Liveness Proofs for Hardware Model Checking

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

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

Abstract. We introduce a generic certificate format for verifying liveness properties in hardware model checking. The format relies purely on propositional predicates and does not involve explicit counters. Our certificates can be efficiently validated using a fixed number of SAT checks. The proposed format is compatible with state-of-the-art liveness checking algorithms. We present certificate generation for several representative techniques, including rLive, liveness-to-safety reduction, and k-liveness, as well as for a preprocessing method based on stabilizing constraint extraction. Experimental results on benchmarks from the Hardware Model Checking Competition demonstrate that our approach is practically effective with very low certification overhead, and our certificate checker successfully validated all generated certificates.

1 Introduction

In hardware verification, formal properties are traditionally classified as either *safety* or *liveness* properties. A safety property asserts that the system must never reach a designated bad state, whereas a liveness property requires that a certain desirable event eventually occurs. Equivalently, liveness checking ensures that no bad cycle, i.e., a recurring sequence of states violating progress, can be reached. While verifying safety properties is generally more straightforward and conceptually easier, proving liveness for hardware implementations is essential in many industrial verification projects [31, 32].

Substantial progress has recently been made in certifying safety properties for hardware model checking [21], and safety certificates are now mandatory in the Hardware Model Checking Competition (HWMCC). In this setting, model checkers produce machine-checkable proofs of correctness, which are then independently validated by a separate checker, significantly improving the reliability and trustworthiness of the model checkers. In contrast, certification of liveness properties remains comparatively underdeveloped. In particular, a generic certificate format suitable for use in the liveness track of the HWMCC has yet to be developed. The main difficulty in establishing a generic proof format is due to the broad range of algorithms involved, such as k-liveness [13], liveness-to-safety

(L2S) [4], and more recent approaches like rLive [46], combined with the fact that preprocessing and model transformations further complicate proof generation.

In this paper, we address this challenge by proposing a certificate format for bit-level liveness model checking. The certificate takes the form of a circuit (in the same format as the model itself), which can be considered as a generalization of inductive invariants and ranking functions. Certificate checking reduces to a fixed number of simple SAT checks, making validation straightforward and efficient. We formally prove the soundness of this certificate definition. We further show how such proof certificates can be generated from several representative liveness-checking algorithms by adding bookkeeping to the model checkers.

Our contribution is summarized as follows. We present the first certificate format for liveness model checking that is generic enough to cover state-of-the-art liveness checking algorithms including k-liveness, liveness-to-safety reduction, rLive, and stabilizer extraction. We present certificate generation for each of these algorithms according to our defined format. We implemented a prototype model checker that supports all approaches described above, together with a standalone certificate checker. Using benchmarks from the Hardware Model Checking Competition (HWMCC), we experimentally demonstrate that our certification approach is efficient and practical with very low overhead, *at only a small fraction of the overall model checking time*. We further compare our approach against other certification methods and show that our technique significantly outperforms them *by an order of magnitude*.

2 Related Work

Hardware verification relies on symbolic model checking of Linear Temporal Logic (LTL) specifications, which can be expressed as compositions of safety and liveness properties [2, 30]. Popular liveness checking approaches include k-liveness [13, 29], liveness-to-safety (L2S) [4], FAIR [11] based on strongly connected components, k-FAIR, which combines k-liveness and FAIR, and the more recent rLive [46]. All of these reduce liveness verification to invariant checking in some form and thus leverage SAT-based safety engines such as IC3 [9].

The idea of certifying model checkers was first introduced in [35], although no implementation was provided. A different line of work [17, 43] focuses on implementing fully verified model checkers within a theorem prover, an approach that typically requires a tremendous engineering effort and does not scale to the performance of state-of-the-art solvers.

In previous work [20,21,47,48], we proposed a generic certification framework for safety properties, which has been adopted by the HWMCC in 2024. This paper extends that framework to liveness properties.

Certification of LTL properties, including liveness, has previously been studied in [26]. Their approach supports k-liveness and stabilizer extraction, works on arbitrary symbolic transition systems, and can handle arbitrary LTL properties. It is based on a deductive proof system with dedicated rules for each model-checking and preprocessing technique. The model checker encodes its reasoning

as a sequence of SMT formulas, each representing a proof step. To check a certificate, an SMT solver directly verifies the unsatisfiability of these formulas. The model checker is thus trusted to faithfully encode the proof, as the checking process considers neither the original model nor the rules of the proof system.

The certificate checker of [42] extends the approach by building on the theorem prover PVS and its integration with the SMT solver Yices. In [41], the authors add support for rLive certification; however, their approach does not generalize to L2S or preprocessing for liveness checking.

Ranking functions, originally introduced in the context of program termination [15,18], have since been generalized to serve as certificates for liveness properties [1,14]. They have been extensively studied in software model checking [16] and program analysis under a wide range of formulations [10,28,37,38,44], and have also been applied to solving parity games [27]. Complementary approaches rely on a variant of liveness-to-safety transformations [31,36] for infinite systems.

Related proof certificates have further been used to reason about the transitive closure of transition systems [34,39]. More recently, machine-learning-based techniques have been proposed to generate liveness certificates in the form of neural networks [23,24], with a particular focus on word-level designs.

3 Problem Formulation

We assume standard semantics and syntax of propositional logic. We write $\mathbb{B}(\mathcal{V})$ to denote the set of all Boolean formulas over variables $\mathcal{V}$. In the following, we describe our formal model in terms of logical circuits, as a symbolic representation of state transition systems. The state variables are partitioned into input variables and state-holding latch variables.

Definition 1. *A circuit is a tuple* $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$ *such that*

- $\mathcal{I}$ *and* $\mathcal{L}$ *are finite ordered sets of Boolean variables referred to as* inputs *and* latches*, respectively. A state* $s \in \{0,1\}^{\mathcal{I} \cup \mathcal{L}}$ *is a complete valuation of all variables in* $\mathcal{I}$ *and* $\mathcal{L}$*.*
- $\mathcal{R} = \{r_\ell \mid \ell \in \mathcal{L}\} \subseteq \mathbb{B}(\mathcal{I} \cup \mathcal{L})$ *is a collection of* reset functions*. For each latch* $\ell \in \mathcal{L}$*,* r_ℓ *defines the initial value of* ℓ *and may only depend on inputs and lower-ordered latches under a strict partial order* $\prec$ *fixed for the circuit, i.e.,* $r_\ell \in \mathbb{B}(\mathcal{I} \cup \mathcal{L}^{\prec \ell})$ *for each* ℓ*.*
- $\mathcal{F} = \{f_\ell \mid \ell \in \mathcal{L}\} \subseteq \mathbb{B}(\mathcal{I} \cup \mathcal{L})$ *is the collection of* transition functions f_ℓ *defining the next state value for each latch.*
- $\mathcal{C}$*,* $\mathcal{P} \in \mathbb{B}(\mathcal{I} \cup \mathcal{L})$ *denote the environment constraint and the safety signal.*
- $\mathcal{Q} \in \mathbb{B}((\mathcal{I} \cup \mathcal{L})^2)$ *is the liveness signal, which references two states.*

In the above definition, we include properties (safety and liveness signals) as part of the circuit definition. A circuit may be associated with both safety and liveness properties, as properties from LTL formulas are typically translated to the AIGER format [7]. Intuitively, the safety property $\mathcal{P}$ is a formula over $\mathcal{I}$ and $\mathcal{L}$ that must hold in all reachable states. The liveness signal $\mathcal{Q}$ refers

to two states (inputs and latches) explicitly, for ease of certification later. This representation can be viewed as describing edges of the transition relation; as for liveness properties, it is natural to reason about progress by considering pairs of states. We will define the formal semantics for liveness below after explaining the elements of the circuit definition.

To refer to some set of initial states, we define the *reset predicate* for a subset of latches $\mathcal{U} \subseteq \mathcal{L}$ as

$$\mathcal{R}[\mathcal{U}] \;:=\; \bigwedge_{\ell \in \mathcal{U}} \big(\ell \leftrightarrow r_\ell(\mathcal{I}, \mathcal{L}) \big). \tag{1}$$

Intuitively, this formula expresses that the latch values are equal to their reset functions. For the full set of reset states of the circuit, we simply write $\mathcal{R}[\mathcal{L}]$. The order $\prec$ on the latches ensures that $\mathcal{R}[\mathcal{U}]$ is always satisfiable, meaning there is at least one valid reset state of the model.

Next, we define the *transition predicate*. Since a transition relates two consecutive states (s, t), we must refer to two copies of the input and latch variables. We use s to index current-state variables and t to index next-state variables; in particular, ℓ_t denotes the next-state copy of a latch ℓ. The notation

$$\mathcal{F}_{st}[\mathcal{U}] \;:=\; \bigwedge_{\ell \in \mathcal{U}} \big(\ell_t \leftrightarrow f_\ell(\mathcal{I}_s, \mathcal{L}_s) \big) \tag{2}$$

allows describing the unrolling of a circuit by only mentioning a subset of latches. Constraints restrict the set of states to be considered and simplify modeling.

To reduce syntactic clutter, we introduce similar short-form notation for the rest of the predicates, i.e., $\mathcal{C}_s, \mathcal{P}_s$ for $\mathcal{C}(\mathcal{I}_s, \mathcal{L}_s)$ and $\mathcal{P}(\mathcal{I}_s, \mathcal{L}_s)$, and we similarly use $\mathcal{Q}_{st}$ to denote $\mathcal{Q}(\mathcal{I}_s, \mathcal{L}_s, \mathcal{I}_t, \mathcal{L}_t)$. When referencing a sequence of states, we use the index in the sequence to increase legibility, e.g., $\mathcal{C}_i$ instead of $\mathcal{C}_{s_i}$.

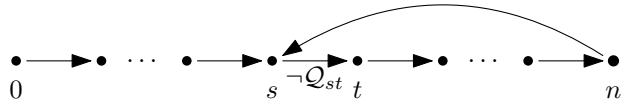

Fig. 1. Trace violating liveness property.

Next, we consider semantics for liveness properties in the form $\Diamond\Box\mathcal{Q}$ [45], such that on every infinite path, the liveness signal $\mathcal{Q}$ must eventually hold forever. A counterexample therefore describes $\mathcal{Q}$ evaluating to *false* infinitely often. Since we consider finite-state systems, such a counterexample implies the existence of a lasso-shaped trace with a stem from an initial state to a loop entry where $\neg\mathcal{Q}$ holds. In Fig. 1, the predicate $\neg\mathcal{Q}_{st}$ labels the edge $(s \to t)$ because it refers to two states. This syntactic choice does not affect the semantics. Because we essentially use Kripke-structure semantics for finite systems, any infinite path violating a liveness property will have an infinite number of pairs of consecutive states violating $\mathcal{Q}$. The set of second states of these pairs is finite and therefore eventually repeats; at that point, we can close the lasso. Thus the standard lasso argument also translates to our setting.

Definition 2. *If $\mathcal{M} = (\mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q})$ is a circuit, a* trace *is an infinite sequence of states $(s_0, s_1, \dots)$ where for all $i \in \mathbb{N}$:*

- *the first state s_0 is a reset state, i.e., $\mathcal{R}_0[\mathcal{L}]$ holds,*
- *transitions (s_i, s_{i+1}) follow the transition functions, i.e., $\mathcal{F}_{i,i+1}[\mathcal{L}]$ holds, and*
- *all states s_i satisfy the environment constraint, i.e., $\mathcal{C}_i$ holds.*

Definition 3 (Safety). *A circuit $\mathcal{M}$ satisfies its* safety *property if, for every trace according to Def. 2 and every $i \in \mathbb{N}$, the safety signal $\mathcal{P}$ holds in state s_i.*

Definition 4 (Liveness). *A circuit $\mathcal{M}$ satisfies its* liveness *property if, for every trace according to Def. 2, there exists an index $k \in \mathbb{N}$ such that for all $i \geq k$, the liveness signal $\mathcal{Q}$ holds on each consecutive state pair (s_i, s_{i+1}).*

4 Witness Circuits as Liveness Certificates

In this section, we present our certificate format. For safety properties, inductive invariants are a standard approach to certification. However, directly synthesizing an inductive invariant has proven to be difficult for certain model checking techniques [33]. The approach was successfully generalized to *witness circuits* [47], which share a common subset of inputs and latches with the model. This allows one to find a circuit that *simulates* the behavior of the model while itself being evidently safe by being *inductive*. We extend this idea by introducing the *ranked* property, which is similarly efficient to check and implies liveness.

The following definitions formalize these requirements. The conditions consider up to three states s, t, u in $\mathcal{M}$ and $\mathcal{W}$ that agree on the shared components, and can therefore be viewed as assignments to $\mathcal{I} \cup \mathcal{L} \cup \mathcal{I}' \cup \mathcal{L}'$. We again index Boolean formulas with the state to which they refer.

Definition 5 (Simulation). *Circuit $\mathcal{W} = \langle \mathcal{I}', \mathcal{L}', \mathcal{R}', \mathcal{F}', \mathcal{C}', \mathcal{P}', \mathcal{Q}' \rangle$* simulates *circuit $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$ if the following hold (with $\mathcal{K} = \mathcal{L} \cap \mathcal{L}'$):*

- *Reset:* $\mathcal{R}_s[\mathcal{K}] \wedge \mathcal{C}_s \rightarrow \mathcal{R}'_s[\mathcal{K}] \wedge \mathcal{C}'_s$
- *Transition:* $\mathcal{F}_{st}[\mathcal{K}] \wedge \mathcal{C}_s \wedge \mathcal{C}_t \wedge \mathcal{C}'_s \rightarrow \mathcal{F}'_{st}[\mathcal{K}] \wedge \mathcal{C}'_t$
- *Safety:* $\mathcal{C}_s \wedge \mathcal{C}'_s \wedge \mathcal{P}'_s \rightarrow \mathcal{P}_s$
- *Liveness:* $\wedge_{i \in \{s,t\}} (\mathcal{C}_i \wedge \mathcal{C}'_i \wedge \mathcal{P}'_i) \wedge \mathcal{F}'_{st}[\mathcal{L}'] \wedge \mathcal{Q}'_{st} \rightarrow \mathcal{Q}_{st}$

Definition 6 (Inductive). *Circuit $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$ is* inductive *if:*

- *Base:* $\mathcal{R}_s[\mathcal{L}] \wedge \mathcal{C}_s \rightarrow \mathcal{P}_s$
- *Induction:* $\mathcal{F}_{st}[\mathcal{L}] \wedge \mathcal{C}_s \wedge \mathcal{C}_t \wedge \mathcal{P}_s \rightarrow \mathcal{P}_t$

Definition 7 (Ranked). *Circuit $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$ is* ranked *if:*

- *Decrease:* $\wedge_{i \in \{s,t\}} (\mathcal{C}_i \wedge \mathcal{P}_i) \wedge \mathcal{F}_{st}[\mathcal{L}] \rightarrow \mathcal{Q}_{ts}$
- *Closure:* $\wedge_{i \in \{s,t,u\}} (\mathcal{C}_i \wedge \mathcal{P}_i) \wedge \mathcal{F}_{st}[\mathcal{L}] \wedge \mathcal{Q}_{su} \rightarrow \mathcal{Q}_{tu}$

Definition 8 (Witness circuit). *A circuit $\mathcal{W}$ is a* witness circuit *for $\mathcal{M}$ if $\mathcal{W}$ simulates $\mathcal{M}$ and $\mathcal{W}$ is both inductive and ranked.*

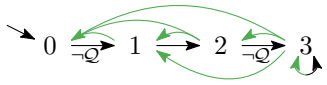

Fig. 2. Minimal witness circuit of a two-bit saturating counter. The black edges illustrate $\mathcal{F}$, with some not satisfying the liveness signal $\mathcal{Q}$: edges $(0 \to 1)$ and $(2 \to 3)$. If the witness does not add too much additional behavior, we can depict it in the same state space by visualizing $\mathcal{Q}'$ as the green edges. Here $\mathcal{P} = \mathcal{P}' = \top$, and are not shown; states violating $\mathcal{C}$ or $\mathcal{C}'$ would be omitted from the depiction. The witness is minimal in the sense that $\mathcal{Q}'$ contains the fewest edges possible, which is exactly the reverse transitive closure of $\mathcal{F}$. To see that the witness is *ranked*, we can for example inspect *Decrease* for transition $(1 \to 2)$ and see that indeed $\mathcal{Q}'_{2,1}$ holds. The *Liveness* condition is fulfilled as no edge labeled with $\neg\mathcal{Q}$ coincides with a $\mathcal{Q}'$ edge in the same direction.

To check the validity of a witness circuit, each of the eight conditions is checked by a SAT solver call. If all checks pass, the model satisfies both its safety and liveness properties. Def. 8 is an extension of the witness circuit format adopted by the Hardware Model Checking Competition [21] for safety properties, as setting $\mathcal{Q}$ and $\mathcal{Q}'$ to `true` trivially satisfies *Liveness*, *Decrease*, and *Closure*.

Before formally proving the soundness of this certificate format, we give an intuitive explanation of the conditions and what they accomplish. The *Reset* and *Transition* conditions together establish that every trace in the model has at least one corresponding trace in the witness. To establish a *simulation relation* between two circuits we also require that the safety and liveness properties of these traces correspond to each other. As without the *Safety* and *Liveness* conditions, unrelated circuits ($\mathcal{K} = \emptyset$) would simulate each other. Finding a simulating circuit is still trivial, as a circuit always simulates itself. To be a useful certificate, we require the safety and liveness of the witness to be *obvious*. This is enforced by checking that it is *inductive* and *ranked*.

Inductiveness is significantly stronger than safety, as it requires that any state satisfying $\mathcal{P}$ can only transition to other states that also satisfy $\mathcal{P}$. This condition is enforced even in the unreachable part of the state space.

The ranked condition fulfills a similar role for liveness. We chose the name to remind readers of *ranking functions*, often used to prove liveness. Indeed, if a ranking function is known, $\mathcal{Q}_{st}$ can easily be defined as "$rank_s \leq rank_t$". However, our format only uses propositional logic and is more flexible, thus sometimes admitting certificates where defining a ranking function may be challenging. We will see such an example in Sec. 5.3, where we discuss *liveness-to-safety*. $\mathcal{Q}$ can be conceptualized as an additional set of edges in the state space. Fig. 2 shows such a visualization for our running example of a saturating binary counter (which counts up to all ones and then remains stuck). With this interpretation, the ranked conditions ensure that $\mathcal{Q}$ contains an over-approximation of the (reversed) transitive closure of $\mathcal{F}$. In other words, if a path from s to t exists, $\mathcal{Q}_{ts}$ must hold. However, $\mathcal{Q}$ itself does not need to be transitive. In algebraic terms, $\mathcal{Q}$ is merely a *right ideal* with respect to $\mathcal{F}$.

We state our main theorem and proceed to prove it in three parts.

Theorem 1 (Soundness). *If $\mathcal{M}$ has a witness circuit, then $\mathcal{M}$ is safe and live.*

Proof. Assume $\mathcal{W} = \langle \mathcal{I}', \mathcal{L}', \mathcal{R}', \mathcal{F}', \mathcal{C}', \mathcal{P}', \mathcal{Q}' \rangle$ is a witness circuit for $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$. By Lemmas 1 and 2 below, $\mathcal{W}$ is safe and live. It remains to show that safety and liveness transfer through the simulation relation. Let $\mathcal{S} = (s_0, s_1, \dots)$ be a trace in $\mathcal{M}$. We extend each s_i to an assignment s_i^* to $\mathcal{I} \cup \mathcal{L} \cup \mathcal{I}' \cup \mathcal{L}'$ such that $\mathcal{S}^* = (s_0^*, s_1^*, \dots)$ is also a trace for $\mathcal{W}$ as follows. First, we assign an arbitrary value (e.g. 0) to all inputs in $\mathcal{I}' \setminus \mathcal{I}$ in every state s_i^*. By the *Reset* condition, $\mathcal{R}_0'[\mathcal{K}] \wedge \mathcal{C}_0'$ already holds. To ensure that the remaining latches can be assigned so that the reset predicate is satisfied, i.e., $\mathcal{R}_0'[\mathcal{L}']$ holds, we note that the strict partial order on the reset dependencies of a circuit implies stratification and refer to [20, Lemma 2]. Now, we inductively construct the states s_i^* for $i \geq 1$: given a state s_{i-1}^* and fixed inputs, the transition functions $\mathcal{F}'$ imply a unique value for all latches in s_i^*. Thanks to the *Transition* condition, this unique value agrees with s_i on the latches in $\mathcal{K}$ and s_i^* satisfies the environment constraint $\mathcal{C}_i'$.

If $\mathcal{W}$ is safe, $\mathcal{P}_i'$ holds for $i \geq 0$ in any such trace $\mathcal{S}^*$. Then $\mathcal{P}_i$ holds for $i \geq 0$ by the *Safety* condition, and thus $\mathcal{M}$ is safe. If $\mathcal{W}$ is additionally assumed to be live, there exists $k \in \mathbb{N}$ such that $\mathcal{Q}_{i,i+1}'$ holds for all $i \geq k$. Since $\mathcal{P}_i'$ and $\mathcal{P}_{i+1}'$ hold by safety of $\mathcal{W}$, $\mathcal{Q}_{i,i+1}$ holds for all $i \geq k$ by the *Liveness* condition, thus $\mathcal{M}$ is live. □

Lemma 1. *If a circuit is inductive, then it is safe.*

Proof. Let $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$ be an inductive circuit. For any trace $\mathcal{S} = (s_0, s_1, \dots)$, the *Base* condition gives $\mathcal{P}_0$. For $i \geq 1$, $\mathcal{P}_i$ follows from $\mathcal{P}_{i-1}$ by the *Induction* condition. Thus $\mathcal{P}_i$ holds for all $i \in \mathbb{N}$ and $\mathcal{M}$ is safe. □

Lemma 2. *If a safe circuit is ranked, then it is live.*

Proof. Let $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \mathcal{P}, \mathcal{Q} \rangle$ be a circuit that is safe and ranked; we show that it is live. We first show that any transition (u, v) that appears more than once in a trace of $\mathcal{M}$ satisfies $\mathcal{Q}_{uv}$. Fix a trace and let $i < j$ be such that $s_i = s_j = u$ and $s_{i+1} = s_{j+1} = v$. Since $\mathcal{M}$ is safe, $\mathcal{P}$ holds everywhere in the trace. By the *Decrease* condition, $\mathcal{Q}_{i+2,i+1}$ holds, and by repeated application of the *Closure* condition also $\mathcal{Q}_{i+3,i+1}, \mathcal{Q}_{i+4,i+1}, \dots$, until $\mathcal{Q}_{j,i+1}$ holds, which is just $\mathcal{Q}_{uv}$. Thus no transition violating $\mathcal{Q}$ can appear twice in a trace. Since $\mathcal{M}$ has only finitely many states, every trace has only finitely many liveness signal violations, and $\mathcal{M}$ is live. □

Before describing how to generate certificates, we discuss a limitation of our approach. Both in practice, where properties are encoded into SAT, and in theory, where arguments are made over edges in the state space, we require properties to be uniformly embedded into the model, as is the case in AIGER. Consequently, the format cannot reason directly about general LTL formulas.

However, the certificate checker can handle liveness properties with *multiple* signals, as obtained from direct encodings of generalized Büchi automata. We omit this generalization here, as it would significantly complicate the presentation of the model-checking techniques in the following section.

5 Certificate Generation

To make a convincing argument that the certificate format introduced in Sec. 4 is useful, we present four commonly used [6] symbolic liveness model checking techniques, describe how to produce certificates for them, and give a proof sketch on why the constructions will always produce valid certificates. These constructions can thus be straightforwardly implemented by certifying model checkers. Due to space restrictions, we only include sketches of their correctness proofs.

In practice, the important point is that the verification result is correct if certificate checking passes. This guarantee comes from Theorem 1 and also holds if the certificate construction (or its implementation in the model checker) is flawed. Moreover, all techniques described below and implemented in our model checker have undergone extensive fuzz-testing using the randomized AIGER test harness described in [19]. This validates not only the presented constructions, but also the implementation used for the experimental evaluation in Sec. 6.

The model checking techniques we consider are k-liveness, stabilizer extraction [13], liveness-to-safety (L2S) [4], and rLive [46]. They have all been developed for liveness signals that do not use inputs from the next state, i.e., $\mathcal{Q}(\mathcal{I}_s, \mathcal{L}_s, \mathcal{I}_t, \mathcal{L}_t)$ does not depend on $\mathcal{I}_t$. The use of $\mathcal{L}_t$, on the other hand, does not depart from standard practice, since these values can always be computed from $\mathcal{I}_s$ and $\mathcal{L}_s$ in models with transition functions and Kripke-structure semantics, as in the AIGER format used in the Hardware Model Checking Competition [6]. While we believe they can be extended to models that make use of all the features in Def. 1, we focus on the commonly used versions of the techniques. The motivation to include the next state inputs $\mathcal{I}_t$ in the definition of $\mathcal{Q}$ is to give model checkers more flexibility when producing witness circuits.

Furthermore, we assume that the model has only a liveness property, i.e., $\mathcal{P} = \top$. Nevertheless, the liveness techniques that we describe all rely on reductions to safety checking in some form. Our witness constructions are agnostic to the particular safety checking technique employed. For ease of presentation, we may make simplifying assumptions about the produced witness circuits in some cases. For a detailed reference how to handle more complex witness circuits, and how to integrate witnesses for additional safety properties, we refer the reader to [48]. Lastly, all of the techniques can also produce counterexamples when the liveness property is violated, which we will not consider further in this paper.

5.1 *k*-liveness

As one of the most commonly used liveness checking techniques, k-liveness [13] relies on the fact that in a finite system, the liveness signal $\mathcal{Q}$ can only be violated finitely many times if the liveness property $\Diamond\Box\mathcal{Q}$ holds. The approach proceeds as follows. First, one checks whether the liveness signal is already a safety property, i.e., whether $\Box\mathcal{Q}$ holds. If not, an additional *live latch* is introduced: a new latch ℓ^1 that is initialized to true and transitions to false whenever $\mathcal{Q}$ is violated.

The approach then proceeds by checking the safety property $\Box(\ell^1 \vee \mathcal{Q})$. If a counterexample still exists, the construction is iterated. At iteration k, a new

live latch ℓ^k is added, whose transition function is defined as

$$f_{\ell^k} = \ell^k \wedge (\ell^{k-1} \vee \mathcal{Q}).$$

Intuitively, ℓ^k allows up to k violations of $\mathcal{Q}$ before permanently disabling itself. The corresponding safety property

$$\mathcal{P}^k = \ell^k \vee \mathcal{Q}$$

is then checked. Since the system is finite, the liveness property holds if and only if the safety check succeeds for some finite k.

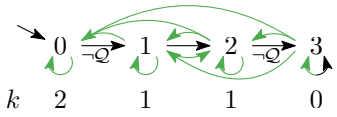

Fig. 3. k-liveness witness. States have an additional k, decreasing on transitions violating liveness signal $\mathcal{Q}$. Every state has $\mathcal{Q}'$ edges to states with a smaller or equal k.

Certificate generation. Once the safety check succeeds for some value of k, the backend safety engine returns a witness circuit $\mathcal{W}^k = \langle \mathcal{I}', \mathcal{L}', \mathcal{R}', \mathcal{F}', \mathcal{C}', \mathcal{P}', \top \rangle$ for $\mathcal{M}^k = \langle \mathcal{I}, \mathcal{L}^k, \mathcal{R}^k, \mathcal{F}^k, \mathcal{C}, \mathcal{P}^k, \top \rangle$, where $\mathcal{P}^k$ is as described above and $\mathcal{L}^k, \mathcal{R}^k$, and $\mathcal{F}^k$ only differ from the original model $\mathcal{M}$ in the added live latches. Since no liveness property is provided at this stage (i.e., $\mathcal{Q} = \top$), the liveness component $\mathcal{Q}'$ of the witness can be assumed to be trivially true. To obtain a witness $\mathcal{W}$ for the original model $\mathcal{M}$, we extend $\mathcal{W}^k$ by defining $\mathcal{Q}'$ as

$$\mathcal{Q}'(\mathcal{I}_s, \mathcal{L}_s, \mathcal{I}_t, \mathcal{L}_t) \;=\; \bigwedge_{i=1}^{k} (\ell_s^i \rightarrow \ell_t^i),$$

that is, $\mathcal{Q}'$ checks that the live latches set in t are a superset of those set in s.

In the following, we present a proof sketch for the correctness of the certificate generation described above. We show that if k-liveness model checking succeeds for $\mathcal{M}$, then the generated witness circuit $\mathcal{W}$ simulates $\mathcal{M}$ and satisfies both the inductive and ranked requirements.

Proof sketch (correctness). Since the reset and transition functions of the original latches remain unchanged in $\mathcal{M}^k$, the *Reset* and *Transition* checks pass for $\mathcal{M}$ and $\mathcal{W}$. *Safety* holds trivially as $\mathcal{P} = \top$ in $\mathcal{M}$, and the inductive conditions follow from $\mathcal{W}^k$ being inductive. *Decrease* holds because by the definition of f_ℓ in $\mathcal{F}^k$, the latches used in $\mathcal{Q}'$ only transition from true to false. To see that *Closure* holds, let s, t, and u be states satisfying the left-hand side of the implication. Since $\mathcal{F}'_{st}[\mathcal{L}']$ is true, *Decrease* implies $\mathcal{Q}'_{ts}$. Combining this with $\mathcal{Q}'_{su}$ and using the fact that $\mathcal{Q}'$ expresses a subset relation between the true live latches yields

$\mathcal{Q}'_{tu}$. Lastly, for *Liveness*, assume towards contradiction that some state s and t form a counterexample. As $\mathcal{P}'$ is the same in $\mathcal{W}$ and $\mathcal{W}^k$, the *Safety* condition for $\mathcal{W}^k$ implies $\mathcal{P}^k_s$. Since $\neg\mathcal{Q}_{st}$, ℓ^k_s must hold and some live latch must flip, i.e., for some $j \in [1,k]$ we have $\ell^j_s \wedge \neg\ell^j_t$. By definition, this gives $\neg\mathcal{Q}'_{st}$, contradicting the assumption. □

Our certificate generation approach also extends directly to variants of k-liveness where instead of encoding k in unary, a binary counter is used [22, 29].

5.2 Stabilizer Extraction

Automatic extraction of stabilizing constraints [13] is a preprocessing technique in which constraints are extracted from the model and then added as assumptions, thereby simplifying the backend model checking task. The extracted stabilizing constraints are in the form $\Diamond\Box\mathcal{S}$. This technique is essential for the performance of k-liveness. The extraction procedure is summarized in Algorithm 1.

Algorithm 1 Extract stabilizing constraints

Input: $\mathcal{M} = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \top, \mathcal{Q} \rangle$
1: $E_q = \top$, Candidates $= \mathcal{L} \cup \{\bar{\ell} \mid \ell \in \mathcal{L}\}$
2: **repeat**
3: choose $l \in$ Candidates ▷ note that l is a literal
4: **if** $\mathcal{F}[\mathcal{L}]_{st} \wedge E_q \wedge l_s \rightarrow l_t$ holds **then**
5: $E_q = E_q \wedge (l_s \leftrightarrow l_t)$ ▷ add both directions!
6: Candidates.remove(l)
7: **until** E_q reaches a fixpoint
8: **return** $\mathcal{M}^E = \langle \mathcal{I}, \mathcal{L}, \mathcal{R}, \mathcal{F}, \mathcal{C}, \top, E_q \rightarrow \mathcal{Q} \rangle$

The intuition for this technique is that for liveness model checking, any finite prefix of an infinite trace can be disregarded. Given $\mathcal{F}_{st} \rightarrow (\ell_s \rightarrow \ell_t)$, in any execution trace of the model the latch ℓ either eventually becomes true and remains true (allowing the finite prefix before stabilization to be ignored) or stays false forever. In both cases, the equivalence $\ell_s \leftrightarrow \ell_t$ holds on the infinite suffix of the trace. This allows us to add $\Diamond\Box(\ell_s \leftrightarrow \ell_t)$ as an assumption to the model. By iterating over all latch literals (and their negations) and repeating until fixpoint, we can identify all such stabilizing signals. This, in turn, allows us to weaken the liveness signal to $E_q \rightarrow \mathcal{Q}$, which is violated less frequently and can therefore potentially reduce the value of k; for a more detailed proof, we refer to [13]. Adding such stabilizing constraints to the liveness signal can substantially improve performance, as illustrated by the running example in Fig. 4.

Certificate generation. Let $\mathcal{M}^E$ be the output of Algorithm 1 for model $\mathcal{M}$, $B = (l^0, \ldots l^n)$ be the stabilizing signals for which the equality is added in line 5, and $\mathcal{W}^E$ be the witness circuit produced by k-liveness for $\mathcal{M}^E$. To get the witness $\mathcal{W}$ for the original model $\mathcal{M}$, we encode a binary comparator over B and,

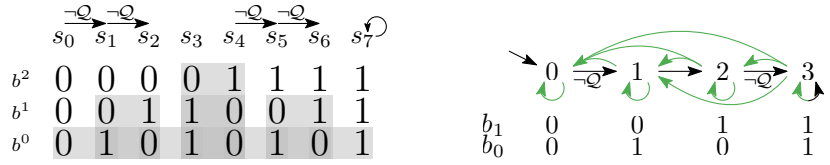

Fig. 4. Exponential reduction in liveness violations. The figure *on the left* considers the saturating counter from the running example with three bits b_2, b_1, b_0. The transition predicate $\mathcal{F}[\mathcal{L}]_{st}$ implies $b_s^2 \to b_t^2$. Adding the equality $b_s^2 \leftrightarrow b_t^2$ removes the transition from s_3 to s_4 from consideration. Now, $b_s^1 \to b_t^1$ is also valid, and adding the equality removes two additional transitions. Finally, the query $b_s^0 \to b_t^0$ is valid, and all transitions except s_7 to s_7 are eliminated. Extracting the stabilizing constraints successively reduces the number of transitions violating $E_q \to \mathcal{Q}$ from 4, to 2, to 0.
On the right, the witness circuit for two bits is depicted. Here every state has a $\mathcal{Q}'$ edge to any state with a lower or equal value, as the bits of the binary counter are identified as stabilizing signals in decreasing order of significance.

in case of equality, fall back to $\mathcal{Q}'^E$, i.e., to define $\mathcal{Q}'(\mathcal{I}_s, \mathcal{L}_s, \mathcal{I}_t, \mathcal{L}_t)$ we encode $eq_{st}^0 = (l_s^0 \leftrightarrow l_t^0)$ and $eq_{st}^i = eq_{st}^{i-1} \wedge (l_s^i \leftrightarrow l_t^i)$; and further $dec_{st}^0 = (l_s^0 \wedge \neg l_t^0)$ and $dec_{st}^i = dec_{st}^{i-1} \vee (eq_{st}^{i-1} \wedge l_s^i \wedge \neg l_t^i)$; and finally $\mathcal{Q}'(\mathcal{I}_s, \mathcal{L}_s, \mathcal{I}_t, \mathcal{L}_t) = dec_{st}^n \vee (eq_{st}^n \wedge \mathcal{Q}_{st}'^E)$. Latches in B that are not present in $\mathcal{W}^E$ are added back together with their transition and reset functions from $\mathcal{M}$.

To show that the certificate generation is correct, we assume that the k-liveness backend successfully produces a witness circuit, and show that the subsequently constructed witness circuit $\mathcal{W}$ passes certificate checking w.r.t. $\mathcal{M}$.

Proof sketch (correctness). Assuming that the k-liveness backend produced a witness circuit for $\mathcal{M}^E$, only the ranked conditions and the *Liveness* condition are left to show. First, we show that $\mathcal{F}'[\mathcal{L}']_{st} \to (eq_{ts}^n \vee dec_{ts}^n)$ holds: If $\neg eq_{ts}^n$ holds, then there exists $j \in [0, n]$ such that $l_s^i \leftrightarrow l_t^i$ for all $i \in [0, j-1]$, and either $\neg l_s^j \wedge l_t^j$ or $l_s^j \wedge \neg l_t^j$ holds. In the first case dec_{ts}^n holds, and the second case leads to a contradiction: Consider the iteration of Algorithm 1 where the l^j equality was added to E_q. At this point, $E_q = \bigwedge_{i<j}(l_s^i \leftrightarrow l_t^i)$, which is satisfied. Furthermore, $\mathcal{F}[\mathcal{L}]_{st}$ holds by the *Transition* condition of $\mathcal{W}^E$ or because $f_\ell \in \mathcal{F}$ has been reintroduced if $\ell \notin \mathcal{L}'$. Then the check in line 4 would have failed for $l = l^j$, and l^j would not have been identified as a stabilizing signal. We also use obvious properties of dec^n and eq^n such as transitivity and (anti-)symmetry.

With this, we can show the *Decrease* condition, since either dec_{ts}^n holds, or the condition is reduced to *Decrease* of $\mathcal{W}^E$. The precondition of *Closure* implies $eq_{ts}^n \vee dec_{ts}^n$ by $\mathcal{F}'[\mathcal{L}]$ and $eq_{su}^n \vee dec_{su}^n$ by $\mathcal{Q}'_{su}$. If eq_{st}^n, eq_{tu}^n and thus eq_{su}^n hold, the condition reduces to *Closure* of $\mathcal{W}^E$, otherwise dec_{tu} is implied by transitivity. *Liveness* preconditions give $\mathcal{Q}'_{st}$, and $\mathcal{F}'_{st}$ implies $\mathcal{Q}'_{ts}$ by *Decrease*. Only eq_{st}^n can satisfy both constraints, and the condition reduces to *Liveness* of $\mathcal{W}^E$. □

Note that in the above proof sketch, we did not use any property of k-liveness other than that it produces a witness circuit for $\mathcal{M}^E$. Thus, the stabilizer extraction technique can be combined with any certifying liveness checking technique.

5.3 Liveness-to-Safety

The liveness-to-safety [4, 40] approach reduces the liveness checking problem of $\mathcal{M}$ to a single safety check of $\mathcal{M}^{\text{L2S}}$. It works by creating a copy ($\mathcal{L}^c$) of the latches, which are initialized the same as their original counterpart and remain unchanged except when a new input i^c is true. Whenever i^c is true, the value of the original latches is copied into $\mathcal{L}^c$. The new safety property is $\mathcal{P}^{\text{L2S}} = (\mathcal{F}_{st}[\mathcal{L}] \rightarrow \mathcal{Q}_{st})[\mathcal{L}_t \mapsto \mathcal{L}^c]$, where we use $\phi[\omega]$ to denote the formula obtained from ϕ by replacing each literal in the domain of ω by its image under ω. $\mathcal{P}^{\text{L2S}}$ expresses that if we are in a state s from which we would transition to the state stored in the copy latches $\mathcal{L}^c$, then the liveness property $\mathcal{Q}$ must hold between s and that stored state. The intuition behind this transformation is that if a trace with infinitely many liveness violations exists in the original circuit, one violating edge has to be visited infinitely often (as the number of states is finite). Therefore, a trace exists in $\mathcal{M}^{\text{L2S}}$ where $\neg\mathcal{Q}_{st}$ is encountered; i^c turns on in the next step, $\mathcal{L}^c$ stores t on transition; and the trace loops through the state space until $\neg\mathcal{Q}_{st}$ is encountered again, resulting in a violation of $\mathcal{P}^{\text{L2S}}$.

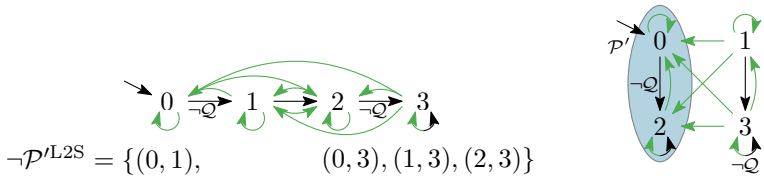

Fig. 5. Liveness-to-safety witness circuits. The L2S safety property fails if state s is encountered with state t stored, where $\neg\mathcal{Q}_{st}$ holds. With the notation $(\mathcal{L}, \mathcal{L}^c)$, the two bad states in $\mathcal{M}^{\text{L2S}}$ are: $(0, 1)$ and $(2, 3)$. The invariant $\mathcal{P}'^{\text{L2S}}$ must at least exclude those states and all recursive predecessors to be inductive. Reinterpreting those pairs as edges in $\mathcal{M}$, $\mathcal{Q}'$ is defined exactly as what is left in $\mathcal{P}'^{\text{L2S}}$. The figure lists $\neg\mathcal{P}'^{\text{L2S}}$, i.e., exactly the edges excluded from $\mathcal{Q}'$.
The second example on the right demonstrates why $\mathcal{P}'$ is necessary in addition to $\mathcal{Q}'$. Without $\mathcal{P}'$, the witness would not pass the *Liveness* condition.

Certificate generation. After transformation, the model given to the backend safety engine is $\mathcal{M}^{\text{L2S}} = \langle \mathcal{I} \cup \{i^c\}, \mathcal{L} \cup \mathcal{L}^c, \mathcal{R} \cup \mathcal{R}^c, \mathcal{F} \cup \mathcal{F}^c, \mathcal{C}, \mathcal{P}^{\text{L2S}}, \top \rangle$, where $\mathcal{P}^{\text{L2S}}$ is as described above, $\mathcal{F}^c$ implements the copy behavior for the new latches, and $\mathcal{R}^c$ initializes the added latches to the value of their original counterpart.

For simplicity of presentation, we assume that if $\mathcal{M}^{\text{L2S}}$ is safe, witness $\mathcal{W}^{\text{L2S}} = \langle \mathcal{I}', \mathcal{L}', \mathcal{R}', \mathcal{F}', \mathcal{C}', \mathcal{P}'^{\text{L2S}}, \top \rangle$ is produced by IC3 [9] or similar techniques that strengthen the invariant without changing the set of latches. All primed symbols used below, except the newly defined $\mathcal{P}'$ and $\mathcal{Q}'$, refer to components inherited from $\mathcal{W}^{\text{L2S}}$. Before describing the witness construction, we provide an intuition for the invariant $\mathcal{P}'^{\text{L2S}}$. First note that an $\mathcal{M}^{\text{L2S}}$-state consists of inputs and two $\mathcal{M}$-states. Slightly abusing notation, below we will talk about tuples (s, t)

of two $\mathcal{M}$-states being an $\mathcal{M}^{\text{L2S}}$ state, where the first component s corresponds to the current state stored in $\mathcal{L}$ and the second component t corresponds to the state stored in $\mathcal{L}^c$. We will do the same for $\mathcal{W}^{\text{L2S}}$-states.

Now, whenever there is a transition from s to t in the original model $\mathcal{M}$, in $\mathcal{M}^{\text{L2S}}$, the same transition could happen with s being stored immediately, i.e., there is a transition from $(s, *)$ to (t, s). The *Induction* check then guarantees that (t, s) must satisfy $\mathcal{P}'^{\text{L2S}}$. Similarly, if there are transitions from u to s and from s to t in $\mathcal{M}$, there are transitions from $(u, *)$ to (s, u) to (t, u) in $\mathcal{M}^{\text{L2S}}$ and hence also (t, u) must satisfy $\mathcal{P}'^{\text{L2S}}$. Continuing this line of reasoning, we see that (t, x) must satisfy $\mathcal{P}'^{\text{L2S}}$ for any x that can reach t in $\mathcal{M}$. In other words, $\mathcal{P}'$ is true for a set of $\mathcal{M}^{\text{L2S}}$-states (t, x) that includes at least all tuples (t, x) such that x can reach t (in $\mathcal{M}$). The *Safety* check will then ensure that if there is a path from x to t and a transition from t to x, then $\mathcal{Q}_{tx}$ must hold, i.e., that if there is a loop, then the involved edges are not liveness violations.

The witness construction builds heavily on the above intuition: since $\mathcal{P}'^{\text{L2S}}$ is true for all (t, x) where x can reach t and the crucial property of $\mathcal{Q}'$ is that it should contain the transitive closure of the transition relation, we can simply take $\mathcal{Q}'$ to be equal to $\mathcal{P}'^{\text{L2S}}$, modulo some renaming (since $\mathcal{P}'^{\text{L2S}}$ is defined over $\mathcal{L}$ and $\mathcal{L}^c$ whereas $\mathcal{Q}'$ is defined over two copies of the latches $\mathcal{L}'_s$ and $\mathcal{L}'_t$):

$$\mathcal{Q}' = \mathcal{P}'^{\text{L2S}}[\mathcal{L} \mapsto \mathcal{L}'_s,\ \mathcal{L}^c \mapsto \mathcal{L}'_t].$$

To obtain $\mathcal{P}'$, we instead use $\mathcal{L}'$ and $\mathcal{F}'$ to replace $\mathcal{L}^c$ and $\mathcal{L}$, which corresponds to storing the current state at every step:

$$\mathcal{P}' = \mathcal{P}'^{\text{L2S}}[\mathcal{L} \mapsto \mathcal{F}',\ \mathcal{L}^c \mapsto \mathcal{L}'].$$

With that, $\mathcal{W} = \langle \mathcal{I}', \mathcal{L}' \setminus \mathcal{L}^c, \mathcal{R}' \setminus \mathcal{R}^c, \mathcal{F}' \setminus \mathcal{F}^c, \mathcal{C}', \mathcal{P}', \mathcal{Q}' \rangle$ is a witness for the original model $\mathcal{M}$. In practice, the substitution operations above are easy to implement and only require a linear scan over the AND-gates.

In the following, we show that if the L2S model checking passes for $\mathcal{M}$, then as described above $\mathcal{W}$ satisfies the witness conditions.

Proof sketch (correctness). We assume the liveness property holds in $\mathcal{M}$. Since only inputs and latches added by the L2S transformation are removed, and resets and transitions remain unchanged, the *Reset* and *Transition* conditions hold immediately. Any three consecutive states s, t, u in $\mathcal{M}$ induce $\mathcal{M}^{\text{L2S}}$ states (t, s) and (u, t), where the first component is the assignment to $\mathcal{L}$ and the second component is the assignment to $\mathcal{L}^c$. They satisfy the transition function with i^c set to $\top$. With this, $\mathcal{P}'^{\text{L2S}}$ implies $\mathcal{P}'$ and the *Base* and *Induction* conditions follow. $\mathcal{P}'$ restricts the considered states exactly to those where the *Decrease* condition holds, rendering the check trivial. For *Closure*, a similar argument applies: consider three states s, t, u where s and t are consecutive in $\mathcal{M}$; then (s, u) and (t, u) are consecutive in $\mathcal{M}^{\text{L2S}}$ for $i^c = \bot$. With this and $\mathcal{P}'^{\text{L2S}}$ being inductive, $\mathcal{Q}'_{tu}$ follows. Lastly, for *Liveness*, we expand the definition of $\mathcal{Q}'_{st}$ twice to get $\mathcal{F}_{st}[\mathcal{L}] \rightarrow \mathcal{Q}_{st}$. By the *Transition* condition and the fact that no latches have been removed, $\mathcal{F}'_{st}[\mathcal{L}']$ implies $\mathcal{F}_{st}[\mathcal{L}]$ and thus $\mathcal{Q}_{st}$. □

For more complex witness circuits which add and remove latches and modify their behavior, the construction needs to extend the above substitutions to all functions, including reset, transition, and constraint. In practical circuit representations, this is automatically done since usually only one and-inverter graph represents all functions. Furthermore, the added latches may be removed only if no remaining function refers to them after these substitutions have been applied.

5.4 rLive

The rLive algorithm [46] is a recent technique that has been shown to be effective in practice. So far it has been exclusively implemented in the model checker nuXmv [12], which won the liveness track of the last HWMCC [6].

It constructs a depth-first search tree of states that can violate the liveness signal. The edges between them represent finite paths in the model. To expand a node s, the algorithm performs a safety check with $\mathcal{Q}$ as the safety property, starting from the symbolic initial state t satisfying $\mathcal{F}_{st} \wedge \neg\mathcal{Q}_{st}$. If the safety-checking problem is unsafe (`SAT`), a state u is returned. If u matches any state already in the tree, an infinite counterexample for the liveness property is identified, otherwise u is added to the search tree and expanded next. If the safety-checking problem is `UNSAT`, the safety engine generates a witness circuit, which identifies a set of states that cannot violate the liveness signal further. This set of states (called a *shoal*) is removed from consideration for any future query by adding its negation to the model's constraint. This automatically closes the branch and the search backtracks. The algorithm terminates once it establishes that the set of initial states is contained within a shoal, thereby proving that no execution of the original model can violate the liveness property.

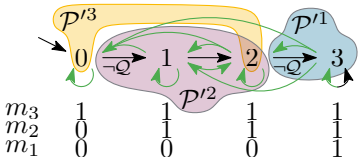

Fig. 6. rLive witness. The shoals $\mathcal{P}'^1, \mathcal{P}'^2, \mathcal{P}'^3$ are depicted as sets of states. At the time when $\mathcal{P}'^2$ is found, state 3 is not considered as part of the state space. Similarly, states 1 and 2 are not considered when constructing $\mathcal{P}'^3$. The monotone shoal indicators m^1, m^2, m^3 identify the lowest indexed shoal that a state is contained in, and $\mathcal{Q}'_{st}$ holds if that index is greater or equal in t compared to s.

Certificate generation. We consider the sequence of witness circuits produced by the safety engine. The witnesses are combined by union over the parts that are shared with the model, tagged union (i.e., renaming of inputs and latches) for the parts unique to the witnesses, and conjunction of the constraints, resulting in a

single circuit with $\mathcal{I}', \mathcal{L}', \mathcal{R}', \mathcal{F}', \mathcal{C}'$. Let $\mathcal{S} = (\mathcal{P}'_1, \ldots, \mathcal{P}'_n)$ be the shoals described by the witnesses in the order they are found.

Since all reachable states are eventually contained in at least one shoal, the disjunction of all shoals covers all reachable states, and we use $\mathcal{P}' = \bigvee_{i=1}^{n} \mathcal{P}'_i$ in the final witness $\mathcal{W}$. To define $\mathcal{Q}'$, we first encode monotone indicators that capture whether a state belongs to shoal i or to any shoal with a lower index: $m^0 = \bot$ and $m^i = m^{i-1} \vee \mathcal{P}'^i$. Finally, $\mathcal{W} = \langle \mathcal{I}', \mathcal{L}', \mathcal{R}', \mathcal{F}', \mathcal{C}', \mathcal{P}', \mathcal{Q}' \rangle$, with

$$\mathcal{Q}'_{st} = \bigwedge_{i=1}^{n} (m^i_t \to m^i_s).$$

Proof sketch (correctness). We assume that rLive model checking passes. Again, the reset and transition functions of the original latches remain unchanged, and therefore the *Reset* and *Transition* conditions hold. The algorithm terminates only once the reset states are contained in a shoal, implying the *Base* condition. For *Induction*, we rely on the inductiveness of each individual shoal; however, they are only inductive with respect to the state space where all previously found shoals are already removed by the constraint. This means a transition can leave a shoal if it enters a shoal with a smaller index, and may also be in multiple shoals at once. Either way, the disjunction $\mathcal{P}'$ is still inductive. By the same argument, *Decrease* holds because a transition from s to t either stays in the same shoal or enters a shoal with a smaller index, in either case satisfying $\mathcal{Q}'_{ts}$. Applied to the preconditions of *Closure*, this gives us $\mathcal{Q}'_{ts}$, which together with $\mathcal{Q}'_{su}$ and the obvious transitivity based on the definition of $\mathcal{Q}'$ gives us $\mathcal{Q}'_{tu}$. Lastly, assuming a violation of the *Liveness* condition, consider the shoal in which state s is contained. Given $\mathcal{Q}'_{st}$, state t is either in the same shoal S or in a shoal with a *higher* index. In the first case S contains $\neg\mathcal{Q}_{st}$ and could not have been produced by the algorithm, and in the latter case S is not inductive. □

6 Experiments

This section presents an experimental evaluation of our certification method for the four model checking algorithms: k-liveness, L2S, rLive, and stabilizer extraction. We implemented all of these techniques in our prototype to support liveness checking and certificate generation for each algorithm.

Our certificate checker generates the SAT formulas described in Def. 8 using the model and witness circuit as input. The implementation can handle multiple safety and liveness properties; moreover, for the latter it also supports multiple liveness signals per property. The generated SAT formulas are checked using Kissat [5]. The combination of our model checker and the certificate checker was subjected to around 16 hours of fuzz-testing [19] on 8 cores at 5.20 GHz.

Experimental Setup The final experiments presented in this paper were run on an HPC cluster using Intel Xeon Platinum 8360Y CPUs at 2.4 GHz. Each model checking instance was allocated 14 GB of memory and one hour of wall time.

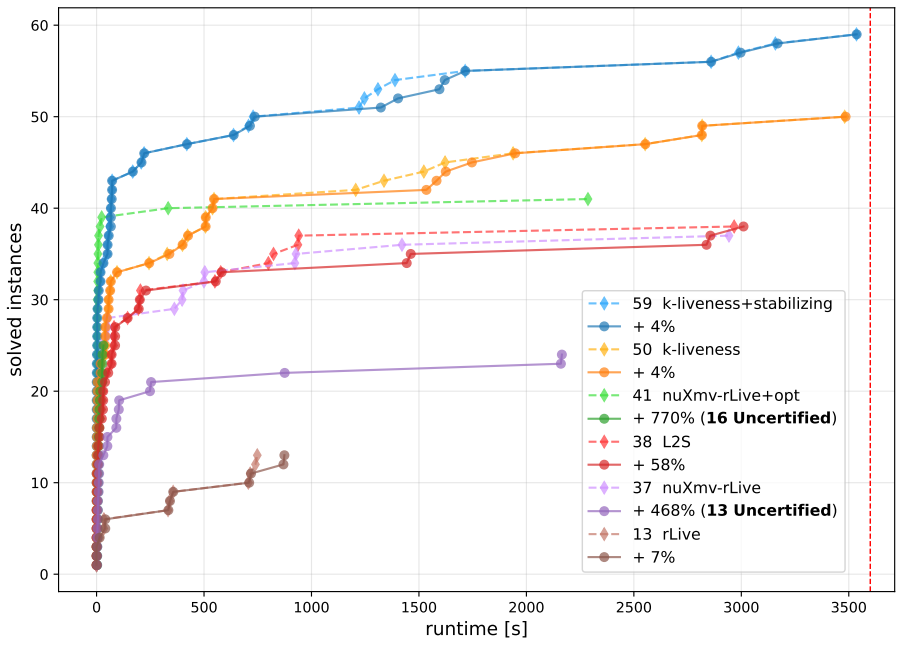

Fig. 7. Performance comparison of our method against the Sindoni et al. approach [41] implemented on top of nuXmv. Solid lines represent the total runtime for model checking *and* certificate checking, whereas dashed lines are model checking only. For each configuration, we display the number of solved instances, and the certification overhead (%). *Our method certified all instances solved by the model checker, with very low overhead.* In contrast, the approach of Sindoni et al. was unable to certify 16 instances with rLive optimizations and 13 instances without optimizations (most of them due to running out of memory), with significantly higher certification overhead of 309% and 469%, respectively (overhead calculated only on the certified instances). Among all certifying configurations, k-liveness with stabilizer extraction achieved the best performance, with only 4% certification overhead.

Benchmarks We use the liveness benchmarks from HWMCC 2025 [6], of which 73 were found to be `UNSAT`. The benchmarks feature explicit fairness constraints [7] and multiple liveness signals per property. While our certificate checker can reason about such benchmarks, the evaluated model checkers cannot. We therefore normalize the benchmarks beforehand using the unverified tool *aigunfair* [8]. This reduces multiple signals to one by adding latches that track violations; once all are set, the liveness signal is disabled for one step and the latches reset. We also include synthetic benchmarks encoding an n-bit saturating counter.

Table 1. Scaled version of an n-bit saturating counter. We report the value of n together with model checking and certificate checking times (seconds). The liveness signal is chosen as the middle bit of the counter. For the rLive implementation in nuXmv, we report both the base version of the algorithm (similar to what is implemented in our model checker) and the default variant with additional optimizations. Our certificate checker was able to validate all certificates generated. The certificate checker of [41] was unable to validate certificates with $n \geq 9$ due to memory exhaustion.

	Our Method								nuXmv			
	k-liveness		L2S		rLive		stabilizing+ k-liveness		rLive		optimizations+ rLive	
n	t_{mc}	t_{cert}	t_{mc}	t_{cert}	t_{mc}	t_{cert}	t_{mc}	t_{cert}	t_{mc}	t_{cert}	t_{mc}	t_{cert}
5	0.55	0.06	0.07	0.06	0.04	0.07	0.03	0.06	0.06	7.60	0.05	7.56
6	11.51	0.07	0.27	0.07	0.08	0.08	0.03	0.06	0.08	12.78	0.06	17.69
7	213.13	0.08	2.22	0.07	0.25	0.08	0.03	0.06	0.19	57.57	0.10	54.93
8	to	-	21.55	0.08	0.97	0.11	0.03	0.06	0.60	513.16	0.16	470.55
9	to	-	360.43	0.16	4.64	0.21	0.03	0.06	1.95	mo	0.46	mo
10	to	-	mo	-	26.29	0.45	0.03	0.06	7.63	mo	1.08	mo
11	to	-	mo	-	140.75	7.05	0.03	0.06	31.66	mo	3.19	mo
12	to	-	mo	-	922.88	23.70	0.03	0.06	147.51	mo	9.00	mo
13	to	-	mo	-	to	-	0.03	0.06	616.55	mo	24.73	mo
14	to	-	mo	-	to	-	0.03	0.06	3596.39	to	147.89	mo
15	to	-	mo	-	to	-	0.03	0.06	to	-	498.72	mo
16	to	-	mo	-	to	-	0.03	0.06	to	-	to	-
2^5							0.03	0.06				
2^6							0.04	0.07				
2^{12}							3.65	3.59				
2^{13}							18.80	9.75				
2^{16}							1182.19	529.63				
2^{17}							to	-				

Baseline We compare our approach against the certifying rLive method [41], which is implemented on top of the nuXmv model checker[6] [12]. To the best of our knowledge, this is the most recent approach for liveness certification. Their implementation is limited to the rLive algorithm, and does not support k-liveness, stabilizer extraction, or L2S. Note that their workflow additionally requires an (unverified) Python-script translation for the input model and the generated certificate prior to checking. Moreover, their certificate checker requires first establishing the rLive proof strategy and LTL deductive rules; we exclude this from the reported certification time, which gives their approach a favorable comparison. We further removed the hard-coded memory limits in their tools. We used Apptainer virtualization to run SBCL on the HPC cluster.

[6] We are thankful to the authors of [41] for kindly providing the binaries.

Model checking vs. certification Table 1 and Fig. 7 report the experimental results on the counter benchmarks and the HWMCC benchmarks, respectively. On the counter benchmarks k-liveness with stabilizing constraint extraction significantly outperforms all other evaluated algorithms. While the benchmarks uniquely favor the technique, as Fig. 4 illustrates, this result extends to the representative HWMCC benchmark set. To put these results into perspective, up to 2^5 bits, explicit-state model checking is feasible, while 2^6 bits are completely out of reach of explicit-state enumeration. Without the preprocessing, the exponential value of k required makes k-liveness reach timeouts for very small instances, while liveness-to-safety quickly reaches the memory limit of 14 GB.

Our rLive implementation not only manages to find the exponentially long trace of liveness violations, but also produces invariants to block them one by one, yielding certificates that are surprisingly efficient to check. *Our certificate checker successfully validated all certificates generated by the model checker.* The nuXmv rLive implementation (closed-source) is more efficient, and its safety backend does not seem to suffer from the efficiency degradation that our IC3 implementation exhibits when dealing with symbolic initial states (as required by rLive). Notably, however, the baseline certification method requires significantly more memory than was available in our setup. Even the certificate for 9 bits, which was produced in 0.46 seconds, caused memory exhaustion during checking. For 8 bits, the observed certification overhead is approaching a factor of 3000.

Our experiments on the HWMCC benchmarks also show that under our approach, certification time constitutes only a small fraction of the model checking time: 4% for k-liveness (with and without constraint extraction) and 7% for our rLive implementation, whereas L2S incurs a substantially higher overhead of 59%. In contrast, for the baseline approach using nuXmv, certification is significantly more expensive than model checking itself, with overheads of 309% for rLive with optimizations enabled and 469% for the unoptimized version. Moreover, the Sindoni et al. approach failed to validate 16 and 13 certificates for the two configurations, respectively, whereas our method successfully validated all.

The reported overhead is exclusively certificate checking time. Since the safety checking algorithms already compute most of the required invariants and witness construction is linear, we expect the remaining overhead to be dominated by disk write time.

Comparison of certificate checking We further compare the certificate-checking efficiency of our approach with the deductive certification framework described in [25,26,41] as directly as possible. Across their publications, the authors present three implementations of their certification approach, each tailored to a different model checker. We run the model checkers on the HWMCC25 benchmark set to generate certificates and compare the certificate checking time with that of the certificates produced by the most similar engine in our model checker. We consider only benchmarks for which both our model checker and the related approach produced a certificate. The model checking time itself is excluded from this evaluation. The results are shown in Figure 8.

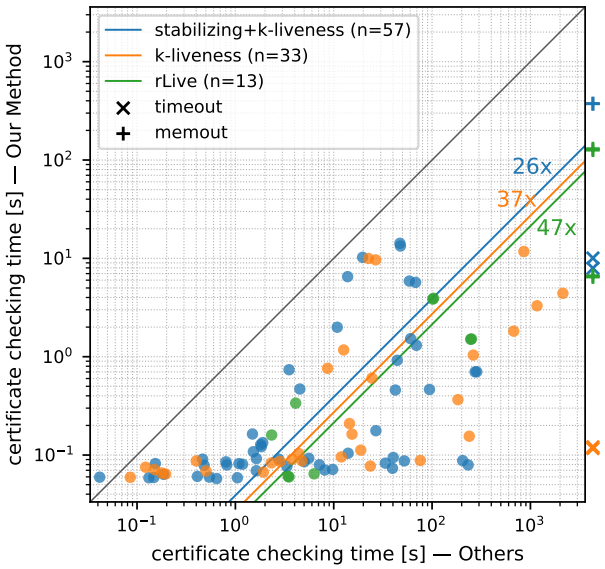

Fig. 8. Comparison of certificate checking time for our method to three implementations of the related approach. We compare the default configuration presented in the related work to the closest engine in our model checker. Blue compares Simplic3 [26] and proof checking performed by CVC5 [3] with our k-liveness engine with stabilizer extraction. Orange compares IC3ia [25] to our base k-liveness engine. The certificates produced by IC3ia include full resolution proofs and are checked with a Python script. Green compares the Sindoni et al. approach [41] to our rLive implementation. For each of the three comparisons, we present the geometric mean of the certificate checking time ratio. Timeouts and memory-outs are displayed in the margin and excluded from the calculation of the geometric mean. Only benchmarks for which certificates were produced by both model checkers are considered; the number is shown in parentheses.

The approach implemented in Simplic3 [26] (blue) is the most similar to ours, in that it dispatches proof obligations to an external, trusted SMT solver. In contrast to our approach, the formulas to be checked are generated directly by the model checker, and the model checker itself is trusted to encode the required proof obligations correctly. This is emphasized by the fact that the certificate checker does not consider the original model. Referring to Def. 8, this is similar to checking only the liveness of the witness (checks 6 and 7), without establishing the simulation relation. A direct comparison of certificate checking time shows that our certificates are 26 times faster to check in the geometric mean, even before accounting for the two timeouts and two memory-outs in which CVC5 failed to check the certificate within one hour and 14 GB of memory. We note two limiting aspects of this comparison: First, the certificates are produced in SMT-LIB format, and we therefore use CVC5 to check them. However, the incremental sequence of calls is purely Boolean and could presumably be checked

more efficiently by a dedicated tool. Second, the model checker producing these certificates is significantly more mature and implements more features, successfully producing certificates for 79 benchmarks with an overall overhead of 133%.

A previous version of the deductive framework presented in [25] (orange) is interesting, as the certificates include full resolution proofs and can therefore be checked in polynomial time. Nevertheless, the prototype checker for this format remains significantly slower than our approach and experiences two timeouts. Finally, the rLive engine of nuXmv [41] produces certificates for a checker based on PVS, which can be argued to have a smaller base of trust than our approach. While the number of considered benchmarks is limited (due to our weak rLive engine), their certificate checker still encounters four memory outs. Ignoring these cases, our certificates are still checked 47 times faster in the geometric mean.

Summary of results The experimental results empirically support the following conclusions: (1) our certification method is practically effective, incurring only a small fraction of the model checking time as overhead; (2) our method efficiently certified all instances solved by the model checker; (3) compared to other certification approaches, our method achieves substantially lower certification overhead, exceeding an order-of-magnitude improvement.

7 Conclusion

We have presented a unifying certificate format for liveness checking and shown how to generate such certificates for four representative model checking techniques. Our certificates are expressed as witness circuits, enabling efficient and lightweight proof checking based solely on SAT solving. We implemented all techniques in a certification toolchain, including a prototype certifying model checker. Experimental results on a broad range of competition benchmarks demonstrate that certification incurs only a very small overhead compared to the model checking time, making the approach practical for real-world use. We additionally compared our method against an existing certifying approach for rLive and showed that our technique achieves significantly better performance.

To the best of our knowledge, this is the first successful generic certification method for liveness checking that works for this wide range of techniques. As future work, we plan to extend the proposed certification framework to both word-level and infinite-state systems. To enable end-to-end certification for workflows in which properties are specified as arbitrary LTL formulas, we intend to develop a certifying translation to AIGER. Moreover, work has begun on developing a fully verified certificate checker in a theorem-proving environment.

Acknowledgments. Partially funded by the European Union project (ERC, CertiFOX, 101122653). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Further funded by, the Research Council of Finland under the project 369068, and a gift from Intel Corporation.

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

Data-Availability Statement All experiments presented in this paper can be reproduced: https://doi.org/10.5281/zenodo.19649248

References

1. Abate, A., Giacobbe, M., Roy, D.: Quantitative supermartingale certificates. In: International Conference on Computer Aided Verification. pp. 3–28. Springer (2025). https://doi.org/10.1007/978-3-031-98679-6_1
2. Alpern, B., Schneider, F.B.: Defining liveness. Information processing letters **21**(4), 181–185 (1985). https://doi.org/10.1016/0020-0190(85)90056-0
3. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., et al.: cvc5: A versatile and industrial-strength SMT solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24
4. Biere, A., Artho, C., Schuppan, V.: Liveness checking as safety checking. In: Cleaveland, R., Garavel, H. (eds.) 7th International ERCIM Workshop in Formal Methods for Industrial Critical Systems, FMICS 2002, ICALP 2002 Satellite Workshop, Málaga, Spain, July 12-13, 2002. Electronic Notes in Theoretical Computer Science, vol. 66, pp. 160–177. Elsevier (2002). https://doi.org/10.1016/S1571-0661(04)80410-9
5. Biere, A., Faller, T., Fleury, M., Froleyks, N., Pollitt, F.: CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2025. In: Codel, C., Fazekas, K., Heule, M., Iser, M. (eds.) Proceedings of SAT Competition 2025: Solver and Benchmark Descriptions. p. 10. Proceedings of SAT Competitions, TU Wien Academic Press (2025). https://doi.org/10.34726/10379
6. Biere, A., Froleyks, N., Preiner, M.: Hardware Model Checking Competition 2025. In: Irfan, A., Kaufmann, D. (eds.) Proceedings of the 25th Conference on Formal Methods in Computer-Aided Design, FMCAD 2025, Menlo Park, CA, USA, October 6-10, 2025. TU Wien Academic Press (2025). https://doi.org/10.34727/2025/ISBN.978-3-85448-084-6_6
7. Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria (2011)
8. Biere, A., Herbstritt, M., Berre, D.L., Wieringa, S., Niemetz, A.: AIGER: and-inverter graph (AIG) format and tools. https://github.com/arminbiere/aiger (2006)
9. Bradley, A.R.: Understanding IC3. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 1–14. Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_1
10. Bradley, A.R., Manna, Z., Sipma, H.B.: The polyranking principle. In: International Colloquium on Automata, Languages, and Programming. pp. 1349–1361. Springer (2005). https://doi.org/10.1007/11523468_109
11. Bradley, A.R., Somenzi, F., Hassan, Z., Zhang, Y.: An incremental approach to model checking progress properties. In: 2011 Formal Methods in Computer-Aided Design (FMCAD). pp. 144–153. IEEE (2011), http://dl.acm.org/citation.cfm?id=2157677

12. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: International Conference on Computer Aided Verification. pp. 334–342. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_22
13. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). pp. 52–59. IEEE (2012), https://ieeexplore.ieee.org/document/6462555/
14. Cook, B., Gotsman, A., Podelski, A., Rybalchenko, A., Vardi, M.Y.: Proving that programs eventually do something good. ACM SIGPLAN Notices **42**(1), 265–276 (2007). https://doi.org/10.1145/1190216.1190257
15. Cook, B., Podelski, A., Rybalchenko, A.: Terminator: Beyond safety: (tool paper). In: International Conference on Computer Aided Verification. pp. 415–418. Springer (2006). https://doi.org/10.1007/11817963_37
16. Dietsch, D., Heizmann, M., Langenfeld, V., Podelski, A.: Fairness modulo theory: A new approach to LTL software model checking. In: International Conference on Computer Aided Verification. pp. 49–66. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_4
17. Esparza, J., Lammich, P., Neumann, R., Nipkow, T., Schimpf, A., Smaus, J.G.: A fully verified executable LTL model checker. In: International Conference on Computer Aided Verification. pp. 463–478. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_31
18. Floyd, R.W.: Assigning meanings to programs. In: Program Verification: Fundamental Issues in Computer Science, pp. 65–81. Springer (1993). https://doi.org/10.1007/978-94-011-1793-7_4
19. Froleyks, N., Yu, E.: Hardware model checking certification with Certifaiger and Cerbtora. In: Proceedings of the International Joint Conference on Automated Reasoning (IJCAR) 2026, Part of FLoC 2026, Lisbon, Portugal, July 26–29, 2026. Lecture Notes in Computer Science, Springer, Lisbon, Portugal (2026)
20. Froleyks, N., Yu, E., Biere, A., Heljanko, K.: Certifying phase abstraction. In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14739, pp. 284–303. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_17
21. Froleyks, N., Yu, E., Preiner, M., Biere, A., Heljanko, K.: Introducing certificates to the Hardware Model Checking Competition. In: Piskac, R., Rakamaric, Z. (eds.) Computer Aided Verification - 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15931, pp. 281–295. Springer (2025). https://doi.org/10.1007/978-3-031-98668-0_14
22. Gan, X., Dubrovin, J., Heljanko, K.: A symbolic model checking approach to verifying satellite onboard software. Sci. Comput. Program. **82**, 44–55 (2014). https://doi.org/10.1016/J.SCICO.2013.03.005
23. Giacobbe, M., Kroening, D., Pal, A., Tautschnig, M.: Neural model checking. Advances in Neural Information Processing Systems **37**, 86375–86398 (2024). https://doi.org/10.52202/079017-2742
24. Giacobbe, M., Kroening, D., Pal, A., Tautschnig, M.: Let a neural network be your invariant. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025)
25. Griggio, A., Roveri, M., Tonetta, S.: Certifying proofs for LTL model checking. In: Bjørner, N.S., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided

Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018. pp. 1–9. IEEE (2018). https://doi.org/10.23919/FMCAD.2018.8603022
26. Griggio, A., Roveri, M., Tonetta, S.: Certifying proofs for SAT-based model checking. Formal Methods in System Design **57**(2), 178–210 (2021). https://doi.org/10.1007/s10703-021-00369-1
27. Heljanko, K., Keinänen, M., Lange, M., Niemelä, I.: Solving parity games by a reduction to SAT. J. Comput. Syst. Sci. **78**(2), 430–440 (2012). https://doi.org/10.1016/j.jcss.2011.05.004
28. Kroening, D., Sharygina, N., Tsitovich, A., Wintersteiger, C.M.: Termination analysis with compositional transition invariants. In: International Conference on Computer Aided Verification. pp. 89–103. Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_9
29. Kuismin, T., Heljanko, K.: Increasing confidence in liveness model checking results with proofs. In: Haifa Verification Conference. pp. 32–43. Springer (2013). https://doi.org/10.1007/978-3-319-03077-7_3
30. Maretic, G.P., Dashti, M.T., Basin, D.A.: LTL is closed under topological closure. Inf. Process. Lett. **114**(8), 408–413 (2014). https://doi.org/10.1016/J.IPL.2014.03.001
31. McMillan, K.L.: Toward liveness proofs at scale. In: International Conference on Computer Aided Verification. pp. 255–276. Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_13
32. McMillan, K.L., Padon, O.: Ivy: A multi-modal verification tool for distributed algorithms. In: International Conference on Computer Aided Verification. pp. 190–202. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_12
33. Mebsout, A., Tinelli, C.: Proof certificates for SMT-based model checkers for infinite-state systems. In: FMCAD. pp. 117–124. IEEE (2016). https://doi.org/10.1109/FMCAD.2016.7886669
34. Murali, V., Trivedi, A., Zamani, M.: Closure certificates. In: Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control. pp. 1–11 (2024). https://doi.org/10.1145/3641513.3650120
35. Namjoshi, K.S.: Certifying model checkers. In: International Conference on Computer Aided Verification. pp. 2–13. Springer (2001). https://doi.org/10.1007/3-540-44585-4_2
36. Padon, O., Hoenicke, J., Losa, G., Podelski, A., Sagiv, M., Shoham, S.: Reducing liveness to safety in first-order logic. Proceedings of the ACM on Programming Languages **2**(POPL), 1–33 (2017). https://doi.org/10.1145/3158114
37. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 239–251. Springer (2004). https://doi.org/10.1007/978-3-540-24622-0_20
38. Podelski, A., Rybalchenko, A.: Transition invariants. In: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004. pp. 32–41. IEEE (2004). https://doi.org/10.1109/LICS.2004.1319598
39. Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. ACM SIGPLAN Notices **40**(1), 132–144 (2005). https://doi.org/10.1145/1047659.1040317
40. Schuppan, V., Biere, A.: Efficient reduction of finite state model checking to reachability analysis. Int. J. Softw. Tools Technol. Transf. **5**(2-3), 185–204 (2004). https://doi.org/10.1007/S10009-003-0121-X

41. Sindoni, G., Griggio, A., Tonetta, S.: Certifying rLive: A new proof strategy for liveness model checking. In: Thiemann, R., Weidenbach, C. (eds.) Frontiers of Combining Systems - 15th International Symposium, FroCoS 2025, Reykjavik, Iceland, September 29 - October 1, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15979, pp. 386–403. Springer (2025). https://doi.org/10.1007/978-3-032-04167-8_21
42. Sindoni, G., Pasini, P., Cabodi, G.: A theorem prover based approach for SAT-based model checking. In: Automated Deduction–CADE 30: 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings. vol. 15943, p. 449. Springer Nature (2025). https://doi.org/10.1007/978-3-031-99984-0_24
43. Sprenger, C.: A verified model checker for the modal μ-calculus in Coq. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 167–183. Springer (1998). https://doi.org/10.1007/BFb0054171
44. Urban, C.: FuncTion: An abstract domain functor for termination: (competition contribution). In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 464–466. Springer (2015)
45. Wolper, P., Vardi, M.Y., Sistla, A.P.: Reasoning about infinite computation paths. In: 24th Annual Symposium on Foundations of Computer Science (SFCS 1983). pp. 185–194. IEEE (1983)
46. Xia, Y., Cimatti, A., Griggio, A., Li, J.: Avoiding the Shoals - A New Approach to Liveness Checking. In: Gurfinkel, A., Ganesh, V. (eds.) Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14681, pp. 234–254. Springer (2024). https://doi.org/10.1007/978-3-031-65627-9_12
47. Yu, E., Biere, A., Heljanko, K.: Progress in certifying hardware model checking results. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12760, pp. 363–386. Springer (2021). https://doi.org/10.1007/978-3-030-81688-9_17
48. Yu, E., Froleyks, N., Biere, A., Heljanko, K.: Towards compositional hardware model checking certification. In: Nadel, A., Rozier, K.Y. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, October 24-27, 2023. pp. 1–11. IEEE (2023). https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_12