CEGAR-based Verification of Temporal Properties of Robotic Systems

Alvin A. George
Indian Institute of Science
Bangalore, India

Deepak D'Souza
Indian Institute of Science
Bangalore, India

Pavithra Prabhakar
Kansas State University
Manhattan, USA

***Abstract*— We consider the problem of verifying linear-time temporal properties, specifically, liveness properties, of discrete-time systems with (uncountable) infinite states often arising in robotic applications. We present a counter-example guided abstraction refinement (CEGAR) based technique tailored to this setting. Specifically, we provide novel validation and refinement algorithms to handle the lasso shaped abstract counterexamples that witness the violation of the liveness properties. We give a characterization of a feasible lasso-shaped abstract counter-examples based on the notion of recurrent sets. Using this, we propose an algorithm for validating counter-examples by iteratively computing pre-sets around the loop. Finally, we use the pre-sets computed by the validation algorithm to refine the abstraction to eliminate spurious counter-examples. We have implemented our approach in a logical setting, with Z3 as the satisfiability checker. Preliminary results show the technique to be effective in terms of both proving that the system satisfies the given property, as well as in finding valid counter-examples (falsification) as compared to two competing verification tools that handle continuous state-space and liveness properties.**

I. Introduction

Robotic systems are increasingly being deployed in a wide range of safety-critical applications, such as autonomous driving, surgical robotics, and space exploration, where ensuring correct and reliable behavior is of paramount importance. Formal methods refer to a class of techniques that provide rigorous techniques for guaranteeing the correctness of systems. Formal verification deals with techniques that take a model of a (robotic) system and a formal specification of the requirement and check if the system meets the specification. Alternately, formal synthesis synthesizes a controller such that the system under this controller satisfies the specification.

In the context of robotics, formal synthesis has been extensively explored to obtain correct-by-construction controller and plans [1]–[3] for properties specified in temporal logics. Given the infinite statespace of the robotic systems, these techniques are often accompanied by abstraction techniques that involve the construction of a simplified finite state model on which the formal synthesis techniques are employed to obtain strategies from which controllers are then extracted. On one hand, larger more precise abstractions are desirable in that they retain a large subset of the controllers satisfying the specification with respect to the original system. On the other hand, larger abstractions are computationally expensive in terms of strategy synthesis. Hence, a challenge toward successful synthesis is to find the sweet spot between precision and size, which we explore in this paper using a paradigm called the counter-example guided abstraction refinement (CEGAR) [4] method. Developing the CEGAR framework for continous infinite statespace and liveness properties is challenging. Hence, instead of the computationally expensive synthesis problem that requires the computation of the controller for a given specification, as a first step, we focus on the simpler problem of verification – proving that a given control/plan for a robotic system indeed satisfies the desired liveness property.

We consider the problem of formal verification of robotic systems, represented by discrete-time dynamical system controlled by a state-feedback control law. We are interested in verifying specifications represented by temporal logics that have been widely used to specify robotic properties including liveness properties such as visiting a goal infinitely often ($GF goal$) and eventually reaching and remaining within a region ($FGregion$) [5], [6]. Mathematically, the problem is that of verifying infinite horizon temporal logic properties against an (uncountable) infinite state transition system. While abstractions and counter-example guided refinement have been explored for infinite state systems, they are typically restricted to countable state-spaces such as those represented by integer variables rather than reals [7]–[9]. On the other hand, real variables have been considered in the context of hybrid systems [10], [11], however, the focus is on safety properties, rather than liveness.

In this paper, we propose a CEGAR-based approach for reasoning about infinite horizon LTL properties, which is applicable to a general class of systems, including discrete-time systems with real-valued variables. Broadly, the CEGAR approach consists of refining abstractions that attempt to eliminate undesirable behaviors in the current abstraction, in the form of abstract counter-examples. Specifically, the abstract counter-examples are validated to check if they correspond to feasible behaviors in the original system, and insights from this analysis are used to refine the abstraction. The challenge arises due to the complex nature of counter-examples for liveness properties, namely, lasso shaped paths. We tackle this by borrowing the notion of recurrent sets, introduced by Gupta et al. [12] in the context of proving non-termination of programs, and propose a semi-algorithm to compute a *maximum* recurrent set by iteratively computing pre-images along the loop of the lasso. Further, we provide a novel refinement algorithm to eliminates a spurious lasso path from the abstraction, based on the computed pre-images.

We implement our approach in a tool called RecCEGAR, and evaluate it on a large set of robotic benchmarks consisting of discrete-time linear dynamics with state-feedback controllers that are piecewise linear. We compare our approach with two related approaches: the MUVAL tool which implements a fixpoint based approach [13], and another CEGAR based approach proposed by Mondok et al. [14]. Our approach performs significantly better than both the tools in terms of both the number of benchmarks for which the property is proved/disproved and time taken for the same.

Related Work. Several works [15]–[17] reduce liveness checking to safety checking of a modified system, in the context of finite-state systems. Cimatti et al [18] extend the K-Liveness technique of [17] for Rectangular Hybrid Automata. The technique is specific to timed systems while our technique is applicable for more general systems. In the context of program termination, Gupta et al. [12] and [19], [20] address termination of lasso paths via recurrent sets and ranking functions respectively. In contrast, we rely on maximal recurrent sets to prove both feasibility and infeasibility of lasso paths. Among the CEGAR-based approaches, [7], [8] rule out fair non-terminating paths in a product of the system and the formula automaton by building an automaton representing infeasible program paths learnt so far. Daniel et al. [21] extend the liveness-to-safety (L2S) approach of [15] to infinite-state systems, using ”implicit abstraction” (IA) built on an IC3 engine. In recent work, Cimatti et al [22] combine this approach with the technique of “shoals” used in liveness-checking for finite-state systems [23], in their tool called rlive-inf. Mondok et al [14] check for feasibility of lasso paths based on recurrent points, and use interpolants to refine the abstraction. In contrast to the above CEGAR approaches, our approach is based on computing recurrent sets for both feasibility and refinement.

Finally, Unno et al [13] give a semi-decision procedure for checking validity of a form of first-order logic with fixpoints called μ-CLP, which can also phrase LTL properties. As we show in experiments, a CEGAR-based approach exhibits complementary strengths vis-a-vis such fixpoint based approaches.

II. OVERVIEW AND ILLUSTRATIVE EXAMPLE

Fig. 1 shows a schematic overview of our approach. We are given a concrete transition system $\mathcal{T}$, specified in a logical representation, and an LTL formula φ representing the property of the system to be checked. The Compute Abstraction module essentially takes a finite label-respecting partition of the state-space of $\mathcal{T}$ and constructs a finite-state abstraction $\mathcal{A}$ of $\mathcal{T}$. The abstraction $\mathcal{A}$ represents an overapproximation of the behaviours of $\mathcal{T}$ and will be successively refined. Initially we use a simple partition based on the initial states of the system and the atomic predicates specified in the property. The abstract system $\mathcal{A}$ thus computed is then model-checked for the property φ by a standard LTL model-checking component. If the model-checker says $\mathcal{A}$ satisfies the property, then we declare that $\mathcal{T}$ satisfies the property and we are done. If not, the model-checker returns a lasso-shaped counter-example path in $\mathcal{A}$ which violates the property. The Validation module then checks if the given counter-example lasso is feasible in that there is an infinite execution in the concrete system $\mathcal{T}$ which lies along this abstract path. This component uses a novel algorithm based on computing a maximal recurrent set for the lasso. If it finds the lasso to be feasible, we can say that the system does not satisfy the property and output an abstract lasso which represents a concrete execution of the system which violates the property. If not, the validation component returns a collection of “pre-sets” which are then used by Compute Refined Partition component to compute a new (refined) label-respecting partition Π of the concrete state-space. This is fed to the Compute Abstraction module, and the cycle repeats.

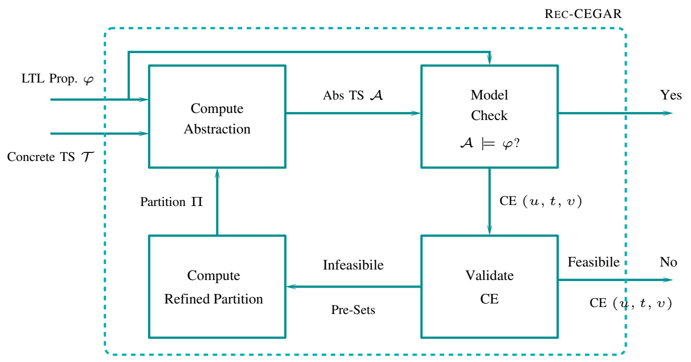


Fig. 1: Schematic diagram of REC-CEGAR

We now illustrate our approach on an example robot gridworld system shown in Fig. 2a. The state space of the system is $\mathbb{R}^2$, and it is divided into a grid of 3×3 unit cells (named $\mathrm{C11}, \mathrm{C12}, \ldots, \mathrm{C33}$), and the area outside the grid O. We use the convention that each cell includes its left and bottom borders, but not its right and top borders. For example, the cell C11 represents the set of points $[0, 1) \times [0, 1)$.

Each cell of the grid has a velocity vector associated with it, represented by an arrow shown inside the cell. For points in O, the vector is assumed to be 0. More precisely, the discrete-time dynamics of the system is defined by $x(t+1) = x(t) + u(t)$, where x is the position vector and

$$u(t) = \begin{cases} (1, 0) & \text{if } x \in \mathrm{C11} \cup \mathrm{C21} \cup \mathrm{C12}, \\ (0, 1) & \text{if } x \in \mathrm{C31}, \\ (0, -1) & \text{if } x \in \mathrm{C13} \cup \mathrm{C33}, \\ (-1, 0) & \text{if } x \in \mathrm{C32}, \\ (0.7, 0.7) & \text{if } x \in \mathrm{C22}, \\ (-0.7, -0.7) & \text{if } x \in \mathrm{C23}, \\ (0, 0) & \text{otherwise.} \end{cases}$$

From each point in the state space we thus have a unique trajectory of the system starting from that point. The property we want to check is the following: does every trajectory of the robot starting from a point in the C33 visit the cell C22 infinitely often? In temporal logic we can express this property as $\mathrm{C33} \implies GF\ \mathrm{C22}$.

Our approach begins by computing an abstraction of the system based on a partition of the state-space induced by the initial states of the system and the atomic predicates in the given property, as shown in Fig. 2b. The abstraction has three states N0, N1 and N2, corresponding respectively to the initial cell C33, goal cell C22, and the rest of the cells along with the outside region O. We now model-check this abstraction for the property $GF(\mathrm{N1})$, and find the counterexample path $\mathrm{N0} \cdot (\mathrm{N2})^{\omega}$.

We then check the feasibility of this abstract counterexample, by computing successive pre-sets along the N2 loop. The computation terminates with a recurrent set comprising the region O, which turns out to be not reachable from the initial region, and hence the counterexample path is proved to be spurious. The computed pre-sets partition the cell C23 into the three regions shown as a, b, and c in Fig. 2c. We now refine the node N2 using the computed pre-sets, to obtain the second abstraction in Fig. 2d. In the resulting abstraction the state N0 is refined into 5 new states: N2.0 the union of C31, C13 and b, N2.1 the union of C12, C32 and c, N2.2 the union of C21 and a, N2.3 the union of C11, and N2.4 the region O. We now we model-check this new abstraction, to find that it satisfies the property. We thus declare that the original system satisfies its property.

Let us now consider checking the property $FG(\mathrm{C22})$ for the same system. The approach follows a similar route to obtain the second abstraction. However, the second abstraction has a counterexample path $N0 \cdot (N1\, N2.1)^{\omega}$. Our approach finds a recurrent set for this loop, comprising the regions d and g of cell C22, shown in Fig. 2e. As the recurrent set turns out to be reachable from the initial node, we say the system violates the FG property and return the feasible lasso-shaped path as a counterexample.

III. Preliminaries

For functions $f : A \to B$ and $g : B \to C$, we define the composition of f followed by g, denoted $f \circ g$, to be the function $h : A \to C$ given by $h(x) = g(f(x))$ for each $x \in A$.

A. Transition Systems

A *transition system* over a set of atomic predicates P is a tuple $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$ where S is the set of states, $I \subseteq S$ is a non-empty set of initial states, $\rightsquigarrow \subseteq S \times S$ is the transition relation, P is a set of atomic predicates, and ℓ is a labelling function from S to 2^P which associates with each state $s \in S$ a set of atomic propositions from P.

Let us fix a transition system $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$ for the rest of this subsection. We will use $s \rightsquigarrow s'$ to denote that $(s, s') \in \rightsquigarrow$. A *finite path* in $\mathcal{T}$ is a (possibly empty) sequence of states $\rho = s_0 s_1 s_2 \cdots s_n$ where each $s_i \in S$ and $s_i \rightsquigarrow s_{i+1}$ for all $0 \leq i < n$. A *simple* path is a finite path in which no state repeats. An *infinite path* in $\mathcal{T}$ is an infinite sequence of states $\rho = s_0 s_1 s_2 \cdots$ where $s_i \in S$ and $s_i \rightsquigarrow s_{i+1}$ for all $0 \leq i$. A finite (or an infinite) *initial* path of $\mathcal{T}$ is a finite (or an infinite) path that starts in an initial state of $\mathcal{T}$. By a *path* in $\mathcal{T}$ we will mean an infinite path in $\mathcal{T}$ unless stated otherwise.

For finite paths u and v in $\mathcal{T}$, we define the concatenation of u and v, denoted $u \cdot v$, as follows. If u is empty, then $u \cdot v$ is defined to be v (and similarly u when v is empty). Otherwise, if $u = q_0 q_1 \cdots q_m$ and $v = s_0 s_1 \cdots s_n$, such that $q_m \rightsquigarrow s_0$, we define $u \cdot v$ to be the path $q_0 q_1 \cdots q_m s_0 s_1 \cdots s_n$ in $\mathcal{T}$. Similarly, for a finite path $u = q_0 q_1 \cdots q_m$ and an infinite path $\rho = s_0 s_1 \cdots$, such that $q_m \rightsquigarrow s_0$, we define the concatenation of u and ρ denoted $u \cdot \rho$, to be the infinite path $q_0 q_1 \cdots q_m s_0 s_1 \cdots$ in $\mathcal{T}$. If u is empty, we define $u \cdot \rho$ to be ρ. For convenience we will sometimes simply write uv instead of $u \cdot v$, etc.

A *lasso-shaped initial path* (or simply a *lasso*) in $\mathcal{T}$, is a tuple (u, t, v), where u and v are finite paths in $\mathcal{T}$, and t is a state in $\mathcal{T}$, such that $utvt$ is a finite initial path in $\mathcal{T}$. The lasso (u, t, v) represents the ultimately periodic initial path $u(tv)^{\omega}$ in $\mathcal{T}$. We call t the *pivot* state, ut the *stem*, and tvt the *loop* of the lasso. A *simple lasso* is a lasso (u, t, v) where neither the loop nor the stem has repeating states (i. e. ut and vt are simple paths).

The *trace* of a path ρ in $\mathcal{T}$, denoted $\mathit{trace}(\rho)$, is the sequence of state labels $\ell(s_0)\ell(s_1)\cdots$ along ρ. The set of traces of $\mathcal{T}$, denoted $\mathit{traces}(\mathcal{T})$, is the set of traces of initial paths in $\mathcal{T}$.

The *preimage* of a set of states Y in $\mathcal{T}$, denoted $\mathit{Pre}_{\mathcal{T}}(Y)$, is the set of states from which we can transition to Y in one step. Thus $\mathit{Pre}_{\mathcal{T}}(Y) = \{s \in S \mid \exists s' \in Y \text{ such that } s \rightsquigarrow s'\}$. The *postimage* of a set of states Y in $\mathcal{T}$, denoted $\mathit{Post}_{\mathcal{T}}(Y)$, is the set of states to which we can transition from Y in one step. That is $\mathit{Post}_{\mathcal{T}}(Y) = \{s' \in S \mid \exists s \in Y \text{ such that } s \rightsquigarrow s'\}$.

A (*finite*) *partition* of $\mathcal{T}$ is a set $\Pi = \{B_0, B_1, \ldots, B_k\}$ of subsets of S, such that $B_i \cap B_j = \varnothing$ for all $i \neq j$ and $\bigcup_{i=0}^{k} B_i = S$. A *label-respecting* partition of $\mathcal{T}$ is a partition $\Pi = \{B_0, B_1, \ldots B_k\}$ of $\mathcal{T}$, such that all states in a block have the same label. i. e. for each B_i, and $s, s' \in B_i$, we have $\ell(s) = \ell(s')$.

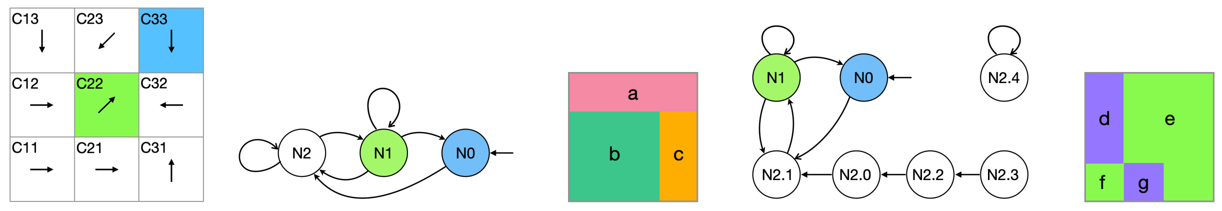
B. Linear-Time Temporal Logic

We will use Linear-Time Temporal Logic (LTL) to specify properties of transitions systems. Let P be a set of atomic predicates. Then the set of LTL formulas over P, written LTL(P), is given by: $\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi$ where $p \in P$. An LTL(P) formula is interpreted over a sequence $\tau = a_0 a_1 \cdots$ of valuations over P in the standard way [24]. In particular

$$\begin{array}{lll} \tau, i \models \varphi U \psi & \text{iff} & \exists k \text{ s. t. } i \leq k, \text{ and } \tau, k \models \psi \text{ and} \\ & & \forall j \text{ s. t. } i \leq j < k, \text{ we have } \tau, j \models \varphi. \end{array}$$

We will use the derived modalities F ("Future") and G ("Globally"), defined as $F\varphi = \mathit{true}\, U \varphi$ and $G\varphi = \neg F(\neg\varphi)$. We say that a sequence τ satisfies an LTL formula φ, written $\tau \models \varphi$, if the sequence τ satisfies the formula at the initial position (i.e. $\tau, 0 \models \varphi$). We denote by $\mathit{traces}(\varphi)$ the set of sequences of valuations over P which satisfy φ.

Let $\mathcal{T}$ be a transition system over a set of predicates P. We say that $\mathcal{T}$ satisfies an LTL(P) formula φ, written $\mathcal{T} \models \varphi$, if $\mathit{traces}(\mathcal{T}) \subseteq \mathit{traces}(\varphi)$ (i. e. all traces of $\mathcal{T}$ satisfy φ). The model-checking problem for LTL is the following: Given a



(a) Robot grid system (b) First abstraction (c) Cell C23 partition (d) Second abstraction (e) Cell C22 rec set

Fig. 2: Example gridworld system and successive abstractions obtained by REC-CEGAR.

transition system $\mathcal{T}$ over P, and an LTL(P) property φ, does $\mathcal{T} \models \varphi$?

IV. Abstraction

In this section we define the notion of one transition system being an abstraction of another.

Definition 4.1 (): We say that a transition system $\mathcal{A} = (\widehat{S}, \widehat{I}, \widehat{\rightsquigarrow}, \widehat{P}, \widehat{\ell})$ is an *abstraction* of another transition system $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$, if $P = \widehat{P}$, and there exists an *onto* function $\alpha : S \to \widehat{S}$ such that nosep

- for each s in S, $\ell(s) = \widehat{\ell}(\alpha(s))$,
- for each $s \in I$, we have $\alpha(s) \in \widehat{I}$, and
- for all s, s' in S, $s \rightsquigarrow s'$ implies $\alpha(s) \widehat{\rightsquigarrow} \alpha(s')$.

In this case we call α an *abstraction function* from $\mathcal{T}$ to $\mathcal{A}$, and write $\mathcal{T} \preceq_\alpha \mathcal{A}$.

We will work with abstractions represented by a label-respecting partition of the concrete system states. Given a transition system $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$, a label-respecting partition $\Pi = \{B_0, B_1, \ldots, B_n\}$ of S represents an abstraction $\mathcal{A}_\Pi$ of $\mathcal{T}$, which is defined as follows. We have $\mathcal{A}_\Pi = (\widehat{S}, \widehat{I}, \widehat{\rightsquigarrow}, \widehat{P}, \widehat{\ell})$, where

- $\widehat{S} = \Pi$
- $\widehat{I} = \{B_i \mid B_i \cap I \neq \varnothing\}$
- The transition relation $\widehat{\rightsquigarrow}$ is given by $B_i \widehat{\rightsquigarrow} B_j$ iff there exists $s_i \in B_i$ and $s_j \in B_j$ such that $s_i \rightsquigarrow s_j$,
- $\widehat{P} = P$
- $\widehat{\ell}$ is given by $\widehat{\ell}(B_i) = \ell(s)$ where s is any state in B_i. We can see that $\widehat{\ell}$ is well-defined since for any s, s' in B_i, $\ell(s) = \ell(s')$ (by virtue Π being label-respecting).

$\mathcal{A}_\Pi$ is an abstraction of $\mathcal{T}$ via the abstraction function α_Π given by $\alpha_\Pi(s) = B_i$ where $s \in B_i$. If Π and Δ are label-respecting partitions of $\mathcal{T}$ such that Δ refines Π (in the standard sense, that if s and s' belong to the same block of Δ they do so in Π too), then it is easy to see that $\mathcal{T}$ is abstracted by $\mathcal{A}_\Delta$ which in turn is abstracted by $\mathcal{A}_\Pi$.

Finally, let $\mathcal{T}$ and $\mathcal{A}$ be transition systems such that $\mathcal{T} \preccurlyeq_\alpha \mathcal{A}$ for some abstraction function α. Then every concrete path $\rho = s_0 s_1 \cdots$ in $\mathcal{T}$ induces a *corresponding* abstract path, denoted $\alpha(\rho)$, in $\mathcal{A}$ given by $\alpha(s_0)\alpha(s_1)\cdots$. Also, if ρ is initial in $\mathcal{T}$, so is $\alpha(\rho)$ in $\mathcal{A}$. It thus follows that:

Proposition 1: If $\mathcal{T}$ and $\mathcal{A}$ are transition systems over a set of propositions P, and $\mathcal{A}$ is an abstraction of $\mathcal{T}$, then:

1) $\mathit{traces}(\mathcal{T}) \subseteq \mathit{traces}(\mathcal{A})$,
2) For any $LTL(P)$ formula φ, if $\mathcal{A} \models \varphi$ then $\mathcal{T} \models \varphi$.

V. Model-Checking

We outline an algorithm for the model-checking problem for a finite-state transition system, following the standard automata-theoretic approach to LTL model-checking by Vardi et al. [24].

We first focus on model-checking FGp formulas, and then show how to generalize this to model-check any LTL(P) formula. Let us fix a finite state transition system $\mathcal{A} = (S, I, \rightsquigarrow, P, \ell)$ and a formula FGp, with $p \in P$. The following is easy to see:

Proposition 2: $\mathcal{A} \models FGp$ iff there is no simple initial lasso in $\mathcal{A}$ of the form (u, t, v), with $t \not\models p$.

This gives us a simple algorithm to model-check $\mathcal{A}$ with respect to FGp. We first find the set of all states T that are reachable from some initial state and that do not satisfy p. Then, for each $t \in T$, we check if t is reachable from t (with a non-zero length path). If such a t exists, then the algorithm returns a path u to t and a path v from t to t, thereby returning a counterexample in the form of a lasso (u, t, v) exhibiting the violation of the propoerty FGp by $\mathcal{A}$. Otherwise, model-checking succeeds.

Now to model-check $\mathcal{A}$ for a general LTL(P) formula φ, we first construct a formula automaton $\mathcal{B}_{\neg\varphi}$ for $\neg\varphi$, which is a (possibly non-deterministic) Büchi automaton which accepts exactly the models of $\neg\varphi$. We assume that $\mathcal{B}_{\neg\varphi}$ is state-labelled, in that each state has an associated label in 2^P. Let $\mathcal{B}_{\neg\varphi} = (Q, Q_0, \Delta, \ell', F)$ where $\Delta \subseteq Q \times Q$ is the non-deterministic transition relation and F the set of final states of the Büchi automaton $\mathcal{B}_{\neg\varphi}$. We now construct the product transition system $\mathcal{A}' = \mathcal{A} \otimes \mathcal{B}_{\neg\varphi} = (S \times Q, I', \to, \{r\}, \ell'')$, where I' is the set $\{(s, q) \mid s \in I, q \in Q_o, \ell(s) = \ell'(q)\}$, the transition relation $\to$ is given by $(s, q) \to (s', q')$ iff $s \rightsquigarrow s'$, $q' \in \Delta(q)$, and $\ell(s') = \ell'(q')$; and $\ell''(s, q) = \{r\}$ if $q \notin F$, and $\{\}$ otherwise. Then it is easy to see that $\mathcal{A}$ satisfies φ iff $\mathcal{A} \otimes \mathcal{B}_{\neg\varphi}$ does not satisfy FGr.

VI. Validation

Given transition systems $\mathcal{T}$ and $\mathcal{A}$, an abstraction function α from $\mathcal{T}$ to $\mathcal{A}$, that is, $\mathcal{T} \preccurlyeq_\alpha \mathcal{A}$, and an initial lasso path $\widehat{\rho}$ in $\mathcal{A}$, the *lasso validation problem* is to determine whether $\widehat{\rho}$ is feasible w.r.t. $\mathcal{T}$; i. e. does there exist an initial path ρ in $\mathcal{T}$ such that $\alpha(\rho) = \widehat{\rho}$. In this section, we give a (semi-)algorithm for the validation problem, which uses the characterization of feasible lasso paths in terms of recurrent sets [12]. The algorithm tries to compute a maximal recurrent set by computing successive preimages of the pivot node along the loop. The preimage sets computed are also returned by the algorithm, for subsequent use in refinement.

Let us fix transition systems $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$ and $\mathcal{A}$, and an abstraction function α, such that $\mathcal{T} \preccurlyeq_\alpha \mathcal{A}$. Let X be a set of states in $\mathcal{T}$ and let v be a finite path in $\mathcal{A}$. We define the *postimage* of X *along* v, denoted $Post_{\mathcal{T},\alpha}(X, v)$, by induction on the length of v as follows:

- $Post_{\mathcal{T},\alpha}(X, \epsilon) = X$, and
- $Post_{\mathcal{T},\alpha}(X, u \cdot t) = Post_{\mathcal{T}}(Post_{\mathcal{T},\alpha}(X, u)) \cap \alpha^{-1}(t)$.

Definition 6.1: We say a lasso (u, t, v) in $\mathcal{A}$ is *feasible* w.r.t. $\mathcal{T}$ and α, if there is an initial path ρ in $\mathcal{T}$ with $\alpha(\rho) = u(tv)^\omega$.

We recall the notion of a recurrent set from [12]:

Definition 6.2: A set of states X in $\mathcal{T}$ is a *recurrent set* w.r.t. a lasso (u, t, v) in $\mathcal{A}$ if $X \subseteq \alpha^{-1}(t)$ and for all $s \in X$, $Post_{\mathcal{T},\alpha}(\{s\}, vt) \cap X \neq \varnothing$.
Note that the empty set is a trivial recurrent set. The following characterization of feasible lassos in terms of recurrent sets is from [12]:

Lemma 6.3 ([12]): A lasso (u, t, v) in $\mathcal{A}$ is *feasible* w.r.t. $\mathcal{T}$ and α iff there exists a recurrent set X w. r. t. (u, t, v), which is reachable via u in $\mathcal{T}$ (i.e $Post_{\mathcal{T},\alpha}(I, u) \cap X \neq \varnothing$).

Our procedure for validation is based on computing maximal recurrent sets:

Definition 6.4: A set of states X in $\mathcal{T}$ is a *maximum recurrent set* w. r. t. a lasso (u, t, v) in $\mathcal{A}$, if X is a recurrent set w. r. t. (u, t, v) and contains all recurrent sets w. r. t. (u, t, v).
A maximal recurrent set w.r.t. (u, t, v) is unique and coincides with the union of all recurrent sets w.r.t. (u, t, v).

Algorithm 1 shows the psuedocode of our proposed procedure. The procedure calls function `RecSet` to compute a *maximum* recurrent X set w. r. t. the given lasso (u, t, v) in $\mathcal{A}$. In case X is empty, the procedure returns “Infeasible”. If X is non-empty, it calls `PathPre` to find the successive pre-sets of X along u. If the final preset intersects with the initial set of states, then it returns “Feasible”, and otherwise it returns “Infeasible”. Assuming `RecSet` correctly returns a maximum recurrent set whenver it does terminate, the algorithm can be seen to be correct: If X is empty, then there can be no recurrent set w. r. t. the given lasso, and by Lemma 6.3 it follows that (u, t, v) is not feasible. If X is non-empty but not reachable along u, then once again by Lemma 6.3 we are justified in returning “Infeasible”. Finally, if X is non-empty and reachable along u, then X is a reachable recurrent set, and we are justified in returning “Feasible”.

We now claim that the procedure `RecSet`, whenever it terminates, computes a maximum recurrent set w. r. t. (u, t, v).

Lemma 6.5: If the `RecSet` function terminates, it returns the maximum recurrent set w. r. t. (u, t, v).
This completes the proof of correctness of Algo. 1.

Algorithm 1: Validate Lasso

```
Input: $\mathcal{T}$, $\mathcal{A}$, $\alpha$, lasso $(u,t,v)$ in $\mathcal{A}$
Output: Whether $u(tv)^\omega$ is feasible in $\mathcal{A}$ w.r.t. $\mathcal{T}, \alpha$;
        along with state splits
1  Function PathPre($X$, $s_1 \cdots s_n$):
2      $X_n = Pre_{\mathcal{T}}(X) \cap \alpha^{-1}(s_n)$
3      foreach $k$ = $n$-1, ..., 1 do
4          $X_k = Pre_{\mathcal{T}}(X_{k+1}) \cap \alpha^{-1}(s_k)$
5      return $X_1, \ldots, X_n$
6
7  Function RecSet($\mathcal{T}$, $\mathcal{A}$, $\alpha$, $(u,t,v)$):
8      Let $tv = n_1 n_2 \cdots n_k$
9      Let $X_1^0, X_2^0, \ldots, X_k^0 :=$
         $\alpha^{-1}(n_1), \alpha^{-1}(n_2), \ldots, \alpha^{-1}(n_k)$
10     $j := 1$
11     repeat
12         $X_1^j, X_2^j, \ldots, X_k^j$ := PathPre($X_1^{j-1}$,
             $n_1 n_2 \cdots n_k$)
13         $j := j + 1$
14     until $X_1^j = X_1^{j-1}$ or $X_1^j = \varnothing$
15     return $\{n_i \mapsto \{X_i^0, \ldots, X_i^j\}\}$
16
17 Let $u = m_1 m_2 \cdots m_l$
18 $Y_1, \ldots, Y_l$ := PathPre($X$, $m_1 m_2 \cdots m_l$)
19 splitmap1 := $m_1 \mapsto Y_1, \ldots, m_l \mapsto Y_l$
20 splitmap2 := RecSet($\mathcal{T}$, $\mathcal{A}$, $\alpha$, (u,t,v))
21 if $X = \varnothing$ then
22     return “Infeasible”, splitmap1,splitmap2
23 if $Y_1 \cap I \neq \varnothing$ then
24     return “Feasible”, splitmap1, splitmap2
25 else
26     return “Infeasible”, splitmap1, splitmap2
```

VII. Refinement

In this section, we define a procedure to compute a refinement of a given abstraction that eliminates an infeasible abstract lasso. First, we formalize our problem.

Definition 7.1: Given transition systems $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$ and $\mathcal{A} = (\widehat{S}, \widehat{I}, \widehat{\rightsquigarrow}, \widehat{P}, \widehat{\ell})$, and abstraction function α, where $\mathcal{T} \preccurlyeq_\alpha \mathcal{A}$, a transition system $\mathcal{R}$ is a refinement of $\mathcal{A}$ w. r. t. $\mathcal{T}$, if there exist abstraction functions α_1 and α_2 such that $\mathcal{T} \preccurlyeq_{\alpha_1} \mathcal{R} \preccurlyeq_{\alpha_2} \mathcal{A}$ and $\alpha_1 \circ \alpha_2 = \alpha$.

Definition 7.2: Given transition systems $\mathcal{T} = (S, I, \rightsquigarrow, P, \ell)$ and $\mathcal{A} = (\widehat{S}, \widehat{I}, \widehat{\rightsquigarrow}, \widehat{P}, \widehat{\ell})$, abstraction function α, where $\mathcal{T} \preccurlyeq_\alpha \mathcal{A}$, and a lasso $\hat{\rho}$ of $\mathcal{A}$, a refinement $\mathcal{R}$ of $\mathcal{A}$ w. r. t. $\mathcal{T}$ is said to eliminate $\hat{\rho}$ if there does not exist a path $\widetilde{\rho}$ in $\mathcal{R}$ such that $\alpha_2(\widetilde{\rho}) = \hat{\rho}$

Let us fix a transition system $\mathcal{T}$ and an abstraction $\mathcal{A} = \mathcal{T}_\Pi$ with $\alpha = \alpha_\Pi$, where Π is a partition of S. Let $\hat{\rho}$ be a lasso in $\mathcal{A}$ given by (u, t, v). Suppose Algorithm 1 returns ”Infeasible”. Our goal is to find a refinement Π' of Π, such that the refinement $\mathcal{R} = \mathcal{T}_{\Pi'}$ eliminates $\hat{\rho}$.

Let $u = m_1 \cdots m_l$ and $tv = n_1 \cdots n_k$. Let *splitmap1* = $\{m_1 \mapsto Y_1, \ldots, m_l \mapsto Y_l\}$ and *splitmap2* = $\{n_i \mapsto \{X_i^0, \ldots, X_i^j\}\}_{i=1}^k$ as returned by the validation algorithm. The refinement Π' will be the same as Π except that the blocks $\alpha^{-1}(m_i)$, $1 \le i \le l$ and $\alpha^{-1}(n_i)$, $1 \le i \le k$ will be split as follows: For $1 \le i \le l$, $\alpha^{-1}(m_i)$ will be split into Y_i and $\alpha^{-1}(m_i) \backslash Y_i$. For $1 \le i \le k$, $\alpha^{-1}(n_i) = X_i^0$ will be split into $Z_i^0, Z_i^1, \cdots, Z_i^j$, where $Z_i^s = X_i^s / X_i^{s+1}$ for $0 \le s < j$ and $Z_i^j = X_i^j$.

Note that Π' is a refinement of Π, and hence $\mathcal{R} = \mathcal{T}_{\Pi'}$ is a refinement of $\mathcal{A}$ with respect to $\mathcal{T}$. We claim that $\mathcal{R}$ eliminates the lasso (u, t, v). We make a few observations:

- For $1 \le s \le j$, $X_i^s = Pre_{\mathcal{T}}(X_{i+1}^s) \cap \alpha^{-1}(n_i)$ for $1 \le i < k$ and $X_k^s = Pre_{\mathcal{T}}(X_1^{s-1}) \cap \alpha^{-1}(n_k)$.
- In $\mathcal{R}$, for $1 \le s \le j$, there are no transitions from Z_i^{s-1} to X_{i+1}^s for $1 \le i < k$, and from Z_k^{s-1} to X_1^{s-1}.
- Hence, for $0 \le s < j$, the only possible transitions from Z_i^s are to $Z_{i+1}^{s'}$ where $s' \le s$ for $1 \le i < k$, and from Z_k^s to $Z_1^{s'}$ where $s' < s$.

The above observations imply that starting at any block Z_i^s, where $0 \le s < j$, the paths in $\mathcal{R}$ following the loop tv will be finite (will follow the loop at most j times).

Consider the case where $X_i^j = \varnothing$, that is, $Z_1^j = \varnothing$. Note that in this case the path $Z_1^j, Z_2^j, \cdots, Z_k^j$ will not exist in $\mathcal{R}$, because some of these sets will be empty. That is, there are no infinite paths corresponding to tv in $\mathcal{R}$.

Next, let us consider the case where $X_1^j \neq \varnothing$. In this case, the only infinite paths corresponding to tv in $\mathcal{R}$ starting from t will start from $X_1^j = Z_1^j$. However, there are no paths in $\mathcal{R}$ that follow u starting from I that reach X_1^j (which is the maximum recurrent set).

The following theorem summarizes the correctness of the refinement method, whose proof is based on the above argument.

Theorem 7.3: The transition system $\mathcal{R} = \mathcal{T}_{\Pi'}$ eliminates the lasso (u, t, v) if it is infeasible.

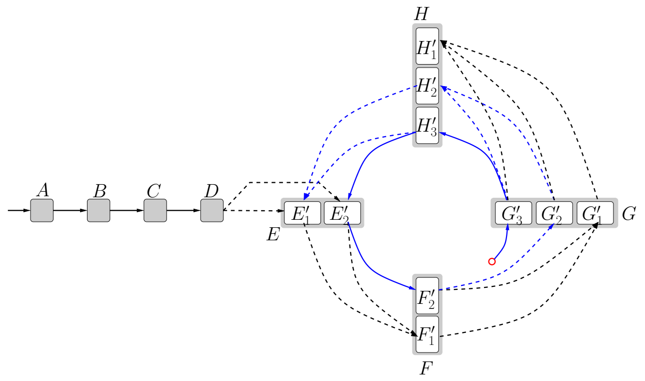


Fig. 3: Illustration of refinement

Let us illustrate the concepts through the example in Figure 3. The validation algorithm starts with the sets $X_1^0 = E$, $X_2^0 = F$, $X_3^0 = G$ and $X_4^0 = H$ corresponding to tv. It iteratively computes the sets Z_i^s for $1 \le i \le 4$ and $1 \le s \le j$, where j is the last iteration. Specifically, $E_s = Z_1^s$, $F_s = Z_2^s$, $G_s = Z_3^s$ and $H_s = Z_4^s$, and correspond to the blocks in the refinement. Note that the transitions are only from E_s to $E_{s'}$, F_s to $F_{s'}$, G_s to $G_{s'}$, where $s' \le s$, and from H_s to $H_{s'}$ where $s' < s$. So, paths in the refinement go through the E, F, G and H sets with the same or lower s and then switch back to E with a strictly lower s and so on. Hence, there are no infinite paths through the cycle in the refinement. If E_j in the last iteration is empty, then there will be no finite paths from E_j that returns to E and paths starting from E_s for $s < j$ will not follow the cycle infinitely either as argued above. If E_j is not empty, then the only way to have an infinite path is to cycle at level j, as switching to lower levels will force the paths to exit the loop in a finite number of cycles as explained above. Hence, if E_j is not reachable from initial state, then there is no instantiation of the lasso in the refinement.

VIII. Top-Level Algorithm

We now present the top-level algorithm that puts together the pieces of the counter-example guided abstraction refinement (CEGAR) procedure. Our broad objective is to solve the model-checking problem: Given a transition system $\mathcal{T}$ and an LTL property φ, check whether $\mathcal{T} \models \varphi$. Algorithm 2 summarizes the overall CEGAR algorithm. Inputs to the algorithm are the transition system $\mathcal{T}$, and the temporal property specified as an LTL formula φ.

Algorithm 2: CEGAR ($\mathcal{T}$, φ)

```
Input: Transition System $\mathcal{T}$, LTL property $\varphi$
Output: whether $\mathcal{T}$ satisfies $\varphi$
1  preds = getpredicates($\varphi$)
2  Construction partition $\Pi$ from preds
3  Construction abstraction $\mathcal{A} = \mathcal{T}_\Pi$
4  loop
5      res, cexlasso = modelCheck($\mathcal{A}$, $\varphi$)
6      if res is "yes" then
7          return "Property holds"
8      ans, map1, map2 = validate
        lasso($\mathcal{T}, \mathcal{A}, \alpha_\Pi, cexlasso$)
9      if ans == "Feasible" then
10         extract cex from map1, map2
11         return "Property violated", cex
12     Extract $\Pi'$ from map1, map2
13     Construct refinement $\mathcal{A} = \mathcal{T}_{\Pi'}$
```

First, we extract the predicates from the property φ and use that to define our initial parition Π and the initial abstraction $\mathcal{A} = \mathcal{T}_\Pi$. Each iteration of the CEGAR loop starts with model-checking $\mathcal{A}$ with respect to φ. If $\mathcal{A}$ satisfies φ, then from the virtue of the abstraction, we can also conclude that $\mathcal{T}$ satisfies the property. Otherwise, the model-checking algorithm returns an abstract lasso, cexlasso in $\mathcal{A}$. Validation algorithm then checks if cexlasso is valid, and returns two maps *map1* and *map2* that contain information regarding splitting the abstract states. If cexlasso is valid, one can extract a concrete path from *map1* and *map2*. Otherwise, these maps are used to construct a refinement Π', leading to the refined system $\mathcal{A} = \mathcal{T}_{\Pi'}$. The next iteration then proceeds with the new abstraction $\mathcal{A}$.

IX. Experimental Evaluation

In this section, we present the details of our evaluation of our counter-example guided abstraction refinement approach on robot gridworld benchmarks and comparison with the tools MUVAL and MONDOK.

A. Implementation

We have implemented our approach in a Python toolbox which we call REC-CEGAR. The tool takes as input a logical description of the transition system as a formula over linear real arithmetic, along with specifications of the form FG Region or GF Goal. The abstraction construction, and preset computations are performed using satisfiability solving and quantifier elimination using the SMT solver Z3 [25]. We have also implemented the CEGAR approach of Mondok et al. [14] and use the MATHSAT solver [26] as an interpolation engine. We refer to this implementation as MONDOK.

B. Benchmarks

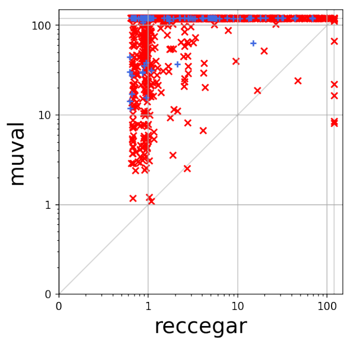
We consider three set of benchmarks all based on the robot gridworld setting as in Sec. II. Specifically, we consider $n \times n$ grids for different values of n, and choose between 9 directions corresponding to north, south, east, west, northeast, northwest, southeast, southwest and stationary.

- Set 1 consists of 48 hand-crafted examples to cover a wide range of scenarios including satisfaction and violation of FG and GF properties.
- Set 2 consists of gridworld scenarios for grid sizes $n \times n$ where $n = 3, 4, 9$. For each grid size, we consider 100 scenarios generated by either randomly choosing among 5 directions - north, south, east, west, and stationary, or 9 directions that include northeast, northwest, southeast, and southwest. And for each grid, we consider an FG region and GF goal property by randomly selecting one cell to be the goal or the region. This amounts to a total of 1200 benchmarks.
- Set 3 is similar to Set 2, except that the directions of the boundary cells are biased to point inwards with probability 0.9, so as to obtain interesting scenarios in which the system remains within the grid.

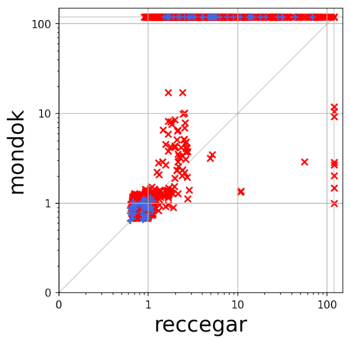
C. Tools compared with

We compare our approach with the tools MUVAL and MONDOK. MUVAL [13] is a state-of-the-art tool based on first-order fixpoint logic similar to Mu-Calculus. MONDOK [14] is a tool based on predicate abstraction and refinement based on interpolation. We run each of the tools on the three sets of benchmarks explained above, and measure the number of benchmarks on which each of the tools proves/disproves as well as the time taken to arrive at the results. We consider a timeout of 120 s.

Other related tools are Ultimate LTL Automizer [8] and RLIVE-INF [22]. Ultimate LTL Automizer cannot handle real variables and RLIVE-INF natively handles only FG properties. Also, RLIVE-INF is a recent work. We plan to perform comparisons on subclass of systems and properties that these tools handle in the future.



(a) REC-CEGAR vs MUVAL



(b) REC-CEGAR vs MONDOK

Fig. 4: Comparison on execution time

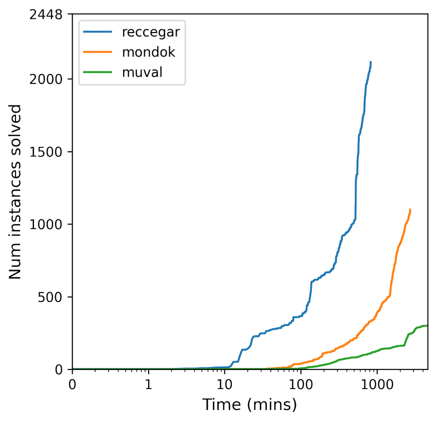


Fig. 5: Comparison on number of instances solved

D. Evaluation

Our experimental evaluation is summarized in the Fig. 4a and Fig. 4b. Our experiments were performed on an Intel Core i7 CPU with 8 cores and 32 GB of RAM. Each point in the plot records the time taken by the tools on a specific benchmark with the x-value representing the time taken by the tool named along the x-axis and similarly for the y-value. Note that our tool REC-CEGAR is depicted along the x-axis, hence, the points above the diagonal refer to benchmarks on which our tool performed better than the tool being compared to. A point on the ceiling refer to benchmarks on which the tool on the y-axis timed out, while the points on the right wall refer to benchmarks on which our tool timed out. A blue plus symbol denotes when one of the tools is able to prove and a red cross symbol denotes when one of the tools disproves the benchmark.

As seen from both the plots, a majority of the points are above the diagonal and on the ceiling, and very few of them are on the wall. Hence, our tool times out on very few benchmarks compared to competing tools, and for the ones on which both terminate within timeout, our tool performs better on many more benchmarks than the competing tools.

Finally, we present the cumulative number of instances solved by each of the tools against time in Fig. 5. Again, we observe that REC-CEGAR is able to solve more instances compared to both MUVAL and MONDOK within any given time and indeed completes solving 2116 of the 2448 within 824 min as compared to 300 by MUVAL in 4502 min and 1100 by MONDOK in 2719 min.

X. CONCLUSION

In this work we have proposed a classical CEGAR-style approach for verifying temporal properties of infinite-state systems. Our technique relies on computing maximum recurrent sets for a lasso-shaped abstract paths, which helps in both counter-example validation and downstream refinement of the abstraction. In the future, we would like to explore logical techniques for learning recurrent sets and invariants, and target the class of neural network controlled discrete-time systems. It would also be interesting to explore the applicability of somewhat orthogonal techniques like [8] for these classes of systems.

REFERENCES

[1] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: a software toolbox for receding horizon temporal logic planning," in *Proc.14th ACM Intl. Conf. Hybrid Sys.: Comput. Control (HSCC 2011), Chicago, USA*. ACM, 2011, pp. 313–314.

[2] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with TuLiP: The Temporal Logic Planning toolbox," in *2016 IEEE Conf. Cont. Applications (CCA 2016), Buenos Aires, Argentina*. IEEE, 2016, pp. 1030–1041.

[3] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Trans. Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[4] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Intl. Conf. Computer Aided Verification (CAV 2000)*. Springer, 2000, pp. 154–169.

[5] N. Spooner, "Evaluation and Benchmarking for Robot Motion Planning Problems Using TuLiP," Aug. 2023.

[6] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Trans. Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[7] B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Y. Vardi, "Proving that programs eventually do something good," in *Proc. 34th Symp. Princ. Prog. Lang. (POPL 2007)*, M. Hofmann and M. Felleisen, Eds. ACM, 2007, pp. 265–276.

[8] D. Dietsch, M. Heizmann, V. Langenfeld, and A. Podelski, "Fairness Modulo Theory: A new approach to LTL Software Model Checking," in *Proc. 27th Intl. Conf. Computer Aided Verification (CAV 2015), San Francisco, USA*, ser. LNCS, vol. 9206. Springer, 2015, pp. 49–66.

[9] M. Brockschmidt, B. Cook, S. Ishtiaq, H. Khlaaf, and N. Piterman, "T2: Temporal Property Verification," in *22nd Intl. Conf. Tools and Alg. Constr. Analysis of Systems (TACAS 2016), Eindhoven, Netherlands*, ser. LNCS, vol. 9636. Springer, 2016, pp. 387–393.

[10] R. Alur, T. Dang, and F. Ivančić, "Counterexample-guided predicate abstraction of hybrid systems," *Theoretical Computer Science*, vol. 354, no. 2, pp. 250–271, 2006, tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397505008662

[11] P. Prabhakar, P. S. Duggirala, S. Mitra, and M. Viswanathan, "Hybrid automata-based CEGAR for rectangular hybrid systems," *Formal Methods Syst. Des.*, vol. 46, no. 2, pp. 105–134, 2015. [Online]. Available: https://doi.org/10.1007/s10703-015-0225-4

[12] A. Gupta, T. A. Henzinger, R. Majumdar, A. Rybalchenko, and R. Xu, "Proving non-termination," in *Proc. 35th Symp. Princ. Prog. Lang. (POPL 2008), San Francisco, USA*. ACM, 2008, pp. 147–158.

[13] H. Unno, T. Terauchi, Y. Gu, and E. Koskinen, "Modular primal-dual fixpoint logic solving for temporal verification," *Proc. ACM Prog. Languages (POPL 2023)*, vol. 7, no. POPL, pp. 2111–2140, 2023.

[14] M. Mondok and A. Vörös, "Abstraction-based model checking of linear temporal properties," in *Proc. 27th PhD MiniSymposium*. Budapest University of Technology and Economics, 2020, pp. 29–32.

[15] A. Biere, C. Artho, and V. Schuppan, "Liveness checking as safety checking," in *7th Work. Formal Meth. Ind. Critical Sys. (FMICS 2002), Málaga, Spain*, ser. ENTCS, vol. 66, no. 2. Elsevier, 2002, pp. 160–177.

[16] A. R. Bradley, F. Somenzi, Z. Hassan, and Y. Zhang, "An incremental approach to model checking progress properties," in *Conf. Formal Methods in Computer-Aided Design (FMCAD 2011), Austin, USA*. FMCAD Inc., 2011, pp. 144–153.

[17] K. Claessen and N. Sörensson, "A liveness checking algorithm that counts," in *Formal Methods in Computer-Aided Design (FMCAD 2012), Cambridge, UK*. IEEE, 2012, pp. 52–59.

[18] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "Verifying LTL properties of hybrid systems with k-liveness," in *Proc. 26th Intl. Conf. Computer Aided Verification (CAV 2014) Vienna, Austria*, ser. LNCS, vol. 8559. Springer, 2014, pp. 424–440.

[19] B. Cook, "Automatically proving program termination," in *Proc. 19th Intl. Conf. (CAV 2007), Berlin, Germany*, ser. LNCS, vol. 4590. Springer, 2007, p. 1.

[20] B. Cook, A. Podelski, and A. Rybalchenko, "Proving program termination," *Commun. ACM*, vol. 54, no. 5, pp. 88–98, 2011.

[21] J. Daniel, A. Cimatti, A. Griggio, S. Tonetta, and S. Mover, "Infinite-State Liveness-to-Safety via Implicit Abstraction and Well-Founded Relations," in *Proc. 28th Intl. Conf. Computer-Aided Verification (CAV 2016), Toronto, Canada*, ser. LNCS, vol. 9779. Springer, 2016, pp. 271–291.

[22] A. Cimatti, A. Griggio, C. Johannsen, K. Y. Rozier, and S. Tonetta, "Infinite-state liveness checking with rlive," in *Proc. 37th Intl. Conf. Computer Aided Verification (CAV 2025), Zagreb, Croatia*, ser. LNCS, vol. 15931. Springer, 2025, pp. 215–236.

[23] Y. Xia, A. Cimatti, A. Griggio, and J. Li, "Avoiding the Shoals - A New Approach to Liveness Checking," in *Proc. 36th Intl. Conf. Computer Aided Verification (CAV 2024), Montreal, Canada*, ser. LNCS, vol. 14681. Springer, 2024, pp. 234–254.

[24] P. Wolper, M. Y. Vardi, and A. P. Sistla, "Reasoning about infinite computation paths (extended abstract)," in *Proc. Symp. Foundations of Computer Science (FOCS 1983), Tucson, USA, 1983*. IEEE Computer Society, 1983, pp. 185–194.

[25] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. Intl. Conf. Tools and Alg. Constr. Analysis Sys. (TACAS 2008)*. Springer, 2008, pp. 337–340.

[26] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The Mathsat5 SMT Solver," in *Proc. Intl. Conf. Tools and Alg. Constr. Analysis Sys. (TACAS 2013)*. Springer, 2013, pp. 93–107.