Formal Methods in System Design
https://doi.org/10.1007/s10703-021-00369-1

Certifying proofs for SAT-based model checking

Alberto Griggio[1] · Marco Roveri[2] · Stefano Tonetta[1]

Accepted: 19 March 2021

Abstract

In the context of formal verification, certifying proofs are evidences of the correctness of a model in a deduction system produced automatically as outcome of the verification. They are quite appealing for high-assurance systems because they can be verified independently by proof checkers, which are usually simpler to certify than the proof-generating tools. Model checking is one of the most prominent approaches to formal verification of temporal properties and is based on an algorithmic search of the system state space. Although modern algorithms integrate deductive methods, the generation of proofs is typically restricted to invariant properties only. Moreover, it assumes that the verification produces an inductive invariant of the original system, while model checkers usually involve a variety of complex pre-processing simplifications. In this paper we show how, exploiting the k-liveness algorithm, to extend proof generation capabilities for invariant checking to cover full linear-time temporal logic (LTL) properties, in a simple and efficient manner, with essentially no overhead for the model checker. Besides the basic k-liveness algorithm, we integrate in the proof generation a variety of widely used pre-processing techniques such as temporal decomposition, model simplification via computation of equivalences with ternary simulation, and the use of stabilizing constraints. These techniques are essential in many cases to prove that a property holds, both for invariant and for LTL model checking, and thus need to be considered within the proof. We implemented the proof generation techniques on top of IC3 engines, and show the feasibility of the approach on a variety of benchmarks taken from the literature and from the Hardware Model Checking Competition. Our results confirm that proof generation results in negligible overhead for the model checker.

Keywords Certifying model checking · Linear-time temporal logic · LTL · Invariant checking · Liveness · Deductive proofs

✉ Alberto Griggio
griggio@fbk.eu

Marco Roveri
marco.roveri@unitn.it

Stefano Tonetta
tonettas@fbk.eu

[1] Fondazione Bruno Kessler, Bruno Kessler, Via Sommarive 18, 38123 Povo, Trento, Italy

[2] University of Trento Via Sommarive 9, 38123 Povo, Trento, Italy

Published online: 24 June 2021

1 Introduction

The application of formal methods in the certification of high-assurance systems demands the qualification of the verification tools to ensure a sufficient level of confidence in their results (see for example the DO-333 standard in the avionic domain [40]). However, verification tools such as model checkers can be quite complex, can leverage on numerous heuristics and combinations of techniques to be efficient (see also [44]). The idea of certifying model checking [35] to generate deductive proofs as byproduct of the verification is therefore quite appealing, because the proof can be verified by independent proof checkers, which are usually simpler to certify than the proof-generating tools.

Most modern model checking techniques integrate search-based and deductive methods such as induction. In particular, many current model checking algorithms are based on a sequence of SAT queries to find inductive invariants incrementally (e.g., IC3 [11]). Nevertheless, most works on certifying model checkers go back a decade, are mainly theoretical and based on μ-calculus, while practical SAT-based approaches are currently limited to invariant properties.

In the case of an invariant property ϕ, the basic idea of certifying model checking is quite intuitive. If you generate an inductive invariant ψ, the proof checker needs to prove that: (i) ψ is inductive; (ii) ψ is satisfied in the initial states; (iii) ψ entails the property ϕ. These are three validity problems that can be solved by a SAT solver. The idea naturally lifts also to k-inductive invariants [33].

However, in order to be effective on real-world benchmarks, most model checkers implement a variety of pre-processing simplifications. The generated invariant is therefore inductive relative to other invariants, for which we do not always have an inductive proof. This is the case for example for techniques such as temporal decomposition [14] and model simplification through computation of equivalences via ternary simulation [9,13].

If we consider liveness properties, the problem is even more complex because simplifications such stabilizing constraints [18] consider properties that are invariants eventually in the future, after an unbounded number of transitions.

Finally, if the property is expressed in temporal logic such as Linear Temporal Logic (LTL) [38], the proof generation needs to consider various transformations that are applied to the problem: model checking is reduced by contradiction to finding a counterexample; LTL formulae are encoded into symbolically-represented automata [19]; multiple fairness conditions resulting from such encoding are reduced to one; liveness is reduced typically to safety.

We propose a sound and complete approach that addresses the above issues. SAT-based model checking is extended to generate a proof for both invariant and LTL properties. In the second case, the proof is generated from the inductive invariant obtained with the k-liveness algorithm [18] by combining standard resolution with inference rules specific for LTL, and reasoning by contradiction: by assuming that initially the negation of the property holds, we prove that a certain fairness condition can be visited at most k times, in contradiction with the validity of the fairness condition itself. The proof is built from the original initial and transition conditions of the system, thus reverting internal transformations including the introduction of a counter for k-liveness, the monitoring variables for the degeneralization of the multiple fairness conditions, the tableau variables for the LTL encoding. Finally, the proof generation takes into account different pre-processing techniques such as temporal decomposition, model simplification through computation of equivalences via ternary simulation, and the use of stabilizing constraints.

The resulting approach is simple and efficient, and it can be implemented on top of any state-of-the-art SAT-based LTL and invariant model checker based on the combination of engines capable of producing inductive invariants (e.g., IC3 [11]). The proposed proof generation techniques result in essentially no overhead for the model checker. It can be applied as is also for generating proofs of the validity of LTL formulae.

We have implemented the proof generation technique within the SIMPLIC3 backend [27] of the NUXMV [15] model checker, and on top of IC3IA [20], a simple and open-source implementation of IC3 that uses the MATHSAT [16] SMT solver as backend. We carried out a thorough experimental evaluation on the several benchmarks taken from the literature and from the latest Hardware Model Checking Competition (HWMCC) [7]. The results show the feasibility of the approach on the considered benchmarks, and confirm the small impact of the proof construction on the overall verification process. Finally, we also implemented a prototype proof-checker in Python, on top of the MATHSAT [16] SMT solver, to check the correctness of the generated proofs, and we executed it on each of the generated proofs. The results show that, for our prototype implementation, the cost of proof checking is comparable with the cost of verification.

This paper is structured as follows. In Sect. 2 we analyze the related works. In Sect. 3 we provide the needed background. In Sect. 4 we discuss the proposed approach to compute proofs for LTL model checking, and in Sect. 5 we show the results of our experimental evaluation. Finally, in Sect. 6 we draw conclusions and outline future work.

2 Related work and contributions

This work extends the techniques first presented in [28], mainly by considering the preprocessing techniques of temporal decomposition, model simplification via computation of equivalences with ternary simulation, and the use of stabilizing constraints. Differently from [28], this work also details the deduction proofs in terms of the rules defined in the standard system for LTL first proposed by [26] (without introducing new rules). The presentation of the material has also been improved, by providing a uniform treatment of proof generation for both invariant and LTL properties. Finally, the experimental evaluation and the results have been extended by considering also a new implementation of proof generation within the SIMPLIC3 [27] backend of NUXMV [15], and by including in the evaluation the benchmarks used in the latest edition of the Hardware Model Checking Competition (HWMCC) [7].

Among the most related works, [31] proposes to reduce liveness to safety (with a variant of k-liveness) and to generate a proof for the resulting invariant property. However, the translation is trusted and the proof does not target the original system but just the result of the reduction. In contrast, our work produces a temporal proof for the original system, so that only the proof checker must be trusted.

Another very relevant work is presented in [35], which describes a deductive proof system for verifying properties expressed in the μ-calculus, and shows how to generate a proof in this system from a model checking run. The proposed approach is applicable both for explicit state and symbolic search. The proof system and the proof generation process draw on results which relate model checking for the μ-calculus to winning parity games [23]. The system was implemented (as a prototype) on top of a BDD-based engine (COSPAN [25]); it is however unclear how to adapt it to modern SAT-based engines. Our approach instead implements proof generation on top of SAT-based algorithms without any substantial overhead or modification of the model checking engine. Moreover, although in terms of expressiveness LTL is more restricted than μ-calculus, in [35], LTL is assumed to be encoded and it is not shown how to convert the proof for the resulting μ-calculus formula to a proof for LTL. Our work instead produces a proof using inference rules for LTL, automatically reverting the internal automata construction.

Other approaches targeting the generation of proof from model checking of LTL properties include [21,32,36,37]. These works are however mostly theoretical, and to the best of our knowledge, with no implementation available.

Related, but slightly-different, problems are addressed in [2,4,17]. The first work gives a technique to incrementally build a (partial) deductive proof from the search performed by a model checker for incomplete (partially specified) systems while proving a given LTL property holds; the second focuses on runtime monitoring, proposing a local proof system for LTL and showing how such a system can be used for the construction of online runtime monitors; the third work instead discusses a proof system to provide evidence why a trace violates an LTL specification, as opposed to certifying why the property holds on the system under verification.

The work in [24] presents an LTL model checker whose code has been completely verified using the Isabelle theorem prover. The proof consists of the formal verification of a few hundred lines of “formalized pseudo-code”, and a verified refinement step in which mathematical sets and other abstract structures are replaced by implementations of efficient structures. The resulting checker is slower than unverified checkers, but it can be used as a trusted reference implementation.

Finally, some theorem provers for LTL can produce proofs, such as TRP++ [29] and TeMP [30]. Both systems are based on the temporal resolution calculus and can produce fine-grained proofs, which can then be inspected and checked to certify the correctness. However, no automatic proof checkers are available.

Overall, to the best of our knowledge, no previous work (apart from the preliminary results of this work presented in [28]) provides the following original contributions:

- a technique that generates temporal deductive proofs from SAT-based LTL model checking;
- a proof-generation technique based on the symbolic encoding of LTL into fair transition systems;
- a proof-generation technique for LTL validity based on model checking;
- a proof-generation technique for SAT-based (invariant or LTL) model checking that considers widely used simplification techniques (temporal decomposition, model simplification with equivalences computed with ternary simulation [9,13], and the use of stabilizing constraints [18]);
- an available effective implementation of proof generation from Invariant and LTL model checking.

3 Background

We work in the setting of Boolean (i.e. propositional) logic, with the standard notions of satisfiability, validity, interpretations and models. We denote propositional variables with v, x, y, and formulae with ϕ, ψ, f, α, β, I, T, possibly with subscripts or primes (e.g v_1, x'). If V, V' are (disjoint) sets of variables, we write $\phi(V, V')$ to stress that all the variables occurring in ϕ belong to $V \cup V'$. We use $ite(\phi_c, \phi_t, \phi_e)$ as a shorthand for $(\phi_c \rightarrow \phi_t) \wedge (\neg\phi_c \rightarrow \phi_e)$. Given a variable v, a formula ϕ and a formula ψ not containing v, we denote with $\phi[v := \psi]$ the result of substituting v with ψ everywhere in ϕ. We extend this to sets of variables in a pointwise manner. If V and V' are two disjoint sets of variables, we might write $\phi[V := V']$ as ϕ'. A counter is an integer-valued variable c that occurs in two kinds of predicates: comparisons with constants, such as $c = 0$ or $c \leq 10$; and conditional increments, such as $ite(f, c' = c + 1, c' = c)$. Abusing notation, and for the sake of readability, in the following we sometimes use counters to denote their equivalent propositional encoding (this can be done in a standard way, using e.g. a unary or a binary encoding for integer numbers and the required comparison and increment operations).

3.1 Transition Systems

A *transition system* M is a tuple $M = \langle V, I, T\rangle$ where V is a set of (propositional) state variables, $I(V)$ is a formula representing the initial states, and $T(V, V')$ is a formula representing the transitions.

A *state* of M is an assignment to the variables V. We denote with Σ_V the set of states. We say that a state $s \in \Sigma_V$ is a model for a propositional formula $\phi(V)$ (denoted $s \models \phi(V)$) if substituting in ϕ the values of the variables in s, the formula ϕ evaluates to $\top$. A [finite] *path* of M is an infinite sequence $s_0, s_1, \ldots$ [resp., finite sequence $s_0, s_1, \ldots, s_k$] of states such that $s_0 \models I$ and, for all $i \geq 0$ [resp., $0 \leq i < k$], $s_i, s'_{i+1} \models T$. Given $\sigma := s_0, s_1, \ldots$, with $\sigma[j]$ we denote the state s_j, and with σ^j the path $s_j, s_{j+1}, \ldots$. Given two transitions systems $M_1 = \langle V_1, I_1, T_1\rangle$ and $M_2 = \langle V_2, I_2, T_2\rangle$, we denote with $M_1 \times M_2$ the synchronous product $\langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2\rangle$.

3.2 Invariant Properties

Given a propositional formula ϕ, the invariant model checking problem, denoted with $M \models_{fin} \phi$, is the problem to check if, for all finite paths $s_0, s_1, \ldots, s_k$ of M, $s_k \models \phi$.

Most model checkers prove an invariant property by generating a stronger invariant formula ψ that is *inductive*, i.e. such that: (i) $I \rightarrow \psi$; (ii) $\psi \wedge T \rightarrow \psi'$; and (iii) $\psi \rightarrow \phi$.

3.3 LTL

Given a set of propositional variables V, LTL formulae are built using Boolean connectives and the temporal operators $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"). Formally,

- a variable $v \in V$ is an LTL formula and $\top$ is an LTL formula;
- if ϕ_1 and ϕ_2 are LTL formulae, then $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\mathbf{X}\phi_1$ and $\phi_1\mathbf{U}\phi_2$ are LTL formulae.

We use the standard abbreviations: $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \rightarrow \phi_2 := \neg\phi_1 \vee \phi_2$, $\phi_1 \leftrightarrow \phi_2 := (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$, $\bot := p \wedge \neg p$, $\mathbf{F}\phi := \top\mathbf{U}\phi$, $\mathbf{G}\phi := \neg\mathbf{F}\neg\phi$, $\mathbf{X}^0\phi := \phi$, and $\mathbf{X}^{n+1} := \mathbf{X}\mathbf{X}^n\phi$ for all $n \geq 0$.

Given an LTL formula ϕ, a sequence σ of assignments to V, and an index i, we define $\sigma, i \models \phi$, i.e., that σ satisfies the formula ϕ in i, as follows:

- $\sigma, i \models \top$
- $\sigma, i \models v$ iff $\sigma[i] \models v$
- $\sigma, i \models \phi \wedge \psi$ iff $\sigma, i \models \phi$ and $\sigma, i \models \psi$
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$
- $\sigma, i \models \mathbf{X}\phi$ iff $\sigma, i + 1 \models \phi$

- $\sigma, i \models \phi \mathbf{U} \psi$ iff for some $j \geq i, \sigma, j \models \psi$ and for all $i \leq k < j, \sigma, k \models \phi$.

Finally, $\sigma \models \phi$ iff $\sigma, 0 \models \phi$.

Given an LTL formula ϕ, the LTL model checking problem, denoted with $M \models \phi$, is the problem to check if, for all (infinite) paths σ of M, $\sigma \models \phi$.

Given an LTL formula ϕ, the LTL validity problem, denoted by $\models \phi$, is the problem of checking if $\sigma \models \phi$ for all (infinite) paths over Σ_V. The validity problem can be reduced to the model checking problem by considering the universal model $M_U = \langle V, \top, \top \rangle$. It is easy to prove that $\models \phi$ iff $M_U \models \phi$.

3.4 Symbolic LTL Model Checking

The automata-based approach [43] to LTL model checking consists of building a transition system $M_{\neg\phi}$ with a set of fairness conditions $F_{\neg\phi}$ such that $M \models \phi$ iff $M \times M_{\neg\phi} \models \neg \bigwedge_{f \in F_{\neg\phi}} \mathbf{GF} f$. This reduces to finding a counterexample as a fair path, i.e., a path of the system that visits each fairness condition in $F_{\neg\phi}$ infinitely many times.

Following [19], the encoding of an LTL formula ϕ over variables V into a transition system $M_{\neg\phi} = \langle V_{\neg\phi}, I_{\neg\phi}, T_{\neg\phi} \rangle$ with fairness conditions $F_{\neg\phi}$ is defined as follows:

- $V_{\neg\phi} = V \cup \{v_{X\beta} \mid \mathbf{X}\beta \in Sub(\phi)\} \cup \{v_{\mathbf{X}(\beta_1 \mathbf{U} \beta_2)} \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\}$
- $I_{\neg\phi} = Enc(\neg\phi)$
- $T_{\neg\phi} = \bigwedge_{v_{\mathbf{X}\beta} \in V_{\neg\phi}} v_{\mathbf{X}\beta} \leftrightarrow Enc(\beta)'$
- $F_{\neg\phi} = \{Enc(\beta_1 \mathbf{U} \beta_2 \rightarrow \beta_2) \mid \beta_1 \mathbf{U} \beta_2 \in Sub(\phi)\}$

where Sub is a function that maps a formula ϕ to the set of its subformulae, and Enc is defined recursively as:

- $Enc(\top) = \top$
- $Enc(v) = v$
- $Enc(\phi_1 \wedge \phi_2) = Enc(\phi_1) \wedge Enc(\phi_2)$
- $Enc(\neg\phi_1) = \neg Enc(\phi_1)$
- $Enc(\mathbf{X}\phi_1) = v_{\mathbf{X}\phi_1}$
- $Enc(\phi_1 \mathbf{U} \phi_2) = Enc(\phi_2) \vee (Enc(\phi_1) \wedge v_{\mathbf{X}(\phi_1 \mathbf{U} \phi_2)})$

3.5 Degeneralization

In explicit-state model checking, the standard way to encode a Generalized Büchi Automaton with n fairness conditions into an equivalent "degeneralized" one (i.e., with one fairness), is to fix an order on the fairness conditions, replicate the automaton n times, and move from the i-th copy to the next one as soon as the i-th fairness condition is visited. Symbolically, this can be achieved as follows.

Given a transition system $M = \langle V, I, T \rangle$ with fairness conditions $F = \{f_1, \ldots, f_n\}$, we build an equivalent system with a single fairness condition f by considering $M \times M_{deg}$, where $M_{deg} = \langle V_{deg}, I_{deg}, T_{deg} \rangle$ is defined as follows:

- $V_{deg} = V \cup \{s\}$
- $I_{deg} = s = 0$
- $T_{deg} = \bigwedge_{0 \leq i < n-1} (s = i \rightarrow ite(f_{i+1}, s' = s + 1, s' = s)) \wedge (s = n - 1 \rightarrow ite(f_n, s' = 0, s' = s))$

and $f = s = 0 \wedge f_1$.

Most standard symbolic model checkers use a different encoding, which does not fix an ordering on the fairness conditions: one propositional variable per fairness condition is set to true whenever the fairness condition is visited, and when all the variables are true they are reset to false. The proof generation described in the next section is based on the above encoding with fixed ordering (see Sect. 4.5 for details on the reason). We analyze the impact of this choice experimentally in Sect. 5.

3.6 K-Liveness and SAT-based Symbolic Model Checking

SAT-based algorithms take as input a propositional transition system and a property, and try to solve the verification problem with a series of satisfiability queries. IC3 [11] is a symbolic model checking algorithm for the verification of invariant properties. It builds an over-approximation of the reachable state space, using clauses obtained by generalization while disproving candidate counterexamples. In the case of finite-state systems, the algorithm is implemented on top of Boolean SAT solvers, fully leveraging their features. IC3 has demonstrated to be extremely effective, and it is a fundamental core in all the engines in hardware verification.

K-liveness [18] is an algorithm recently proposed to reduce liveness checking (and so also LTL verification) to a sequence of invariant checking problems. K-liveness uses the standard approach, outlined above, to reduce the LTL verification problem $M \models \varphi$ to $M \times M_{\neg\varphi} \times M_{deg} \models \neg \mathbf{GF} f$. In [18], it is proved that, for finite-state systems, $M \models \neg \mathbf{GF} f$ iff there exists k such that f can be visited at most k times along a path of M. The last check can be reduced to an invariant checking problem of the form $M \times M_c \models_{fin} (c \leq k)$, where $M_c := \langle V_c, I_c, T_c \rangle$ is defined as follows: $V_c := \{c\}$, $I_c := c = 0$, $T_c := ite(f, c' = c + 1, c' = c)$. K-liveness is therefore a simple loop that increases k at every iteration and calls a subroutine SAFE to check the invariant $(c \leq k)$ on $M \times M_c$. In particular, the implementation in [18] uses IC3 as SAFE and exploits the incrementality of IC3 to solve the sequence of invariant problems in an efficient way.

3.7 Simplifications Used in SAT-Based Model Checkers

State-of-the-art symbolic model checkers apply several pre-processing techniques, aimed at simplifying the input model before starting the actual property verification task. In this work, we consider some of the most popular techniques applied in SAT-based tools, namely temporal decomposition [14], extraction of stabilizing constraints [18], and equivalence detection via ternary simulation [9,13].

3.7.1 Temporal Decomposition

Temporal decomposition [14] is a technique that aims at simplifying the input system by detecting and removing logic that corresponds to initialization/reset sequences of the circuit (i.e. operations performed only in the first few steps of the system executions). Its basic idea is to compute (or guess) the length k of the reset sequence, and then replace the initial states of the system $M := \langle V, I, T \rangle$ with the set of states reachable in up to k steps, to obtain a new system $M^k := \langle V, Reach^k(I), T \rangle$, where $Reach^k(I)$ denotes the states reachable after k transition steps, defined inductively as follows:

$$Reach^k(\varphi) := \begin{cases} \varphi & \text{if } k = 0 \\ Reach^{k-1}((\exists V.\varphi \wedge T(V, V'))[V' := V]) & \text{otherwise.} \end{cases} \tag{1}$$

M^k can then be further simplified using other techniques (e.g. via equivalence detection – see Sect. 3.7.3). If the property P under verification is an invariant, the application of temporal decomposition also requires to check that for P no violation is found in the first k steps.

In SAT-based algorithms, the computation of $Reach^k$ is typically not performed explicitly, and instead it is replaced by a k-step unrolling of the system, in the style of bounded model checking [6]:

$$BMC(M)^k := \|I(V)\|^0 \wedge \bigwedge_{0 \le i < k} \|T(V, V')\|^i,$$

where $\|\alpha\|^i$ replaces each variable $v \in V$ in α with a corresponding fresh variable v^i and each v' with a corresponding fresh variable v^{i+1}. The search of the classical model checking is then performed using as transition relation $\|T(V, V')\|^k$ and as initial states $BMC(M)^k$, where all the variables v^i with $0 \le i < k$ are considered as inputs, and implicitly existentially quantified.

3.7.2 Stabilizing Constraints

A formula ψ is a *stabilizing constraint* [18] for M and ϕ iff $M \models \phi$ is equivalent to $M \models (\mathbf{FG}\psi) \rightarrow \phi$.

As shown in [18], if we are able to prove that $\mathbf{FG}(v \rightarrow v')$ for a variable $v \in V$, then we can use $(v = v')$ as stabilizing constraint. This can be strengthened further as follows. Assume we are solving $M \models \bigvee_i \mathbf{FG}\neg f_i$ for a a set of fairness conditions f_i, and that we have found a stabilizing constraint $(x = x')$ for a variable $x \in V$. If we find that $\mathbf{FG}(x \rightarrow \neg f_i)$ for some f_i, then it is safe to add $\neg x$ as stabilizing constraint and check $M \models \mathbf{FG}\neg x \rightarrow \bigvee_i \mathbf{FG}\neg f_i$. Dually, if we find that $\mathbf{FG}(\neg x \rightarrow \neg f_i)$ for some f_i, then we can add x as stabilizing constraint and check $M \models \mathbf{FG}x \rightarrow \bigvee_i \mathbf{FG}\neg f_i$.

As shown in [18], stabilizing constraints may also be used to find other stabilizing constraints. If we have found the stabilizing constraint $(x = x')$ by showing $M \models \mathbf{FG}x \rightarrow x'$ then we may use it when considering another candidate $y \in V$ as follows: $M \models (\mathbf{FG}x = x') \rightarrow (\mathbf{FG}y \rightarrow y')$.

The naive use of stabilizing constraints requires the use of a liveness checker to answer queries like $M \models \mathbf{FG}\alpha \rightarrow \mathbf{FG}b$, where α is the conjunction of the stabilizing constraints already found, and b is a proof obligation that may give rise to a new stabilizing constraint. As proposed in [18], the check can be approximated with a SAT check that only talks about two consecutive states of M: for example, instead of checking if $\mathbf{FG}(v \rightarrow v')$, we check if $(\alpha \wedge T) \rightarrow (v \rightarrow v')$, where T is the transition condition and α is the conjunction of previously found stabilizing constraints.

Finally, if α is the conjunction of all the found stabilizing constraints, instead of checking $M \models \mathbf{FG}\alpha \rightarrow \bigvee_i \mathbf{FG}\neg f_i$, [18] proposes to check the stronger condition $M \models \bigvee_i \mathbf{FG}(\alpha \rightarrow \neg f_i)$.

3.7.3 Equivalence and constant propagation

Several common pre-processing techniques are based on the simplification of formulae through the identification of equivalence classes of variables. These approaches are such that $I \wedge \mathbf{G}T \models \mathbf{G}\hat{T}$, where $\hat{T}$ is obtained by performing equivalence and constant propagations wrt. the equivalence classes identified. They are based on the following justification: if $\psi \models x = \phi$, then $\psi \models \varphi \leftrightarrow \varphi[x := \phi]$.

A simple and effective method for discovering equivalences is to use *ternary simulation* [42]. Given the input system $M := \langle V, I, T\rangle$, the method first partitions the variables V into *state variables* X and *input variables* Y, such that $V = X \cup Y$, $X \cap Y = \emptyset$, and I and T can be expressed as follows:[1]

$$I(X) := \bigwedge_{x_i \in X^+ \subseteq X} x_i \wedge \bigwedge_{x_j \in X^- \subseteq (X \setminus X^+)} \neg x_j \tag{2}$$

$$T(X \cup Y, X') := \bigwedge_{x_i \in X} x_i' \leftrightarrow f_i(X, Y). \tag{3}$$

Ternary simulation performs a symbolic simulation of the circuit using three-valued logic, in which a variable can assume also the value $\mathtt{X}$ (for “unknown”). The simulation starts from the state assigning to $\top$ all the variables in X^+, to $\bot$ all those in X^- (see (2)), and all the others to $\mathtt{X}$. Successor states are computed by applying (3) to the current state, until a fixpoint is reached (i.e. either an already-seen state is found, or a state with all variables set to $\mathtt{X}$ is reached). This amounts to computing an overapproximation of the reachable states of the circuit, that in turn is an invariant for the circuit itself. The accumulated set of states is analyzed to extract constant and equivalent variables as follows. A variable is constant if it is always assigned the same concrete value (i.e. either $\top$ or $\bot$) in all the states explored during the simulation. Two variables x_i and x_j are equivalent if they are never assigned an unknown value, and they always have the same value in each state. [2] This information is then used to simplify the transition relation.

3.8 Deduction Systems

A deduction system consists of a set of axiom schemes and inference rules. We use natural deduction [39] notation to represent proofs. A proof is a tree of formulae where leaves are axioms or hypothesis, and any other formula is obtained by the application of an inference rule. Proofs for propositional formulae can be built using the following inference rules:

[1] In general, in a hardware verification context, the input system is already in this *functional* form. Moreover, the technique can be extended to work also in the presence of further relational constraints on both X and Y, but this is omitted here for simplicity.

[2] This can be easily generalised to discover also XORs, i.e. cases in which x_i is equivalent to $\neg x_j$.

$$\frac{\alpha \rightarrow \beta \quad \alpha}{\beta}\ \text{IMP- E}\ (\mathit{modus\ ponens}) \qquad \begin{array}{c}[\alpha]\\ \vdots\\ \beta\\ \hline \alpha \rightarrow \beta\end{array}\ \text{IMP- I}$$

$$\frac{\alpha \wedge \beta}{\alpha}\ \text{AND- EL} \qquad \frac{\alpha \wedge \beta}{\beta}\ \text{AND- ER} \qquad \frac{\alpha \quad \beta}{\alpha \wedge \beta}\ \text{AND- I}$$

$$\begin{array}{c}[\neg\alpha]\\ \vdots\\ \bot\\ \hline \alpha\end{array}\ \text{RAA}\ (\mathit{reductio\ ad\ absurdum})$$

Resolution proofs obtained by SAT solvers (see e.g. [34]) can be converted to use the above rules. In order to use resolution proofs inside other proofs, we use the reductio ad absurdum rule. If a proof of $\bot$ can be derived using $\neg\alpha$ as hypothesis, we can extend it to a proof of α, removing α from the hypothesis.

A complete deductive system for LTL was first presented in [26]. Converting its axioms in natural deduction rules we get the following inference rules: [3]

$$\begin{array}{c}\vdots\\ \alpha\\ \hline \mathbf{G}\alpha\end{array}\ \text{G}\ (\mathit{generalization}) \qquad \frac{\alpha \quad \mathbf{G}(\alpha \rightarrow \mathbf{X}\alpha)}{\mathbf{G}\alpha}\ \text{IND}\ (\mathit{induction})$$

$$\frac{}{\mathbf{G}\alpha \leftrightarrow (\alpha \wedge \mathbf{XG}\alpha)}\ \text{G- EXP} \qquad \frac{}{(\alpha\mathbf{U}\beta) \leftrightarrow (\beta \vee (\alpha \wedge \mathbf{X}(\alpha\mathbf{U}\beta)))}\ \text{U- EXP}$$

$$\frac{}{\mathbf{X}\alpha \leftrightarrow \neg\mathbf{X}\neg\alpha}\ \text{LIN} \qquad \frac{\alpha\mathbf{U}\beta}{\mathbf{F}\beta}\ \text{F}$$

$$\frac{\mathbf{G}(\alpha \rightarrow \beta)}{\mathbf{G}\alpha \rightarrow \mathbf{G}\beta}\ \text{G- IMP- DIS} \qquad \frac{\mathbf{X}(\alpha \rightarrow \beta)}{\mathbf{X}\alpha \rightarrow \mathbf{X}\beta}\ \text{X- IMP- DIS}$$

$$\frac{}{\alpha}\ \text{PROP} \quad \text{for any propositional tautology } \alpha$$

Note that, in the generalization inference rule, a proof of $\mathbf{G}\alpha$ can be derived from α only when the proof of α does not have any hypothesis. From this, we can derive a similar rule to introduce $\mathbf{X}$:

$$\begin{array}{c}\vdots\\ \alpha\\ \hline \mathbf{X}\alpha\end{array}\ \text{X}$$

The derivation of the following rules can be found also in [3]:

$$\frac{}{\mathbf{G}(\alpha \wedge \beta) \leftrightarrow (\mathbf{G}\alpha \wedge \mathbf{G}\beta)}\ \text{G- AND- DIS}$$

$$\frac{}{\mathbf{X}(\alpha \wedge \beta) \leftrightarrow (\mathbf{X}\alpha \wedge \mathbf{X}\beta)}\ \text{X- AND- DIS}$$

$$\frac{}{\mathbf{GG}\alpha \leftrightarrow \mathbf{G}\alpha}\ \text{G- TRANS}$$

$$\frac{}{\mathbf{XG}\alpha \leftrightarrow \mathbf{GX}\alpha}\ \text{X- G- COM}$$

$$\frac{\mathbf{FG}\alpha}{\mathbf{GF}\alpha}\ \text{F- G- COM}$$

The following expansion rule is derived from multiple application of U- EXP:

$$\frac{}{\alpha \leftrightarrow Exp(\alpha)}\ \text{EXP}$$

[3] Note that here $\mathbf{F}\alpha$ and $\mathbf{G}\alpha$ are just abbreviations for $\top\mathbf{U}\alpha$ and $\neg(\top\mathbf{U}\neg\alpha)$ respectively, as introduced in Sect. 3.3. In principle, we could have used a system with simpler rules defined for the primitive operators. We preferred to keep the rules defined in [26].

where Exp is defined recursively as:

- $Exp(v) = v$
- $Exp(\phi_1 \wedge \phi_2) = Exp(\phi_1) \wedge Exp(\phi_2)$
- $Exp(\neg\phi_1) = \neg Exp(\phi_1)$
- $Exp(\mathbf{X}\phi_1) = \mathbf{X}\phi_1$
- $Exp(\phi_1 \mathbf{U} \phi_2) = Exp(\phi_2) \vee (Exp(\phi_1) \wedge \mathbf{X}(\phi_1 \mathbf{U} \phi_2))$

X distribution is obtained by multiple application of X- NOT- DIS, X- AND- DIS:

$$\frac{\mathbf{X}\alpha}{Next(\alpha)} \text{ NEXT}$$

where $Next$ is defined recursively as:

- $Next(v) = \mathbf{X}v$
- $Next(\phi_1 \wedge \phi_2) = Next(\phi_1) \wedge Next(\phi_2)$
- $Next(\neg\phi_1) = \neg Next(\phi_1)$
- $Next(\mathbf{X}\phi_1) = \mathbf{XX}\phi_1$
- $Next(\phi_1 \mathbf{U} \phi_2) = Next(\phi_2) \vee (Next(\phi_1) \wedge \mathbf{XX}(\phi_1 \mathbf{U} \phi_2))$

3.9 Certifying Model Checking for Invariants

In case of an invariant property ϕ, an inductive invariant ψ can be used to generate a proof of ϕ. In fact, since the formulae $I \rightarrow \psi$, $\psi \wedge T \rightarrow \psi'$, $\psi \rightarrow \phi$ are valid, we can obtain a resolution proof for each of them. Using an inductive inference rule, we can then deduce that ϕ holds in all reachable states.

Remark 1 Note that, there are two levels of proof generation and proof checking. In the first level, we can generate a set of proof obligations (which would in this case coincide with the three valid formulae built with the inductive invariant) and the proof checker has the task of discharging them by proving they are valid. In the second level, the proof generation can further generate a resolution proof for each proof obligation and in this case the proof checker has only the task of checking the single inference rules in the proof. In the first case, we need to trust the SAT solver proving the formulae valid (the input problem is simpler than the original model checking one, but the solver may be still very complex to be efficient). In the second case, we need to trust only a proof checker, which is usually much simpler. However, the generation of compact resolution proofs from SAT is a challenge on its own, and typically certifying model checkers are limited to the first level.

4 Certifying proofs for SAT-based LTL and invariant model checking

4.1 LTL Model Checking and LTL Validity

Consider the LTL model checking problem $M \models \phi$, where $M = \langle V, I, T \rangle$. With abuse of notation, we consider T also as an LTL formula, identifying v' with $\mathbf{X}v$ for every variable $v \in V$. In order to prove that $M \models \phi$, we provide a proof of $(I \wedge \mathbf{G}T) \rightarrow \phi$.

Note that, in case the original problem is the validity of an LTL formula ϕ, we reduce it to the model checking problem $M_U \models \phi$ (as explained in Sect. 3.4) generating a proof of ϕ since the initial and transition conditions of M_U are $\top$.

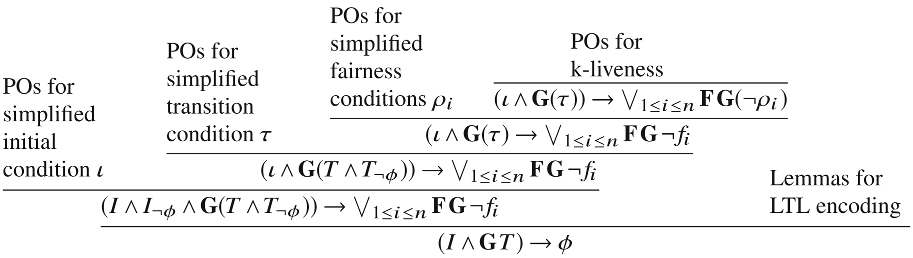

POs for simplified initial condition ι
POs for simplified transition condition τ
POs for simplified fairness conditions ρ_i
POs for k-liveness
$(\iota \wedge \mathbf{G}(\tau)) \rightarrow \bigvee_{1\leq i\leq n} \mathbf{FG}(\neg\rho_i)$
$(\iota \wedge \mathbf{G}(\tau) \rightarrow \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i$
$(\iota \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \rightarrow \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i$
$(I \wedge I_{\neg\phi} \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \rightarrow \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i$
Lemmas for LTL encoding
$(I \wedge \mathbf{G}T) \rightarrow \phi$

Fig. 1 Overall proof structure for $M \models \phi$

4.2 Overview of the deduction proof

As described in Sect. 3, the standard symbolic LTL model checking approach proceeds through a sequence of transformations. Thus, from the original problem $M \models \phi$, we arrive at the problem $M' \times M_{\neg\phi} \times M_{deg} \times M_c \models_{fin} c \leq k$, where M' is a simplification of the original M. In order to generate the proof for the original problem, we conceptually reverse this sequence showing how to generate a proof for each step. The overall proof structure is shown in Fig. 1.

All steps use the inference rules described in Sect. 3.8. Each step but last one is based on a set of proof obligations that are generated from the model checking algorithm. The last step is based only on valid formulas that show the correctness of the LTL encoding.

In the following we detail the proof obligations and how they are generated from the invariants extracted with the model checker. In Sect. 4.3, we define the inference rule and the respective proof obligations for the k-liveness reduction. In Sect. 4.4, we show how to generate them from an inductive invariant ψ obtained from k-liveness applied to one fairness; in Sect. 4.5, we generalize the approach to consider k-liveness with multiple fairness; in Sect. 4.7, we show how to generate the proof taking into account the simplifications of fairness, initial, and transition conditions; finally, in Sect. 4.6, we show how to generate the proof for the LTL property.

4.3 Inference Rule for K-Liveness

In the proofs generated from model checking with k-liveness, we use the following derived rule, denoted with KL[k] (see Section A.2 for the deduction details):

$$\frac{(Pi) \quad (Pn_1) \quad (pp_1) \quad \ldots \quad (pn_k) \quad (pp_k)}{(\iota \wedge \mathbf{G}\tau) \rightarrow \bigvee_{j\in J} \mathbf{FG}\neg\rho_j} \ \text{KL}[k]$$

where the $2k + 1$ premises are:

$$\iota \rightarrow \mathbf{F}\alpha_1 \qquad (Pi)$$

$$\mathbf{G}((\alpha_1 \wedge \tau \wedge \neg\rho_{j_1}) \rightarrow \mathbf{X}\alpha_1) \qquad (Pn_1)$$

$$\mathbf{G}((\alpha_1 \wedge \tau \wedge \rho_{j_1}) \rightarrow \mathbf{X}\alpha_2) \qquad (pp_1)$$

$$\ldots$$

$$\mathbf{G}((\alpha_k \wedge \tau \wedge \neg\rho_{j_k}) \rightarrow \mathbf{X}\alpha_k) \qquad (pn_k)$$

$$\mathbf{G}((\alpha_k \wedge \tau \wedge \rho_{j_k}) \rightarrow \bot) \qquad (pp_k)$$

Intuitively, this means that if there exist k conditions $\alpha_1, \ldots, \alpha_{k+1}$ such that α_1 is implied by ι and for each α_i there exists a corresponding ρ_{j_i} (can be even the same for all i) such that α_i is inductive relative to $\neg\rho_{j_i}$ for $1 \leq i \leq k$, α_{i+1} is implied by $\alpha_i \wedge \rho_{j_i}$ after a transition for $1 \leq i \leq k$, and $\alpha_{k+1} = \bot$, then any path starting from ι is such that there exists a j_i such that the condition ρ_{j_i} is visited finitely many times only.

4.4 Proof Obligations for Single Fairness

We consider first the special case of proving $M \models \neg\mathbf{GF}f$, where f is a propositional formula over V.

When $M \models \neg\mathbf{GF}f$ has been model checked by proving that f cannot be visited more than $k+1$ times, we instantiate the rule KL[k] using $\iota = I$, $\tau = T$, $\rho_i = f$ for all i, and the α_i are obtained by the inductive invariant generated with k-liveness.

If c is the counter introduced by k-liveness to count the occurrences of f and ψ is the inductive invariant over $V \cup \{c\}$ obtained to prove that $c < k$, then we instantiate the rule KL[k] using $\alpha_i = \psi[c := i-1]$.

Since ψ is the inductive invariant obtained with k-liveness we know that the following propositional formulae are valid:

$$I \wedge c = 0 \rightarrow \psi$$

$$\psi \wedge T \wedge ite(f, c' = c+1, c' = c) \rightarrow \psi'$$

$$\psi \rightarrow c < k$$

Therefore, the following formulae are also valid:

$$I \rightarrow \psi[c := 0] \qquad (pi)$$

$$(\psi[c := 0] \wedge T \wedge \neg f) \rightarrow \mathbf{X}\psi[c := 0] \qquad (pp_1)$$

$$(\psi[c := 0] \wedge T \wedge f) \rightarrow \mathbf{X}\psi[c := 1] \qquad (pn_1)$$

$$\ldots$$

$$(\psi[c := k-1] \wedge T \wedge \neg f) \rightarrow \mathbf{X}\psi[c := k-1] \qquad (pp_k)$$

$$(\psi[c := k-1] \wedge T \wedge f) \rightarrow \bot \qquad (pn_k)$$

Note that, the formulae $\alpha_1, \ldots, \alpha_k$ above are not required to be in any specific form. In particular, when instantiating them with the inductive invariant ψ, we can apply standard equivalence-preserving simplifications (e.g. $\alpha \wedge \top \equiv \alpha$) after the substitution of counter values.

For each of the above valid formulae, we can obtain a propositional resolution proof, which combined with the rule KL[k], yields a proof of $(I \wedge \mathbf{G}T) \rightarrow \neg\mathbf{GF}f$ in the following form:

$$\dfrac{\dfrac{\overline{pi}^{\;\text{PROP}}}{\mathbf{G}(pi)}\,\text{G} \quad \dfrac{\overline{pp_1}^{\;\text{PROP}}}{\mathbf{G}(pp_1)}\,\text{G} \quad \dfrac{\overline{pn_1}^{\;\text{PROP}}}{\mathbf{G}(pn_1)}\,\text{G} \quad \dots \quad \dfrac{\overline{pn_k}^{\;\text{PROP}}}{\mathbf{G}(pn_k)}\,\text{G}}{(I \wedge \mathbf{G}T) \rightarrow \neg\mathbf{GF}f}\,\text{KL}[k]$$

4.5 Generalization to Multiple Fairness Conditions

We now consider the case $M \models \neg(\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n)$, where $f_1, \ldots, f_n$ are propositional formulae over V. If we proved that $(I \wedge \mathbf{G}T) \rightarrow \neg(\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n)$ with k-liveness by proving that there exists an f_i can be visited at most $k+1$ times, we can instantiate the rule KL[$k \cdot n$] using $k \cdot n$ premises, which, for simplicity, we index with two indices i and j, where $1 \leq i \leq k$ and $1 \leq j \leq n$. Again we choose $\iota = I$, $\tau = T$, $\rho_{ij} = f_j$ for all i, and the α_i are obtained by the inductive invariant generated with k-liveness (when using the degeneralization described in Sect. 3.5).

More concretely, if c is the counter used to count the occurrences of the fairness conditions, s is the counter used to track if the i-th fairness has been visited, and ψ is the inductive invariant, we set $\alpha_{ij} = \psi[c := i-1, s := j-1]$ and generate a resolution proof for the following valid formulae (as in the previous case, we can simplify the formulae after substituting counter values, before generating the proofs):

$$I \rightarrow \psi[c := 0, s := 0] \tag{pi}$$

for $1 \leq i \leq k, 1 \leq j \leq n$

$$(\psi[c := i-1, s := j-1] \wedge T \wedge \neg f_j) \rightarrow \mathbf{X}\psi[c := i-1, s := j-1] \tag{pn_{ij}}$$

for $1 \leq i \leq k, 1 \leq j < n$

$$(\psi[c := i-1, s := j-1] \wedge T \wedge f_j) \rightarrow \mathbf{X}\psi[c := i-1, s := j] \tag{pp_{ij}}$$

for $1 \leq i < k$

$$(\psi[c := i-1, s := n-1] \wedge T \wedge f_n) \rightarrow \mathbf{X}\psi[c := i, s := 0] \tag{p_{in}}$$

$$(\psi[c := k-1, s := n-1] \wedge T \wedge f_n) \rightarrow \bot \tag{p_{kn}}$$

Similarly to the previous case, we can transform the resolution proofs for these lemmas in temporal proofs for the premises of the rule KL.

4.6 Certifying Proofs for LTL

We consider here the general case of $M \models \phi$. The procedure described in Sect. 3.6 reduces the problem to $M \times M_{\neg\phi} \models \neg(\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n)$, where $M \times M_{\neg\phi} = \langle V \cup V_{\neg\phi}, I \wedge I_{\neg\phi}, T \wedge T_{\neg\phi}\rangle$. Applying the procedure described above, we obtain a temporal proof of $(I \wedge I_{\neg\phi} \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \rightarrow \neg(\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n)$.

Every variable $v_{\mathbf{X}\beta} \in V_{\neg\phi}$ is associated with a temporal formula $\mathbf{X}\beta$. We denote by $Enc^{-1}(\alpha)$ the formula obtained from α by substituting every $v_{\mathbf{X}\beta}$ with $\mathbf{X}\beta$. By applying this substitution in the mentioned proof, we obtain a proof of $(I \wedge Enc^{-1}(I_{\neg\phi}) \wedge \mathbf{G}(T \wedge Enc^{-1}(T_{\neg\phi}))) \rightarrow \neg(\mathbf{GF}Enc^{-1}(f_1) \wedge \ldots \wedge \mathbf{GF}Enc^{-1}(f_n))$.

From this, as detailed in Section A.6, we derive a proof of $(I \wedge \mathbf{G}T) \rightarrow \phi$ with three resolution steps, using $\mathbf{GF}Enc^{-1}(f_i)$, $\mathbf{G}Enc^{-1}(T_{\neg\phi})$, and $\neg Enc^{-1}(I_{\neg\phi}) \rightarrow \phi$ as lemmas.

Finally, we derive a proof for each lemma (see again Section A.6 for the details). Note that, given the specific construction of $M_{\neg\phi}$, $Enc^{-1}(T_{\neg\phi})$ and $\mathbf{GF}Enc^{-1}(f_i)$ are always valid formulae. Moreover, note that $Enc^{-1}(I_{\neg\phi}) = Exp(\neg\phi) = \neg Exp(\phi)$ and that $Enc^{-1}(T_{\neg\phi})$ is in the form $\bigwedge_\beta \mathbf{X}\beta \leftrightarrow Next(Exp(\beta))$.

The following are therefore proofs for the above lemmas:[4]

$$\dfrac{\dfrac{}{\phi \leftrightarrow Exp(\phi)}\;\text{EXP}}{\neg Enc^{-1}(I_{\neg\phi}) \rightarrow \phi}\;\text{AND- EL} \qquad \dfrac{\dfrac{\dfrac{}{\beta \leftrightarrow Exp(\beta)}\;\text{EXP}}{\mathbf{X}(\beta \leftrightarrow Exp(\beta))}\;\text{X}}{\mathbf{X}\beta \leftrightarrow Next(Exp(\beta))}\;\textsc{Next}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{G}(\beta_1\mathbf{U}\beta_2 \wedge \neg\beta_2)]}{\mathbf{G}(\beta_1\mathbf{U}\beta_2) \wedge \mathbf{G}\neg\beta_2}\;\text{G- AND- DIS}}{\mathbf{G}\beta_1\mathbf{U}\beta_2}\;\text{AND- EL}}{(\beta_1\mathbf{U}\beta_2) \wedge \mathbf{XG}(\beta_1\mathbf{U}\beta_2)}\;\text{G- EXP}}{\beta_1\mathbf{U}\beta_2}\;\text{AND- EL}}{\mathbf{F}\beta_2}\;\text{F} \qquad \dfrac{\dfrac{[\mathbf{G}(\beta_1\mathbf{U}\beta_2 \wedge \neg\beta_2)]}{\mathbf{G}(\beta_1\mathbf{U}\beta_2) \wedge \mathbf{G}\neg\beta_2}\;\text{G- AND- DIS}}{\mathbf{G}\neg\beta_2}\;\text{AND- ER}}{\dfrac{\dfrac{\perp}{\mathbf{F}(\beta_1\mathbf{U}\beta_2 \rightarrow \beta_2)}\;\text{RAA}}{\mathbf{GF}(\beta_1\mathbf{U}\beta_2 \rightarrow \beta_2)}\;\text{G}}\;\text{IMP- E}$$

Example 1 We work out a full example showing the different steps from model checking to proof generation.

Le us consider the transition system $M = \langle V, I, T\rangle$ where:

$$V := \{x, y, z\} \qquad I := \top \qquad T := (x \rightarrow y') \wedge (y \rightarrow z')$$

and let us consider the property $\phi = \mathbf{G}(x \rightarrow \mathbf{F}z)$. Expanding $\mathbf{G}$, ϕ is equal to $\neg\mathbf{F}\neg(x \rightarrow \mathbf{F}z)$ and thus contains two $\mathbf{U}$-formulae: $\mathbf{F}(\neg(x \rightarrow \mathbf{F}z))$, which we abbreviate by $\mathbf{F}_1$, and $\mathbf{F}z$.

The transition system for the negation $\neg\phi$ is $M_{\neg\phi} = \langle V_{\neg\phi}, I_{\neg\phi}, T_{\neg\phi}\rangle$ where:

- $V_{\neg\phi} = \{x, z, v_{\mathbf{XF}_1}, v_{\mathbf{XF}z}\}$
- $I_{\neg\phi} = Enc(\neg\phi) = (x \wedge \neg(z \vee v_{\mathbf{XF}z})) \vee v_{\mathbf{XF}_1}$
- $T_{\neg\phi} = (v_{\mathbf{XF}_1} \leftrightarrow ((x' \wedge \neg(z' \vee v'_{\mathbf{XF}z})) \vee v'_{\mathbf{XF}_1})) \wedge (v_{\mathbf{XF}z} \leftrightarrow (z' \vee v'_{\mathbf{XF}z}))$

with fairness conditions $Enc(f_1)$ and $Enc(f_2)$ where:

$$f_1 = \neg\mathbf{F}_1 \vee \neg(x \rightarrow \mathbf{F}z) \qquad f_2 = \neg\mathbf{F}z \vee z$$

[4] Note that $(\phi \leftrightarrow Exp(\phi))$ is an abbreviation for $((\phi \rightarrow Exp(\phi)) \wedge (Exp(\phi) \leftrightarrow \phi))$ so that we can apply the AND- EL.

M_{deg} and M_c are defined as in Sects. 3.5 and 3.6.

Let us suppose that k-liveness produces the following inductive invariant:

$$\begin{aligned}\psi =&(Enc(\neg\phi) \wedge (\neg x \vee z \vee \upsilon_{\mathbf{XF}z}) \wedge s = 0 \wedge c = 0) \vee \\ &(y \wedge \neg z \wedge \neg \upsilon_{\mathbf{XF}z} \wedge s = 1 \wedge c = 0)\end{aligned}$$

After substituting and simplifying, we obtain: $\alpha_{01} = Enc(\neg\phi) \wedge (\neg x \vee z \vee \upsilon_{\mathbf{XF}z})$ $\alpha_{02} = y \wedge \neg z \wedge \neg \upsilon_{\mathbf{XF}z}$ $\alpha_{11} = \alpha_{12} = \bot$

Let us consider only a non-trivial case and produce a proof for $TL := (\neg\phi \wedge (\neg x \vee z \vee \mathbf{XF}z) \wedge T \wedge T_{\neg\phi} \wedge f_1) \rightarrow \mathbf{X}(y \wedge \neg z \wedge \neg \mathbf{XF}z)$. From the SAT solver we can obtain the following resolution proof for $L = Enc(\neg\phi) \wedge (\neg x \vee Enc(\mathbf{F}z)) \wedge T \wedge T_{\neg\phi} \wedge Enc(f_1) \wedge (\neg y' \vee Enc(\mathbf{F}z)')$:

$$\dfrac{\dfrac{\dfrac{L}{x \rightarrow y'} \quad \dfrac{\dfrac{\dfrac{L}{Enc(\neg\phi)} \quad \dfrac{L}{Enc(f_1)}}{x \wedge \neg Enc(\mathbf{F}z)}}{x}}{y'} \qquad \dfrac{\dfrac{\dfrac{L}{T_{\neg\phi}} \quad \dfrac{\dfrac{\dfrac{L}{Enc(\neg\phi)} \quad \dfrac{L}{Enc(f_1)}}{x \wedge \neg Enc(\mathbf{F}z)}}{\neg Enc(\mathbf{F}z)}}{\neg Enc(\mathbf{F}z)'} \quad \dfrac{L}{\neg y' \vee Enc(\mathbf{F}z)'}}{\neg y'}}{\bot}$$

In order to obtain a proof of TL it is sufficient to substitute in the above proof the variables $\upsilon_{\mathbf{XF}_1}$ and $\upsilon_{\mathbf{XF}z}$ with respectively $\mathbf{XF}_1$ and $\mathbf{XF}z$ and apply X- AND- DIS to obtain the $\mathbf{X}$ in the right-hand side of the implication in TL.

Finally, to obtain a proof of $(I \wedge \mathbf{G}T) \rightarrow \phi$ we instantiate the lemmas to remove $I_{\neg\phi}$, $T_{\neg\phi}$, and the fairness conditions.

For example, the proof for the lemma $\mathbf{GF}f_1$ is obtained by substituting β_1 with $\mathbf{F}_1$ and β_2 with $\neg(x \rightarrow \mathbf{F}z)$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{G}\neg f_1]}{\mathbf{GF}_1 \wedge \mathbf{G}(x \rightarrow \mathbf{F}z)}\,\text{G- AND- DIS}}{\mathbf{GF}_1}\,\text{AND- EL}}{\mathbf{F}_1 \wedge \mathbf{XGF}_1}\,\text{G- EXP}}{\mathbf{F}_1}\,\text{AND- EL} \qquad \dfrac{\dfrac{[\mathbf{G}\neg f_1]}{\mathbf{GF}_1 \wedge \mathbf{G}(x \rightarrow \mathbf{F}z)}\,\text{G- AND- DIS}}{\mathbf{G}(x \rightarrow \mathbf{F}z)}\,\text{AND- ER}}{\bot}\,\text{IMP- E}}{\mathbf{F}f_1}\,\text{RAA}}{\mathbf{GF}f_1}\,\text{G}$$

4.7 Handling Simplifications

In this section we detail the additional steps to extend the proof when the model checker applies simplifications of the initial, transition or fairness conditions. These simplifications are clearly optional and the general schema of Fig. 1 is valid even if some of them are not applied. In that case, $\iota = I \wedge I_{\neg\phi}$, $\tau = T \wedge T_{\neg\phi}$, and/or $\rho_i = f_i$ and the related extra proof obligations are not present.

4.7.1 Simplified initial condition with temporal decomposition

When using temporal decomposition (see Sect. 3.7.1), the initial condition is replaced by a condition that overappoximates the reachable states after k transitions. Therefore, we extend the proof using the following derived rule (see Section A.5 for the deduction details):

$$\frac{(\alpha \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(\beta)) \rightarrow \mathbf{X}^k(\iota) \quad \mathbf{G}((\iota \wedge \mathbf{G}(\beta)) \rightarrow \bigvee_i \mathbf{F}\gamma_i)}{(\alpha \wedge \mathbf{G}\beta) \rightarrow \bigvee_i \mathbf{F}\gamma_i} \text{ TD}$$

We instantiate the rule considering $\alpha = I \wedge I_{\neg\phi}$, $\beta = T \wedge T_{\neg\phi}$, $\gamma_i = \neg f_i$, and $\iota = \alpha_0 = \alpha[c := 0]$, where α is the inductive invariant obtained with k-liveness. In fact, since α_0 is implied by the simplified initial condition then

$$(\|I\|^0 \wedge \|I_{\neg\phi}\|^0 \wedge \bigwedge_{0 \le i < k} (\|T\|^i \wedge \|T_{\neg\phi}\|^i)) \rightarrow \|\alpha_0\|^k$$

thus the following formulae are valid

$$(I \wedge I_{\neg\phi} \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(T \wedge T_{\neg\phi})) \rightarrow \mathbf{X}^k\alpha_0$$

$$(\alpha_0 \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \rightarrow \bigvee_i \mathbf{F}\neg f_i$$

Since they are valid, we can apply rule G to the second one and obtain the premises of the rule TD.

4.7.2 Simplified transition condition with additional invariants

When using the optimization described in Sect. 3.7.3, some invariant ξ (for example, equalities between variables) is used to simplify the transition relation. Therefore, we extend the proof using the following derived rule (see Section A.4 for the deduction details):

$$\frac{\alpha \rightarrow \chi \quad \mathbf{G}((\chi \wedge \beta) \rightarrow \chi') \quad \mathbf{G}(\chi \rightarrow \xi) \quad (\alpha \wedge \mathbf{G}(\xi \wedge \beta \wedge \xi')) \rightarrow \gamma}{(\alpha \wedge \mathbf{G}(\beta)) \rightarrow \gamma} \text{ ST}$$

In this case, we simply instantiate the rule considering $\alpha = \iota$, $\beta = T \wedge T_{\neg\phi}$, $\gamma = \bigvee_i \mathbf{F}\neg f_i$, and χ is an inductive invariant that proves ξ.

In particular, when using ternary simulation to discover equalities, we need to extract a corresponding inductive invariant. Note that, ternary simulation computes at each step an over-approximation of the states reachable with that step. Thus, after reaching fixpoint, the computed abstract states are an over-approximation of the reachable states, which is inductive. Therefore, in order to compute an inductive invariant that entails the discovered equalities, we take the disjunction of the assignments to the variables having value different from X in each state reached with ternary simulation until fixpoint.

4.7.3 Simplified fairness conditions with stabilizing constraints

When the model checker uses stabilizing constraints as described in Sect. 3.7.2, a condition γ is built iteratively with two rules: if the transition relation and the current γ entail $x \rightarrow x'$ for some x, then γ is strengthened with $x = x'$; if moreover $\neg x$ entails $\neg f$ for some fairness f, then γ is strengthened with x. The condition γ is then used to weaken the fairness condition.

Therefore, we extend the proof system with the following derived rules (the derivation details are given in Section A.3):

$$\frac{\mathbf{G}(\tau \to (\gamma \to (x \to x')))\mathbf{G}\tau \to (\mathbf{FG}\gamma \vee \psi)}{\mathbf{G}\tau \to (\mathbf{FG}(\gamma \wedge x = x') \vee \psi)}\ \text{STAB}_1$$

$$\frac{\mathbf{G}(\tau \to (\gamma \to (x \to x')))\mathbf{G}(\tau \to (\gamma \to (\neg x \to \neg f)))\mathbf{G}\tau \to (\mathbf{FG}\gamma \vee \mathbf{FG}\neg f)}{\mathbf{G}\tau \to (\mathbf{FG}(\gamma \wedge x) \vee \mathbf{FG}\neg f)}\ \text{STAB}_2$$

These rules are applied iteratively following the sequence $\gamma_0, \gamma_1, \ldots, \gamma_m$ of increasingly stronger constraints found following the procedure described in Sect. 3.7.2. Thus, τ instantiated by $T \wedge T_{\neg\phi}$, $\gamma_0 = \top$ and the right premise of STAB_1 and STAB_2 is valid. After applying STAB_1 or STAB_2, γ_1 is either $x = x'$ or x. After applying again STAB_1 or STAB_2, γ_2 is either $\gamma_0 \wedge y = y'$ or $\gamma_0 \wedge y$ (for some y), and so on.

At the end, we obtain a proof of $\mathbf{G}\tau \to (\mathbf{FG}\gamma_m \vee \bigvee_i \mathbf{FG}\neg f_i)$, from which in combination with the proof of $(\iota \wedge \mathbf{G}\tau) \to (\bigvee_i \mathbf{FG}(\gamma_m \to \neg f_i))$, we derive $(\iota \wedge \mathbf{G}\tau) \to (\bigvee_i \mathbf{FG}\neg f_i)$.

4.8 Extended Certifying Proofs for Invariants

We extend the traditional proof for invariant based on the generation of an inductive invariant to take into account the simplification of the initial conditions with temporal decomposition and the transition condition based on discovered equalities. For the latter, the extension is the same applied to liveness properties in Sect. 4.7.2. As for temporal decomposition, we need to add a proof obligation and use the following variant of the derived rule (also proved in Section A.5):

$$\frac{(\alpha \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(\beta)) \to \mathbf{X}^k(\iota) \quad (\alpha \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(\beta)) \to (\bigwedge_{0 \le i < k} \mathbf{X}^i\gamma) \quad (\iota \wedge \mathbf{G}\beta) \to \mathbf{G}\gamma}{(\alpha \wedge \mathbf{G}(\beta)) \to \mathbf{G}\gamma}\ \text{TDI}$$

Thus, the extended overall proof for invariant is as follows:

$$\dfrac{\text{POs for simplified initial condition } \iota \qquad \dfrac{\text{POs for simplified transition condition } \tau \qquad \dfrac{\iota \to \psi \quad (\psi \wedge \tau) \to \psi' \quad \psi \to P}{(\alpha \wedge \mathbf{G}(\tau)) \to \mathbf{G}P}}{(\iota \wedge \mathbf{G}T) \to \mathbf{G}P}\ \text{ST}}{(I \wedge \mathbf{G}T) \to \mathbf{G}P}\ \text{TDI}$$

4.9 Correctness

Differently from the conference version of this paper, we used only inference rules belonging to the deduction system of [26] (see Sect. 3.8), which is sound and complete. Thus, every proof generated with the above method is correct. Moreover, since we are considering finite-state models and the considered model checking algorithms are complete, the above method can produce a proof for every valid formula.

5 Experimental evaluation

5.1 Setup

5.1.1 Implementation

We have implemented our proof generation procedure on top of two model checking tools: IC3IA [20] and SIMPLIC3 [27].

IC3IA is a simple, open-source implementation of IC3 that uses the MATHSAT [16] SMT solver as backend. The tool supports invariant and full LTL model checking of both finite and infinite-state systems (using a combination of implicit abstraction and well-founded relations, as described in [20]). Currently, proof generation is only available for finite-state systems. Its support for LTL is based on an automata based approach [43] relying on the symbolic encoding of [19]. Upon successful verification, it generates a deduction proof which can be checked by a simple companion proof checker, using purely-syntactic operations. The resolution proofs for the individual proof obligations are generated using the off-the-shelf proof-production capabilities provided by MATHSAT.

SIMPLIC3 is the backend of the NUXMV model checker [15] that participated in the latest Hardware Model Checking Competition (HWMCC) [7]. It accepts hardware model checking problems in the Aiger [8] format used for the HWMCC [7]. It supports the model checking of both invariants and LTL properties in the form $\neg \bigwedge_i (\mathbf{GF} f_i)$ for a set of propositional formulas f_i. Moreover, SIMPLIC3 supports all the simplification techniques described in Sect. 3.7, i.e. temporal decomposition [14], the use of stabilizing constraints [18], and model simplification via ternary simulation [9,13]. Upon successful verification, SIMPLIC3 generates a set of proof obligations (as described in this paper) which can then be checked by any SAT solver. To this extent, the generated proof obligations can be checked internally or dumped in the SMTLIB [1] format to enable for the respective checks by any off-the-shelf SMT solver.

Thus, for SIMPLIC3, we limit the proof generation to the first level described in Remark 1, relying on an existing SAT solver to discharge the proof obligations. For IC3IA, we instead generate also the deduction steps and we developed a prototype proof-checker, built on top of the MATHSAT [16] SMT solver leveraging its Python interface. This implementation reads the deduction proofs generated by IC3IA and check them using purely-syntactic operations. The core of this prototype proof-checker consists of about 500 lines of Python code.

We remark that both IC3IA and SIMPLIC3 can perform the degeneralization with a monitor that can either check the fairness in any order, or check them according to a fixed order, as described previously to enable the proof generation. Instead, the simplifications described in Sect. 3.7 are implemented only in SIMPLIC3.

5.1.2 Benchmark sets

For our evaluation, we have collected a total of 1343 problem instances from different sources.

- 265 safe problem instances (both safety and liveness) from the 2017 edition of HWMCC that were declared safe by at least one solver within the time and resource constraints adopted at the competition.
  This set is composed by 84 safe LTL problems (denoted HWMCC LTL) and by 181 safe invariant problems (denoted HWMCC Inv).
  Several of the instances are very challenging also for state-of-the-art tools; all the properties in the HWMCC LTL family are of the form $\neg \bigwedge_i (\mathbf{GF} f_i)$;

- 519 unsatisfiable LTL formulae from a benchmark set used in previous work on LTL satisfiability checking [41] (denoted Schuppan in the following); this set contains instances of varying difficulty, ranging from trivial to moderately-challenging; several instances are randomly-generated.
- 568 LTL model checking problems resulting from the verification of contracts of a component-based model of an aircraft wheel braking system [10] (denoted WBS in the following); the instances are typically easy, and many are in fact trivial.[5]

5.1.3 Execution parameters

We carried out several analyses with the aim to demonstrate the feasibility of proof generation in practice, and to show that the overhead for proof generation is in many cases negligible. In all cases, we used a timeout of 1200 seconds and a memory limit of 7Gb; all experiments were run on a cluster of Linux machines with 2.10GHz Intel Xeon E5-2620 CPUs and 128Gb of RAM.

For SIMPLIC3 we considered the following configurations. A reference configuration (named **default**) that participated to the HWMCC 2017 competition with all the relevant pre-processing steps enabled, and with no ordering constraint imposed in the degeneralization automaton used to reduce multiple fairness conditions to a single one. We denote with **proofgen** the variant of **default** with proof generation enabled, which in turn activates also the use of a fixed order of the fairness conditions in the degeneralization automaton. Finally, we denote with the **noproofgen** the configuration obtained from **proofgen** by disabling proof generation (thus differing from **default** only in the degeneralization automaton).

For IC3IA, we refer with **IC3IA default** to the original version of [20], which is used as reference. We use **IC3IA proofgen** to denote the version extended with proof generation and with the modified degeneralization. Finally, we denote with **IC3IA noproofgen** the configuration with proof generation disabled.

We executed the two tools on two different sets of benchmarks, namely:

- For the HWMCC instances, we only ran SIMPLIC3, as it consistently outperforms IC3IA on these problems.
- For the Schuppan and WBS instances, we performed the analyses only with IC3IA, since SIMPLIC3 does not have a direct native support for LTL properties (given that the Aiger format supports for a very limited kind of LTL properties).

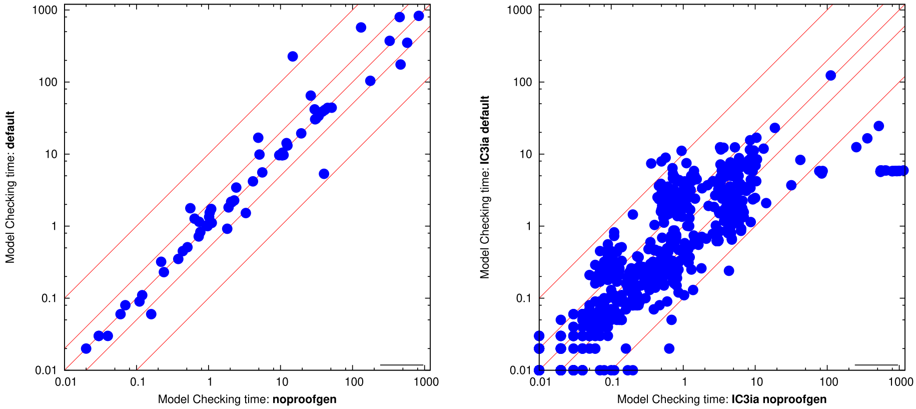
The source code of the extended version of SIMPLIC3, of IC3IA and of the proof checker is available at https://es.fbk.eu/people/griggio/papers/fmsd2019-ltlproofs.tar.bz2, together with all the benchmark instances used in our experimental evaluation, the log files of our results and the scripts to reproduce them.

5.2 Results

5.2.1 Performance impact of modified monitor on LTL model-checking time

We first evaluate the performance impact of the modified monitor for handling multiple fairness constraints with k-liveness, which is the only modification required at model checking time for being able to produce proofs for the LTL case.

[5] This is the case e.g. for some proof obligations generated for components with a trivial assumption.



(a) noproofgen vs **default**
HWMCC LTL

(b) IC3IA **noproofgen** vs IC3IA **default**
WBS+Schuppan

Fig. 2 Performance impact of the modified encoding for handling multiple fairness constraints

The scatter plots in Fig. 2a, reports the comparison of running SIMPLIC3 with the modified monitor that records the fairness conditions in a fixed order (x-axis), i.e. the **noproofgen** configuration, against the results when using the standard monitor that does not impose any order for recording the fairness conditions (y-axis), i.e. **default** configuration.[6] Figure 2b, reports the results of the same comparison but running IC3IA.

The plots show no clear trend for the vast majority of the considered instances, suggesting that the two encodings are essentially equivalent in terms of performance on average.

We remark that the choice of which encoding to use for handling multiple fairness conditions can have an impact on performance for two different, and at least partially conflicting, reasons. On one hand, forcing to record fairness conditions in a fixed order and one at a time has the effect of making model checker consider longer sequences of transitions before it can converge to an inductive invariant (e.g. for IC3 this causes the exploration of a longer sequence of relatively-inductive frames before reaching the fixpoint); on the other hand, however, using the modified monitor allows k-liveness to prove properties with smaller values of k, which in turn might allow the model checker to converge faster. We illustrate both situations with a simple example.

Example 2 Consider the following system $M := \langle V, I, T \rangle$:

$$
\begin{aligned}
&V := \{c, f_1, \ldots, f_{n+1}\} \qquad I := c = 0 \wedge \bigwedge_{i=1}^{n+1} \neg f_i \\
&T := ite(c < n, c' = c + 1, c' = c) \wedge \bigwedge_{i=1}^{n+1} (f_i' \leftrightarrow (c < n))
\end{aligned}
$$

and suppose that $n \geq 1$. M clearly satisfies the property $\varphi := \neg(\bigwedge_{i=1}^{n+1} \mathbf{GF} f_i)$, since all the $f_i's$ will stabilize to false after $n + 1$ transition steps.

When using the monitor that doesn't force an ordering for recording the fairness conditions, the k value needed for a k-liveness proof is n, since all fairness conditions are true for the first n steps. However, when using the modified monitor, $M \models \varphi$ can be proved with $k = 1$.

[6] For this comparison we restrict to the HWMCC LTL benchmarks set where the use of pre-processing has an impact.

Consider instead the following variant of M, in which T is modified as follows:

$$T := ite(c < 1, c' = c + 1, c' = c) \wedge \bigwedge_{i=1}^{n+1} (f_i' \leftrightarrow (c < 1)).$$

In this case, $k = 1$ is enough in both cases. However, the modified monitor will cause IC3 to explore a much deeper sequence of frames before finding an inductive invariant.

5.2.2 Overhead of proof generation

In our second experiment, we evaluated the impact of proof generation on the total execution time. We perform a comparison analyzing the HWMCC Inv, HWMCC LTL, WBS and Schuppan separately reporting results for the two implemented tools.

Figure 3 shows plots comparing the total time taken by SIMPLIC3/IC3IA with proof generation active (x-axis), i.e. configuration **proofgen**, against the time required to model-check the instances without generating a proof certificate (y-axis), i.e. configuration **default**. For the case of the HWMCC problem instances we plot only the results for SIMPLIC3 since it outperforms IC3IA and provides all the main simplifications widely adopted by the state-of-the-art hardware model checkers.

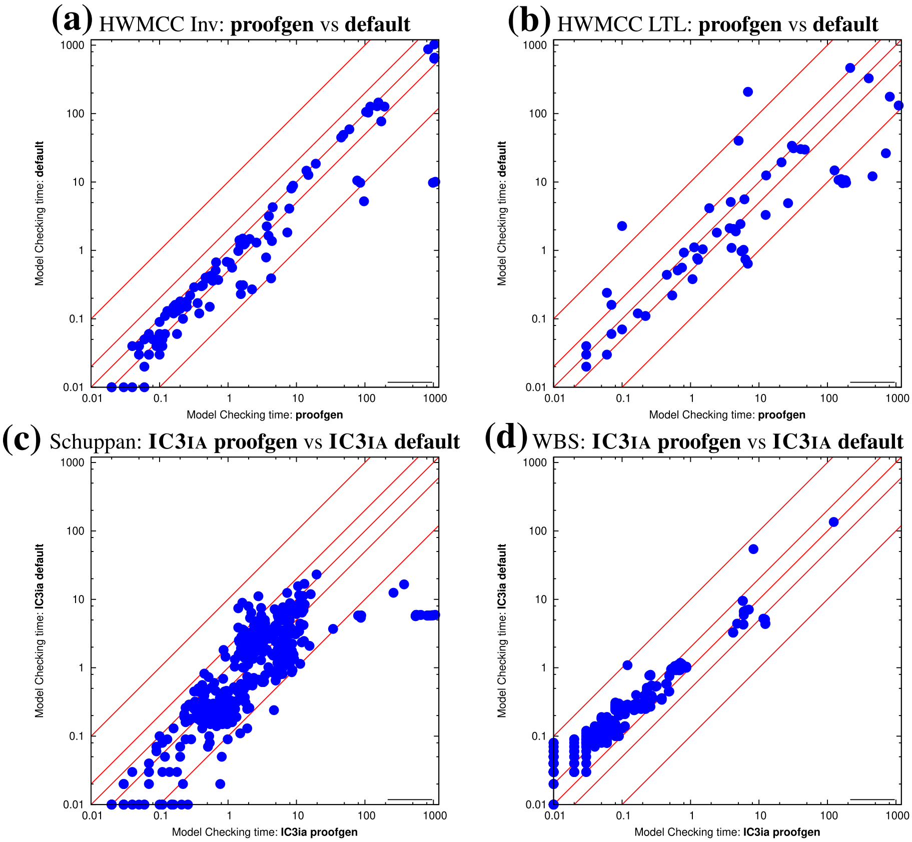


Fig. 3 Performance impact of proof generation

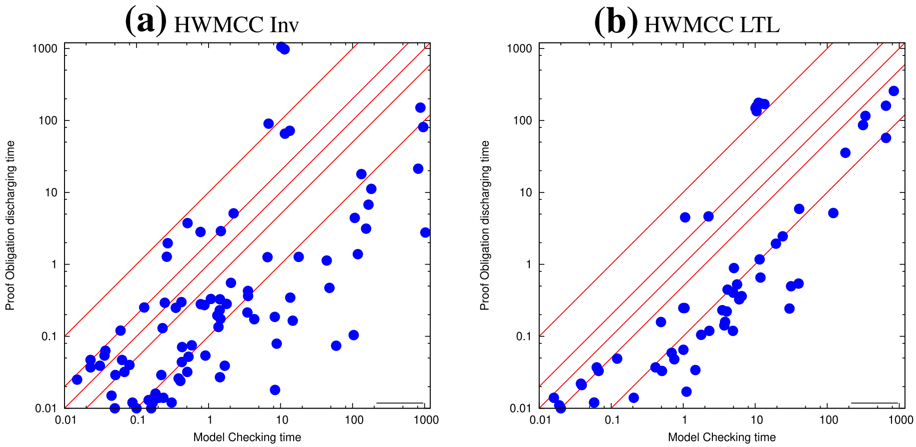


Fig. 4 Performance of discharging proof obligations (y-axis) versus verification time by SIMPLIC3 (x-axis)

In Fig. 3 we analyze the performance of all the considered configurations. Overall, enabling proof generation results in losing 3 instances compared to the original encoding (**default**) for what concerns the HWMCC Inv problem instances. For HWMCC LTL problem instances, **proofgen** solves 5 less instances than **noproofgen**, which in turn solves 4 less instances than **default**. Thus overall, proof generation results in solving 9 less instances than the default configuration for SIMPLIC3. It should be noted, however, that in some cases **proofgen** is faster than **default**. As discussed in the previous section, this is due to the use of a different encoding of multiple fairness conditions, which can sometimes result in better performance for the model checker.

Regarding IC3IA, Fig. 3 shows the results for the Schuppan (Fig. 3c) and WBS (Fig. 3d) instances. Overall, for the Schuppan instances enabling proof generation results in only one lost instance compared to model checking only when using the same encoding for handling multiple fairness conditions; compared to the original encoding, 14 instances are lost. For the WBS benchmarks instead, no instance is lost when enabling proof generation.

5.2.3 Cost of proof checking

We conclude the section presenting some data about the performance of the checks to discharge the proof obligations and of the proof checker.

Figure 4 shows two scatter plots comparing, for each instance, verification by SIMPLIC3 (x-axis) and time for discharging with the MiniSat SAT solver [22] the generated proof obligations (y-axis). From the plots, we can see that in the vast majority of cases, the time for discharging the proof obligations is a small fraction of the model checking time. There are, however, a number of outliers for which proof (obligation) checking is significantly more expensive than model checking. A more in-depth analysis of the results revealed that those cases involved instances which were heavily simplified by our pre-processing techniques (most notably ternary simulation), resulting in problems which were trivial for the model checker (but which were not solvable without pre-processing). It should not be very surprising therefore that checking the proof obligations, which involve a sequence of SAT solver calls, is more expensive than performing a few rounds of ternary simulation followed by variable substitution.

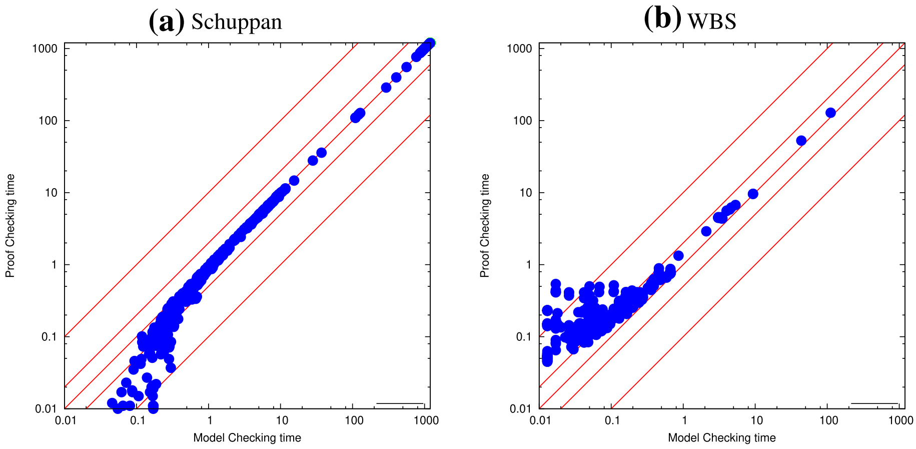


Fig. 5 Performance of proof checking (y-axis) versus verification time by IC3IA (x-axis)

Figure 5 shows two scatter plots comparing, for each instance, verification (x-axis) by IC3IA and proof checking (y-axis) times of the proof generated by IC3IA, whereas Table 1 presents some statistics about the size of the generated proofs. We remark that here the proof checking time is the time to perform the syntactic checks of the proof rules generated by IC3IA. Moreover, while IC3IA is written in C++, the current implementation of the proof checker is a prototype written in Python. We expect that reimplementing the checker in C++ would lead to very significant performance improvements.

6 Conclusions and future work

In this paper, we have presented a sound and complete approach for generating proofs for invariant and LTL model checking problems. The support for LTL leverages on the k-liveness algorithm. Moreover, we also show how to extend the proof generation to consider also the most important pre-processing techniques adopted in modern SAT-based model checkers. The technique can be easily and efficiently implemented, and, as demonstrated by our experimental evaluation, results in proofs/proof obligations that can be checked/discharged by independent tools. Incidentally, we also observe that the same ideas can be applied also to other, non-SAT-based, model checking algorithms, as long as they can generate inductive invariants (e.g., using BDDs).

We see several directions for future work. First, we would like to extend the technique to be applicable also to other SAT-based LTL model checking algorithms, such as the liveness-to-safety transformation of [5] and the FAIR algorithm of [12]. We would also like to investigate generalizations of the approach to infinite-state systems, using model checking algorithms that combine liveness-to-safety, k-liveness and ranking function synthesis [20]. Finally, from the practical perspective, we will enhance our implementation to cover the complete flow and extend the proof generation from the SIMPLIC3 backend to the entire flow of NUXMV [15].

Table 1 Statistics on the size of generated LTL proofs produced by IC3IA, divided by benchmark family.

	HWMCC	*Schuppan*	*WBS*	*All*
Proof size				
Median	31	151	11	31
9th percentile	64	363	31	307
Min	4	4	4	4
Max	78	723	51	723
Proof steps				
Median	125, 858	9601	1215	1590
9th percentile	524, 519	169, 854	17, 054	128, 901
Min	46	5	5	5
Max	6, 377, 311	1, 025, 799	1, 674, 373	6, 377, 311
Temporal steps				
Median	0	1031	9	17
9th percentile	0	15, 073	1	8705
Min	0	0	111	0
Max	0	128, 921	355	128, 921
Fairness conditions				
Median	4	37	2	5
9th percentile	8	90	7	76
Min	1	1	1	1
Max	10	180	12	180
Memory used (MB)				
Median	56.2	19.7	15.5	16.1
9th percentile	1163.8	31.7	29.3	31.3
Min	13.7	12.7	12.8	12.7
Max	1620.0	191.1	793.1	1620.0

Proof size: number of resolution proofs (generated by the SMT solver) for proving $I \wedge I_{\neg\phi} \wedge \mathbf{G}(T \wedge T_{\neg\phi}) \rightarrow \neg(\bigwedge_i \mathbf{GF} f_i)$.
Proof steps: total number of inference rules applied.
Temporal steps: total number of inference rules involving temporal axioms (from $M_{\neg\phi}$).

A Derivation proofs

In this section we detail the derivation of the rules derived from the system defined in [26] and used in the proof generated as described in the previous sections. Each derived rule corresponds to a valid formula, which has been proved valid also with the NUXMV model checker. However, to avoid circularity in the proof generation, we cannot exploit the model checker result and we given an explicit version of the deduction proofs.

A.1 Basic lemmas

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{G}\neg(\psi\wedge\mathbf{G}\phi)]}{\mathbf{G}(\mathbf{G}\phi\rightarrow\neg\psi)}\text{ PROP}}{\mathbf{GG}\phi\rightarrow\mathbf{G}\neg\psi}\text{ G- IMP- DIS}}{\mathbf{G}\phi\rightarrow\mathbf{G}\neg\psi}\text{ G- TRANS}\qquad\mathbf{G}\phi}{\mathbf{G}\neg\psi}\text{ IMP- E}\qquad\mathbf{F}\psi}{\bot}\text{ IMP- E}}{\mathbf{F}(\psi\wedge\mathbf{G}\phi)}\text{ IMP- I}\tag{4}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{GF}(\neg\alpha\vee\neg\beta)]\qquad\dfrac{\mathbf{FG}\alpha\wedge\mathbf{FG}\beta}{\mathbf{FG}\alpha}\text{ AND- EL}}{\mathbf{F}(\mathbf{GF}(\neg\alpha\vee\neg\beta)\wedge\mathbf{G}\alpha)}\text{ (4)}}{\mathbf{F}(\mathbf{GF}(\neg\alpha\vee\neg\beta)\wedge\mathbf{GG}\alpha)}\text{ G- TRANS}}{\mathbf{FG}((\neg\alpha\vee\neg\beta)\wedge\mathbf{G}\alpha)}\text{ G- AND- DIS}}{\mathbf{FG}((\neg\alpha\vee\neg\beta)\wedge\alpha)}\text{ G- EXP}}{\mathbf{GF}\neg\beta}\ res\qquad\dfrac{\mathbf{FG}\alpha\wedge\mathbf{FG}\beta}{\mathbf{FG}\beta}\text{ AND- ER}}{\mathbf{F}(\mathbf{GF}\neg\beta\wedge\mathbf{G}\beta)}\text{ (4)}}{\mathbf{F}(\mathbf{GF}\neg\beta\wedge\mathbf{GG}\beta)}\text{ G- TRANS}}{\mathbf{FG}(\mathbf{F}\neg\beta\wedge\mathbf{G}\beta)}\text{ G- AND- DIS}}{\bot}\text{ IMP- E}}{\mathbf{FG}(\alpha\wedge\beta)}\text{ RAA}\tag{5}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{FG}\neg\alpha]\quad\mathbf{GF}(\alpha\wedge\beta)}{\mathbf{F}(\mathbf{G}\neg\alpha\wedge\mathbf{GF}(\alpha\wedge\beta))}\text{ (4)}}{\mathbf{F}(\mathbf{GG}\neg\alpha\wedge\mathbf{GF}(\alpha\wedge\beta))}\text{ G- TRANS}}{\mathbf{FG}(\mathbf{G}\neg\alpha\wedge\mathbf{F}(\alpha\wedge\beta))}\text{ G- AND- DIS}}{\mathbf{FGF}(\mathbf{G}\neg\alpha\wedge(\alpha\wedge\beta))}\text{ (4)}}{\mathbf{FGF}(\neg\alpha\wedge(\alpha\wedge\beta))}\text{ G- EXP}}{\bot}\text{ PROP}}{\mathbf{GF}\alpha}\text{ RAA}\qquad\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{FG}\neg\beta]\quad\mathbf{GF}(\alpha\wedge\beta)}{\mathbf{F}(\mathbf{G}\neg\beta\wedge\mathbf{GF}(\alpha\wedge\beta))}\text{ (4)}}{\mathbf{F}(\mathbf{GG}\neg\beta\wedge\mathbf{GF}(\alpha\wedge\beta))}\text{ G- TRANS}}{\mathbf{FG}(\mathbf{G}\neg\beta\wedge\mathbf{F}(\alpha\wedge\beta))}\text{ G- AND- DIS}}{\mathbf{FGF}(\mathbf{G}\neg\beta\wedge(\alpha\wedge\beta))}\text{ (4)}}{\mathbf{FGF}(\neg\beta\wedge(\alpha\wedge\beta))}\text{ G- EXP}}{\bot}\text{ PROP}}{\mathbf{GF}\beta}\text{ RAA}}{\mathbf{GF}\alpha\wedge\mathbf{GF}\beta}\text{ RAA}\tag{6}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{FG}(\alpha\rightarrow\beta)\quad[\mathbf{GF}\neg\beta]}{\mathbf{F}(\mathbf{G}(\alpha\rightarrow\beta)\wedge\mathbf{GF}\neg\beta)}\text{ (4)}}{\mathbf{FG}((\alpha\rightarrow\beta)\wedge\neg\beta)}\text{ G- AND- DIS}}{\mathbf{FG}\neg\alpha}\text{ PROP}}{\mathbf{FG}(\alpha\wedge\neg\alpha)}\text{ (5)}}{\bot}\text{ PROP}}{\mathbf{FG}\beta}\text{ RAA}}{\mathbf{FG}\alpha\rightarrow\mathbf{FG}\beta}\text{ IMP- I}\tag{7}$$

A.2 Proofs for k-liveness

The main step to prove the k-liveness derivation is the following deduction:

$$\dfrac{\mathbf{G}\tau \quad \mathbf{GF}\rho \quad \mathbf{F}\alpha_1 \quad \mathbf{G}((\alpha_1 \wedge \tau \wedge \neg\beta) \rightarrow \mathbf{X}\alpha_1) \quad \mathbf{G}((\alpha_1 \wedge \tau \wedge \rho) \rightarrow \mathbf{X}\alpha_2)}{\mathbf{F}\alpha_2}\ \text{KLB}$$

which is derived as follows:

$$\begin{prooftree}
\AxiomC{$[\mathbf{G}(\alpha_1 \rightarrow \neg\rho)]$}
\AxiomC{$\mathbf{G}\tau$}
\AxiomC{$\mathbf{G}((\alpha_1 \wedge \tau \wedge \neg\rho) \rightarrow \mathbf{X}\alpha_1)$}
\RightLabel{G- AND- DIS}
\TrinaryInfC{$\mathbf{G}((\alpha_1 \rightarrow \neg\rho) \wedge \tau \wedge ((\alpha_1 \wedge \tau \wedge \neg\rho) \rightarrow \mathbf{X}\alpha_1))$}
\RightLabel{PROP}
\UnaryInfC{$\mathbf{G}(\alpha_1 \rightarrow \mathbf{X}\alpha_1)$}
\AxiomC{$\mathbf{F}\alpha_1$}
\RightLabel{(4)}
\BinaryInfC{$\mathbf{F}(\mathbf{G}(\alpha_1 \rightarrow \mathbf{X}\alpha_1) \wedge \alpha_1)$}
\RightLabel{IND}
\UnaryInfC{$\mathbf{FG}\alpha_1$}
\AxiomC{$[\mathbf{G}(\alpha_1 \rightarrow \neg\rho)]$}
\RightLabel{G- EXP}
\UnaryInfC{$\mathbf{FG}(\alpha_1 \rightarrow \neg\rho)$}
\RightLabel{(7)}
\UnaryInfC{$\mathbf{FG}\alpha_1 \rightarrow \mathbf{FG}\neg\rho$}
\RightLabel{IMP- E}
\BinaryInfC{$\mathbf{FG}\neg\rho$}
\AxiomC{$\mathbf{GF}\rho$}
\RightLabel{IMP- E}
\BinaryInfC{$\perp$}
\RightLabel{RAA}
\UnaryInfC{$\mathbf{F}(\alpha_1 \wedge \rho)$}
\end{prooftree}$$

$$\begin{prooftree}
\AxiomC{$\mathbf{F}(\alpha_1 \wedge \rho)$}
\AxiomC{$\mathbf{G}\tau$}
\AxiomC{$\mathbf{G}((\alpha_1 \wedge \tau \wedge \rho) \rightarrow \mathbf{X}\alpha_2)$}
\RightLabel{G- AND- DIS}
\BinaryInfC{$\mathbf{G}(\tau \wedge (\alpha_1 \wedge \tau \wedge \rho) \rightarrow \mathbf{X}\alpha_2)$}
\RightLabel{(4)}
\BinaryInfC{$\mathbf{F}(\alpha_1 \wedge \rho \wedge \tau \wedge ((\alpha_1 \wedge \tau \wedge \rho) \rightarrow \mathbf{X}\alpha_2))$}
\RightLabel{PROP}
\UnaryInfC{$\mathbf{FX}\alpha_2$}
\RightLabel{LIN}
\UnaryInfC{$\mathbf{F}\neg\mathbf{X}\neg\alpha_2$}
\RightLabel{X- G- COM}
\UnaryInfC{$\neg\mathbf{XG}\neg\alpha_2$}
\RightLabel{G- EXP}
\UnaryInfC{$\mathbf{F}\alpha_2$}
\end{prooftree}$$

Combining the KLB in a chain, we obtain a full derivation of a proof for the KL[k] rule:

$$\begin{prooftree}
\AxiomC{$\mathbf{F}\alpha_1$}
\AxiomC{$\mathbf{G}((\alpha_1 \wedge \tau \wedge \neg\rho_1) \rightarrow \mathbf{X}\alpha_1)$}
\AxiomC{$\mathbf{G}((\alpha_1 \wedge \tau \wedge \rho_1) \rightarrow \alpha_2)$}
\AxiomC{$[\bigwedge_j \mathbf{GF}\rho_j]$}
\RightLabel{PROP}
\UnaryInfC{$\mathbf{GF}\rho_{j_1}$}
\AxiomC{$[\mathbf{G}\tau]$}
\RightLabel{KLB}
\QuinaryInfC{$\mathbf{F}\alpha_1$}
\RightLabel{KLB}
\dottedLine
\UnaryInfC{$\mathbf{F}\alpha_k$}
\AxiomC{$\mathbf{G}((\alpha_k \wedge \tau \wedge \neg\rho_k) \rightarrow \mathbf{X}\alpha_k)$}
\AxiomC{$\mathbf{G}((\alpha_k \wedge \tau \wedge \rho_k) \rightarrow \perp)$}
\AxiomC{$[\bigwedge_j \mathbf{GF}\rho_j]$}
\RightLabel{PROP}
\UnaryInfC{$\mathbf{GF}\rho_{j_k}$}
\AxiomC{$[\mathbf{G}\tau]$}
\RightLabel{KLB}
\QuinaryInfC{$\perp$}
\RightLabel{IMP- I}
\UnaryInfC{$\bigvee_i \mathbf{FG}\neg\rho_i$}
\RightLabel{IMP- I}
\UnaryInfC{$\mathbf{G}\tau \rightarrow \bigvee_i \mathbf{FG}\neg\rho_i$}
\end{prooftree}$$

A.3 Proofs for stabilizing constraints

The following derived rules are used in the proofs exploiting stabilizing constraints:

$$\dfrac{\mathbf{G}(\tau \rightarrow (\gamma \rightarrow (x \rightarrow x'))) \quad \mathbf{G}\tau \rightarrow (\mathbf{FG}\gamma \vee \psi)}{\mathbf{G}\tau \rightarrow (\mathbf{FG}(\gamma \wedge x = x') \vee \psi)}\ \text{STAB}_1$$

$$\dfrac{\mathbf{G}(\tau \rightarrow (\gamma \rightarrow (x \rightarrow x'))) \quad \mathbf{G}(\tau \rightarrow (\gamma \rightarrow (\neg x \rightarrow \neg f))) \quad \mathbf{G}\tau \rightarrow (\mathbf{FG}\gamma \vee \mathbf{FG}\neg f)}{\mathbf{G}\tau \rightarrow (\mathbf{FG}(\gamma \wedge x) \vee \mathbf{FG}\neg f)}\ \text{STAB}_2$$

These rules are derived as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{GF}(x' \wedge \neg x)]}{\mathbf{GF}x' \wedge \mathbf{GF}\neg x}\;(6)}{\mathbf{GF}x'}\;\text{AND- EL}}{\mathbf{GXF}x}\;\text{X- G- COM}}{\mathbf{XGF}x}\;\text{X- G- COM}}{\mathbf{GF}x}\;\text{AND- EL} \quad \mathbf{FG}(x \rightarrow x')}{\mathbf{F}(\mathbf{GF}x \wedge \mathbf{G}(x \rightarrow x'))}\;(4)}{\mathbf{F}(\mathbf{F}x \wedge \mathbf{G}(x \rightarrow x'))}\;\text{G- EXP}}{\mathbf{FF}(x \wedge \mathbf{G}(x \rightarrow x'))}\;(4)}{\mathbf{FFG}x}\;\text{IND}}{\mathbf{FG}x}\;\text{G- TRANS} \quad \dfrac{\dfrac{[\mathbf{GF}(x' \wedge \neg x)]}{\mathbf{GF}x' \wedge \mathbf{GF}\neg x}\;(6)}{\mathbf{GF}\neg x}\;\text{AND- ER}}{\bot}\;\text{IMP- E}}{\mathbf{FG}(x' \rightarrow x)}\;\text{RAA} \quad \mathbf{FG}(x \rightarrow x')}{\mathbf{FG}((x' \rightarrow x) \wedge (x \rightarrow x'))}\;\text{AND- I}}{\mathbf{FG}(x = x')}\;\text{RAA} \tag{8}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{FG}(x = x') \quad \dfrac{[\mathbf{GF}x \wedge \mathbf{GF}\neg x]}{\mathbf{GF}x}\;\text{AND- EL}}{\mathbf{F}(\mathbf{G}(x = x') \wedge \mathbf{GF}x)}\;(4)}{\mathbf{F}(\mathbf{GG}(x = x') \wedge \mathbf{GF}x)}\;\text{G- TRANS}}{\mathbf{FG}(\mathbf{G}(x = x') \wedge \mathbf{F}x)}\;(4)}{\mathbf{FGF}(\mathbf{G}(x = x') \wedge x)}\;(4)}{\mathbf{FGFG}x}\;\text{IND}}{\mathbf{GFFG}x}\;\text{F- G- COM}}{\mathbf{FFG}x}\;\text{G- EXP}}{\mathbf{FG}x}\;\text{G- TRANS}}{\bot}\;\text{IMP- E} \quad \dfrac{[\mathbf{GF}x \wedge \mathbf{GF}\neg x']}{\mathbf{GF}\neg x}\;\text{AND- ER}}{\mathbf{FG}\neg x \vee \mathbf{FG}x}\;\text{RAA} \tag{9}$$

$$\dfrac{\dfrac{\dfrac{[\mathbf{FG}\gamma] \quad \dfrac{[\mathbf{FG}\gamma] \quad \mathbf{FG}\gamma \rightarrow \mathbf{FG}\beta}{\mathbf{FG}\beta}\;\text{IMP- E}}{\mathbf{FG}\gamma \wedge \mathbf{FG}\beta}\;\text{AND- I}}{\mathbf{FG}(\gamma \wedge \beta)}\;(5)}{\mathbf{FG}\gamma \rightarrow \mathbf{FG}(\gamma \wedge \beta))}\;\text{IMP- I} \tag{10}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{G}(\tau \rightarrow (\gamma \rightarrow (x \rightarrow x')))}{\mathbf{G}\tau \rightarrow \mathbf{G}(\gamma \rightarrow (x \rightarrow x')))}\;\text{G- AND- DIS}}{\mathbf{G}\tau \rightarrow \mathbf{FG}(\gamma \rightarrow (x \rightarrow x')))}\;\text{G- EXP}}{\mathbf{G}\tau \rightarrow (\mathbf{FG}\gamma \rightarrow \mathbf{FG}(x \rightarrow x'))}\;(7)}{\mathbf{G}\tau \rightarrow (\mathbf{FG}\gamma \rightarrow \mathbf{FG}x = x')}\;(8)}{\mathbf{G}\tau \rightarrow (\mathbf{FG}\gamma \rightarrow \mathbf{FG}(\gamma \wedge x = x'))}\;(10) \quad \mathbf{G}\tau \rightarrow (\mathbf{FG}\gamma \vee \psi)}{\mathbf{G}\tau \rightarrow (\mathbf{FG}(\gamma \wedge x = x') \vee \psi)}\;\text{PROP}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{G}(\tau \to (\gamma \to (x \to x')))}{\mathbf{G}\tau \to \mathbf{G}(\gamma \to (x \to x')))}\;\text{G- IMP- DIS}}{\mathbf{G}\tau \to \mathbf{FG}(\gamma \to (x \to x')))}\;\text{G- EXP}}{\mathbf{G}\tau \to (\mathbf{FG}\gamma \to \mathbf{FG}(x \to x'))}\;(7)}{\mathbf{G}\tau \to (\mathbf{FG}\gamma \to (\mathbf{FG}(x = x')))}\;(8)}{\mathbf{G}\tau \to (\mathbf{FG}\gamma \to (\mathbf{FG}x \vee \mathbf{FG}\neg x))}\;(9) \quad \mathbf{G}\tau \to (\mathbf{FG}\gamma \vee \mathbf{FG}\neg f) \quad \dfrac{\dfrac{\dfrac{\mathbf{G}(\tau \to (\gamma \to (\neg x \to \neg f)))}{\mathbf{G}\tau \to \mathbf{G}(\gamma \to (\neg x \to \neg f)))}\;\text{G- IMP- DIS}}{\mathbf{G}\tau \to \mathbf{FG}(\gamma \to (\neg x \to \neg f)))}\;\text{G- EXP}}{\mathbf{G}\tau \to (\mathbf{FG}\gamma \to \mathbf{FG}(\neg x \to \neg f))}\;\mathbf{F}\text{G- IMP- DIS}}{\dfrac{\dfrac{\dfrac{\mathbf{G}\tau \to ((\mathbf{FG}\gamma \wedge \mathbf{FG}x) \vee (\mathbf{FG}\neg x \wedge \mathbf{FG}(\neg x \to \neg f)) \vee \mathbf{FG}\neg f)}{\mathbf{G}\tau \to ((\mathbf{FG}(\gamma \wedge x) \vee (\mathbf{FG}(\neg x \wedge (\neg x \to \neg f))) \vee \mathbf{FG}\neg f)}\;(5)}{\mathbf{G}\tau \to ((\mathbf{FG}(\gamma \wedge x) \vee \mathbf{FG}\neg f \vee \mathbf{FG}\neg f)}\;\text{PROP}}{\mathbf{G}\tau \to (\mathbf{FG}(\gamma \wedge x) \vee \mathbf{FG}\neg f)}\;\text{PROP}}\;\text{PROP}$$

A.4 Proof for simplified transition relation

The following derived rule is used to simplify T with some invariant ξ:

$$\dfrac{\alpha \to \chi \quad \mathbf{G}((\chi \wedge T) \to \chi') \quad \mathbf{G}(\chi \to \xi) \quad (\alpha \wedge \mathbf{G}(\xi \wedge T \wedge \xi')) \to \gamma}{(\alpha \wedge \mathbf{G}(T)) \to \gamma}\;\text{ST}$$

which is derived as follows:

$$\dfrac{\dfrac{\dfrac{[(\alpha \wedge \mathbf{G}(T))] \quad \dfrac{\dfrac{\dfrac{\alpha \to \chi \quad \mathbf{G}((\chi \wedge T) \to \chi')}{\mathbf{G}\chi}\;\text{IND} \quad \dfrac{\mathbf{G}(\chi \to \xi)}{\mathbf{G}\chi \to \mathbf{G}\xi}\;\text{G- IMP- DIS}}{\mathbf{G}\xi}\;\text{IMP- E}}{\mathbf{G}\xi \wedge \mathbf{G}\xi'}\;\text{G- EXP}}{\alpha \wedge \mathbf{G}(\xi \wedge T \wedge \xi')}\;\text{IMP- E} \quad (\alpha \wedge \mathbf{G}(\xi \wedge T \wedge \xi')) \to \gamma}{\gamma}\;\text{IMP- E}}{(\alpha \wedge \mathbf{G}(T)) \to \gamma}\;\text{IMP- I}$$

A.5 Proof for temporal decomposition

The derived rule used with temporal decomposition for liveness properties is the following:

$$\dfrac{(\alpha \wedge \bigwedge_{0 \le i \le k-1} \mathbf{X}^i(\beta)) \to \mathbf{X}^k(\iota) \quad \mathbf{G}((\iota \wedge \mathbf{G}(\beta)) \to \bigvee_i \mathbf{F}\gamma_i)}{(\alpha \wedge \mathbf{G}\beta) \to \bigvee_i \mathbf{F}\gamma_i}\;\text{TD}$$

which is derived as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\alpha \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(\beta)) \to \mathbf{X}^k(\iota)}{(\alpha \wedge \mathbf{G}(\beta)) \to \mathbf{X}^k(\iota)}\;\text{G- EXP} \quad \alpha \wedge \mathbf{G}(\beta)}{\mathbf{X}^k\iota}\;\text{IMP- E} \quad \dfrac{\alpha \wedge \mathbf{G}(\beta)}{\mathbf{G}(\beta)}\;\text{AND- ER}}{\mathbf{X}^k\iota \wedge \mathbf{G}(\beta)}\;\text{AND- I}}{\mathbf{X}^k\iota \wedge \mathbf{X}^k\mathbf{G}(\beta))}\;\text{G- EXP}}{\mathbf{X}^k(\iota \wedge \mathbf{G}(\beta))}\;\text{X- AND- DIS} \quad \dfrac{\mathbf{G}((\iota \wedge \mathbf{G}(\beta) \to \gamma)}{\mathbf{X}^k((\iota \wedge \mathbf{G}(\beta)) \to \gamma)}\;\text{G- EXP}}{\dfrac{\dfrac{\mathbf{X}^k(\iota \wedge \mathbf{G}(\beta)) \wedge \mathbf{X}^k((\iota \wedge \mathbf{G}(\beta)) \to \gamma))}{\mathbf{X}^k(\iota \wedge \mathbf{G}(\beta) \wedge ((\iota \wedge \mathbf{G}(\beta)) \to \gamma))}\;\text{X- AND- DIS}}{\mathbf{X}^k(\gamma)}\;\text{IMP- E}}\;\text{AND- I}$$

$$\dfrac{\dfrac{\dfrac{(\alpha \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(\beta)) \to \mathbf{X}^k(\iota) \quad [\alpha \wedge \mathbf{G}(\beta)] \quad \mathbf{G}((\iota \wedge \mathbf{G}(\beta)) \to \bigvee_i \mathbf{F}\gamma_i)}{\mathbf{X}^k(\bigvee_i \mathbf{F}\gamma_i)}\;(11)}{\bigvee_i \mathbf{F}\gamma_i}\;\text{G- EXP}}{(\alpha \wedge \mathbf{G}(\beta)) \to \bigvee_i \mathbf{F}\gamma_i}\;\text{IMP- I} \tag{11}$$

Similarly for invariant properties, we have the following derived rule:

$$\dfrac{(\alpha \wedge \bigwedge_{0 \le i < k} \mathbf{X}^i(\beta)) \to \mathbf{X}^k(\iota) \quad (\alpha \wedge \bigwedge_{0 \le i \le k-1} \mathbf{X}^i(\beta)) \to (\bigwedge_{0 \le i < k} \mathbf{X}^i\gamma) \quad (\iota \wedge \mathbf{G}\beta) \to \mathbf{G}\gamma}{(\alpha \wedge \mathbf{G}(\beta)) \to \mathbf{G}\gamma}\;\text{TDI}$$

which is derived as follows.

$$\dfrac{\dfrac{\dfrac{(\alpha \wedge \bigwedge_{0\le i<k} \mathbf{X}^i(\beta)) \to \mathbf{X}^k(\iota) \quad [\alpha \wedge \mathbf{G}(\beta)] \quad \mathbf{G}((\iota \wedge \mathbf{G}(\beta)) \to \mathbf{G}\gamma)}{\mathbf{X}^k(\mathbf{G}\gamma)}\,(11) \qquad \dfrac{\dfrac{(\alpha \wedge \bigwedge_{0\le i\le k-1} \mathbf{X}^i(\beta)) \to (\bigwedge_{0\le i<k} \mathbf{X}^i\gamma)}{(\alpha \wedge \mathbf{G}(\beta)) \to (\bigwedge_{0\le i<k} \mathbf{X}^i\gamma)}\,\text{G- EXP} \quad [\alpha \wedge \mathbf{G}(\beta)]}{\bigwedge_{0\le i<k} \mathbf{X}^i\gamma}\,\text{IMP- E}}{\mathbf{G}\gamma}\,\text{G- EXP}}{(\alpha \wedge \mathbf{G}(\beta)) \to \mathbf{G}\gamma}\,\text{IMP- I} \tag{12}$$

A.6 Proofs for LTL encoding

Here we use the equivalence $(\phi \leftrightarrow Exp(\phi)) \leftrightarrow ((\phi \to Exp(\phi)) \wedge (Exp(\phi) \leftrightarrow \phi))$ to match the structure of the AND- EL. Moreover, given the specific construction of $M_{\neg\phi}$, $Enc^{-1}(T_{\neg\phi})$ and $\mathbf{GF}Enc^{-1}(f_i)$ are always valid formulae. Moreover, we leverage that $Enc^{-1}(I_{\neg\phi}) = Exp(\neg\phi) = \neg Exp(\phi)$, and the fact that $Enc^{-1}(T_{\neg\phi})$ is in the form $\bigwedge_\beta \mathbf{X}\beta \leftrightarrow Next(Exp(\beta))$.

$$\dfrac{\dfrac{}{\phi \leftrightarrow Exp(\phi)}\,\text{EXP}}{\neg Enc^{-1}(I_{\neg\phi}) \to \phi}\,\text{AND- EL} \tag{13}$$

$$\dfrac{\dfrac{\dfrac{}{\beta \leftrightarrow Exp(\beta)}\,\text{EXP}}{\mathbf{X}(\beta \leftrightarrow Exp(\beta))}\,\text{X}}{\mathbf{X}\beta \leftrightarrow Next(Exp(\beta))}\,\text{NEXT} \tag{14}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{G}(\beta_1\mathbf{U}\beta_2 \wedge \neg\beta_2)]}{\mathbf{G}(\beta_1\mathbf{U}\beta_2) \wedge \mathbf{G}\neg\beta_2}\,\text{G- AND- DIS}}{\mathbf{G}\beta_1\mathbf{U}\beta_2}\,\text{AND- EL}}{(\beta_1\mathbf{U}\beta_2) \wedge \mathbf{XG}(\beta_1\mathbf{U}\beta_2)}\,\text{G- EXP}}{\beta_1\mathbf{U}\beta_2}\,\text{AND- EL}}{\mathbf{F}\beta_2}\,\text{F} \qquad \dfrac{\dfrac{[\mathbf{G}(\beta_1\mathbf{U}\beta_2 \wedge \neg\beta_2)]}{\mathbf{G}(\beta_1\mathbf{U}\beta_2) \wedge \mathbf{G}\neg\beta_2}\,\text{G- AND- DIS}}{\mathbf{G}\neg\beta_2}\,\text{AND- ER}}{\perp}\,\text{IMP- E}}{\mathbf{F}(\beta_1\mathbf{U}\beta_2 \to \beta_2)}\,\text{RAA}}{\mathbf{GF}(\beta_1\mathbf{U}\beta_2 \to \beta_2)}\,\text{G} \tag{15}$$

$$\dfrac{\dfrac{\dfrac{(I \wedge I_{\neg\phi} \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \to \bigvee_{1\le i\le n} \mathbf{FG}\neg f_i \quad \dfrac{}{\bigwedge_{1\le i\le n} \mathbf{GF}Enc^{-1}(f_i)}\,(15)}{(I \wedge Enc^{-1}(I_{\neg\phi}) \wedge \mathbf{G}T \wedge \mathbf{G}Enc^{-1}(T_{\neg\phi})) \to \perp}\,\text{RES} \quad \dfrac{}{\mathbf{G}Enc^{-1}(T_{\neg\phi})}\,(14)}{(I \wedge \mathbf{G}T) \to \neg Enc^{-1}(I_{\neg\phi})}\,\text{RES} \quad \dfrac{}{\neg Enc^{-1}(I_{\neg\phi}) \to \phi}\,(13)}{(I \wedge \mathbf{G}T) \to \phi}\,\text{RES} \tag{16}$$

A.7 Overall proof for LTL

$$\dfrac{\text{POs for simplified initial condition } \iota \qquad \dfrac{\dfrac{\text{POs for simplified transition condition } \tau \qquad \dfrac{\dfrac{\text{POs for simplified fairness conditions } \gamma \to \neg f_i}{\mathbf{G}\tau \to (\mathbf{FG}\gamma \vee \bigvee \neg f_i)}\,\text{STAB}_1\text{ or STAB}_2 \qquad \dfrac{\dfrac{\text{POs for k-liveness}}{(\iota \wedge \mathbf{G}(\tau)) \to \bigvee_{1\leq i\leq n} \mathbf{FG}(\gamma \to \neg f_i)}\,\text{KL}}{(\iota \wedge \mathbf{G}(\tau)) \to \bigvee_{1\leq i\leq n} (\mathbf{FG}\gamma \to \mathbf{FG}\neg f_i))}\,(7)}{(\iota \wedge \mathbf{G}(\tau) \to \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i}\,\text{PROP}}{(\iota \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \to \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i}\,\text{ST}}{\mathbf{G}((\iota \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \to \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i)}\,\text{G}}{(I \wedge I_{\neg\phi} \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \to \bigvee_{1\leq i\leq n} \mathbf{FG}\neg f_i}\,\text{TD} \qquad \text{Lemmas for LTL encoding}}{(I \wedge \mathbf{G}T) \to \phi}\,(16)$$

A.8 Overall proof for invariant

$$\dfrac{\text{POs for simplified initial condition } \iota \qquad \dfrac{\text{POs for simplified transition condition } \tau \qquad \dfrac{\iota \to \psi \quad (\psi \wedge \tau) \to \psi' \quad \psi \to P}{(\alpha \wedge \mathbf{G}(\tau)) \to \mathbf{G}P}}{(\iota \wedge \mathbf{G}T) \to \mathbf{G}P}\,\text{ST}}{(I \wedge \mathbf{G}T) \to \mathbf{G}P}\,\text{TDI}$$

References

1. Barrett C, Fontaine P, Tinelli C (2017) The SMT-LIB standard: version 2.6. Tech. rep., Department of Computer Science, The University of Iowa. www.SMT-LIB.org
2. Basin D, Bhatt BN, Traytel D (2018) Optimal proofs for linear temporal logic on lasso words . https://www21.in.tum.de/~traytel/papers/expl/expl.pdf
3. Ben-Ari M (1993) Mathematical logic for computer science. Prentice Hall International series in computer science. Prentice Hall
4. Bernasconi A, Menghi C, Spoletini P, Zuck LD, Ghezzi C (2017) From model checking to a temporal proof for partial models. In: SEFM, LNCS, vol. 10469, pp 54–69. Springer
5. Biere A, Artho C, Schuppan V (2002) Liveness checking as safety checking. Electr Notes Theor Comput Sci 66(2):160–177. https://doi.org/10.1016/S1571-0661(04)80410-9
6. Biere A, Cimatti A, Clarke EM, Strichman O, Zhu Y (2003) Bounded model checking. Adv Comput 58:117–148. https://doi.org/10.1016/S0065-2458(03)58003-2(03)58003-2
7. Biere A, van Dijk T, Heljanko K (2017) Hardware model checking competition 2017. In: Proceedings of the 17th conference on formal methods in computer-aided design, FMCAD '17, pp 9. FMCAD Inc, Austin, TX . http://dl.acm.org/citation.cfm?id=3168451.3168458
8. Biere A, Heljanko K, Wieringa S (2011) AIGER 1.9 and beyond. Tech. rep., FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria
9. Bjesse P, Kukula JH (2005) Automatic generalized phase abstraction for formal verification. In: ICCAD, pp 1076–1082. IEEE Computer Society
10. Bozzano M, Cimatti A, Pires AF, Jones D, Kimberly G, Petri T, Robinson R, Tonetta S (2015) Formal design and safety analysis of AIR6110 wheel brake system. In: CAV (1), *LNCS*, vol 9206, pp 518–535. Springer
11. Bradley A (2011) SAT-based model checking without unrolling. In: VMCAI, LNCS, vol 6538, pp 70–87. Springer
12. Bradley AR, Somenzi F, Hassan Z, Zhang Y (2011) An incremental approach to model checking progress properties. In: FMCAD, pp 144–153. FMCAD Inc

13. Case ML, Baumgartner J, Mony H, Kanzelman R (2011).Optimal redundancy removal without fixedpoint computation. In: Bjesse P, Slobodová A (eds) international conference on formal methods in computer-aided design, FMCAD ’11, Austin, TX, USA, October 30–November 02, 2011, pp 101–108. FMCAD Inc. http://dl.acm.org/citation.cfm?id=2157672
14. Case ML, Mony H, Baumgartner J, Kanzelman R (2009) Enhanced verification by temporal decomposition. In: FMCAD. IEEE
15. Cavada R, Cimatti A, Dorigatti M, Griggio A, Mariotti A, Micheli A, Mover S, Roveri M, Tonetta S (2014) The nuXmv symbolic model checker. In: CAV, LNCS, vol 8559, pp 334–342. Springer
16. Cimatti A, Griggio A, Schaafsma BJ, Sebastiani R (2013) The MathSAT5 SMT solver. In: TACAS, LNCS, vol 7795. Springer
17. Cini C, Francalanza A (2015) An LTL proof system for runtime verification. In: TACAS, LNCS, vol 9035, pp 581–595. Springer
18. Claessen K, Sörensson N (2012) A liveness checking algorithm that counts. In: Cabodi G, Singh S (eds) FMCAD, pp 52–59. IEEE
19. Clarke EM, Grumberg O, Hamaguchi K (1997) Another look at LTL model checking. Formal Methods Syst Des 10(1):47–71
20. Daniel J, Cimatti A, Griggio A, Tonetta S, Mover S (2016) Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In: CAV (1), LNCS, vol 9779. Springer
21. Dax C, Hofmann M, Lange M (2006) A proof system for the linear time ¯-calculus. In: FSTTCS, LNCS, vol 4337, pp 273–284. Springer
22. Eén N, Sörensson N (2003) An extensible sat-solver. In: Giunchiglia E, Tacchella A (eds) Theory and applications of satisfiability testing, 6th international conference, SAT 2003. Santa Margherita Ligure, Italy, May 5–8, 2003 Selected Revised Papers, Lecture Notes in Computer Science, vol 2919, pp 502–518. Springer . https://doi.org/10.1007/978-3-540-24605-3_37
23. Emerson EA, Jutla CS, Sistla AP (2001) On model checking for the ¯-calculus and its fragments. Theor Comput Sci 258(1–2):491–522. https://doi.org/10.1016/S0304-3975(00)00034-7
24. Esparza J, Lammich P, Neumann R, Nipkow T, Schimpf A, Smaus J (2014) A fully verified executable LTL model checker. Arch Formal Proofs 2014
25. Fisler K, Kurshan RP (1997) Verifying VHDL designs with COSPAN. In: FHV, LNCS, vol 1287, pp 206–247. Springer
26. Gabbay DM, Pnueli A, Shelah S, Stavi J (1980) On the temporal basis of fairness. In: Conference record of the seventh annual ACM symposium on principles of programming languages, Las Vegas, Nevada, USA, January 1980, pp 163–173 . https://doi.org/10.1145/567446.567462
27. Griggio A, Roveri M (2016) Comparing different variants of the ic3 algorithm for hardware model checking. IEEE Trans CAD Integrated Circuits Syst **35**(6), 1026–1039 . https://doi.org/10.1109/TCAD.2015.2481869
28. Griggio A, Roveri M, Tonetta S (2018) Certifying proofs for LTL model checking. In: 2018 formal methods in computer aided design, FMCAD 2018, Austin, TX, USA, October 30–November 2, 2018, pp 1–9. https://doi.org/10.23919/FMCAD.2018.8603022
29. Hustadt U, Konev B (2003) TRP++2.0: a temporal resolution prover. In: CADE-19, LNCS, vol 2741. Springer
30. Hustadt U, Konev B, Riazanov A, Voronkov A (2004) Temp: a temporal monodic prover. In: IJCAR, LNCS, vol 3097. Springer
31. Kuismin T, Heljanko K (2013) Increasing confidence in liveness model checking results with proofs. In: Bertacco V, Legay A (eds) Hardware and software: verification and testing - 9th international haifa verification conference, HVC 2013, Haifa, Israel, November 5–7, 2013, Proceedings, Lecture Notes in Computer Science, vol 8244, pp 32–43. Springer. https://doi.org/10.1007/978-3-319-03077-7_3
32. Kupferman O, Vardi MY (2005) From complementation to certification. Theor Comput Sci 345(1):83–100. https://doi.org/10.1016/j.tcs.2005.07.021
33. Mebsout A, Tinelli C (2016) Proof certificates for SMT-based model checkers for infinite-state systems. In: 2016 formal methods in computer-aided design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016, pp 117–124 . https://doi.org/10.1109/FMCAD.2016.7886669
34. de Moura LM, Bjørner N (2008) Proofs and refutations, and Z3. In: LPAR workshops, CEUR workshop proceedings, vol 418. CEUR-WS.org
35. Namjoshi KS (2001) Certifying model checkers. In: CAV, LNCS, vol 2102. Springer
36. Peled DA, Pnueli A, Zuck LD (2001) From falsification to verification. In: Hariharan R, Mukund M, Vinay V (eds.) FST TCS 2001, LNCS, vol 2245, pp 292–304. Springer
37. Peled DA, Zuck LD (2001) From model checking to a temporal proof. In: SPIN, LNCS, vol 2057. Springer
38. Pnueli A (1977) The temporal logic of programs. In: FOCS, pp 46–57. 10.1109/SFCS.1977.32

39. Prawitz D (2006) Natural deduction: a proof-theoretical study. Dover Books on Mathematics, Dover Publications
40. RTCA DO-333: Formal Methods Supplement to DO-178C and DO-278A (2011)
41. Schuppan V, Darmawan L (2011) Evaluating LTL satisfiability solvers. In: ATVA, LNCS, vol 6996. Springer
42. Seger CH, Bryant RE (1995) Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods Syst Des 6(2):147–189. https://doi.org/10.1007/BF01383966
43. Vardi MY (1995) An automata-theoretic approach to linear temporal logic. In: Banff Higher Order Workshop, LNCS, vol 1043, pp 238–266. Springer
44. Wagner LG, Mebsout A, Tinelli C, Cofer DD, Slind K (2017) qualification of a model checker for avionics software verification. In: NASA formal methods - 9th international symposium, NFM 2017, Moffett Field, CA, USA, May 16–18, 2017, Proceedings, pp 404–419 . https://doi.org/10.1007/978-3-319-57288-8_29