UNIVERSITY OF CALIFORNIA

Los Angeles

Contributions toward Scalability of Correct-by-Construction Control Software Synthesis

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Omar Abdellatif E A Hussien

2018

ABSTRACT OF THE DISSERTATION

Contributions toward Scalability of Correct-by-Construction Control Software Synthesis

by

Omar Abdellatif E A Hussien
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2018
Professor Paulo Tabuada, Chair

As cyber-physical systems (CPS) become more complex, the verification of CPS control software becomes notoriously challenging. One way to alleviate the need for verification is to adopt a correct-by-construction approach. By synthesizing the control software along with a proof of correctness, the correct-by-construction approach eliminates, or greatly reduces, the need for verification. A common correct-by-construction approach is based on the computation of a finite-state abstraction of the control system. Given a specification expressed in a formal language such as temporal logic, 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. Despite the promise of correct-by-construction control software, this design methodology is not yet widely applicable as the computation of abstractions scales exponentially with the number of variables in the differential equation model of the system to be controlled. In this thesis, we discuss two approaches to mitigate this problem: 1) exploiting system structure and 2) lazy controller synthesis. In the first part of the thesis, we show how system structure can be exploited by discussing the class of partially feedback linearizable control systems. We show how the linearized part and the zero dynamics can be independently abstracted and subsequently composed to obtain an abstraction of the original continuous system. We also illustrate through examples how this compositional approach significantly reduces the time required for the construction of abstractions. Moreover, we discuss how this approach can be further generalized to a larger class of systems.

In the second part of the thesis, we present a lazy controller synthesis approach to tackle the lack of scalability of control software synthesis. Instead of synthesizing a controller using a precomputed abstraction of the full system, fragments of the abstraction are computed lazily, as needed, to synthesize a controller for various specifications in temporal logic. We illustrate, through different examples, how this lazy approach significantly reduces the total time required for the synthesis of correct-by-construction controllers.

In addition to exploiting structure and lazy synthesis, we also discuss possible future extensions to these two approaches.

The dissertation of Omar Abdellatif E A Hussien is approved.

University of California, Los Angeles

2018

To my parents, my wife, and my son ...

Table of Contents

List of Figures

List of Tables

Acknowledgments

All praise is due to Allah, Most Merciful, the Lord of the Worlds, Who taught man what he knew not. I would like to thank God Almighty for bestowing upon me the chance, strength and ability to complete this work. ”O my Lord! so order me that I may be grateful for Thy favours, which thou hast bestowed on me and on my parents, and that I may work the righteousness that will please Thee: And admit me, by Thy Grace, to the ranks of Thy righteous Servants.”, Quran[27:19]

This dissertation does not only hold the results of years of study and research, but also reflects the relationships with many generous and stimulating people. This is the time where I have the opportunity to present my appreciation to all of them.

I would like to express my gratitude to my advisor, Prof. Paulo Tabuada, for his guidance, advice, encouragement, invaluable support and immense help towards the completion of this work. His great personality, patience, and creativity provided me with an excellent atmosphere for work, while his technical insight and experience helped me a lot in my research. Thank you for everything, Paulo.

I want to thank my committee members, Todd Millstein, Christina Fragouli, and Panagiotis Christofides for their invaluable feedback. Also, I want to thank my labmates in CyPhy lab; Ayca Balkan, Joris Kenanian, Matthias Rungger, Yasser Shoukry, and Sina Caliskan for their unlimited support and hard work. I especially want to thank my coauthors Matthias Rungger and Ayca Balkan for all the fruitful discussions and the brilliant ideas we exchanged together.

I would like to thank all my friends here at UCLA for all the support they provided in dark and desperate times. I especially want to thank Ayca Balkan, Joris Kenanian, Mohammed Karmoose, Yahya Ezzeldin, Ahmed Alaa, Moustafa Alzantot, and Ahmed Hareedy.

I will never find words enough to express the gratitude and appreciation that I owe to my father, Prof. Abdellatif Elseddeek Hussien, my great mother, my brothers and sisters. Their tender love and support have always been the cementing force for building the blocks of my career. The all round support rendered by them provided the much needed stimulant to sail through the phases of stress and strain. I especially want to thank my brother, Faisal Hussien, for the invaluable advice and the unlimited support whenever needed. Thank you, Faisal!

Finally, I would like to thank my beloved wife, Alaa, for her great patience, unlimited support during the most critical stages in my PhD. Thank you for taking care of all the responsibilities of our life and our son, Youssef, to provide me with the right atmosphere to focus and concentrate at my work. I now have to repay you and our son the countless nights and weekends spent in the working done in this dissertation.

Vita

2009	B.Sc., Computer Engineering, Cairo University, Cairo, Egypt.
2009–2013	Research and Teaching Assistant, Computer Engineering Department, Cairo University.
2011–2013	Part-time Digital Design Engineer, Si-Ware Systems, Cairo, Egypt.
2013	M.Sc., Computer Engineering, Cairo University, Cairo, Egypt.
Spring 2017	Research Intern, Toyota ITC, Mountain View, CA

CHAPTER 1

Introduction

In the recent years, the verification of control software for cyber-physical systems became more challenging given the increased complexity of these systems. A recent approach to handle the verification problem is to synthesize a control software using correct-by-construction methods. These are techniques that synthesize both the control software as well as a proof of its correctness so that a-posteriori verification is greatly reduced or not even required. One of the widely used correct-by-construction techniques is based on the construction of a finite-state abstraction for the given control system. A controller enforcing the specification can then be synthesized for the abstraction and subsequently lifted to a controller acting on the control system. Control software synthesis based on abstractions has two advantages over more traditional control design techniques: 1) it allows the use of more complex specifications such as those expressed in temporal logic; 2) controller synthesis is completely automated and consists of computing a fixed-point over the finite-state abstraction, which can always be done in finite time for finite-state abstractions [Tab09]. Construction of abstractions for control systems that are incrementally input-to-state stable was presented in [PGT08], [PT09] and [PPD10]. In [ZPM12] the authors showed that finite-state abstractions can still be computed even if incremental input-to-state stability fails to hold. Furthermore, various software tools for correct-by-construction controller synthesis, using abstractions, have been developed and include PESSOA [MDT10], CoSyMa [MGG13] and SCOTS [RZ16].

The main drawback of the abstraction based control software synthesis is that the computation of abstractions does not scale well with the number of states of the control system. Hence, it becomes infeasible to compute abstractions for large systems.

The main contribution of this thesis is to alleviate this problem using two different approaches: 1) exploiting system structure and 2) lazy controller synthesis.

In Chapter 2, we discuss a way to alleviate this problem that avoids computing the abstraction monolithically and, instead, computes it compositionally. Recent results for the compositional construction of abstractions were presented for discrete control systems in [PPD14]. The authors in [Rei10] computed abstractions compositionally for nonlinear control systems that can be decoupled into smaller subsystems using the same inputs. Similar results were introduced in [NO16] for a collection of identical decoupled switched systems subject to counting constraints and in [MGW15] for nonlinear cooperative control systems in which the interaction between subsystems is modeled as a disturbance. However, these assumptions address a limited class of systems. To the best of our knowledge, the current literature, except [MGW15], does not address nonlinear continuous control systems with coupling dynamics which is common in many concrete applications. Nevertheless, the approach in [MGW15] may be excessively specific as it treats the interactions between systems as a cooperative disturbance based on small gain arguments.

In this thesis, we focus on control systems that are partially feedback linearizable. We exploit this assumption to decompose such systems into its feedback linearizable part and its zero dynamics. Such cascade decomposition is then exploited to compute abstractions compositionally: abstractions of the feedback linearizable part and of the zero dynamics are independently computed and then composed to obtain an abstraction of the original system.

By focusing on a different class of systems, the proposed compositional approach complements the compositional results reported in the literature. Moreover, since the class of partially feedback linearizable systems is reasonably large, the proposed results are quite useful in practice. To further substantiate this claim we show in Section 6 how the proposed results enable us to compute abstractions faster than the monolithic approach. It is worth mentioning that our results motivated work by other researchers. In [SJZ18] our results were extended to cascade interconnections where different types of abstractions were used for the different subsystems. Moreover, they also presented results on compositional controller synthesis.

In Chapter 3, we discuss another way to tackle the lack of scalability of abstraction based control software synthesis that adopts 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 various specifications. Results that are based on computing an initial coarse abstraction which is gradually refined on-demand can be found in [HJM03] and [DR07]. The authors in [HJM03] extended the method of counterexample-guided refinement (CEGAR) [CGJ00] from verification to controller synthesis. In [DR07] 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. Similarily, in [HMM18] the authors introduced a multi-layered abstraction approach where they 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.

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 Chapter 3, we discuss how to synthesize a controller by lazily computing the abstraction as needed for various specifications, namely, safety, reachability, persistence, and recurrence. Similarly to the approach in [DR07], 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 [DR07] and [HMM18] refines states.

We present lazy controller synthesis algorithms for the aforementioned specifications and we illustrate through different examples how the lazy approach is significantly faster as compared to controller synthesis using a precomputed abstraction.

Finally, we conclude and discuss in Chapter 4 possible future extensions to the aforementioned approaches.

CHAPTER 2

Abstracting partially feedback linearizable systems compositionally

In this chapter, we discuss how to compute abstractions for control systems that are partially feedback linearizable compositionally. We exploit this assumption to decompose such systems into its feedback linearizable part and its zero dynamics. Such cascade decomposition is then exploited to compute abstractions compositionally: abstractions of the feedback linearizable part and of the zero dynamics are independently computed and then composed to obtain an abstraction of the original system.

This chapter is organized as follows. Section 1 introduces the class of control systems we consider in this chapter. In Section 2 we review the definition of different types of approximate simulation relations. Our main contribution appears in Section 3. In Section 4 we show that our approach is a generalization for the results introduced earlier in [ZPM12]. We illustrate the benefits of our approach through examples in Section 6. The chapter ends with several concluding remarks in Section 7

1 Control systems and cascade decompositions

1.1 Notation

We use $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}_0^+$ to denote the set of integer, real, positive and nonnegative real numbers. Given a vector $x \in \mathbb{R}^n$ and a matrix $A \in \mathbb{R}^{m\times n}$, we denote by $\|x\|$ and $\|A\|$ the infinity norm of x and A, respectively. We define the discretization of $S \subset \mathbb{R}^n$ 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 $\|f\|_\infty$. A continuous function $\gamma : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ belongs to class $\mathcal{K}$ if it is strictly increasing and $\gamma(0) = 0$; furthermore γ belongs to class $\mathcal{K}_\infty$ if $\gamma \in \mathcal{K}$ and $\gamma(r) \to \infty$ as $r \to \infty$. A continuous function $\beta : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$ belongs to class $\mathcal{KL}$ if for every fixed s, the function $\beta(\cdot, s)$ belongs to class $\mathcal{K}_\infty$ and for every fixed r, the function $\beta(r, \cdot)$ is decreasing and $\beta(r, s) \to 0$ as $s \to \infty$.

1.2 Control Systems and Cascade Decompositions

We work with continuous time control systems defined as follows.

Definition 1.1 (Control System). 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 Lipschitz continuous map $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ defining the dynamics of the system.

We say Σ is a single-input control system when $m = 1$. We denote the trajectory of a control system Σ by $\xi_{xv} :]a, b[\to \mathbb{R}$ if there exists $v \in \mathcal{U}$ such that $\dot{\xi}_{xv} = f(\xi_{xv}, v)$. We also use the notation $\xi_{xv}(\tau)$ to denote the point reached by system Σ, at time τ, from the initial state x while applying the input v. Note that this point is uniquely determined due to the Lipschitz continuity assumption on f [Kha96]. A control system Σ is said to be forward complete if every trajectory is defined on an interval of the form $]a, \infty[$, where $a \in \mathbb{R}$. Necessary and sufficient conditions for forward completeness can be found in [AS99]

The results presented in this work are proved for the class of control systems that can be defined as a cascade composition of smaller subsystems. We now present a definition of cascade decomposition tailored to the decompositions that arise from single-input partially feedback linearizable systems.

Definition 1.2. Let $\Sigma = (\mathbb{R}^n, U, \mathcal{U}, f)$ be a control system, let $x = (z, w) \in \mathbb{R}^n$, where $z = (x_1, \ldots, x_p)$ and $w = (x_{p+1}, \ldots, x_n)$ for some $p \leq n$, and let $v = (z, u)$ for $u \in U \subseteq \mathbb{R}$. System Σ admits a cascade decomposition into $\Sigma_1 = (\mathbb{R}^p, U, \mathcal{U}_1, f_1)$ and $\Sigma_2 = (\mathbb{R}^{n-p}, \mathbb{R}^{p+1}, \mathcal{U}_2, g)$ if:

$$f(x,u) = (f_1(z,u), f_2(w,v))$$
$$f_1(z,u) = (x_2, x_3, ..., x_p, u)$$
$$f_2(w,v) = g(w,v).$$

Accordingly, system Σ can be seen as a cascade composition of Σ_1, of the form $\dot{z} = f_1(z,u)$, and Σ_2, of the form $\dot{w} = g(w,v)$, where the input of Σ_2 is connected to the output of Σ_1 defined by $v = (z,u)$. Given a single input partially feedback linearizable system Σ of relative degree p, we can always decompose it into its feedback linearized part and the residual dynamics. The feedback linearized part corresponds to subsystem Σ_1 while the residual dynamics corresponds to Σ_2. Note that although Definition 2.2 describes a decomposition of Σ into two subsystems, Σ_1 and Σ_2, all the following results are still valid when we have N subsystems, e.g., when we have more than one input in a partially feedback linearizable system. We use $\xi_{x\upsilon}$, $\xi_{z\upsilon}$, and $\xi_{w\nu}$ to denote the trajectories Σ, Σ_1 and Σ_2, respectively. Let $\pi_1 : \mathbb{R}^n \to \mathbb{R}^p$ and $\pi_2 : \mathbb{R}^n \to \mathbb{R}^{n-p}$ be the natural projections on the first p and last $n-p$ entries, respectively. Rather than writing $\pi_1 \circ \xi_{x\upsilon}$ and $\pi_2 \circ \xi_{x\upsilon}$ we use the simpler notation $\xi^1_{x\upsilon}$ and $\xi^2_{x\upsilon}$, respectively. Note that whenever the input curves are assumed to be constant, we will use u and v instead of the Greek letters υ and ν, respectively.

1.3 Divergence of trajectories

To prove the existence of different types of simulation relations between abstractions and control systems, we need to define a bound on the divergence of trajectories. We introduce the notion of incremental forward completeness [ZPM12].

Definition 1.3 (Incremental forward completeness). A control system Σ is incrementally forward complete (δ-FC) if it is forward complete and there exist continuous functions $\beta : \mathbb{R}^+_0 \times \mathbb{R}^+_0 \to \mathbb{R}^+_0$ and $\gamma : \mathbb{R}^+_0 \times \mathbb{R}^+_0 \to \mathbb{R}^+_0$ such that for every $s \in \mathbb{R}^+$, the functions $\beta(\cdot, s)$

and $\gamma(\cdot, s)$ belong to class $\mathcal{K}_\infty$ and for any $x, x' \in \mathbb{R}^n$, any $\tau \in \mathbb{R}^+$ and any $\upsilon, \upsilon' : [0, \tau[\rightarrow \mathbb{R}$, the following condition is satisfied for all $t \in [0, \tau]$:

$$\|\xi_{x\upsilon}(t) - \xi_{x'\upsilon'}(t)\| \leq \beta(\|x - x'\|, t) + \gamma(\|\upsilon - \upsilon'\|_\infty, t). \tag{2.1}$$

In other words, a control system is incrementally forward complete if the distance between any two trajectories starting from different initial states while applying different inputs for the same duration of time can be bounded by the functions β and γ, appearing in (2.1), that depend on the difference between the initial states and the difference between the inputs, respectively.

Note that for a linear control system defined as

$$\dot{x} = Ax + Bu, \tag{2.2}$$

the functions β and γ will be

$$\begin{aligned} \beta(r, t) &= \left\|e^{At}\right\| r \\ \gamma(r, t) &= \left(\int_0^t \left\|e^{As}B\right\| ds\right) r, \end{aligned} \tag{2.3}$$

where $\left\|e^{At}\right\|$ denotes the infinity norm of e^{At}.

2 Symbolic models and approximate simulation relations

2.1 Systems

We briefly introduce the notion of system which will be used later to model all the systems of interest. Further details on the notion of system are provided in [Tab09].

Definition 2.1 (System). A system S is a quintuple $S = (X, U, \longrightarrow, Y, H)$ consisting of:

- A set of states X;
- A set of inputs U;

- A transition relation $\longrightarrow \subseteq X \times U \times X$;
- An output set Y;
- An output map $H : X \longrightarrow Y$.

A system S is metric, if there exist a metric $d : Y \times Y \rightarrow \mathbb{R}_0^+$. We call state x' a u-successor for state x if the transition $x \xrightarrow{u} x'$ exists in the system. We also introduce the set of u-successors of a state x, denoted by $\mathbf{Post}_u(x)$, as well as the set of inputs $u \in U$, denoted by $U(x)$, such that $\mathbf{Post}_u(x)$ is nonempty.

2.2 Simulation Relations

We now introduce different types of simulation relations which we use to relate the computed abstractions to the control system of interest. We start with the notion of approximate simulation relation [GP07].

Definition 2.2. Let $S_a = \left(X_a, U_a, \underset{a}{\longrightarrow}, Y_a, H_a\right)$ and $S_b = \left(X_b, U_b, \underset{b}{\longrightarrow}, Y_b, H_b\right)$ be metric systems with the same output sets $Y_a = Y_b$ and metric d, and consider a precision $\varepsilon \in \mathbb{R}^+$. A relation $R \subseteq X_a \times X_b$ is said to be an ε-approximate simulation relation from S_a to S_b if the following three conditions are satisfied:

1. for every $x_a \in X_a$, there exists $x_b \in X_b$ with $(x_a, x_b) \in R$;
2. for every $(x_a, x_b) \in R$ we have $d(H_a(x_a), H_b(x_b)) \leq \varepsilon$;
3. for every $(x_a, x_b) \in R$ we have that $x_a \xrightarrow{u_a} x'_a$ in S_a implies the existence of $x_b \xrightarrow{u_b} x'_b$ in S_b satisfying $(x'_a, x'_b) \in R$.

We denote the existence of an ε-approximate simulation relation from S_a to S_b by $S_a \preceq_{\mathcal{S}}^{\varepsilon} S_b$.

While simulation relations are useful for verification purposes, when the objective is the synthesis of controllers, the relevant notion is alternating simulation. See [Tab09] for a comparison between these two different, but related, notions.

Definition 2.3. Let $S_a = \left(X_a, U_a, \underset{a}{\longrightarrow}, Y_a, H_a\right)$ and $S_b = \left(X_b, U_b, \underset{b}{\longrightarrow}, Y_b, H_b\right)$ be metric systems with the same output sets $Y_a = Y_b$ and metric d, and consider a precision $\varepsilon \in \mathbb{R}^+$. A relation $R \subseteq X_a \times X_b$ is said to be an ε-approximate alternating simulation relation from S_a to S_b if the following three conditions are satisfied:

1. for every $x_a \in X_a$, there exists $x_b \in X_b$ with $(x_a, x_b) \in R$;
2. for every $(x_a, x_b) \in R$ we have $d\left(H_a\left(x_a\right), H_b\left(x_b\right)\right) \leq \varepsilon$;
3. for every $(x_a, x_b) \in R$ and for every $u_a \in U_a\left(x_a\right)$ there exists $u_b \in U_b\left(x_b\right)$ such that for every $x'_b \in \mathbf{Post}_{u_b}\left(x_b\right)$ there exists $x'_a \in \mathbf{Post}_{u_a}\left(x_a\right)$ satisfying $(x'_a, x'_b) \in R$.

We denote the existence of an ε-approximate alternating simulation relation from S_a to S_b by $S_a \preceq^{\varepsilon}_{\mathcal{AS}} S_b$.

Note that the existence of these simulation relations enables the refinement of controllers synthesized for the abstractions to controllers that act on the control system.

2.3 Symbolic Models

Since the introduced simulation relations are defined for discrete systems, we define the discretization of a control system Σ denoted by $S_\tau(\Sigma)$, where $\tau \in \mathbb{R}^+$ is the sampling time, as follows

$$S_\tau\left(\Sigma\right) = \left(\mathbb{R}^n, U_\tau, \underset{\tau}{\longrightarrow}, \mathbb{R}^n, 1_{\mathbb{R}^n}\right), \tag{2.4}$$

where

- $U_\tau = \{u : [0, \tau[\rightarrow \mathbb{R} | u(t) = u(0), t \in [0, \tau[\}$;
- $x_\tau \xrightarrow{u_\tau} x'_\tau$ if $\xi_{x_\tau u_\tau}\left(\tau\right) = x'_\tau$,

for $x_\tau, x'_\tau \in \mathbb{R}^n$ and $u_\tau \in U_\tau$.

We compute an abstraction of a control system by discretization of state space, input space and time.

Definition 2.4 (Abstraction). Given the control system $\Sigma = (\mathbb{R}^n, U, \mathcal{U}, f)$, a map $\delta : \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$, and the triple $q = (\tau, \eta, \mu)$ of quantization parameters, where $\tau \in \mathbb{R}^+$ is the sampling time, $\eta \in \mathbb{R}^+$ is the state space quantization, and $\mu \in \mathbb{R}^+$ is the input quantization, the abstraction of Σ associated with q and δ is the system $S_{q\delta}(\Sigma) = (X, U, \longrightarrow, Y, H)$ defined by:

- $X = [\mathbb{R}^n]_\eta$;
- $U = [\mathbb{R}^m]_\mu$;
- $x \xrightarrow{u} x'$ if $\|\xi_{x,u}(\tau) - x'\| \leq \delta(\varepsilon, \tau, \eta, \mu)$;
- $Y = \mathbb{R}^n$;
- $H = \imath : X \hookrightarrow Y$,

where $\varepsilon \in \mathbb{R}^+$ and $\imath$ is the natural embedding of X into Y.

Note that, an abstraction is finite if it has a finite set of states and a finite set of inputs, which can be achieved if the state space and the input space are restricted to bounded sets.

Given a system Σ that admits a cascade decomposition into Σ_1 and Σ_2, as in Definition 1.2, instead of Σ_2 we work with

$$\tilde{\Sigma}_2 = \left(\mathbb{R}^{n-p}, \mathbb{R}^{p+1}, \mathcal{U}_2, g\left(w, \xi_{zu}, u\right)\right) \tag{2.5}$$

where $\mathcal{U}_2$ is the set of constant curves, and we use trajectories of Σ_1 as an input to $\tilde{\Sigma}_2$.

Definition 2.5 (Composed Abstraction). Let the abstractions of Σ_1 and $\tilde{\Sigma}_2$ be $S_{q_1\delta_1}(\Sigma_1) = \left(X_1, U_1, \underset{1}{\longrightarrow}, Y_1, H_1\right)$ and $S_{q_2\delta_2}(\tilde{\Sigma}_2) = \left(X_2, U_2, \underset{2}{\longrightarrow}, Y_2, H_2\right)$, respectively where:

$$q_1 = (\tau, \eta_1, \eta_1), \quad q_2 = (\tau, \eta_2, \eta_1), \quad \tau, \eta_1, \eta_2 \in \mathbb{R}^+,$$

The composed abstraction of Σ denoted by $S_{q_1\delta_1}(\Sigma_1) \circledcirc S_{q_2\delta_2}(\tilde{\Sigma}_2) = (X, U, \longrightarrow, Y, H_{q_1\delta_1 q_2\delta_2})$ is defined by:

- $X = X_1 \times X_2$;

- $U = U_1$;

- $(x_1, x_2) \xrightarrow{u} (x_1', x_2')$ if $x_1 \xrightarrow{u} x_1'$ in $S_{q_1\delta_1}(\Sigma_1)$, $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$ and $u_2 = (x_1, u_1)$;

- $Y = \mathbb{R}^n = Y_1 \times Y_2$;

- $H_{q_1\delta_1 q_2\delta_2} = (\imath_1, \imath_2) : X_1 \times X_2 \hookrightarrow Y_1 \times Y_2$,

3 Symbolic models for δ-FC cascade control systems

In this section we present our main result. Given a control system Σ that can be decomposed into smaller subsystems as in Definition 1.2, we prove the existence of different types of simulation relations between the control system and the computed abstraction by composing abstractions of smaller subsystems as in Definition 2.5.

Theorem 3.1. Let Σ be a control system that can be decomposed into Σ_1 and Σ_2, as in Definition 1.2, and let $\tilde{\Sigma}_2$ be the system defined in (2.5). Let $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2)$ be the composed abstraction given in Definition 2.5 and consider any precision $\varepsilon \in \mathbb{R}^+$. Under the following assumptions:

- $\max\{\eta_1, \eta_2\} \leq \varepsilon$;

- Σ_1 and $\tilde{\Sigma}_2$ are δ-FC control systems;

- $\delta_1(q_1) = \beta_1(\varepsilon, \tau) + \eta_1$;

- $\delta_2(q_2) = \beta_2(\varepsilon, \tau) + \gamma_2(\varepsilon, \tau) + \eta_2$,

where β_i and γ_i for $i = 1, 2$ are the functions in Definition 1.3, we have:

$$S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2) \preceq^{\varepsilon}_{\mathcal{AS}} S_{\tau}(\Sigma)\,. \tag{2.6}$$

Moreover, if δ_1 is defined by

$$\delta_1(q_1) = \beta_1(\varepsilon, \tau) + \gamma_1(\eta_1, \tau) + \eta_1,$$

we also have:

$$S_\tau(\Sigma) \preceq^{\varepsilon}_{\mathcal{S}} S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2). \tag{2.7}$$

Note that if (2.6) holds, a controller synthesized for the abstraction can be refined to a controller for the original control system. However, non existence of a controller for the abstraction does not imply non existence of a controller for the original control system. If (2.7) holds, the abstraction is rich (e.g., every trajectory of $S_\tau(\Sigma)$ is also a trajectory of $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2)$) and finding a controller is relatively easier. Also, it is worth mentioning that although (2.6) and (2.7) are the same inequalities that appear in [ZPM12], our approach results in a more conservative abstraction. Accordingly, there might be controllers for a monolithic abstraction that cannot be found when working with $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2)$. We return to this point in Section 6 in the context of a specific example.

Proof. **First** we prove (2.6). Consider the relation $R \subseteq X \times \mathbb{R}^n$ defined by $((x_1, x_2), (z, w)) \in R$ iff

$$\begin{aligned} d(H_{q_1\delta_1q_2\delta_2}(x_1, x_2), H(z, w)) &= \|(x_1, x_2) - (z, w)\| \\ &\le \varepsilon. \end{aligned}$$

By choosing $z = x_1$ and $w = x_2$, $((x_1, x_2), (z, w)) \in R$ and conditions (1-2) in Definition 2.3 are satisfied. Now we show that condition (3) in Definition 2.3 is satisfied for every $((x_1, x_2), (z, w)) \in R$. Consider any $u_1 \in U_1$ and let $u \in U_\tau$ be equal to u_1. Consider the unique transition $(z, w) \xrightarrow{u} (z', w') = \xi_{xu}(\tau) \in \mathbf{Post}_u(z, w)$ in $S_\tau(\Sigma)$. To prove the existence of a transition in $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2)$ we need to show that: (i) $x_1 \xrightarrow{u_1} x_1'$ in $S_{q_1\delta_1}(\Sigma_1)$, (ii) $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$, and (iii) $u_2 = (x_1, u_1)$ hold.

Since for all $((x_1, x_2), (z, w)) \in R$, $\|(x_1, x_2) - (z, w)\| \le \varepsilon$ and as we are using the infinity norm, we obtain

$$\max\{\|x_1 - z\|, \|x_2 - w\|\} = \|(x_1, x_2) - (z, w)\| \leq \varepsilon. \tag{2.8}$$

We start by proving (i) as follows. Consider $x_1' = [\xi_{zu}]_{\eta_1}$, we have

$$\left\|\xi^1_{xu}(\tau) - x_1'\right\| = \left\|\xi^1_{xu}(\tau) - [\xi_{zu}]_{\eta_1}\right\| \leq \eta_1. \tag{2.9}$$

Given that Σ_1 is δ-FC, $u = u_1$ and using (2.8) and (2.9), we have

$$\begin{aligned}
\|\xi_{x_1u_1}(\tau) - x_1'\| &\leq \|\xi_{x_1u_1}(\tau) - \xi_{xu}(\tau)\| + \|\xi_{xu}(\tau) - x_1'\| \\
&\leq \beta_1(\|x_1 - z\|, \tau) + \left\|\xi^1_{xu}(\tau) - x_1'\right\| \\
&\leq \beta_1(\varepsilon, \tau) + \left\|\xi^1_{xu}(\tau) - x_1'\right\| \\
&\leq \beta_1(\varepsilon, \tau) + \eta_1,
\end{aligned} \tag{2.10}$$

which implies the existence of $x_1 \xrightarrow{u_1} x_1'$ in $S_{q_1\delta_1}(\Sigma_1)$.

Now we show that (ii) and (iii) hold. Consider $x_2' = [\xi_{wv}(\tau)]_{\eta_2}$, we obtain

$$\left\|\xi^2_{xu}(\tau) - x_2'\right\| = \left\|\xi^2_{xu}(\tau) - [\xi_{wv}(\tau)]_{\eta_2}\right\| \leq \eta_2. \tag{2.11}$$

Given that $\tilde{\Sigma}_2$ is δ-FC, $u_2 = (x_1, u_1) = (x_1, u)$ and using (2.8) and (2.11), we have

$$\begin{aligned}
\|\xi_{x_2u_2}(\tau) - x_2'\| \leq& \left\|\xi_{x_2u_2}(\tau) - \xi^2_{xu}(\tau)\right\| + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\
\leq& \beta_2(\|w - x_2\|, \tau) + \gamma_2(\|u_2 - v\|, \tau) + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\
\leq& \beta_2(\varepsilon, \tau) + \gamma_2(\|u_2 - v\|, \tau) + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\
\leq& \beta_2(\varepsilon, \tau) + \gamma_2(\|x_1 - z\|, \tau) + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\
\leq& \beta_2(\varepsilon, \tau) + \gamma_2(\varepsilon, \tau) + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\
\leq& \beta_2(\varepsilon, \tau) + \gamma_2(\varepsilon, \tau) + \eta_2,
\end{aligned} \tag{2.12}$$

which implies the existence of $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$.

From (2.10), (2.12) and $u_2 = (x_1, u_1)$, we conclude the existence of $(x_1, x_2) \xrightarrow{u} (x_1', x_2')$ in the composed system $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2)$. Using (2.9) and (2.11), we obtain

$$\begin{aligned} \|(x_1', x_2') - (z', w')\| &= \max\{\|x_1' - z'\|, \|x_2' - w'\|\} \\ &= \max\{\eta_1, \eta_2\} \leq \varepsilon, \end{aligned} \tag{2.13}$$

which implies that $((x_1', x_2'), (z', w'))$ belongs to R, hence $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2) \preceq^{\varepsilon}_{\mathcal{AS}} S_\tau(\Sigma)$.

Second we prove (2.7). Consider the relation $R \subseteq \mathbb{R}^n \times X$ defined by $((z, w), (x_1, x_2)) \in R$ iff

$$\begin{aligned} d(H(z, w), H_{q_1\delta_1q_2\delta_2}(x_1, x_2)) &= \|(z, w) - (x_1, x_2)\| \\ &\leq \varepsilon. \end{aligned}$$

Since for all $(z, w) \in \mathbb{R}^n$, there exist a $(x_1, x_2) \in X$ satisfying

$$\begin{aligned} \|(z, w) - (x_1, x_2)\| &= \max\{\|z - x_1\|, \|w - x_2\|\} \\ &= \max\{\eta_1, \eta_2\} \leq \varepsilon, \end{aligned} \tag{2.14}$$

as we are using the infinity norm, hence $((z, w), (x_1, x_2)) \in R$ and conditions (1-2) in Definition 2.2 are satisfied. Now we show that condition (3) in Definition 2.2 is satisfied for every $((z, w), (x_1, x_2)) \in R$. Consider any $u \in U_\tau$ we pick u_1 such that

$$\|u - u_1\| \leq \eta_1. \tag{2.15}$$

Consider the transition $(z, w) \xrightarrow{u} (z', w') = \xi_{xu}(\tau)$ in $S_\tau(\Sigma)$. To prove the existence of a transition in $S_{q_1\delta_1}(\Sigma_1) \circleddash S_{q_2\delta_2}(\tilde{\Sigma}_2)$ we need to show that: (i) $x_1 \xrightarrow{u_1} x_1'$ in $S_{q_1\delta_1}(\Sigma_1)$, (ii) $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$, and (iii) $u_2 = (x_1, u_1)$ hold.
We start by proving (i) as follows. Consider $x_1' = [\xi_{zu}]_{\eta_1}$, we obtain

$$\left\|\xi^1_{xu}(\tau) - x_1'\right\| = \left\|\xi^1_{xu}(\tau) - [\xi_{zu}]_{\eta_1}\right\| \leq \eta_1. \tag{2.16}$$

Given that Σ_1 is δ-FC, and using (2.14), (2.15), and (2.16), we have

$$\begin{aligned}\left\|\xi_{x_1u_1}(\tau)-x_1'\right\| \leq& \left\|\xi_{x_1u_1}(\tau)-\xi_{xu}^1(\tau)\right\|+\left\|\xi_{xu}^1(\tau)-x_1'\right\| \\ \leq&\beta_1\left(\left\|x_1-z\right\|,\tau\right)+\gamma_1\left(\left\|u_1-u\right\|,\tau\right)+\left\|\xi_{xu}^1(\tau)-x_1'\right\| \\ \leq&\beta_1(\varepsilon,\tau)+\gamma_1\left(\left\|u_1-u\right\|,\tau\right)+\left\|\xi_{xu}^1(\tau)-x_1'\right\| \\ \leq&\beta_1(\varepsilon,\tau)+\gamma_1(\eta_1,\tau)+\left\|\xi_{xu}^1(\tau)-x_1'\right\| \\ \leq&\beta_1(\varepsilon,\tau)+\gamma_1(\eta_1,\tau)+\eta_1,\end{aligned} \tag{2.17}$$

which implies the existence of $x_1 \xrightarrow{u_1} x_1'$ in $S_{q_1\delta_1}(\Sigma_1)$.

Now we show that (ii) and (iii) hold. Consider $x_2'=\left[\xi_{x,u}^2(\tau)\right]_{\eta_2}$, we obtain

$$\left\|\xi_{xu}^2(\tau)-x_2'\right\|=\left\|\xi_{xu}^2(\tau)-\left[\xi_{wv}(\tau)\right]_{\eta_2}\right\|\leq\eta_2. \tag{2.18}$$

Given that $\tilde{\Sigma}_2$ is δ-FC, $u_2=(x_1,u_1)$ and using (2.14), (2.15) and (2.18), we have

$$\begin{aligned}\left\|\xi_{x_2u_2}(\tau)-x_2'\right\| \leq& \left\|\xi_{x_2u_2}(\tau)-\xi_{xu}^2(\tau)\right\|+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq&\beta_2\left(\left\|w-x_2\right\|,\tau\right)+\gamma_2\left(\left\|u_2-v\right\|,\tau\right)+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq&\beta_2(\varepsilon,\tau)+\gamma_2\left(\left\|u_2-v\right\|,\tau\right)+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq&\beta_2(\varepsilon,\tau)+\gamma_2\left(\left\|(x_1,u_1)-v\right\|,\tau\right)+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq&\beta_2(\varepsilon,\tau)+\gamma_2(\max\{\varepsilon,\eta_1\},\tau)+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq&\beta_2(\varepsilon,\tau)+\gamma_2(\varepsilon,\tau)+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq&\beta_2(\varepsilon,\tau)+\gamma_2(\varepsilon,\tau)+\eta_2,\end{aligned} \tag{2.19}$$

which implies the existence of $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$.

From (2.17), (2.28) and $u_2=(x_1,u_1)$, we conclude the existence of $(x_1,x_2) \xrightarrow{u} (x_1',x_2')$

in the composed system $S_{q_1\delta_1}(\Sigma_1) \circledcirc S_{q_2\delta_2}(\tilde{\Sigma}_2)$. Using (2.16) and (2.18) we obtain

$$\begin{aligned} \|(z', w') - (x_1', x_2')\| &= \max\{\|z' - x_1'\|, \|w' - x_2'\|\} \\ &= \max\{\eta_1, \eta_2\} \le \varepsilon, \end{aligned} \tag{2.20}$$

which implies that $((z', w'), (x_1', x_2')) \in R$, hence $S_\tau(\Sigma) \preceq_{\mathcal{S}}^{\varepsilon} S_{q_1\delta_1}(\Sigma_1) \circledcirc S_{q_2\delta_2}(\tilde{\Sigma}_2)$. $\square$

4 Symbolic models for δ-FC control systems

In this section, we show that results presented in Section 3 are generalization for the results introduced earlier in [ZPM12]. This can be shown by comparing results when: 1) cascade control system is reduced to subsystem Σ_1 and 2) cascade control system is reduced to subsystem $\tilde{\Sigma}_2$ defined in (2.5).

When the cascade control system is reduced to subsystem Σ_1, our results exactly match the results from the literature. This conclusion follows from the straightforward observation that Σ_1 is not affected by $\tilde{\Sigma}_2$. However, this is not the case when the cascade control system is reduced to subsystem $\tilde{\Sigma}_2$. It suffices to show that using an abstraction similar to the abstraction used in [ZPM12] we have $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right) \preceq_{\mathcal{AS}}^{\varepsilon} S_\tau\left(\tilde{\Sigma}_2\right) \preceq_{\mathcal{S}}^{\varepsilon} S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$.

Corollary 4.1. Let Σ, Σ_1 and $\tilde{\Sigma}_2$ be the systems defined in Theorem 3.1. Consider that Σ can be reduced to $\tilde{\Sigma}_2$. Let $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$ be an abstraction of $\tilde{\Sigma}_2$ where:

$$q_2 = (\tau, \eta_2, \eta_1), \quad \tau, \eta_1, \eta_2 \in \mathbb{R}^+,$$

and consider any precision $\varepsilon \in \mathbb{R}^+$. Under the following assumptions:

- $\eta_2 \le \varepsilon$;
- $\tilde{\Sigma}_2$ is δ-FC control systems;
- $\delta_2(q_2) = \beta_2(\varepsilon, \tau) + \eta_2$,

where β_2 and γ_2 are the functions in Definition 1.3, we have:

$$S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right) \preceq^{\varepsilon}_{\mathcal{AS}} S_{\tau}\left(\tilde{\Sigma}_2\right). \tag{2.21}$$

Furthermore, consider $\delta_2(q_2) = \beta_2(\varepsilon, \tau) + \gamma_2(\eta_1, \tau) + \eta_1$, we have:

$$S_{\tau}\left(\tilde{\Sigma}_2\right) \preceq^{\varepsilon}_{\mathcal{S}} S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right). \tag{2.22}$$

Proof. **First** we prove that $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right) \preceq^{\varepsilon}_{\mathcal{AS}} S_{\tau}\left(\tilde{\Sigma}_2\right)$. Consider the relation $R \subseteq X_2 \times \mathbb{R}^{n-p}$ defined by $(x_2, w) \in R$ iff

$$\begin{aligned} d(H_{q_2\delta_2}(x_2), H(w)) &= \|x_2 - w\| \\ &\leq \varepsilon. \end{aligned}$$

By choosing $w = x_2$, $(x_2, w) \in R$ and conditions (1-2) in Definition 2.3 are satisfied. Now we show that condition (3) in Definition 2.3 is satisfied for every $(x_2, w) \in R$. Consider any $u_3 \in U_2$ and let $v \in U_\tau$ be equal to u_2. Note that, since Σ is reduced to $\tilde{\Sigma}_2$, states of Σ_1 are treated as external input to $\tilde{\Sigma}_2$.

Consider the unique transition $w \xrightarrow{v} w' = \xi^2_{xu}(\tau) \in \mathbf{Post}_v(w)$ in $S_\tau\left(\tilde{\Sigma}_2\right)$. To prove the existence of a transition in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$ we need to show that $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$. Consider $x_2' = [\xi_{wv}(\tau)]_{\eta_2}$, we obtain

$$\left\|\xi^2_{xu}(\tau) - x_2'\right\| = \left\|\xi^2_{xu}(\tau) - [\xi_{wv}(\tau)]_{\eta_2}\right\| \leq \eta_2. \tag{2.23}$$

Given that $\tilde{\Sigma}_2$ is δ-FC, $u_2 = v$, $\|x_2 - w\| \leq \varepsilon$ for all $(x_2, w) \in R$, and (2.23), we have

$$\begin{aligned}\|\xi_{x_2u_2}(\tau) - x_2'\| \leq &\left\|\xi_{x_2u_2}(\tau) - \xi^2_{xu}(\tau)\right\| + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\ \leq &\beta_2(\|w - x_2\|, \tau) + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\ \leq &\beta_2(\varepsilon, \tau) + \left\|\xi^2_{xu}(\tau) - x_2'\right\| \\ \leq &\beta_2(\varepsilon, \tau) + \eta_2,\end{aligned} \tag{2.24}$$

which implies the existence of $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$. From (2.23),we conclude that (x_2', w') belongs to R, hence $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right) \preceq^{\varepsilon}_{\mathcal{AS}} S_\tau\left(\tilde{\Sigma}_2\right)$.

Second we prove that $S_\tau\left(\tilde{\Sigma}_2\right) \preceq^{\varepsilon}_{\mathcal{S}} S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$. Consider the relation $R \subseteq \mathbb{R}^{n-p} \times X_2$ defined by $(w, x_2) \in R$ iff

$$d(H(w), H_{q_2\delta_2}(x_2)) = \|w - x_2\| \leq \varepsilon.$$

Since for all $w \in \mathbb{R}^{n-p}$, there exists $x_2 \in X_2$ satisfying

$$\|w - x_2\| \leq \eta_2 \leq \varepsilon, \tag{2.25}$$

hence $(w, x_2) \in R$ and conditions (1-2) in Definition 2.2 are satisfied. Now we show that condition (3) in Definition 2.2 is satisfied for every $(w, x_2) \in R$. Consider any $v \in U_\tau$ we pick u_2 such that

$$\|v - u_2\| \leq \eta_1. \tag{2.26}$$

Consider the transition $w \xrightarrow{v} w' = \xi^2_{xu}(\tau)$ in $S_\tau\left(\tilde{\Sigma}_2\right)$. To prove the existence of a transition in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$ we need to show that $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$.

Consider $x_2' = [\xi_{wv}(\tau)]_{\eta_2}$, we obtain

$$\left\|\xi^2_{xu}(\tau) - x_2'\right\| = \left\|\xi^2_{xu}(\tau) - [\xi_{wv}(\tau)]_{\eta_2}\right\| \leq \eta_2. \tag{2.27}$$

Given that $\tilde{\Sigma}_2$ is δ-FC, and using (2.25), (2.26) and (2.27), we have

$$\begin{aligned}\left\|\xi_{x_2u_2}(\tau)-x_2'\right\| \leq & \left\|\xi_{x_2u_2}(\tau)-\xi_{xu}^2(\tau)\right\|+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq & \beta_2\left(\left\|w-x_2\right\|,\tau\right)+\gamma_2\left(\left\|u_2-v\right\|,\tau\right) \\ & +\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq & \beta_2(\varepsilon,\tau)+\gamma_2(\eta_1,\tau)+\left\|\xi_{xu}^2(\tau)-x_2'\right\| \\ \leq & \beta_2(\varepsilon,\tau)+\gamma_2(\eta_1,\tau)+\eta_2,\end{aligned} \tag{2.28}$$

which implies the existence of $x_2 \xrightarrow{u_2} x_2'$ in $S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$. From (2.27), we conclude that $(w', x_2') \in R$, hence $S_\tau\left(\tilde{\Sigma}_2\right) \preceq_{\mathcal{S}}^{\varepsilon} S_{q_2\delta_2}\left(\tilde{\Sigma}_2\right)$. □

5 Experimental Results

In this section we illustrate our results on two examples. First, we compare our results to the monolithic approach using a truck and trailer system, similar to the example considered in [RMT13]. We show how the proposed compositional abstraction technique scales better, as the number of trailers increases, than the monolithic approach. In the second example we synthesize a controller, using an abstraction computed with the proposed compositional approach, for the two-link biped robot, also known as the compass biped, which appeared in Section 3.4.6 in [WGC07]. All the computations were done on a 3.4 GHz iMac with 32GB of RAM.

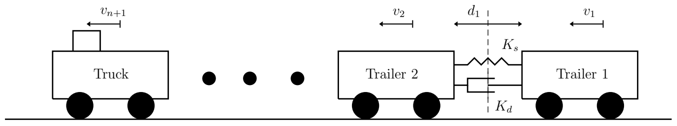


Figure 2.1: Truck and trailers system.

5.1 Truck and trailer example

Consider a truck connected to n trailers by a spring-damper system, shown in Fig. 2.1, which can be modeled by:

$$\begin{aligned}
\dot{d}_1 &= v_2 - v_1, \\
\dot{v}_1 &= \frac{K_s}{m} d_1 + \frac{K_d}{m}(v_2 - v_1), \\
\dot{d}_2 &= v_3 - v_2, \\
\dot{v}_2 &= \frac{K_s}{m} d_2 + \frac{K_d}{m}(v_3 - v_2), \\
&\vdots \\
\dot{v}_{n+1} &= u,
\end{aligned} \tag{2.29}$$

where d_i is the distance between trailers i and $i+1$, and v_i is the velocity of trailer i, for $i = 1, \cdots, n$. The spring-damper constants are denoted by K_s and K_d, m is the mass of the trailer, u is the external acceleration input acting on the truck and v_{n+1} is the velocity of the truck, respectively.

We can regard the system described by (2.29) as a cascade composition of system Σ_1 given by:

$$\begin{aligned}
\dot{d}_2 &= v_3 - v_2, \\
\dot{v}_2 &= \tfrac{K_s}{m} d_2 + \tfrac{K_d}{m}(v_3 - v_2), \\
&\vdots \\
\dot{v}_{n+1} &= u,
\end{aligned} \tag{2.30}$$

and system Σ_2:

$$\begin{aligned}
\dot{d} &= v_2 - v_1, \\
\dot{v}_1 &= \tfrac{K_s}{m} d + \tfrac{K_d}{m}(v_2 - v_1).
\end{aligned} \tag{2.31}$$

Note that Σ is partially feedback linearizable since it is a linear system. However, our

Number of trailers	1	2	3
Compositional approach	10[sec]	700[sec]	6[hr]
Monolithic approach	25[sec]	2000[sec]	>24[hr]

Table 2.1: Time spent to compute abstractions, for different number of trailers, using the compositional approach and the monolithic approach.

approach only relies on the ability of rendering the partially feedback linearizable part linear which is already the case in (2.30). Hence, we directly abstract (2.30) without designing a preliminary feedback rendering it a chain of integrators. This illustrates that our results are more general than the specific technical statement in Theorem 3.1.

We computed abstractions of system Σ_1 and Σ_2, using the MATLAB toolbox PESSOA [MDT10], for different numbers of trailers. The state space and input space discretization parameters used were $\eta = 1$ and $\mu = 1$, respectively, whereas we used $\tau = 0.5$ for the sampling time. A comparison of the time spent to construct the abstraction of the full system, for 1, 2, and 3 trailers, using the proposed compositional approach and the traditional monolithic approach is listed in Tab.2.1. Note that the addition of each trailer increases the number of continuous states by 2. As the number of trailers increases, we observe in Tab. 2.1 a speedup by a factor of 4 in the time required to compute the abstraction when we have 3 trailers. Note also that only the relative time is of significance since the implementation of PESSOA is now several years old and can be optimized in several different ways.

5.2 Compass biped robot example

Consider the compass biped robot model [WGC07], shown in Fig. 3.2, and given by:

$$\begin{aligned} \ddot{q}_1 =& v, \\ \dot{q}_2 =& \frac{\sigma_2}{D_{2,2}(q_1)} - \frac{D_{2,1}(q_1)}{D_{2,2}(q_1)}\dot{q}_1, \\ \dot{\sigma}_2 =& -G_2(q_1, q_2, \alpha), \end{aligned} \tag{2.32}$$

where q_1 is the angle between the two legs, q_2 is the angle between the stance leg and the vertical to the ground, σ_2 is the momentum conjugate to q_2, v is the actuator torque applied at the joint between the two legs of the robot, α is the ground slope, and $D(q_1)$

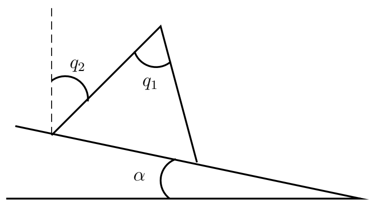


Figure 2.2: Illustration of a compass bipedal robot over sloped ground.

and $G(q_1, q_2, \alpha)$ are the inertia matrix and the gravity vector obtained from (3.58) and (3.60) in [WGC07], respectively. The parameters for this model were taken from Table 3.1 in [WGC07].

Equation (3.12) describes the motion of a biped while one of the feet is above ground. To complete the model we need to describe what happens when a foot strikes the ground. We model this phenomenon by reset map in a hybrid automaton with a single mode. The foot strikes the ground whenever:

$$q_1 \quad = \quad 2q_2. \tag{2.33}$$

Upon this event, the role of the stance and swing legs is reversed and this is captured by an instantaneous change in the states described by the reset maps:

$$\begin{bmatrix} q_1^+ \\ q_2^+ \end{bmatrix} = \Delta_q \begin{bmatrix} q_1^- \\ q_2^- \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \dot{q}_1^+ \\ \sigma_2^+ \end{bmatrix} = \Delta_{\dot{q}}(q) \begin{bmatrix} \dot{q}_1^- \\ \sigma_2^- \end{bmatrix} \tag{2.34}$$

defined by (3.54) and (3.56) in [WGC07].

Although Theorem 3.1 is not stated for hybrid systems, it can be applied to the hybrid system modeling a bipedal robot by first (compositionally) computing an abstraction of the continuous dynamics on its domain and then changing the abstraction to account for the effect of the reset map. We can regard the system described by (3.12) as a cascade composition of system Σ_1 given by:

$$\ddot{q}_1 = \quad v, \tag{2.35}$$

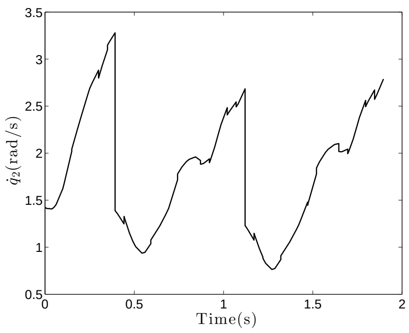
and system Σ_2:

$$\begin{aligned} \dot{q}_2 = & \ \tfrac{\sigma_2}{D_{2,2}(q_1)} - \tfrac{D_{2,1}(q_1)}{D_{2,2}(q_1)}\dot{q}_1, \\ \dot{\sigma}_2 = & \ -G_2(q_1, q_2, \alpha). \end{aligned} \tag{2.36}$$

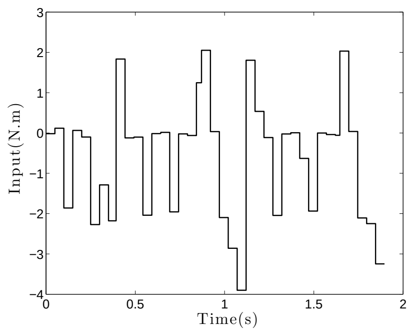
We computed abstractions of system Σ_1 and $\tilde{\Sigma}_2$ using the MATLAB toolbox PESSOA [MDT10]. For a desired precision $\varepsilon = 0.05$, the used state space and input space discretization parameters were $\eta = 0.05$ and $\mu = 0.05$, respectively, whereas we used $\tau = 0.05$ for the sampling time. The abstractions of Σ_1 and Σ_2 were computed in 20 seconds and 100 minutes, respectively, while composing them took 20 minutes. This resulted in 120 minutes to compute an abstraction compositionally. Constructing an abstraction for the full model monolithically, using the same discretization parameters, took 350 minutes. Hence, the proposed compositional approach was three times faster in this example.

In order to force the robot to move forward, $\dot{q}_2$ needs to be always greater than zero. Hence, we synthesized a controller that enforces $\dot{q}_2$ to be always greater than ε, i.e., greater than zero plus the precision of the abstraction. Fig. 2.3 shows the closed-loop simulation results and the phase portrait for the compass bipedal robot. The phase portrait indicates that non-periodic walking is achieved thereby illustrating the difference with existing design methods [McG90], [WGC07] that produce periodic gaits.

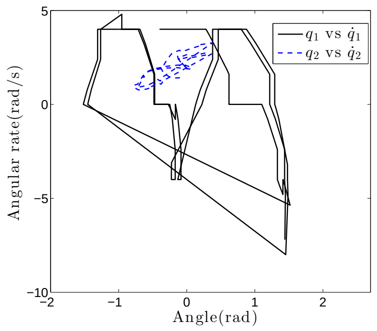
We also synthesized a controller for the same specification using the monolithic abstraction. In order to illustrate that compositional abstractions can be conservative, we compare in Figure 2.4 the number of inputs available to enforce the specification for a wide range of values of $\dot{q}_1$ and σ_2 when $q_1 = 0.4$ and $q_2 = 0.2$ rad. We can observe in Figure 2.4 that the controller synthesized using the monolithic abstraction is more permissive than the controller synthesized using our approach. However, the reduction in the available inputs is not substantial and thus only has a marginal effect in the ability to control the robot.



(a) Evolution of $\dot{q}_2$ over time.



(b) Evolution of the applied control input v over time.



(c) Phase portrait of the closed-loop system.

Figure 2.3: Closed-loop simulation using the synthesized controller.

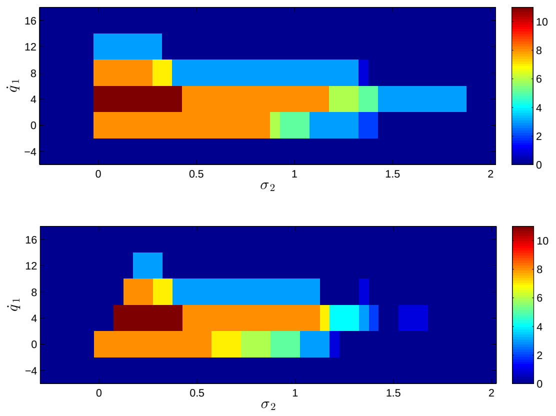


Figure 2.4: Number of inputs available to enforce the specification at the cross section $q_1 = 0.4$ rad and $q_2 = 0.2$ rad using the monolithic abstraction (top) and using the compositional abstraction (bottom).

6 Conclusion

In this chapter, we presented a compositional approach to compute abstractions of continuous time control systems that admit cascade decompositions into smaller subsystems arising from partially feedback linearizability. Although the compositional approach is more conservative than the monolithic approach, it leads to considerable speedups. Using the ball and hoop system, we illustrated that using our approach we had a 3x speedup compared to the traditional monolithic approach while being able to synthesize a controller.

CHAPTER 3

Lazy Controller Synthesis

In this chapter, we present a lazy approach to improve the scalability of abstraction-based synthesis. 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. We use a three-valued abstraction refinement approach, similarly to the approach in [DR07]. 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. We present lazy controller synthesis algorithms for safety, reachability, persistence, and recurrence 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 chapter is organized as follows. Section 1 introduces notation and the definitions of control systems and abstractions that we consider in this chapter. Our proposed algorithms for different specifications appear in Sections 2-5. We illustrate the benefits of our approach through examples in Section 6. Conclusions follow in Section 7.

1 Control systems and Abstractions

1.1 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. Given a vector $x \in \mathbb{R}^n$ and a matrix $A \in \mathbb{R}^{m \times n}$, we denote by $\|x\|$ and $\|A\|$

the infinity norm of x and A, respectively. 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.

1.2 Abstraction

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

Definition 1.1. 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. 1.1 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. 1.1 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 different specifications in the following Sections 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 $x' \in X$.

2 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 chapter 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.

2.1 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 $\Box K$, Algorithm 1 returns the set of states for which there exists a control input that enforces the specification, denoted by $[\![\Box K]\!]$. The computation of $[\![\Box K]\!]$ makes use of the set $\mathrm{Pre}_{\mathrm{T}}(Q)$ defined by:

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

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

2.2 Lazy Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ. Given the specification $\Box K$, we propose Algorithm 2 to lazily compute $[\![\Box K]\!]$. 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 $[\![\Box K]\!]$.

```
1: function STAY($T$, $K$)
2:     $Q \leftarrow K$
3:     repeat
4:         $Q' \leftarrow \mathrm{Pre}_\mathrm{T}(Q)$
5:         $Q \leftarrow Q' \cap Q$
6:     until fixed point on $Q$ is reached
7:     return $Q$
8: end function
```

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

```
1: function LAZYSTAY($T'$, $K$)
2:     $V \leftarrow K$
3:     repeat
4:         $\underline{V} \leftarrow \underline{\mathrm{Pre}}_\mathrm{T}(V)$
5:         $\overline{V} \leftarrow \overline{\mathrm{Pre}}_\mathrm{T}(V)$
6:         $V \leftarrow \overline{V} \cap V$
7:         $T' \leftarrow \mathrm{Refine}_N(T', \overline{V} - \underline{V})$
8:     until fixed point on $V$ is reached and $\overline{V} = \underline{V}$
9:     return $V$
10: end function
```

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 6 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 4.1 to prove termination and Lemma 4.2 to show that Algorithm 2 computes the largest controlled invariant set in K, i.e., the set $[\![\Box K]\!]$.

Lemma 2.1. If the set of states X and the set of inputs U are finite, Algorithm 2 terminates in finite time.

Proof. Let X be the maximal set of states in the abstraction, i.e., the set of sates in an abstraction where all the transitions have been computed. This set is finite and since

$\underline{V} \subseteq X$ and $\overline{V} \subseteq X$, the sets $\underline{V}$ and $\overline{V}$ are also finite. This implies that there are finitely many sets $\overline{V} - \underline{V}$. Since, the function Refine monotonically decreases the number of transitions mapped to $1/2$, and since the set of inputs U is finite, after finitely many steps all the transitions will be mapped to 0 or 1, even if, in the worst case, all the sets of the form $\overline{V} - \underline{V}$ are generated by the algorithm. Once all the transitions are mapped to 0 or 1, the algorithm is reduced to lines 4 and 6 and is thus equivalent to algorithm 1 (simply map line 4 in Alg. 2 to line 4 in Alg. 1 and line 6 in Alg. 2 to line 5 in Alg. 1). Since algorithm 1 terminates in finitely many steps [MPS95], so does algorithm 2. □

Lemma 2.2. If Algorithms 1 and 2 run on the same input, we have $Q = V$ upon termination of each Algorithm.

Proof. As shown in the proof of Lemma 4.1, after finitely many steps (say j) Algorithm 2 becomes identical to Algorithm 1. Therefore, the output of Algorithm 2 can be produced by running algorithm 1 on the input V_j, i.e., on the set V_j computed by Algorithm 2 at iteration j, once we have $\overline{V} = \underline{V}$. If V_j contains the largest controlled invariant set in K, then algorithm 2 produces the same output on K or V_j (since the largest controlled invariant set in K and V_j is the same). Hence, to finish the proof we only need to establish that V_j contains the largest controlled invariant set in K and it is sufficient to establish that $Q_i \subseteq V_i$ where Q_i and V_i are the sets Q and V computed at iteration i of Algorithms 1 and 2, respectively. For $i = 0$ we have $Q_0 = K \subseteq K = V_0$. Assume now that $Q_{i-1} \subseteq V_{i-1}$. We observe that it follows from the definition of Pre that $Q_{i-1} \subseteq V_{i-1}$ implies $\mathrm{Pre}_{\mathrm{T}}(Q_{i-1}) \subseteq \mathrm{Pre}_{\mathrm{T}}(V_{i-1})$. Accordingly, we have:

$$Q_i' = \mathrm{Pre}_{\mathrm{T}}(Q_{i-1}) \subseteq \mathrm{Pre}_{\mathrm{T}}(V_{i-1}). \tag{3.1}$$

Let T be a transition relation where all the transitions are mapped to 0 or 1 only and let T' be a lazy over-approximation of T, i.e., all the transitions that T maps to 1 are mapped by T' to 1 or $1/2$. Since a transition that is mapped to 1 in T will be either mapped to 1/2 or 1 in T', we have:

$$\mathrm{Pre}_{\mathrm{T}}(V_{i-1}) \subseteq \overline{\mathrm{Pre}}_{\mathrm{T}}(V_{i-1}) = \overline{V_i}. \tag{3.2}$$

From (3.5) and (3.6), we obtain:

$$Q_i = Q'_i \cap Q_{i-1} \subseteq \overline{V_i} \cap V_{i-1} = V_i.$$

□

The next result summarizes the consequences of Lemmas 2.1 and 2.2.

Theorem 2.3. 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.

```
function Refine$_N$($T'$,$V$)
    for $i := 1, \cdots, N$ do
        $x :\in V$
        $V \leftarrow V \setminus \{x\}$
        $u :\in \{u \in U \mid \forall x' \in X,\ T'(x,u,x') = 1/2\}$
        for each $x' \in X$ do
            $T'(x,u,x') \leftarrow 1$ if $x' \in \mathrm{Post}_u(x)$
            $T'(x,u,x') \leftarrow 0$ if $x' \notin \mathrm{Post}_u(x)$
        end for
    end for
    return $T'$
end function
```

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

3.1 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 $[\![\Diamond K]\!]$.

3.2 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.

Lemma 3.1. If the set of states X and the set of inputs U are finite, Algorithm 5 terminates in finite time.

Proof. Let X be the maximal set of states in the abstraction, i.e., the set of sates in an abstraction where all the transitions have been computed. This set is finite and since $\underline{V} \subseteq X$ and $\overline{V} \subseteq X$, the sets $\underline{V}$ and $\overline{V}$ are also finite. This implies that there are finitely many sets $\overline{V} - \underline{V}$. Since, the function Refine monotonically decreases the number of transitions mapped to $1/2$, and since the set of inputs U is finite, after finitely many steps all the transitions will be mapped to 0 or 1, even if, in the worst case, all the sets of the form $\overline{V} - \underline{V}$ are generated by the algorithm. Once all the transitions are mapped to 0 or 1, the algorithm is reduced to lines 4 and 6 and is thus equivalent to algorithm 4 (simply map line 4 in Alg. 5 to line 4 in Alg. 4 and line 6 in Alg. 5 to line 5 in Alg. 4). Since algorithm 4 terminates in finitely many steps [MPS95], so does algorithm 5. □

Lemma 3.2. If Algorithms 4 and 5 run on the same input, we have $Q = V$ upon termination of each Algorithm.

Proof. As shown in the proof of Lemma 3.1, after finitely many steps (say j) Algorithm 5 becomes identical to Algorithm 4. Therefore, the output of Algorithm 5 can be produced by running algorithm 4 on the input V_j, i.e., on the set V_j computed by Algorithm 5 at iteration j, once we have $\overline{V} = \underline{V}$. If V_j contains the largest reachable set towards K, then algorithm 5 produces the same output on K or V_j (since the largest reachable set towards K and V_j is the same). Hence, to finish the proof we only need to establish that V_j is a subset of the largest reachable set towards K and it is sufficient to establish that $V_i \subseteq Q_i$ where Q_i and V_i are the sets Q and V computed at iteration i of Algorithms 4 and 5, respectively. For $i = 0$ we have $V_0 = K \subseteq K = Q_0$. Assume now that $V_{i-1} \subseteq Q_{i-1}$. We observe that it follows from the definition of Pre that $V_{i-1} \subseteq Q_{i-1}$ implies $\mathrm{Pre}_{\mathrm{T}}(V_{i-1}) \subseteq \mathrm{Pre}_{\mathrm{T}}(Q_{i-1})$. Accordingly, we have:

$$Q'_i = \mathrm{Pre}_{\mathrm{T}}(Q_{i-1}) \supseteq \mathrm{Pre}_{\mathrm{T}}(V_{i-1}). \tag{3.3}$$

Let T be a transition relation where all the transitions are mapped to 0 or 1 only and let T' be a lazy over-approximation of T, i.e., all the transitions that T maps to 1 are mapped by T' to 1 or 1/2. Since a transition that is mapped to 1 in T will be either mapped to 1/2 or 1 in T', we have:

$$\mathrm{Pre}_{\mathrm{T}}(V_{i-1}) \supseteq \underline{\mathrm{Pre}}_{\mathrm{T}}(V_{i-1}) = \underline{V_i}. \tag{3.4}$$

From (3.5) and (3.6), we obtain:

$$Q_i = Q'_i \cup Q_{i-1} \supseteq \underline{V_i} \cup V_{i-1} = V_i.$$

□

The next result summarizes the consequences of Lemmas 3.1 and 3.2.

Theorem 3.3. Algorithm 5 terminates in finite time and upon termination returns the largest reachable set towards K.

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

```
1: function Reach($T$, $K$)
2:     $Q \leftarrow K$
3:     repeat
4:         $Q' \leftarrow \mathrm{Pre}_{\mathrm{T}}(Q)$
5:         $Q \leftarrow Q' \cup Q$
6:     until fixed point on $Q$ is reached
7:     return $Q$
8: end function
```

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

```
1: function LazyReach($T'$, $K$)
2:     $V \leftarrow K$
3:     repeat
4:         $\underline{V} \leftarrow \underline{\mathrm{Pre}}_{\mathrm{T}}(V)$
5:         $\overline{V} \leftarrow \overline{\mathrm{Pre}}_{\mathrm{T}}(V)$
6:         $V \leftarrow \underline{V} \cup V$
7:         $T' \leftarrow \mathrm{Refine}_N(T', \overline{V} - \underline{V})$
8:     until fixed point on $V$ is reached and $\overline{V} = \underline{V}$
9:     return $V$
10: end function
```

4 Controller Synthesis for Persistence Specifications

In this section we present two algorithms to synthesize a controller for persistence specification, i.e., eventually reach a desired set and stay therein for all future time. The first algorithm is classical and assumes the existence of an abstraction. The second algorithm is another contribution of this chapter 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 same controlled invariant set as the classical algorithm.

4.1 Classical Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ. Given an "eventually always K" specification, denoted by $\Diamond\Box K$, Algorithm 6 returns the set of states for which there exists a control input that enforces the specification, denoted by $[\![\Diamond\Box K]\!]$. Algorithm 6 consists of an inner loop, where the set Q is updated until a fixed-point is reached, and an outer loop updating R until a fixed point is reached.

4.2 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 $\Diamond\Box K$, we propose Algorithm 7 to compute $[\![\Diamond\Box K]\!]$.

Algorithm 6 Computation of $[\![\Diamond\Box K]\!]$.

```
1:  function ReachStay($T$, $K$)
2:      $R \leftarrow \emptyset$
3:      repeat
4:          $Q \leftarrow K$
5:          repeat
6:              $R' \leftarrow \mathrm{Pre_T}(R)$
7:              $Q' \leftarrow \mathrm{Pre_T}(Q)$
8:              $Q \leftarrow (Q' \cap Q) \cup R'$
9:          until fixed point on $Q$ is reached
10:         $R \leftarrow Q$
11:     until fixed point on $R$ is reached
12:     return $R$
13: end function
```

Algorithm 7 Lazy Computation of $[\![\Diamond\Box K]\!]$.

```
1: function ReachStay($T'$, $K$)
2:     $W \leftarrow \emptyset$
3:     repeat
4:         $V \leftarrow K$
5:         repeat
6:             repeat
7:                 $\underline{W} \leftarrow \underline{\mathrm{Pre}}_{\mathrm{T}}(W)$
8:                 $\overline{W} \leftarrow \overline{\mathrm{Pre}}_{\mathrm{T}}(W)$
9:                 $W' \leftarrow \underline{W} \cup \overline{W}$
10:                $T' \leftarrow \mathrm{Refine}_N(T', \overline{W} - \underline{W})$
11:            until $\overline{W} = \underline{W}$
12:            $\underline{V} \leftarrow \underline{\mathrm{Pre}}_{\mathrm{T}}(V)$
13:            $\overline{V} \leftarrow \overline{\mathrm{Pre}}_{\mathrm{T}}(V)$
14:            $V \leftarrow (\overline{V} \cap V) \cup W'$
15:            $T' \leftarrow \mathrm{Refine}_N(T', \overline{V} - \underline{V})$
16:        until fixed point on $V$ is reached and $\overline{V} = \underline{V}$
17:        $W \leftarrow V$
18:    until fixed point on $W$ is reached
19:    return $W$
20: end function
```

We prove that Algorithm 7 terminates in finite time and upon termination it returns the largest invariant set in Theorem 4.5, by showing termination conditions as well as the relation between the sets it computes and those from the classical algorithm in the following Lemmas. We refer to lines $\{5-9\}$ in Algorithm 6 and lines $\{5-16\}$ in Algorithm 7 as the inner loop. Also, we refer to lines $\{3-11\}$ in Algorithm 6 and lines $\{3-18\}$ in Algorithm 7 as the outer loop.

Lemma 4.1. If the set of states X and the set of inputs U are finite, the inner loop in Algorithm 7 terminates in finite time.

Proof. First, we show that the loop defined by the lines $\{6-11\}$ in Algorithm 7 terminates in finite time. Let X be the maximal set of states in the abstraction, i.e., the set of sates in an abstraction where all the transitions have been computed. This set is finite and since $\underline{W} \subseteq X$ and $\overline{W} \subseteq X$, the sets $\underline{W}$ and $\overline{W}$ are also finite. This implies that there are finitely many sets $\overline{W} - \underline{W}$. Since, the function Refine monotonically decreases the number of transitions mapped to $1/2$, and since the set of inputs U is finite, after finitely many steps all the transitions will be mapped to 0 or 1, even if, in the worst case, all the sets of the form $\overline{W} - \underline{W}$ are generated by the algorithm. Once all the transitions are mapped to 0 or 1, the loop defined by the lines $\{6 - 11\}$ is reduced to lines $\{7\}$ and $\{9\}$ which is equivalent to line $\{6\}$ in algorithm 6.

Once we have $\overline{W} = \underline{W}$, the inner loop in Algorithm 7 is reduced to the lines $\{7\}$, $\{9\}$ and $\{12 - 16\}$. By following the same argument we used for $\overline{W}$ and $\underline{W}$, on $\overline{V}$ and $\underline{V}$, after finitely many steps we will have $\overline{V} = \underline{V}$. Once $\overline{V} = \underline{V}$, the inner loop is reduced once more to the lines $\{7\}$, $\{9\}$, $\{12\}$ and $\{14\}$ and is thus equivalent to the inner loop in Algorithm 6 (simply map line $\{7\}$ and $\{9\}$ in Alg. 7 to line $\{6\}$ in Alg. 6 and lines $\{12\}$ and $\{14\}$ in Alg. 7 to lines $\{7\}$ and $\{8\}$ in Alg. 6). Since the inner loop in Algorithm 6 terminates in finitely many steps, so does algorithm 7. □

Lemma 4.2. If Algorithms 6 and 7 run on the same input, we have $Q = V$ upon termination of the inner loop of each algorithm.

Proof. Given that both algorithms run on the same input, we have $W = R$. From the proof of Lemma 4.1, we know that after finite number of steps the loop in Algorithm 7 defined by the lines $\{6 - 11\}$ is reduced to lines $\{7\}$ and $\{9\}$ which is equivalent to line $\{6\}$ in algorithm 6. Since neither W nor R are updated inside the inner loop in the two algorithms, we conclude that after finite number of steps, we have $W' = R'$. Also, as shown in the proof of Lemma 4.1, after finitely many steps (say j) the inner loop in Algorithm 7 becomes identical to the inner loop in Algorithm 6. Therefore, the output of the inner loop in Algorithm 7 can be produced by running the inner loop in algorithm 6 on the input V_j, i.e., on the set V_j computed by the inner loop in Algorithm 7 at iteration j, once we have $\overline{V} = \underline{V}$. If V_j contains the largest controlled invariant set in K, then the inner loop in algorithm 7 produces the same output on K or V_j (since the largest controlled invariant set in K and V_j is the same). Hence, to finish the proof we only need to establish that V_j contains the largest controlled invariant set in K and it is sufficient to establish that $Q_i \subseteq V_i$ where Q_i and V_i are the sets Q and V computed at iteration i of the inner loops in Algorithms 6 and 7, respectively. For $i = 0$ we have $Q_0 = K \subseteq K = V_0$. Assume now that $Q_{i-1} \subseteq V_{i-1}$. We observe that it follows from the definition of Pre that

$Q_{i-1} \subseteq V_{i-1}$ implies $\mathrm{Pre}_{\mathrm{T}}(Q_{i-1}) \subseteq \mathrm{Pre}_{\mathrm{T}}(V_{i-1})$. Accordingly, we have:

$$Q'_i = \mathrm{Pre}_{\mathrm{T}}(Q_{i-1}) \subseteq \mathrm{Pre}_{\mathrm{T}}(V_{i-1}). \tag{3.5}$$

Let T be a transition relation where all the transitions are mapped to 0 or 1 only and let T' be a lazy over-approximation of T, i.e., all the transitions that T maps to 1 are mapped by T' to 1 or 1/2. Since a transition that is mapped to 1 in T will be either mapped to 1/2 or 1 in T', we have:

$$\mathrm{Pre}_{\mathrm{T}}(V_{i-1}) \subseteq \overline{\mathrm{Pre}}_{\mathrm{T}}(V_{i-1}) = \overline{V_i}. \tag{3.6}$$

From (3.5) and (3.6) and $W' = R'$, we obtain:

$$Q_i = (Q'_i \cap Q_{i-1}) \cup R' \subseteq (\overline{V_i} \cap V_{i-1}) \cup W' = V_i.$$

□

Lemma 4.3. Given that Algorithm 6 and Algorithm 7 are starting from the same initial conditions, the sets R and W computed at each iteration of the outer loop in Algorithm 6 and Algorithm 7 are equivalent.

Proof. Using Lemma 4.1 and Lemma 4.2, we know that the inner loop in Algorithm 7 terminates in finite time and upon termination the fixed point on V is equal to the fixed point on Q, obtained from running the inner loop in Algorithm 6, starting from the same initial conditions. Since, at each iteration of the outer loop in Algorithm 6 and Algorithm 7, R and W are updated with the value of Q and V, respectively, and since $V = Q$, we deduce that $W = R$. □

Lemma 4.4. Given that the set of states X and the set of inputs U are finite, the outer loop in Algorithm 7 terminates in finite time.

Proof. Using Lemmas 4.1, 4.2, 4.3, we know that the inner loop in Algorithm 7 terminates in finite time and at each iteration of the outer loop we have $Q = V$ and $R = W$. Since the outer loop in Algorithm 6 terminates in finite time if X and U are finite and since $Q = V$ and $R = W$ at each iteration of the outer loop in Algorithm 7, a fixed point on W is reached in finite time. Accordingly, the outer loop in Algorithm 7 terminates in finite time.

$\square$

The next result summarizes the consequences of the previous Lemmas.

Theorem 4.5. Given that the set of states X and the set of inputs U are finite, Algorithm 7 terminates in finite time and upon termination returns the largest controlled invariant subset of K as Algorithm 6.

5 Controller Synthesis for Recurrence Specifications

In this section we present two algorithms to synthesize a controller for recurrence specification, i.e., reach a desired set infinitely often. The first algorithm is classical and assumes the existence of an abstraction. The second algorithm is last contribution of this chapter 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 same controlled invariant set as the classical algorithm.

5.1 Classical Algorithm

Let $S(\Sigma)$ be an abstraction of a control system Σ. Given an "always eventually K" specification, denoted by $\Box\Diamond K$, Algorithm 8 returns the set of states for which there exists a control input that enforces the specification, denoted by $[\![\Box\Diamond K]\!]$. Algorithm 8 consists of an inner loop, where the set Q is updated until a fixed-point is reached, and an outer loop updating R until a fixed point is reached.

5.2 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 $\Box\Diamond K$, we propose Algorithm 9 to compute $[\![\Box\Diamond K]\!]$.

Algorithm 8 Computation of $[\![\Box\Diamond K]\!]$.

```
1: function AlwaysReach($T$, $K$)
2:     $R \leftarrow \emptyset$
3:     repeat
4:         $Q \leftarrow K$
5:         repeat
6:             $R' \leftarrow \mathrm{Pre}_{\mathrm{T}}(R)$
7:             $Q' \leftarrow \mathrm{Pre}_{\mathrm{T}}(Q)$
8:             $Q \leftarrow (Q' \cup Q) \cap R'$
9:         until fixed point on $Q$ is reached
10:        $R \leftarrow Q$
11:    until fixed point on $R$ is reached
12:    return $R$
13: end function
```

Algorithm 9 Lazy Computation of $[\![\Box\Diamond K]\!]$.

```
1: function AlwaysReach($T'$, $K$)
2:     $W \leftarrow \emptyset$
3:     repeat
4:         $V \leftarrow K$
5:         repeat
6:             repeat
7:                 $\underline{W} \leftarrow \underline{\mathrm{Pre}}_{\mathrm{T}}(W)$
8:                 $\overline{W} \leftarrow \overline{\mathrm{Pre}}_{\mathrm{T}}(W)$
9:                 $W' \leftarrow \underline{W} \cup \overline{W}$
10:                $T' \leftarrow \mathrm{Refine}_N(T', \overline{W} - \underline{W})$
11:            until $\overline{W} = \underline{W}$
12:            $\underline{V} \leftarrow \underline{\mathrm{Pre}}_{\mathrm{T}}(V)$
13:            $\overline{V} \leftarrow \overline{\mathrm{Pre}}_{\mathrm{T}}(V)$
14:            $V \leftarrow (\underline{V} \cup V) \cap W'$
15:            $T' \leftarrow \mathrm{Refine}_N(T', \overline{V} - \underline{V})$
16:        until fixed point on $V$ is reached and $\overline{V} = \underline{V}$
17:        $W \leftarrow V$
18:    until fixed point on $W$ is reached
19:    return $W$
20: end function
```

We prove that Algorithm 9 terminates in finite time and upon termination it returns the largest invariant set in Theorem 5.5, by showing termination conditions as well as the relation between the sets it computes and those from the classical algorithm in the following Lemmas. We refer to lines $\{5-9\}$ in Algorithm 8 and lines $\{5-16\}$ in Algorithm 9 as the inner loop. Also, we refer to lines $\{3-11\}$ in Algorithm 8 and lines $\{3-18\}$ in Algorithm 9 as the outer loop.

Lemma 5.1. If the set of states X and the set of inputs U are finite, the inner loop in Algorithm 7 terminates in finite time.

Proof. First, we show that the loop defined by the lines $\{6-11\}$ in Algorithm 9 terminates in finite time. Let X be the maximal set of states in the abstraction, i.e., the set of sates in an abstraction where all the transitions have been computed. This set is finite and since $\underline{W} \subseteq X$ and $\overline{W} \subseteq X$, the sets $\underline{W}$ and $\overline{W}$ are also finite. This implies that there are finitely many sets $\overline{W} - \underline{W}$. Since, the function Refine monotonically decreases the number of transitions mapped to $1/2$, and since the set of inputs U is finite, after finitely many steps all the transitions will be mapped to 0 or 1, even if, in the worst case, all the sets of the form $\overline{W} - \underline{W}$ are generated by the algorithm. Once all the transitions are mapped to 0 or 1, the loop defined by the lines $\{6-11\}$ is reduced to lines $\{7\}$ and $\{9\}$ which is equivalent to line $\{6\}$ in algorithm 8.

Once we have $\overline{W} = \underline{W}$, the inner loop in Algorithm 9 is reduced to the lines $\{7\}$, $\{9\}$ and $\{12-16\}$. By following the same argument we used for $\overline{W}$ and $\underline{W}$, on $\overline{V}$ and $\underline{V}$, after finitely many steps we will have $\overline{V} = \underline{V}$. Once $\overline{V} = \underline{V}$, the inner loop is reduced once more to the lines $\{7\}$, $\{9\}$, $\{12\}$ and $\{14\}$ and is thus equivalent to the inner loop in Algorithm 8 (simply map line $\{7\}$ and $\{9\}$ in Alg. 9 to line $\{6\}$ in Alg. 8 and lines $\{12\}$ and $\{14\}$ in Alg. 9 to lines $\{7\}$ and $\{8\}$ in Alg. 8). Since the inner loop in Algorithm 8 terminates in finitely many steps, so does algorithm 9. □

Lemma 5.2. If Algorithms 6 and 7 run on the same input, we have $Q = V$ upon termination of the inner loop of each algorithm.

Proof. Given that both algorithms run on the same input, we have $W = R$. From the proof of Lemma 5.1, we know that after finite number of steps the loop in Algorithm

9 defined by the lines $\{6 - 11\}$ is reduced to lines $\{7\}$ and $\{9\}$ which is equivalent to line $\{6\}$ in algorithm 8. Since neither W nor R are updated inside the inner loop in the two algorithms, we conclude that after finite number of steps, we have $W' = R'$. Also, as shown in the proof of Lemma 5.1, after finitely many steps (say j) the inner loop in Algorithm 9 becomes identical to the inner loop in Algorithm 8. Therefore, the output of the inner loop in Algorithm 9 can be produced by running the inner loop in algorithm 8 on the input V_j, i.e., on the set V_j computed by the inner loop in Algorithm 9 at iteration j, once we have $\overline{V} = \underline{V}$. If V_j is a subset of the largest reachable set towards K, then the inner loop in algorithm 9 produces the same output on K or V_j (since the largest reachable set towards K and V_j is the same). Hence, to finish the proof we only need to establish that V_j is a subset of the largest reachable set towards K and it is sufficient to establish that $V_i \subseteq Q_i$ where Q_i and V_i are the sets Q and V computed at iteration i of the inner loops in Algorithms 8 and 9, respectively. For $i = 0$ we have $V_0 = K \subseteq K = Q_0$. Assume now that $V_{i-1} \subseteq Q_{i-1}$. We observe that it follows from the definition of Pre that $V_{i-1} \subseteq Q_{i-1}$ implies $\mathrm{Pre}_\mathrm{T}(V_{i-1}) \subseteq \mathrm{Pre}_\mathrm{T}(Q_{i-1})$. Accordingly, we have:

$$Q'_i = \mathrm{Pre}_\mathrm{T}(Q_{i-1}) \supseteq \mathrm{Pre}_\mathrm{T}(V_{i-1}). \tag{3.7}$$

Let T be a transition relation where all the transitions are mapped to 0 or 1 only and let T' be a lazy over-approximation of T, i.e., all the transitions that T maps to 1 are mapped by T' to 1 or 1/2. Since a transition that is mapped to 1 in T will be either mapped to 1/2 or 1 in T', we have:

$$\mathrm{Pre}_\mathrm{T}(V_{i-1}) \supseteq \overline{\mathrm{Pre}}_\mathrm{T}(V_{i-1}) = \underline{V_i}. \tag{3.8}$$

From (3.7) and (3.8) and $W' = R'$, we obtain:

$$Q_i = (Q'_i \cup Q_{i-1}) \cap R' \supseteq (\underline{V_i} \cup V_{i-1}) \cap W' = V_i.$$

□

Lemma 5.3. Given that Algorithm 8 and Algorithm 9 are starting from the same initial conditions, the sets R and W computed at each iteration of the outer loop in Algorithm 8 and Algorithm 9 are equivalent.

Proof. Using Lemma 5.1 and Lemma 5.2, we know that the inner loop in Algorithm 7 terminates in finite time and upon termination the fixed point on V is equal to the fixed point on Q, obtained from running the inner loop in Algorithm 6, starting from the same initial conditions. Since, at each iteration of the outer loop in Algorithm 8 and Algorithm 9, R and W are updated with the value of Q and V, respectively, and since $V = Q$, we deduce that $W = R$. □

Lemma 5.4. Given that the set of states X and the set of inputs U are finite, the outer loop in Algorithm 9 terminates in finite time.

Proof. Using Lemmas 5.1, 5.2, 5.3, we know that the inner loop in Algorithm 9 terminates in finite time and at each iteration of the outer loop we have $Q = V$ and $R = W$. Since the outer loop in Algorithm 8 terminates in finite time if X and U are finite and since $Q = V$ and $R = W$ at each iteration of the outer loop in Algorithm 9, a fixed point on W is reached in finite time. Accordingly, the outer loop in Algorithm 9 terminates in finite time.

□

The next result summarizes the consequences of the previous Lemmas.

Theorem 5.5. Given that the set of states X and the set of inputs U are finite, Algorithm 9 terminates in finite time and upon termination returns the largest controlled invariant subset of K as Algorithm 8.

6 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 [MDT10] and SCOTS [RZ16]. 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 [Ame11] 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.

6.1 Unicycle navigation example

Consider the unicycle vehicle, which can be modeled by:

$$\dot{x} = v\cos(\theta) \tag{3.9}$$

$$\dot{y} = v\sin(\theta) \tag{3.10}$$

$$\dot{\theta} = \omega, \tag{3.11}$$

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. 3.1. 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. 3.1 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 [MDT10] as well as SCOTS [RZ16]. Tables 3.1 and 3.2 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

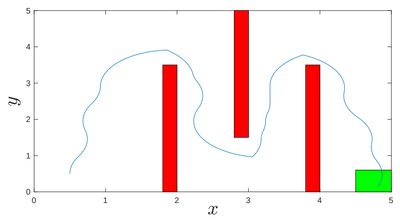


Figure 3.1: Closed loop simulation result using the controller synthesized with lazy Algorithms 2 and 5.

time to compute the abstraction and synthesize the controller, respectively, in seconds. We observe in Tab. 3.1 that when $N = 2000$, we can achieve a speedup of up to 4 and 85 times compared to SCOTS and PESSOA, respectively. In Tab. 3.2, 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.

	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 3.1: 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 3.2: Comparison of Algorithm 4 in PESSOA and SCOTS and Algorithm 5.

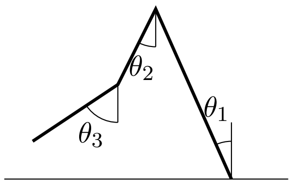


Figure 3.2: A kneed biped over horizontal ground.

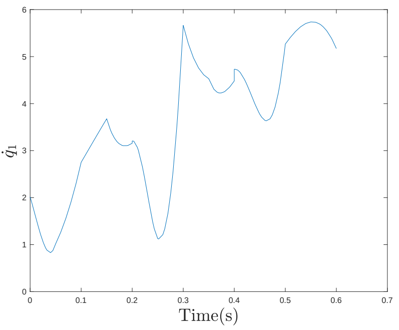


Figure 3.3: Closed-loop simulation results, showing the evolution of $\dot{q}_1$ over time, using the controller synthesized with Algorithm 2.

6.2 Biped robot example

Consider the kneed biped robot model, shown in Fig. 3.2, and given by:

$$M(\theta)\ddot{\theta} + C(\dot{\theta}, \theta)\dot{\theta} + G(\theta) = Bu, \tag{3.12}$$

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 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 (3.12) 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 [Ame11].

We synthesized a controller that forces the robot to always move forward, which is cap-

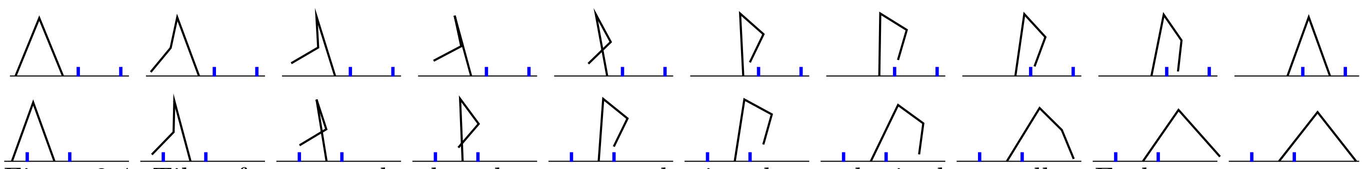


Figure 3.4: Tiles of two steps by the robot generated using the synthesized controller. Each row represents the tiles of a single step.

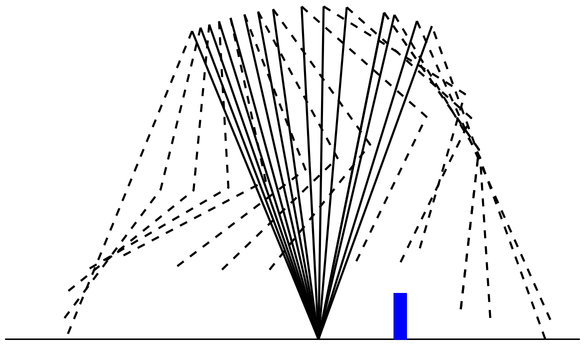


Figure 3.5: A walking gait generated using the synthesized controller. The stance leg is shown as a solid line, whereas the swing leg is shown as a dashed line.

tured 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 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 [RWR17] 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. 3.3 shows the closed-loop simulation results whereas Fig. 3.4 and 3.5 show tiles of two steps and one of the resulting walking gaits using the controller synthesized for the kneed biped model, respectively.

7 Conclusion

In this chapter 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 algorithms for safety, reachability, persistence, and recurrence specifications which are guaranteed to terminate in finite time and upon termination 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.

CHAPTER 4

Future work

As discussed in Chapter 1, the main drawback of abstraction based control software synthesis is the lack of scalability in the computation of the abstractions. Consequently, it becomes infeasible to compute abstractions for large systems. We presented two different approaches to enhance the scalability of correct-by-construction control software synthesis: 1) exploiting system structure and 2) lazy controller synthesis. In this chapter, future directions in each of these approaches are proposed.

1 Abstracting partially feedback linearizable systems compositionally

We presented an approach that exploits the system structure to compute abstractions compositionally for control systems that are partially feedback linearizable. While a controller synthesized for the composed abstraction can be refined to a controller for the original control system, non existence of a controller for the abstraction does not imply the non existence of a controller for the original system. Given that our compositional approach is more conservative than the monolithic approach, we might not be able to find a controller. It is then important to investigate similar results under the more stringent assumption of incrementally input-to-state stability since in this case we can guarantee the existence of an alternating approximate bisimulation between the original control system and its compositional abstraction. The existence of an alternating approximate bisimulation relation provides guarantees that the non existence of a controller for the abstraction implies the non existence of a controller for the original system. Such results would reduce conservatism in the composed abstractions at the expense of having more restrictive assumptions on control systems.

2 Lazy controller synthesis

All the lazy algorithms in Chapter 3 use Algorithm 3 to compute a fragment of the abstraction for N states. As can be seen from the presented unicycle example, the choice of how many states are refined in the refinement block decreases the execution time drastically. Therefore, it would be important to investigate how to optimize performance based on the choice of the parameter N.

It is also important to develop similar results for mode-target [BVT15], and GR(1) [BJP12] specifications by following the same approach. We start with an abstraction having may-states that are refined incrementally as needed. At each iteration, we would expect to have over- and under-approximation sets of states satisfying the specifications where a fixed point is reached when they are equal.

Finally, the results presented in Chapter 3 can be integrated with other specification-guided approaches, such as those presented in [HMM18] by lazily computing several abstractions with different precisions. Controller synthesis would start by lazily computing the abstraction with coarsest precision and move to finer precision if needed. Hence, it will enable the controller synthesis for larger control systems.

References

[Ame11] Aaron 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*, pp. 163–172. ACM, 2011.

[AS99] David Angeli and Eduardo D Sontag. "Forward completeness, unboundedness observability, and their Lyapunov characterizations." *Systems & Control Letters*, **38**(4):209–217, 1999.

[BJP12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of reactive (1) designs." *Journal of Computer and System Sciences*, **78**(3):911–938, 2012.

[BVT15] Ayca Balkan, Moshe Vardi, and Paulo Tabuada. "Controller synthesis for mode-target games." *IFAC-PapersOnLine*, **48**(27):343–350, 2015.

[CGJ00] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. "Counterexample-guided abstraction refinement." In *Computer aided verification*, pp. 154–169. Springer, 2000.

[DR07] Luca De Alfaro and Pritam Roy. "Solving games via three-valued abstraction refinement." In *CONCUR 2007–Concurrency Theory*, pp. 74–89. Springer, 2007.

[GP07] Antoine Girard and George J Pappas. "Approximation metrics for discrete and continuous systems." *Automatic Control, IEEE Transactions on*, **52**(5):782–798, 2007.

[HJM03] Thomas A Henzinger, Ranjit Jhala, and Rupak Majumdar. *Counterexample-guided control.* Springer, 2003.

[HMM18] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin 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.

[Kha96] Hassan K Khalil. *Noninear Systems.* Prentice-Hall, New Jersey, 1996.

[McG90] Tad McGeer et al. "Passive dynamic walking." *I. J. Robotic Res.*, **9**(2):62–82, 1990.

[MDT10] Manuel Mazo Jr, Anna Davitian, and Paulo Tabuada. "Pessoa: A tool for embedded controller synthesis." In *Computer Aided Verification*, pp. 566–569. Springer, 2010.

[MGG13] Sebti Mouelhi, Antoine Girard, and Gregor Gössler. "CoSyMA: a tool for controller synthesis using multi-scale abstractions." In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pp. 83–88. ACM, 2013.

[MGW15] Pierre-Jean Meyer, Antoine Girard, and Emmanuel Witrant. "Safety control with performance guarantees of cooperative systems using compositional abstractions." *IFAC-PapersOnLine*, **48**(27):317–322, 2015.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. "On the synthesis of discrete controllers for timed systems." In *Annual Symposium on Theoretical Aspects of Computer Science*, pp. 229–242. Springer, 1995.

[NO16] Petter Nilsson and Necmiye 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*, pp. 205–214. ACM, 2016.

[PGT08] Giordano Pola, Antoine Girard, and Paulo Tabuada. "Approximately bisimilar symbolic models for nonlinear control systems." *Automatica*, **44**(10):2508–2516, 2008.

[PPD10] Giordano Pola, Pierdomenico Pepe, Maria D Di Benedetto, and Paulo Tabuada. "Symbolic models for nonlinear time-delay systems using approximate bisimulations." *Systems & Control Letters*, **59**(6):365–373, 2010.

[PPD14] Giordano Pola, Pierdomenico Pepe, and Maria Domenica Di Benedetto. "Symbolic models for networks of discrete-time nonlinear control systems." In *American Control Conference (ACC), 2014*, pp. 1787–1792. IEEE, 2014.

[PT09] Giordano Pola and Paulo Tabuada. "Symbolic models for nonlinear control systems: Alternating approximate bisimulations." *SIAM Journal on Control and Optimization*, **48**(2):719–733, 2009.

[Rei10] Gunther Reißig. "Abstraction based solution of complex attainability problems for decomposable continuous plants." In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 5911–5917. IEEE, 2010.

[RMT13] Matthias Rungger, Manuel Mazo Jr, and Paulo Tabuada. "Specification-guided controller synthesis for linear systems and safe linear-time temporal logic." In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pp. 333–342. ACM, 2013.

[RWR17] Gunther Reissig, Alexander Weber, and Matthias Rungger. "Feedback refinement relations for the synthesis of symbolic controllers." *IEEE Transactions on Automatic Control*, **62**(4):1781–1796, 2017.

[RZ16] Matthias Rungger and Majid 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.

[SJZ18] Adnane Saoud, Pushpak Jagtap, Majid Zamani, and Antoine Girard. "Compositional Abstraction-based Synthesis for Cascade Discrete-Time Control Systems." 2018.

[Tab09] Paulo Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[WGC07] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback control of dynamic bipedal robot locomotion*, volume 28. CRC press, 2007.

[ZPM12] Mahdi Zamani, Giordano Pola, Manuel Mazo, and Paulo Tabuada. "Symbolic models for nonlinear control systems without stability assumptions." *Automatic Control, IEEE Transactions on*, **57**(7):1804–1809, 2012.