2018 IEEE Conference on Decision and Control (CDC)
Miami Beach, FL, USA, Dec. 17-19, 2018

Lazy Controller Synthesis using Three-valued Abstractions for Safety and Reachability Specifications*

Omar Hussien and Paulo Tabuada

Abstract— **The synthesis of correct-by-design control software is a promising direction to address the well known difficulties in formally verifying complex cyber-physical systems. Despite the promise of this approach, it is currently limited to small systems since it typically requires the computation of a finite-state abstraction whose size grows exponentially with the number of continuous states. In this paper we present a new way to tackle the lack of scalability of control software synthesis by adopting a lazy controller synthesis approach. Instead of synthesizing a controller using a precomputed abstraction of the full system, the abstraction is computed lazily as needed for safety and reachability specifications. We illustrate, through different examples, how this lazy approach significantly reduces the total time required for the synthesis of correct-by-design controllers.**

I. Introduction

As cyber-physical systems (CPS) become more complex, the verification of CPS control software becomes extremely challenging. One way to alleviate the need for verification is to adopt a correct-by-design approach. By synthesizing the control software along with a proof of correctness, the correct-by-design approach eliminates, or greatly reduces, the need for verification. A common correct-by-design approach is based on the computation of a finite-state abstraction of the control system. Given a specification expressed in some formal language, a controller that enforces this specification on the abstraction is first synthesized and then refined to a controller enforcing the same specification on the original system. Construction of abstractions for incrementally input-to-state stable control systems was introduced in [20], [23] and [22]. In [30] the authors showed how to compute finite-state abstractions if incremental input-to-state stability does not hold. A variety of software tools for abstraction based correct-by-design controller synthesis have been developed and include PESSOA [14], CoSyMa [16], TuLiP [29], and SCOTS [27]. However, the computation of abstractions scales exponentially with the number of variables in the differential equation model of the system to be controlled. Hence, it becomes challenging to compute abstractions for large systems.

One way to mitigate this problem is to compute abstractions compositionally. Recent efforts toward computing abstractions of control systems compositionally are based on exploiting the structure of the control system: see [18] for results exploiting linearity, see [10] for results exploiting monotonicity, see [21] and [11] for results exploiting the discrete-time nature, see [17] for results exploiting the switched nature, and [13] for results exploiting the stochastic nature of control systems. Moreover, nonlinear control systems that can be decomposed into smaller subsystems with different types of interconnections were addressed in [24], [15], [12], [6], [9]. Results that are based on computing an initial coarse abstraction which is gradually refined on-demand can be found in [7] and [4]. The authors in [7] extended the method of counterexample-guided refinement (CEGAR) [3] from verification to controller synthesis. In [4] a specification-guided approach was introduced using three-valued abstraction refinement where under- and over-approximations of the winning region are computed and denoted by must-win and may-win states, respectively. If the controller synthesized for the abstraction that under-approximates the concrete system is not able to enforce the specification from the initial states, refinement is done by splitting the may-win states. Similarly, multi-layered abstraction approaches where introduced in [8] and [5]. In [8], the authors simultaneously maintain several abstractions with different precisions. Controller synthesis starts from the coarser abstraction and moves on to finer precision if needed depending on the given control problem. In [5], the authors present a multi-layered approach for incrementally stable switched systems by allowing transitions to have different durations. The coarser abstraction is used for transitions with longer duration whereas a finer abstraction is used for transitions with shorter duration. Computation of abstractions on the fly was presented in [26] for discrete-time nonlinear systems to synthesize controllers that drive the system into a desired set while minimizing a given cost. The authors in [19] presented an approach to integrate controller synthesis with the computation of the abstraction for nonlinear control systems that are incrementally input-to-state stable [2].

In this paper, we present a novel, and orthogonal to previous work, way to improve the scalability of abstraction-based synthesis by adopting a lazy approach. Instead of synthesizing a controller using a precomputed abstraction of the full system, we lazily compute the fragment of the abstraction that is required for controller synthesis. In Sections III and IV, we discuss how to synthesize a controller for safety and reachability specifications, respectively, by

*This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA and by the NSF CPS Awards 1239085 and 1645824.

O. Hussien and P. Tabuada are with the Department of Electrical Engineering, University of California Los Angeles, Los Angeles, CA, ohussien@ucla.edu, tabuada@ee.ucla.edu.

lazily computing the abstraction as needed. Similarly to the approach in [4], we use a three-valued abstraction refinement approach. However, may-states in our approach denote the states for which the set of successors has not yet been computed for all inputs. Instead of computing a coarse abstraction and gradually refining it by splitting may-states, we refine the transitions stemming from may-states. This is accomplished by computing new transitions from a may-state that may make it a must-state if an input is found for which all its successors land on the desired set. Hence, our approach refines transitions while the approach in [4] and [8] refines states. We present lazy controller synthesis algorithms for safety and reachability specifications and we illustrate through different examples how the lazy approach is significantly faster as compared to controller synthesis using a precomputed abstraction.

The remainder of the paper is organized as follows. Section II introduces notation and the definitions of control systems and abstractions that we consider in this paper. Our proposed algorithms for safety and reachability specifications appear in Section III and Section IV, respectively. We illustrate the benefits of our approach through examples in Section V. Conclusions follow in Section VI.

II. Control systems and Abstractions

A. Notation

We use $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}_0^+$ to denote the set of natural integer, real, positive and nonnegative real numbers. The discretization of $S \subset \mathbb{R}^n$ is defined by:

$$[S]_\alpha = \{s \in S | s_i = k_i\alpha, k_i \in \mathbb{Z}, i = 1, \cdots, n\},$$

where $\alpha \in \mathbb{R}^+$ is the discretization parameter. Given a measurable function $f : \mathbb{R}_0^+ \to \mathbb{R}^n$, we denote the (essential) supremum (ess) $\sup_{t\in\mathbb{R}_0^+} \|f(t)\|$ by $\|f\|_\infty$. We use $\wedge$, $\vee$, and $\neg$ to denote the logical conjunction, disjunction, and negation respectively. The temporal operators always and eventually are denoted by $\Box$ and $\Diamond$, respectively. Given sets A and B, we use $A-B$ to denote the set of all elements in A and not in B. We use the notation $x :\in X$ to represent the assignment to x of an element of X.

B. Control Systems

In this paper we work with continuous time control systems defined as follows.

Definition 2.1: A control system $\Sigma = (\mathbb{R}^n, U, \mathcal{U}, f)$ consists of:

- the state space $\mathbb{R}^n$;
- the input set $U \subseteq \mathbb{R}^m$;
- the admissible input curves $\mathcal{U}$, a subset of all the piecewise continuous functions of time from intervals of the form $]a, b[\subset \mathbb{R}$ to U with $a < 0 < b$;
- the locally Lipschitz continuous map $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ defining the dynamics of the system.

C. Abstraction

We compute an abstraction of a given control system by discretizing the states, the inputs, and time.

Definition 2.2: Given the control system $\Sigma = (\mathbb{R}^n, U, \mathcal{U}, f)$ and the discretization parameters (τ, η, μ), where $\tau \in \mathbb{R}^+$ is the sampling time, $\eta \in \mathbb{R}^+$ is the state space discretization, and $\mu \in \mathbb{R}^+$ is the input discretization, the abstraction $S(\Sigma)$ of Σ is the triple (X, U, T) consisting of:

- $X = [\mathbb{R}^n]_\eta$;
- $U = [\mathbb{R}^m]_\mu$;
- $T : X \times U \times X \to \{0, 1/2, 1\}$,

where the map T describes the transition relation in the abstraction. Note that the set of states X and the set of inputs U in Def. 2.2 have infinitely many elements. However, states and inputs of most CPS hold physical meaning and can not exceed certain values under normal operation, i.e., they are always restricted to compact sets. Hence, the abstraction presented in Def. 2.2 becomes finite once we replace $\mathbb{R}^n$ and $\mathbb{R}^m$ with a compact set of states and inputs, respectively.

The proposed lazy algorithms for safety and reachability specifications in Sections III and IV start with an empty transition relation, i.e., an abstraction without transitions, and compute the transitions lazily as needed for controller synthesis. For the purpose of analyzing the algorithms, we assume that all the relevant states have already been computed. In practice, however, we store the abstractions on efficient data structures, e.g., Binary Decision Diagrams, and states for which no transitions have been computed are not stored in memory. The existence of a transition from a state x to another state x' using an input u, is recorded by having T map (x, u, x') to 1. The absence of a transition from a state x to another state x' using an input u, is recorded by having T map (x, u, x') to 0. In addition to these two cases, we use $1/2$ to record that transitions from state x under input u have not yet been computed. This means that every state x' is a potential successor of x under u and is recorded as $T(x, u, x') = 1/2$ for every x'.
We denote the set of u-successors of a state x by $\mathrm{Post}_u(x)$ and we assume that $\mathrm{Post}_u(x)$ is always nonempty. Note that, if an abstraction is precomputed, then $T(x, u, x') \in \{0, 1\}$ for all $x \in X$, $u \in U$ and $x' \in X$.

III. Controller Synthesis for Safety Specifications

In this section we present two algorithms to synthesize controller for safety specifications, i.e., always stay in a desired set for all time. The first algorithm is classical and assumes the existence of an abstraction. The second algorithm is the first contribution of this paper and rather than requiring an existing abstraction, it lazily computes the fragment of the abstraction that is needed for controller synthesis. We show that the lazy algorithm terminates and upon termination returns the largest set of states for which

there exists a control input that enforces the state to stay in K. This set is also known as the largest controlled invariant set in K. Once this set is computed, controller synthesis is straightforward as it amounts to choosing an input that forces the system to remain in the controlled invariant set. Existence of such input is guaranteed by notion of controlled invariant set. Accordingly, we present algorithms that compute this set.

A. Classical Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ. Given an "always K" specification for a set $K \subseteq X$, denoted by $\square K$, Algorithm 1 returns the set of states for which there exists a control input that enforces the specification, denoted by $[\![\square K]\!]$. The computation of $[\![\square K]\!]$ makes use of the set $\mathrm{Pre}_{\mathrm{T}}(Q)$ defined by:

$$\mathrm{Pre}_{\mathrm{T}}(Q) = \{x \in X | \exists u, \forall x' \in X, T(x,u,x') = 1 \Rightarrow x' \in Q\}.$$

In Algorithm 1, the set Q is updated until a fixed-point is reached. Upon termination, the set Q identifies the set $[\![\square K]\!]$.

B. Lazy Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ. Given the specification $\square K$, we propose Algorithm 2 to lazily compute $[\![\square K]\!]$. Algorithm 2 starts with a transition relation T' for which no transition has yet been computed, i.e., $T'(x,u,x') = 1/2$ for all $x \in X$, $u \in U$ and $x' \in X$. Given that T is a precomputed transition relation for $S(\Sigma)$, we define the sets $\underline{\mathrm{Pre}}_{\mathrm{T}}(V)$ and $\overline{\mathrm{Pre}}_{\mathrm{T}}(V)$ to be an under- and over-approximation of the set $\mathrm{Pre}_{\mathrm{T}}(V)$, respectively, as follows:

$$\begin{aligned}\underline{\mathrm{Pre}}_{\mathrm{T}}(V) =& \mathrm{Pre}_{\mathrm{T}'}(V),\\ \overline{\mathrm{Pre}}_{\mathrm{T}}(V) =& \{x \in X | \exists u \in U, \forall x' \in X, T'(x,u,x') = 1/2\}\\ & \cup\ \mathrm{Pre}_{\mathrm{T}'}(V),\end{aligned}$$

where T' is any over-approximation of the transition relation T. By an over-approximation of T we mean any transition relation T' for which $T(x,u,x') = 1 \Rightarrow T'(x,u,x') \geq 1/2$. $\underline{\mathrm{Pre}}_{\mathrm{T}}$ is an under-approximation of $\mathrm{Pre}_{\mathrm{T}}$ since every transition mapped by T' to 1 is also mapped by T to 1 although there might be transitions mapped to 1 by T that are mapped to $1/2$ by T'. Conversely, $\overline{\mathrm{Pre}}_{\mathrm{T}}$ is an over-approximation of $\mathrm{Pre}_{\mathrm{T}}$ since every transition mapped by T to 1 is mapped by T' to 1 or $1/2$ and all such transitions are used in computing $\overline{\mathrm{Pre}}_{\mathrm{T}}$.

Algorithm 1 Computation of $[\![\square K]\!]$.

```
1: function STAY(T, K)
2:     Q ← K
3:     repeat
4:         Q' ← Pre_T(Q)
5:         Q ← Q' ∩ Q
6:     until fixed point on Q is reached
7:     return Q
```

Algorithm 2 Lazy Computation of $[\![\square K]\!]$.

```
1: function LAZYSTAY(T', K)
2:     V ← K
3:     repeat
4:         V_under ← Pre_T(V)
5:         V_over ← Pre_T(V)
6:         V ← V_over ∩ V
7:         T' ← Refine_N(T', V_over − V_under)
8:     until fixed point on V is reached and V_over = V_under
9:     return V
```

We define the Refine_N function in Algorithm 3, where N denotes the number of states that will be refined. The proof of correctness and termination of Algorithm 2 does not depend on the value of N. However, the choice of N can affect the performance significantly. We return to this point in Section V in the context of different examples.

Now we show that Algorithm 2 always terminates and upon termination it returns the largest controlled invariant set in K. We use Lemma 3.1 to show termination and Lemma 3.2 to show that Algorithm 2 computes the largest controlled invariant set in K, i.e., the set $[\![\square K]\!]$.

Lemma 3.1: If the set of states X and the set of inputs U are finite, Algorithm 2 terminates in finite time.

Lemma 3.2: If Algorithms 1 and 2 run on the same input and the set of states X and the set of inputs U are finite, we have $Q = V$ upon termination of each Algorithm.
The next result summarizes the consequences of Lemmas 3.1 and 3.2.

Theorem 3.3: Given that the set of states X and the set of inputs U are finite, Algorithm 2 terminates in finite time and upon termination returns the largest controlled invariant subset of K.

Algorithm 3 Abstraction Refinement.

```
1: function Refine_N(T',V)
2:     for i := 1, ··· , N do
3:         x :∈ V
4:         V ← V \ {x}
5:         u :∈ {u ∈ U | ∀x' ∈ X, T'(x,u,x') = 1/2}
6:         for each x' ∈ X do
7:             T'(x,u,x') ← 1 if x' ∈ Post_u(x)
8:             T'(x,u,x') ← 0 if x' ∉ Post_u(x)
9:     return T'
```

IV. Controller Synthesis for Reachability Specifications

In this section we present two algorithms to synthesize a controller for reachability specifications, i.e., eventually reach a desired set. The first algorithm is classical and uses a precomputed abstraction. The second algorithm is the second

contribution of this paper which lazily computes fragment of the abstraction on the fly as needed for controller synthesis.

A. Classical Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ. Given an "eventually K" specification, denoted by $\Diamond K$, Algorithm 4 returns $[\![\Diamond K]\!]$.

B. Lazy Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ for which no transition has yet been computed, i.e., $T'(x,u,x') = 1/2$ for all $x \in X$, $u \in U$ and $x' \in X$. Given the specification $\Diamond K$, we propose Algorithm 5 to compute $[\![\Diamond K]\!]$. Note that we use the same Refine_N function defined by Algorithm 3.

Since the reachability problem is dual to the safety problem [28], the proof of termination and correctness of Algorithm 5 follows similarly to Lemmas 3.1 and 3.2, respectively. Note that in Algorithm 2, V is an over-approximation of Q in Algorithm 1, whereas in Algorithm 5, V is an under-approximation of Q in Algorithm 4.

Theorem 4.1: Algorithm 5 terminates in finite time and upon termination returns the largest reachable set towards K.

Algorithm 4 Computation of $[\![\Diamond K]\!]$.

function REACH(T, K)
  $Q \leftarrow K$
  repeat
    $Q' \leftarrow \mathrm{Pre}_{\mathrm{T}}(Q)$
    $Q \leftarrow Q' \cup Q$
  until fixed point on Q is reached
  return Q

Algorithm 5 Lazy Computation of $[\![\Diamond K]\!]$.

function LAZYREACH(T', K)
  $V \leftarrow K$
  repeat
    $\underline{V} \leftarrow \underline{\mathrm{Pre}}_{\mathrm{T}}(V)$
    $\overline{V} \leftarrow \overline{\mathrm{Pre}}_{\mathrm{T}}(V)$
    $V \leftarrow \underline{V} \cup V$
    $T' \leftarrow \mathrm{Refine}_N(T', \overline{V} - \underline{V})$
  until fixed point on V is reached and $\overline{V} = \underline{V}$
  return V

V. EXPERIMENTAL RESULTS

In this section we illustrate our results on two different examples. We synthesize controllers for each example using the lazy algorithms and compare them with controllers that we obtain using PESSOA [14] and SCOTS [27]. In the first example, we compare our results using a unicycle system for safety and reachability specifications. In the second example we synthesize a controller for a kneed biped robot which appeared in [1] for safety specifications. The lazy algorithms were implemented in C++ and all the computations were done on a 3.4 GHz iMac with 32GB of RAM.

A. Unicycle navigation example

Consider the unicycle vehicle, which can be modeled by:

$$\dot{x} = v\cos(\theta) \quad (1)$$

$$\dot{y} = v\sin(\theta) \quad (2)$$

$$\dot{\theta} = \omega, \quad (3)$$

where (x, y) denotes the position of the vehicle, θ denotes its orientation, and the control inputs v, ω denote the linear and angular velocities, respectively. We used Algorithms 2 and 5 to synthesize a controller that should always avoid the red obstacles and eventually reach a desired green area shown in Fig. 1. This was performed in 2 steps. In the first step we synthesized a controller that avoids collisions with the obstacles using Alg. 2. In the second step we used the controlled invariant set computed in step 1 as the domain over which we solved the reachability problem for the green area using Alg. 5. The state space and input space discretization parameters used were $\eta = 0.1$ and $\mu = 0.1$, respectively, whereas we used $\tau = 0.5$ for the sampling time. Fig. 1 shows the closed loop simulation results for the unicycle model using the synthesized controller for "eventually", reach the desired area, while "always", avoid the obstacles, specifications. We also computed abstractions of this model, with the same parameters, and synthesized a controller for the same specifications, using the MATLAB toolbox PESSOA [14] as well as SCOTS [27]. Tables I and II list a comparison of PESSOA, SCOTS and our lazy safety and reachability algorithms, respectively, using different values of N, where N determines how many states are refined each time the Refine_N function is executed, t_{abs} and t_{syn} are the time to compute the abstraction and synthesize the controller, respectively, in seconds. We observe in Tab. I that when $N = 2000$, we can achieve a speedup of up to 4 and 85 times compared to SCOTS and PESSOA, respectively. In Tab. II, when $N = 1000$, we can achieve a speedup of up to 3 and 63 times compared to SCOTS and PESSOA, respectively. Also, we observe that the choice of N greatly affects the performance of the lazy algorithms. Note that t_{abs} has the same value for Algorithm 1 and Algorithm 4 for PESSOA, and SCOTS, because they synthesize a controller by computing the same abstraction for different specifications.

B. Biped robot example

Consider the kneed biped robot model, shown in Fig. 2, and given by:

$$M(\theta)\ddot{\theta} + C(\dot{\theta},\theta)\dot{\theta} + G(\theta) = Bu, \quad (4)$$

where $\theta = (\theta_1, \theta_2, \theta_3) \in \mathbb{S}^3$, $M(\theta)$ is the inertia matrix, $C(\dot{\theta},\theta)$ is the Coriolis matrix, $G(\theta)$ is the gravity vector, and the matrix B maps the torques vector u to generalized

	N	t_{abs}	t_{syn}
Algorithm 1 (PESSOA)	-	12105	60
Algorithm 1 (SCOTS)	-	530	20
Algorithm 2	100	-	505
	200	-	380
	500	-	290
	1000	-	230
	2000	-	140
	5000	-	200

TABLE I: Comparison of Algorithm 1 in PESSOA and SCOTS and Algorithm 2.

	N	t_{abs}	t_{syn}
Algorithm 4 (PESSOA)	-	12105	40
Algorithm 4 (SCOTS)	-	530	50
Algorithm 5	100	-	540
	200	-	400
	500	-	330
	1000	-	190
	2000	-	250
	5000	-	260

TABLE II: Comparison of Algorithm 4 in PESSOA and SCOTS and Algorithm 5.

forces. Note that a kneed biped is a hybrid system with two phases: 1) The unlocked knee phase starts at the beginning of a new step where the knee can bend and the system dynamics is modeled by (4) and lasts until the swing leg goes forward and straightens the knee; 2) The locked knee phase starts as soon as the knee straightens out and lasts until the swing leg hits the ground. The locked knee dynamics is modeled using different configurations of masses with the same dynamics of the unlocked knee phase. Switching between the two phases is governed by different guards based on the angles of the robot. Reset maps are applied to angles and angular velocities whenever one of the guards is reached. For further details on the model, we refer interested readers to [1].

We synthesized a controller that forces the robot to always move forward, which is captured by having $\dot{\theta}_1$ always greater than zero, and avoid obstacles on the ground. The state space and input space discretization parameters used were $\eta = 0.01$ and $\mu = 0.01$, respectively, whereas we used $\tau = 0.01$ for

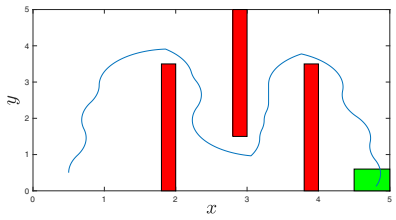

Fig. 1: Closed loop simulation result using the controller synthesized with lazy Algorithms 2 and 5.

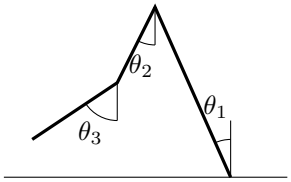

Fig. 2: A kneed biped over horizontal ground.

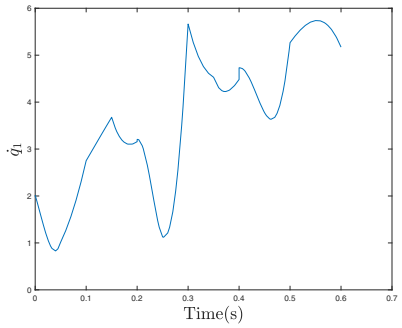

Fig. 3: Closed-loop simulation results, showing the evolution of $\dot{q}_1$ over time, using the controller synthesized with Algorithm 2.

the sampling time. Controller synthesis using Algorithm 2 took 10 hours, while PESSOA crashed after running for 5 consecutive days. It is worth mentioning that we were not able to try SCOTS on this example as it uses the notion of growth bound [25] to compute the reachable sets from each state x using input u, denoted by $\mathrm{Post}_u(x)$ in Refine_N, that does not handle the presence of guard and reset maps for hybrid systems as the kneed biped. Fig. 3 shows the closed-loop simulation results whereas Fig. 4 and 5 show tiles of two steps and one of the resulting walking gaits using the controller synthesized for the kneed biped model, respectively.

VI. Conclusion

In this paper we presented a lazy approach for controller synthesis. Instead of using a precomputed abstraction, we lazily compute the fragments of the abstraction that are needed to synthesize a controller. We presented two algorithms for safety and reachability specifications which are guaranteed to terminate in finite time and upon termination they return the same output as the classical algorithms. Using the unicycle example, we illustrated that we can achieve a speedup of up to 4 and 85 times for safety and 3 and 63 for reachability specifications compared to SCOTS and PESSOA, respectively. Moreover, we illustrated the novel lazy algorithm, for safety specifications, on a kneed biped robot example by synthesizing a controller that enforces the biped to move forward and avoid obstacles on the ground. As a future work, we intend to investigate the use of system-based heuristics to select the number of the states to be refined in the Refine_N function, as well as the input used for the refinement to achieve better performance. In addition, we will be investigating extensions of the proposed algorithms to recurrence and persistency specifications.

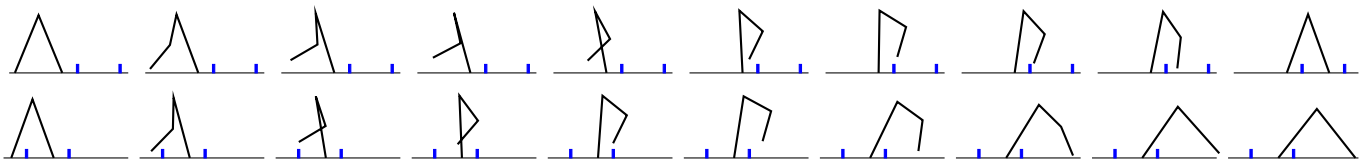

Fig. 4: Tiles of two steps by the robot generated using the synthesized controller. Each row represents the tiles of a single step.

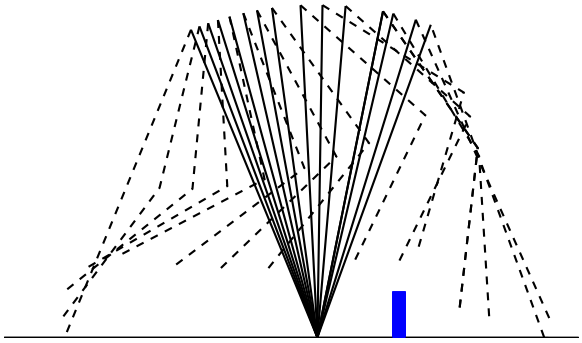

Fig. 5: A walking gait generated using the synthesized controller. The stance leg is shown as a solid line, whereas the swing leg is shown as a dashed line.

REFERENCES

[1] A. D. Ames. Characterizing knee-bounce in bipedal robotic walking: A zeno behavior approach. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 163–172. ACM, 2011.

[2] D. Angeli. A lyapunov approach to incremental stability properties. *IEEE Transactions on Automatic Control*, 47(3):410–421, 2002.

[3] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[4] L. De Alfaro and P. Roy. Solving games via three-valued abstraction refinement. In *CONCUR 2007–Concurrency Theory*, pages 74–89. Springer, 2007.

[5] A. Girard, G. Gössler, and S. Mouelhi. Safety controller synthesis for incrementally stable switched systems using multiscale symbolic models. *IEEE Transactions on Automatic Control*, 61(6):1537–1549, 2016.

[6] F. Gruber, E. S. Kim, and M. Arcak. Sparsity-sensitive finite abstraction. *arXiv preprint arXiv:1704.03951*, 2017.

[7] T. A. Henzinger, R. Jhala, and R. Majumdar. *Counterexample-guided control*. Springer, 2003.

[8] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck. Multi-layered abstraction-based controller synthesis for continuous-time systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*. ACM, 2018.

[9] O. Hussien, A. Ames, and P. Tabuada. Abstracting partially feedback linearizable systems compositionally. *IEEE Control Systems Letters*, 1(2):227–232, 2017.

[10] E. S. Kim, M. Arcak, and S. A. Seshia. Symbolic control design for monotone systems with directed specifications. *Automatica*, 83:10–19, 2017.

[11] E. S. Kim, M. Arcak, and M. Zamani. Constructing control system abstractions from modular components. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*. ACM, 2018.

[12] R. Majumdar, K. Mallik, and A.-K. Schmuck. Compositional synthesis of finite state abstractions. *arXiv preprint arXiv:1612.08515*, 2016.

[13] K. Mallik, S. E. Z. Soudjani, A.-K. Schmuck, and R. Majumdar. Compositional construction of finite state abstractions for stochastic control systems. *arXiv preprint arXiv:1709.09546*, 2017.

[14] M. Mazo Jr, A. Davitian, and P. Tabuada. Pessoa: A tool for embedded controller synthesis. In *Computer Aided Verification*, pages 566–569. Springer, 2010.

[15] P.-J. Meyer, A. Girard, and E. Witrant. Safety control with performance guarantees of cooperative systems using compositional abstractions. *IFAC-PapersOnLine*, 48(27):317–322, 2015.

[16] S. Mouelhi, A. Girard, and G. Gössler. Cosyma: a tool for controller synthesis using multi-scale abstractions. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 83–88. ACM, 2013.

[17] P. Nilsson and N. Ozay. Control synthesis for large collections of systems with mode-counting constraints. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 205–214. ACM, 2016.

[18] P. Nilsson and N. Ozay. Synthesis of separable controlled invariant sets for modular local control design. In *American Control Conference (ACC), 2016*, pages 5656–5663. IEEE, 2016.

[19] G. Pola, A. Borri, and M. D. Di Benedetto. Integrated design of symbolic controllers for nonlinear systems. *IEEE Transactions on Automatic Control*, 57(2):534–539, 2012.

[20] G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.

[21] G. Pola, P. Pepe, and M. D. Di Benedetto. Symbolic models for networks of control systems. *IEEE Transactions on Automatic Control*, 61(11):3663–3668, 2016.

[22] G. Pola, P. Pepe, M. D. Di Benedetto, and P. Tabuada. Symbolic models for nonlinear time-delay systems using approximate bisimulations. *Systems & Control Letters*, 59(6):365–373, 2010.

[23] G. Pola and P. Tabuada. Symbolic models for nonlinear control systems: Alternating approximate bisimulations. *SIAM Journal on Control and Optimization*, 48(2):719–733, 2009.

[24] G. Reißig. Abstraction based solution of complex attainability problems for decomposable continuous plants. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5911–5917. IEEE, 2010.

[25] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4):1781–1796, 2017.

[26] M. Rungger and O. Stursberg. On-the-fly model abstraction for controller synthesis. In *American Control Conference (ACC), 2012*, pages 2645–2650. IEEE, 2012.

[27] M. Rungger and M. Zamani. Scots: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM, 2016.

[28] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[29] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 313–314. ACM, 2011.

[30] M. Zamani, G. Pola, M. Mazo, and P. Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *Automatic Control, IEEE Transactions on*, 57(7):1804–1809, 2012.