Verifying LTL properties of hybrid systems with K-LIVENESS*

Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta

Fondazione Bruno Kessler

Abstract. The verification of liveness properties is an important challenge in the design of real-time and hybrid systems.
In contrast to the verification of safety properties, for which there are several solutions available, there are really few tools that support liveness properties such as general LTL formulas for hybrid systems, even in the case of timed automata.
In the context of finite-state model checking, K-Liveness is a recently proposed algorithm that tackles the problem by proving that an accepting condition can be visited at most K times. K-Liveness has shown to be very efficient, thanks also to its tight integration with IC3, a very efficient technique for safety verification.
Unfortunately, the approach is neither complete nor effective (even for simple properties) in the case of infinite-state systems with continuous time.
In this paper, we extend K-Liveness to deal with LTL for hybrid systems. On the theoretical side, we show how to extend the reduction from LTL to the reachability of an accepting condition in order to make the algorithm work with continuous time. In particular, we prove that the new reduction is complete for a class of rectangular hybrid automata, in the sense that the LTL property holds if and only if there exists K such that the accepting condition is visited at most K times. On the practical side, we present an efficient integration of K-Liveness in an SMT-version of IC3, and demonstrate its effectiveness on several benchmarks.

1 Introduction

Hybrid systems are an ideal modeling paradigm to represent embedded systems since they combine discrete behaviors, useful to model protocols and control components, with continuous behaviors, useful to model physical entities such as time, temperature, speed, etc. Hybrid systems are becoming increasingly interesting in order to apply formal methods to the design of safety-critical systems in different domains such as aerospace, railways, and automotive.

The verification of liveness properties on hybrid systems is very challenging because infinite paths must be considered. In particular, we focus on Linear-time Temporal Logic (LTL), which is suitable to represent many safety and liveness properties. The standard approach to verify if a model M satisfies an LTL property ϕ builds the automaton $M_{\neg\phi}$ equivalent to the negation of ϕ and check if the accepting state of the product $M \times M_{\neg\phi}$ can be visited infinitely often.

* This work was carried out within the D-MILS project, which is partially funded under the European Commission's Seventh Framework Programme (FP7).

In the context of finite-state model checking, many efficient algorithms reduce liveness properties to one or more safety properties. For example, K-LIVENESS is a recently proposed technique that proves that the accepting state is visited finitely many times by checking that it is visited at most K times for increasing values of K. The latter can be easily reduced to a reachability problem. K-Liveness has shown to be very efficient, thanks also to its tight integration with IC3, probably the current most effective technique for safety verification. Unfortunately, the approach is neither complete nor effective (even for simple properties) in the case of infinite-state systems with continuous time.

The main problem of techniques based on the reduction to safety is that they rely for soundness or completeness on the existence of a lasso-shape counterexample, but in the case of infinite-state systems such as hybrid systems, there may be infinite traces that do not correspond to any lasso-shape fair path. Moreover, the model may include Zeno paths where time converges, which must be excluded when checking the liveness properties. Techniques based on abstraction refinement can prove that a property holds, but in general the refinement is not guaranteed to converge.

In this paper, we provide a new method that, by forcing the progress of time beyond symbolic bounds, links the number of iterations of K-LIVENESS to the time elapsed in the counterexamples, rather than to the number of transitions. We prove that the reduction is complete for initialized Rectangular Hybrid Automata (RHA) with bounded non-determinism even in the presence of parameters. We implemented the techniques on top of HYCOMP [1], a tool for the verification of hybrid systems. The verification of reachability is based on an SMT version of IC3 that integrates predicate abstraction in an efficient way. An experimental evaluation demonstrates the efficiency of the approach on several benchmarks. To the best of our knowledge, this is the first effective tool that verifies general LTL properties on Hybrid Automata.

The paper is organized as follows: Section 2 presents some basic notations on RHA, LTL, and SMT-based techniques to verify hybrid systems; we also give a brief overview of IC3 and K-LIVENESS; in Section 3, we present the new approach to the LTL verification of hybrid systems; in Section 4, we overview the related work; in Section 5, we describe the implementation, the experimental evaluation, and we present the results; finally, in Section 6 we draw some conclusions and discuss future directions.

2 Background

2.1 Hybrid and Timed Automata

Hybrid systems have a discrete part, which ranges over the nodes of a graph, and a continuous part, which ranges over an Euclidian space $\mathbb{R}^n$. Although the approach presented in this paper can be applied to any hybrid system that can be encoded into a symbolic transition system, the theoretical results are restricted to the parametric version of *Rectangular Hybrid Automata* [2]. A Parametric Rectangular Hybrid Automaton (PRHA) is a tuple $H = \langle P, Q, Q_0, E, X, \mathit{flow}, \mathit{init}, \mathit{inv}, \mathit{jump}, \mathit{guard}, \mathit{update} \rangle$ where:

- P is a finite set of parameters,
- Q is the (possibly infinite) set of locations,

– $Q_0 \subseteq Q$ is the (possibly infinite) set of initial locations,
– $E \subseteq Q \times Q$ is the (possibly infinite) set of discrete transitions,
– X is the finite set of continuous variables,
– *flow* : $Q \to X \to \mathcal{R}$ is the flow function,
– *init* : $Q \to X \to \mathcal{R}(P)$ is the initial function,
– *inv* : $Q \to X \to \mathcal{R}(P)$ is the invariant function,
– *jump* : $E \to 2^X$ is the jump function,
– *guard* : $E \to X \to \mathcal{R}(P)$ is the guard function,
– *update* : $E \to X \to \mathcal{R}(P)$ is the update function,

where $\mathcal{R}$ is the set of (possibly unbounded) real intervals and $\mathcal{R}(P)$ represents the set of parametric intervals, whose endpoints are either a constant, or $\pm\infty$, or a parameter in P (e,g., $[0,0], (1,+\infty), (-\infty,p]$). We can see parametric intervals as function from an evaluation of the parameters to the intervals of $\mathbb{R}$. So, if c is an assignment to the parameters in P and $I \in \mathcal{R}(P)$, then $I(c)$ is a real interval.

A Rectangular Hybrid Automaton (RHA) is simply a PRHA with $P = \emptyset$. A (Parametric) Timed Automaton (TA) is an RHA (resp. PHRA) such that, for all $q \in Q$, for all $x \in X$, *flow*$(q)(x) = [1,1]$, *init*$(q)(x) = [0,0]$, and for all $e \in E$, for all $x \in X$, *update*$(e)(x) = [0,0]$. A (P)RHA H is *initialized* iff for every edge $\langle q, q' \rangle \in E$, for all $x \in X$, if *flow*$(q)(x) \neq$ *flow*$(q')(x)$, then $x \in$ *jump*$(\langle q, q' \rangle)$. H has bounded nondeterminism iff for all $x \in X$, for all $q \in Q$, for all $e \in E$, *init*$(q)(x)$, *flow*$(q)(x)$, and *update*$(e)(x)$ are bounded.

As in [3], we use a variable $pc \notin X \cup P$ as a control variable that ranges over the set Q of locations (properly encoded in $\mathbb{R}$). Moreover, we use a variable $time$ to represent the elapsing time and let $V_H = \{time, pc\} \cup P \cup X$. A state is an assignment to V_H, i.e., a function $V_H \to \mathbb{R}$. We can see a state also as a tuple $\langle q, s, c, t \rangle$ where $q \in Q$, s is an assignment to X, c is an assignment to P, and t is an assignment to $time$. A path of a PRHA H is a sequence of states $\langle q_0, s_0, c_0, t_0 \rangle, \langle q_1, s_1, c_1, t_1 \rangle, \ldots$ such that:

– for all $i, j \geq 0$, $c_i = c_j = c$, for some c;
– $t_0 = 0$ and for all $i \geq 0$, $t_i \leq t_{i+1}$; let $\delta_i = t_{i+1} - t_i$;
– for all $i \geq 0$, if $\delta_i > 0$, then $q_{i-1} = q_i$ and, for all $x \in X$, $\frac{s_{i+1}(x) - s_i(x)}{\delta_i} \in$ *flow*$(q_i)(x)$ (note that, in more general classes of hybrid automata, this would require a condition on all time points);
– $q_0 \in Q_0$ and, for all $x \in X$, $s_0(x) \in$ *init*$(q_0)(x)(c)$;
– for all $i \geq 0$, if $\delta_i = 0$, then
  • $\langle q_i, q_{i+1} \rangle \in E$,
  • for all $x \notin$ *jump*$(\langle q_i, q_{i+1} \rangle)$, $s_{i+1}(x) = s_i(x)$;
  • for all $x \in X$, $s_i(x) \in$ *guard*$(\langle q_i, q_{i+1} \rangle)(x)(c)$;
  • for all $x \in$ *jump*$(\langle q_i, q_{i+1} \rangle)$, $s_{i+1}(x) \in$ *update*$(\langle q_i, q_{i+1} \rangle)(x)(c)$;
– for all $i \geq 0$, for all $x \in X$, $s_i(x) \in$ *inv*$(q_i)(x)(c)$.

Given a sequence of states $\sigma = \sigma_0, \sigma_1, \ldots$, we denote with $\sigma[i]$ the $i+1$-th state σ_i and with σ^i the suffix sequence starting from the $\sigma[i]$.

A path whose sequence $t_0, t_1, \ldots$ of time points does not diverge is called *Zeno path* (non-Zeno otherwise). A state s is *Zeno* or *time-locking* iff there is no non-Zeno path starting from s. A state s is reachable iff there exists a non-Zeno path σ such that $\sigma[i] = s$ for some $i \geq 0$.

2.2 LTL

We use Linear-time Temporal Logic (LTL) [4] to specify properties on a PRHA H. The atomic formulas *Atoms* are predicates over the variables V_H. Besides the Boolean connectives, LTL uses the temporal operators $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"). Formally,

- a predicate $a \in \mathit{Atoms}$ is an LTL formula,
- if ϕ_1 and ϕ_2 are LTL formulas, then $\neg\phi_1$, and $\phi_1 \wedge \phi_2$ are LTL formulas,
- if ϕ_1 and ϕ_2 are LTL formulas, then $\mathbf{X}\phi_1$ and $\phi_1\mathbf{U}\phi_2$ are LTL formulas.

We use the standard abbreviations: $\top := p \vee \neg p$, $\bot := p \wedge \neg p$, $\mathbf{F}\phi := \top\mathbf{U}\phi$, $\mathbf{G}\phi := \neg\mathbf{F}\neg\phi$, and $\phi_1\mathbf{R}\phi_2 := \neg(\neg\phi_1\mathbf{U}\neg\phi_2)$.

Given an LTL formula ϕ and a sequence σ of states of H, we define $\sigma \models \phi$, i.e., that the path σ satisfies the formula ϕ, as follows:

- $\sigma \models a$ iff $\sigma[0] \models a$
- $\sigma \models \neg\phi$ iff $\sigma \not\models \phi$
- $\sigma \models \phi\mathbf{U}\psi$ iff for some $j \geq 0$, $\sigma^j \models \psi$ and for all $0 \leq k < j$, $\sigma^k \models \phi$.
- $\sigma \models \phi \wedge \psi$ iff $\sigma \models \phi$ and $\sigma \models \psi$
- $\sigma \models \mathbf{X}\phi$ iff $\sigma^1 \models \phi$

Given a PRHA H and an LTL formula ϕ over V_H, we focus on the model checking problem of finding if, for all non-Zeno paths σ of H, $\sigma \models \phi$.

Note that, although the predicates can contain references to the $time$ variable, the logic is interpreted over discrete sequences of states.

The problem is in general undecidable for PRHA and decidable for some fragments such as initialized RHA with bounded non-determinism [2].

2.3 Transition Systems

A *transition system* M is a tuple $M = \langle V, I, T\rangle$ where V is a set of (state) variables, $I(V)$ is a formula representing the initial states, and $T(V, V')$ is a formula representing the transitions. In this paper, we shall deal with *linear rational arithmetic* formulas, that is, Boolean combinations of propositional variables and linear inequalities over rational variables. A *state* of M is an assignment to the variables V. We denote with Σ_V the set of states. 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 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 $\langle V_1 \cup V_2, I_1 \wedge I_2, T_1 \wedge T_2\rangle$.

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, for all i, $0 \leq i \leq k$, $s_i \models \phi$.

Given a 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, $\sigma \models \phi$.

The automata-based approach [5] to LTL model checking is to build a transition system $M_{\neg\phi}$ with a fairness condition $f_{\neg\phi}$ such that $M \models \phi$ iff $M \times M_{\neg\phi} \models \mathbf{FG}\neg f_{\neg\phi}$. This reduces to finding a counterexample as a fair path, i.e., a path of the system that visits the fairness condition $f_{\neg\phi}$ infinitely many times. In case of finite-state systems, if the property fails there is always a counterexample in a lasso-shape, i.e., formed by a prefix and a loop.

2.4 SMT-based Verification of Reachability for PRHA

Given a PRHA H, we encode H into a transition system M_H in order to apply SMT-based verification techniques for infinite-state systems. Such kind of encoding has been widely used in the literature (e.g., [6, 7]). $M_H = \langle V_H, I_H, T_H \rangle$ is defined as follows:

- $I_H \stackrel{\text{def}}{=} (time = 0) \wedge \bigwedge_{q \in Q} \bigwedge_{x \in X} x \in \mathit{init}(q)(x) \wedge x \in \mathit{inv}(q)(x)$.
- $T_H \stackrel{\text{def}}{=} (\text{TIMED} \vee \text{UNTIMED}) \wedge \bigwedge_{q \in Q} \mathit{inv}(q)(X) \wedge \mathit{inv}(q)(X') \wedge \bigwedge_{p \in P} p' = p$, where

  $\text{UNTIMED} \stackrel{\text{def}}{=} \quad \delta = 0 \wedge \bigwedge_{\langle q,q' \rangle \in E} (pc = q \wedge pc = q') \wedge \bigwedge_{x \in X} \mathit{guard}(q)(x) \wedge$
  $\bigwedge_{x \notin \mathit{jump}(\langle q,q' \rangle)} x' = x \wedge \bigwedge_{x \in \mathit{jump}(\langle q,q' \rangle)} x' \in \mathit{update}(q)(x)$

  $\text{TIMED} \stackrel{\text{def}}{=} \quad \delta > 0 \wedge pc' = pc \wedge \bigwedge_{q \in Q} \bigwedge_{x \in X} (pc = q \rightarrow (x' - x) \in \delta \cdot \mathit{flow}(q)(x))$

  $\delta \stackrel{\text{def}}{=} time' - time$.

There is a one-to-one mapping between the states of H and those of M_H, and also between the paths of H and those of M_H. We say that a path of M_H is Zeno [non-Zeno] iff the sequence of assignments to $time$ does not diverge [resp., diverges].

Given a PRHA H, assuming that H does not have Zeno states, a state s is reachable in H iff $M_H \not\models_{fin} \neg s$ (where s is seen as a formula).

2.5 IC3 and K-LIVENESS

SAT-based algorithms take in input a propositional (with Boolean variables) transition system and a property, and try to solve the verification problem with a series of satisfiability queries. These algorithms can be naturally lifted to SMT in order to tackle the verification of infinite-state systems.

IC3 [8] is a SAT-based algorithm for the verification of invariant properties of transition systems. It builds an over-approximation of the reachable state space, using clauses obtained by generalization while disproving candidate counterexamples.

We recently presented in [9] a novel approach to lift IC3 to the SMT case, which is able to deal with infinite-state systems by means of a tight integration with *predicate abstraction* (PA) [10]. The approach leverages *Implicit Abstraction* (IA) [11], which allows to express abstract transitions without computing explicitly the abstract system, and is fully incremental with respect to the addition of new predicates.

In this paper, we focus on K-LIVENESS [12], an algorithm recently proposed to reduce liveness (and so also LTL verification) to a sequence of invariant checking. Differently from other reductions (such as [13]), it lifts naturally to infinite-state systems without requiring counterexamples to be in a lasso-shape form. K-LIVENESS uses a standard approach to reduce LTL verification for proving that a certain signal f is eventually never visited ($\mathbf{FG}\neg f$). The key insight of K-LIVENESS 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.

Given a transition system M, a Boolean combination of predicates ϕ, and a positive integer K, for every finite path σ of M, let $\sigma \models_{fin} \sharp(\phi) \leq K$ iff the size of the set $\{i \mid \sigma[i] \models \phi\}$ is less or equal to K. In [12], it is proved that, for finite-state systems, $M \models \mathbf{FG}\neg f$ iff there exists K such that $M \models_{fin} \sharp(f) \leq K$. The last check can be reduced to an invariant checking problem. K-LIVENESS is therefore a simple loop that increases K at every iteration and calls a subroutine SAFE to check the invariant. In

particular, the implementation in [12] uses IC3 as SAFE and exploits the incrementality of IC3 to solve the sequence of invariant problems in an efficient way.

3 SMT-Based Verification of LTL for PRHA

3.1 K-LIVENESS for Hybrid Automata

K-LIVENESS is not complete for infinite-state systems, because even if the property holds, the system may visit the fairness condition an unbounded number of times. Consider for example a system with an integer counter and a parameter p such that the counter is used to count the number of times the condition f is visited and once the counter reaches the value of p, the condition is no more visited. This system satisfies $\mathbf{FG}\neg f$ because for any value of p, f is visited at most p times. However, K-LIVENESS will obtain a counterexample to the safety property $\sharp(f) \leq K$ for every K, by setting p to K.

Similarly, K-LIVENESS does not work on the transition system representing a TA. In particular, a fair Zeno path forbids K-LIVENESS to prove the property: for every K, the fairness is visited more than K times, but in a finite amount of (real) time. Removing Zeno paths by adding an automaton to force progress is not sufficient for PTA and in general hybrid systems. In fact, in these systems a finite amount of time can be bounded by a parameter or a variable that is dynamically set. Therefore, in some cases, there is no K to bound the occurrences of the fairness, although there is no fair non-Zeno path.

In the following, we show how we make K-LIVENESS work on hybrid automata. The goal is to provide a method so that K-LIVENESS checks if there is a bound on the number of times the fairness is visited along a diverging sequence of time points. The essential point is to use a symbolic expression β based on the automaton structure to force a minimum distance between two fair time points. We use an additional transition system Z_β, with a condition f_Z, to reduce the problem of proving that $H \models \phi$ to proving that $M_H \times M_{\neg\phi} \times Z_\beta \models \mathbf{FG}\neg f_Z$. In Section 3.2, we prove that the two problems are equivalent for any positive β. In Section 3.3, we define β so that K-LIVENESS is not deemed to diverge and, on the contrary, must converge for some class of automata.

3.2 Linking the fairness to time progress

In this section, we define the transition system Z_β that is later used to make K-LIVENESS converge. We first define a simpler version Z_B that works only for timed automata.

Consider the fair transition system $M = M_H \times M_{\neg\phi}$ resulting from the product of the encoding of an PRHA H and of the negation of the property ϕ. Let f be the fairness condition of M. We build a new transition system $Z_B(f, time)$ that filters the occurrences of f along a time sequence where $time$ values are distant more than B time units. $Z_B(f, time)$ is depicted in Figure 1. It has two locations (represented by a Boolean variable l) and a local real variable t_0. The initial condition is $l = 0$. The fairness condition f_Z is $l = 1$. The system moves or remains in $l = 0$ keeping t_0 unchanged. It moves or remains in $l = 1$ if f is true and $time \geq t_0 + B$ and sets t_0 to $time$.

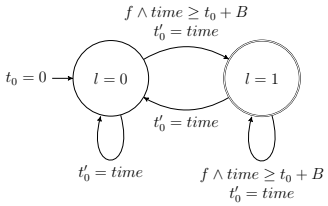


Fig. 1. Monitor $Z_B(f, time)$.

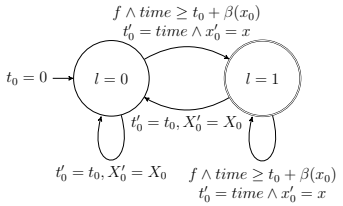


Fig. 2. Monitor $Z_\beta(f, time, X)$.

We reduce the problem of checking whether ϕ holds in H to checking that the fairness condition f_Z cannot be true infinitely often in $M_H \times M_{\neg\phi} \times Z_B$, i.e. $M_H \times M_{\neg\phi} \times Z_B \models \mathbf{FG}\neg f_Z$.

Theorem 1. *If $B > 0$, $H \models \phi$ iff $M_H \times M_{\neg\phi} \times Z_B \models \mathbf{FG}\neg f_Z$.*

Proof. If there exists a non-Zeno path π of M_H that violates ϕ, then there exists a fair path π' of $M_{\neg\phi}$ so that $\pi \times \pi'$ is a fair non-Zeno path of $M \times M_{\neg\phi}$. We can build a matching path π_Z of Z_B. In fact, if the path $\pi_Z[i]$ is in $l = 0$, there are infinitely many $j \geq i$ such that $\pi'(j) \models f_{\neg\phi}$ and we can pick one moving to $l = 1$ with $time(j) > time(i) + B$ since π is non-Zeno.

If a path π of $M \times M_{\neg\phi} \times Z_B$ visits f_Z infinitely often, then for infinitely many points $i \geq 0$, $\pi^i \models f_{\neg\phi}$ and there exists $j \geq i$ such that $\pi^j \models f_{\neg\phi} \wedge time > t_0 + B$. Therefore the projection of π over M_H corresponds to a fair non-Zeno path of H violating ϕ. □

We generalize the construction of Z_B considering as bound on time a function β over some continuous variables of the model. The new monitor is $Z_\beta(f, time, X)$ shown in Figure 2. It has a local variable x_0 for every variable x occurring in β. X_0 is the set of such variables. Now, when t_0 is set to $time$, we set also x_0 to x and this value is kept until moving to $l = 1$. The condition on time is now $time > t_0 + \beta(X_0)$. It is easy to see that we can still prove that if $\beta(X)$ is always positive, then $H \models \phi$ iff $M_H \times M_{\neg\phi} \times Z_\beta \models \mathbf{FG}\neg f_Z$.

We say that the reduction is *complete* for K-LIVENESS for a certain class $\mathcal{H}$ of automata iff for every $H \in \mathcal{H}$ there exists β_H such that $H \models \phi$ iff there exists K such that $M \times M_{\neg\phi} \times Z_{\beta_H} \models_{fin} \sharp(f_Z) \leq K$. Thus, if $H \models \phi$, and the reduction is complete, and the subroutine SAFE terminates at every call, then K-LIVENESS also terminates proving the property.

3.3 The K-ZENO algorithm

The K-ZENO algorithm is a simple extension of K-LIVENESS which, given the problem $H \models \phi$, builds $M = M_H \times M_{\neg\phi} \times Z_\beta$ and calls K-LIVENESS with inputs M and f_Z. As K-LIVENESS, either K-ZENO proves that the property holds or diverges increasing K up to a certain bound. The crucial part is the choice of β, because the completeness of the reduction depends on β. Note that the reduction may be complete, but the completeness of K-ZENO still depends on the completeness of the SAFE algorithm.

As for TAs, we take as β the maximum among the constants of the model and 1. For example, consider the TA in figure 3 (it is actually a compact representation of the TA where $loc1$ is split into two locations corresponding to $b = \top$ and $b = \bot$). It represents an unbounded number of switches of b within 1 time unit. The model satisfies the property $\mathbf{FG}pc = loc2$. Taking $\beta = 1$, K-ZENO proves the property with $K = 1$. In fact, starting from the location $loc1$, after 1 time unit, the automaton cannot reach $loc1$ anymore. For PTAs, we consider as β the maximum among the parameters, the constants of the model and 1.

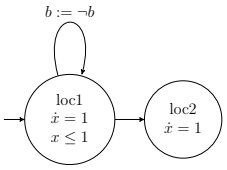


Fig. 3. Example of TA.

We generalize the above idea to consider PRHA with bounded non-determinism. We also assume an endpoint of a flow interval is 0, it cannot be open (must me included in the interval). Guards and invariants of PRHA are conjunctions of inequalities of the form $x \bowtie B$ where $\bowtie \in \{\leq, \geq, <, >\}$. Hereafter, we refer to one of such inequalities as a *constraint* of the PRHA.

For every constraint g in the form $x \leq B$ or $x < B$ (guard or invariant) of HA, we consider the minimum positive lower bound r_g for the derivative of x, if exists. For example, if we have three locations with $\dot{x} \in [1, 2]$, $\dot{x} \in [0, 3]$, $\dot{x} \in [-1, 2]$, we take $r_g = 1$ (since 0 and -1 are not positive). We consider the minimum lower bound v_g for the non-deterministic reset of x. For example, if we have three transitions with resets $x' \in [1, 2]$, $x' \in [0, 3]$, $x' \in [-1, 2]$, we take $v_g = -1$. In case g is in the form $x \geq B$ or $x > B$, we define r_g and v_g similarly by considering the maximum negative upper bound of the derivative of x and the maximum upper bound of the reset of x. We define the bound $\beta_g(x_0)$ as follows: $\beta_g(x_0) = max((B - x_0)/r_g, (B - v_g)/r_g)$.

Finally, as β we take the maximum among the β_g for all g in the automaton H for which r_g exists and the constant 1. Note that this coincides with the β defined above for TA and Parametric TA, where r_g is always 1 and v_g is always 0 and x_0 is always non negative.

3.4 Completeness for Rectangular Hybrid Automata

In this section, we restrict the focus to PRHA that are initialized and have bounded non-determinism. Moreover, we restrict the LTL formula to have the atoms that predicate over pc only. In this settings, we prove that the reduction to K-LIVENESS defined in the previous section is complete.

Given a PRHA $H = \langle P, Q, Q_0, E, X, \mathit{flow}, \mathit{init}, \mathit{inv}, \mathit{jump}, \mathit{guard}, \mathit{update}\rangle$ and an LTL formula ϕ with transition system $M_{\neg\phi} = \langle V, I, T\rangle$ and fairness condition $f_{\neg\phi}$, we build a new PRHA $H_{\neg\phi} = \langle P, Q', Q'_0, E', X, \mathit{flow}', \mathit{init}', \mathit{inv}', \mathit{jump}', \mathit{guard}', \mathit{update}'\rangle$ where:

- $Q' = \{q \times s \in Q \times \Sigma_V \mid q \in Q, s \models q\}$;
- $Q'_0 = \{q \times s \in Q' \mid q \in Q_0, s \models I\}$;
- $E' = \{\langle q \times s, q' \times s'\rangle \in Q' \times Q' \mid \langle q, q'\rangle \in E, s, s' \models T\}$;
- for all $q \times s \in Q$, $\mathit{flow}'(q \times s) = \mathit{flow}(q)$, $\mathit{init}'(q \times s) = \mathit{init}(q)$, $\mathit{inv}'(q \times s) = \mathit{inv}(q)$; for all $\langle q \times s, q' \times s'\rangle \in E'$, $\mathit{jump}'(\langle q \times s, q' \times s'\rangle) = \mathit{jump}(\langle q, q'\rangle)$, $\mathit{guard}'(\langle q \times s, q' \times s'\rangle) = \mathit{guard}(\langle q, q'\rangle)$, $\mathit{update}'(\langle q \times s, q' \times s'\rangle) = \mathit{update}(\langle q, q'\rangle)$.

It is easy to see that $H \models \phi$ iff $H_{\neg\phi} \models \mathbf{FG}\neg f_{\neg\phi}$ iff $M_{H_{\neg\phi}} \times Z_\beta \models \mathbf{FG}\neg f_Z$.

In order to prove that the reduction to K-LIVENESS is complete, we prove the following lemma.

Lemma 1. *Consider an initialized with bounded non-determinism PRHA H. Suppose $M_{H_{\neg\phi}} \times Z_\beta \models \mathbf{FG}\neg f_Z$. Let K_H and N_H be respectively the number of edges and locations of $H_{\neg\phi}$. Then $M_{H_{\neg\phi}} \times Z_\beta \models_{fin} \sharp(f_Z) \leq (K_C \cdot N_C) + 1$.*

Proof. We prove the lemma by induction on K_H. Suppose $K_H = 0$, i.e., there is no edge. Therefore, there cannot be a reset of the variables and, therefore, the time spent along a path of $M_{H_{\neg\phi}} \times Z_\beta$ must be less than $\beta(X_0)$ where X_0 is the initial value of X. Thus, f_Z cannot be visited twice.

Suppose $K_H \geq 1$. First, note that, since $M_{H_{\neg\phi}} \times Z_\beta \models \mathbf{FG}\neg f_Z$, $M_{H_{\neg\phi}} \times Z_\beta$ cannot have fair non-Zeno paths. Therefore, for every fair path σ of $M_{H_{\neg\phi}}$, there must be a constraint (or more than one) of $H_{\neg\phi}$ that eventually blocks the transition to f_Z in Z_β. Suppose f_Z is visited at least once (although for a finite number of times). Then, there must exist an edge e of $H_{\neg\phi}$ that is eventually no more taken along σ. Therefore, σ, after a certain point t, will coincide with a path of $M_{H'}$ where $M_{H'}$ is the encoding of a PRHA H' obtained from $H_{\neg\phi}$ by removing e and setting as initial state the state reached by σ at point t. Therefore $M_{H'} \times Z_\beta \models \mathbf{FG}\neg f_Z$ and H' has $K_H - 1$ edges. Thus, by induction f_Z must be visited less or equal than $(K_H - 1) \cdot N_H + 1$ times.

Finally we have to show that the number of times f_Z is visited before t is less than or equal to N_H. Suppose by contradiction, f_Z is visited $N_H + 1$ times. Then, at least one fair location of $H_{\neg\phi}$ is repeated so that we have a fair loop in the graph of $H_{\neg\phi}$. Due to the definition of Z_β, for any constraint $\overline{g}$ in the form $\overline{x} \leq B$ (and similar for the other cases), if $\overline{x}$ has a positive derivative and is not reset, $\overline{g}$ must become false between two fair states. Thus, either $\overline{g}$ is not used along the loop or $\overline{x}$ is reset or the derivative may be non positive. Since H is initialized, if the lower bound of the derivative becomes non positive, $\overline{x}$ must be reset. Therefore, every variable involved in a constraint along the path must be reset or the derivative can remain non-positive never violating the constraint. This means that there would be an infinite fair non-Zeno loop, which contradicts the hypothesis. We conclude that the number of times f_Z can be visited along σ is less than $(K_H - 1) \cdot N_H + 1 + N_H = (K_H \cdot N_H) + 1$. □

Theorem 2. *If $H \models \phi$, then there exists K such that $M_H \times M_{\neg\phi} \times Z_\beta \models_{fin} \sharp(f_Z) \leq K$.*

Proof. Since there exists a one-to-one mapping between the paths of $M_H \times M_{\neg\phi} \times Z_\beta$ and those of $M_{H_{\neg\phi}} \times Z_\beta$, by Lemma 1, $M_H \times M_{\neg\phi} \times Z_\beta \models_{fin} \sharp(f_Z) \leq (K_H \cdot N_H) + 1$ where K_H and N_H are respectively the edges and locations of $H_{\neg\phi}$. □

If the hybrid automaton falls outside of the class of initialized PRHA with bounded non-determinism, K-ZENO is still sound, but no longer guaranteed to be complete. A simple counterexample is shown in Figure 4, in which the stopwatch variable x is not reset when its dynamic changes. The automaton satisfies the property $(\mathbf{FG}good)$, because the invariant on x and the guard on y make sure

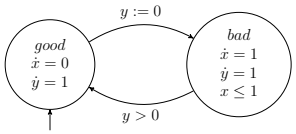


Fig. 4. Stopwatch automaton

that the total time spent in bad is at most 1 time unit. However, K-ZENO cannot prove it with any K because time can pass indefinitely in $good$, while x is stopped. Therefore, it is always possible to visit bad and f_Z an unbounded number of times. Finally, note that K-ZENO is able to prove other properties such as for example that the stopwatch automaton satisfies the formula $\mathbf{GF}good$.

4 Related Work

There are many works that focus on the verification of safety properties on hybrid systems [14–19], see [20] for a recent survey. We concentrate on the problem of liveness and deal with a semantics based on infinite paths.

The problem of checking time progress is well known and efficient solutions for TAs are based either on transforming the automaton in a strongly non-Zeno automaton [21], forcing the original TA to move to an additional (non-accepting) location in case of Zeno behavior, or on checking if from all reachable states, time can elapse from the 1 time unit [22]. In UPPAAL, this is achieved by taking the product of the model with a monitor automaton that changes state every c time units (where c is a "constant set to a good value w.r.t. the rest of the model") [23]. This approach is also used by DIVINE [24], an explicit-state model checker that is capable of verifying LTL properties over UPPAAL models. The monitors that we use in K-ZENO can be seen as a generalization of this approach. As discussed, using a constant is not enough for PTAs and (P)RHAs. Our method uses as bounds for time progress symbolic expressions over variables that change along a path.

A well-known reduction of liveness to safety is presented in [13]. The approach uses copies of the state variables to store the value of a state and search for a fair loop. In [25], the above technique is extended for different kinds of infinite-state systems such as pushdown systems and TAs, but each reduction is ad-hoc for the specific class of systems. A similar technique is also used in [26] for hybrid systems. The technique is not sound in general for infinite-state systems because there are simple cases where there are counterexamples but none of them has a lasso shape. So, it may be possible that the invariant holds in the reduced system but the original property does not hold. K-LIVENESS and K-ZENO are always sound: if the invariant holds for some K, then the original property is true.

Restricted to timed automata, apart from DIVINE, the UPPAAL model checker [27] does not support LTL, but a different fragment of temporal properties. However, this fragment of temporal logic does not consider infinite Zeno paths, but it is based on finite paths that possibly end in time-locks. A recent approach [28] considers LTL model checking for TAs. However, the authors explicitly assume to have TAs without Zeno paths. First, our approach differs since we allow Zeno paths in the model. Then, our technique is more general, since we handle hybrid automata with parameters.

With respect to hybrid systems, an interesting liveness property is stability, which requires that all the paths of the system eventually stay in a region. The work [29] reduces the verification of stability properties to compute a special kind of relations, called *snapshot sequences*, and to prove that these relations are well-founded. In principle, the same approach could be used to verify general LTL properties. However, the

application of this technique to LTL properties seems not straightworward and we are not aware of a publicly-available implementation with which to compare. Stability is reduced to termination analysis also in [30], assuming non-Zeno hybrid automata (i.e. bounded switching speed). In contrast, we focus on LTL properties and we take into account Zeno paths. In [31] the authors consider the problem of LTL model checking for discrete-time robust hybrid systems. Instead, we consider continuous-time systems.

Another line of work [32–34] for timed and hybrid systems focuses only on the falsification problem (i.e. find a counterexample if the LTL property does not hold). For timed automata, the work in [32] extends SMT-based BMC to search for a lasso-shaped path in the region abstraction. The proposed encoding also removes Zeno paths and, due to its nature, it could be used to complement our technique in the TA case to find counterexamples. For hybrid systems, the approach presented in [33] falsifies an LTL property by a randomized search while the one in [34] falsifies an MTL property under robustness assumptions. Both approaches do not consider Zeno paths and are not able to prove that a property holds.

Using the technique of [35], LTL model checking of infinite-state systems (including hybrid automata) may be reduced to finding disjunctively well-founded transition invariants, whose discovery can then be attempted with a solver for recursive Horn-like clauses like HSF [36]. However, the current implementation of HSF does not handle strict inequalities with real variables (e.g. $A < B$ is converted into $A + 1 \leq B$), and thus it cannot be applied easily to real-time systems.

5 Experimental Evaluation

5.1 Implementation

We have implemented the K-ZENO algorithm on top of the SMT extension of IC3 described in [9]. Given a symbolic system M and an LTL property ϕ, we use HYCOMP [1] (an extension of the NUSMV model checker) to generate the transition system $M_{\neg\phi}$ and to compute the function β for the transition system $Z_\beta(f, time, X)$ of Figure 2. $Z_\beta(f, time, X)$ is then added automatically to the system. In order to count the number of violations of f_Z, we use a simple integer counter. We remark that, although the completeness results hold only for initialized PHRA with bounded non-determinism, our implementation supports a more general class of HAs with rectangular dynamics. However, it currently can only be used to *verify* LTL properties, and not to disprove them. If a property does not hold, our tool does not terminate. Similarly to the Boolean case [12], our implementation consists of relatively few (and simple) lines of code on top of IC3. Both the tool and the benchmarks used in the evaluation can be downloaded at `http://es.fbk.eu/people/griggio/papers/cav14-kzeno.tar.bz2` for reproducing our results.

5.2 Benchmarks

We tried our approach on various kinds of benchmarks and properties.

Fischer family benchmarks. We considered 4 different versions of the Fischer mutual

exclusion protocol: the TA version from the UPPAAL distribution (*Fischer*), a parametric version (*Fischer Param*), a hybrid one (*Fischer Hybrid*), and one that ensures that every request is eventually served (*Fischer Fair*). All the variants are scalable in the number of processes involved, except for *Fischer Fair* that only considers 2 processes.

Distributed Controller [37] models the interactions of n sensors with a preemptive scheduler and a controller. We scaled the benchmark increasing the number of sensors.

Nuclear Reactor [38] models the control of a nuclear reactor with n rods. The benchmark is scaled increasing the number of control rods in the reactor.

Navigation family benchmarks: the models are inspired by the benchmarks presented in [39]. The benchmark describes the movement of an object in an nxn grid of square cells. Independently from the initial position, the object will eventually reach and stay in a target region. We created two versions of the benchmark, depending on whether the initial position of the object is given (*NavigationInit*) or not (*NavigationFree*). The benchmark is scaled by increasing the number of cells in the grid.

Diesel Generator [32]: the benchmark is an *industrial* model of an emergency diesel generator intended for the use in a nuclear power plant. The benchmark has three different versions (*small, medium, large*).

Bridge: the benchmark is from the UPPAAL distribution and models the bridge and torch puzzle. We used the same LTL properties used in the distribution of DIVINE [24].

Counter: the benchmark consists of an automaton with two locations, bad and $good$, and $n+1$ clocks, $x_0, x_1, \ldots, x_n$. The initial location bad has the invariant $x_0 \leq 1$ and a transition to $good$. bad has n self loops: each i-th self loop has guard $x_i \leq 1$ and reset x_{i-1}. The automaton is shown in Figure 5. On this model the LTL property $(\mathbf{FG}\ good)$ holds, since x_0 will eventually reach $x = 1$, forcing the transition to $good$. The example is interesting because the actual K needed to prove the property depends on the number of edges of the model, as shown in Lemma 1.

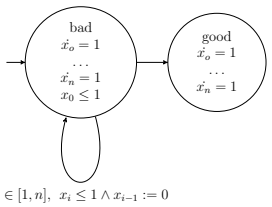


Fig. 5. *Counter* with $n+1$ clocks.

Note that the benchmarks fall in different classes: some of them are timed automata (*Fischer*, *Diesel Generator*, *Bridge*, *Counter*), some are parametrized timed automata (*Fischer Param*, *Fischer Fair*), some are initialized rectangular automata (*Fischer Hybrid*, *Nuclear Reactor*), while some have rectangular dynamics but are not initialized (*Distributed Controller*, *NavigationInit*, *NavigationFree*).

We manually generated several meaningful LTL properties for the benchmarks of the Fischer family, the *Distributed Controller* and the *Nuclear Reactor*. The properties match several common patterns for LTL like fairness ($\mathbf{GF}p$), strong fairness ($\mathbf{GF}p \rightarrow \mathbf{GF}q$), and "leads to" ($\mathbf{G}(p \rightarrow \mathbf{F}q)$). Moreover, in several cases we added additional fairness constraints to the common patterns to generate properties that hold in the model. For the *Bridge* and *Diesel Generator* benchmarks we used the properties already specified in the models. For the navigation benchmark we checked that eventually the object will stay forever in the "stability" region. Finally, we used the property $(\mathbf{FG}\ good)$ in the *Counter* benchmarks.

Table 1. Selected experimental results.

Instance	Class	Property	# Bool vars	# Real vars	Trans size	k	Time
Fischer (8 processes)	T	$(\bigwedge_{i=1}^{17} \mathbf{GF}p_i) \rightarrow \mathbf{G}(\neg p_{18} \rightarrow \mathbf{F}p_{18})$	132	20	1286	3	6.37
Fischer Fair (2 processes)	P	$(p_1 \wedge \mathbf{GF}p_2) \rightarrow \mathbf{G}(p_3 \rightarrow \mathbf{F}p_4)$	38	12	622	4	76.14
Fischer Hybrid (10 procs)	R	$(\mathbf{GF}p_1 \wedge \mathbf{GF}p_2 \wedge \mathbf{FG}p_3) \rightarrow \mathbf{G}(p_4 \rightarrow \mathbf{F}p_5)$	106	64	8759	1	325.03
Dist Controller (3 sensors)	N	$(\mathbf{GF}p_1) \rightarrow (\mathbf{GF}p_2)$	58	27	1737	1	397.24
Nuclear Reactor (9 rods)	R	$\mathbf{G}(p_1 \rightarrow \mathbf{F}p_2)$	82	24	3258	1	530.40
NavigationInit (3x3)	N	$\mathbf{FG}(p_1 \vee p_2 \vee p_3 \vee p_4)$	16	8	808	2	4.37
NavigationInit (10x10)	N	$\mathbf{FG}(p_1 \vee p_2 \vee p_3 \vee p_4)$	22	8	4030	2	453.74
NavigationFree (3x3)	N	$\mathbf{FG}(p_1 \vee p_2 \vee p_3 \vee p_4)$	16	8	808	2	3.37
NavigationFree (9x9)	N	$\mathbf{FG}(p_1 \vee p_2 \vee p_3 \vee p_4)$	22	8	3461	2	872.07
Counter 10	T	$\mathbf{FG}p$	10	24	294	10	52.74
Diesel Gen (small)	T	$\mathbf{G}(p_1 \rightarrow \mathbf{F}(\neg p_2 \vee p_3))$	84	24	724	1	16.55
Diesel Gen (medium)	T	$\mathbf{G}(p_1 \rightarrow \mathbf{F}(\neg p_2 \vee p_3))$	140	30	1184	1	51.24
Diesel Gen (large)	T	$\mathbf{G}(p_1 \rightarrow \mathbf{F}(\neg p_2 \vee p_3 \vee p_4))$	264	62	2567	1	538.39

Classes: T: timed, **P:** parametric timed, **R:** rectangular, **N:** non-initialized rectangular.

5.3 Evaluation

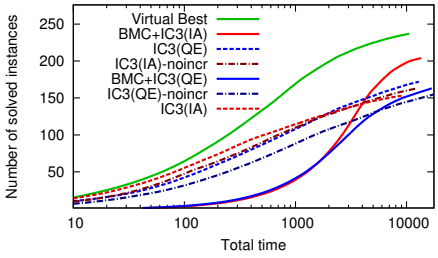
Effectiveness. In order to evaluate the feasibility of our approach, we have run it on a total of 276 verification tasks, consisting of various LTL properties on the benchmark families described above. Our best configuration could solve 205 instances within the resource constraints (900 seconds of CPU time and 3Gb of memory). If instead we consider the "Virtual Best" configuration, obtained by picking the best configuration for each individual task, our implementation could solve 238 problems. We report details about some of the properties we could prove in Table 1. On each row, the table shows the model name, the class of instances it belongs to (timed, parametric, rectangular, non-initialized rectangular), the property proved (with variables p_i's used as placeholders for atomic propositions), the size of the symbolic encoding (number of Boolean and Real variables, and number of nodes in the formula DAG of the transition relation), the value of k reached by K-LIVENESS[1], and the total execution time. We remark that we are not aware of any other tool capable of verifying similar kinds of LTL properties on the full class of instances we support.

Heuristics and Implementation Choices. We analyze the performance impact of different heuristics and implementation choices along the following dimensions:

Invariant checking engine. We have two versions of SMT-based IC3, one based on approximated preimage computations with quantifier elimination (called IC3(QE) here), and one based on implicit predicate abstraction (IC3(IA)). Our recent results [9] indicate that IC3(IA) is generally superior to IC3(QE) on software verification benchmarks. However, the situation is less clear in the domain of timed and hybrid systems.

Incrementality. We compare our fully-incremental implementation of K-LIVENESS to a non-incremental one, in which IC3 is restarted from scratch every time the K-LIVENESS

[1] On most of the instances the value of k reached by K-LIVENESS is small. The explanation is that, on real models, the number of constraints that must be violated inside a loop that contains $f_{\neg\phi}$ before time diverges is usually low. The benchmarks of the *Counter* family were created on purpose, to show that k can increase arbitrarily.



Configuration	# solved	Tot time
Virtual Best	238	10603
BMC+IC3(IA)	205	13525
IC3(QE)	173	12895
IC3(IA)-NOINCR	164	12476
BMC+IC3(QE)	164	16888
IC3(QE)-NOINCR	156	17643
IC3(IA)	154	8493

Fig. 6. Experimental comparison of various configuration options.

counter is incremented.

Initial value of the K-LIVENESS *counter.* We consider the impact of starting the search with a right (or close to) value for the K-LIVENESS counter k, instead of always starting from zero, in IC3. For this, we use a simple heuristic that uses BMC to guess a value for the counter: we run BMC for a limited time (20 seconds in our experiments), increasing k every time a violation is detected. We then start IC3 with the k value found.

Overall, we considered six different configurations: IC3(IA) and IC3(QE) are the default, incremental versions of K-LIVENESS with IC3, using either approximate quantifier elimination or implicit abstraction; IC3(IA)-NOINCR and IC3(QE)-NOINCR are the non-incremental versions; BMC+IC3(IA) and BMC+IC3(QE) are the versions using a time-limited initial BMC run for computing an initial value for the K-LIVENESS counter k. The six configurations are compared in Fig. 6, showing the number of instances solved (y-axis) and the total execution time (x-axis). The figure also includes the "Virtual Best" configuration, constructed by taking the best result for each individual instance.

Fig. 6 shows that, differently from the case of software verification, the default version of IC3(QE) performs much better than IC3(IA). Although we currently do not have a clear explanation for this, our conjecture is that this is due to the "bad quality" of the predicates found by IC3(IA) in the process of disproving invariants when the value of k is too small. Since IC3(IA) never discards predicates, and it only tries to add the minimal amount of new predicates when performing refinements, it might simply get lost in computing clauses of poor quality due to the "bad" language of predicates found. This might also be the reason why IC3(IA)-NOINCR performs better than IC3(IA), despite the runtime cost of restarting the search from scratch every time k changes: when restarting, IC3(IA)-NOINCR can also throw away bad predicates. A similar argument can also be applied to BMC+IC3(IA): using BMC to skip the bad values of k allows IC3(IA) to find predicates that are more relevant/useful for proving the property with the good (or close to) value of k.

The situation for IC3(QE) is instead completely different. In this case, not only turning off incrementality significantly hurts performance, as we expected, but also using BMC is detrimental. This is consistent with the behavior observed in the finite-state case for the original K-LIVENESS implementation [12]. However, as the authors of [12], also in this case we do not have a clear explanation for this behavior.

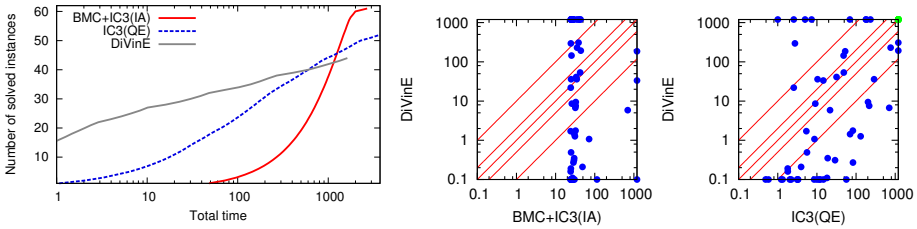


Fig. 7. Comparison with DIVINE.

Comparison with Other Tools. We conclude our evaluation with a comparison of our implementation with alternative tools and techniques working on similar systems. As already remarked above, we are not aware of any tool that is able to handle arbitrary LTL properties on the class of systems that we support. Therefore, we concentrate our comparison only on Timed Automata, comparing with DIVINE [24]. We use a total of 64 instances from the *Fischer*, *Bridge* and *Counter* families. Unfortunately, we could not include the industrial *Diesel Generator* model, since it is modeled as a symbolic transition system, whereas DIVINE expects a network of timed automata (in UPPAAL format) as input. However, the *Diesel Generator* benchmark was reported to be very challenging for explicit-state approaches [32].

The results are shown in Fig. 7, where we compare DIVINE with our two best configurations, BMC+IC3(IA) and IC3(QE). We can see that DIVINE is very fast for simple instances, outperforming our tool by orders of magnitude. However, its performance degrades quickly as the size of the instances increases. In contrast, both BMC+IC3(IA) and IC3(QE) scale better to larger instances. This is particularly evident for BMC+IC3(IA): after having found a good initial value for the K-LIVENESS counter with BMC, IC3(IA) can solve almost all the instances in just a few seconds.

6 Conclusions and Future Work

We presented a new approach to the verification of liveness properties on hybrid systems, in particular of LTL properties, with SMT-based techniques. The approach relies on the K-LIVENESS idea of reducing the problem for finite-state systems to proving that an accepting condition can be visited at most K times. The new algorithm, K-ZENO, exploits the divergence of time to make the reduction succeed in proving properties on hybrid systems. We prove that the reduction is complete for a class of parametric rectangular hybrid automata. An extensive evaluation shows the effectiveness and scalability of the approach.

There are various directions for future work. Some of our objectives are to find optimizations for linear hybrid systems using relational abstraction [40], to apply the approach to LTL satisfiability in order to enable compositional contract-based reasoning [41], and to extend the idea to deal with continuous-time temporal logics and first-order theories different from reals.

References

1. HyComp: https://es.fbk.eu/tools/hycomp/
2. Henzinger, T., Kopke, P., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: STOC. (1995) 373–382
3. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In: Hybrid Systems. (1992) 209–229
4. Pnueli, A.: The Temporal Logic of Programs. In: FOCS. (1977) 46–57
5. Vardi, M.: An Automata-Theoretic Approach to Linear Temporal Logic. In: Banff Higher Order Workshop. (1995) 238–266
6. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying industrial hybrid systems with mathsat. Electr. Notes Theor. Comput. Sci. **119**(2) (2005) 17–32
7. Cimatti, A., Mover, S., Tonetta, S.: Quantifier-free encoding of invariants for hybrid systems. Formal Methods in System Design (2013) 1–24
8. Bradley, A.: SAT-Based Model Checking without Unrolling. In: VMCAI. Volume 6538 of LNCS., Springer (2011) 70–87
9. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 Modulo Theories via Implicit Predicate Abstraction. In: TACAS. LNCS, Springer (2014) To Appear.
10. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: CAV. (1997) 72–83
11. Tonetta, S.: Abstract Model Checking without Computing the Abstraction. In: FM. (2009) 89–105
12. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In Cabodi, G., Singh, S., eds.: FMCAD, IEEE (2012) 52–59
13. Schuppan, V., Biere, A.: Efficient reduction of finite state model checking to reachability analysis. STTT **5**(2-3) (2004) 185–204
14. Henzinger, T.A., Ho, P., Wong-Toi, H.: HYTECH: A Model Checker for Hybrid Systems. STTT **1**(1-2) (1997) 110–122
15. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable Verification of Hybrid Systems. In: CAV. (2011) 379–395
16. Platzer, A.: Differential Dynamic Logic for Hybrid Systems. J. Autom. Reasoning **41**(2) (2008) 143–189
17. Alur, R., Dang, T., Ivancic, F.: Predicate abstraction for reachability analysis of hybrid systems. ACM Trans. Embedded Comput. Syst. **5**(1) (2006) 152–199
18. Clarke, E.M., Fehnker, A., Han, Z., Krogh, B.H., Ouaknine, J., Stursberg, O., Theobald, M.: Abstraction and counterexample-guided refinement in model checking of hybrid systems. Int. J. Found. Comput. Sci. **14**(4) (2003) 583–604
19. Prabhakar, P., Duggirala, P.S., Mitra, S., Viswanathan, M.: Hybrid automata-based cegar for rectangular hybrid systems. In: VMCAI. (2013) 48–67
20. Alur, R.: Formal verification of hybrid systems. In: EMSOFT. (2011) 273–278
21. Tripakis, S., Yovine, S., Bouajjani, A.: Checking timed büchi automata emptiness efficiently. Formal Methods in System Design **26**(3) (2005) 267–292
22. Tripakis, S.: Verifying Progress in Timed Systems. In: ARTS. (1999) 299–314
23. David, A., Larsen, K.: More features in UPPAAL
24. Barnat, J., Brim, L., Havel, V., Havlícek, J., Kriho, J., Lenco, M., Rockai, P., Still, V., Weiser, J.: DiVinE 3.0 - An Explicit-State Model Checker for Multithreaded C & C++ Programs. In: CAV. (2013) 863–868
25. Schuppan, V., Biere, A.: Liveness Checking as Safety Checking for Infinite State Spaces. Electr. Notes Theor. Comput. Sci. **149**(1) (2006) 79–96

26. Bresolin, D.: HyLTL: a temporal logic for model checking hybrid systems. In: HAS. (2013) 73–84
27. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. STTT **1**(1-2) (1997) 134–152
28. Laarman, A., Olesen, M.C., Dalsgaard, A.E., Larsen, K.G., van de Pol, J.: Multi-core Emptiness Checking of Timed Büchi Automata Using Inclusion Abstraction. In Sharygina, N., Veith, H., eds.: CAV. Volume 8044 of LNCS., Springer (2013) 968–983
29. Podelski, A., Wagner, S.: Region stability proofs for hybrid systems. In: FORMATS. (2007) 320–335
30. Duggirala, P., Mitra, S.: Abstraction Refinement for Stability. In: ICCPS. (2011) 22–31
31. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of ltl properties of non-linear robust discrete time hybrid systems. Int. J. Found. Comput. Sci. **18**(1) (2007) 63–86
32. Kindermann, R., Junttila, T.A., Niemelä, I.: Beyond lassos: Complete smt-based bounded model checking for timed automata. In: FMOODS/FORTE. (2012) 84–100
33. Plaku, E., Kavraki, L.E., Vardi, M.Y.: Falsification of ltl safety properties in hybrid systems. STTT **15**(4) (2013) 305–320
34. Nghiem, T., Sankaranarayanan, S., Fainekos, G.E., Ivancic, F., Gupta, A., Pappas, G.J.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: HSCC. (2010) 211–220
35. Podelski, A., Rybalchenko, A.: Transition Invariants. In: LICS, IEEE Computer Society (2004) 32–41
36. Grebenshchikov, S., Lopes, N.P., Popeea, C., Rybalchenko, A.: Synthesizing software verifiers from proof rules. In Vitek, J., Lin, H., Tip, F., eds.: PLDI. (2012) 405–416
37. Henzinger, T.A., Ho, P.H.: Hytech: The cornell hybrid technology tool. In: Hybrid Systems. (1994) 265–293
38. Wang, F.: Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures. IEEE Trans. Software Eng. **31**(1) (2005) 38–51
39. Fehnker, A., Ivancic, F.: Benchmarks for hybrid systems verification. In: HSCC. (2004) 326–341
40. Mover, S., Cimatti, A., Tiwari, A., Tonetta, S.: Time-aware relational abstractions for hybrid systems. In: EMSOFT. (2013) 1–10
41. Cimatti, A., Dorigatti, M., Tonetta, S.: OCRA: A tool for checking the refinement of temporal contracts. In: ASE. (2013) 702–705