Certificates for Model Checking under Constraints

Anonymous Author(s)*

Abstract

In order to simplify modeling and encoding, extend the applicability of model checking algorithms, as well as to boost their performance, it is common to make use of constraints — assumptions on the environment that restrict the state space. They are an explicit part of the AIGER format used in hardware model checking competitions (HWMCCs). In addition, existing works have shown benefits of extracting constraints as a preprocessing step to model checking. In this paper, we address the challenge of certifying model checking in the presence of constraints. We first present a certificate format for model checking under constraints, which relies solely on SAT for validation. To allow a more general class of constraints as well as more sophisticated reset logic, we provide alternative checks with one quantifier alternation. We proceed to provide a detailed construction for certifying constraint extraction. Finally, we propose a novel method to incorporate k-induction with simple paths constraints into our certification framework. We implement these approaches in our toolkit and evaluate them empirically on HWMCC benchmarks.

Keywords: Certification, Constraints, Model Checking, Hardware Verification

1 Introduction

In state-of-the-art hardware verification, constraints play a crucial role in enhancing the verification process. They are used to encode environmental behaviors of a system, acting as assumptions that the environment must satisfy. In hardware model checking, the goal is to verify whether a given specification, typically a safety property, holds within a model, under the condition that the constraints are satisfied. This helps gain confidence in the correctness of the design. Taking this a step further, certifying the model checking results requires generating a certificate that proves the correctness of the specification under the given constraints. While certification for model checking has been studied before [1, 11, 14, 17, 21], so far, to the best of our knowledge, it has not been discussed in the context of constraints.

In recent hardware model checking competitions (HWMCCs) [18], the latest benchmarks in the AIGER 1.9 format [6] all include constraints. These constraints, also called invariant constraints, are assumed to hold from the initial states up to the point where a bad state is encountered. Notably, the upcoming HWMCC'24 has announced the introduction of a certifying track, where it is mandatory for participating model checkers to provide certificates. While the certificate format from [12] generally works well for both preprocessing and base model checking techniques, it does not account for constraints. To certify benchmarks in the AIGER 1.9 format, it is necessary to extend the existing certificate format to incorporate constraints.

In addition to the explicitly defined constraints provided with the model, methods have been proposed to extract hidden constraints for both safety properties [8] and liveness properties [9]. This approach essentially serves as a preprocessing step before the model checking procedure, such that the validity of the property is preserved after adding additional constraints. By identifying constraints embedded in the design, the model checking problem is reduced to verifying only the subset of states that satisfy these constraints, thereby making the verification process more efficient. However, when a base model checker is called for the reduced model checking task, it only produces a certificate that proves the correctness of the reduced circuit after preprocessing. The challenge thus becomes certifying the correctness of the *original model*, which we address in this paper.

Along with the concept of constraints in general, uniqueness constraints, used in k-induction [19], enforce the loop-free condition, ensuring that all states are different within the induction step, turning k-induction into a complete model checking algorithm. These constraints also play a key role in bounded model checking, where they are used to compute the occurrence diameter [3]. Certifying k-induction under uniqueness constraints has been an open problem. This paper introduces further relaxed checks that require quantifiers but address this need. Finally, we propose a certification method for k-induction under uniqueness constraints.

Our contributions. In this work, we present a certificate format for certifying hardware model checking under constraints, which can be validated by an independent checker. We also tackle the problem of certifying extracted constraints by demonstrating how to build a valid certificate according to the defined format. The resulting certificate proves that the model checking result is correct for the original model

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

before preprocessing. Specifically, we consider three categories of constraints that can be mined from given models. Moreover, we address the challenge of certifying k-induction under uniqueness constraints and present a novel method for it. We experimentally evaluate our method on hardware model checking competition benchmarks.

The paper is organized as follows. Section 2 introduces our notation and the certificate format that we propose, together with formal proofs for its correctness. Section 3 presents certificate constructions for three classes of hidden constraints. In Section 4, we demonstrate how to certify uniqueness constraints. Finally, Section 5 presents experimental results for some of the presented techniques.

2 Constrained circuits

We employ the following notation: let $\mathcal{V}$ be a set of Boolean variables, we consider formulas over $\mathcal{V}$ with the Boolean operators $\neg, \vee, \wedge, \Rightarrow, \equiv$. The last denotes equivalence and can have an infix negation $\not\equiv$. A (partial) assignment gives each variable in (a subset of) $\mathcal{V}$ the value *true* or *false*. 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. We also say f under s. Here 'dependent' means syntactically dependent, i.e. no variable v in s appears in $f(s)$. A formula f *semantically* depends on a variable v if $f(v) \not\equiv f(\neg v)$. 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 write $\mathcal{U}, \mathcal{V}$ to denote their union. 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 [21]. 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. The inputs can have arbitrary values and model non-determinism.

Definition 2.1 (Circuit)**.** A circuit $M = (I, L, R, F, P, C)$ is defined as the following:

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 associated with a latch l.
4. $F = \{f_l(I, L) \mid l \in L\}$, where $f_l(I, L)$ is a transition function associated with a latch l.
5. $P(I, L)$ represents the set of good states.
6. $C(I, L)$ encodes the set of constrained states.

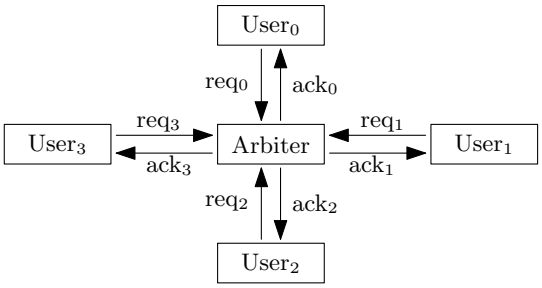

Figure 1. An arbiter for four users.

For circuits that do not come with explicitly given constraints, we simply consider $C(I, L) = \top$. In a scenario where there are multiple constraints $c_0, c_1, \ldots$, we take their conjunction to form a single one $C(I, L) = \bigwedge c_i$. A circuit is said to be *safe* if the property P holds in all constrained states reachable from the original states defined by R.

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. For this purpose, the authors of [22] 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 $R\{L\} = \bigwedge_{l \in L}(l \equiv r_l(I, L))$ is always satisfiable, which represents the set of initial states of the system. In what follows we fix the same notation $R\{\mathcal{V}\}$ when referring to a subset of latches $\mathcal{V} \subseteq L$. Similarly, for $\mathcal{V}_0 \subseteq L_0$ and $\mathcal{V}_1 \subseteq L_1$ of the same size (i.e., $|\mathcal{V}_0| = |\mathcal{V}_1|$), we write $F_{0,1}\{\mathcal{V}\} = \bigwedge_{i \in [0, |\mathcal{V}|)}(l_1^i \equiv f_{l^i}(I_0, L_0))$ where l^i and l_1^i are the i-th variable of $\mathcal{V}$ and $\mathcal{V}_1$ respectively. We also write C_0 $[P_0]$ for $C(\mathcal{V}_0)[P(\mathcal{V}_0)]$ to omit the explicit reference to variables.

Verifying safety under constraints. A circuit $M = (I, L, R, F, P, C)$ is safe, if P holds in all states reachable under the constraint. 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:

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

Example 2.2. A common problem in hardware design is to arbitrate the usage of shared resources. In the following, we illustrate with such an example, also described in Fig 1. Each user i can send a request for the shared resources 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 become quite complex when additional properties, such as fairness, are considered. To make the model checking task easier, the constraint $C = \bigwedge_{i \neq j} \neg(ack_i \wedge ack_j)$ can be used. This constraint can either be extracted by the model checker in a preprocessing step (we later discuss in Sec. 3), or it can be specified explicitly by the designer. In both cases, the implementation details of the arbiter might be abstracted away from the design.

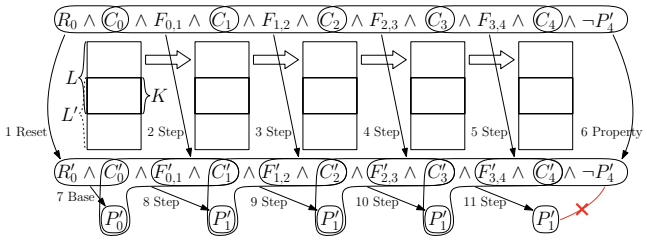

Figure 2. Assuming that a model M with a valid witness M' has a bad trace leads to a contradiction. Depicted are the overlapping sets of variables and the order in which the conditions of the witness check are used to construct the bad trace in M' and finally arrive at a contradiction.

We now give the definition of the certificate format, namely a *witness circuit*.

Definition 2.3 (Certificate format)**.** 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 holds:

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: $P'_0 \wedge F'_{0,1}\{L'\} \wedge C'_0 \wedge C'_1 \Rightarrow P'_1$.

The five conditions above are simple SAT checks. An additional polynomial check on M' that R' is stratified is also needed. If all of them pass, M' is a valid certificate for M that certifies its safety property. The first three conditions refer to both circuits and establish a simulation relation between the two, such that if M' is safe, M is also safe. The intuition is that an initial state in the original circuit M corresponds to an initial state in the witness circuit, as well as a transition. The property P' is a strengthening of P. In such a way, the safety of M' implies the safety of M. In summary, a bad trace in M corresponds to a bad trace in M'. A sketch of the traces for both M and M' is shown in Fig. 2. The latter two checks (Def. 2.3.4 and Def. 2.3.5) prove P' to be an inductive invariant, entailing the safety of M'.

Before we formally prove that the safety of M' certifies the safety of M, we first show that by Def. 2.3, a reset state in M corresponds to a reset state in M'.

Lemma 2.4. *For two circuits $M = (I, L, R, F, P, C)$ and $M' = (I', L', R', F', P', C')$ satisfying the reset check (2.3.1) and R' stratified, any assignment to $I \cup L$ satisfying $R\{K\} \wedge C$ can be extended to satisfy $R'\{L'\} \wedge C'$.*

Proof. Assume the reset check passes and R' is stratified, let s be an arbitrary but fixed assignment satisfying $R\{K\} \wedge C$. The assumptions imply that s satisfies $R'\{K\} \wedge C'$; however, note that s does not necessarily satisfy $R'\{L'\}$. Now assume there is a latch $l \in K$ with $r'_l(s) \not\equiv r'_l(s_u)$ where s_u is the same as s except for the value of some literal $u \in (I' \cup L') \setminus (I \cup L)$. We have $l \not\equiv r'_l(s_u)$ and therefore $R'\{K\}$ false under s_u. However, u is not in $I \cup L$ and $R\{K\} \wedge C$ still evaluates to true under s_u, thus implying $R'\{K\}$, and leading to the desired contradiction.

It follows, that for any assignment s to $I \cup L$ satisfying $R\{K\} \wedge C$ the (semantic) dependencies of the remaining reset functions under the assignment $\{r'_l(s) \mid l \in L' \setminus L\}$ can be seen as a graph with a topological sorting where the shared variables $(I \cup L) \cap (I' \cup L')$ are the bottom. Since R' is stratified s can always be extended to satisfy $R'\{L'\}$ by assigning in the reverse of that order. This new assignment still satisfies $R\{K\} \wedge C$ and thereby C'. □

We now move on to prove the correctness of the certificate format in the following theorem. A visualization of the proof can be found in Fig. 2.

Theorem 2.5. *Given two circuits $M = (I, L, R, F, P, C)$ and $M' = (I', L', R', F', P', C')$ with R' stratified. If M' is a valid witness circuit for M, then M is safe.*

Proof. Assume, for contradiction, M is not safe, i.e., there is a bad trace of some finite length n in the form of an assignment to n copies of $I \cup L$ satisfying:

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

We show how to extend this assignment to each copy of the literals in $I' \setminus I \cup L' \setminus L$ that satisfies:

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

Let $X' = (I' \cup L') \setminus (I \cup L)$. The assignment satisfies $R_0\{K\} \wedge C_0$ and by Lemma 2.4 can be extended to X'_0 satisfying $R'_0\{L'\} \wedge C'$. With that and the transition check $F'_{0,1}\{K\} \wedge C'_1$ is satisfied and the assignment can be extended to X'_1 satisfying $F'_{0,1}\{L'\} \wedge C'_1$ by the definition of transition functions. Applying the same argument n times yields an assignment to $(I \cup L \cup I' \cup L')^n$ 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 M', thus contradicting the initial assumption that a bad trace exists in M. □

In order for the reset and transition checks in Def. 2.3 to pass, the constraint C' needs to hold for all possible extensions generated by the reset and transition functions. A conservative way to check if this is the case, is to ensure that the constraint only contains variables shared between the two circuits. This can be a natural assumption. For standard model checking algorithms, for example IC3/PDR [7] under constraints, the resulting certificate M' will simply be $M' = (I, L, R, F, P', C)$, 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 witness constructions where it is necessary to constrain new variables in the witness circuit as well as to allow cyclic reset definitions, the format in Def. 2.3 has to be relaxed to include quantifiers in the checks. As a result, it provides more flexibility in generating witness circuits.

Definition 2.6 (Quantified Reset and Transition)**.** Given a circuit M, the circuit M', with $M = (I, L, R, F, P, C)$, $M' = (I', L', R', F', P', C')$, and $X' = (I' \cup L') \setminus (I \cup L)$, is a witness circuit if it meets Def. 2.3, where either or both reset and transition checks have been relaxed to:

- Reset$^{\exists}$: $\exists X'.R(L) \wedge C \Rightarrow R'(L') \wedge C'$;
- Transition$^{\exists}$:

$$\exists X'_1.F(L_0, L_0) \wedge C_0 \wedge C_1 \wedge C'_0 \Rightarrow F'(L'_0, L'_1) \wedge C'_1.$$

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

When the reset functions of M' are cyclic, i.e., not stratified, it is necessary to use the reset$^{\exists}$ check. Without the quantification, it is not sufficient to prove Theorem 2.5. However, since the reset$^{\exists}$ condition directly guarantees that a reset state in M corresponds to a reset state in M', it is no longer necessary to impose the stratification assumption on resets. Even with stratified resets, it might be necessary to use the reset$^{\exists}$ check if the reset functions of the witness do not ensure that the constraint is fulfilled. However, in case the quantifier-free reset check passes, certification is successful anyways. The same goes for the transition check. Note that if the constraint does not use any variables in X', it is never necessary to use the relaxed versions. The rest of the checks stay the same as in Def. 2.3. We formally prove that this is a valid certificate in the following theorem.

Theorem 2.7. *Given a circuit M and a witness circuit M' according to Def 2.6, M is safe.*

Proof. The reset$^{\exists}$ and transition$^{\exists}$ checks directly imply the existence of a reset state, 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 M' can be constructed iteratively from reset to bad state. The rest follows the same arguments as our proof for Theorem 2.5. □

Transition systems. As an alternative to the circuit definition, symbolic transition systems are commonly used as a form of model representation. The main difference is that transition relations are used instead of transition functions. We write a transition system as $M_T = (L, Init, T, P)$ where L is the set of Boolean state variables (same as latch variables in a circuit): (1) $Init$ is a formula over variables L, representing the set of initial states; (2) $T(L_0, L_1)$ is a formula over variables L_0 and L_1, where L_1 is a copy of L and $L_0 = L$; (3) P represents the set of good states. Here the non-determinism is defined by the transition relation $T(L_0, L_1)$, whereas in a circuit we need inputs for this. To add a given constraint C to a transition system M_T, we can simply rewrite it to $M'_{T'} = (L, Init \wedge C, T(L_0, L_1) \wedge C_0 \wedge C_1, P \wedge C)$.

We now proceed to provide an equivalent certificate format for transition systems. For a given $M_T = (L, Init, T, P)$, and $M'_{T'} = (L', Init', T', P')$. Let $X' = (I' \cup L') \setminus (I \cup L)$. Then $M'_{T'}$ is a certificate for M_T if it satisfies the following:

1. $\exists X'.Init \Rightarrow Init'$;
2. $\exists X'_1.T(L_0, L_1) \Rightarrow T'(L'_0, L'_1)$;
3. $P' \Rightarrow P$.
4. $Init' \Rightarrow P'$;
5. $P'_0 \wedge T'(L'_0, L'_1) \Rightarrow P'_1$.

3 Certifying Constraint Mining

In addition to the explicit constraints that are given as part of the circuit specification, constraint mining is an effective preprocessing step that simplifies the task for a terminal model checker. This has been discussed in the context of both safety [8] and liveness [9] properties. In this section, we consider the problem of certifying the hidden constraints that have been extracted.

3.1 Model Constraints

The first type of constrained we consider are model constraints. Intuitively, they are those that hold in all reachable states, similar to a safety property. It is therefore obvious that adding them does not change the set of reachable states in the model. In industry practice, it is common to verify hundreds of properties for the same design, and it can be beneficial for the verification performance to add already proven properties as constraints [10, 13].

Definition 3.1 (Model constraint)**.** Given a circuit $M = (I, L, R, F, P, C)$, the formula $\hat{C}(I, L)$ is said to be a model constraint if it holds in all reachable states, i.e., the following formula is unsatisfiable for any n:

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

By Def. 3.1, during model checking, the extracted model constraints can simply be appended to strengthen C to form a new constraint $C \wedge \hat{C}$. A base model checker that uses for instance IC3/PDR or k-induction can be then employed. Once the verification process is completed, the base model checker also provides a witness circuit. However, this witness circuit only certifies the safety property on a smaller circuit where the constraint holds, rather than the original model checking problem. Next we are going to show how to construct a witness circuit for the original circuit.

Definition 3.2 (Composing circuits)**.** When it comes to multiple properties, it is possible that the model checker splits the task into several subproblems. The sub-circuits that undergo model checking can be segments of the original. For a circuit $M = (I, L, R, F, P, C)$ with $P = \bigwedge_{i \in [0,n]} P^i$, each sub-circuit M^i with the property P^i obtains its witness circuit

$W^i = (I^i, L^i, R^i, F^i, \phi^i, C^i)$ that satisfies Def. 2.3 w.r.t M^i. In order to output a witness circuit for the original property P, we can simply compose the witness circuits to obtain $W = (I', L', R', F', P', C')$ such that:

- $I' = \bigcup_{i\in[0,n]} I^i$;
- $L' = \bigcup_{i\in[0,n]} L^i$;
- $R' = \bigcup_{i\in[0,n]} R^i$;
- $F' = \bigcup_{i\in[0,n]} F^i$;
- $P' = \bigwedge_{i\in[0,n]} \phi^i$.
- $C' = \bigwedge_{i\in[0,n]} C^i$.

We assume here, that variables unique to the witness circuits $(I^i \cup L^i) \setminus (I \cup L)$ are disjunct from other witness. Otherwise, renaming is necessary. It is straightforward to see that in this way the composed witness circuit is a valid witness circuit for any of the original model checking problems.

Theorem 3.3. *Given two circuits M^0 and M^1 with witness circuits W^0 and W^1 as defined in Def. 2.3, composing W^0 and W^1 yields a witness W for both M^0 and M^1.*

Proof. Let $M = (I, L, R, F, P, C)$ be a circuit with stratified resets, with $P = P^0 \wedge P^1$. Let $W^0 = (I^0, L^0, R^0, F^0, P^0, C^0)$ and $W^1 = (I^1, L^1, R^1, F^1, P^1, C^1)$ be the witness circuits for $M^0 = (I, L, R, F, P^0, C)$ and $M^1 = (I, L, R, F, P^1, C)$ respectively. We construct a composed witness circuit $W = (I', L', R', F', P', C')$ as above. Now we show that W is a witness circuit for M. Since W^0 and W^1 are witness circuits, R^0 and R^1 are both stratified. The latches $L^0 \setminus L$ and $L^1 \setminus L$ are disjoint, and by Lemma 2.4 the latches L have the lowest topological ordering, R is stratified. By our assumption, $R\{K^0\} \wedge C \Rightarrow R^0\{K^0\} \wedge C^0$ and $R\{K^1\} \wedge C \Rightarrow R^1\{K^1\} \wedge C^1$, where K^0 (and K^1) are the common variables of L and L^0 (and L^1). Therefore we have $R\{K^0 \cup K^1\} \Rightarrow R'\{K^0 \cup K^1\} \wedge C^0 \wedge C^1$. Thus the reset check passes.

Similarly, since we have $F_{0,1}\{K^0\} \wedge C_0 \wedge C_1 \wedge C_0^0 \Rightarrow F_{0,1}^0\{K^0\} \wedge C_1^0$ and $F_{0,1}\{K^1\} \wedge C_0 \wedge C_1 \wedge C_0^1 \Rightarrow F_{0,1}^1\{K^1\} \wedge C_1^1$, we immediately have $F_{0,1}\{K^0 \cup K^1\} \wedge C_0 \wedge C_1 \wedge C_0' \Rightarrow F_{0,1}'\{K^0 \cup K^1\} \wedge C_1'$. Thus the transition check passes. For the property check, we have $(C \wedge C^0) \Rightarrow (P^0 \Rightarrow P)$ and $(C \wedge C^1) \Rightarrow (P^1 \Rightarrow P)$, which gives us $(C \wedge C^0 \wedge C^1) \Rightarrow (P^1 \wedge P^0 \Rightarrow P)$. The rest of the proof follows the same logic. □

The correctness of the construction for n circuits can be proven by iterative application of the above theorem.

3.2 Inductive Constraints

Next, we consider inductive constraints [8].

Definition 3.4 (Inductive constraint)**.** An inductive constraint $\hat{C}(I, L)$ of a circuit $M = (I, L, R, F, P, C)$ satisfies:
(1) $F_{0,1}\{L\} \wedge \neg C_0 \Rightarrow \neg C_1$; (2) $\neg C \Rightarrow P$.

Similar to model constraints, once the inductive constraint is extracted from the model, it can be used to strengthen the explicit constraint $C \wedge \hat{C}$. We illustrate the nature of inductive constraints in Fig. 3. Intuitively, it is correct to add any inductive constraint, since no bad traces can be eliminated. Once a trace leaves C it cannot cross back, since $\neg C$ is inductive, and no bad state can be reached since $\neg C \Rightarrow P$. Therefore the model checker only needs to ensure P holds in the states that satisfy C (i.e., the area marked blue).

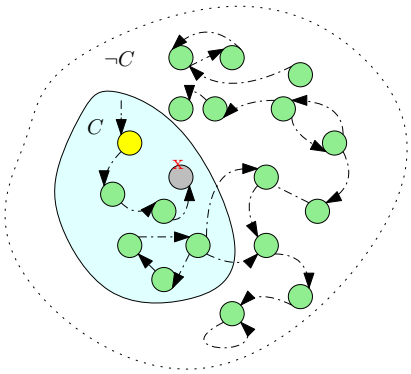

Figure 3. An illustration of the state space for inductive constraints. The bad states are marked gray; and the initial state is marked yellow. The blue region are the sates satisfying the constraint C. Note that C is of course not a subset of $\neg C$.

In the following we show something stronger: Adding an inductive constraint does not only preserve the safety of a model, but addition (or removal) of an inductive constraint does not change if a property is an inductive invariant.

Note the slightly confusing naming of *inductive* constraints taken from [8]; it is the negation of C that is inductive, and it is not an inductive invariant, as it does not necessarily hold in the reset states.

Theorem 3.5. *Given a circuit $M = (I, L, R, F, P, C)$ with inductive constraint $\hat{C}$, let $\hat{M} = (I, L, R, F, P, C \wedge \hat{C})$. The base and step condition hold in M, if and only if they hold in $\hat{M}$.*

Proof. Let M and $\hat{C}$ be fixed in the following. Slight rearrangement of the conditions for inductive constraints give us: $\hat{C}_1 \wedge F_{0,1} \Rightarrow \hat{C}_0$ and $\neg P \Rightarrow \hat{C}$. The step check for M fails if an assignment s exists that satisfies: $P_0 \wedge F_{0,1}\{L\} \wedge C_0 \wedge C_1 \wedge \neg P_1$. Since $\hat{C}$ is an inductive constraint an s satisfies $\neg P_1$, the next state version of the inductive constraint $\hat{C}_1$ also holds in s. With that s also satisfies $\hat{C}_0$, together giving us s satisfies: $P_0 \wedge F_{0,1}\{L\} \wedge C_0 \wedge \hat{C}_0 \wedge C_1 \wedge \hat{C}_1 \wedge \neg P_1$ and thus the step check also fails for M'.

The argument for the base check is similar, it fails in M if an assignment b satisfies: $R\{L\} \wedge C \wedge \neg P$. Since $\neg P$ holds and $\hat{C}$ is an inductive constraint, b also satisfies $R\{L\} \wedge C \wedge \hat{C} \wedge \neg P$. Thus the base check also fails for M'. The other direction is trivial for both cases. □

We further show how to produce witness circuits for models with added inductive constraints.

Definition 3.6 (Unconstrained witness circuit)**.** Let $\hat{M} = (I, L, R, F, P, C \wedge \hat{C})$ be the circuit with an extracted inductive constraint $\hat{C}$. The original model M is $M = (I, L, R, F, P, C)$. Let $\hat{P}'$ be an inductive invariant of $\hat{M}$. We construct a circuit $W = (I, L, R, F, P', C)$ that is a witness circuit for the original model M where $P' = \neg\hat{C} \vee \hat{P}'$.

Informally, the goal is to obtain an inductive invariant for the original transitions, thus the witness circuit P' makes use of the inductive nature of $\neg\hat{C}$. We formally prove the correctness of the construction in the following theorem.

Theorem 3.7. *A valid unconstrained witness circuit correctly certifies model checking using inductive constraints.*

Proof. Let $\hat{C}$ be an inductive constraint for a circuit $M = (I, L, R, F, P, C)$ such that R is stratified. Then let $\hat{M} = (I, L, R, F, P, C \wedge \hat{C})$ be the circuit under inductive constraints with its inductive invariant $\hat{P}'$. Let $M' = (I', L', R', F', P', C')$ be an unconstrained witness circuit constructed as Def. 3.6. We show in the following that M' satisfies Def. 2.3.

Since W differs from M only in the property, the reset and transition checks trivially hold. By Def. 3.6, $\hat{P}' \Rightarrow P$, and $\hat{C} \Rightarrow P$, thus $P' \Rightarrow P$. Hence we get $(C \wedge C') \Rightarrow (P' \Rightarrow P)$ therefore the property check passes.

Now we proceed to prove that P' is an inductive invariant in W. For the base check, let s be a satisfying assignment to $R\{L\} \wedge C$ (If there is no such s, then the base step simply holds.). By our assumption that $\hat{P}'$ is an inductive invariant of $\hat{M}$, $R\{L\} \wedge C \Rightarrow \hat{P}'$ holds. Thus the base check passes.

For the step check, we need to split cases. Let $s_0 \wedge s_1$ be the assignment (two consecutive states) to $I_0 \cup L_0 \cup I_1 \cup L_1$ that satisfies the transition $F_{0,1}\{L_0, L_1\}$. We assume s_0 satisfies P'_0. First of all, because of the disjunctive operator in P'_0, we consider the case where s_0 satisfies $\hat{C}_0$. This implies the transition $s_0 \wedge s_1$ satisfies $F_{0,1}\{L_0, L_1\} \wedge \hat{C}_0$. If $\hat{C}_1$, since $\hat{P}'$ is an inductive invariant in $\hat{M}$, s_1 also satisfies $\hat{P}'_1$ therefore P'_1. Otherwise, we have $\neg\hat{C}_1$ therefore P'_1. Secondly, we consider the case where s_0 satisfies $\neg\hat{C}_0$. By Def. 3.4, $\neg\hat{C}$ is inductive thus s_1 satisfies $\neg\hat{C}_1$ therefore P'_1. We conclude the inductive check passes, and M' is a valid witness circuit for M. □

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 mining algorithm. Intuitively, property constraints are implied by the property itself.

Definition 3.8 (Property constraint)**.** A property constraint $\hat{C}(I, L)$ of a circuit $M = (I, L, R, F, P, C)$ satisfies: $P \Rightarrow \hat{C}$.

Once they are extracted, the use of property constraints differ from the previous two that we have mentioned. So far, for both model constraints and inductive constraints, they are directly appended to the explicit constraint i.e., $C \wedge \hat{C}$. Property constraints, on the other hand, require to be integrated into the transition functions.

Model checking under property constraints. Let $M = (I, L, R, F, P, C)$ be a given circuit with a property constraint $\hat{C}$. The circuit under the property constraint is $\hat{M} = (I, L, R, \hat{F}, P, C)$ where: $\hat{F} = \{\hat{f}_l \mid l \in L\}$: for $l \in L$, $\hat{f}_l = ite(\hat{C}(I, L), f_l, l)$.

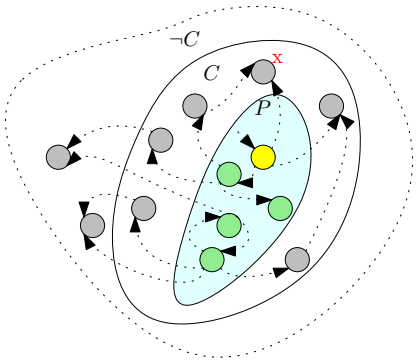

Figure 4. An illustration of property constraints. Since the property constraints act as conditions in the transition function, outgoing transitions only take place in states that satisfy C. There is no transition from states in $\neg C$ to C.

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 change 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.

Definition 3.9 (Witness circuit for property constraints)**.** Given a circuit $M = (I, L, R, F, P, C)$ with a property constraint $\hat{C}$. Let $\hat{W} = (\hat{I}', \hat{L}', \hat{R}', \hat{F}', \hat{P}', \hat{C}')$ be a witness circuit of $\hat{M} = (I, L, R, \hat{F}, P, C)$. Then witness circuit for property constraints $W' = (I', L', R', F', P', C')$ is defined as:

- $I' = I \cup \hat{I}'; L' = L \cup \hat{L}'$;
- $R' = R \cup \hat{R}'; F' = \{f_l \mid l \in L\} \cup \{f^{\hat{}}_l \mid l \in (\hat{L}' \backslash L)\}$;
- $P' = \hat{P}'; C' = \hat{C}'$.

The property $\hat{P}'$ of the reduced circuit is in fact also inductive when we add the removed transitions back. We prove in the following theorem that this construction certifies the original model checking result.

Theorem 3.10. *The witness circuit for property constraints certifies model checking using property constraints.*

Proof. We omit the proofs for the simulation relation between M and W' as well as the base check since it follows the same logic as our proof for Theorem 3.7. Now we prove that $\hat{P}'$ is an inductive invariant in W'.

Let $s_0 \wedge s_1$ be the assignment to $I_0 \cup L_0 \cup I_1 \cup L_1$ that satisfies the transition $F'_{0,1}\{L'_0, L'_1\} \wedge P'_0$. We consider two cases depending on if $\hat{C}_0$ holds. If we assume s_0 satisfies $\hat{C}_0$, $\hat{P}'_1$ follows immediately since $s_0 \wedge s_1$ would also satisfy $\hat{F}'_{0,1}\{\hat{L}'_0, \hat{L}'_1\} \wedge \hat{P}'_0$. In the second case where s_0 satisfies $\neg\hat{C}_0$, we also have $\neg\hat{P}_0$. Therefore we conclude that P' is inductive. □

4 Extending Circuits with Oracles

At the core of certification via witness circuits lies the concept of using additional inputs and latches to allow additional behavior for which safety is easier to show. While sufficient for most model checking techniques, requiring inductiveness has proven difficult when certifying techniques that reason over multiple paths in some way. One example of such a technique is k-induction with the addition of uniqueness constraints. Since, adding uniqueness constraints is only valid because any potential bad state that can be reached, can also be reached with a simple path. In this section we extend witness circuits with *oracles* which allow to reason over multiple states in the witness circuit.

Definition 4.1 (Step check with oracles)**.** For a given circuit $M = (I, L, R, F, P, C)$, the circuit $M' = (I', O', L', R', F', P', C')$ is a witness if R' and F' are semantically independent of O' and it meets Def. 2.3, where the step check is relaxed to:

$$\text{Step}^{\exists}\colon F'_{0,1}\{L'\} \wedge C'_1 \wedge \neg P'_1 \Rightarrow (\forall O'_0.\neg C'_0) \vee (\exists O'_0.C'_0 \wedge \neg P'_0).$$

If R' or F' are dependent on O', we need additional quantifiers in the respective checks similar to Def. 2.6.

Definition 4.2 (Reset and Transition with oracles)**.** Given a circuit M, the circuit M', with $M = (I, L, R, F, P, C)$, $M' = (I', L', R', F', P', C')$, and $X' = (I' \cup L') \setminus (I \cup L)$, is a witness circuit if it meets Def. 2.3, where the step check has been relaxed to $\text{Step}^{\exists}$ and either or both reset and transition checks have been relaxed to:

- $\text{Reset}^{\exists\forall}\colon \exists X' \forall O.R(L) \wedge C \Rightarrow R'(L') \wedge C'$;
- $\text{Transition}^{\exists\forall}$:

$$\exists X'_1 \forall O_1.F(L_0, L_0) \wedge C_0 \wedge C_1 \wedge C'_0 \Rightarrow F'(L'_0, L'_1) \wedge C'_1.$$

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

Theorem 4.3. *A witness circuit that satisfies the certificate format with oracles is a valid certificate.*

Proof. Since the oracles are only in the witness circuit, the $\text{reset}^{\exists\forall}$ and $\text{transition}^{\exists\forall}$ conditions still allow to do the construction of the trace in M' the same way as in the proof of Theorem 2.7. The same is true if R' or F' are syntactically independent of O and the quantifier-free checks are used. What is left to show is that no bad state is reachable in M'. We will be considering the equivalence classes induced by O on the state space of M', i.e., states only differing in O are in the same equivalence class. Assume there is a bad trace $s_0, \ldots, s_n$ in M'. 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. The transition check implies that all states in the equivalence class of s_0 have a transition to all states in the equivalence class of s_1, which allows us to use the step condition to show that all states in the class of s_1 are 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, leading to the desired contradiction. □

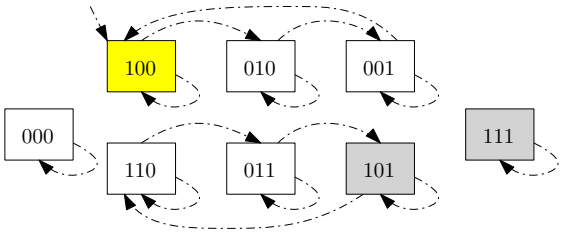

Figure 5. The state-transition diagram of a 3-bit freezable counter. The bad states are marked gray.

4.1 Uniqueness constraints

The authors in [22] presented a method to certify k-induction, which is one of the most common model checking techniques. Other approaches have also attempted to generate certificates for this technique [15, 16]. However, all these approaches are limited to k-induction *without* uniqueness constraints, sometimes also referred to as simple path constraints. For finite-state systems, k-induction is complete under uniqueness constraints, as the value of k can increase up to the recurrence diameter of the model.

***k*-induction with uniqueness constraints [19].** A property P of a given circuit $M = (I, L, R, F, P, C)$ is said to be k-inductive under uniqueness constraints iff:

1. $R_0\{L\} \wedge (\bigwedge_{i\in[0,k-1)} F_{i,i+1}\{L\}) \Rightarrow \bigwedge_{i\in[0,k)} P_i$.
2. $(\bigwedge_{i\in[0,k)} F_{i,i+1}\{L\}) \wedge (\bigwedge_{i\in[0,k)} P_i) \wedge \mathit{unique}_k \Rightarrow P_k$.

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

The first formula checks 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 consecutively then it also holds at the next step. Intuitively, the uniqueness constraint ensures that, in the inductive step, the k consecutive states are all distinct from one another.

Example 4.4. Consider in Figure 5 a 3-bit freezable ring counter that shifts all bits to the right, with the rightmost bit wrapping around to the leftmost position. The counter also has the option to freeze, remaining in its current state. The initial state is 100. The property asserts that the counter value never exceeds 6 (i.e., 110). Due to the presence of self-loops, the property is *3-inductive* only under the uniqueness constraint. Without this constraint, the model checking program will never terminate until it reaches a time-out.

To certify k-induction under uniqueness constraints, we can construct a witness circuit as follows.

Definition 4.5 (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 $M' = (I', O', L, R, F, P', C')$ is defined as:

- $I' = I^0 \cup \cdots \cup I^{k-1}$ where $I^0 = I, O' = I' \backslash I$;
- $P' = \bigwedge_{i\in[0,k)} P(I^i, L^i)$, $C' = \bigwedge_{i\in[0,k)} C(I^i, L^i)$ where $L^0 = L$ and $L^{i+1} = \{f_l(I^i, L^i) \mid l \in L)\}$.

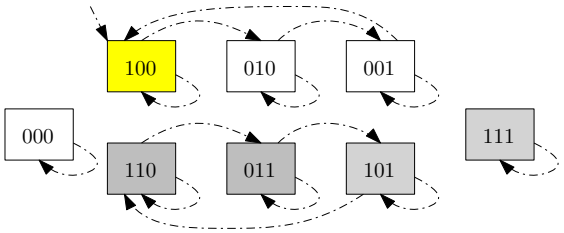

Figure 6. The k-witness circuit for the 3-bit freezable counter.

In the above definition, $L^1, \ldots, L^{k-1}$ are not latches. We simply use the literal for the transition functions, and in practice, they are and-gates. The latches are the same in the original circuit as well as 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.

For the ring counter from Fig. 5, considering the self-loops, each state has two successor states. The property P' requires all successor states reachable within two steps to be good states. An illustration of the witness circuit for this example is displayed in Fig. 6.

Even though Def. 4.5 is also a valid construction for k-induction *without* uniqueness constraints, it is expensive for the certificate checker to have QBF checks. Hence in the case where uniqueness constraints are not required, the witness circuit construction should still follow [22].

Theorem 4.6. *Given a circuit $M = (I, L, R, F, P, C)$ with some $k \in \mathbb{N}^+$, and a k-witness circuit $M' = (I', L, R, F, P', C)$. If M is k-inductive, then M' is a valid witness circuit for M.*

Proof. Since L, R, F remain unchanged and C' implies C, the reset and transition condition hold trivially. P' clearly implies P, giving us the property check. 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 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 C induced by the assignment to L in u' and v' respectively. The shortest path from v to a bad state has exactly k steps, all states are different, and all but the last state are good. If 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 be different from the bad state, prepending the path with u yields a path in C consisting of k unique good states followed by a bad state, thus C is not k-inductive under uniqueness constraints. □

5 Experimental Evaluation

We implemented the proposed certificate format (Def. 2.3) in the certificate checker CERTIFAIGER, 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 [2] to generate k-witness circuits as defined in Def. 4.5 when uniqueness constraints are enabled. Additionally, we implement the tool AIGMERGE that uses Def. 3.2 to compose arbitrary circuits.

CERTIFAIGER utilizes KISSAT [4] for SAT checks and the QBF solverQUABS [20] 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.

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 ran the 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. 2.3. For each original property run the property check. We compare the total time of these checks, to total time it take to certifying 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,

In the second experiment, we study the effectiveness of our certification method for k-induction under uniqueness constraints. We selected the benchmarks 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 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 generally more challenging than SAT solving, we believe the significant performance gap is also

	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 (O). The other columns display the number of and-gates (A), total time taken in seconds for checking individual witness circuit 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 O 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.

due to the fact that Kissat has been recently optimized for circuit-solving tasks [5]. It is highly likely that incorporating these optimizations into QuAbs could lead to substantial performance improvements.

	t_{MC}	t_{Cert}	t_{SAT}	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 give 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}). All times presented are in seconds. The last three columns denote the inductive depth k, the number of input variables I, and gates A respectively.

6 Conclusion

In this work, we present a certification method that addresses both explicit and extracted hidden constraints. Our approach enables the certification of extracted hidden constraints, which we illustrate on three different classes. A key feature of our method is the construction of a witness circuit during the model checking process, that can be validated by an independent certificate checker. Another of our contributions is a certification approach for k-induction under uniqueness constraints. We demonstrate the effectiveness of our technique using benchmarks from the hardware model checking competition. An intriguing direction for future research lies in optimizing the certification process, particularly by reducing the size of the certificates.

References

[1] Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann, and Andreas Stahlbauer. 2015. Witness validation and stepwise testification across software verifiers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 721–733.

[2] Armin Biere and Robert Brummayer. 2008. Consistency checking of all different constraints over bit-vectors within a SAT solver. In *2008 Formal Methods in Computer-Aided Design*. IEEE, 1–4.

[3] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. 1999. Symbolic Model Checking without BDDs. In *TACAS (Lecture Notes in Computer Science, Vol. 1579)*. Springer, 193–207.

[4] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. 2024. CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024. In *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions (Department of Computer Science Report Series B, Vol. B-2024-1)*, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda (Eds.). University of Helsinki, 8–10.

[5] Armin Biere, Katalin Fazekas, Mathias Fleury, and Nils Froleyks. 2024. Clausal congruence closure. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[6] Armin Biere, Keijo Heljanko, and Siert Wieringa. 2011. AIGER 1.9 and beyond. (2011).

[7] Aaron R. Bradley. 2011. SAT-Based Model Checking without Unrolling. In *VMCAI (Lecture Notes in Computer Science, Vol. 6538)*. Springer, 70–87.

[8] Gianpiero Cabodi, Paolo Camurati, Luz Amanda Garcia, Marco Murciano, Sergio Nocco, and Stefano Quer. 2009. Speeding up model checking by exploiting explicit and hidden verification constraints. In *DATE*. IEEE, 1686–1691.

[9] Koen Claessen and Niklas Sörensson. 2012. A liveness checking algorithm that counts. In *FMCAD*. IEEE, 52–59.

[10] Sourav Das, Aritra Hazra, Pallab Dasgupta, Sudipta Kundu, and Himanshu Jain. 2024. PURSE: Property Ordering Using Runtime Statistics for Efficient Multi - Property Verification. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1–6. https://doi.org/10.23919/DATE58400.2024.10546895

[11] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. 2013. A fully verified executable LTL model checker. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*. Springer, 463–478.

[12] Nils Froleyks, Emily Yu, Armin Biere, and Keijo Heljanko. 2024. Certifying Phase Abstraction. In *IJCAR (1) (Lecture Notes in Computer Science, Vol. 14739)*. Springer, 284–303.

[13] Eugene Goldberg, Matthias Güdemann, Daniel Kroening, and Rajdeep Mukherjee. 2018. Efficient verification of multi-property designs (The benefit of wrong assumptions). In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 43–48. https://doi.org/10.23919/DATE.2018.8341977

[14] Alberto Griggio, Marco Roveri, and Stefano Tonetta. 2021. Certifying proofs for SAT-based model checking. *Formal Methods in System Design* 57, 2 (2021), 178–210.

[15] Arie Gurfinkel and Alexander Ivrii. 2017. K-induction without unrolling. In *FMCAD*. IEEE, 148–155.

[16] Alain Mebsout and Cesare Tinelli. 2016. Proof certificates for SMT-based model checkers for infinite-state systems. In *FMCAD*. IEEE, 117–124.

[17] Kedar S. Namjoshi. 2001. Certifying Model Checkers. In *CAV (Lecture Notes in Computer Science, Vol. 2102)*. Springer, 2–13.

[18] Mathias Preiner, Armin Biere, and Nils Froleyks. 2020. Hardware model checking competition 2020. (2020).

[19] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. 2000. Checking Safety Properties Using Induction and a SAT-Solver. In *FMCAD (Lecture Notes in Computer Science, Vol. 1954)*. Springer, 108–125.

[20] Leander Tentrup. 2019. CAQE and quabs: Abstraction based QBF solvers. *Journal on Satisfiability, Boolean Modeling and Computation* 11, 1 (2019), 155–210.

[21] Emily Yu, Armin Biere, and Keijo Heljanko. 2021. Progress in Certifying Hardware Model Checking Results. In *CAV (2) (Lecture Notes in Computer Science, Vol. 12760)*. Springer, 363–386.

[22] Emily Yu, Nils Froleyks, Armin Biere, and Keijo Heljanko. 2022. Stratified Certification for k-Induction. In *FMCAD*. IEEE, 59–64.

[23] Emily Yu, Nils Froleyks, Armin Biere, and Keijo Heljanko. 2023. Towards Compositional Hardware Model Checking Certification. In *FMCAD*. IEEE, 1–11.