails, the original circuit is not k-inductive. Assume two consecutive states u' and v' in C' for which the Step$^{\exists}$ check fails. Let u and v be the states in M induced by the assignment to L in u' and v' respectively. The shortest path from v to a bad state has exactly k states, all are different, satisfy the constraint, and all but the last are good. If

	A	O	Σ	$\cup$	Σ_P	$\cup_P$	Σ_F	$\cup_F$
Mean (39)	25411	64	777.20	37.18	36.23	30.77	660.37	4.15
picorv32 (8)	54042	201	3455.78	144.77	107.99	128.20	3037.94	14.89
mentor (1)	31613	36	505.96	38.70	23.40	25.53	440.87	12.37
dspfilters (17)	28397	49	120.07	9.45	29.66	7.58	58.22	1.22
sm98 (3)	5126	2	25.17	21.35	1.56	1.16	4.83	2.38
zipcpu (7)	2946	2	3.46	2.57	1.49	1.46	0.54	0.23
zipversa (3)	2775	3	5.92	3.44	2.17	2.13	0.70	0.26

Table 1: The HWMCC set contains 39 multi-property benchmarks, split into 7 families, each with the same number of properties (B). The other columns display the number of and-gates (A), total time taken in seconds for checking individual witness circuits for a single property (Σ), and time taken for checking the composed witness circuit ($\cup$). Additionally, the table presents the time taken up just by the Property check for the individual witnesses (Σ_P) and the composed witness ($\cup_P$), which in both cases has to be done B times. As well as the Transition check, which has to be done for each individual witness (Σ_F) but only once for the composed witness ($\cup_F$). This is the same for every other check, but the transition check is the most complex on this set of benchmarks. Each row presents the mean over the benchmarks in the family (number given in parentheses). The mean over all benchmarks is presented at the top.

the number of steps was less, P' would not hold in u', and if it was any more, P' would hold in v'. The other two claims follow from the path being the shortest. Since u is guaranteed to satisfy C and be different from the bad state, prepending the path with u yields a path in M consisting of k unique good states followed by a bad state, thus M is not k-inductive under uniqueness constraints.

6 Experimental Evaluation

We implemented the proposed certificate format in our certificate checker CERTIFAIGER[6], together with the relaxed checks that require quantifiers. Note that in all HWMCC benchmarks, all reset functions are stratified, as the standard AIGER format used in the competitions only supports stratified resets. We also extended the open-source k-induction-based model checker MCAIGER [32] to generate k-witness circuits as defined in Def. 10 when uniqueness constraints are enabled. Additionally, we implement our tool AIGMERGE[7] that uses the construction described in Theorem 1 to compose arbitrary circuits.

[6] https://github.com/Froleyks/certifaiger

[7] AIGMERGE may eventually be added to the set of AIGER utilities at https://github.com/arminbiere/aiger.

CERTIFAIGER uses KISSAT [33] for SAT checks and QUABS [34] for quantified checks. All input circuits are in the AIGER 1.9 format. Each certificate check is generated as combinatorial circuits in either AIGER (for SAT checks) or QAIGER (for QBF checks) and is then translated to CNF or the circuit-based QCIR format [35].

	t_{MC} (s)	t_{Cert} (s)	t_{SAT} (s)	k	I	A
bobcount	426.68	26.86	2.40	76	3	77
eijks298	383.05	2192.33	9.77	138	3	225
pdtvispeterson	6.29	249.35	2.76	59	2	700
pdtvisvending00	14.25	to	2.37	35	2	959
pdtvisvending02	199.05	to	2.03	33	2	958
pdtvisvending05	3.13	26663.62	1.70	28	2	951
pdtvisvending07	3.41	22117.90	2.34	29	2	952
pdtvisvending08	117.81	to	2.07	33	2	950

Table 2: In the HWMCC'10 benchmark set, 8 additional instances are solved with uniqueness constraints. We list the model checking (t_{MC}) and certification time (t_{Cert}). The table also presents the portion of the certification time taken for the 4 SAT checks (t_{SAT}). The last three columns denote the inductive depth k, the number of input variables I, and gates A respectively.

The goal of our first experiment is to evaluate the certification method for composing witness circuits, which also provides insight into certifying extracted constraints. Although we could not find an open-source model checker that implements explicit constraint extraction, the concept of model constraints naturally arises in circuits where multiple properties are of interest. Therefore, we focus on multi-property benchmarks that are present in HWMCC'12 and HWMCC'19. We first ran our certifying model checker VOIRAIG for each property individually, which left us with one witness circuit per property that could be proven. We then used AIGMERGE to combine all the witness circuits for an individual model into one. Certificate checking for such a certificate proceeds as follows: verify the reset, transition, base, and step condition for the composed witness according to Def. 2. For each original property run the property check. We compare the total time of these checks to the total time it takes to certify the original witness circuits for each property individually. We present the results obtained in Table 1. As can be seen, directly verifying the composed witness circuit is significantly more efficient than verifying individual witness circuits for each property. Interestingly, the transition check appears to be the bottleneck during the certification process, which differs from the experimental results in [36], where the step check always takes significantly longer.

In the second experiment, we study the effectiveness of our certification method for k-induction under uniqueness constraints. We selected the bench-

marks that require uniqueness constraints from HWMCC'10. Results are summarized in Table 2. We used a timeout of 50 000 seconds for certificate checking. Despite the QBF check creating a bottleneck in certification performance, our certifier successfully validated 5 out of 8 instances. The QBF solver QuAbs failed to complete the $\text{Step}^{\exists}$ checks for three instances within the time limit, whereas for the same instances the rest of the checks were solved in less than 10 seconds (see t_{SAT} in the table). While QBF solving is inherently more challenging than SAT solving, we attribute the substantial performance gap in part to recent advances in SAT solving, particularly the circuit-specific optimizations introduced in Kissat [37]. We believe that incorporating similar optimizations into QuAbs could significantly improve the efficiency of QBF-based certificate checking. Exploring such enhancements is an important direction for future work.

7 Conclusion

Recent hardware model checking competitions have employed a standardized certification approach, relying on simple SAT-based certificate validation. This approach has proven highly effective, with certification overhead constituting only a small fraction of the total verification effort. In this work, we introduce certification generation techniques tailored to both explicit and implicitly extracted constraints for three different constraint classes. Furthermore, we present alternative QBF-based checks to replace pure SAT checks in the certificate format, addressing the challenges of certificate generation posed by intricate reset logic and complex encodings in model checking. Extending this, we also demonstrate certification generation for k-induction under uniqueness constraints. The empirical evaluation, conducted on a range of competition benchmarks, demonstrates the effectiveness and practical relevance of our method. Looking forward, we plan to incorporate the remaining techniques in bit-level hardware model checking, such as retiming [38] and localization. Furthermore, we aim to broaden the scope of our certification method to infinite-state systems to support program verification as well as exploring the certification of security properties.

Acknowledgement

Partially funded by the European Union (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, the Research Council of Finland under the project 369068, and a gift from Intel Corporation.

References

1. Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann, and Andreas Stahlbauer. Witness validation and stepwise testification across software verifiers.

In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 721–733, 2015.
2. Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. volume 2014, 2014.
3. Emily Yu, Armin Biere, and Keijo Heljanko. Progress in certifying hardware model checking results. In *CAV (2)*, volume 12760 of *Lecture Notes in Computer Science*, pages 363–386. Springer, 2021.
4. Alberto Griggio, Marco Roveri, and Stefano Tonetta. Certifying proofs for SAT-based model checking. *Formal Methods Syst. Des.*, 57(2):178–210, 2021.
5. Kedar S. Namjoshi. Certifying model checkers. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2001.
6. Matt Kaufmann, Andrew Martin, and Carl Pixley. Design constraints in symbolic model checking. In *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 477–487. Springer, 1998.
7. Yoav Hollander, Matthew Morley, and Amos Noy. The e language: A fresh separation of concerns. In *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 38*, pages 41–50. IEEE, 2001.
8. Rajeev Alur and Thomas A Henzinger. Reactive modules. *Formal methods in system design*, 15:7–48, 1999.
9. Armin Biere, Keijo Heljanko, and Siert Wieringa. Aiger 1.9 and beyond. 2011.
10. Armin Biere, Nils Froleyks, and Mathias Preiner. Hardware model checking competition 2024. In *FMCAD*, pages 7–7. TU Wien Academic Press, 2024.
11. Nils Froleyks, Emily Yu, Mathias Preiner, Armin Biere, and Keijo Heljanko. Introducing certificates to the hardware model checking competition. In *CAV (1)*, volume 15931 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2025.
12. Emily Yu, Nils Froleyks, Armin Biere, and Keijo Heljanko. Stratified certification for k-induction. In *FMCAD*, pages 59–64. IEEE, 2022.
13. Sharad Malik. Analysis of cyclic combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(7):950–956, 1994.
14. J-HR Jiang, Alan Mishchenko, and Robert K Brayton. On breakable cyclic definitions. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 411–418. IEEE, 2004.
15. Marc D Riedel. *Cyclic combinational circuits*. California Institute of Technology, 2004.
16. Gianpiero Cabodi, Paolo Camurati, Luz Amanda Garcia, Marco Murciano, Sergio Nocco, and Stefano Quer. Speeding up model checking by exploiting explicit and hidden verification constraints. In *DATE*, pages 1686–1691. IEEE, 2009.
17. Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In *FMCAD*, pages 52–59. IEEE, 2012.
18. Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. Springer, 2010.
19. MS Jahanpour and OA Mohamed. Automatic generation of model checking properties and constraints from production based specification. In *The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS'04.*, volume 3, pages iii–435. IEEE, 2004.
20. Jun Yuan, Ken Albin, Adnan Aziz, and Carl Pixley. Constraint synthesis for environment modeling in functional verification. In *Proceedings of the 40th annual Design Automation Conference*, pages 296–299, 2003.

21. Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.
22. Zhengqi Emily Yu. Certifying hardware model checking/submitted by emily zhengqi yu. 2023.
23. Emily Yu, Nils Froleyks, Armin Biere, and Keijo Heljanko. Towards compositional hardware model checking certification. In *FMCAD*, pages 1–11. IEEE, 2023.
24. Eugene Goldberg, Matthias Güdemann, Daniel Kroening, and Rajdeep Mukherjee. Efficient verification of multi-property designs (the benefit of wrong assumptions). In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 43–48, 2018.
25. Sourav Das, Aritra Hazra, Pallab Dasgupta, Sudipta Kundu, and Himanshu Jain. Purse: Property ordering using runtime statistics for efficient multi - property verification. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2024.
26. Aaron R Bradley. Understanding IC3. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 1–14. Springer, 2012.
27. Per Bjesse and Koen Claessen. SAT-based verification without state space traversal. In *International Conference on Formal Methods in Computer-Aided Design*, pages 409–426. Springer, 2000.
28. Zhengqi Yu, Armin Biere, and Keijo Heljanko. Certifying hardware model checking results. In *International Conference on Formal Engineering Methods*, pages 498–502. Springer, 2019.
29. Arie Gurfinkel and Alexander Ivrii. K-induction without unrolling. In *FMCAD*, pages 148–155. IEEE, 2017.
30. Nikolaj S. Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II*, volume 9300 of *Lecture Notes in Computer Science*, pages 24–51. Springer, 2015.
31. Adrien Champion, Alain Mebsout, Christoph Sticksel, and Cesare Tinelli. The Kind 2 model checker. In *International Conference on Computer Aided Verification*, pages 510–517. Springer, 2016.
32. Armin Biere and Robert Brummayer. Consistency checking of all different constraints over bit-vectors within a SAT solver. In *2008 Formal Methods in Computer-Aided Design*, pages 1–4. IEEE, 2008.
33. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.
34. Leander Tentrup. Caqe and quabs: Abstraction based qbf solvers. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):155–210, 2019.
35. Charles Jordan, Will Klieber, and Martina Seidl. Non-CNF QBF solving with QCIR. In *AAAI Workshop: Beyond NP*, volume WS-16-05 of *AAAI Technical Report*. AAAI Press, 2016.
36. Nils Froleyks, Emily Yu, Armin Biere, and Keijo Heljanko. Certifying phase abstraction. In *IJCAR (1)*, volume 14739 of *Lecture Notes in Computer Science*, pages 284–303. Springer, 2024.

37. Armin Biere, Katalin Fazekas, Mathias Fleury, and Nils Froleyks. Clausal congruence closure. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.
38. Narendra Shenoy and Richard Rudell. Efficient implementation of retiming. *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*, pages 615–630, 2003.