Certifying Proofs for LTL Model Checking

Alberto Griggio, Marco Roveri and Stefano Tonetta
FBK-irst, Trento, Italy
Email: {griggio,roveri,tonettas}@fbk.eu

***Abstract*—In the context of formal verification, certifying proofs are proofs 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.

In this paper, we solve this issue in the context of Linear-time Temporal Logic. By exploiting the k-liveness algorithm, we show how to extend proof generation capabilities for invariant checking to cover full LTL properties, in a simple and efficient manner, with essentially no overhead for the model checker. We implemented the technique on top of an IC3 engine, and show the feasibility of the approach on a variety of benchmarks.

I. 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. However, verification tools such as model checkers can be quite complex with numerous heuristics and combinations of techniques. The idea of certifying model checking [**?**] 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 [**?**]). 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 this paper, we address the problem of generating a proof in the context of Linear Temporal Logic (LTL) [**?**] model checking. The main obstacle to proof generation is due to the various transformations applied to the problem: model checking is reduced by contradiction to finding a counterexample; LTL formulae are encoded into symbolically-represented automata; multiple fairness conditions resulting from such encoding are reduced to one; liveness is reduced typically to safety.

We propose a sound and complete approach, where the proof is generated from the inductive invariant obtained with the k-liveness algorithm [**?**] 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 resulting approach is simple and efficient, and it can be implemented on top of any state-of-the-art LTL model checker based on the combination of k-liveness with an engine for invariant properties that is capable of producing inductive invariants (e.g., IC3 [**?**]). The proposed approach results in essentially no overhead for the model checker. We have implemented the technique on top of the IC3IA [**?**] engine leveraging on the MATHSAT [**?**] SMT solver as backend. Our experimental results show the feasibility of the approach on a variety of benchmarks taken from the literature, and confirm the small impact of the proof construction on the overall verification process. Finally, we also implemented a prototype proof-checker in Python 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. II we analyze the related works. In Sect. III we provide the needed background. In Sect. IV we discuss the proposed approach to compute proofs for LTL model checking, and in Sect. V we show the results of our experimental evaluation. Finally, in Sect. VI we draw conclusions and outline future work.

II. Related Work and Contributions

Deductive systems for temporal logics have been widely studied [**?**], [**?**], [**?**], [**?**], [**?**]. The idea of certifying model checking is to generate deductive proofs automatically as byproduct of a model checking algorithm.

The closest work to ours is [**?**], 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 [**?**]. The system was implemented (as a prototype) on top of a BDD-based engine (COSPAN [**?**]);

This work has received funding from the EU's H2020 research and innovation programme under the Grant Agreement No. 700665 (project CITADEL) and from the EU's ECSEL JU and Italy's MIUR under the Grant Agreement No. 692474 (project AMASS).

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 [**?**], 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 model checking of linear-time properties include [**?**], [**?**], [**?**], [**?**] and [**?**]. 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 [**?**], [**?**], and [**?**]. 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 [**?**] 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 pseudocode", 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++ [**?**] and TeMP [**?**]. 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 provides the following contributions:

- a proof-generation technique extending a SAT-based LTL model checking algorithm;
- a proof-generation technique for LTL based on symbolic encoding, reverting the internal automata construction;
- a proof-generation technique for LTL validity based on model checking;
- an available effective implementation of proof generation from LTL model checking.

III. 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 $\phi, \psi, f, \alpha, \beta, I, T$, 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.[1]

A. 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 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 $\langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2\rangle$.

B. Invariant Properties

Given a Boolean combination ϕ of predicates, 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) $\psi \rightarrow \phi$.

C. LTL

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

- a variable $v \in V$ 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: $\top := p \vee \neg p$, $\bot := p \wedge \neg p$, $\mathbf{F}\phi := \top\mathbf{U}\phi$, and $\mathbf{G}\phi := \neg\mathbf{F}\neg\phi$.

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:

- $\sigma, i \models v$ iff $\sigma[i] \models v$

[1] 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; we refrain from giving the full details of this in order to save space.

- $\sigma, i \models \phi \wedge \psi$ 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 $M_U \models \phi$.

D. Symbolic LTL Model Checking

The automata-based approach [**?**] to LTL model checking is to build 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 [**?**], 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:

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

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

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

E. 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 Section IV-D for details on the reason). We analyze the impact of this choice experimentally in Section V.

F. 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 [**?**] 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 [**?**] 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 $M \times M_{\neg\varphi} \times M_{deg} \models \neg\mathbf{GF} f$. Its key insight is that, for finite-state systems, this is equivalent to find a k such that f is visited at most k times, which in turn can be reduced to invariant checking.

In [**?**], 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 [**?**] uses IC3 as SAFE and exploits the incrementality of IC3 to solve the sequence of invariant problems in an efficient way.

G. Deduction Systems

A deduction system consists of a set of axiom schemes and inference rules. We use natural deduction [**?**] 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 resolution rule and set of axioms (see e.g. [**?**]):

$$\frac{\alpha_1 \vee \psi \quad \neg\psi \vee \alpha_2}{\alpha_1 \vee \alpha_2}\ \text{RES}$$

$$\frac{}{\neg(a \vee b) \vee a \vee b}\ \text{OR-L} \qquad \frac{}{\neg a \vee (a \vee b)}\ \text{OR-R}$$

$$\frac{}{\neg(a \wedge b) \vee a}\ \text{AND-L} \qquad \frac{}{\neg a \vee \neg b \vee (a \wedge b)}\ \text{AND-R}$$

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.

$$\frac{\begin{array}{c}[\neg\alpha]\\ \vdots \\ \bot\end{array}}{\alpha}\ \text{RAA}$$

As for temporal operators, we use the following generalization inference rule: if a proof of α can be derived without any hypothesis, then we can derive $\mathbf{G}\alpha$, and thus also $\mathbf{X}\alpha$:

$$\frac{\overline{\alpha}}{\mathbf{G}\alpha}\ \text{G} \qquad\qquad \frac{\overline{\alpha}}{\mathbf{X}\alpha}\ \text{X}$$

and we use the following axioms:

$$\frac{}{(a\mathbf{U}b) \leftrightarrow (b \vee (a \wedge \mathbf{X}(a\mathbf{U}b)))}\ \text{UNTIL}$$

$$\frac{}{(a\mathbf{U}(b_1 \vee b_2)) \leftrightarrow ((a\mathbf{U}b_1) \vee (a\mathbf{U}b_2))}\ \text{UNTIL-OR}$$

$$\frac{}{((b_1 \wedge b_2)\mathbf{U}a) \leftrightarrow ((b_1\mathbf{U}a) \wedge (b_2\mathbf{U}a))}\ \text{UNTIL-AND}$$

$$\frac{}{\mathbf{X}\neg a \leftrightarrow \neg\mathbf{X}a}\ \text{NEXT-NOT}$$

$$\frac{}{\mathbf{X}(a \vee b) \leftrightarrow (\mathbf{X}a \vee \mathbf{X}b)}\ \text{NEXT-OR}$$

$$\frac{}{\mathbf{X}(a \wedge b) \leftrightarrow (\mathbf{X}a \wedge \mathbf{X}b)}\ \text{NEXT-AND}$$

The following are abbreviations of multiple applications of the above rules:

- LTL expansion is obtained by multiple application of UNTIL, AND-L, AND-R, OR-L, OR-R:

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

  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 NEXT-NOT, NEXT-AND, NEXT-OR:

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

  where $Next$ is defined recursively as:
  - $Next(v) = \mathbf{X}v$
  - $Next(\phi_1 \wedge \phi_2) = Next(\phi_1) \wedge Next(\phi_2)$
  - $Next(\neg\phi_1) = \neg Next(\phi_1)$
  - $Next(\mathbf{X}\phi_1) = \mathbf{X}\mathbf{X}\phi_1$
  - $Next(\phi_1\mathbf{U}\phi_2) = Next(\phi_2) \vee (Next(\phi_1) \wedge \mathbf{X}\mathbf{X}(\phi_1\mathbf{U}\phi_2))$
- $\wedge$ removal is obtained by combining RES with AND-L:

  $$\frac{\alpha_1 \wedge \alpha_2}{\alpha_1}\ \text{ANDL} \qquad\qquad \frac{\alpha_1 \wedge \alpha_2}{\alpha_2}\ \text{ANDR}$$

- **U** distribution is obtained by combining RES with UNTIL-AND:

  $$\frac{(\alpha_1 \wedge \alpha_2)\mathbf{U}\beta}{\alpha_1\mathbf{U}\beta}\ \text{UAL} \qquad\qquad \frac{(\alpha_1 \wedge \alpha_2)\mathbf{U}\beta}{\alpha_2\mathbf{U}\beta}\ \text{UAR}$$

- **G** distribution is obtained by combining RES with UNTIL-OR:

  $$\frac{\mathbf{G}(\alpha_1 \wedge \alpha_2)}{\mathbf{G}\alpha_1}\ \text{GAL} \qquad\qquad \frac{\mathbf{G}(\alpha_1 \wedge \alpha_2)}{\mathbf{G}\alpha_2}\ \text{GAR}$$

- **G** removal is obtained by combining UNTIL and OR-R:

  $$\frac{\mathbf{G}\alpha}{\alpha}\ \text{GN}$$

H. Resolution Proofs for Invariant Properties

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.

IV. Certifying Proofs for LTL Model Checking

A. Overview of the Approach

As described in Section III, 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 $M \times M_{\neg\phi} \times M_{deg} \times M_c \models_{fin} c \leq k$ from which we can extract an inductive invariant ψ. In order to generate the proof for the original problem, we conceptually reverse this sequence showing how to generate a proof for each step. In Section IV-C, we show how to generate a proof from ψ of $M \times M_{\neg\phi} \times M_{deg} \models \neg\mathbf{GF}f$; in Section IV-D, we show how to generate a proof from ψ of $M \times M_{\neg\phi} \models \neg(\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n)$; finally, in Section IV-E, we show how to generate a proof from ψ of $M \models \phi$.

B. 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 Section III-D) generating a proof of ϕ since the initial and transition conditions of M_U are true.

C. Certifying Proofs for K-Liveness

We consider first the special case of proving $M \models \neg\mathbf{GF}f$, where f is a propositional formula over V. In order to prove $(I \wedge \mathbf{G}T) \rightarrow \neg\mathbf{GF}f$, we use the following inference rule, denoted with KL (see Section IV-F for proof of correctness):

$$\frac{(Pi) \quad (Pn_0) \quad (Pp_0) \quad \ldots \quad (Pn_k) \quad (Pp_k)}{(\iota \wedge \mathbf{G}\tau) \rightarrow \neg\mathbf{GF}\rho}\ \text{KL}$$

where the premises of the rule KL are:

$$\iota \rightarrow \alpha_0 \qquad (Pi)$$
$$\mathbf{G}((\alpha_0 \wedge \tau \wedge \neg\rho) \rightarrow \mathbf{X}\alpha_0) \qquad (Pn_0)$$
$$\mathbf{G}((\alpha_0 \wedge \tau \wedge \rho) \rightarrow \mathbf{X}\alpha_1) \qquad (Pp_0)$$
$$\ldots$$
$$\mathbf{G}((\alpha_k \wedge \tau \wedge \neg\rho) \rightarrow \mathbf{X}\alpha_k) \qquad (Pn_k)$$
$$\mathbf{G}((\alpha_k \wedge \tau \wedge \rho) \rightarrow \bot) \qquad (Pp_k)$$

Intuitively, this means that if we have conditions $\alpha_0, \ldots, \alpha_{k+1}$ such that α_0 is implied by ι, α_i is inductive relative to $\neg\rho$ for $0 \leq i \leq k$, α_{i+1} is implied by $\alpha_i \wedge \rho$ after a transition for $0 \leq i \leq k$, and $\alpha_{k+1} = \bot$, then any path starting from ι can visit ρ finitely many times only.

When checking $M \models \neg\mathbf{GF}f$, we instantiate the rule using $\iota = I$, $\tau = T$, $\rho = f$, and 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 $c \leq k$, then we instantiate the rule KL using $\alpha_i = \psi[c := i]$.

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

$$I \wedge c = 0 \rightarrow \psi$$
$$\psi \wedge T \wedge ite(f, c' = c + 1, c' = c) \rightarrow \psi'$$
$$\psi \rightarrow c \leq k$$

Therefore also the following formulae are valid:

$$I \rightarrow \psi[c := 0] \qquad (p0)$$
$$(\psi[c := 0] \wedge T \wedge \neg f) \rightarrow \mathbf{X}\psi[c := 0] \qquad (p00)$$
$$(\psi[c := 0] \wedge T \wedge f) \rightarrow \mathbf{X}\psi[c := 1] \qquad (p01)$$
$$\ldots$$
$$(\psi[c := k] \wedge T \wedge \neg f) \rightarrow \mathbf{X}\psi[c := k] \qquad (pkk)$$
$$(\psi[c := k] \wedge T \wedge f) \rightarrow \bot \qquad (pk)$$

Note that, the formulae $\alpha_0, \ldots, \alpha_k$ 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 formula $p \in \{p0, p00, p01, ..., pk\}$, we can obtain a resolution proof:

$$\begin{array}{c} p \\ \vdots \\ \bot \end{array}$$

Using rules RAA and G, we obtain a proof for each premise of the rule KL, obtaining a proof of $(I \wedge \mathbf{G}T) \rightarrow \neg\mathbf{GF}f$ in the following form:

$$\dfrac{\dfrac{\dfrac{[\neg p0]}{\vdots}}{\dfrac{\bot}{p0}\,\text{RAA}}}{\mathbf{G}(p0)}\,\text{G} \quad \dfrac{\dfrac{\dfrac{[\neg p00]}{\vdots}}{\dfrac{\bot}{p00}\,\text{RAA}}}{\mathbf{G}(p00)}\,\text{G} \quad \dfrac{\dfrac{\dfrac{[\neg p01]}{\vdots}}{\dfrac{\bot}{p01}\,\text{RAA}}}{\mathbf{G}(p01)}\,\text{G} \quad \ldots \quad \dfrac{\dfrac{\dfrac{[\neg pk]}{\vdots}}{\dfrac{\bot}{pk}\,\text{RAA}}}{\mathbf{G}(pk)}\,\text{G}}{(I \wedge \mathbf{G}T) \rightarrow \neg\mathbf{GF}f}\,\text{KL}$$

D. 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. In order to prove $(I \wedge \mathbf{G}T) \rightarrow \neg(\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n)$, we generalize the rule KL into rule GKL. Rule GKL derives $(\iota \wedge \mathbf{G}\tau) \rightarrow \neg(\mathbf{GF}\rho_1 \wedge \ldots \wedge \mathbf{GF}\rho_n)$ from the following premises:

$$\iota \rightarrow \alpha_{01} \qquad (P_{01})$$
$$\text{for } 0 \leq i \leq k, 1 \leq j \leq n$$
$$\mathbf{G}((\alpha_{ij} \wedge \tau \wedge \neg\rho_j) \rightarrow \mathbf{X}\alpha_{ij}) \qquad (Pn_{ij})$$
$$\text{for } 0 \leq i \leq k, 1 \leq j < n$$
$$\mathbf{G}((\alpha_{ij} \wedge \tau \wedge \rho_j) \rightarrow \mathbf{X}\alpha_{ij'}) \qquad (Pp_{ij})$$
$$\text{for } 0 \leq i < k$$
$$\mathbf{G}((\alpha_{in} \wedge \tau \wedge \rho_n) \rightarrow \mathbf{X}\alpha_{i'1}) \qquad (P_{in})$$
$$\mathbf{G}((\alpha_{kn} \wedge \tau \wedge \rho_n) \rightarrow \bot) \qquad (P_{kn})$$

where $j' = j + 1$ and $i' = i + 1$.

Again, this rule can be instantiated from the inductive invariant generated by k-liveness when using the degeneralization described in Section III-E. 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 $\alpha_{ij} = \psi[c := i, s := j - 1]$ 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):

$$I \rightarrow \psi[c := 0, s := 0] \qquad (p_{01})$$
$$\text{for } 0 \leq i \leq k, 1 \leq j \leq n$$
$$(\psi[c := i, s := j - 1] \wedge T \wedge \neg f_j) \rightarrow \mathbf{X}\psi[c := i, s := j - 1] \qquad (pn_{ij})$$
$$\text{for } 0 \leq i \leq k, 1 \leq j < n$$
$$(\psi[c := i, s := j - 1] \wedge T \wedge f_j) \rightarrow \mathbf{X}\psi[c := i, s := j] \qquad (pp_{ij})$$
$$\text{for } 0 \leq i < k$$
$$(\psi[c := i, s := n-1] \wedge T \wedge f_n) \rightarrow \mathbf{X}\psi[c := i+1, s := 0] \qquad (p_{in})$$
$$(\psi[c := k, s := n - 1] \wedge T \wedge f_n) \rightarrow \bot \qquad (p_{kn})$$

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

E. Certifying Proofs for LTL

We consider here the general case of $M \models \phi$. The procedure described in Section III-F 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 shown in Figure 1, 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 provide a proof for each lemma. 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 $Enc^{-1}(I_{\neg\phi}) = Exp(\neg\phi)$ 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:

$$\dfrac{\dfrac{}{\neg\phi \leftrightarrow Exp(\phi)}\,\text{EXP}}{\neg Enc^{-1}(I_{\neg\phi}) \rightarrow \phi}\,\text{ANDL} \qquad \dfrac{\dfrac{\dfrac{}{\beta \leftrightarrow Exp(\beta)}\,\text{EXP}}{\mathbf{X}(\beta \leftrightarrow Exp(\beta))}\,\text{X}}{\mathbf{X}\beta \leftrightarrow Next(Exp(\beta))}\,\text{XDIS}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{G}(\beta_1\mathbf{U}\beta_2 \wedge \neg\beta_2)]}{\mathbf{G}\beta_1\mathbf{U}\beta_2}\,\text{GAL}}{\beta_1\mathbf{U}\beta_2}\,\text{GN}}{\mathbf{F}\beta_2}\,\text{UAR} \qquad \dfrac{[\mathbf{G}(\beta_1\mathbf{U}\beta_2 \wedge \neg\beta_2)]}{\mathbf{G}\neg\beta_2}\,\text{GAR}}{\bot}\,\text{RES}}{\dfrac{\dfrac{\mathbf{F}(\beta_1\mathbf{U}\beta_2 \rightarrow \beta_2)}{\mathbf{GF}(\beta_1\mathbf{U}\beta_2 \rightarrow \beta_2)}\,\text{G}}{}\,\text{RAA}}$$

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 $\phi = \mathbf{G}(x \rightarrow \mathbf{F}z)$.

ϕ 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 $M_{\neg\phi} = \langle V, I_{\neg\phi}, T_{\neg\phi}\rangle$ 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:

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

M_{deg} and M_c are defined as in Sections III-E and III-F.

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

$$\psi = (Enc(\neg\phi) \wedge (\neg x \vee z \vee v_{\mathbf{XF}z}) \wedge s = 0) \vee (y \wedge \neg z \wedge v_{\mathbf{XG}\neg z} \wedge s = 1)$$

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

Let us consider only a non-trivial case and produce a proof for $TL := (Enc(\neg\phi) \wedge (\neg x \vee z \vee \mathbf{XF}z) \wedge T \wedge T_{\neg\phi} \wedge f_1) \rightarrow (\mathbf{X}y \wedge \mathbf{X}\neg z \wedge \neg\mathbf{XF}z)$.

From the SAT solver we can obtain the following resolution proof for $L = Enc(\neg\phi) \wedge (\neg x \vee Enc(\mathbf{F}z)) \wedge T \wedge T_{\neg\phi} \wedge Enc(f_1) \wedge (\neg y' \vee Enc(\mathbf{F}z)')$:

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

In order to obtain a proof of TL it is sufficient to substitute in the above proof the variables $v_{\mathbf{XF}_1}$ and $v_{\mathbf{XF}z}$ with respectively $\mathbf{XF}_1$ and $\mathbf{XF}z$.

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:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\mathbf{G}\neg f_1]}{\mathbf{GF}_1}\,\text{GAL}}{\mathbf{F}_1}\,\text{GN}}{\mathbf{F}(\neg(x \rightarrow \mathbf{F}z))}\,\text{UAR} \qquad \dfrac{[\mathbf{G}\neg f_1]}{\mathbf{G}\neg(\neg(x \rightarrow \mathbf{F}z))}\,\text{GAR}}{\bot}\,\text{RES}}{\dfrac{\dfrac{\mathbf{F}f_1}{\mathbf{GF}f_1}\,\text{G}}{}\,\text{RAA}}$$

F. Correctness

In the above proofs, we only used the rules defined in Section III-G and the new rule GKL (rule KL is a special case of GKL where $n = 1$).

Let us denote deducibility with this set of rules by $\vdash_{\text{GKL}}$. In the following, we prove soundness and completeness of the proofs.

Theorem 1: If $\vdash_{\text{GKL}} \alpha$ then $\models \alpha$.

Proof. All rules and axioms described above are trivial apart from rule GKL. So, we prove that if σ satisfies (P_{01}), (Pn_{ij}) for $0 \le i \le k, 1 \le j \le n$, (Pp_{ij}) for $0 \le i \le k, 1 \le j < n$, and (P_{in}) for $0 \le i \le k$, then $\sigma \models (\iota \wedge \mathbf{G}\tau) \rightarrow \neg(\mathbf{GF}\rho_1 \wedge \ldots \wedge \mathbf{GF}\rho_n)$ holds. By contradiction, suppose $\sigma \models (\iota \wedge \mathbf{G}\tau) \wedge (\mathbf{GF}\rho_1 \wedge \ldots \wedge \mathbf{GF}\rho_n)$, then σ satisfies each ρ_j infinitely many times. So, let us define $(k+1) \times n$ points t_{ij} such that $t_{00} = 0$ and for all i, j, $0 \le i \le k$, $1 \le j \le n$, $t_{i,j}$ is such that for all h, $t_{i,j-1} \le h < t_{i,j}$ $\sigma, h \not\models \rho_j$ and $\sigma, t_{ij} \models \rho_j$ and, for all i, $0 \le i < k$, $t_{i+1,0} = t_{i,n} + 1$. Due to (P_{01}), $\sigma, t_{00} \models \alpha_{01}$. Due to (Pn_{ij}), for all i, j, $0 \le i \le k$, $1 \le j \le n$, $\sigma, t_{ij} \models \alpha_{i,j}$. Due to (Pp_{ij}), for all i, j, $0 \le i \le k$, $1 \le j < n$, $\sigma, t_{ij} + 1 \models \alpha_{ij'}$. Due to (P_{in}), for all i, $1 \le i \le k$, $\sigma, t_{i0} \models \alpha_{i1}$. Due to (P_{kn}), $\sigma, t_{k,n} \models \bot$, which is a contradiction. Therefore $\sigma \models (\iota \wedge \mathbf{G}\tau) \rightarrow \neg(\mathbf{GF}\rho_1 \wedge \ldots \wedge \mathbf{GF}\rho_n)$. ◇

Corollary 1: If $\vdash_{\text{GKL}} (I \wedge \mathbf{G}T) \rightarrow \phi$ then $M \models \phi$.

Theorem 2: If $M \models \phi$ then $\vdash_{\text{GKL}} (I \wedge \mathbf{G}T) \rightarrow \phi$.

Proof. Let $S := M \times M_{\neg\phi} \times M_{deg} \times M_c$, where $M_{\neg\phi}$ has fairness conditions $f_1, \ldots, f_n$ and accepts the language of $\neg\phi$, M_{deg} has a fairness condition f and accepts the language of $\mathbf{GF}f_1 \wedge \ldots \wedge \mathbf{GF}f_n$, and M_c has a counter that counts the occurrence of f. Since M has finitely many states, if $M \models \phi$, then there exists

$$\dfrac{\dfrac{\dfrac{(I \wedge I_{\neg\phi} \wedge \mathbf{G}(T \wedge T_{\neg\phi})) \rightarrow \neg(\bigwedge_{1\le i\le n} \mathbf{GF} f_i) \quad \bigwedge_{1\le i\le n} \mathbf{GF} f_i}{(I \wedge Enc^{-1}(I_{\neg\phi}) \wedge \mathbf{G}T \wedge \mathbf{G}Enc^{-1}(T_{\neg\phi})) \rightarrow \bot}\,\text{RES} \quad \mathbf{G}Enc^{-1}(T_{\neg\phi})}{(I \wedge \mathbf{G}T) \rightarrow \neg Enc^{-1}(I_{\neg\phi})}\,\text{RES} \quad \neg Enc^{-1}(I_{\neg\phi}) \rightarrow \phi}{(I \wedge \mathbf{G}T) \rightarrow \phi}\,\text{RES}$$

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

k such that $S \models_{fin} c \le k$, and thus there exists an inductive invariant ψ such that $I_S \rightarrow \psi$, $\psi \wedge T_S \rightarrow \psi'$, and $\psi \rightarrow c \le k$. Then, formulae (p_{01}), (pn_{ij}) for $0 \le i \le k, 1 \le j \le n$, (pp_{ij}) for $0 \le i \le k, 1 \le j < n$, and (p_{in}) for $0 \le i \le k$ are all valid. Following the construction shown in Sections IV-D and IV-E, we can generate a proof of $(I \wedge \mathbf{G}T) \rightarrow \phi$. ◇

Corollary 2: If $\models \alpha$ then $\vdash_{\text{GKL}} \alpha$.

V. Experimental Evaluation

We have implemented our proof generation procedure on top of IC3IA, a simple, open-source implementation of IC3 that uses the MathSAT [**?**] SMT solver as backend. The tool supports LTL model checking of both finite and infinite-state systems (using a combination of implicit abstraction and well-founded relations, as described in [**?**]), but currently proof generation is only available for finite-state systems. Upon successful verification, IC3IA generates a proof certificate which can be checked by a simple companion proof checker, using purely-syntactic operations. The resolution proofs for the individual proof obligations, as described in the previous sections, are generated using the off-the-shelf proof-production capabilities provided by MathSAT. The core of the (prototype) proof checker consists of about 500 lines of Python code. The source code of both IC3IA and the proof checker is available at http://es.fbk.eu/people/griggio/papers/fmcad2018-ltlproofs.tar.bz2, together with the benchmark instances used in our experimental evaluation, the log files of our results and the scripts to reproduce them.

For our evaluation, we have collected a total of 1150 instances from three different sources:

- the 63 safe LTL model checking problems from the 2015 hardware model checking competition (denoted HWMCC in the following); all the instances in this set are non-trivial for the model checker, with several that are very challenging also for state-of-the-art tools; all the properties in this family are of the form $\neg \bigwedge_i(\mathbf{GF} f_i)$;[2]
- 519 unsatisfiable LTL formulae from a benchmark set used in previous work on LTL satisfiability checking [**?**] (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 [**?**] (denoted WBS in the following); the instances are typically easy, and many are in fact trivial. [3]

The main objective of our experimental analysis is to demonstrate the feasibility of proof generation in practice. For this, we performed three sets of experiments. 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.

A. Performance impact at model-checking time

In the first experiment, we evaluated 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. The results are shown in the scatter plot of Fig. 2, in which we compare the results of running IC3IA with the modified monitor that records the fairness conditions in a fixed order (x-axis) against the results when running using the standard monitor that doesn't impose any order for recording the fairness conditions (y-axis). The plot shows no clear trend for the vast majority of the instances, suggesting that the two encodings are essentially equivalent in terms of performance on average.

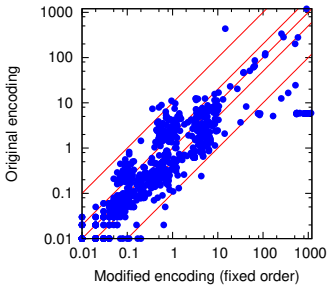

Fig. 2. Performance impact of the modified encoding for handling multiple fairness constraints.

A notable exception is the subset of problems in the `TRP/N12y` group of the Schuppan set: for these instances, the modified monitor results in a significant slowdown (up to two orders of magnitude on some instances), leading to 13 more timeouts. For these instances, it seems that the initial ordering of fairness conditions used by IC3IA, which is based on the unique internal IDs of expressions, is particularly problematic. Randomly shuffling the initial list of fairness conditions greatly mitigates the problem in this case. Although further more in-depth analyses of the correlation between the introduced overhead and the structure of the LTL properties under consideration are out of the scope of the present paper, and therefore left for future work, we can however observe that

[2]The benchmarks are in the Aiger format, which doesn't support arbitrary LTL properties, but only liveness properties of the above form. For most of the benchmarks, the input system therefore already corresponds to $M \times M_{\neg\phi}$ for some LTL property ϕ.

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

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 $M := \langle V, I, T \rangle$:

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

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 $k = 1$.

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.

B. Overhead of proof generation

In our second experiment, we evaluated the impact of proof generation on the total execution time. Fig.3 shows a plot comparing the total time taken by IC3IA (x-axis) against the time required to model-check the instances, without generating a proof certificate (y-axis).

As we can see, the overhead of generating a proof gets progressively smaller as the instances become harder for the model checker. Overall, 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.[4] A summary of the performance of the different configurations of IC3IA is reported in Table I, where the number of successfully solved instances for each benchmark family is shown.

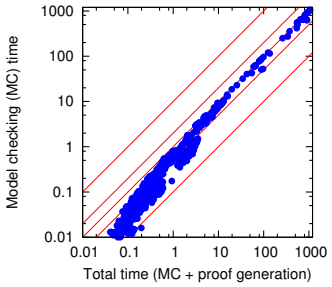

Fig. 3. Performance impact of proof generation.

[4]See the discussion above about this.

TABLE I
SUCCESSFULLY SOLVED INSTANCES BY BENCHMARK FAMILY.

	HWMCC	Schuppan	WBS	All
Model Checking only (original monitor for multi fairness)	31 / 63	495 / 519	568 / 568	1094 / 1150
Model Checking only (modified monitor for multi fairness)	31 / 63	482 / 519	568 / 568	1081 / 1150
MC + Proof Generation	31 / 63	481 / 519	568 / 568	1080 / 1150

TABLE II
STATISTICS ON THE SIZE OF GENERATED PROOFS.

	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	125858	9601	1215	1590
9th percentile	524519	169854	17054	128901
Min	46	5	5	5
Max	6377311	1025799	1674373	6377311
Temporal steps				
Median	0	1031	9	17
9th percentile	0	15073	1	8705
Min	0	0	111	0
Max	0	128921	355	128921
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 $I \wedge GT \rightarrow \neg(\bigwedge_i \mathbf{GF} f_i)$.
Proof steps: total number of inference rules applied.
Temporal steps: total number of inference rules involving temporal axioms (from $M_{\neg\phi}$).

C. Cost of proof checking

We conclude the section presenting some data about the performance of the proof checker.

Fig. 4 shows a scatter plot comparing, for each instance, verification (x-axis) and proof checking (y-axis) times, whereas Table II presents some statistics about the size of the generated proofs. We remark though that 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.

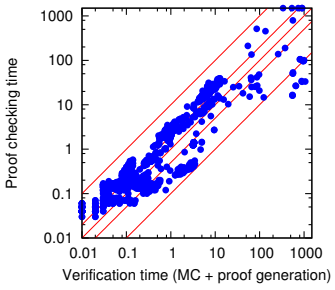

Fig. 4. Performance of proof checking (y-axis) vs verification time (x-axis).

VI. Conclusions and Future Work

In this paper, we have presented a sound and complete approach for generating proofs for LTL model checking problems using the k-liveness algorithm. The technique can be easily and efficiently implemented on top of modern SAT-based model checkers, as demonstrated by our experimental evaluation, and results in proofs that can be efficiently checked (by independent tools) using purely-syntactic rules.

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 [**?**] and the FAIR algorithm of [**?**]. 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 [**?**]. Finally, from the practical perspective, we will enhance our implementation and extend it from the current prototype to a state-of-the art tool like nuXmv [**?**].