Certifying Phase Abstraction

Nils Froleyks[1(✉)], Emily Yu[2], Armin Biere[3], and Keijo Heljanko[4,5]

[1] Johannes Kepler University, Linz, Austria
N.froleyks@gmail.com
[2] Institute of Science and Technology Austria, Klosterneuburg, Austria
[3] Albert–Ludwigs–University, Freiburg, Germany
[4] University of Helsinki, Helsinki, Finland
[5] Helsinki Institute for Information Technology, Helsinki, Finland

Abstract. Certification helps to increase trust in formal verification of safety-critical systems which require assurance on their correctness. In hardware model checking, a widely used formal verification technique, phase abstraction is considered one of the most commonly used preprocessing techniques. We present an approach to certify an extended form of phase abstraction using a generic certificate format. As in earlier works our approach involves constructing a witness circuit with an inductive invariant property that certifies the correctness of the entire model checking process, which is then validated by an independent certificate checker. We have implemented and evaluated the proposed approach including certification for various preprocessing configurations on hardware model checking competition benchmarks. As an improvement on previous work in this area, the proposed method is able to efficiently complete certification with an overhead of a fraction of model checking time.

1 Introduction

Over the past few decades, symbolic model checking [2,22,23] has been put forward as one of the most effective techniques in formal verification. A lot of trust is placed into model checking tools when assessing the correctness of safety-critical systems. However, model checkers themselves and the symbolic reasoning tools they rely on, are exceedingly complex, both in the theory of their algorithms and their practical implementation. They often run for multiple days, distributed across hundreds of interacting threads, ultimately yielding a single bit of information signaling the verification result. To increase trust in these tools, several approaches have attempted to implement fully verified model checkers in a theorem proving environment such as Isabelle [1,27,54]. However, the scalability as well as versatility of those tools is often rather limited. For example, a technique update tends to require the entire tool to be re-verified.

An alternative is to make model checkers provide machine-checkable proofs as certificates that can be validated by independent checkers [8–10,31,32,39,42,47], which is already a successful approach in SAT [34,35], i.e., proofs are mandatory in the SAT competition since 2016 [3], and they are a very hot topic

© The Author(s) 2024
C. Benzmüller et al. (Eds.): IJCAR 2024, LNAI 14739, pp. 284–303, 2024.
https://doi.org/10.1007/978-3-031-63498-7_17

in SMT [4,5,36,51] and beyond [4]. Crucially, these certificates need to be simple enough to allow the implementation of a fully verified proof checker [33,37,41], and preferably verifiable "end-to-end", i.e., certifying all stages of the model checking process, including all forms of preprocessing steps.

The approach in [15,56,57] introduces a generic certificate format that can be directly generated from hardware model checkers via book-keeping. More specifically, the certificate is in the form of a Boolean circuit that comes with an inductive invariant, such that it can be verified by six simple SAT checks. So far, it has shown to be effective across several model checking techniques, but has not covered phase abstraction [16]. The experimental results from [15,56,57] also show performance challenges with more complex model checking problems. In this paper, we focus on refining the format for smaller certificates while accommodating additional techniques such as cone-of-influence analysis reduction [22].

Phase abstraction [16] is a popular preprocessing technique which tries to simplify a given model checking problem by detecting and removing periodic signals that exhibit clock-like behaviors. These signals are essentially the clocks embedded in circuit designs, often due to the design style of multi-phase clocking [46]. Phase abstraction helps reduce circuit complexity therefore making the backend model checking task easier. Differently from [6,7] where the concept was first suggested, requiring syntactic analysis and user inputs, phase abstraction [16] makes use of ternary simulation to automatically identify a group of clock-like latches. Beside this, ternary simulation has also been utilized in the context of temporal decomposition [20] for detecting transient signals.

In industrial settings, due to the use of complex reset logic as well as circuit synthesis optimizations, clock signals are sometimes delayed by a number of initialization steps [19]. To further optimize the verification procedure we extend phase abstraction by exploiting the power of ternary simulation to capture different classes of periodic signals including those that are considered partially as clocks as well as equivalent signals [26]. An optimal phase number is computed based on globally extracted patterns, which then is used to unfold the circuit multiple times. The resulting unfolded circuit further undergoes rewriting and cone-of-influence reduction, before it is passed on to a base model checker for final verification. To summarize our contributions are as follows:

1. We formalize, revisit and extend the original phase abstraction [16] by introducing periodic signals, that are then identified and removed for circuit reduction. Our technique also subsumes temporal decomposition [20].
2. Building upon [15,56,57], we propose a refined certificate format for hardware model checking based on a new *restricted simulation* relation. We demonstrate how to build such a certificate for extended phase abstraction.
3. We present MC2, a certifying model checker that implements our proposed preprocessing technique and generates certificates for the entire model checking process. We show empirically that the approach requires small certification overhead in contrast to [15,56,57].

After background in Sect. 2, Sect. 3 introduces the notion of periodic signals. In Sect. 4 we present an extended variant of phase abstraction that simplifies

the original model with periodic signals. In Sect. 5 we define a refined certificate format and present a general certification approach for phase abstraction. In Sect. 6 we describe the implementation of MC2 and then show the effectiveness of our new certification approach in Sect. 7.

2 Background

Given a set of Boolean variables $\mathcal{V}$, a literal l is either a variable $v \in \mathcal{V}$ or its negation $\neg v$. A *cube* is considered to be a non-contradictory set of literals. Let c be such a cube over a set of variables L and assume L' are copies of L, i.e., each $l \in L$ corresponds bijectively to an $l' \in L'$. Then we write $c(L')$ to denote the resulting cube after replacing the variables in c with its corresponding variables in L'. For a Boolean formula f, we write $f|_l$ and $f|_{\neg l}$ to denote the formula after substituting all occurrences of the literal l with $\top$ and $\bot$ respectively. We use equality symbols $\simeq$ [24] and $\equiv$ to denote syntactic and semantic equivalence and similarly $\rightarrow$ and $\Rightarrow$ to denote syntactic and semantic logical implication.

Definition 1 (Circuit). *A circuit C is represented by a quintuple (I, L, R, F, P), where I and L are (finite) sets of input and latch variables. The reset functions are given as $R = \{r_l(I, L) \mid l \in L\}$ where the individual reset function $r_l(I, L)$ for a latch $l \in L$ is a Boolean formula over inputs I and latches L. Similarly the set of transition functions is given as $F = \{f_l(I, L) \mid l \in L\}$. Finally $P(I, L)$ denotes a safety property corresponding to set of good states again encoded as a Boolean formula over the inputs and latches.*

This notion can be extended to more general circuits involving for instance word-level semantics or even continuous variables by replacing in this definition Boolean formulas by corresponding predicates and terms in first-order logic modulo theories. For simplicity of exposition we focus in this work on Boolean semantics, which matches the main application area we are targeting, i.e., industrial-scale gate-level hardware model checking. We claim that extensions to "circuits modulo theories" are quite straightforward.

A concrete state is an assignment to variables $I \cup L$. Therefore the set of reset states of a circuit is the set of satisfying assignments to $R(L) = \bigwedge_{l \in L} (l \simeq r_l(I, L))$. Note the use of syntactic equality "$\simeq$" in this definition.

As in previous work [15] we assume acyclic reset functions. Therefore $R(L)$ is always satisfiable. A circuit with acyclic reset functions is called *stratified*.

As in bounded model checking [11], with I_i and L_i "temporal" copies of I and L at time step i, the *unrolling* of a circuit up to length k is expressed as:

$$U_k = \bigwedge_{i \in [0,k)} (L_{i+1} \simeq F(I_i, L_i)).$$

Cube simulation [57] subsumes ternary simulation such that a lasso found by ternary simulation can also be found via cube simulation. A cube simulation is a sequence of cubes $c_0, \ldots, c_\delta, \ldots, c_{\delta+\omega}$ over latches L such that (1) $R(L) \Rightarrow c_0$;

(2) $c_i \wedge (L' \simeq F(I, L)) \Rightarrow c'_{i+1}$ for all $i \in [0, \delta + \omega)$, where c'_{i+1} is the primed copy of c_{i+1}. It is called a cube lasso if $c_{\delta+\omega} \wedge (L' \simeq F(I, L)) \Rightarrow c'_\delta$. In which case δ is the stem length and ω is the loop length. For $\delta = 0$, the initial cube is already part of the loop and for $\omega = 0$, the lasso ends in a self-loop.

3 Periodic Signals

In sequential hardware designs, signals that eventually stabilize to a constant, i.e., to $\top$ or $\bot$, after certain initialization steps are called *transient* signals [20,57], whereas oscillating signals have clock-like or periodic behaviors. A simplest example of a clock is a latch that always oscillates between $\top$ and $\bot$.

Since hardware designs typically consist of complex initialization logic, there are occurrences of delayed oscillating signals, like clocks that start ticking after several reset steps, with a combination of transient and clock behaviours. We generalize this concept to categorize latches as periodic signals associated with a *duration* (i.e., the number of time steps for which a signal is delayed) and a *phase number* (i.e., the period length in a periodic behavior). Moreover, our generalization also captures equivalent and antivalent signals [26], as well as those that exhibit partial periodic behaviours. See Fig. 1 for an example.

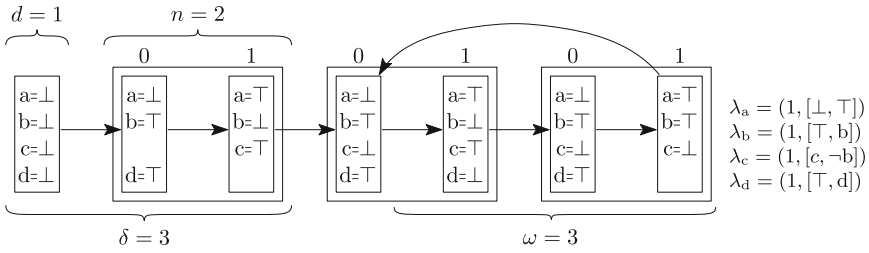

Fig. 1. An example of a cube lasso over the latches $l \in L = \{a, b, c, d\}$. In the example the tall rectangles represent cubes as partial assignments, i.e., the second cube from the left is $(\neg a) \wedge b \wedge d$. Phase 0 and 1 are marked on top of the cubes. As shown, duration $d = 1$ and phase number $n = 2$ yield a high number of useful signals for this cube lasso. Each latch l is associated with a periodic pattern λ_l on the right describing its behaviors for phase 0 and 1. If a latch is missing from a cube for a certain phase, it has no constraint thus we use the equality of the latch to itself in the signal. Latch a turns out to be a simple clock delayed by one step. Latches b and d behave clock-like but only in phase 0. Latch c always has the opposite value of latch b in phase 1. Note that we could also have $\neg c$ in phase 1 of signal λ_b but choosing a single representative for a set of equivalent signals is beneficial for the following simplification steps.

Definition 2 (Periodic Signal). *Given a circuit $C = (I, L, R, F, P)$ and a cube lasso $c_0, \ldots c_\delta, \ldots, c_{\delta+\omega}$. A periodic signal λ_l for a latch $l \in L$ is defined as $\lambda_l = (d, [v^0, \ldots, v^{n-1}])$ where $d \in \mathbb{N}$, $n \in \mathbb{N}^+$ and v^i is a latch literal or a constant, with $d \leq \delta$. We further require that there exist $k^0, k^1 \in \mathbb{N}^+$ with*

$k^0 \cdot n + d = \delta$ *and* $k^1 \cdot n = \omega + 1$ *such that for all* $i \in [0, n)$ *and* $j \in [0, k^0 + k^1)$ *we have* $c_{i+j \cdot n} \Rightarrow (l \simeq v^i)$.

For a signal $\lambda_l = (d, [v^0, \ldots, v^{n-1}])$ we will write λ_l^i to refer to the i-th element of $[v^0, \ldots, v^{n-1}]$, which we refer to as its phase. See Fig. 1 for an example where $k^0 = 1$ and $k^1 = 2$.

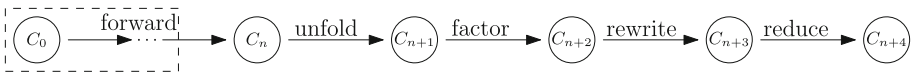

Fig. 2. Circuit transformation using phase abstraction.

4 Extending Phase Abstraction

In this section, we revisit and extend phase abstraction by defining it as a sequence of preprocessing steps, as illustrated in Fig. 2. Differently from the approach in [16], we present phase abstraction as part of a compositional framework, that handles a more general class of periodic signals. As our approach subsumes temporal decomposition adopted from the framework in [57], we first apply *circuit forwarding* [57] for duration d (i.e., unrolling the reset states of a circuit by d steps) before unfolding is performed.

Figure 2 illustrates the flow of phase abstraction. The process begins by using cube simulation to identify a set of periodic signals as defined in Sect. 3 and computing an optimal duration and phase number based on a selected cube lasso. Once the circuit is unfolded n times, factoring is performed by assigning constant values to the clock-like signals as well as replacing latches with their equivalent or antivalent representative latches in each phase. Next, the property is rewritten by applying structural rewriting techniques and the circuit is further simplified using cone-of-influence reduction. Finally, the simplified circuit (C_{n+4} in Fig. 2) is checked using a base model checking approach such as IC3/PDR [17] or continue to be preprocessed further.

In Fig. 3, we show intuitively an example of a circuit with 4-bit states representing 0,...,9 and so on, where the initial state is 0. After forwarding the circuit by one step ($d = 1$), the initial state becomes 1. Subsequently with an unfolding of $n = 3$, every state (marked with rectangles) in the unfolded circuit consists of three states from the original circuit. We introduce the formal definitions below.

Unfolding a circuit simply means to copy the transition function multiple times to compute n steps of the original circuit at once. Each copy of the transition function only has to deal with a single phase and can therefore be optimized.

Definition 3 (Unfolded circuit). *Given a circuit* $C = (I, L, R, F, P)$ *and a phase number* $n \in \mathbb{N}^+$. *The unfolded circuit* $C' = (I', L', R', F', P')$ *is:*

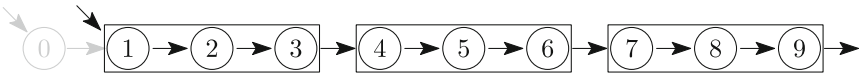

Fig. 3. An example of a forwarded ($d = 1$) and unfolded ($n = 3$) circuit. The circles denote states in the original circuit (0 is the initial state). The rectangle are states in the unfolded circuit.

1. $I' = I^0 \cup \cdots \cup I^{n-1}$; $L' = L^0 \cup \cdots \cup L^{n-1}$.
2. $R' = \{r'_l \mid l \in L'\}$: *for* $l \in L^0, r'_l = r_l$;
   for $i \in (0,n), l^i \in L^i, r'_{l^i} = F(I^i, L^{i-1})$.
3. $F' = \{f'_l \mid l \in L'\}$: *for* $l \in L^0, f'_l = f_l(I^0, L^{n-1})$;
   for $i \in (0,n), l^i \in L^i, f'_{l^i} = f_{l^i}(I^i, L^{i-1})$.
4. $P' = \bigwedge_{i \in [0,n)} P(I^i, L^i)$.

We obtain a simplified circuit by replacing the periodic signals with constants and equivalent/antivalent latches in the unfolded circuit.

Definition 4 (Factor circuit). *For a fixed duration d and phase number n, given a d-forwarded and n-unfolded circuit $C = (I, L, R, F, P)$ and a periodic signal with duration d and phase number n for each latch, the factor circuit $C' = (I, L, R', F', P)$ is defined by:*

$R' = \{r'_l \mid l \in L\}$:
- $r'_{l^i} = \lambda^i_l$, *if* $\lambda^i_l \in \{\bot, \top\}$;
- $r'_{l^i} = r_{\lambda^i_l}$, *if* $\lambda^i_l \in L$.
- $r'_{l^i} = \neg r_{\neg\lambda^i_l}$, *otherwise.*

$F' = \{f'_l \mid l \in L\}$:
- $f'_{l^i} = \lambda^i_l$, *if* $\lambda^i_l \in \{\bot, \top\}$;
- $f'_{l^i} = f_{\lambda^i_l}$, *if* $\lambda^i_l \in L$.
- $f'_{l^i} = \neg f_{\neg\lambda^i_l}$, *otherwise.*

Replaced latches will be removed by a combination of rewriting and cone-of-influence reduction introduced in the following definitions. There are various rewriting techniques also including SAT sweeping [30,38,43–45,59].

Definition 5 (Rewrite circuit). *Given a circuit $C = (I, L, R, F, P)$, a rewrite circuit $C' = (I, L, R, F, P')$ satisfies $P \equiv P'$.*

For a given circuit, we apply cone-of-influence reduction to obtain a reduced circuit such that latches and inputs outside the cone of influence are removed.

Definition 6 (Reduced circuit). *Given a circuit $C = (I, L, R, F, P)$. The reduced circuit $C' = (I', L', R', F', P)$ is defined as follows:*
- $I' = I \cap coi(P)$;
- $L' = L \cap coi(P)$;
- $R' = \{r_l \mid l \in L'\}$;
- $F' = \{f_l \mid l \in L'\}$,

where the cone of influence of the property $coi(P) \subseteq (I \cup L)$ is defined as the smallest set of inputs and latches such that $vars(P) \subseteq coi(P)$ as well as $vars(r_l) \subseteq coi(P)$ and $vars(f_l) \subseteq coi(P)$ for all latches $l \in coi(P)$.

5 Certification

We define a revised certificate format that allows smaller and more optimized certificates. We then propose a method for producing certificates for phase abstraction. The proofs for our main theorems can be found in the Appendix.

5.1 Restricted Simulation

In the following, we define a new variant of the stratified simulation relation [15], which we call *restricted simulation*, that considers the intersection of latches shared between two given circuits as a common component.

Definition 7 (Restricted Simulation). *Given stratified circuits C' and C with $C' = (I', L', R', F', P')$ and $C = (I, L, R, F, P)$. We say C'* simulates *C under the* restricted simulation relation *iff*

1. *For $l \in (L \cap L'), r_l(I, L) \equiv r'_l(I', L')$.*
2. *For $l \in (L \cap L'), f_l(I, L) \equiv f'_l(I', L')$.*
3. *$P'(I', L') \Rightarrow P(I, L)$.*

This new simulation relation differs from [15,56], where inputs were required to be identical in both circuits ($I = I'$), and latches in C had to form a subset of latches in C' ($L \subseteq L'$). Therefore, under those previous "combinational" [56] or "stratified" [15] simulation relations the simulating circuit C' cannot have fewer latches than L. This is a feature we need for instance when incorporating certificates for cone-of-influence reduction [22], a common preprocessing technique. It opens up the possibility to reduce certificate sizes substantially.

Still, as for stratified simulation, restricted simulation can be verified by three simple SAT checks, i.e., separately for each of the three requirements in Definition 7.

Definition 8 (Semantic independence). *Let $\mathcal{V}$ be a set of variables and $v \in \mathcal{V}$. Then a formula $f(\mathcal{V})$ is said to be semantically independent of v iff*

$$f(\mathcal{V})|_v \equiv f(\mathcal{V})|_{\neg v}.$$

Semantic dependency [28,40,49,53] allows us to assume that a formula only depends on a subset of variables, which without loss of generality simplifies proofs used for the rest of this section. The stratified assumption for reset functions entails no cyclic dependencies thus $R'(L')$ is satisfiable. A reset state in a circuit is simply a satisfying assignment to the reset predicate $R(L)$. Based on the reset condition (Definition 7.1), it is however necessary to show that for every reset state in C it can always be extended to a reset state in C', while the common variables have the same assignment in both circuits. This is stated in the lemma below, and the proofs can be found in the Appendix.

Lemma 1. *Let $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$ be two stratified circuits satisfying the reset condition defined in Definition 7.1. Then $R'(L \cap L')$ is semantically dependent only on their common variables.*

In fact, semantic independence is a direct consequence of restricted simulation; thus no separate check is required. We make a further remark that if the reset function is dependent on an input variable, then it has to be an input variable common to both circuits.

Based on this, we conclude with the main theorem for restricted simulation such that C is safe if C' is safe (i.e., no bad state that violates the property is reachable from any initial state).

Theorem 1. *Let $C = (I, L, R, F, P)$ and $C' = (I', L', R', F', P')$ be two stratified circuits, where C' simulates C under restricted simulation.*
If C' is safe, then C is also safe.

Intuitively, if there is an unsafe trace in C, Definition 7.1 together with Lemma 1 allow us to find a simulating reset state and transition it with Definition 7.2 to a simulating state also violating the property in C' by Definition 7.3. Here a state in C' simulates a state in C if they match on all common variables. Building on this, we present witness circuits as a format for certificates. Verifying the restricted simulation relation requires three SAT checks, and another three SAT checks are needed for validating the inductive invariant [56]. Therefore certification requires in total six SAT checks as well as a polynomial time check for reset stratification.

Definition 9 (Witness circuit). *Let $C = (I, L, R, F, P)$ be a stratified circuit. A witness circuit $W = (J, M, S, G, Q)$ of C satisfies the following:*

- *W simulates C under the restricted simulation relation.*
- *Q is an inductive invariant in W.*

The witness circuit format subsumes [15,57], thus every witness circuit in their format is also valid under Definition 9.

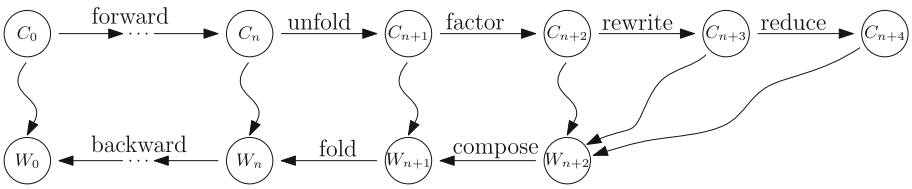

Fig. 4. Certification for (extended) phase abstraction. Base model checking is performed on circuit C_{n+4}, which produces a witness circuit W_{n+2}, that certifies C_{n+2}, C_{n+3}, and C_{n+4}. We construct step-wise to obtain W_0, which is a certificate for the entire model checking procedure.

5.2 Certifying Phase Abstraction

The certificate format is generic, subsumes [57], and is designed to potentially be used as a standard in future hardware model checking competitions. We proceed

to demonstrate how a certificate can be constructed for a model checking pipeline that includes phase abstraction. The theorems in this section state that this construction guarantees that a certificate will be produced. We illustrate our certification pipeline in Fig. 4. After phase abstraction and base model checking, we can build a certificate backwards based on the certificate produced by the base model checker. The following theorem states that the witness circuit of the reduced circuit serves as a witness circuit for the original circuit too.

Theorem 2. *Given a circuit* $C = (I, L, R, F, P)$ *and its reduced circuit* $C' = (I', L', R', F', P')$. *A witness circuit of* C' *is also a witness circuit of* C.

The outcome of rewriting is a circuit with a simplified property that maintains semantic equivalence with the original property. Therefore in our framework, the certificate for the simplified property is also valid for the original property. Furthermore, certificates can be optimized by rewriting at any stage. We summarize this in the following proposition.

Proposition 1. *Given a circuit* C *and its rewrite circuit* C'. *A witness circuit of* C' *is also a witness circuit of* C.

We define the composite witness circuit to combine the certificates for cube simulation and the factor circuit.

Definition 10 (Composite witness circuit). *Given a stratified circuit* $C = (I, L, R, F, P)$ *and its factor circuit* $C' = (I', L', R', F', P')$, *and the unfolded loop invariant* $\phi = \vee_{i\in[0,m)} \wedge_{j\in[0,n)} c_{i*n+j+d}$, *with* $m = (\delta + \omega - d + 1)/n$, *obtained from the cube lasso. Let* $W' = (J', M', S', G', Q')$ *be a witness circuit of* C'. *The composite witness circuit* $W = (J, M, S, G, Q)$ *is defined as follows:*

1. $J = I \cup J'$.
2. $M = L \cup (M' \backslash L')$.
3. $S = \{s_l \mid l \in M\}$:
   (a) *for* $l \in L$, $s_l = r_l$;
   (b) *for* $l \in M' \backslash L'$, $s_l = s'_l$.
4. $G = \{g_l \mid l \in M\}$:
   (a) *for* $l \in L$, $g_l = f_l$;
   (b) *for* $l \in M' \backslash L'$, $g_l = g'_l$.
5. $Q = \phi(L) \wedge Q'(J', M')$.

Theorem 3. *Given circuit* $C = (I, L, R, F, P)$, *and factor circuit* $C' = (I', L', R', F', P')$. *Let* $W' = (J', M', S', G', Q')$ *be a witness circuit of* C', *and* $W = (J, M, S, G, Q)$ *constructed as in Definition 10. Then* W *is a witness circuit of* C.

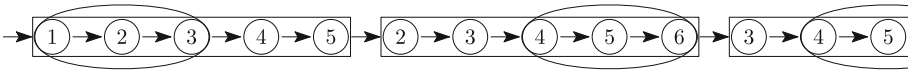

Fig. 5. Every fully initialized state of a 3-folded witness circuit contains 3 original states that form an unfolded state. Two consecutive 3-folded states contain either the same unfolded states or two states consecutive in the unfolded circuit.

In the construction of an n-folded witness circuit from the unfolded witness W', a single instance of W''s latches (N), yet multiples of the original latches L are used. As illustrated in Fig. 5, these L record a history, contrasting with their role in the unfolded circuit where they calculate multi-step transitions.

Definition 11 (n-folded witness circuit). *Given a circuit $C = (I, L, R, F, P)$ with a phase number $n \in \mathbb{N}^+$, and its unfolded circuit $C' = (I', L', R', F', P')$. Let $W' = (J', M', S', G', Q')$ be the witness circuit of C'. The n-folded witness circuit $W = (J, M, S, G, Q)$ is defined as follows:*

1. $J = I^0 \cup J^0$, *where* I^0 *and* J^0 *are* I *and* J' *respectively.*
2. $M = I^1 \cdots I^m \cup L^0 \cdots L^m \cup N \cup J^1 \cup \{b^0 \cdots b^m, e^0 \cdots e^{n-2}\}$, *where* $m = 2 \times n - 2$, $N = M' \setminus L'$, *and* I^i, L^i *are copies of* I *and* L, *and* J^1 *is a copy of* J'.
3. $S = \{s_l \mid l \in M\}$:
   (a) $s_{b^0} = \top$;
   (b) *For* $i \in (0, m]$, $s_{b^i} = \bot$.
   (c) *For* $i \in [0, n-1)$, $s_{e^i} = \bot$.
   (d) *For* $l \in L^0$, $s_l = r'_l$.
   (e) *For* $l \in (I^1 \cdots I^m \cup L^1 \cdots L^m \cup J^1)$, $s_l = l$.
   (f) *For* $l \in N$, $s_l = s'_l$.
4. $G = \{g_l \mid l \in M\}$:
   (a) $g_{b^0} = \top$.
   (b) *For* $i \in [1, m]$, $g_{b^i} = b^{i-1}$.
   (c) $g_{e^0} = b^{n-1} \wedge \neg e^{n-2}$.
   (d) *For* $i \in [1, n-1)$, $g_{e^i} = e^{i-1} \wedge \neg e^{n-2}$.
   (e) *For* $l \in L^0$, $g_l = f_l$.
   (f) *For* $l^1 \in J^1$, $g_{l^1} = l^0$.
   (g) *For* $i \in [1, m]$, $l^i \in (I^i \cup L^i)$, $g_{l^i} = l^{i-1}$.
   (h) *For* $l \in N$,
   $g_l = ite(e^{n-2}, g'_l(J^1, M' \cap (I^{m-n+1} \cdots I^m \cdots L^{m-n+1} \cdots L^m \cup N)), l)$.
5. $Q = \bigwedge_{i \in [0,6]} q^i$:
   (a) $q^0 = P(I^0, L^0)$.
   (b) $q^1 = b^0$.
   (c) $q^2 = \bigwedge_{i \in [1,m]} (b^i \rightarrow b^{i-1})$.
   (d) $q^3 = \bigwedge_{i \in [1,m]} (b^i \rightarrow (L^i \simeq F(I^{i-1}, L^{i-1})))$.
   (e) $q^4 = \bigwedge_{i \in [1,m]} ((\neg b^i \wedge b^{i-1}) \rightarrow (R(L^{i-1}) \wedge S'(N)))$.
   (f) $q^5 = b^m \rightarrow (\bigvee_{i \in [0,n)} ((\bigwedge_{j \in [i,n-1)} \neg e^j) \wedge (\bigwedge_{j \in [0,i)} e^j) \wedge Q'(J^0, M' \cap (L^i \cdots \cup L^{i+n-1} \cup N)))$.
   (g) $q^6 = \bigwedge_{i \in [1,n-2]} (e^i \rightarrow e^{i-1})$.
   (h) $q^7 = \bigwedge_{i \in [0,n-2]} (e^i \rightarrow b^{n+i})$.

(i) $q^8 = \bigwedge_{i \in [0, n-2)} ((\neg b^m \wedge b^{n+i}) \rightarrow e^i)$.

The b^is are used for encoding initialization. So that inductiveness is ensured when not all copies are initialized. The $n-1$ bits e^i are used to determine which set of n consecutive original states form an unfolded state (a state in the unfolded circuit). This information is used to determine on which copies the unfolded property needs to hold and to transition the latches in N (the part of the witness circuit added by the backend model checker) once every n steps.

Theorem 4. *Given a circuit* $C = (I, L, R, F, P)$ *with a phase number* $n \in \mathbb{N}^+$*, its unfolded cicuit* $C' = (I', L', R', F', P')$ *with a witness circuit* $W' = (J', M', S', G', Q')$*. Let* $W = (J, M, S, G, Q)$ *be the circuit constructed as in Definition 11. Then* W *is a witness circuit of* C*.*

After the witness circuit has been folded, the same construction from [57] can be used to construct the backward witness. With that, the pipeline outlined in Fig. 4 is completed. If phase abstraction is the first technique applied by the model checker, a final witness is obtained. Otherwise, further witness processing steps still need to be performed. An example of the entire process is illustrated in Fig. 6.

6 Implementation

In this section, we present MC2, a certifying model checker implementing phase abstraction and IC3. We implement our own IC3 since no existing model checker supports reset functions or produces certificates in the desired format. We used fuzzing to increase trust in our tool. The version of MC2 used for the evaluation, was tested on over 25 million randomly generated circuits [14] in combination with random parameter configurations. All produced certificates where verified.

To extract periodic signals we perform ternary simulation [52] while using a forward-subsumption algorithm based on a one-watch-literal data structure [58] to identify supersets of previously visited cubes, and thereby a set of cube lassos. For each cube lasso we consider every factor of the loop length ω as a phase number candidate n. We also consider every duration d, that renders the leftover tail length $(\delta - d)$ divisible by n. To keep the circuit sizes manageable, we limit both n and d to a maximum of 8. We call each pair (d, n) an unfolding candidate and compute the corresponding periodic signal (Definition 2) for each latch.

For each phase, equivalences are identified by inserting a bit string corresponding to the signs of each latch into a hash table. After identifying the signals, forwarding and unfolding are performed on a copy of the circuit, followed by rudimentary rewriting. Currently the rewriting does not include structural hashing and is mostly limited to constant propagation. Afterwards a sequential cone-of-influence analysis starting from the property is performed. After performing these steps for each candidate, we pick the duration-phase pair that yields a circuit with the fewest latches and give it to a backend model checker.

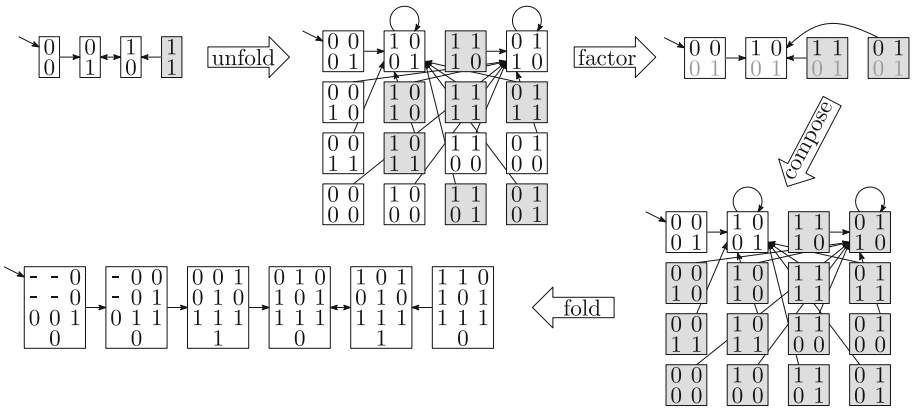

Fig. 6. A concrete example of the model checking and certification pipeline. The original circuit has two latches; the bottom latch alternates and the top copies the previous value of this clock. The property is that at least one bit is unset. Bad states are marked gray. After unfolding with phase number two, the size of the state space is squared. Since the bottom bit is periodic, we can replace it with a constant in each phase (factor). On this circuit terminal model checking is performed, since the property is already inductive (no transition from good to bad), the circuit serves as its own witness. To produce the final witness circuit, the clock is added back as a latch, and the property is extended with the loop invariant asserting that the clock has the correct value for each phase. Lastly, the circuit is *folded* to match the speed of the original circuit. Three initialization bits b^i are introduced and one additional bit e^0 that determins which pair of consecutive states need to fulfill the property (0 for the right pair and 1 for the left). This check is only part of the property once full initialization is reached. For this final witness circuit, only the good states are depicted. Also, the first two states represent sets of good states with the same behavior.

We evaluated the preprocessor on three backend model checkers: the open-source k-induction-based model checker McAiger [12](Kind in the following), the state-of-the-art IC3 implementation in ABC [18] and our own version of IC3 that supports reset functions and produces certificates. Since ABC does not support reset functions, it is not able to model check any forwarded circuit (note that implementing this feature on ABC is also a non-trivial task), therefore for this configuration we only ran phase abstraction without forwarding thus no temporal decomposition.

Our IC3 implementation on MC2 does feature cube minimization via ternary simulation [25], however it is missing proof-obligation rescheduling. In fact, we currently use a simple stack of proof obligations as opposed to a priority queue. Despite using one SAT solver instance per frame, we also do not feature cones-on-demand, but instead always translate the entire circuit using Tseitin [55].

Lastly, we also modified the open source implementation of Certifaiger [21] to support certificates based on restricted simulation. For a witness circuit C' of

C, the new certificate checker encodes the following six checks as combinatorial AIGER circuits and then uses the `aigtocnf` to translate them to SAT:

Ⓐ The property of C' holds in all initial states.
Ⓑ The property of C' implies the property for successor states.
Ⓒ The property of C' holds in all good states.
Ⓓ The reset functions of common latches are equivalent. (Definition 7.1)
Ⓔ The transition functions of common latches are equivalent. (Definition 7.2)
Ⓕ The property of C' implies the property of C. (Definition 7. 3)

The first three checks are unchanged and encode the standard check for P' being an inductive invariant in C'. Since P' is both the inductive invariant and the property we are checking, Ⓒ can technically be omitted. However, in our implementation, the inductiveness checker is an independent component from the simulation checker and would also works for scenarios where the inductive invariant is a strengthening of the property in C'.

7 Experimental Evaluation

This section presents experimental results for evaluating the impact of preprocessing on the different backends, as well as the effectiveness of our proposed certification approach. The experiments were run in parallel on a cluster of 32 nodes. Each node was equipped with two 8-core Intel Xeon E5-2620 v4 CPUs running at 2.10 GHz and 128 GB of main memory. We allocated 8 instances to each node, with a timeout of 5000 s for model checking and 10 000 s for certificate checking. Memory is limit to 8 GB per instance in both cases.

The benchmarks are obtained from HWMCC2010 [13] which contains a good number of industrial problems. As we observe from the experiments in general, prepossessing is usually fast. Ignoring one outlier in our benchmark set, it completes within an average of 0.07 seconds and evaluates no more than 17 unfolding candidates per benchmark. Interestingly, for the outlier "bobsmnut1", 3019 unfoldings are computed for 179 different cube lassos within 34 seconds.

Table 1 presents the effect of our preprocessing on different backends, further illustrated in Fig. 7. Our preprocessor was able to improve the performance of the sophisticated IC3/PDR implementation in ABC, allowing us to solve five more instances, all from the intel family. For each benchmark from this family, our heuristic computed an optimal phase number of 2. A likely explanation for this is that the real-world industrial designs tend to contain strict two-phase clocks [6]. The positive effect of phase abstraction is also clear in combination with the k-induction (Kind) backend. Circuit forwarding provides a further improvement, that is especially notable on the prodcell benchmarks. These also illustrate how forwarding enables more successful unfolding. Without forwarding, preprocessing only unfolds 61 out of the 818 benchmarks with an average phase number of 2, with forwarding 152 circuits are unfolded with an average of 4.

Even though our prototype implementation of IC3 is missing a number of important features present in ABC, it still solves a large number of benchmarks.

Table 1. We presents the effect of preprocessing in combination with different backend engines on model checking time. We compare no preprocessing to only phase abstraction without forwarding (PA) and full preprocessing (Full). Note that, ABC does not support reset functions and can therefore not be combined with full preprocessing. For each model we present the phase number without forwarding $\overline{n}$ for PA and the duration d and phase number n corresponding to Full. Models where the property holds are marked as safe. The first two rows present the number of solved instances and the PAR2 score [29] over all 818 benchmarks. The table shows all instances where preprocessing had either a positive or negative impact on model checking success, with the exception of those instances rendered unsolvable for our IC3 implementation by forwarding.

	Model				ABC		Our IC3			Kind		
	Safe	$\overline{n}$	d	n		PA		PA	Full		PA	Full
Solved					740	**745**	715	715	604	533	538	544
PAR2					996	**941**	1357	1351	2699	3533	3472	3399
abp4p2ff	✓	1	1	1	1.12	**1.08**	6.35	6.23	6.18	2.50	2.50	
bjrb07		3	0	3	0.12	0.10	0.11	**0.03**	0.07		**0.03**	0.04
nusmvb5p2		5	0	5	0.12	0.10	**0.01**	**0.01**	**0.01**		**0.01**	**0.01**
nusmvb10p2		5	0	5	0.22	0.13	0.10	0.03	0.04		**0.02**	**0.02**
prodcell0	✓	1	5	8	26.97	27.07	228.46	243.73	49.76			**2.37**
prodcell0neg	✓	1	5	8	16.36	15.93	230.57	230.67	36.62			**2.39**
prodcell1	✓	1	7	8	23.45	23.38	654.21	665.86	59.67			**4.43**
prodcell1neg	✓	1	7	8	28.36	28.33	681.11	738.61	61.74			**4.48**
prodcell2	✓	1	7	8	24.98	24.58	661.71	663.37	56.74			**4.43**
prodcell2neg	✓	1	7	8	20.23	20.28	778.39	768.75	56.14			**4.47**
bc57sen0neg	✓	1	1	1	503.61	**494.55**	910.72	906.87	1760.41			830.92
abp4ptimo	✓	1	1	1	4.14	**4.13**	28.93	29.91	6.32			608.55
boblivea		1	2	1	3.70	**3.68**			7.85			
bobsm5378d2		1	8	1	**4.04**	4.12			88.36			
bobsmnut1		8	5	8	**10.95**	40.08			2504.07			
prodcell3neg	✓	2	2	8	27.88	10.86	310.22				837.43	**2.73**
prodcell4neg	✓	2	2	8	44.31	9.90	404.12				26.04	**2.77**
prodcell3	✓	2	2	8	23.45	11.24	320.23			1103.29	19.22	**2.48**
prodcell4	✓	2	2	8	31.40	10.08	398.83			29.71	18.67	**2.68**
pdtvisvsar29		1	2	5					1523.73	0.36	**0.29**	0.40
intel042	✓	1	3	2						**3876.04**	4061.38	
intel022		2	2	2		**1852.29**						
intel021		2	2	2		2752.86		**651.56**				
intel023		2	2	2		**2257.94**		3728.38				
intel029		2	2	2		**2550.14**		3437.64				
intel024		2	2	2		**167.96**	676.64	4526.60				
intel019		2	2	2				**2716.40**				

However, as opposed to ABC it does lose a number of benchmarks with phase abstraction. This can be explained by the lack of sophisticated rewriting that can exploit the unfolded circuits structure. The addition of forwarding is highly

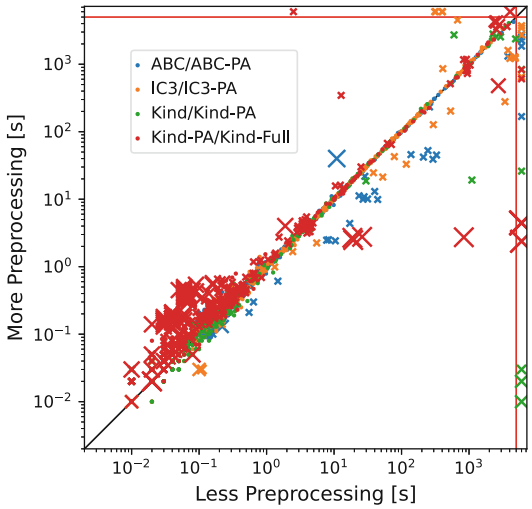

Fig. 7. Comparison of model checking performance. We compare four pairs of configurations; the three backend engines with and without phase abstraction (with fixed duration 0) and for Kind we present the effect of additionally allowing forwarding. The size of the markers represents $n + d$. The dots represent instances where the preprocessing heuristic decided not to alter the circuit. The red lines mark the timeout of 5000 s. Markers beyond that line represent instances solved by one configuration but not the other. (Color figure online)

detrimental to performance, losing 115 instances. This is due to our implementation following the PDR design outlined in [25]. It requires any blocked cube not to intersect the initial states after generalization. If only a single reset state exists this check is linear in the size of the cube. However, in the presence of reset functions it is implemented with a SAT call. While also slower the main problem however is that the reset-intersection check is also more likely to block generalization. On the 115 lost benchmarks generalization failed 96% of the time, while it only failed in 1.8% of the cases without forwarding. We keep the optimization of our IC3 implementation in the presence of reset functions for future work.

Figure 8 displays certification results on MC2 in comparison to model checking time. IC3 provides certificates that are easily verifiable, as confirmed by our experiments with cumulative overhead of only 3%. The addition of phase abstraction (i.e., including constructing n-folded witnesses as in Fig. 4, without witness back-warding) does not bring significant additional overhead. When forwarding is allowed, the certification overhead increases to 10%. The run time of certificates generation and encoding to SAT is negligible for all configurations. The certification time is dominated by the SAT solving time for the transition (Definition 7.2) and consecution check. Overall, this is a significant improvement over related work from [57] which reported 1154% overhead on the same set of benchmarks using a k-induction engine as the backend.

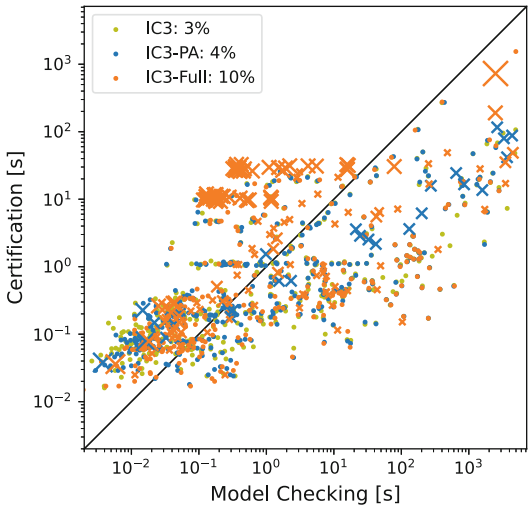

Fig. 8. Certification vs. model checking time for three configurations of our IC3 engine. The legend shows the cumulative overhead of including certification for all solved instances. The size of the markers represents $n + d$. The dots represent instances where preprocessing did not alter the circuit.

8 Conclusion

In this paper, we present a certificate format that can be effectively validated by an independent certificate checker. We demonstrate its versatility by applying it to an extended version of phase abstraction, which we introduce as one of the contributions of this paper. We have implemented the proposed approach on a new certifying model checker MC2. The experimental results on HWMCC instances show that our approach is effective and yields very small certification overhead, as a vast improvement over related work. Our certificate format allows for smaller certificates and is designed to be possibly used in hardware model checking competitions as a standardized format.

Beyond increasing trust in model checking, certificates can be utilized in many other scenarios. For instance, such certificates will allow the use of model checkers as additional hammers in interactive theorem provers such as Isabelle [48] via Sledgehammer [50], with the potential of significantly reducing the effort needed for using theorem provers in domains where model checking is essential, such as formal hardware verification, our main application of interest. Currently in Isabelle, Sledgehammer allows to encode the current goal for automatic theorem provers or SMT solvers and then call one of many tools to solve the problem. The tool then provides a certificate which is lifted to a proof that can be replayed in Isabelle. We plan to add our model checker as an additional hammer to increase the automatic proof capability of Isabelle. This further motivates us to investigate certificate trimming via SAT proofs.

Acknowledgements. This work is supported by the Austrian Science Fund (FWF) under the project W1255-N23, the LIT AI Lab funded by the State of Upper Austria, the ERC-2020-AdG 101020093, the Academy of Finland under the project 336092 and by a gift from Intel Corporation.

References

1. Amjad, H.: Programming a symbolic model checker in a fully expansive theorem prover. In: Basin, D., Wolff, B. (eds.) TPHOLs 2003. LNCS, vol. 2758, pp. 171–187. Springer, Heidelberg (2003). https://doi.org/10.1007/10930755_11
2. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
3. Balyo, T., Heule, M.J.H.: Proceedings of SAT competition 2016 – solver and benchmark descriptions. Department of Computer Science Series of Publications B, vol. B-2016-1. University of Helsinki (2016)
4. Barbosa, H., et al.: Generating and exploiting automated reasoning proof certificates. Commun. ACM **66**(10), 86–95 (2023). https://doi.org/10.1145/3587692
5. Barbosa, H., et al.: Flexible proof production in an industrial-strength SMT solver. In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) IJCAR 2022. LNCS, vol. 13385, pp. 15–35. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-10769-6_3
6. Baumgartner, J., Heyman, T., Singhal, V., Aziz, A.: Model checking the IBM gigahertz processor: an abstraction algorithm for high-performance netlists. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. LNCS, vol. 1633, pp. 72–83. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48683-6_9
7. Baumgartner, J., Heyman, T., Singhal, V., Aziz, A.: An abstraction algorithm for the verification of level-sensitive latch-based netlists. Formal Methods Syst. Des. **23**, 39–65 (2003)
8. Beyer, D., Chien, P., Lee, N.: Bridging hardware and software analysis with Btor2C: a word-level-circuit-to-C translator. In: Sankaranarayanan, S., Sharygina, N. (eds.) TACAS 2023. LNCS, vol. 13994, pp. 152–172. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30820-8_12
9. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: exchanging verification results between verifiers. In: SIGSOFT FSE, pp. 326–337. ACM (2016)
10. Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Lemberger, T., Tautschnig, M.: Verification witnesses. ACM Trans. Softw. Eng. Methodol. **31**(4), 57:1–57:69 (2022)
11. Biere, A.: Bounded model checking. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability - Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336, pp. 739–764. IOS Press (2021). https://doi.org/10.3233/FAIA201002
12. Biere, A., Brummayer, R.: Consistency checking of all different constraints over bit-vectors within a SAT solver. In: FMCAD, pp. 1–4. IEEE (2008)
13. Biere, A., Claessen, K.: Hardware model checking competition 2010 (2010). http://fmv.jku.at/hwmcc10/
14. Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Technical report, FMV Reports Series, Inst. FMV, JKU Linz, Austria (2011)
15. Biere, A., Yu, E., Froleyks, N.: Stratified certification for k-induction. In: FMCAD, vol. 3, p. 59. TU Wien Academic Press (2022)
16. Bjesse, P., Kukula, J.H.: Automatic generalized phase abstraction for formal verification. In: ICCAD, pp. 1076–1082. IEEE Computer Society (2005)

17. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18275-4_7
18. Brayton, R., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 24–40. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_5
19. Case, M.L., Baumgartner, J., Mony, H., Kanzelman, R.: Approximate reachability with combined symbolic and ternary simulation. In: FMCAD, pp. 109–115. FMCAD Inc. (2011)
20. Case, M.L., Mony, H., Baumgartner, J., Kanzelman, R.: Enhanced verification by temporal decomposition. In: FMCAD, pp. 17–24. IEEE (2009)
21. Certifaiger: Certifaiger (2021). http://fmv.jku.at/certifaiger
22. Clarke, E.M., Grumberg, O., Kroening, D., Peled, D.A., Veith, H.: Model Checking, 2nd edn. MIT Press, Cambridge (2018)
23. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8
24. Degtyarev, A., Voronkov, A.: Equality reasoning in sequent-based calculi. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 611–706. Elsevier and MIT Press (2001)
25. Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: FMCAD, pp. 125–134. FMCAD Inc. (2011)
26. van Eijk, C.A.J., Jess, J.A.G.: Exploiting functional dependencies in finite state machine verification. In: ED&TC, pp. 9–14. IEEE Computer Society (1996)
27. Esparza, J., Lammich, P., Neumann, R., Nipkow, T., Schimpf, A., Smaus, J.-G.: A fully verified executable LTL model checker. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 463–478. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_31
28. Fleury, M., Biere, A.: Mining definitions in Kissat with Kittens. Formal Methods Syst. Des. 1–24 (2023)
29. Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: Sat competition 2020. Artif. Intell. **301**, 103572 (2021)
30. Fujita, M.: Toward unification of synthesis and verification in topologically constrained logic design. Proc. IEEE **103**(11), 2052–2060 (2015)
31. Griggio, A., Roveri, M., Tonetta, S.: Certifying proofs for LTL model checking. In: FMCAD, pp. 1–9. IEEE (2018)
32. Griggio, A., Roveri, M., Tonetta, S.: Certifying proofs for SAT-based model checking. Formal Methods Syst. Des. **57**(2), 178–210 (2021)
33. Heule, M., Hunt, W., Kaufmann, M., Wetzler, N.: Efficient, verified checking of propositional proofs. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP 2017. LNCS, vol. 10499, pp. 269–284. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66107-0_18
34. Heule, M.J.: Proofs of unsatisfiability. In: Handbook of Satisfiability, pp. 635–668. IOS Press (2021)
35. Heule, M.J., Biere, A.: Proofs for satisfiability problems. In: All About Proofs, Proofs for All, vol. 55, no. 1, pp. 1–22 (2015)
36. Hoenicke, J., Schindler, T.: A simple proof format for SMT. In: Déharbe, D., Hyvärinen, A.E.J. (eds.) Proceedings of the 20th Internal Workshop on Satisfiability Modulo Theories co-located with the 11th International Joint Conference on Automated Reasoning (IJCAR 2022) part of the 8th Federated Logic Conference (FLoC 2022), Haifa, Israel, 11–12 August 2022. CEUR Workshop Proceedings, vol. 3185, pp. 54–70. CEUR-WS.org (2022)

37. Kaufmann, D., Fleury, M., Biere, A., Kauers, M.: Practical algebraic calculus and nullstellensatz with the checkers pacheck and pastèque and nuss-checker. Formal Methods Syst. Des. 1–35 (2022)
38. Kuehlmann, A., Paruthi, V., Krohm, F., Ganai, M.K.: Robust boolean reasoning for equivalence checking and functional property verification. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **21**(12), 1377–1394 (2002)
39. Kuismin, T., Heljanko, K.: Increasing confidence in liveness model checking results with proofs. In: Bertacco, V., Legay, A. (eds.) HVC 2013. LNCS, vol. 8244, pp. 32–43. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03077-7_3
40. Lagniez, J.M., Lonca, E., Marquis, P.: Definability for model counting. Artif. Intell. **281**, 103229 (2020)
41. Lammich, P.: Efficient verified (UN)SAT certificate checking. J. Autom. Reason. **64**(3), 513–532 (2020)
42. Mebsout, A., Tinelli, C.: Proof certificates for SMT-based model checkers for infinite-state systems. In: FMCAD, pp. 117–124. IEEE (2016)
43. Mishchenko, A., Chatterjee, S., Brayton, R.K.: Dag-aware AIG rewriting a fresh look at combinational logic synthesis. In: DAC, pp. 532–535. ACM (2006)
44. Mishchenko, A., Chatterjee, S., Brayton, R.K., Eén, N.: Improvements to combinational equivalence checking. In: ICCAD, pp. 836–843. ACM (2006)
45. Mishchenko, A., Chatterjee, S., Jiang, R., Brayton, R.K.: FRAIGs: a unifying representation for logic synthesis and verification. Technical report, ERL Technical Report (2005)
46. Mony, H., Baumgartner, J., Aziz, A.: Exploiting constraints in transformation-based verification. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 269–284. Springer, Heidelberg (2005). https://doi.org/10.1007/11560548_21
47. Namjoshi, K.S.: Certifying model checkers. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 2–13. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44585-4_2
48. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45949-9
49. Padoa, A.: Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une theorie déductive quelconque. In: Bibliothèque du Congrès international de philosophie, vol. 3, pp. 309–365 (1901)
50. Paulson, L., Nipkow, T.: The sledgehammer: let automatic theorem provers write your isabelle scripts (2023)
51. Schurr, H., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: towards a generic SMT proof format (extended abstract). In: Keller, C., Fleury, M. (eds.) Proceedings Seventh Workshop on Proof eXchange for Theorem Proving, PxTP 2021, Pittsburg, PA, USA, 11 July 2021. EPTCS, vol. 336, pp. 49–54 (2021). https://doi.org/10.4204/EPTCS.336.6
52. Seger, C.H., Bryant, R.E.: Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods Syst. Des. **6**(2), 147–189 (1995)
53. Slivovsky, F.: Interpolation-based semantic gate extraction and its applications to QBF preprocessing. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 508–528. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_24
54. Sprenger, C.: A verified model checker for the modal μ-calculus in Coq. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 167–183. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054171

55. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970, pp. 466–483 (1983)
56. Yu, E., Biere, A., Heljanko, K.: Progress in certifying hardware model checking results. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12760, pp. 363–386. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81688-9_17
57. Yu, E., Froleyks, N., Biere, A., Heljanko, K.: Towards compositional hardware model checking certification. In: FMCAD (2023)
58. Zhang, L.: On subsumption removal and on-the-fly CNF simplification. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 482–489. Springer, Heidelberg (2005). https://doi.org/10.1007/11499107_42
59. Zhu, Q., Kitchen, N., Kuehlmann, A., Sangiovanni-Vincentelli, A.L.: SAT sweeping with local observability don't-cares. In: DAC, pp. 229–234. ACM (2006)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.