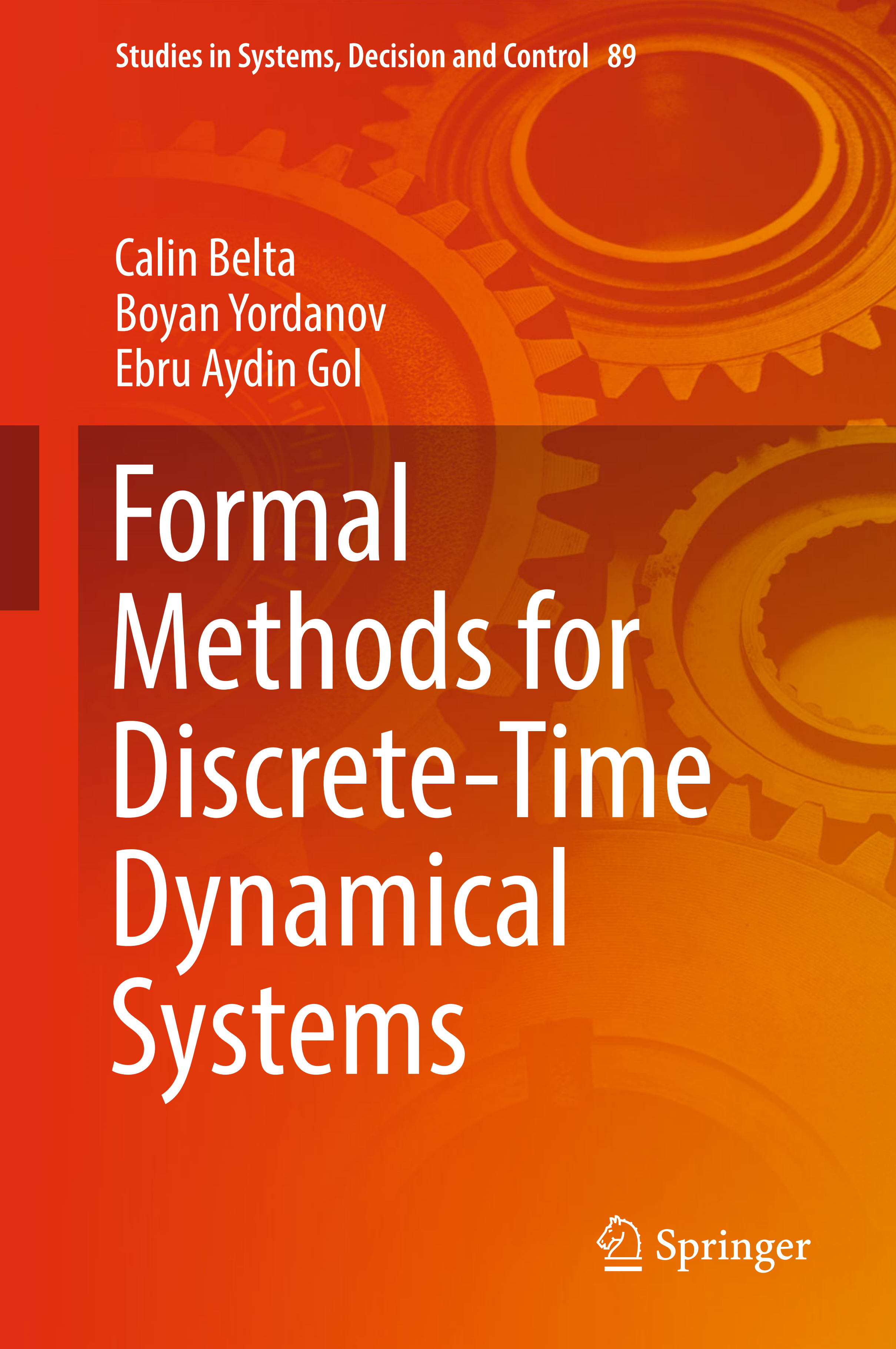
Studies in Systems, Decision and Control 89

Calin Belta
Boyan Yordanov
Ebru Aydin Gol

Formal Methods for Discrete-Time Dynamical Systems



Springer

Studies in Systems, Decision and Control

Volume 89

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

About this Series

The series “Studies in Systems, Decision and Control” (SSDC) covers both new developments and advances, as well as the state of the art, in the various areas of broadly perceived systems, decision making and control- quickly, up to date and with a high quality. The intent is to cover the theory, applications, and perspectives on the state of the art and future developments relevant to systems, decision making, control, complex processes and related areas, as embedded in the fields of engineering, computer science, physics, economics, social and life sciences, as well as the paradigms and methodologies behind them. The series contains monographs, textbooks, lecture notes and edited volumes in systems, decision making and control spanning the areas of Cyber-Physical Systems, Autonomous Systems, Sensor Networks, Control Systems, Energy Systems, Automotive Systems, Biological Systems, Vehicular Networking and Connected Vehicles, Aerospace Systems, Automation, Manufacturing, Smart Grids, Nonlinear Systems, Power Systems, Robotics, Social Systems, Economic Systems and other. Of particular value to both the contributors and the readership are the short publication timeframe and the world-wide distribution and exposure which enable both a wide and rapid dissemination of research output.

More information about this series at http://www.springer.com/series/13304

Calin Belta · Boyan Yordanov
Ebru Aydin Gol

Formal Methods for Discrete-Time Dynamical Systems

Springer

Calin Belta
Mechanical Engineering, Electrical and Computer Engineering, and Systems Engineering
Boston University
Boston, MA
USA

Ebru Aydin Gol
Department of Computer Engineering
Middle East Technical University
Cankaya, Ankara
Turkey

Boyan Yordanov
Biological Computation Group
Microsoft Research Ltd
Cambridge, Cambridgeshire
UK

ISSN 2198-4182 ISSN 2198-4190 (electronic)
Studies in Systems, Decision and Control
ISBN 978-3-319-50762-0 ISBN 978-3-319-50763-7 (eBook)
DOI 10.1007/978-3-319-50763-7

Library of Congress Control Number: 2016963168

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To our families

Foreword

It is my distinct pleasure to write a foreword for the wonderful research monograph Formal Methods for Discrete-Time Dynamical Systems written by Calin Belta, Boyan Yordanov, and Ebru Aydin Gol.

In the era of internet of things and cyber-physical systems, it is becoming evident that computation is increasingly interacting with safety-critical control systems. In the absence of software, control theory has provided many tools for the rigorous design of closed loop systems. In the absence of physical models, the formal methods community has provided powerful tools for verifying the safe operation of software (or hardware). As computation and control systems are tightly interconnected in modern complex systems, simply bringing tools from control theory and formal methods together does not suffice in understanding the software-controlled system. There are many new safety problems that are created when software interacts with control systems. Furthermore, a new science needs to be developed focusing on the rapprochement between formal methods and control.

Over the last two decades, there has been intense research activity on the interface between formal methods and control systems, under the broader research agenda of hybrid systems, embedded systems, and now cyber-physical systems. This is a very challenging area that has the ultimate goal of creating a modern systems science for systems that include both physical and computational elements. Researchers from formal methods have introduced notions of physicality into automata leading to so-called hybrid automata, which are systems that are heavy on the logic side and light on the physical side. The challenge in this approach is verifying the discrete side of the hybrid systems regardless of the simple control dynamics. On the other hand, researchers in control systems have introduced elements of switching logic in control systems, resulting in switched control systems, which are heavy on the physical side and light on the discrete side. The challenge in this approach is ensuring that the control system is stable regardless of the switching logic.

While both these approaches are very useful, only in the past decade we have seen truly hybrid approaches where formal method ideas are applied in complex control systems. This is the approach that is presented in this research monograph by the authors who have significantly contributed to the development of this new theory. While the fundamental system that is being considered in this monograph is that of a discrete-time linear system, the specification for the systems is no longer classical stability or controllability, but rather a specification that is formally described in a suitable temporal logic. This is very important for two reasons. First we depart from classical control requirements towards requirement that are closer to software requirements. And second, specifying control system requirements in temporal logic is formal, rigorous, and compositional, something that is not the case with classical requirements (such as rise-time, overshoot requirements).

In order to address the problem of analyzing or designing control systems with specifications expressed in temporal logic, one needs to be very well educated on both control systems and formal methods. The monograph does an excellent exposition of relevant topics from formal methods and control systems. The fundamental concept that semantically bridges the world of control systems and formal methods is the concept of simulation and bisimulation, where a discrete-time linear system is abstracted by a transition system (automaton), which is equivalent from the perspective of the temporal logic property being considered. The notions of simulation and bisimulation are fundamental concepts that formally relate systems across discrete and continuous domains, bridging control systems and software systems in a formal and rigorous manner.

I have known Calin Belta since his graduate student days here at Penn, and we have collaborated on many of these topics over the years. Over the past decade he and his students, two of them being co-authors, have been pushing the state of the art in the development of a science for formal methods for control systems. Leveraging much of their outstanding research, the authors are ideal for this research monograph. While they focus on discrete-time linear systems, the interested reader will easily see connections to continuous-time systems, nonlinear systems, or different logics.

The timing of the book is excellent. Over the past decade, the theory in this area has matured and stabilized to a set of concepts and the authors have captured some important ones. Readers from formal methods can read this book to get a brief introduction to control systems and see how formal methods ideas can be expanded to address these modern challenges. Readers from control theory will get a much needed introduction to specifying system behavior using temporal logics, and how to connect their models to discrete models through the notions of simulation and bisimulation. But I project the biggest impact will be with new graduate students interested in entering this area that has limited textbook or monograph options. This wonderful research monograph will serve as fundamental and foundational introduction to this emerging new field.

George J. Pappas
University of Pennsylvania
Philadelphia, USA

Preface

Motivation and Objectives

In control theory, complex models of physical processes, such as systems of differential or difference equations, are usually checked against simple specifications, such as stability and set invariance. In formal methods, rich specifications, such as languages and formulas of temporal logics, are checked against simple models of software programs and digital circuits, such as finite transition systems. With the development and integration of cyber-physical and safety-critical systems, there is an increasing need for computational tools for verification and control of complex systems from rich, temporal logic specifications. For example, in a persistent surveillance application, an unmanned aerial vehicle might be required to "take photos of areas A and B infinitely often while always avoiding unsafe areas C and D." In the emergent area of synthetic biology, the goal is to design small gene networks from specifications that are naturally given as temporal logic statements about the concentrations of species of interest, e.g., "if inducer u_1 is low and inducer u_2 is high, then protein y should eventually be expressed and remain in this state for all future times."

Central to the existing approaches for formal verification and control of infinite-state systems is the notion of abstraction. Roughly, an abstract model can be seen as a finite transition graph, whose states label equivalent sets of states of the original system, and whose transitions match the trajectories of the original system among the equivalence classes. Once constructed, such an abstraction can be used for verification (using off-the-shelf model checking tools) or control (using automata game techniques) in lieu of the original system.

The main objective of this book is to present formal verification and control algorithms for a class of discrete-time systems generically referred to as *linear*. Most of the results are formulated for piecewise linear (or affine) systems, which are described by a collection of linear (affine) dynamics associated to the regions of a polytopic partition of the state space. Such systems are quite general, as they have been shown to approximate nonlinear system with arbitrary accuracy. There also exist computational tools for identifying such systems (both the polytopes and the corresponding dynamics) from experimental data.

This book is based on the work of the authors, and is, as a result, biased and non-comprehensive. The specifications are restricted to formulas of Linear Temporal Logic (LTL) and fragments of LTL, even though other temporal logics have been used by other authors. While some of the results can be extended to continuous-time systems, the focus is on discrete-time systems only. We only cover deterministic and purely non-deterministic systems, even though existing results, including ours, show that extensions to stochastic systems and probabilistic temporal logics are possible. The equivalence notion that we use is classical bisimulation—extensions to approximate bisimulations and probabilistic bisimulations have been developed recently.

Intended Audience

This book is intended to a broad audience of scientists and engineers with interest in formal methods and controls. In particular, it is our hope that this book will help bridge the gap between the computer science and control theory communities. Computer scientists are shown that simulations and bisimulations, normally used to reduce the size of finite models of computer programs, can be used to abstract infinite-state systems. The book also provides a self-contained exposition of temporal logic control for finite non-deterministic systems, which is useful even for seasoned formal methods researchers. Control theorists are introduced to notions such as abstractions, temporal logics, formal verification, and formal synthesis, and are shown that such techniques can be used for classical systems such as discrete-time linear systems.

Book Outline and Usage

This book is self-contained. While some level of mathematical maturity is expected, no mathematical background in control or automata theory is necessary. Most of the formal definitions and algorithms are explained in plain language and illustrated with several examples. Most examples include explanatory illustrations.

The book is organized in three parts. Part I covers the types of systems and specifications used throughout the rest of the book. Specifically, it introduces (non-deterministic) transition systems, a formalism that can be used to model a large spectrum of dynamical systems. Simulation and bisimulations relations and corresponding abstractions for transitions systems are defined. The syntax and semantics of Linear Temporal Logic (LTL) and one of its fragments, called syntactically co-safe LTL (scLTL), are introduced and illustrated with several examples. Finite state automata, Büchi automata, and Rabin automata accepting languages satisfying LTL formulas are also defined.

Part II focuses on finite systems, i.e., transition systems with finitely many states, inputs, and observations. After reviewing the classical LTL model checking problem, we solve the problem of finding the largest set of states from which all trajectories of a system satisfy an LTL formula. We show that the control version of this problem can be mapped to a Büchi game, a Rabin game, or a graph reachability problem depending on the structure of the specification formula. We present ready to implement solutions to all these problems and include illustrative examples.

In Part III, which is the most involved part of the book, we bring together the concepts and techniques introduced in Parts I and II and present computational frameworks for verification and control of (infinite) discrete-time linear and piecewise affine systems from LTL specifications. We cover LTL verification problems for systems with fixed and uncertain parameters, parameter synthesis problems, and control synthesis problems. We also provide algorithms for the construction of finite bisimulations for some classes of discrete-time linear systems. Finally, we establish a connection between optimality and correctness by requiring a linear system to satisfy a temporal logic correctness requirement while optimizing a cost function.

This book can be read and used in two ways. First, by covering Parts I and II (excluding Sect. 1.2 from Chap. 1 in Part I), it can be used as a stand-alone introduction to verification and control for finite non-deterministic transition systems from LTL formulas. This can be used as a first mini-course on formal methods for engineers and computer scientists. It can also be useful for formal methods researchers who have expertise in verification only. Second, the whole book can be used as a graduate level course on formal methods for dynamical systems, with particular focus on discrete-time linear and piecewise affine systems. Most of the algorithms presented in this book were implemented as user-friendly software packages that can be downloaded from the first author's webpage or can be provided on request.

Related Books

The related books on formal methods for dynamical systems are [123, 5, 162, 144]: [123, 5] are comprehensive expositions of theory and practice of embedded and cyber-physical systems, together with corresponding verification and synthesis techniques; [162, 144] are research monographs on formal methods for hybrid systems, which combine continuous and discrete dynamics. The focus in [144] is on theorem proving. The closest related to this book is [162].

There are three main features that set this book apart from [123, 5, 162, 144]. First, we provide a complete and self-contained treatment of the formal synthesis problem from specifications given as LTL formulas. This can be, for example, combined with the partition-based abstraction method from [162] to implement a computational tool for LTL synthesis for a quite large class of dynamical systems. Second, we focus on particular types of dynamical systems (i.e., discrete-time piecewise affine systems) and exploit their geometry to efficiently construct abstractions. Third, we explore the connection between optimality and correctness in control.

Acknowledgements

It is a great pleasure to acknowledge George J. Pappas, Rajeev Alur, and Vijay Kumar (all from the University of Pennsylvania), who fostered the interest of the first author in this topic early in his career. Jana Tumova, Ivana Cerna, and Jiri Barnat from Masaryk University contributed to the results presented in Chap. 9. Mircea Lazar from the Technical University of Eindhoven was a collaborator for the work described in Chaps. 10–12. The authors are grateful to support from the National Science Foundation (NSF), the Air Force Office of Scientific Research (AFOSR), the Office of Naval Research (ONR), and the Army Research Office (ARO). The first author would particularly like to thank Helen Gill, Fariba Fahroo, and Marc Steinberg for their enthusiastic support over the past several years.

Finally, we would like to thank our numerous colleagues and friends who provided comments and suggestions on earlier versions of the manuscript. In particular, we would like to thank Ezio Bartocci, Sam Coogan, Mircea Lazar, Rupak Majumdar, Necmiye Ozay, Giordano Pola, Vasumathi Raman, Paulo Tabuada, and Jana Tumova.

Boston, MA, USA Calin Belta
Cambridge, UK Boyan Yordanov
Ankara, Turkey Ebru Aydin Gol

Contents

Part I Transition Systems, Automata, and Temporal Logics

Part II Analysis and Control of Finite Transition Systems

Notations

$\emptyset$	Empty set
$\|\Sigma\|$	Cardinality of a finite set Σ
2^{Σ}	Power set (set of all subsets) of a set Σ
$\Sigma_1 \times \Sigma_2$	Cartesian product of sets Σ_1 and Σ_2
$w_{\Sigma} = w_{\Sigma}(1)w_{\Sigma}(2)w_{\Sigma}(3)\ldots, w_{\Sigma}(i) \in \Sigma$	Word over set Σ
$(w(1)w(2)\ldots w(k))^{\omega}$	Infinitely many repetitions of sequence $w(1)w(2)\ldots w(k)$
Σ^*	Set of finite words over Σ
Σ^{ω}	Set of infinite words over Σ
Σ^+	Set of finite words over Σ excluding the empty word
$\mathbb{R}$	Set of real numbers
$\mathbb{Z}$	Set of integer numbers
$\mathbb{N}$	Set of natural numbers (including 0)
$\mathbb{N}_+$	Set of positive natural numbers
$\mathbb{R}^N$	Euclidean space of dimension N
$x_i, i = 1, 2, \ldots$	States in a finite set of states of a transition system or components of the state x of a dynamical system
$[a\ b]$	Row or column vector in $\mathbb{R}^2$ (exact meaning determined from the context)
$\top$ and $\bot$	Boolean constants True and False
$\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$	Boolean operators negation, conjunction, disjunction, implication, and equivalence
$\bigcirc$	"Next" temporal operator
U	"Until" temporal operator
$\diamondsuit$	"Eventually" ("Future") temporal operator

$\Box$	"Always" ("Globally") temporal operator
$\sim$	Observational equivalence relation
$\approx$	Observation preserving bisimulation
$\sim_{\phi}$	ϕ-equivalent relation
$\approx_{\phi}$	Equivalence relation that is both ϕ-equivalent and $\neg\phi$-equivalent
$cl(\mathbf{X})$	Closure of polytope $\mathbf{X}$
$int(\mathbf{X})$	Interior of polytope $\mathbf{X}$
$V(\mathbf{X})$	Set of vertices of polytope $\mathbf{X}$
$hull(\Sigma)$	Convex hull of set Σ
$\oplus$	Minkowski (set) sum
$\otimes$	Product between transition systems or between a transition system and an automaton
$A^{\top}$	Transpose of matrix A
$f(S)\,\text{and} f^{-1}(S)$	Image and pre-image of set S through function f
0, $\mathbf{0}$	(scalar) zero and zero vector
$\mathcal{O}(\cdot)$	Complexity class

Part I
Transition Systems, Automata, and Temporal Logics

Chapter 1
Transition Systems

In this book, we focus on transition systems as a modeling formalism for a wide range of processes. Although mathematically simple, transition systems are general enough to capture the behavior of systems defined over discrete (finite or infinite) or continuous (infinite) spaces. In subsequent chapters, this richness will allow us to use transition systems as a unifying framework for modeling both (infinite-state) discrete-time systems and their (finite-state) abstractions. In the following chapters we will also describe techniques for the analysis and control of finite and infinite transition systems, which are inspired by automata-theoretic model checking.

In this chapter, we define the syntax and semantics of transition systems, and provide several illustrative examples. In particular, we present different (deterministic, nondeterministic, finite, and infinite) transition system representations for discrete-time dynamical systems. We also introduce simulation and bisimulation relations, which are central for the construction of finite abstractions throughout the book.

1.1 Definitions and Examples

Definition 1.1 (*Transition system*) A transition system is a tuple $T = (X, \Sigma, \delta, O, o)$, where

- X is a (possibly infinite) set of states,
- Σ is a (possibly infinite) set of inputs (controls or actions),
- $\delta : X \times \Sigma \to 2^X$ is a transition function,
- O is a (possibly infinite) set of observations, and
- $o : X \to O$ is an observation map.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_1

A subset $X_r \subseteq X$ is called a *region* of T. A transition $\delta(x, \sigma) = X_r$ indicates that, while the system is in state x, it can make a transition to any state $x' \in X_r$ in region $X_r \subseteq X$ under input σ. We denote the set of inputs available at state $x \in X$ by

$$\Sigma^x = \{\sigma \in \Sigma \mid \delta(x, \sigma) \neq \emptyset\}. \tag{1.1}$$

A transition $\delta(x, \sigma)$ is *deterministic* if $|\delta(x, \sigma)| = 1$ and the transition system T is deterministic if for all states $x \in X$ and all inputs $\sigma \in \Sigma^x$, $\delta(x, \sigma)$ is deterministic. T is *non-blocking* if, for every state $x \in X$, $\Sigma^x \neq \emptyset$. Throughout this book, only non-blocking transition systems are considered. Transition system T is called *finite* if its sets of states X, inputs Σ, and observations O are all finite.

An *input word* of the system is defined as an infinite sequence $w_\Sigma = w_\Sigma(1)w_\Sigma(2)$ $w_\Sigma(3)\ldots \in \Sigma^\omega$. A *trajectory* or *run* of T produced by input word w_Σ and originating at state $x_1 \in X$ is an infinite sequence $w_X = w_X(1)w_X(2)w_X(3)\ldots$ with the property that $w_X(k) \in X$, $w_X(1) = x_1$, and $w_X(k+1) \in \delta(w_X(k), w_\Sigma(k))$, for all $k \geq 1$. We denote the set of all trajectories of a transition system T originating at x by $T(x)$. We use $T(X_r) = \cup_{x' \in X_r} T(x')$ to denote the set of all trajectories of T originating in region $X_r \subseteq X$. As a consequence, $T(X)$ will denote the set of all trajectories of T.

A run $w_X = w_X(1)w_X(2)w_X(3)\ldots$ defines an *output word* (which we will refer to simply as *word*) $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$, where $w_O(k) = o(w_X(k))$ for all $k \geq 1$. The set of all words generated by the set of all trajectories starting at $x \in X$ is called the *language* of T originating at x and is denoted by $\mathscr{L}_T(x)$. The language of T originating at a region $X_r \subseteq X$ is $\mathscr{L}_T(X_r) = \bigcup_{x' \in X_r} \mathscr{L}_T(x')$. The language of T is defined as $\mathscr{L}_T(X)$, which for simplicity is also denoted as $\mathscr{L}_T$. We often represent an infinite word as a finite *prefix* followed by an infinite *suffix* as shown in Example 1.1.

For an arbitrary region $X_r \subseteq X$ and set of inputs $\Sigma' \subseteq \Sigma$, we define the set of states $Post_T(X_r, \Sigma')$ that can be reached from X_r in one step by applying an input in Σ' (called *successors* of X_r under Σ') as

$$Post_T(X_r, \Sigma') = \{x \in X \mid \exists x' \in X_r,\ \exists \sigma \in \Sigma',\ x \in \delta(x', \sigma)\} \tag{1.2}$$

Similarly, the set of states that reach some $X_r \subseteq X$ in one step under the application of some input from $\Sigma' \subseteq \Sigma$ (called *predecessors* of X_r under Σ') can be defined as

$$Pre_T(X_r, \Sigma') = \{x \in X \mid \exists x' \in X_r,\ \exists \sigma \in \Sigma',\ x' \in \delta(x, \sigma)\} \tag{1.3}$$

For a deterministic T, each state x has a single successor under a given input σ, i.e. $Post_T(x, \sigma) = \delta(x, \sigma)$[1] is a singleton, but, in general, can have multiple predecessors, i.e., $Pre_T(x, \sigma)$ is a region of T. However, for a nondeterministic T it is possible that both $Post_T(x, \sigma)$ and $Pre_T(x, \sigma)$ are regions of T.

[1] Since the $Post$ operator was defined for a set of inputs, the correct notation here is $Post_T(x, \{\sigma\})$. For simplicity, and with a slight abuse of notation, we omit the set notation when only a singleton input is considered. The same observation applies to the Pre operator.

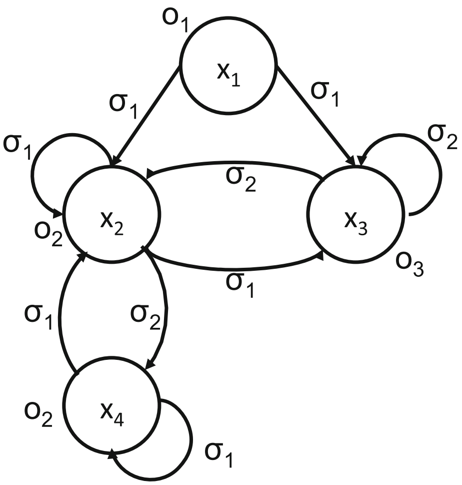


Fig. 1.1 Graphical representation of the transition system defined in Example 1.1. Each state is represented by a *circle* containing the state label. The observation of each state is shown close to its *circle* and transitions are represented by *arrows* between states. The input enabling a transition is shown on *top* of the corresponding *arrow*

Example 1.1 The finite, non-deterministic transition system $T = (X, \Sigma, \delta, O, o)$ shown in Fig. 1.1 is defined formally by

- $X = \{x_1, x_2, x_3, x_4\}$,
- $\Sigma = \{\sigma_1, \sigma_2\}$,
- $\delta(x_1, \sigma_1) = \{x_2, x_3\}$, $\delta(x_2, \sigma_1) = \{x_2, x_3\}$, $\delta(x_2, \sigma_2) = \{x_4\}$, $\delta(x_3, \sigma_2) = \{x_2, x_3\}$, $\delta(x_4, \sigma_1) = \{x_2, x_4\}$,
- $O = \{o_1, o_2, o_3\}$,
- $o(x_1) = o_1$, $o(x_2) = o(x_4) = o_2$, $o(x_3) = o_3$.

The set of inputs available at the states of the system are $\Sigma^{x_1} = \{\sigma_1\}$, $\Sigma^{x_2} = \{\sigma_1, \sigma_2\}$, $\Sigma^{x_3} = \{\sigma_2\}$, and $\Sigma^{x_4} = \{\sigma_1\}$. States x_1 and x_3 form a region $X_r = \{x_1, x_3\}$. Region X_r can be reached from states x_1 and x_2 under σ_1, and, therefore, $Pre_T(X_r, \sigma_1) = \{x_1, x_2\}$. Similarly, states x_2 and x_3 are reachable from region X_r under σ_1, and therefore $Post_T(X_r, \sigma_1) = \{x_2, x_3\}$.

Starting from x_1, under input word $w_\Sigma = \sigma_1\sigma_1\sigma_2(\sigma_1)^\omega$, one possible run of the system is $w_X = x_1x_2x_2(x_4)^\omega$. Sequences $\sigma_1\sigma_1\sigma_2$, $x_1x_2x_2$ are prefixes (repeated once) and σ_1, x_4 are suffixes (repeated infinitely many times) for input word and run, respectively. This run originates in region X_r and defines an infinite (output) word $w_O = o_1(o_2)^\omega \in \mathscr{L}_T(X_r)$. There is an infinite number of infinite words in the language $\mathscr{L}_T(X_r) = \{o_1(o_2)^\omega, o_1o_2(o_3)^\omega, o_1(o_3)^\omega, (o_3)^\omega, o_3o_3(o_2)^\omega, \ldots\}$.

A transition system $T = (X, \Sigma, \delta, O, o)$ is usually referred to as a *control transition system*. It is used as a model in control problems, where the goal is to generate a control strategy from a control specification, possibly given as a temporal logic statement over the observations of the system. This topic will be addressed in detail later in the book. For example, a control strategy enforcing the specification "if o_1 is satisfied initially, then eventually reach and remain in a region where either o_1 or o_2 are satisfied, or o_3 is satisfied" is derived in Chap. 5 for the transition system from Example 1.1. In analysis problems, we are usually interested in checking whether all the trajectories of a system satisfy such a specification, under arbitrary input choices. Since the input is irrelevant, the input set Σ is a singleton, where the only input enables all transitions available at all states. For simplicity of notation, this input set can be omitted completely, leading to a simpler (autonomous, uncontrolled) transition system $T = (X, \delta, O, o)$, where X, O, and o are as given in Definition 1.1 and the transition function assumes the simpler form $\delta : X \to 2^X$. In the uncontrolled case, the definitions of the successor and predecessor sets (Eqs. (1.2) and (1.3)) also assume simpler forms:

$$Post_T(X_r) = \{x' \in X \mid \exists x \in X_r,\ x' \in \delta(x)\} = \bigcup_{x \in X_r} \delta(x), \tag{1.4}$$

$$Pre_T(X_r) = \{x \in X \mid \exists x' \in X_r,\ x' \in \delta(x)\}, \tag{1.5}$$

where $X_r \subseteq X$ is a region. We will use both control transition systems and (autonomous) transition systems throughout the book. We will refer to both simply as *transition systems* and we will use the same symbols to denote their states, transitions, outputs, and output maps. The exact meaning of the transition function and of the $Post$ and Pre operators will be clear from the context. In particular, a transition system with no inputs is deterministic if it has at most one transition from each state.

Example 1.2 An example of a finite, non-deterministic transition system (without inputs) $T = (X, \delta, O, o)$ is given in Fig. 1.2. It is characterized by

- $X = \{x_1, x_2, x_3, x_4\}$,
- $\delta(x_1) = \{x_1, x_2\}$, $\delta(x_2) = \{x_4\}$, $\delta(x_4) = \{x_3\}$, $\delta(x_3) = \{x_3, x_1\}$,
- $O = \{o_1, o_2, o_3\}$,
- $o(x_1) = o(x_2) = o_1$, $o(x_3) = o_3$, $o(x_4) = o_2$.

For region $X_r = \{x_1, x_3\}$, we have $Pre_T(X_r) = \{x_1, x_3, x_4\}$ and $Post_T(X_r) = \{x_1, x_2, x_3\}$.

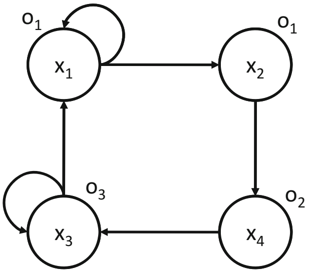


Fig. 1.2 Graphical representation of the transition system from Example 1.2. Unlike the control transition system from Fig. 1.1, there are no inputs labeling the transitions between the states

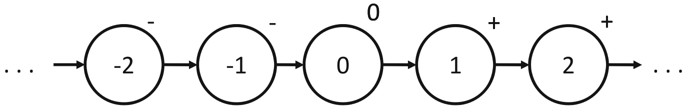


Fig. 1.3 Graphical representation of the transition system defined in Example 1.3

Example 1.3 The infinite, deterministic transition system $T = (X, \delta, O, o)$ shown in Fig. 1.3 is defined formally by

- $X = \mathbb{Z}$,
- $\delta(x) = x + 1,\ \forall x \in \mathbb{Z}$
- $O = \{-, 0, +\}$,
- $o(x) = \begin{cases} + & \text{if } x > 0 \\ - & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases}$

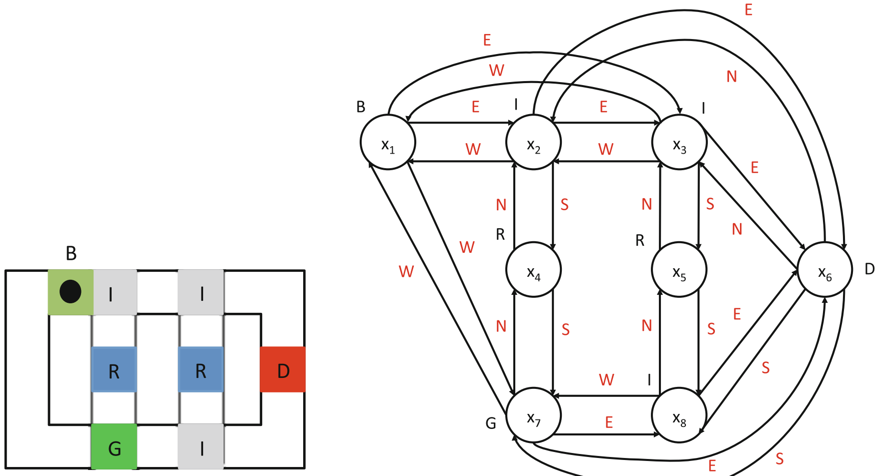
At each state, only the sign of the integer labeling the state is observed.

Example 1.4 Consider a messenger robot moving in the environment depicted in Fig. 1.4a. The robot (black disk) starts at the base B. When it detects a current region of interest (base B, data gather region G, recharge regions R, dangerous region D) or an intersection I, the robot is required to stop and choose the next immediate direction of motion (West, East, South, North). Then the robot turns towards the chosen direction and keeps following the road. We assume that the robot's actuators and sensors are unreliable, and as a result, the transition to a next intersection is not guaranteed. For example, while driving East from

B, the robot can end up at either of the two intersections to the right of B in Fig. 1.4a, i.e., it can fail to stop at the first intersection. The motion of the robot in this environment can be modeled as the nondeterministic transition system T shown in Fig. 1.4b, which is described by

- $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$;
- $\Sigma = \{W, E, S, N\}$;
- $O = \{B, G, R, D, I\}$; and
- δ and o as shown in Fig. 1.4b.

Example 1.5 A simple gene network is represented schematically in Fig. 1.5a. The network consist of three genes A, B and C expressed from separate promoters. When expressed, each gene produces a different protein (also denoted respectively by A, B and C). Protein C acts as a repressor and binds to the promoters of genes A and B. Therefore, whenever protein C is present, proteins A and B are not produced. Similarly, proteins A and B act as activators for the promoter of gene C, which can be expressed only if either protein A or B is present.



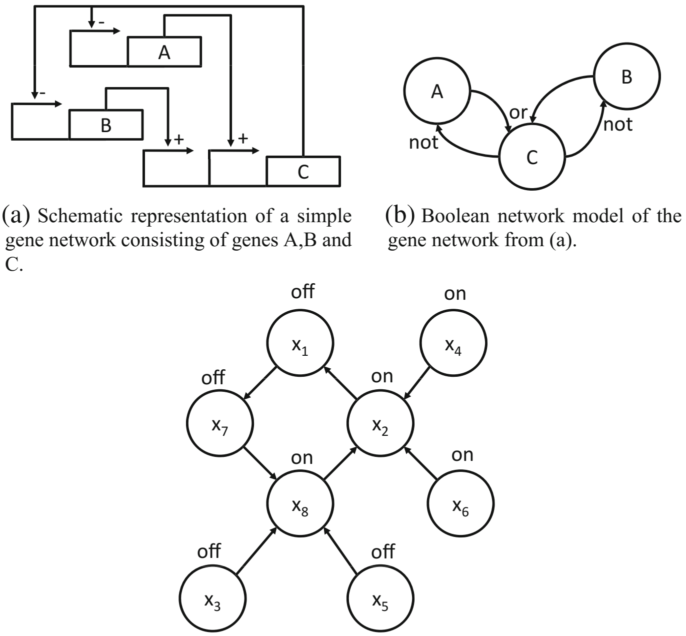
(a) A robot in an indoor environment

(b) The corresponding transition system

Fig. 1.4 A finite nondeterministic transition system representation for the motion of a robot in an indoor environment (see Example 1.4)

The regulatory interactions of this simple gene network can be captured using the Boolean network model represented schematically in Fig. 1.5b. The model consists of 3 Boolean variables signifying that each gene can be in one of two possible states (i.e., expressed (on) or not expressed (off)). At each step, the model evolves dynamically according to the following rules: the next state of variable A (denoted by A') is given by the Boolean formula $A' = \neg C$ and, similarly, $B' = \neg C$ and $C' = A \vee B$. At each discrete time step, all the variables are updated simultaneously.

The Boolean network model from Fig. 1.5b can be translated into the finite, deterministic transition system $T = (X, \delta, O, o)$ shown in Fig. 1.5c. The set of states of the system $X = \{x_1, x_2, \ldots, x_8\}$ captures all possible valuation of the Boolean variables from the model (i.e., $x_1 := (A = \bot, B = \bot, C = \bot)$, $x_2 := (A = \bot, B = \bot, C = \top)$, etc., where $\bot$ and $\top$ denotes the Boolean "false" and "true", respectively). Transitions between states of T are assigned using the dynamics of the Boolean network (i.e., $\delta(x_1) = x_7$ since at state x_1



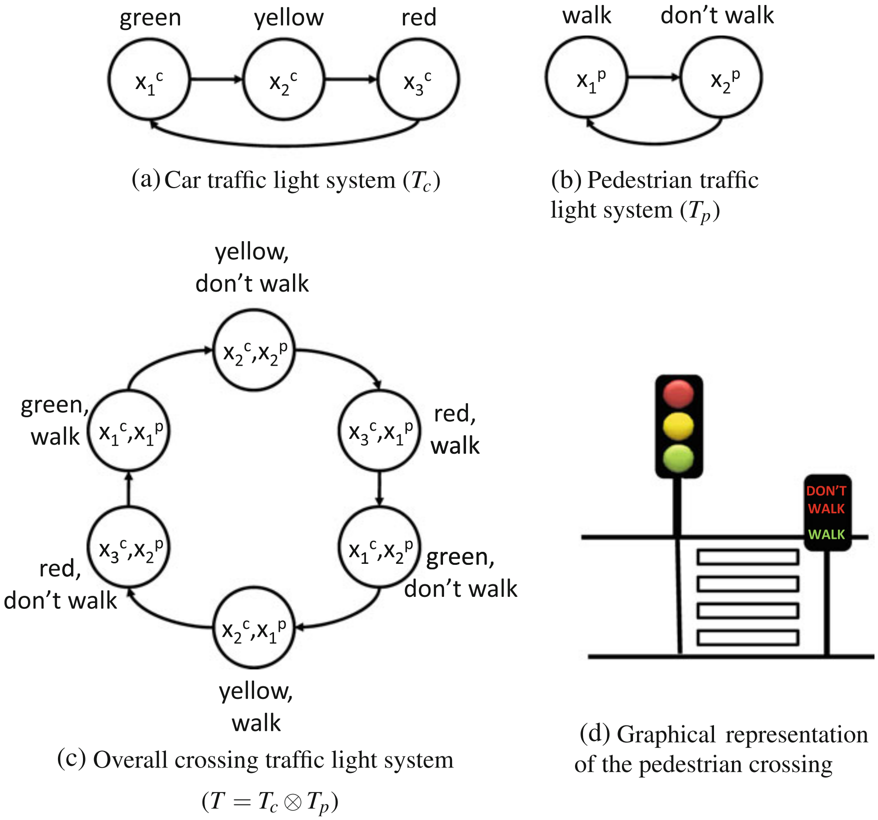
(a) Schematic representation of a simple gene network consisting of genes A,B and C.

(b) Boolean network model of the gene network from (a).

(c) Transition system representation of the Boolean network from (b) where only the state of geneC (on/off) is observed.

Fig. 1.5 Gene networks can be modeled as Boolean networks, which in turn can be described by finite, deterministic transition systems (see Example 1.5 for additional details)

we have ($A = \bot$, $B = \bot$, $C = \bot$) and therefore at the next step ($A' = \top$, $B' = \top$, $C' = \bot$), which defines state x_7). Assuming that we can only observe whether gene C is expressed or not, the set of observations can be given by $O = \{$"on", "off"$\}$ where the observation map of T is defined for all states $x \in X$ by $o(x) =$"on" if and only if $C = \top$ at state x and $o(x) =$"off" otherwise. Throughout this book, we use quotes around observations defined as strings (such as "on","off") but not when the observations are symbols (such as o_1, o_2, o_3 in Example 1.2 or $-, 0, +$ in Example 1.3).



(a) Car traffic light system (T_c)

(b) Pedestrian traffic light system (T_p)

(c) Overall crossing traffic light system ($T = T_c \otimes T_p$)

(d) Graphical representation of the pedestrian crossing

Fig. 1.6 A finite transition system representation of the pedestrian intersection traffic light described in Example 1.6

Example 1.6 Consider the model of a traffic light at a pedestrian crossing (Fig. 1.6) shown graphically in Fig. 1.6d.

The system consists of two components:

(i) a car traffic light with a "red", "yellow" and a "green" signal, and
(ii) a pedestrian light with a "walk" and "don't walk" signal.

Initially, the car and pedestrian traffic light components are modeled as separate transition systems T_c and T_p (shown respectively in Fig. 1.6a, b), where each signal is captured by an observation (i.e., O_c ={"red", "yellow", "green"} and O_p ={"walk", "don't walk"}). For both transition systems, each state has a unique observation (i.e., in each state, each traffic light is displaying a unique signal). To distinguish between states of T_c and T_p, we denote them by $X_c = \{x_1^c, x_2^c, x_3^c\}$ and $X_p = \{x_1^p, x_2^p\}$, respectively.

By assuming that the transitions of the two systems are perfectly synchronized (i.e., transitions are taken at the same time), the overall model of the crossing is obtained by constructing the product $T = T_c \otimes T_p$ (Fig. 1.6c) of the car and pedestrian traffic light components. Each state of the product T specifies the entire state of the crossing. The observations are obtained by collecting the individual observations and the transitions are obtained by matching the transitions of the individual systems. For example, in state (x_3^c, x_1^p) the car traffic light displays the "red" signal (i.e., transition system T_c is in state x_3^c) and the pedestrian traffic light displays the "walk" signal (i.e., transition system T_p is in state x_1^p).

Note that the pedestrian crossing is modeled as an autonomous, deterministic transition system. In other words, it is assumed that this system is not influenced by the environment or any external inputs (i.e., the system is autonomous) and, instead, it keeps changing the signals according to a predetermined program, always following the same sequence (i.e., the system is deterministic).

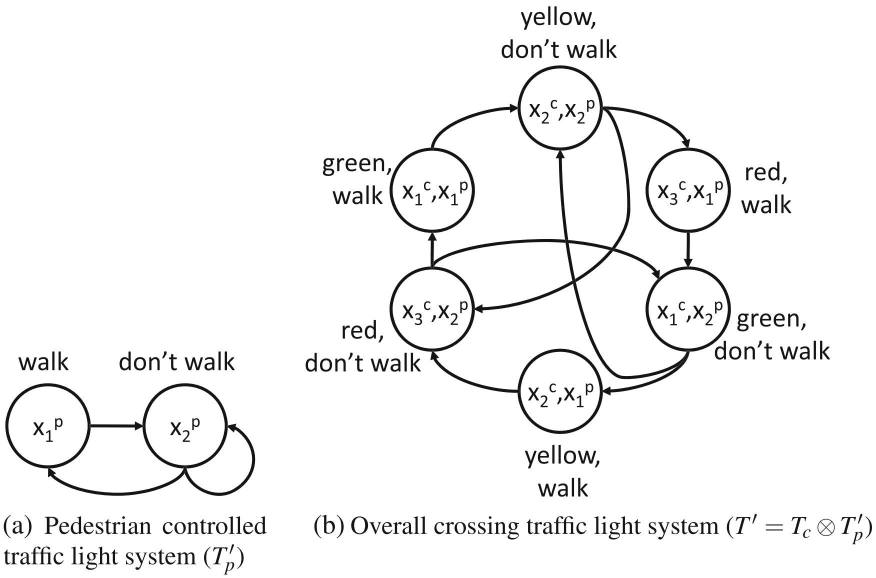
The traffic light model from this example includes a dangerous state (x_1^c, x_1^p) with the observation "green", "walk". Approaches for identifying such unsafe behaviors will be presented in Part II of this book (Examples 3.1 and 4.3).

Example 1.7 We further complicate Example 1.6 by considering a crossing where a pedestrian must push a button to request the "walk" signal. As in Example 1.6, we model the overall crossing as the composition of the car and pedestrian traffic light components, where the car traffic light component T_c remains unchanged as in Fig. 1.6a.

To capture the behavior of the pedestrian traffic light, we use the nondeterministic transition system T_p' shown in Fig. 1.7a. Whenever this system is in state x_2^p (showing the "don't walk" signal), a nondeterministic transition to either states x_1^p ("walk") or x_2^p ("don't walk") can be taken, depending if a pedestrian pushes the button. An explicit input is not considered in this case to represent whether the button is pressed. Instead, this action is considered as a nondeterministic event.

The pedestrian and car traffic light components are synchronized as in Example 1.6 by constructing the product $T' = T_c \otimes T_p'$ shown in Fig. 1.7b, which is nondeterministic since T_p' is nondeterministic. The product T' captures all possible behavior of the traffic light system — at any discrete time step, a "walk" request might or might not occur depending on the behavior of a pedestrian, which results in the nondeterministic transitions in T'.

Note that the dangerous state (x_1^c, x_1^p) with observation "green", "walk" from the initial traffic light model described in Example 1.6 is not eliminated by considering a "walk" request button. In fact, the set of states remains unchanged and only additional (nondeterministic) transitions are introduced.



(a) Pedestrian controlled traffic light system (T_p')

(b) Overall crossing traffic light system ($T' = T_c \otimes T_p'$)

Fig. 1.7 A finite transition system representation of the pedestrian intersection traffic light described in Example 1.7

1.2 Discrete-Time Dynamical Systems as Transition Systems

Consider the following discrete-time control system:

$$\mathscr{D}: \begin{array}{r} x(k+1) = f(x(k), u(k)),\ k = 0, 1, 2, \ldots, \\ y(k) = g(x(k)), \end{array} \tag{1.6}$$

where $x(k) \in \mathbb{R}^N$ is the state at time k, $u(k) \in U \subseteq \mathbb{R}^M$ is the control input at time k (U is the control constraint set), $f : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^N$ is a vector function describing the dynamics of the system, $y(k) \in \mathbb{R}^P$ is the output at time k, and $g : \mathbb{R}^N \to \mathbb{R}^P$ is the output function.

The discrete-time system from Eq. (1.6) can be easily represented as an infinite transition system $T_{\mathscr{D}}^{1,c} = (X, \Sigma, \delta, O, o)$, where

- the set of states $X = \mathbb{R}^N$,
- the set of inputs $\Sigma = U$,
- the transition function $\delta = f$,[2]
- the set of observations $O = \mathbb{R}^P$, and
- the observation map $o = g$.

The transition system $T_{\mathscr{D}}^{1,c}$ is called the one-step embedding of $\mathscr{D}$, since it captures all the transitions that $\mathscr{D}$ can take in one discrete-time step. It is also a timed and controlled embedding, because it preserves the time and control information from the original system. This is the most natural embedding of $\mathscr{D}$, since $T_{\mathscr{D}}^{1,c}$ and $\mathscr{D}$ include exactly the same amount of information. It is easy to see that $T_{\mathscr{D}}^{1,c}$ is non-blocking and deterministic.

Other types of embeddings can also be defined for $\mathscr{D}$. For example, in an analysis problem, we might be interested in capturing all the possible runs of $\mathscr{D}$, and we are not interested in recording the controls. The one-step, control-abstract embedding of $\mathscr{D}$ is defined as a transition system $T_{\mathscr{D}}^{1} = (X, \delta, O, o)$ where X, O, and o are defined exactly as above. The transition function δ is defined as $x' = \delta(x)$ if and only if there exists $u \in U$ such that $x' = f(x, u)$.

Transition system $T_{\mathscr{D}}^{1}$ is a particular case of another timed, control-abstract embedding of $\mathscr{D}$. This embedding, which we will denote by $T_{\mathscr{D}}^{\mathbb{N}} = (X, \Sigma, \delta, O, o)$, is a transition system whose X, O, and o are inherited from $T_{\mathscr{D}}^{1}$ (and $T_{\mathscr{D}}^{1,c}$). Its set of inputs $\Sigma = \mathbb{N}$ and its transition function δ is defined as $x' = \delta(x, k)$ if and only if there exist $u(0), u(1), \ldots, u(k-1) \in U$ driving system (1.6) from $x(0) = x$ to $x' = x(k)$. Finally, a time-abstract, control abstract embedding of $\mathscr{D}$, denoted simply by $T_{\mathscr{D}} = (X, \delta, O, o)$, is a transition system that can be defined by taking the transition function δ as $x' = \delta(x)$ if and only if there exist $k \in \mathbb{N}$ and $u(0), u(1), \ldots, u(k-1) \in U$ driving system (1.6) from $x(0) = x$ to $x' = x(k)$.

[2] As before, without the risk of confusion, and to keep the notation simple, we slightly abuse the notation when we refer to singletons and sets made of just one element. Formally, δ outputs a set, while f outputs a singleton. In this case, the set produced by δ has just one element.

There are several situations in which only a finite number of properties that the states can satisfy are of interest. Such properties, or predicates, can be given as a set of functions over the state space and can be accommodated by our definition of an embedding transition system. For example, assume that we have a set of scalar polynomials $\{p_1, p_2, \dots, p_L\}$ defined in $\mathbb{R}^n$. Each p_i defines a partition of $\mathbb{R}^n$ into three regions, which correspond to $p_i < 0$, $p_i = 0$, and $p_i > 0$, and which can be labeled simply as n, z, and p, respectively. If all L polynomials are considered, the state space is partitioned into 3^L regions that are labeled as $(p_1^e, p_2^e, \dots, p_L^e)$, with $p_i^e \in \{n, z, p\}, i = 1, 2, \dots, L$. If we are interested in how the trajectories of $\mathscr{D}$ move among these regions, the set of observations can be defined as

$$O = \{(p_1^e, p_2^e, \dots, p_L^e) \mid p_i^e \in \{n, z, p\},\ i = 1, 2, \dots, L\}. \tag{1.7}$$

The observation map is given by:

$$o(x) = (p_1^e, p_2^e, \dots, p_L^e), \tag{1.8}$$

where $p_i^e = n$ if $p_i(x) < 0$, $p_i^e = z$ if $p_i(x) = 0$, and $p_i^e = p$ if $p_i(x) > 0$, for all $i = 1, 2, \dots, L$.

Alternatively, polynomials $p_i, i = 1, 2, \dots, L$ can be used to define a set of predicates, e.g.,

$$\pi_i:\ p_i < 0,\ i = 1, 2, \dots, L. \tag{1.9}$$

The set of observations is then the power set of the set of predicates

$$O = 2^{\{\pi_i,\, i=1,2,\dots,L\}} \tag{1.10}$$

and the observation map $o : X \to O$ is defined as

$$o(x) = \{\pi_i \mid p_i(x) < 0\} \tag{1.11}$$

Finally, in several practical applications, the observations are defined as labels $\{P_1, P_2, \dots, P_L\}$ for a set of possibly overlapping “regions” that cover the state space (i.e., $\cup_{i=1,\dots,L} P_i = X$). In this case, with the observation set $O = 2^{\{P_1, P_2, \dots, P_L\}}$, the observation map $o : X \to O$ is defined as

$$o(x) = \{P_i \mid x \in P_i\} \tag{1.12}$$

Example 1.8 Consider a planar, discrete-time affine control system described by

$$\mathscr{D}_{lin}: \begin{array}{r}x(k+1) = Ax(k) + Bu(k) + b,\ k = 0, 1, 2, \dots,\\ y(k) = Cx(k),\end{array} \tag{1.13}$$

where

$$A = \begin{bmatrix} 0.95 & -0.5 \\ 0.5 & 0.65 \end{bmatrix}, \ B = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \ b = \begin{bmatrix} 0.5 \\ -1.3 \end{bmatrix}, \ C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \qquad (1.14)$$

The one-step timed controlled embedding $T^{1,c}_{\mathscr{D}_{lin}} = (X, \Sigma, \delta, O, o)$ is defined by

- the set of states $X = \mathbb{R}^2$,
- the set of inputs $\Sigma = \mathbb{R}$,
- the transition function $\delta(x, u) = Ax + Bu + b$,
- the set of observations $O = \mathbb{R}$, and
- the observation map $o(x) = Cx$.

The input word 0, 0, 0, ... determines the (autonomous) run

$$\begin{bmatrix} 8.0 \\ 5.0 \end{bmatrix}, \begin{bmatrix} 5.600 \\ 5.950 \end{bmatrix}, \begin{bmatrix} 2.8450 \\ 5.3675 \end{bmatrix}, \begin{bmatrix} 0.5190 \\ 3.6114 \end{bmatrix}, \begin{bmatrix} -0.8126 \\ 1.3069 \end{bmatrix}, \begin{bmatrix} -0.9255 \\ -0.8568 \end{bmatrix}, \begin{bmatrix} 0.0492 \\ -2.3197 \end{bmatrix}, \ldots$$

shown in blue in Fig. 1.8 (left) and the output word

$$8.0000, 5.6000, 2.8450, 0.5190, -0.8126, -0.9255, 0.0492, \ldots$$

The input word $u(0)u(1), \ldots, u(100)$ with $u(k) = 0.04k$ for $k \leq 50$ and $u(k) = -0.04k + 4$ for $50 < k \leq 100$, shown in Fig. 1.8 (right), produces the run

$$\begin{bmatrix} 8.0 \\ 5.0 \end{bmatrix}, \begin{bmatrix} 5.600 \\ 5.950 \end{bmatrix}, \begin{bmatrix} 2.8042 \\ 5.4491 \end{bmatrix}, \begin{bmatrix} 0.3578 \\ 3.8073 \end{bmatrix}, \begin{bmatrix} -1.1862 \\ 1.5985 \end{bmatrix}, \begin{bmatrix} -1.5894 \\ -0.5275 \end{bmatrix}, \begin{bmatrix} -0.9503 \\ -2.0294 \end{bmatrix}, \ldots$$

shown in red in Fig. 1.8 (left) and the output word

$$8.0000, 5.6000, 2.8042, 0.3578, -1.1862, -1.5894, -0.9503, \ldots$$

Assume now that we are only interested in how the runs of the system behave with respect to a set of affine functions, e.g.,

$$\begin{aligned} p_1(x) &= [1 \ -1]^\top x + 1.8, \\ p_2(x) &= [1 \ 1]^\top x - 6, \\ p_3(x) &= [0.6 \ -1]^\top x - 3.5, \\ p_4(x) &= [0.5 \ 1]^\top x + 1.7. \end{aligned}$$

If the approach described in Eqs. (1.7) and (1.8) is used, then the word generated by the blue run from the left of Fig. 1.8 is

$$(p, p, n, p),\ (p, p, n, p),\ (n, p, n, p),\ (n, n, n, p),\ (n, n, n, p),\ (p, n, n, p),\ (p, n, n, n),\ (p, n, p, n), \ldots. \tag{1.15}$$

The word generated by the red run coincides with the above word in the first 7 entries; the 8th entry is replaced by (p, n, n, n).

If the properties of interest are formulated in terms of a set of predicates, as in Eqs. (1.9), (1.10), and (1.11), then the word of the blue run is

$$\{\pi_3\},\ \{\pi_3\},\ \{\pi_1, \pi_3\},\ \{\pi_1, \pi_2, \pi_3\},\ \{\pi_1, \pi_2, \pi_3\},\ \{\pi_2, \pi_3\},\ \{\pi_2, \pi_3, \pi_4\},\ \{\pi_2, \pi_4\}, \ldots. \tag{1.16}$$

In the red run, the 8th entry is replaced by $\{\pi_2, \pi_3, \pi_4\}$.

1.3 Simulation and Bisimulation

A number of analysis and control techniques, such as the ones presented in Part II, have been developed to handle only finite transition systems (e.g. when an explicit representation of all system states is required). In addition, these methods become computationally challenging as the size of the state set of the system increases, which limits the applicability of these methods due to the infamous "curse of dimensionality". In this section, we introduce *finite abstractions*, which can be used to reduce the size of a finite system or map an infinite system to a finite one for the purpose of analysis and control.

Intuitively, an abstraction of a transition system T preserves some of its details required for analysis and control but ignores aspects that do not influence the results.

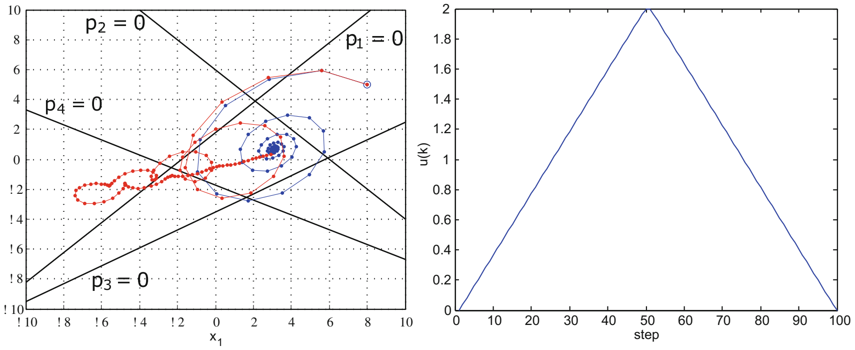


Fig. 1.8 Sample runs of the discrete-time system from Eqs. (1.13), (1.14) are shown on the *left* the *blue run* corresponds to the autonomous system ($u(k) = 0$, $k = 0, 1, 2, \ldots$), while the *red run* is produced by the input shown on the *right*. The two components of the state vector x are denoted by x_1 and x_2

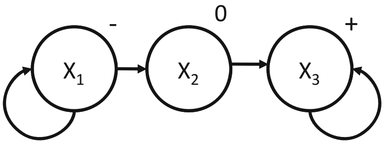


Fig. 1.9 An abstraction of the infinite transition system T with a set of states $X = \mathbb{Z}$ from Example 1.3 might preserve only the observation of a state (the sign of a number) but ignore the other details (the exact value). All states $x \in X$ with the property that $o(x) =$ "$-$" ($o(x) =$ "$+$") are equivalent and are grouped in an equivalence class X_1 (X_3)

More specifically, a state of the abstract model represents a large or infinite set of states in the original, *concrete* model that are somehow equivalent (e.g. all equivalent states might have the same observation). An abstraction could be equivalent to the concrete model with respect to the satisfaction of all specifications. Alternatively, it could provide an approximation, guaranteeing that satisfaction of a specification in the abstract model implies satisfaction in the original system. Equivalent abstractions are based on the notion of *bisimulation*, while approximate abstractions are constructed using *simulation relations*. In Sect. 4.5 we will also explore the idea of constructing abstractions which are equivalent only with respect to a given specification, and are therefore coarser than bisimulation. In all these cases, analysis or control of the large or infinite system can then be performed instead on its finite abstract model (see Fig. 1.9).

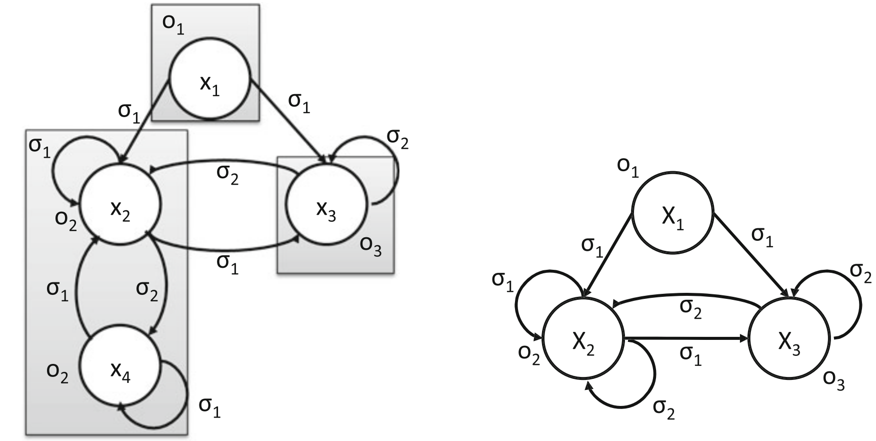
The observation map o of a transition system $T = (X, \Sigma, \delta, O, o)$ induces an equivalence relation $\sim \subseteq X \times X$ over the set of states X of T.

Definition 1.2 (*Observational equivalence*) States $x_1, x_2 \in X$ are observationally equivalent (written as $x_1 \sim x_2$) if and only if $o(x_1) = o(x_2)$.

Definition 1.3 (*Quotient transition system*) The observational equivalence relation $\sim$ naturally induces a *quotient transition system* $T/_\sim = (X/_\sim, \Sigma, \delta_\sim, O, o_\sim)$, where

- the set of states $X/_\sim$ is the quotient space (i.e., the set of all equivalence classes),
- the set of inputs Σ is inherited from the original system,
- the transition relation $\delta_\sim$ is defined as follows: for states $X_i, X_j \in X/_\sim$ and input $\sigma \in \Sigma$, we include transition $X_j \in \delta_\sim(X_i, \sigma)$ if and only if there exist states x_1 and x_2 of T in equivalence classes X_i and X_j, respectively, such that x_2 is reachable from x_1 in one step under input σ (i.e., $x_2 \in \delta(x_1, \sigma)$),
- the set of observations O is inherited from T, and
- the observation $o_\sim(X)$ of a state $X_i \in X/_\sim$ is given by $o_\sim(X_i) = o(x)$ for all states x from equivalence class X_i.

Given an equivalence class $X_i \in X/_\sim$, we denote the set of all equivalent states of T in that class by $con(X_i) \subseteq X$, where con stands for *concretization map*. For a state X_i of $T/_\sim$ the set $con(X_i)$ is, in general, a region of T and if $\mathbb{X} \subseteq X/_\sim$ is a region of $T/_\sim$, then $con(\mathbb{X}) = \bigcup_{X_i \in \mathbb{X}} con(X_i)$ is a region of T. The observation map $o_\sim$ of $T/_\sim$ is well defined, since all states $x \in con(X_i)$ from an equivalence class $X_i \in X/_\sim$ have the same observation (i.e., $\forall x \in con(X_i), o(x) = o_\sim(X_i)$).



(a) A finite, nondeterministic control transition system T.

(b) The quotient $T/_\sim$ of control transition system T.

Fig. 1.10 The quotient $T/_\sim$ (**b**) of system T (**a**) under the observational equivalence relation $\sim$. States that are equivalent in T (i.e., states sharing the same observation) are highlighted (see Example 1.9 for additional details)

Given states $X_i, X_j \in X/_\sim$, we can assign transitions in $T/_\sim$ through computation of the successor states (Eq. (1.2)) of each equivalence class, i.e.,

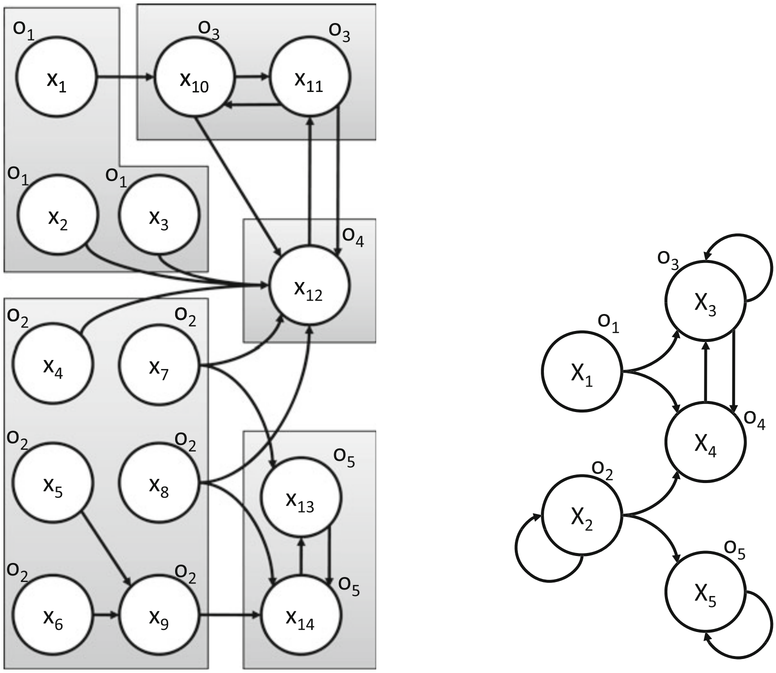
$$X_j \in \delta_\sim(X_i, \sigma) \text{ if and only if } Post_T(con(X_i), \sigma) \cap con(X_j) \neq \emptyset. \tag{1.17}$$

Equivalently, we can assign transitions in $T/_\sim$ by computing the set of predecessors (Eq. (1.3)) of each class

$$X_j \in \delta_\sim(X_i, \sigma) \text{ if and only if } con(X_i) \cap Pre_T(con(X_j), \sigma) \neq \emptyset. \tag{1.18}$$

Example 1.9 Consider the transition system from Fig. 1.1 shown again for convenience in Fig. 1.10a. The states with the same observations are equivalent. The corresponding equivalence classes and highlighted in Fig. 1.10a. The set of states of the quotient $T/_\sim$, shown in Fig. 1.10b, is $X/_\sim = \{X_1, X_2, X_3\}$. The set of all states of T in an equivalence class X_i is given by the concretization map $con()$: $con(X_1) = \{x_1\}$, $con(X_2) = \{x_2, x_4\}$, and $con(X_3) = \{x_3\}$. The observations of $T/_\sim$ are inherited from T. The transitions in $\delta_\sim$ are assigned as stated in Definition 1.3.

The above definitions can be immediately adapted for uncontrolled transition systems by considering the particular case when $|\Sigma| = 1$ followed by the deletion of the control symbol from the transition function.



(a) A finite, nondeterministic transition system T.

(b) The quotient $T/_{\sim}$ of transition system T.

Fig. 1.11 The quotient $T/_{\sim}$ (**b**) of system T (**a**) under the observational equivalence relation $\sim$. States that are equivalent in T (i.e., states sharing the same observation) are highlighted (see Example 1.10 for additional details)

Example 1.10 We construct the finite quotient $T/_{\sim}$ under the observational equivalence relation $\sim$ for transition system T given in Fig. 1.11a. Each state of T from the set $X = \{x_1, \ldots, x_{14}\}$ has a single observation from the set $O = \{o_1, \ldots, o_5\}$. States that have the same observation are equivalent and are highlighted in Fig. 1.11a, revealing the equivalence classes of the system. The quotient $T/_{\sim}$ has a set of states $X/_{\sim} = \{X_1, \ldots, X_5\}$ where, for $i = 1, \ldots, 5$, each state X_i represents the set of equivalent states with an observation o_i and, as a result, the observation map $o_{\sim}$ is clearly defined (i.e., $o_{\sim}(X_i) = o_i$). The set of all states of T in an equivalence class X_i is given by the concretization map $con()$: $con(X_1) = \{x_1, x_2, x_3\}$, $con(X_2) = \{x_4, \ldots, x_9\}$, $con(X_3) = \{x_{10}, x_{11}\}$, $con(X_4) = \{x_{12}\}$ and $con(X_5) = \{x_{13}, x_{14}\}$. Finally, transitions in $\delta_{\sim}$ are assigned as stated in Definition 1.3, resulting in the finite quotient $T/_{\sim}$ shown in Fig. 1.11b.

From Definition 1.3, it follows that for all states (equivalence classes) $X_i \in X/_\sim$ of $T/_\sim$, we have

$$\mathscr{L}_T(con(X_i)) \subseteq \mathscr{L}_{T/_\sim}(X_i). \tag{1.19}$$

In other words, the quotient control transition system $T/_\sim$ can produce any word w_O that can be produced by the original, concrete transition system T. However, in general there exists words in $\mathscr{L}_{T/_\sim}(X_i)$ that are *spurious* and do not represent valid behavior of T (i.e., they are not part of $\mathscr{L}_T(con(X_i))$).

Since any behavior of T can be reproduced by $T/_\sim$, we say that $T/_\sim$ *simulates* T. As we will discuss later in the book, this guarantees that, if a linear temporal logic formula is satisfied at some state X_i of $T/_\sim$, then the formula will be satisfied at all the states of T contained in $con(X_i)$.

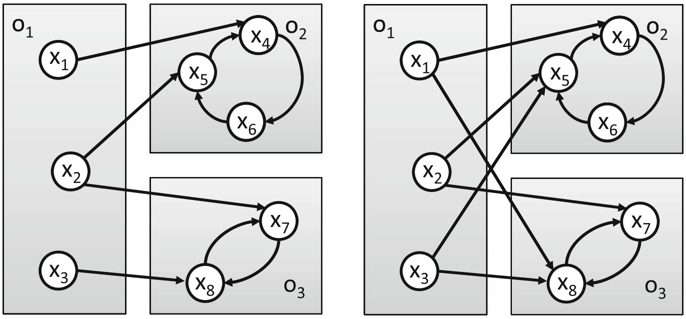
Definition 1.4 (*Bisimulation*) The equivalence relation $\sim$ induced by the observation map o is a bisimulation of a transition system $T = (X, \Sigma, \delta, O, o)$ if, for all states $x_1, x_2 \in X$, and all inputs $\sigma \in \Sigma$, if $x_1 \sim x_2$ and $x_1' \in \delta(x_1, \sigma)$, then there exist $x_2' \in X$ such that $x_2' \in \delta(x_2, \sigma)$ and $x_1' \sim x_2'$.

If $\sim$ is a bisimulation, then the quotient transition system $T/_\sim$ is called a *bisimulation quotient* of T, and the transition systems T and $T/_\sim$ are called *bisimilar*. To explicitly distinguish between a simulation and a bisimulation, we sometimes denote the latter by $\approx$ and use $T/_\approx$ to denote a bisimulation quotient. In other words, the quotient $T/_\approx$ is the quotient $T/_\sim$ when Definition 1.4 is satisfied.

Simulations and bisimulations relations are generally defined between transition systems sharing the same sets of observations, through equivalence relations between their states. In this book, we restrict our attention to equivalence relations defined over the states of a transition system, and the simulation/bisimulation relations are between the original system and its quotient. These notions are usually called observational simulation/bisimulation but throughout the rest of this book we will simply denote them as simulation/bisimulation.

For example, in Fig. 1.10b, the quotient $T/_\sim$ is not a bisimulation of the transition system T shown in Fig. 1.10a. Note that x_2, which is in the equivalence class X_2 has a transition to some state in the equivalence class X_3 under input σ_1. However, x_4, which is also in the equivalence class X_2 does not have a transition to some state in the equivalence class X_3 under σ_1. Examples of transition systems with no inputs for which the observational equivalence relation $\sim$ is not or is a bisimulation are given in Fig. 1.12a, b, respectively.

Definition 1.4 establishes bisimulation as a property of the quotient $T/_\sim$, when transitions originating at equivalent states in T satisfy certain conditions. In the following, for the particular case of transition systems with no inputs, we consider alternative conditions guaranteeing that the quotient $T/_\sim$ is a bisimulation quotient, which are easier to test computationally.



(a) $\sim$ is not a bisimulation.

(b) $\sim$ is a bisimulation.

Fig. 1.12 The observational equivalence relation $\sim$ (shown as *shaded rectangles*) is a bisimulation for the transition system from (**b**) but not for the one from (**a**) (for both systems only the observations for the entire equivalence classes are given)

Proposition 1.1 *For a transition system* $T = (X, \delta, O, o)$*, the equivalence relation* $\sim$ *is a bisimulation if the quotient* $T/_\sim$ *is deterministic.*

Proof Assume by contradiction that $\sim$ is not a bisimulation. Then, there exist $X_i, X_j \in X/_\sim, x_1, x_2 \in con(X_i)$, and $x_1' \in con(X_j)$ such that $x_1 \to x_1'$ but there does not exist $x_2' \in con(X_j)$ such that $x_2 \to x_2'$. However, since T is nonblocking, there exists $x_2'' \in X$ and $X_k \in X/_\sim$, $X_k \neq X_j$ such that $x_2'' \in \delta(x_2)$, where $x_2'' \in con(X_k)$. In the quotient $T_\sim$, this induces transitions $X_j \in \delta_\sim(X_i)$ and $X_k \in \delta_\sim(X_i)$, which implies that $T/_\sim$ is nondeterministic and contradicts the hypothesis. ■

Proposition 1.1 offers a computationally attractive sufficient condition for bisimulation for transition systems with no inputs, where only the number of outgoing transitions from each state in the quotient is counted. For deterministic transition systems, this result becomes stronger:

Proposition 1.2 *An equivalence relation* $\sim$ *defined on a deterministic transition system* $T = (X, \delta, O, o)$ *is a bisimulation if and only if the quotient* $T/_\sim$ *is deterministic.*

Proof From Proposition 1.1 it follows that if the quotient is deterministic then the equivalence relation is a bisimulation. Assume by contradiction that $T/_\sim$ is not deterministic. Then, there exist $X_i, X_j, X_k \in X/_\sim$ such that $X_j \in \delta_\sim(X_i)$ and $X_k \in \delta_\sim(X_i)$. However, since $\sim$ is a bisimulation, there exists $x_i, x_j, x_k \in X$, $x_i \in con(X_i)$, $x_j \in con(X_j)$, $x_k \in con(X_k)$ such that $x_j \in \delta(x_i)$ and $x_k \in \delta(x_i)$, which implies that T is non-deterministic and contradicts the hypothesis. ■

Relevant to model checking and analysis problems that we will define later in the book, as an immediate consequence of bisimulation, we can guarantee the language equivalence between the quotient $T/_\approx$ and the concrete system T. In other words, for all states $X_i \in X/_\approx$, it holds that

$$\mathscr{L}_T(con(X_i)) = \mathscr{L}_{T/\approx}(X_i). \tag{1.20}$$

In Definition 1.4, we gave conditions on the transitions originating at equivalent states in system T, required for the observational equivalence relation $\sim$ to be a bisimulation. Following from Definition 1.4, a computationally attractive characterization of bisimulation can be given by considering the set of predecessors (Eq. (1.5)) of each equivalence class.

Theorem 1.1 (Bisimulation characterization) *The equivalence relation $\sim$ is a bisimulation for a transition system $T = (X, \Sigma, \delta, O, o)$ if and only if for all equivalence classes $X_i \in X/_\sim$ and for all inputs $\sigma \in \Sigma$, $Pre_T(con(X_i), \sigma)$ is either empty or a finite union of equivalence classes. Equivalently, the bisimulation property from Definition 1.4 is violated at state $X_i \in X/_\sim$ if there exist an input $\sigma \in \Sigma$ and a state $X_j \in X/_\sim$, such that*

$$\emptyset \subset con(X_i) \cap Pre_T(con(X_j), \sigma) \subset con(X_i). \tag{1.21}$$

As a particular case, for a transition system $T = (X, \delta, O, o)$ with no inputs, the bisimulation characterization takes the form of Eq. 1.21 with σ removed.

Example 1.11 Consider the quotient $T/_\sim$ (Fig. 1.13) of transition system T from Fig. 1.12a. Using the characterization from Theorem 1.1, we can verify that the bisimulation property (Definition 1.4) is violated by equivalence class $X_1 \in X/_\sim$. The set $Pre(con(X_3)) = \{x_2, x_3\}$ is not empty and has a nonempty intersection with $con(X_1) = \{x_1, x_2, x_3\}$ but there exist a state $x_1 \in con(X_1)$ such that $x_1 \notin Pre((con(X_3))$. Equivalently, $Pre((con(X_3))$ is not a finite union of equivalence classes but is, in fact, a subset of an equivalence class. Since the characterization from Theorem 1.1 is violated at state X_1 (i.e., Eq. (1.21) is satisfied at that state), the quotient $X/_\sim$ is not bisimilar with T.

Algorithm 1 $\approx$=BISIMULATION(T): Construct the coarsest observation-preserving bisimulation quotient $\approx$ of $T = (X, \Sigma, \delta, O, o)$

```
Initialize $\sim_r := \sim$
while there exist equivalence classes $X_i, X_j \in X/_{\sim_r}$ and $\sigma \in \Sigma$ such that
        $\emptyset \subset con(X_i) \cap Pre_T(con(X_j), \sigma) \subset con(X_i)$ do
    Construct equivalence class $X_1$ such that $con(X_1) := con(X_i) \cap Pre_T(con(X_j), \sigma)$
    Construct equivalence class $X_2$ such that $con(X_2) := con(X_i) \setminus Pre_T(con(X_j), \sigma)$
    $X/_{\sim_r} := X/_{\sim_r} \setminus \{X_i\} \bigcup \{X_1, X_2\}$
end while
return $\sim_r$ ($\sim_r = \approx$)
```

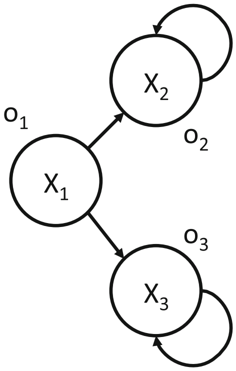


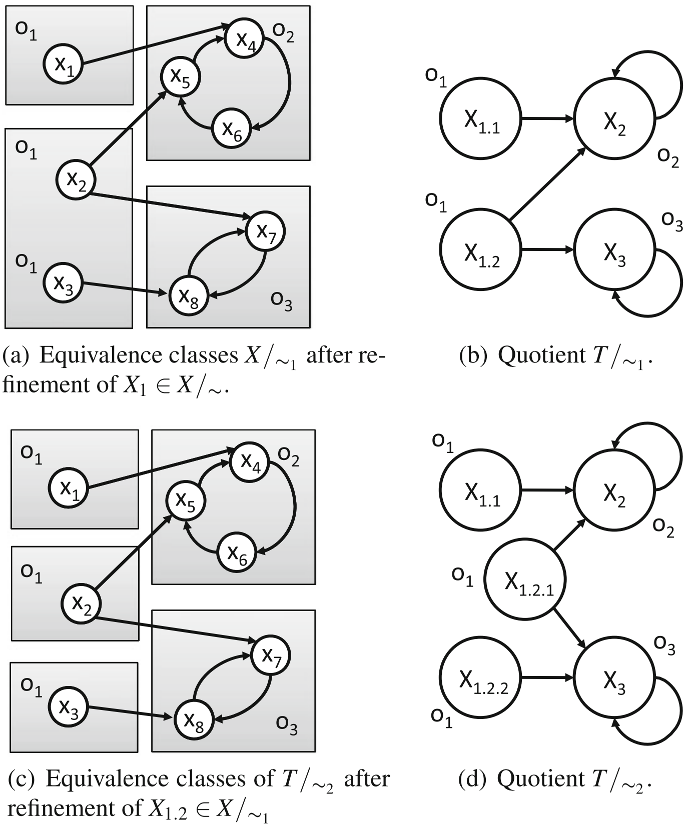
Fig. 1.13 Quotient $T/_\sim$ is the same for the transition systems from Fig. 1.12a, b. While $T/_\sim$ is a bisimulation quotient for the system from Fig. 1.12b, it only simulates the one from Fig. 1.12a. For that system, the bisimulation characterization from Theorem 1.1 is violated at state X_1. See Example 1.11 for details

Equation (1.21) leads to an approach for the construction of the coarsest bisimulation $\approx$. Given a transition system T, the iterative procedure known as the "bisimulation algorithm" (summarized as Algorithm 1) starts with the observational equivalence relation $\sim$ and uses it to identify equivalence classes from $X/_\sim$ that satisfy Eq. (1.21). Then, it iteratively refines these classes until the characterization from Theorem 1.1 is satisfied. This guarantees that the equivalence relation $\sim_r$ returned after the algorithm terminates is indeed $\approx$—a bisimulation of T, which can be used to construct the bisimulation quotient $T/_\approx$.

Example 1.12 Consider transition system T from Fig. 1.12a, with quotient $T/_\sim$ (Fig. 1.13) induced by the observational equivalence relation $\sim$. As already discussed in Example 1.11, $T/_\sim$ is not a bisimulation quotient (the characterization from Theorem 1.1 is violated at state X_1, for which Eq. (1.21) is satisfied). In order to obtain the coarsest observation preserving equivalence relation of T, the bisimulation algorithm (Algorithm 1) is applied to T.

First, equivalence classes X_1 and X_3 of $T/_\sim$ are considered, where $\emptyset \subset con(X_1) \cap Pre(con(X_3)) \subset con(X_1)$. Equivalence class X_1 is refined into $X_{1.1}$ and $X_{1.2}$, such that $con(X_{1.1}) = con(X_1) \setminus Pre(con(X_3)) = \{x_1\}$ and $con(X_{1.2}) = Pre(con(X_3)) \cap con(X_1) = \{x_2, x_3\}$, which leads to the construction of the intermediate equivalence relation $\sim_1$ represented in Fig. 1.14a. Equivalence relation $\sim_1$ induces quotient $T/_{\sim_1}$ (Fig. 1.14b) but is still not a bisimulation (the characterization from Theorem 1.1 is violated for $X_{1.2} \in X/_{\sim_1}$). Therefore, Algorithm 1 is applied again.

Equivalence classes $X_{1.2}$ and X_2 of $X/_{\sim_1}$ are then considered and $X_{1.2}$ is refined into $X_{1.2.1}$ and $X_{1.2.2}$, such that $con(X_{1.2.1}) = con(X_{1.2}) \cap Pre(con(X_2)) = \{x_3\}$ and $con(X_{1.2.2}) = con(X_{1.2}) \setminus Pre(con(X_2)) = \{x_2\}$. The resulting equivalence relation $\sim_2$ represented in Fig. 1.14c is a bisimulation and induces the bisimulation quotient $T/_{\sim_2} = T/_\approx$ shown in Fig. 1.14d.



(a) Equivalence classes $X/_{\sim_1}$ after refinement of $X_1 \in X/_{\sim}$.

(b) Quotient $T/_{\sim_1}$.

(c) Equivalence classes of $T/_{\sim_2}$ after refinement of $X_{1.2} \in X/_{\sim_1}$

(d) Quotient $T/_{\sim_2}$.

Fig. 1.14 The equivalence relations $\sim_1$ and $\sim_2$, obtained at successive steps while applying the bisimulation algorithm to system T from Fig. 1.12a, induce quotients $T/_{\sim_1}$ and $T/_{\sim_2}$, respectively. See Example 1.12 for additional details

1.4 Notes

In this chapter we introduced transition systems as a modeling formalism for a wide range of processes [11]. Besides capturing the behavior of many processes directly, transition systems provide a semantical model for various high-level formalisms for concurrent systems including Kripke structures [45], process algebras [88, 131, 132], statecharts [81] and Petri nets [140] (also see [39] for additional discussion on discrete event systems). In addition, the formalisms of Mealy and Moore machines, which are related to finite state automata and are commonly used in hardware synthesis and analysis, can be described by transition systems [110]. Finally, in biological applications, Boolean networks [101] are a popular model approximating the behavior of large genetic and signaling networks. The state spaces of both Boolean networks and their extending qualitative networks [154] and generalized logical networks [166] can also be expressed as finite, and in many cases deterministic, transition systems.

We showed that infinite transition systems are rich enough to describe dynamical (control) systems. Such embeddings have been proposed by several authors [5, 7, 72, 74, 105, 106, 138, 162, 163, 180, 184, 185]. While such embeddings can be easily defined for continuous-time systems as well [105, 106, 138], we do not give these definitions as the focus in this book is on discrete-time systems only.

The transition system definition that we consider here is somewhat different from the ones encountered elsewhere [11, 15, 45]. We allow a state $x \in X$ of the system to have only a single observation from the set of observations O, which is given by the observation map $o(x) \in O$. In more classical definitions, a state can have several observations from the set O (or, equivalently, a state might satisfy several propositions from a given set) and, as a result, the observation map o gives the set of observations $o(x) \in 2^O$ at state $x \in X$. Such a formulation can be reduced without loss of generality to the one we use by redefining the set of observations (i.e., by defining a new observation for each subset from 2^O). As it will become clear in Chap. 2, this assumption will also induce a slightly different interpretation of the semantics of the temporal logic formulas.

In this book, we deal with transition systems that are not initialized (i.e., we do not specify a set of initial states for the system) unlike the systems that are commonly encountered elsewhere [15, 45]. Such a definition is more appropriate for the analysis applications we consider in subsequent chapters. We still formulate the model checking problem as deciding if all runs from a given region satisfy a specification, which is equivalent to model checking a transition system that is initialized at that region. In other texts, only transition systems that have been initialized at a single state can be deterministic but according to our formulation, determinism is a property of the transitions and not of the initial states of a system. This allows us to search for sets of initial states rather than individual initial states as part of the analysis problem we consider in Chap. 4, even when deterministic systems are considered.

As probabilistic dynamics are not covered in this book, the transition systems defined in this chapter capture only purely deterministic and nondeterministic behaviors. The probabilistic version of the transition system considered here is the well known Markov decision process (MDP), in which the inputs enable transitions with given probability distributions among the states of the system [15]. Throughout the book, we also assume that the states of the system are observable. In other words, the current state of the system is known and available for state-feedback control. Readers interested in systems for which this assumption is relaxed are referred to transition systems with nondeterministic observations [8], for which the current state is known only to belong to a given set, and probabilistic versions such as hidden Markov models (HMM) [26], partially observable Markov decision processes (POMDP) [98], and mixed observability Markov decision processes (MOMDPs) [135].

The notions of simulation and bisimulation that we consider here are relations between systems and their quotients. Such relations can be defined, in general, between systems sharing the same sets of observations [11]. There also exist relaxed notions of bisimulations, such as alternating [6], weak [131], probabilistic [116], and approximate bisimulations [67, 145], which go beyond the scope of this book.

Chapter 2
Temporal Logics and Automata

Throughout this book, we consider analysis and control specifications given as formulas of a particular type of temporal logic, called Linear Temporal Logic (LTL). Such formulas are expressive enough to capture a rich spectrum of properties, including safety (nothing bad will ever happen), liveness (something good will eventually happen), and more complex combinations of Boolean and temporal statements. For example, for the robot from Example 1.4, an LTL formula can express a rich mission specification such as: "Keep on collecting messages from data gather region G and bring them back to the base B. Collect a message and recharge at one of the R regions between any two visits to the base. Always avoid the dangerous region D". In this chapter, we introduce the syntax and semantics of LTL and of one of its fragments, called syntactically co-safe LTL (scLTL), and we illustrate them through several examples. We also define the automata that will be later used for system analysis and control from such specifications.

2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) formulas are constructed from a set of observations, Boolean operators, and temporal operators. We use the standard notation for the Boolean operators (i.e., $\top$ (true), $\neg$ (negation), $\wedge$ (conjunction)) and the graphical notation for the temporal operators (e.g., $\bigcirc$ ("next"), U ("until")). The $\bigcirc$ operator is a unary prefix operator and is followed by a single LTL formula, while U is a binary infix operator. Formally, we define the syntax of LTL formulas as follows:

Definition 2.1 (*LTL Syntax*) A (propositional) Linear Temporal Logic (LTL) formula ϕ over a given set of observations O is recursively defined as

$$\phi = \top \mid o \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 U \phi_2, \tag{2.1}$$

where $o \in O$ is an observation and ϕ, ϕ_1 and ϕ_2 are LTL formulas.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_2

Unary operators have a higher precedence than binary ones and $\neg$ and $\bigcirc$ bind equally strong. The temporal operator U takes precedence over $\neg$ and $\wedge$ and is right-associative (e.g., $\phi_1 U \phi_2 U \phi_3$ stands for $\phi_1 U (\phi_2 U \phi_3)$).

To obtain the full expressivity of propositional logic, additional operators are defined as

$$\begin{aligned}\phi_1 \vee \phi_2 &:= \neg(\neg\phi_1 \wedge \neg\phi_2)\\ \phi_1 \rightarrow \phi_2 &:= \neg\phi_1 \vee \phi_2\\ \phi_1 \leftrightarrow \phi_2 &:= (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)\end{aligned}$$

In addition, the temporal operators $\diamondsuit$ ("eventually") and $\square$ ("always") are defined as follows:

$$\begin{aligned}\diamondsuit\phi &:= \top U \phi\\ \square\phi &:= \neg\diamondsuit\neg\phi\end{aligned}$$

By combining the various temporal operators, more complicated expressions can be obtained. For example, we will frequently use the combinations $\diamondsuit\square$ ("eventually always") and $\square\diamondsuit$ ("always eventually").

LTL formulas are interpreted over infinite words made of observations from O, i.e., over O^ω. Formally, the LTL semantics are defined as follows:

Definition 2.2 (*LTL Semantics*) The satisfaction of formula ϕ over a set of observations O at position $k \in \mathbb{N}_+$ by word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$, denoted by $w_O(k) \models \phi$, is defined recursively as follows:

- $w_O(k) \models \top$,
- $w_O(k) \models o$ for some $o \in O$ if $w_O(k) = o$,
- $w_O(k) \models \neg\phi$ if $w_O(k) \not\models \phi$,
- $w_O(k) \models \phi_1 \wedge \phi_2$ if $w_O(k) \models \phi_1$ and $w_O(k) \models \phi_2$,
- $w_O(k) \models \bigcirc\phi$ if $w_O(k+1) \models \phi$,
- $w_O(k) \models \phi_1 U \phi_2$ if there exist $j \geq k$ such that $w_O(j) \models \phi_2$ and, for all $k \leq i < j$, we have $w_O(i) \models \phi_1$.

A word w_O satisfies an LTL formula ϕ, written as $w_O \models \phi$, if $w_O(1) \models \phi$. We denote the language of infinite words that satisfy formula ϕ by $\mathscr{L}_\phi$.

In the following, we give an informal interpretation of the satisfaction of some frequently used LTL formulas.

- $\bigcirc\phi$ is satisfied at the current step if ϕ is satisfied at the next step.
- $\phi_1 U \phi_2$ is satisfied if ϕ_1 is satisfied "until" ϕ_2 becomes satisfied,
- $\square\phi$ is satisfied if ϕ is satisfied at each step (i.e., ϕ is "always" satisfied).

- $\Box\neg\phi$ is satisfied if $\neg\phi$ is satisfied at each step (i.e., ϕ is "never" satisfied).
- $\Diamond\phi$ is satisfied if ϕ is satisfied at some future step (i.e., ϕ is "eventually" satisfied).
- $\Diamond\Box\phi$ is satisfied if ϕ becomes satisfied at some future step and remains satisfied for all following steps (i.e., ϕ is satisfied "eventually forever").
- Formula $\Box\Diamond\phi$ is satisfied if ϕ always becomes satisfied at some future step (i.e., ϕ is satisfied "infinitely often").

Example 2.1 Consider the transition system T defined in Example 1.2 and shown in Fig. 1.2. A possible run of the system $w_X = x_1x_2x_4x_3(x_1)^\omega$ defines the output word $w_O = o_1o_1o_2o_3(o_1)^\omega$ which satisfies LTL formulas $\phi_1 = o_1$, $\phi_2 = \Diamond\Box o_1$ and $\phi_3 = o_1 \mathcal{U} o_2$. A different run $w'_X = (x_1x_2x_4x_3)^\omega$ defines the output word $w'_O = (o_1o_1o_2o_3)^\omega$ which satisfies formulas ϕ_1, ϕ_3 and $\phi_4 = \Box\Diamond o_3$. However, word w_O does not satisfy formula ϕ_4 and w'_O does not satisfy ϕ_2.

Example 2.2 Consider the robot transition system described in Example 1.4. Assume that the robot is required to keep collecting messages from data gather region G and to bring them back to the base B. While doing this, it needs to recharge at one of the recharge regions R. The robot must always avoid the dangerous region D. This task can be represented as an LTL formula

$$\phi = \Box\Diamond G \wedge \Box\Diamond B \wedge \Box\Diamond R \wedge \Box\neg D.$$

An additional requirement might be that the robot needs to collect a message and recharge between any two visits to the base. The overall task can be expressed as the following LTL formula

$$\psi = \Box\Diamond B \wedge \Box\neg D \wedge \Box(B \Rightarrow \bigcirc(\neg B \mathcal{U} G)) \wedge \Box(B \Rightarrow \bigcirc(\neg B \mathcal{U} R)).$$

Control strategies for the transition system from Example 1.4 from these specifications will be derived in Example 5.8.

An LTL formula belongs to the class of syntactically co-safe LTL formulas if it contains only the temporal operators $\bigcirc$, $\mathcal{U}$ and $\Diamond$, and it is written in positive normal form (the negation operator $\neg$ occurs only in front of an observation). Formally, we define the syntax of scLTL formulas as follows:

Definition 2.3 (*scLTL Syntax*) A (propositional) syntactically co-safe linear temporal logic (scLTL) formula ϕ over a set of observations O is recursively defined as

$$\phi = \top \mid o \mid \neg o \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 U \phi_2 \tag{2.2}$$

where $o \in O$ is an observation and ϕ, ϕ_1 and ϕ_2 are scLTL formulas.

Temporal operator $\diamondsuit$ is defined in scLTL as before, i.e., $\diamondsuit\phi := \top U \phi$. However, temporal operator $\Box$ can not be expressed in scLTL since only observations can be negated, i.e., $\neg\diamondsuit\neg\phi$ does not belong to the scLTL fragment.

Even though scLTL formulas are interpreted over infinite words, i.e., over O^ω, as explained in Definition 2.2, their satisfaction is guaranteed in finite time. Any infinite word $w_O = w_O(1)w_O(2)w_O(3)\ldots$ that satisfies formula ϕ contains a finite "good" prefix $w_O(1)w_O(2)\ldots w_O(n)$ such that all infinite words that contain the prefix, i.e., $w_O(1)w_O(2)\ldots w_O(n)w'_O$, $w'_O \in O^\omega$, also satisfy ϕ. We denote the language of finite good prefixes of an scLTL formula ϕ by $\mathscr{L}_{pref,\phi}$.

Example 2.3 Consider again transition system T from Example 1.2 and Fig. 1.2. The run $w_X = x_2x_4(x_3)^\omega$ defines the output word $w_O = o_1o_2(o_3)^\omega$. The word w_O satisfies scLTL formulas $\phi_1 = \diamondsuit o_1$ and $\phi_2 = \diamondsuit o_3 \wedge (o_1 U o_2)$ since w_O contains a good prefix of each of the formulas, i.e., o_1 for ϕ_1 and $o_1o_2o_3$ for ϕ_2. In particular, all output words defined by system runs originating from $X_r = \{x_1, x_2\}$ contain the finite prefix o_1, and therefore satisfy ϕ_1.

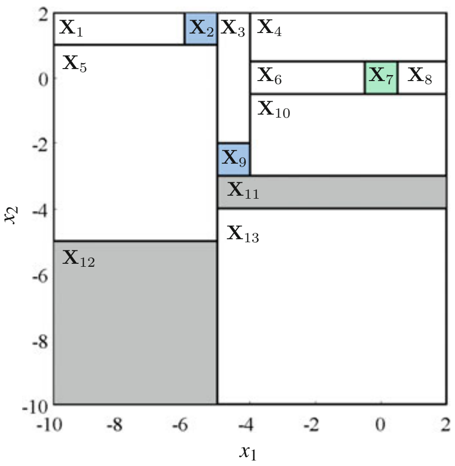


Fig. 2.1 The partitioned planar environment for Example 2.4. A control strategy driving a simple vehicle modeled as a discrete-time double integrator such that its motion satisfies specification (2.3) will be derived in Example 11.5

Example 2.4 Consider an agent moving in the planar environment from Fig. 2.1. The specification is to visit regions $\mathbf{X}_2$ or $\mathbf{X}_9$ and then the target region $\mathbf{X}_7$, while avoiding $\mathbf{X}_{11}$ and $\mathbf{X}_{12}$, and staying inside $\mathbf{X} = [-10, 2]^2$ until the target region is reached. This specification translates to the following scLTL formula:

$$\phi = ((\neg\mathbf{X}_{11} \ \wedge \ \neg\mathbf{X}_{12} \ \wedge \ \neg Out) \ U \ \mathbf{X}_7) \wedge (\neg\mathbf{X}_7 \ U \ (\mathbf{X}_2 \ \vee \ \mathbf{X}_9)), \qquad (2.3)$$

where $Out = \mathbb{R}^2 \setminus \mathbf{X}$.

2.2 Automata

We will use automata that accept languages satisfying LTL and scLTL formulas over the set of observations O. There is, therefore, no coincidence that the input alphabets of the automata defined below is O.

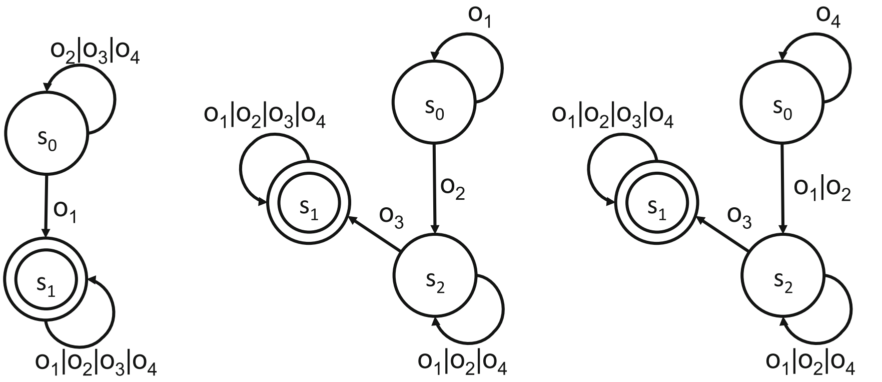
Definition 2.4 (*Finite state automaton*) A finite state automaton (FSA) is a tuple $A = (S, s_0, O, \delta, F)$, where

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- O is the input alphabet,
- $\delta : S \times O \to S$ is a transition function, and
- $F \subseteq S$ is the set of accepting (final) states.

The semantics of a finite state automaton are defined over finite input words in O^*. A run of A over a word $w_O = w_O(1)w_O(2), \ldots, w_O(n) \in O^*$ is a sequence $w_S = w_S(1)w_S(2), \ldots, w_S(n+1) \in S^*$ where $w_S(1) = s_0$ and $w_S(k+1) = \delta(w_S(k), w_O(k))$ for all $k = 1, 2, \ldots, n$. The word w_O is accepted by A if and only if the corresponding run ends in a final automaton state, i.e., $w_S(n+1) \in F$. The language accepted by A is the set of all words accepted by A, and is denoted by $\mathscr{L}_A$.

A finite state automaton with a non-deterministic transition function, i.e., $\delta \colon S \times O \to 2^S$, and a set of initial states $S_0 \subseteq S$ instead of the singleton s_0 is called a non-deterministic finite state automaton (NFA). Every NFA can be translated to an equivalent FSA. For this reason, we only consider deterministic finite state automata in this book.

An scLTL formula ϕ over a set O can always be translated into an FSA A_ϕ with input alphabet O with $\mathscr{O}(2^{2^{|\phi|}})$ states ($|\phi|$ denotes the length of ϕ, which is defined as the total number of occurrences of observations and operators) that accepts all and only good prefixes of ϕ (i.e., $\mathscr{L}_{A_\phi} = \mathscr{L}_{pref,\phi}$). Some notes on available tools for this translation are given in Sect. 2.3 at the end of the chapter.



(a) $\phi_1 = \Diamond o_1$ (b) $\phi_2 = \Diamond o_3 \wedge (o_1 U o_2)$ (c) $\phi_3 = (\neg o_3 U (o_1 \vee o_2)) \wedge \Diamond o_3$

Fig. 2.2 Graphical representation of the finite state automata for some scLTL formulas over the set of observations $O = \{o_1, o_2, o_3, o_4\}$. For all automata, s_0 is the initial state and the final state is indicated by a *double circle*. For simplicity of the representation, if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown

Example 2.5 The finite state automata that accept the good prefixes of scLTL formulas $\phi_1 = \Diamond o_1$, $\phi_2 = \Diamond o_3 \wedge (o_1 U o_2)$, and $\phi_3 = (\neg o_3 U (o_1 \vee o_2)) \wedge \Diamond o_3$ over the set of observations $O = \{o_1, o_2, o_3, o_4\}$ are shown in Fig. 2.2.

Definition 2.5 (*Büchi automaton*) A (nondeterministic) Büchi automaton is a tuple $B = (S, S_0, O, \delta, F)$, where

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- O is the input alphabet,
- $\delta : S \times O \to 2^S$ is a nondeterministic transition function, and
- $F \subseteq S$ is the set of accepting (final) states.

A Büchi automaton is deterministic if S_0 is a singleton and $\delta(s, o)$ is either $\emptyset$ or a singleton for all $s \in S$ and $o \in O$. The semantics of a Büchi automaton are defined over infinite input words in O^ω. A run of B over a word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$ is a sequence $w_S = w_S(1)w_S(2)w_S(3)\ldots \in S^\omega$ where $w_S(1) \in S_0$ and $w_S(k+1) \in \delta(w_S(k), w_O(k))$ for all $k \geq 1$.

Definition 2.6 (*Büchi acceptance*) Let $\inf(w_S)$ denote the set of states that appear in the run w_S infinitely often. An input word w_O is accepted by B if and only if there exists at least one run w_S over w_O that visits F infinitely often, i.e., $\inf(w_S) \cap F \neq \emptyset$.

We denote by $\mathscr{L}_B$ the language accepted by B, i.e., the set of all words accepted by B. An LTL formula ϕ over a set O can always be translated into a Büchi automaton

B_ϕ with input alphabet O and $\mathscr{O}(|\phi| \cdot 2^{|\phi|})$ states ($|\phi|$ denotes the length of ϕ, which is defined as the total number of occurrences of observations and operators) that accepts all and only words satisfying ϕ (i.e., $\mathscr{L}_{B_\phi} = \mathscr{L}_\phi$). This translation can be performed using efficient, off-the-shelf software tools, which are reviewed at the end of the chapter in Sect. 2.3.

Note that, in general, a nondeterministic Büchi automaton is obtained by translating an LTL formula. While certain Büchi automata can be determinized, a sound and complete procedure for determinizing general Büchi automata does not exist and, in fact, there exist LTL formulas which cannot be converted to deterministic Büchi automata.

Example 2.6 Examples of Büchi automata for some commonly encountered LTL formulas are shown in Fig. 2.3. Even when a nondeterministic Büchi automaton was obtained through the translation with LTL2BA tool, the automaton was simplified and determinized by hand whenever possible (e.g., for formulas ϕ_1, ϕ_4 and ϕ_5). Even so, some of the automata, such as the ones obtained for LTL formulas ϕ_3, ϕ_6 and ϕ_7 cannot be determinized. In fact, it is known that formulas of the type $\diamondsuit\square\phi$ cannot be converted to a deterministic Büchi automaton. For example, the Büchi automaton for LTL formula ϕ_3 in Fig. 2.3c can be naively converted into a deterministic automaton by removing o_1 from the self transition at s_0. However, then word $o_1 o_2 (o_1)^\omega$, which is obviously satisfying, would not be accepted. While, in general, deterministic Büchi automata can be obtained for a class of LTL formulas through alternative approaches other than simply converting non-deterministic to deterministic Büchi automata, no such automaton exists for ϕ_3. To understand why formulas ϕ_6 and ϕ_7 result in nondeterministic Büchi automata, we can rewrite them as

$$\phi_6 = \square\diamondsuit o_1 \wedge \neg\square\diamondsuit o_2 = \square\diamondsuit o_1 \wedge \diamondsuit\square\neg o_2 \tag{2.4}$$

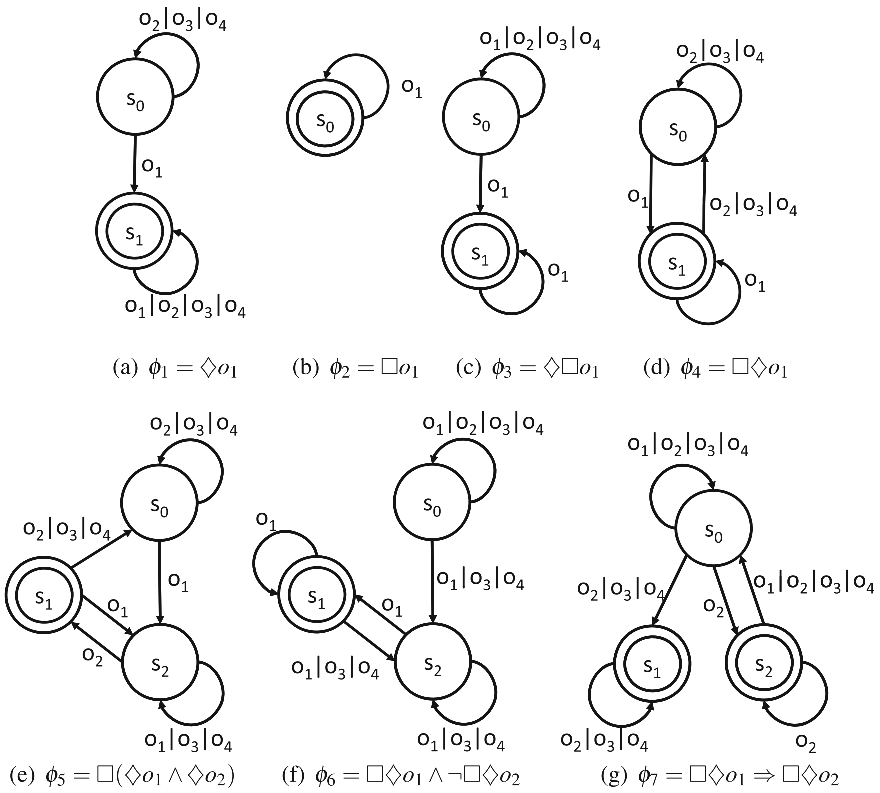
$$\phi_7 = \square\diamondsuit o_1 \Rightarrow \square\diamondsuit o_2 = (\square\diamondsuit o_1 \wedge \square\diamondsuit o_2) \vee \neg\square\diamondsuit o_1 = \tag{2.5}$$

$$= (\square\diamondsuit o_1 \wedge \square\diamondsuit o_2) \vee \diamondsuit\square\neg o_1 \tag{2.6}$$

to reveal that both contain a $\diamondsuit\square$ sub-formula.

Definition 2.7 (*Rabin automaton*) A (nondeterministic) Rabin automaton is a tuple $R = (S, S_0, O, \delta, F)$, where

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of initial states,
- O is the input alphabet,



(a) $\phi_1 = \Diamond o_1$ (b) $\phi_2 = \Box o_1$ (c) $\phi_3 = \Diamond\Box o_1$ (d) $\phi_4 = \Box\Diamond o_1$

(e) $\phi_5 = \Box(\Diamond o_1 \wedge \Diamond o_2)$ (f) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$ (g) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Fig. 2.3 Graphical representation of the Büchi automata for some commonly used LTL formulas over the set of observations $O = \{o_1, \ldots, o_4\}$. For all automata, s_0 is the initial state and the final states are indicated by *double circles*. As in Fig. 2.2, for simplicity of the representation if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown. For additional details, see Example 2.6

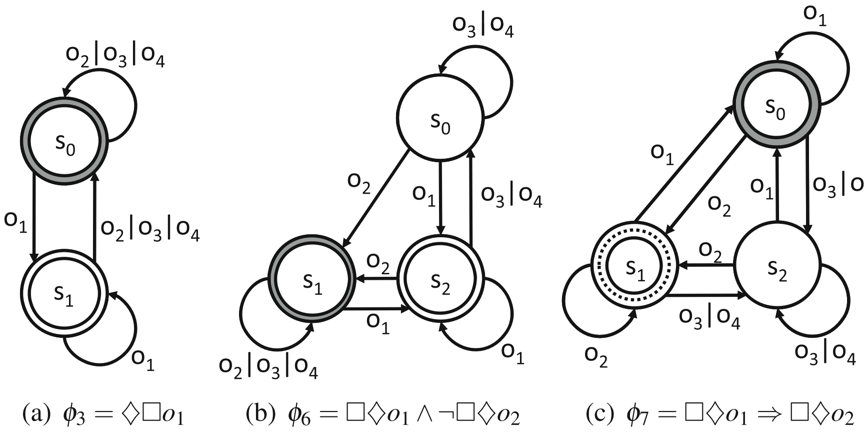
- $\delta : S \times O \to 2^S$ is a transition map, and
- $F = \{(G_1, B_1), \ldots, (G_n, B_n)\}$, where $G_i, B_i \subseteq S$, $i = 1, 2, \ldots, n$ is the acceptance condition.

A Rabin automaton R is deterministic if S_0 is a singleton and $\delta(s, o)$ is either $\emptyset$ or a singleton, for all $s \in S$ and $o \in O$. The semantics of a Rabin automaton are defined over infinite input words in O^ω. A run of R over a word $w_O = w_O(1)w_O(2)w_O(3)\ldots \in O^\omega$ is a sequence $w_S = w_S(1)w_S(2)w_S(3)\ldots \in S^\omega$, where $w_S(1) \in S_0$ and $w_S(k+1) \in \delta(w_S(k), w_O(k))$ for all $k \geq 1$.

Definition 2.8 (*Rabin acceptance*) Let $\inf(w_S)$ denote the set of states that appear in the run w_S infinitely often. A run w_S is accepted by R if $\inf(w_S) \cap G_i \neq \emptyset \wedge \inf(w_S) \cap B_i = \emptyset$ for some $i \in \{1, \ldots, n\}$. An input word w_O is accepted by a Rabin automaton R if some run over w_O is accepted by R.

We denote by $\mathscr{L}_R$ the language accepted by R, i.e., the set of all words accepted by R. Given an LTL formula ϕ, one can build a deterministic Rabin automaton R with input alphabet O, $2^{2^{\mathcal{O}(|\phi|\cdot\log|\phi|)}}$ states, and $2^{\mathcal{O}(|\phi|)}$ pairs in its acceptance condition, such that $\mathscr{L}_R = \mathscr{L}_\phi$. The translation can be done using off-the-shelf software tools reviewed in Sect. 2.3. Note that a Büchi automaton B is a Rabin automaton R with one pair in its acceptance condition $F_R = \{(G, B)\}$ where $G = F_B$ and $B = \emptyset$.

Example 2.7 Even though LTL formulas ϕ_3, ϕ_6 and ϕ_7 could only be translated into nondeterministic Büchi automata in Example 2.6, we can translate them instead into the deterministic Rabin automata shown in Fig. 2.4. The Rabin automata for formulas ϕ_3 and ϕ_6 contain only a single pair in their acceptance conditions, while the one for ϕ_7 contains two pairs.



(a) $\phi_3 = \Diamond\Box o_1$ (b) $\phi_6 = \Box\Diamond o_1 \wedge \neg\Box\Diamond o_2$ (c) $\phi_7 = \Box\Diamond o_1 \Rightarrow \Box\Diamond o_2$

Fig. 2.4 Graphical representation of the Rabin automata for the LTL formulas from Example 2.6 resulting in nondeterministic Büchi automata. For each automaton, s_0 is the initial state. For the automata accepting formulas ϕ_3 and ϕ_6 the acceptance condition F is defined by one pair of singletons (G, B) where $G = \{s_1\}$, $B = \{s_0\}$ in (**a**) and $G = \{s_2\}$, $B = \{s_1\}$ in (**b**) (good and bad states are denoted by *unshaded* or *shaded double circles*, respectively). For the automaton accepting ϕ_7 the acceptance condition includes two pairs where $G_1 = \{s_1, s_2\}$, $B_1 = \{s_0\}$ and $G_2 = \{s_1\}$, $B_2 = \emptyset$ (the single bad state is denoted by a *shaded circle* and the good state that is common for both pairs of the acceptance conditions is denoted by a *solid* and *dashed circle* in (**c**)). As in Fig. 2.3, for simplicity of the representation if several transitions are present between two states, only one transition labeled by the set of all inputs (separated by the symbol |) labeling all transitions is shown. For additional details, see Example 2.7

2.3 Notes

Temporal logics were originally developed by philosophers to reason about how truth and knowledge change over time. They were later adapted in computer science and used to specify the correctness of digital circuits and computer programs. Besides several more expressive temporal logics, Linear Temporal Logic (LTL), Computation Tree Logic (CTL) and the CTL* framework [45, 56], which is a superset of LTL and CTL, are the most commonly encountered. For the applications we consider, the computational expense involved in model checking (see Chap. 3) CTL* outweighs the gains in expressivity and therefore this logic is not considered. LTL and CTL are incomparable in the sense that there exist LTL formulas that cannot be expressed in CTL and vice versa. CTL is a branching time logic that allows for the quantification of specifications over the executions of the system. In other words, a CTL property can be satisfied by the system if it is satisfied over all paths (universal quantification) or if there exists a path that satisfies it (existential quantification). However, the additional semantics of CTL might make the formulation of specifications prone to errors [130, 155], as one must consider all possible executions of a system at the same time. On the other hand, expressing specifications in LTL is more natural because executions are considered one at a time. In the worst case, model checking CTL and LTL specifications respectively requires polynomial and exponential time in the size of the formula. While CTL model checking is computationally cheaper, empirical results suggest that performance is similar [172] for formulas expressible in both logics, since formulas in CTL can be larger than their equivalent LTL representation. Because of its resemblance to natural language, we adopt LTL as a specification formalism.

Fragments of LTL, such as GR(1) [143] and syntactically co-safe LTL (scLTL) [111, 156], have also been proposed as specification languages for verification and control. With particular relevance to this book, scLTL has been primarily used to verify safety of a system [111, 156]. As we will discuss in the next chapter, analysis of a system from an LTL formula ϕ involves constructing an automaton from the negation of the formula, i.e., $B_{\neg\phi}$. A safety property asserts that nothing bad happens to the system, e.g., $\Box$ "*safe*", and negation of a safety formula is called a co-safe formula, e.g., $\Diamond\neg$ "*safe*". As we presented in this chapter, an FSA is sufficient to recognize the words that satisfy a syntactically co-safe LTL formula, which reduces the computational complexity associated with the analysis of the corresponding safety property due to the simple acceptance condition of an FSA. In addition to analysis of safety properties, scLTL formulas are also used to express finite horizon specifications [30].

There are also some differences between the terms used here and elsewhere. The symbols appearing in an LTL formula are usually called *atomic propositions* in the formal methods community [45]. However, we call them *observations* as in this book we use LTL formulas to specify properties of words over observations O produced by transition systems $T = (X, \Sigma, \delta, O, o)$ (see Definition 1.1). This is consistent with control theoretic nomenclatures, where the term *output* is also used [162].

The semantics of LTL formulas are usually given over infinite words in the power set of the set of observations 2^O, as they are normally used to specify properties of transition systems with possibly several observations (atomic propositions satisfied) at each state (see Sect. 1.4). The available off-the-shelf tools for construction of FSA from scLTL formulas (SCHECK2 [117], based on the algorithm from [111]), Büchi automata from LTL formulas (LTL2BA [65, 66], based on algorithms from [173]), and Rabin automata from LTL formulas (LTL2DSTAR [102]) produce automata with input alphabet 2^O, i.e., which accept words over 2^O. This is commonly denoted by labeling transitions of the automaton with Boolean formulas over the observations from O (see Example 2.8). A transition is enabled by the set of subsets of O (i.e., the elements of 2^O) that satisfies the corresponding Boolean formula. However, as the transition systems that we consider in this book have exactly one observation at each state (see Definition 1.1), we simplify the automata produced by SCHECK2, LTL2BA, and LTL2DSTAR to only accept satisfying words over O. For example, Boolean terms or formulas that cannot be satisfied by any individual element of O (e.g., the conjunction $o_1 \wedge o_2$ of any two observations $o_1, o_2 \in O$) are not relevant for the applications we consider and can therefore be simplified. Such a simplification for a Büchi automaton is shown in Example 2.8 and similar simplifications apply to FSA and Rabin automata.

Example 2.8 Consider the LTL formula $\phi = (o_1 U o_2) \wedge \Diamond o_3$, defined over observations $O = \{o_1, o_2, o_3, o_4\}$. The Büchi automaton representation of the formula is obtained using LTL2BA and is given in Fig. 2.5 (states s_0 and s_3 are respectively the initial and final (accepting) state). A transition labeled by an observation is enabled by any subset of O that includes the observations (e.g., the self loop at state s_0 is enabled by observations (o_1),(o_1, o_2),(o_1, o_3),...,(o_1, o_2, o_3),...). Similarly, a transition labeled by a conjunction of observations is enabled under any subset of observations that includes both observations (e.g., the transition between states s_0 and s_1 labeled by the conjunction $o_1 \wedge o_3$, is enabled by observations (o_1, o_3), (o_1, o_2, o_3),(o_1, o_3, o_4),...). Finally, a transition labeled by "true" is enabled by any subset from 2^O.

In this book, we consider only observation from the set O and not the set of subsets 2^O. Therefore, a transition under any conjunction of inputs can never be enabled (i.e., the set of observations satisfying such conjunctions is always empty) and transitions that are never enabled can be safely ignored. In Fig. 2.5, we ignore the transitions from state s_0 to state s_1 and from s_0 to s_3. As a result, state s_1 becomes unreachable and can be ignored as well. This simplification reduces the number of states and transitions in the Büchi automaton, which improves the complexity of the methods that will be discussed subsequently.

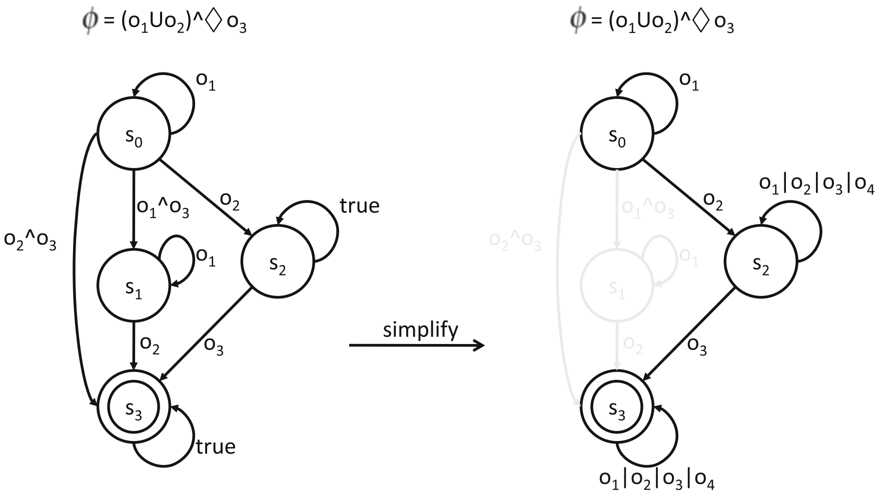


Fig. 2.5 Büchi automata used for our applications can be simplified as described in Example 2.8

Most temporal logics, including LTL, have probabilistic versions. In particular, the probabilistic version of LTL, called Probabilistic LTL (PLTL), is simply defined by adding a probability operator that quantifies the satisfaction probability in front of the formula. Its semantics is defined over a Markov decision process (MDP), the probabilistic version of the transition system defined in Chap. 1 (see Sect. 1.4). Probabilistic temporal logics go beyond the scope of this book, and the interested reader is referred to [4, 14, 15].

There also exist logics, such as Bounded Linear Temporal Logic (BLTL) [188], Signal Temporal Logic (STL) [126], and Metric Temporal Logic (MTL) [108], in which the temporal operators have specific time intervals. In such logics, one can specify eventuality with deadlines (e.g., $\diamondsuit_{[2,4]}o_1$—“o_1 will happen in between times 2 and 4”), persistence with time bounds (e.g., $\square_{[3,7]}o_2$—“o_2 will be true for all times between 3 and 7”), etc. In particular, MTL and STL also have quantitative semantics, which allow to quantify how far a system execution is from satisfying a given formula. Recent works [3, 12, 19–21, 54, 59, 91, 94, 95, 107, 151, 176] showed that logics with quantitative semantics can be used to formulate machine learning and control problems as optimization problems with costs induced by quantitative semantics.

Part II
Analysis and Control of Finite Transition Systems

Chapter 3
Model Checking

In this chapter, we introduce model checking, which is the most basic analysis problem in formal verification. As the focus of the book is on LTL, we restrict our attention to LTL specifications. Since we focus on analysis, we consider transition systems with no inputs. Informally, the LTL model checking problem consists of determining whether the language originating at a state of a finite transition system satisfies an LTL formula over its set of observations. The algorithms presented in this chapter will be extended in subsequent chapters to solve more difficult problems, such as finding the largest satisfying region for finite transition systems and infinite transition systems embedding discrete-time dynamical systems.

Given a finite transition system with no inputs $T = (X, \delta, O, o)$ and an LTL formula ϕ over O, checking whether all output words of T from a given subset of its states satisfy ϕ is called *LTL model checking*. Since we only use LTL in this book, LTL model checking is simply referred to as *model checking*.

Definition 3.1 (*LTL satisfaction*) Transition system T satisfies formula ϕ from a given region $X_r \subseteq X$, written as $T(X_r) \vDash \phi$, if and only if all words $w_O \in \mathscr{L}_T(X_r)$ produced by trajectories of T originating in X_r satisfy ϕ. Formally,

$$T(X_r) \vDash \phi \Leftrightarrow \forall w_O \in \mathscr{L}_T(X_r), w_O \vDash \phi, \tag{3.1}$$

which can be expressed equivalently as

$$T(X_r) \vDash \phi \Leftrightarrow \mathscr{L}_T(X_r) \subseteq \mathscr{L}_\phi. \tag{3.2}$$

Problem 3.1 (*Model checking)* Given a finite transition system T, a region $X_r \subseteq X$ of T, and an LTL formula ϕ over its set of observations O, determine if $T(X_r) \vDash \phi$.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_3

Model checking provides a technique for automatically determining if T satisfies ϕ when T is finite. An off-the-shelf model checker (see Sect. 3.1 for a review of such tools) takes as input a finite transition system T and a formula ϕ and returns a positive verification result only if all words produced by runs of the system originating in X_r satisfy the formula (i.e., if Eq. (3.2) is satisfied). Otherwise, the model-checker returns a non-satisfying run of T as a *counterexample* certifying the violation of the formula.

Deciding if the inclusion from Eq. (3.2) holds can be computationally challenging. Therefore, model checking algorithms often rely on computation using the negation of the LTL formula and the following equivalence:

$$\mathcal{L}_T(X_r) \subseteq \mathcal{L}_\phi \Leftrightarrow \mathcal{L}_T(X_r) \cap \mathcal{L}_{\neg\phi} = \emptyset \tag{3.3}$$

In other words, through Eq. (3.3) the model checking problem reduces to finding a word $w_O \in \mathcal{L}_T(X_r) \cap \mathcal{L}_{\neg\phi}$ and, if no such word exists, it is guaranteed that the system satisfies the formula from region X_r. To find a run w_X of T, which produces a word w_O such that $w_O \models \neg\phi$ (i.e., w_X is a counterexample in T), automata-theoretic model checking algorithms rely on the construction of a *product automaton* between the transition system and the Büchi automaton corresponding to the formula $\neg\phi$.

Definition 3.2 (*Uncontrolled Büchi Product Automaton*)[1] The *uncontrolled Büchi product automaton* $P = T \otimes B$ of a finite uncontrolled transition system $T = (X, \delta, O, o)$ and a Büchi automaton $B = (S, S_0, O, \delta_B, F)$ is defined as $P = (S_P, S_{0P}, \delta_P, F_P)$, where

- $S_P = X \times S$ is the set of states,
- $S_{0P} = X \times S_0$ is the set of initial states,
- δ_P is the transition function where, for a state $(x, s) \in S_P$, we have $\delta_P((x, s)) = \{(x', s') \in S_P \mid x' \in \delta(x) \text{ and } s' \in \delta_B(s, o(x))\}$,
- $F_P = X \times F$ is the set of accepting states.

This product automaton is a nondeterministic Büchi automaton with input alphabet containing only one element, which is therefore omitted. In addition, P is finite since both T and B are finite. A state $(x, s) \in S_P$ of P can be projected into a state $x \in X$ of T. We denote this projection by $\alpha : S_P \to X$, where $\alpha(x, s) = x$. A run $w_{S_P} = (x_1, s_1)(x_2, s_2)\ldots$ that is accepted by P can be projected into a run $w_X = \alpha(w_{S_P}) = \alpha(x_1, s_1)\alpha(x_2, s_2)\ldots = x_1x_2\ldots$ of T, such that $o(x_1)o(x_2)\ldots$ is accepted by B. If the product automaton $P_\phi = T \otimes B_\phi$ of a transition system T and the Büchi automaton B_ϕ is constructed (where B_ϕ accepts the language $\mathcal{L}_\phi$ for some

[1] There will be several versions of product automata throughout the book, e.g., controlled Büchi product automaton, controlled Rabin product automaton, controlled finite state product automaton. We will have formal definitions for all these product automata. However, whenever possible, we will call them simply product automata in the text, as their meaning will be clear from the context.

LTL formula ϕ as described in Definition 2.5), a run accepted by P_ϕ is projected into a run of T that produces a word w_O over O accepted by B_ϕ (i.e., w_O satisfies ϕ).

Note that a product automaton $P = T \otimes R$ analogous to the one from Definition 3.2 can be constructed with a Rabin automaton R (see Definition 5.3). In that case, the acceptance condition is inherited from the Rabin acceptance condition (see Definition 2.8).

In model checking a finite transition system T from region X_r, we construct the product automaton $P_{\neg\phi} = T \otimes B_{\neg\phi}$ and restrict the set of initial states of $P_{\neg\phi}$ to $S_{0P_{\neg\phi}} = X_r \times S_0$. A run $w_{S_{P_{\neg\phi}}} = (x_1, s_1)(x_2, s_2)\ldots$ that is accepted by $P_{\neg\phi}$ can be projected into a run $w_X = x_1 x_2 \ldots$ of T. If w_O is the word produced by w_X, then we have $w_O \vDash \neg\phi$ or, equivalently, $w_O \nvDash \phi$. Therefore, w_X is a counterexample proving that $T(X_r) \nvDash \phi$ and the model-checking problem reduces to finding the run $w_{S_{P_{\neg\phi}}}$ that is accepted by $P_{\neg\phi}$.

Finding an accepting run in $P_{\neg\phi}$ can be accomplished efficiently by treating it as a directed graph and decomposing it into *maximal strongly connected components* (SCCs)—maximal regions of $P_{\neg\phi}$ for which each state is reachable from every other one. Considering states of $P_{\neg\phi}$ that belong to the same SCC as equivalent induces a *quotient system*, where each state is a SCC (an equivalence class of states from $S_{P_{\neg\phi}}$) and a transition between states (SCCs) C_1 and C_2 exists if and only if there was a transition between some states $s_1 \in C_1$ and $s_2 \in C_2$ in $P_{\neg\phi}$. The corresponding SCC quotient is a directed acyclic graph. Initial and accepting states of the quotient system are SCCs that contain at least one initial or accepting state from $S_{P_{\neg\phi}}$, respectively. The existence of an accepting state (SCC) in the quotient system that is reachable from an initial state (SCC) is equivalent with the existence of a run that is accepted by $P_{\neg\phi}$. The former can be checked efficiently with basic graph algorithms. This computation provides the core of an LTL model checking algorithm and is summarized as Algorithm 2. The overall complexity of LTL model checking is $\mathscr{O}(|X| \cdot 2^{|\phi|})$.

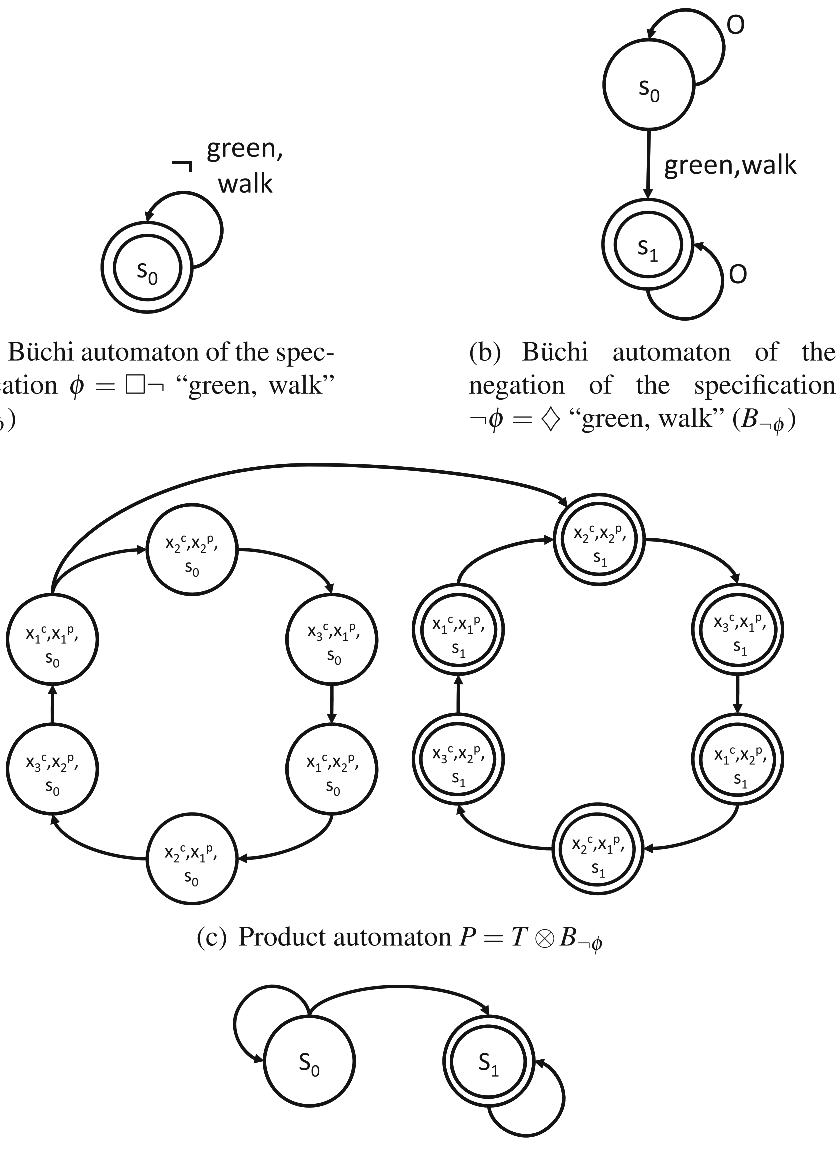
Algorithm 2 MODEL- CHECK(T, X_r, ϕ): Determine if finite transition system T satisfies LTL formula ϕ from region X_r

```
Translate the negation of the formula $\neg\phi$ to Büchi automaton $B_{\neg\phi}$
Construct the product automaton $P_{\neg\phi} = T \otimes B_{\neg\phi}$
Restrict the initial states of $P_{\neg\phi}$ to $S_{0P_{\neg\phi}} = X_r \times S_0$
Decompose $P_{\neg\phi}$ into SCCs
if there exists a run from an SCC containing an initial state (a state in $S_{0P_{\neg\phi}}$)
to an SCC containing an accepting state (a state from $F_{P_{\neg\phi}}$) then
  return false and the corresponding counterexample ($T(X_r) \nvDash \phi$)
else
  return true ($T(X_r) \vDash \phi$)
end if
```

Example 3.1 In order to illustrate the model checking procedure, we consider the pedestrian crossing traffic light system introduced in Example 1.6. For this system, the crucial safety property requires that pedestrians and cars are not allowed through the intersection at the same time—a mutual exclusion specification. Formally, we can express this requirement with the LTL formula $\phi = \Box\neg$ "green, walk" (shown as a Büchi automaton B_ϕ in Fig. 3.1a), specifying that the system should never visit a state where the car traffic light displays the "green" signal, while the pedestrian light displays the "walk" signal. To guarantee that the crossing operates safely, we are interested in model checking system T from Example 1.6 against specification ϕ from region $X_r = \{(x_1^c, x_1^p) \ldots (x_3^c, x_2^p)\}$, which includes all states of the system.

For transition system T shown in Fig. 1.6c it is clear that, regardless of the initial state, every trajectory of the system visits state (x_1^c, x_1^p), where "green, walk" is satisfied, and therefore specification ϕ is violated. To illustrate the process, in the following we go through the individual steps of the model checking procedure summarized in Algorithm 2.

i. The negation $\neg\phi$ of specification $\phi = \Box\neg$ "green, walk" (shown as a Büchi automaton in Fig. 3.1a) is translated into the Büchi automaton $B_{\neg\phi}$ shown in Fig. 3.1b (s_0 is the initial state of $B_{\neg\phi}$). Transitions of $B_{\neg\phi}$ are enabled by observations from the set $O = \{$"yellow, don't walk", "red, walk", "green, don't walk", "yellow, walk", "red, don't walk", "green, walk" $\}$.
ii. The product automaton $P_{\neg\phi} = T \otimes B_{\neg\phi}$ shown in Fig. 3.1c is constructed. Since all states are initial in T and only state s_0 is initial in $B_{\neg\phi}$, all states $(x_1^c, x_1^p, s_0) \ldots (x_3^c, x_2^p, s_0)$ are initial in $P_{\neg\phi}$. Similarly, since state s_1 is accepting in $B_{\neg\phi}$, all states $(x_1^c, x_1^p, s_1) \ldots (x_3^c, x_2^p, s_1)$ are accepting in $P_{\neg\phi}$.
iii. The product automaton $P_{\neg\phi}$ is decomposed into strongly connected components (SCCs) and the corresponding quotient system is shown in Fig. 3.1d, where state S_0 is initial and state S_1 is accepting.
iv. Since a run from state S_0 to state S_1 exists in the SCC quotient of $P_{\neg\phi}$, then there exists a trajectory of T that violates the specification. In other words, there exists a sequence of signals displayed at the intersection that eventually leads to the unsafe "green, walk" signal being displayed. In fact, for this particular system, all trajectories eventually lead to such unsafe behavior and can therefore serve as counterexamples.



(a) Büchi automaton of the specification $\phi = \Box\neg$ “green, walk” (B_ϕ)

(b) Büchi automaton of the negation of the specification $\neg\phi = \Diamond$ “green, walk” ($B_{\neg\phi}$)

(c) Product automaton $P = T \otimes B_{\neg\phi}$

(d) SCC quotient graph of P

Fig. 3.1 Automata involved in the model checking problem described in Example 3.1 for the pedestrian intersection traffic light system. For the Büchi automaton shown in (**a**), the transition labeled by ¬ “green, walk” signifies that there exists a transition enabled by each observation from the set O, except for one enabled by observation “green, walk”. For the Büchi automaton shown in (**b**), a transition labeled by the set of observations O signifies that there exists a transition enabled by each observation from O. See Example 3.1 for additional details

3.1 Notes

An in-depth discussion on model checking can be found in [45], where model checking algorithms for branching-time logics such as CTL are also included. For a more recent and comprehensive treatment, the reader is referred to [15], which also includes probabilistic verification, in which a Markov Chain is checked against a probabilistic CTL or LTL formula. Model checking algorithms based on nested depth-first search [48] have been developed as more memory-efficient alternatives to the SCC-based approach discussed in this chapter. Popular off-the-shelf implementations of model checking algorithms include tools such as SPIN [89], NuSMV [43], PRISM [114], and DiVinE [18].

As the complexity of model checking algorithms increases very fast with the sizes of systems, various symbolic representations have been developed to describe and manipulate finite transition systems. The most popular symbolic approach is based on binary decision diagrams (BDDs) [129]. The main limitation of the BDD-based representation is that its size depends on variable ordering and manual adjustments are usually necessary. Methods based on encodings such as Boolean satisfiability (SAT) problems, which take advantage of modern SAT solvers, were shown to perform well on large systems despite theoretical hardness results [31]. To deal with the difficulty of framing model checking problems as Boolean satisfiability problems, encodings based on Satisfiability Modulo Theories (SMT) have also been proposed [10].

In this book, model checking is performed against finite models or finite (simulation or bisimulation) abstractions of infinite models. There exist, however, model checking algorithms specifically adapted to apply directly to some classes of infinite systems, such as timed automata [7], pushdown automata [34, 58], or, more generally, well structured systems [2]. Even though the abstraction is not explicitly constructed in these methods, they are based on notions of decidability and computability similar to the ones used in this book.

Chapter 4
Largest Finite Satisfying Region

In Chap. 3, we introduced LTL model checking—a computational method for deciding automatically if a finite transition system T (Definition 1.1) satisfies an LTL formula ϕ (Sect. 2.1). This technique produces Yes/No answers (i.e., T satisfies ϕ or the specification is violated). In many applications, more quantitative results for the satisfaction of ϕ by T are required. For example, if the set of states was partitioned into satisfying and non-satisfying states, then the ratio of the cardinalities of the two sets would give us some indication on the degree of satisfaction. In this chapter, we focus on the problem of analyzing a finite transition system with the goal of partitioning its state space into satisfying and non-satisfying subsets of states. Since the focus is on analysis, as in Chap. 3, we consider transition systems with no inputs. In Chap. 5, we will present a control version of the same problem for finite transition systems. These techniques will then be extended in Chaps. 7 and 9 to infinite transition systems embedding discrete-time dynamical systems. The problem that we consider in this chapter can be formally stated as follows:

Problem 4.1 (*Largest satisfying region problem*) Given a finite transition system $T = (X, \delta, O, o)$ and an LTL formula ϕ over its set of observations O, find the largest subset of X from which ϕ is satisfied.

Before we begin developing our solution to Problem 4.1 in the following sections, we introduce several additional definitions necessary to formalize the problem and give an overview of our approach.

Definition 4.1 (*Largest satisfying region*) Given a transition system $T = (X, \delta, O, o)$ and an LTL formula ϕ over O,

$$X_T^{\phi} = \{x \in X \mid T(x) \vDash \phi\} \tag{4.1}$$

is the *largest satisfying region* of T—the set of all states of T where ϕ is satisfied.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_4

The set X_T^ϕ is the largest region of T satisfying ϕ since, for all states x of T where $x \notin X_T^\phi$, there exists a word in $\mathscr{L}_T(x)$ that violates ϕ. While any subset of X_T^ϕ contains only satisfying states and is therefore a "satisfying region" of T, the *largest satisfying region* X_T^ϕ is uniquely defined.

Definition 4.2 (*Largest violating region*) Given a transition system $T = (X, \delta, O, o)$ and an LTL formula ϕ over O, the set of all states of T where ϕ is not satisfied is the *largest violating region* of T

$$X \setminus X_T^\phi = \{x \in X \mid T(x) \nvDash \phi\} \tag{4.2}$$

Equivalently, there exists at least one run of T that violates ϕ originating in every state from the largest violating region.

Definition 4.3 (*Strictly violating region*) Given a transition system $T = (X, \delta, O, o)$ and an LTL formula ϕ over O,

$$X_T^{\neg\phi} = \{x \in X \mid T(x) \vDash \neg\phi\} \tag{4.3}$$

is the *strictly violating region* of T—the set of all states of T where the negation $\neg\phi$ is satisfied.

Relevant to analysis applications, no runs of T satisfying ϕ originate at any state $x \in X_T^{\neg\phi}$. Therefore, the strictly violating region is always a subset of the largest violating region. When T is deterministic, only a single run originates at each state $x \in X$ and satisfies either the formula ϕ or its negation $\neg\phi$ (i.e., $T(x) \vDash \phi$ or $T(x) \vDash \neg\phi$), allowing x to be included respectively in X_T^ϕ or $X_T^{\neg\phi}$ (see Example 4.1). As a result, X_T^ϕ and $X_T^{\neg\phi}$ partition the states of T (i.e., $X = X_T^\phi \cup X_T^{\neg\phi}$) for a deterministic T and the strictly violating region is also the largest violating region.

Example 4.1 Consider the finite, deterministic transition system T from Example 1.5 (Fig. 4.1) and specification $\phi = \bigcirc$ "on" requiring that, in the next time step, a state with the observation "on" is visited. The largest region of T satisfying ϕ is $X_T^\phi = \{x_3, \ldots, x_8\}$ and the strictly violating region for ϕ is $X_T^{\neg\phi} = \{x_1, x_2\}$ (both regions are shown in Fig. 4.1). Since transition system T is deterministic, the strictly violating region is also the largest region violating of T and the set of states $X = \{x_1, \ldots, x_8\}$ is partitioned into X_T^ϕ and $X_T^{\neg\phi}$, as shown in Fig. 4.1.

In general, when T is nondeterministic it is possible that both runs satisfying ϕ and runs satisfying $\neg\phi$ originate at a state $x \in X$ and therefore T does not satisfy either ϕ or $\neg\phi$ from x (i.e., $T(x) \nvDash \phi$ and $T(x) \nvDash \neg\phi$). Then, state x is an *uncertain* state of T with respect to the satisfaction of ϕ and is not included in either X_T^ϕ or $X_T^{\neg\phi}$ (see Example 4.2).

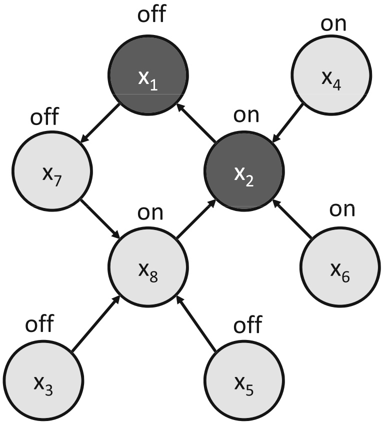


Fig. 4.1 Satisfying (*light gray*) and violating (*dark gray*) states of the deterministic system introduced in Example 1.5 (Fig. 1.5) for specification $\phi = \bigcirc$ "on" (for additional details, see Example 4.1)

Definition 4.4 (*Uncertain region*) Given a transition system $T = (X, \delta, O, o)$ and an LTL formula ϕ over O,

$$X_T^{\phi?} = \{x \in X \mid T(x) \nvDash \phi \text{ and } T(x) \nvDash \neg\phi\}. \tag{4.4}$$

is the *uncertain region* of T—the set of all states of T where neither ϕ nor $\neg\phi$ is satisfied.

The set $X_T^{\phi?}$ includes all uncertain states and, for a nondeterministic T where $X_T^{\phi?} \neq \emptyset$, the states of T is partitioned into the largest satisfying, strictly violating and uncertain regions $X = X_T^{\phi} \cup X_T^{\neg\phi} \cup X_T^{\phi?}$. As a result, the strictly violating region $X_T^{\neg\phi}$ is not necessarily the largest violating region (as for a deterministic T). Instead, the largest violating region is the union of the strictly violating and the uncertain regions, expressed equivalently as $X \setminus X_T^{\phi}$.

Example 4.2 Consider the finite, nondeterministic transition system T from Example 1.2 and specification $\phi = \bigcirc o_1$ requiring that, in the next time step, a state with the observation o_1 is visited. The largest region of T satisfying ϕ is $X_T^{\phi} = \{x_1\}$, while the strictly violating region for ϕ is $X_T^{\neg\phi} = \{x_2, x_4\}$. Runs satisfying both ϕ and $\neg\phi$ originate at state x_3, so $x_3 \notin X_T^{\phi}$ and $x_3 \notin X_T^{\neg\phi}$ and, therefore, state x_3 is uncertain (i.e., $x_3 \in X_T^{\phi?}$). The set of states $X = \{x_1, \ldots, x_4\}$ of T is partitioned into X_T^{ϕ}, $X_T^{\neg\phi}$ and $X_T^{\phi?}$, which are shown in Fig. 4.2.

While the solution to Problem 4.1 amounts to the computation of set X_T^{ϕ} only, sets $X_T^{\neg\phi}$ and $X_T^{\phi?}$ provide valuable information about system T and their explicit computation leads to significant optimizations of the overall analysis procedure, as it will become clear in the following sections. In the following, we refer to the computation

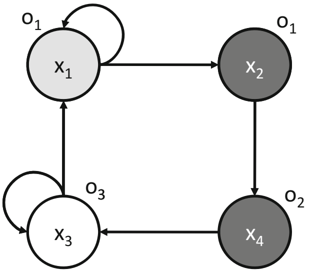


Fig. 4.2 Satisfying (*light gray*), violating (*dark gray*) and uncertain (*unshaded*) states of the nondeterministic system introduced in Example 1.2 (Fig. 1.2) for specification $\phi = \bigcirc o_1$ (for additional details, see Example 4.2)

of X_T^{ϕ}, $X_T^{\neg\phi}$ and $X_T^{\phi ?}$ as the analysis of a transition system. For a finite T, which is the focus of this chapter, all three sets are computable through a direct application of model checking (as illustrated in Sect. 4.1), thus providing a solution to Problem 4.1. However, such analysis becomes computationally challenging when T is large. To address this issue, in the following sections we present alternative abstraction-based analysis strategies, generally leading to conservative solutions of Problem 4.1, where under-approximations of sets X_T^{ϕ} and $X_T^{\neg\phi}$ are computed. Specifically, we focus on approaches for the iterative refinement and analysis of quotients of T, allowing an approximate solution to be computed and improved incrementally. In addition, we derive conditions guaranteeing that an exact solution to Problem 4.1 is obtained. Besides providing a more feasible strategy for the analysis of large finite transition systems, these abstraction techniques are also applicable to infinite transition systems, which we will exploit in Chap. 7.

4.1 Model-Checking-Based Solution

Given a finite transition system T, the subset of states where an LTL formula ϕ is satisfied is computable by model-checking T against ϕ from individual states. We denote this procedure by ANALYZE() where, given a region $X_r \subseteq X$,

$$\text{ANALYZE}(T, X_r, \phi) = \{x \in X_r \mid T(x) \vDash \phi\} \tag{4.5}$$

is the subset of X_r satisfying the formula ϕ. For a finite transition system T, the largest satisfying region is computed as X_T^{ϕ} =ANALYZE(T, X, ϕ), which provides a solution to Problem 4.1. Similarly, the strictly violating region is computed as $X_T^{\neg\phi}$ =ANALYZE$(T, X, \neg\phi)$. Once X_T^{ϕ} and $X_T^{\neg\phi}$ have been computed, the uncertain region of T is computed as $X_T^{\phi ?} = X \setminus (X_T^{\phi} \cup X_T^{\neg\phi})$.

An implementation of ANALYZE(), based on the function MODEL- CHECK() (Algorithm 2) from Chap. 3, is given in Algorithm 3. This implementation is applicable whenever model checking is feasible (i.e., when T is finite) but might require many model-checking steps. Therefore, if T has a large number of states, applying function ANALYZE() is computationally expensive. In the following sections we develop analysis procedures suitable for larger systems, based on the construction and analysis of finite abstractions of T.

Example 4.3 In Example 3.1 we showed that the traffic light system described in Example 1.6 was unsafe with respect to specification $\phi_1 = \Box\neg$ “green, walk”. In other words, there existed runs of the system which led to an unsafe state, where both cars and pedestrians were allowed to cross the intersection simultaneously. In this example, we modify the system and use the analysis procedure described in Algorithm 3 to find initial states, guaranteeing the safe behavior of the traffic light.

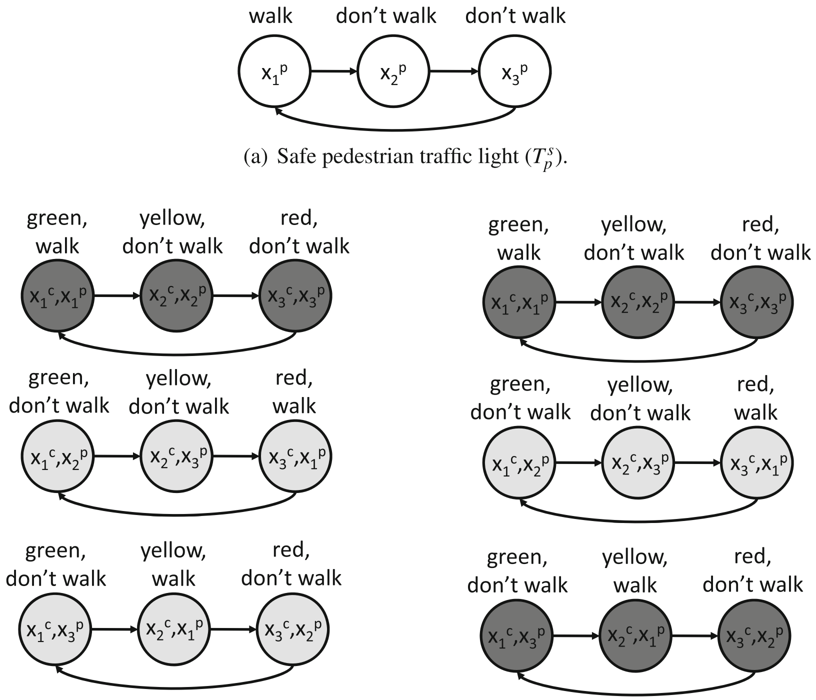
As before, we construct the overall system as the product $T^s = T_c \otimes T_p^s$ of the car and pedestrian traffic light components, where the car traffic light component T_c remains unchanged as in Example 3.1 (Fig. 1.6a). The pedestrian traffic light component T_p^s is modified from the one described in Example 1.6 and includes one additional state x_3^p with the observation “don’t walk” (see Fig. 4.3a).

By analyzing the product T^s with specification $\phi_1 = \Box\neg$ “green, walk”, we compute the violating region $X_{T^s}^{\neg\phi_1} = \{(x_1^c, x_1^p), (x_2^c, x_2^p), (x_3^c, x_3^p)\}$, where all other states of the system are satisfying and belong to $X_{T^s}^{\phi_1}$ (see Fig. 4.3b). Therefore, we can guarantee that the safety specification ϕ_1 is satisfied, as long as the system is initialized in a state from the satisfying region $X_{T^s}^{\phi_1}$.

We can also analyze the system with the stronger safety specification $\phi_2 = \Box\neg$ (“green, walk” $\vee$ “yellow, walk”), which requires that the potentially unsafe state of the system where pedestrians are allowed to cross but cars are not explicitly stopped by a red signal is also avoided. Analysis using Algorithm 3 reveals the satisfying region $X_{T^s}^{\phi_2} = \{(x_1^c, x_2^p), (x_2^c, x_3^p), (x_3^c, x_1^p)\}$ (Fig. 4.3c) and allows us to guarantee that the intersection is safe with respect to specification ϕ_2, provided that the system is initialized at a state from $X_{T^s}^{\phi_2}$.

Algorithm 3 ANALYZE(T, X_r, ϕ): Set of states of T in region X_r satisfying ϕ

```
Initialize $X_T^\phi := \emptyset$
for each state $x \in X_r$ do
    if MODEL- CHECK($T, x, \phi$) then
        $X_T^\phi := X_T^\phi \bigcup \{x\}$
    end if
end for
return $X_T^\phi$
```



(a) Safe pedestrian traffic light (T_p^s).

(b) Satisfying and violating regions for specification $\phi_1 = \Box\neg$ "green, walk".

(c) Satisfying and violating regions for specification $\phi_2 = \Box\neg$ ("green, walk" $\vee$ "yellow, walk").

Fig. 4.3 Satisfying (*light gray*) and violating (*dark gray*) regions are computed by analyzing the safe pedestrian intersection traffic light described in Example 4.3

4.2 Abstraction-Based Solution

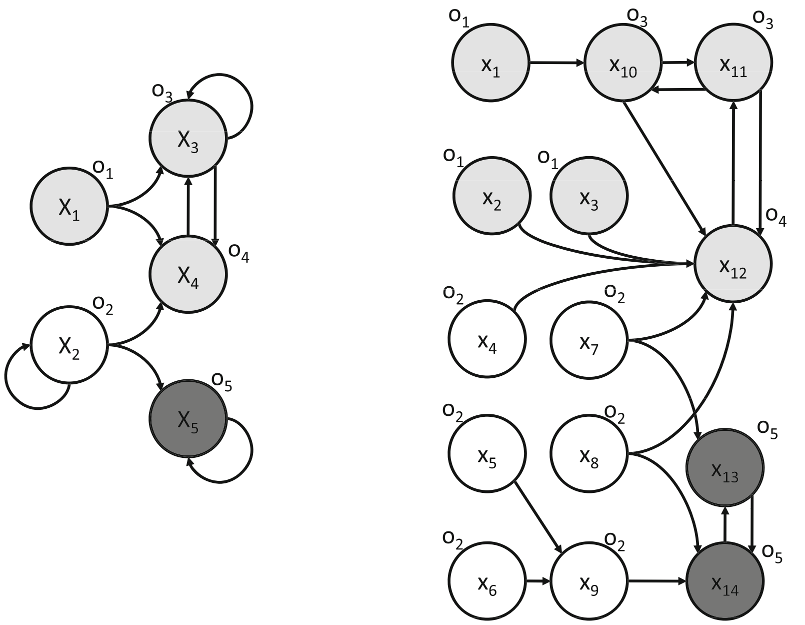
In Sect. 1.3 we introduced the notion of a quotient transition system $T/_\sim = (X/_\sim, \delta_\sim, O, o_\sim)$ as an abstraction of the concrete transition system $T = (X, \delta, O, o)$. The language inclusion property of Eq. (1.19) stated that, for all quotient states (equivalence classes of T) $X_i \in X/_\sim$, all words from the language of T originating from region $con(X_i)$ (representing the equivalent states of T from class X_i) are included in the language of the quotient $T/_\sim$ originating from state X_i (i.e., $\mathscr{L}_T(con(X_i)) \subseteq \mathscr{L}_{T/_\sim}(X_i)$). In other words, any behavior of T is reproduced by $T/_\sim$ (i.e., $T/_\sim$ simulates T), which guarantees that, for all quotient states $X_i \in X/_\sim$ and all LTL formulas ϕ, the concrete system T satisfies ϕ from region $con(X_i)$ only if the quotient $T/_\sim$ (usually much smaller than T) satisfies the formula from state X_i, i.e.,

$$T/_\sim(X_i) \vDash \phi \Rightarrow T(con(X_i)) \vDash \phi \tag{4.6}$$

This property is relevant to model checking, since it allows us to extend results obtained for the quotient $T/_\sim$ to the concrete transition system T. Equation (4.6) also guarantees that when a state X_i of $T/_\sim$ satisfies the negation $\neg\phi$, all states $x \in con(X_i)$ of T satisfy $\neg\phi$ and, therefore, violate ϕ. Note that this strategy is conservative, since identifying states of $T/_\sim$ that do not satisfy ϕ or $\neg\phi$ does not lead directly to any guarantees for T.

Similar to the model checking approach described above, we apply Eq. (4.6) in order to extend analysis results obtained for the quotient $T/_\sim$ to the original, concrete system T. Given LTL formula ϕ, we use Algorithm 3 to compute the largest satisfying region $X^{\phi}_{T/_\sim}$ of $T/_\sim$ (i.e., $X^{\phi}_{T/_\sim}$ =ANALYZE($T/_\sim$, $X/_\sim$, ϕ)). From Eq. (4.6) we guarantee that $con(X^{\phi}_{T/_\sim})$ is a satisfying region in T but, in general, it is not the largest satisfying region (i.e., $con(X^{\phi}_{T/_\sim}) \subseteq X^{\phi}_{T}$). However, this computation provides a strategy for obtaining satisfying regions of T when X^{ϕ}_{T} =ANALYZE(T, X, ϕ) cannot be computed directly (e.g., when T is large or infinite). The same approach is used to compute the strictly violating region $X^{\neg\phi}_{T/_\sim}$ =ANALYZE($T/_\sim$, $X/_\sim$, $\neg\phi$) in $T/_\sim$ with the guarantee that $con(X^{\neg\phi}_{T/_\sim})$ is a violating region of T but, in general, only a subset of its strictly violating region (i.e., $con(X^{\neg\phi}_{T/_\sim}) \subseteq X^{\neg\phi}_{T}$). While the computation of the uncertain region of $T/_\sim$ as $X^{\phi?}_{T/_\sim} = X/_\sim \setminus (X^{\phi}_{T/_\sim} \cup X^{\neg\phi}_{T/_\sim})$ is straightforward, in general, region $con(X^{\phi?}_{T/_\sim})$ is not the uncertainty region of T (i.e., it is possible that formula ϕ or its negation $\neg\phi$ is satisfied by all runs originating at a state $x \in con(X^{\phi?}_{T/_\sim})$). In fact, due to the under-approximation of the largest satisfying and strictly violating regions of T through the analysis of the quotient $T/_\sim$, region $con(X^{\phi?}_{T/_\sim})$ provides an over-approximation of the uncertain region of T (i.e., $X^{\phi?}_{T} \subseteq con(X^{\phi?}_{T/_\sim})$).

Example 4.4 We apply the analysis procedure based on quotient construction and Algorithm 3 to the transition system T from Example 1.10. We consider specification $\phi = \Box\Diamond o_3$, requiring that runs of the system keep visiting states with the observation o_3. The quotient $T/_\sim$ is constructed and, by applying the analysis procedure from Algorithm 3, the largest satisfying region $X^{\phi}_{T/_\sim} = \{X_1, X_3, X_4\}$ and the strictly violating region $X^{\neg\phi}_{T/_\sim} = \{X_5\}$ of $T/_\sim$ are identified (see Fig. 4.4a). This implies that the sets $con(X^{\phi}_{T/_\sim}) = \{x_1, x_2, x_3, x_{10}, x_{11}, x_{12}\}$ and $con(X^{\neg\phi}_{T/_\sim}) = \{x_{13}, x_{14}\}$ are respectively a satisfying and violating region of T (see Fig. 4.4b). However, these sets are only subsets of the largest satisfying and violating regions $X^{\phi}_{T} = \{x_1, x_2, x_3, x_4, x_{10}, x_{11}, x_{12}\}$ and $X^{\neg\phi}_{T} = \{x_5, x_6, x_9, x_{13}, x_{14}\}$ (Fig. 4.4c). We can also compute the uncertain region $X^{\phi?}_{T/_\sim} = \{X_2\}$ but region $con(X^{\phi?}_{T/_\sim}) = \{x_4, \ldots, x_9\}$ is an over-approximation of the uncertain region of T (i.e., only states x_7 and x_8 are uncertain in T).



(a) Satisfying (light gray) and violating (dark gray) states of $T/_{\sim}$.

(b) Satisfying (light gray) and violating (dark gray) states of T induced from the analysis of $T/_{\sim}$.

Fig. 4.4 Analysis of the finite quotient $T/_{\sim}$ with specification $\Box\Diamond o_3$ allows us to identify satisfying and violating states of T (see Example 4.4 for additional details)

The analysis approach discussed so far was conservative, due to the construction and analysis of a conservative simulation quotient of T—a limitation that is addressed through the construction of a bisimulation quotient. Equation (1.20) stated that the language of such a quotient is equivalent to the language of T (i.e., $\mathscr{L}_T(con(X_i)) = \mathscr{L}_{T/_{\approx}}(X_i)$), which allows us to guarantee that for all quotient states (equivalence classes of T) $X_i \in X/_{\approx}$ and all LTL formulas ϕ, we have

$$T/_{\approx}(X_i) \vDash \phi \Leftrightarrow T(con(X_i)) \vDash \phi. \tag{4.7}$$

Following from Eq. (4.7), we use the bisimulation quotient $T/_{\approx}$ equivalently instead of T for model checking (which was not true for the simulation quotient $T/_{\sim}$). In other words, by applying Algorithm 3 to the bisimulation quotient $T/_{\approx}$, we can compute the largest satisfying and violating regions of T through the computation of the largest satisfying and violating regions of $T/_{\approx}$ as

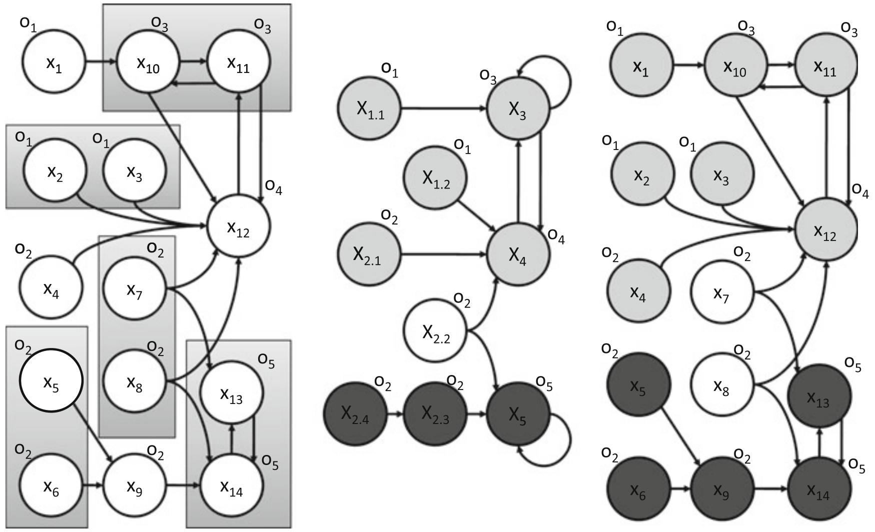
$$X_T^{\phi} = con(X_{T/\approx}^{\phi}) \text{ and } X_T^{\neg\phi} = con(X_{T/\approx}^{\neg\phi}) \tag{4.8}$$

As a consequence of Eq. (4.8), the set of uncertain states is computed as $X_T^{\phi ?} = con(X_{T/\approx}^{\phi ?})$, where $X_{T/\approx}^{\phi ?} = X/_\approx \setminus (X_{T/\approx}^{\phi} \cup X_{T/\approx}^{\neg\phi})$.

Using the results discussed so far, we obtain a solution to Problem 4.1, even when system T cannot be analyzed directly (e.g., when model checking is not feasible because T has too many or infinitely many states). The approach requires the computation of the quotient $T/_\sim$ induced by the observational equivalence relation $\sim$. When $T/_\sim$ simulates T, only an approximate solution to Problem 4.1 is obtained using this technique but the solution is exact when a bisimulation quotient $T/_\approx$ can be constructed. A strategy for the computation of bisimulation quotients was already presented in Sect. 1.3 as Algorithm 1. Thus, the overall analysis procedure described in this section involves (i) computing the coarsest observation-preserving equivalence relation $\approx$=BISIMULATION(T) (Algorithm 1), (ii) constructing the bisimulation quotient $T/_\approx$ and (iii) identifying the largest satisfying, strictly violating and uncertain regions of T through analysis of $T/_\approx$ using Algorithm 3 (see Example 4.5).

Example 4.5 The analysis approach discussed in this section is applied to study transition system T from Fig. 1.11a with specification $\Box\Diamond o_3$. In Example 4.4, we showed that simply constructing the quotient $T/_\sim$ under the observational equivalence relation $\sim$ and analyzing it using Algorithm 3 led to the identification of the satisfying region $\{x_1, x_2, x_3, x_{10}, x_{11}, x_{12}\}$ and violating region $\{x_{13}, x_{14}\}$ of T but the largest satisfying and strictly violating regions of T were not identified (i.e., the exact solution to Problem 4.1 was not obtained).

By applying the bisimulation algorithm to T we compute the bisimulation $\approx$ and construct the bisimulation quotient $T/_\approx$ (Fig. 4.5b with equivalence classes shown in Fig. 4.5a). Using Algorithm 3, we identify the largest satisfying and strictly violating regions of $T/_\approx$ as $X_{T/\approx}^{\phi} = \{X_{1.1}, X_{1.2}, X_{2.1}, X_3, X_4\}$ and $X_{T/\approx}^{\neg\phi} = \{X_{2.3}, X_{2.4}, X_5\}$, respectively. Following from the discussion from Sect. 1.3, we can guarantee that regions $con(X_{T/\approx}^{\phi}) = \{x_1, x_2, x_3, x_4, x_{10}, x_{11}, x_{12}\}$ and $con(X_{T/\approx}^{\neg\phi}) = \{x_5, x_6, x_9, x_{13}, x_{14}\}$ are respectively the largest satisfying and strictly violating region in T (see Fig. 4.5c). In addition, we can compute the region of uncertain states $X_{T/\approx}^{\phi ?} = X/_\sim \setminus (X_{T/\approx}^{\phi} \cup X_{T/\approx}^{\neg\phi}) = \{X_{2.2}\}$ of $T/_\approx$ and guarantee that all states from the region $con(X_{T/\approx}^{\phi ?}) = \{x_7, x_8\}$ are uncertain in T.



(a) Equivalence classes $X/_{\approx}$.

(b) Satisfying and violating states of the bisimulation quotient $T/_{\approx}$.

(c) Satisfying and violating states of T.

Fig. 4.5 The bisimulation $\approx$ (**a**) of system T from Fig. 1.11a is obtained using Algorithm 1 by refining the equivalence classes $X/_{\sim}$ highlighted in Fig. 1.11a (the quotient $T/_{\sim}$ was shown in Fig. 1.11b). This allows the construction of the bisimulation quotient $T/_{\approx}$ (**b**). The largest satisfying (*light gray*), strictly violating (*dark gray*) and uncertain (*white*) regions of $T/_{\approx}$ for specification $\square\lozenge o_3$ can then be identified using Algorithm 3. This allows for the computation of the largest satisfying, strictly violating and uncertain regions of T (**c**). For additional details, see Example 4.5

4.3 Iterative Strategies

In Sect. 4.1, we presented a model-checking-based strategy (Algorithm 3), which could be applied directly on T to provide a complete solution to Problem 4.1. However, this approach was computationally expensive when T included a large number of states and could not be used directly when T was infinite as will be discussed in subsequent chapters. Algorithm 3 could also be applied to the quotient $T/_{\sim}$ (constructed with the observational equivalence relation $\sim$) but in general this led only to an approximate solution to Problem 4.1, which was too conservative as illustrated in Sect. 4.2. Finally, Algorithm 3 could also be applied to the bisimulation quotient $T/_{\approx}$, thus providing a complete solution to Problem 4.1 as discussed in the previous section. However, such an analysis strategy required that the bisimulation algorithm (Algorithm 1) has terminated before applying model-checking techniques. While this approach guarantees that the largest satisfying region of the system is identified, it is not practical for certain systems (e.g., when many refinement steps are required to compute a bisimulation quotient) and cannot be applied directly to infinite systems, which will be our focus in subsequent chapters. As a compromise, instead of relying on the termination of the bisimulation algorithm, the equivalence relation produced at each step of Algorithm 1 allows the construction of simulation quotients. Applying the analysis procedure (Algorithm 3) to these quotients leads to approximate solutions to Problem 4.1, which get increasingly better with subsequent iterations of Algorithm 1.

Formally, the equivalence relation $\sim_r$, initialized with the observational equivalence relation $\sim$ in Algorithm 1 and refined within the main loop of the algorithm, is used to construct quotient $T/_{\sim_r}$ at each step. Computing the largest satisfying and strictly violating regions $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$ of $T/_{\sim_r}$ through Algorithm 3 allows the computation of the subsets $con(X^{\phi}_{T/_{\sim_r}})$ and $con(X^{\neg\phi}_{T/_{\sim_r}})$ of the largest satisfying and strictly violating region of T. While the solution to Problem 4.1 obtained this way is not exact, a major advantage of the procedure is that an initial, rough approximation is obtained at little computational cost during the initial iterations and is improved through additional refinement (if more computational resources are available).

Let $\sim_1$ and $\sim_2$ denote the equivalence relation $\sim_r$ obtained during successive iterations of Algorithm 1 and consider the quotients $T/_{\sim_1} = (X/_{\sim_1}, \delta_{\sim_1}, O, o_{\sim_1})$ and $T/_{\sim_2} = (X/_{\sim_2}, \delta_{\sim_2}, O, o_{\sim_2})$. For any state $X_i \in X/_{\sim_2}, con(X_i) \subseteq con(X_j)$ for some $X_j \in X/_{\sim_1}$ (i.e., the equivalence classes of $X/_{\sim_1}$ are refined in $X/_{\sim_2}$). Given states $X_1, X_2 \in X/_{\sim_2}$ such that $con(X_1) \cup con(X_2) = con(X_i)$ for some $X_i \in X/_{\sim_1}$ (i.e., equivalence class X_i was refined into X_1 and X_2)

i. $o_{\sim_2}(X_1) = o_{\sim_2}(X_2) = o_{\sim_1}(X_i)$ (i.e., all subsets of a refined equivalence class inherit the observation of the parent) and
ii. for any $X_j \in X/_{\sim_2}$ such that $X_j \in \delta_{\sim_2}(X_1)$ or $X_j \in \delta_{\sim_2}(X_2)$, there exist an $X_k \in X/_{\sim_1}$ such that $con(X_j) \subseteq con(X_k)$ and $X_k \in \delta_{\sim_1}(X_i)$ (i.e., all transitions are preserved).

Since words can only be removed from the language of the quotient through refinement

$$\mathscr{L}_T \subseteq \mathscr{L}_{T/_{\sim_2}} \subseteq \mathscr{L}_{T/_{\sim_1}}. \tag{4.9}$$

and, from Eq. (4.9),

$$con(X^{\phi}_{T/_{\sim_1}}) \subseteq con(X^{\phi}_{T/_{\sim_2}}) \subseteq X^{\phi}_{T}. \tag{4.10}$$

Therefore, the largest satisfying region $X^{\phi}_{T/_{\sim_r}}$ of the quotient $T/_{\sim_r}$ obtained after more iterations of Algorithm 1 provides a better approximation of the largest satisfying region X^{ϕ}_{T} of T than the one obtained with less iterations. The same result holds for the largest violating region $X^{\neg\phi}_{T/_{\sim_r}}$.

The analysis strategy described in this section is summarized as Algorithm 4. While this illustrates the combination of quotient refinement and model checking, performing analysis after every single refinement step is prohibitive. To develop a solution more amenable to practical implementations, in the following section we present a number of improvements and optimizations to this method.

Algorithm 4 $[X^{\phi}, X^{\neg\phi}]$ =ITERATIVEANALYSIS(T, ϕ): Given formula ϕ, compute the (under-approximations) $X^{\phi} \subseteq X_T^{\phi}$ and $X^{\neg\phi} \subseteq X_T^{\neg\phi}$.

```
Initialize $\sim_r := \sim$
while there exist equivalence classes $X_i, X_j \in X/_{\sim_r}$ such that
            $\emptyset \subset con(X_i) \cap Pre(con(X_j)) \subset con(X_i)$ do
    Construct equivalence class $X_1$ such that $con(X_1) := con(X_i) \cap Pre(con(X_j))$
    Construct equivalence class $X_2$ such that $con(X_2) := con(X_i) \setminus Pre(con(X_j))$
    $X/_{\sim_r} := X/_{\sim_r} \setminus \{X_i\} \bigcup \{X_1, X_2\}$
    Construct quotient $T/_{\sim_r}$
    $X^{\phi}_{T/_{\sim_r}}$ :=ANALYZE$(T/_{\sim_r}, X/_{\sim_r}, \phi)$
    $X^{\neg\phi}_{T/_{\sim_r}}$ :=ANALYZE$(T/_{\sim_r}, X/_{\sim_r}, \neg\phi)$
    $X^{\phi} := con(X^{\phi}_{T/_{\sim_r}})$
    $X^{\neg\phi} := con(X^{\neg\phi}_{T/_{\sim_r}})$
end while
```

4.4 Conservative Quotient Refinement

If the initial steps of the analysis procedure outlined in the previous section fail to produce an adequate approximation of the solution to Problem 4.1, computation might become challenging due to the possible explosion in the number of states of $T/_{\sim_r}$ as refinement progresses. Therefore, minimizing the amount of refinement performed at each step is critical to the feasibility of the method. One possible strategy for controlling the number of states produced during the refinement involves refining and analyzing the quotient only at states where this can improve the solution (i.e., increase $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$). Targeting our computation in such a way requires identifying states where refinement might be beneficial and developing a refinement strategy that operates locally on particular states. Through the rest of this section we discuss both issues and use the results to develop an improved analysis procedure.

To understand where refinement should be targeted in the quotient $T/_{\sim_r}$, we first provide some additional intuition about why analysis of $T/_{\sim_r}$ leads only to a conservative solution to Problem 4.1 and how state refinement can help. Given a state $x \in X$, system T either satisfies formula ϕ from x (i.e., $T(x) \vDash \phi$), its negation $\neg\phi$ (i.e., $T(x) \vDash \neg\phi$), or x is uncertain in T with respect to the satisfaction of ϕ (i.e., $T(x) \nvDash \phi$ and $T(x) \nvDash \neg\phi$). Given two states $x_1, x_2 \in X$ such that $T(x_1) \vDash \phi$ and $T(x_2) \vDash \neg\phi$, if the states are equivalent (i.e., $x_1 \sim_r x_2$) and belong to equivalence class $X_i \in X/_{\sim_r}$ (i.e., $x_1, x_2 \in con(X_i)$), state X_i is uncertain in $T/_{\sim_r}$ with respect to the satisfaction of ϕ. However, refining X_i might separate states x_1 and x_2 and allow the resulting subsets of X_i to be characterized as satisfying or violating.

Example 4.6 Consider analyzing system T from Fig. 1.12a with specification $\phi = \Diamond\Box o_2$. We focus on the equivalent states x_1, x_2 and x_3 with observation o_1. All runs of T originating at state x_1 satisfy ϕ and no runs originating at x_3 satisfy it (i.e., all runs originating there satisfy $\neg\phi$). Furthermore, state x_2 is uncertain and both runs satisfying ϕ and $\neg\phi$ originate there. In the quotient $T/_\sim$ (Fig. 1.13), $con(X_1) = \{x_1, x_2, x_3\}$ and state X_1 is uncertain. However, refinement of state X_1 can separate satisfying and violating states and allow the subsets of X_1 to be characterized. In the refined quotient $T/_{\sim_2}$ (Fig. 1.14d), which is also the bisimulation quotient $T/_\approx$ obtained through Algorithm 1, state $X_{1.1}$ is satisfying, state $X_{1.2.2}$ is violating and state $X_{1.2.1}$ is uncertain.

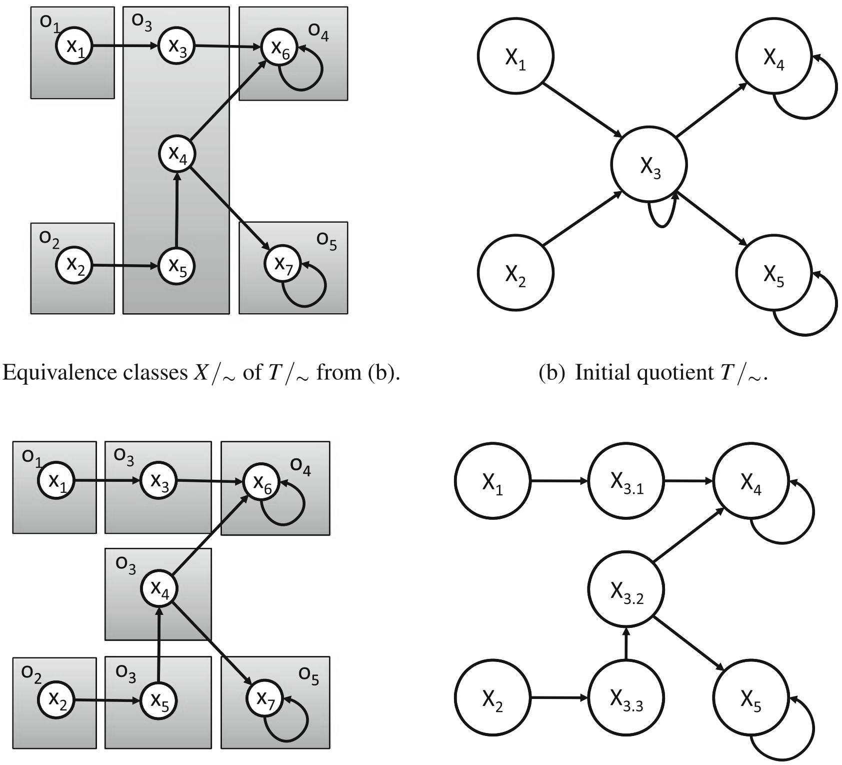
We hope to apply state refinement in order to separate the subsets of states of $T/_{\sim_r}$, from which only trajectories satisfying either ϕ or its negation $\neg\phi$ originate. Refinement of the quotient at any state does not change the satisfaction of the formula at state X_i, which has already been characterized (i.e., where $X_i \in X^{\phi}_{T/_{\sim_r}}$ or $X_i \in X^{\neg\phi}_{T/_{\sim_r}}$). More specifically, for any satisfying state of the quotient $X_i \in X^{\phi}_{T/_{\sim_r}}$ refinement of any other states can only "shrink" the language from X_i (as discussed previously), which guarantees the satisfaction of ϕ from X_i (the same result holds for violating states with the negation of the formula). Similarly, refining a state X_i can only "shrink" the language produced from states outside X_i.

For certain LTL formulas (e.g., $\Diamond\Box\phi$ and $\Box\Diamond\phi$, where ϕ is a logical expression over the atomic propositions) the refinement of previously characterized states does not influence the satisfaction of the formula by other, un-characterized states of the quotient. Therefore, once a state has been identified as satisfying the formula or its negation it is no longer considered for refinement or analysis. Formally, refinement of any state of the quotient $X_i \in X/_{\sim_r}$, where $X_i \in X^{\phi}_{T/_{\sim_r}}$ is unnecessary, since all trajectories originating there satisfy the formula. Similarly, refinement of any state $X_i \in X^{\neg\phi}_{T/_{\sim_r}}$ is also unnecessary, since only trajectories violating the formula originate there. The approximation of the solution obtained by analyzing $T/_{\sim_r}$ might be improved only by refining states, from which some but not all trajectories satisfy the formula. Therefore, state refinement is targeted to states $X_i \in X/_{\sim_r}$, such that $X_i \notin X^{\phi}_{\hat{T}/_{\sim_r}}$ and $X_i \notin X^{\neg\phi}_{\hat{T}/_{\sim_r}}$ or, equivalently, $X_i \in X^{\phi?}_{\hat{T}/_{\sim_r}}$. While this optimization is an additional source of conservatism for general formulas, this limitation is addressed through the improved procedure introduced in the following section.

In order to complete the analysis method described in this section, a refinement procedure is required that can be applied locally at particular states of a quotient, where this can improve the solution to Problem 4.1. Motivated by the fact that if $T/_{\sim_r}$ becomes the bisimulation quotient $T/_\approx$ through refinement, an exact solution is obtained by applying Algorithm 3, our refinement procedure REFINE() (Algorithm 5) is inspired by the bisimulation algorithm (Algorithm 1). Unlike the bisimulation algorithm, which refines the equivalence relation $\sim_r$ globally, REFINE($T/_{\sim_r}, X_i$) refines the quotient $T/_{\sim_r}$ locally at a state $X_i \in X/_{\sim_r}$. In particular, this procedure considers all successors of X_i in $T/_{\sim_r}$ and refines it in such a way that the bisimulation property (Theorem 1.1) is satisfied at X_i. Note that, this does not guarantee that X_i is not going to be refined further during subsequent iterations and, in fact, if any of the successors from $\delta_{\sim_r}(X_i)$ are refined, refinement of X_i might be necessary. However, since this procedure is only applied locally at a state, its termination is guaranteed.

Our refinement procedure allows us to target computation to specific states, while the quotient is updated (by updating the transitions $\delta_{\sim_r}$ and observation map $o_{\sim_r}$) instead of recomputed every time refinement is performed. When refinement is performed using the $Pre()$ operation as part of function REFINE() (Algorithm 5), outgoing transitions of the newly formed states are implicitly induced. Given states $X_i, X_j \in X/_{\sim_r}$ such that $X_j \in \delta_{\sim_r}(X_i)$, the subset $con(X_i) \cap Pre(con(X_j))$ always has a transition to state X_j. Additionally, any subset of $con(X_i) \setminus Pre(con(X_j))$ can never have a transition to state X_j. In the particular case when state X_i has a self transition ($X_i \in \delta_{\sim_r}(X_i)$), transitions from subset $con(X_i) \cap Pre(con(X_i))$ to all subsets of X_i resulting from its refinement are possible and must be recomputed using Eq. (1.17) or (1.18)—in Algorithm 5 this computation is performed by the code starting at line 11. Incoming transitions from all predecessor states $X_j \in Pre(X_i)$ to all newly formed states are updated by the loop starting at line 22 of Algorithm 5, which completes the construction of $\delta_{\sim_r}$. All subsets of a refined state inherit the observation of the parent and, therefore, $o_{\sim_r}$ is easily updated.

Example 4.7 To demonstrate the refinement procedure implemented by Algorithm 6, we apply it to state X_3 of the quotient $T/_{\sim}$ from Fig. 4.6b with equivalence classes shown in Fig. 4.6a. Only states X_3, X_4 and X_5 are reachable from X_3 and are considered when refining X_3, where $con(X_3) = \{x_3, x_4, x_5\}$. For the subsets $X_{3.1}$, $X_{3.2}$ and $X_{3.3}$, we have $con(X_{3.1}) = (con(X_3) \cap Pre(con(X_4))) \setminus (Pre(con(X_3)) \cup Pre(con(X_5))) = \{x_3\}$, $con(X_{3.2}) = (con(X_3) \cap Pre(con(X_4)) \cap Pre(con(X_5))$ and $con(X_{3.3}) = (con(X_3) \cap Pre(con(X_3))$. Transitions $\delta_{\sim_r}(X_{3.1}) = \{X_4\}$ and $\delta_{\sim_r}(X_{3.2}) = \{X_4, X_5\}$ are implicitly induced due to the refinement (lines 5–6 of Algorithm 5). While we can guarantee that there is a transition from state $X_{3.3}$ to a subset of X_3, additional computation (using Eqs. (1.17) or (1.18)) is required to find transition $\delta_{\sim_r}(X_{3.3}) = \{X_{3.2}\}$ (loop starting at line 11 of Algorithm 5). Incoming transitions from the predecessors X_1 and X_2 of state X_3 are also recomputed (loop starting at line 22 of Algorithm 5), resulting in the quotient $T/_{\sim_r} =$ REFINE($T/_{\sim}$, X_3) shown in Fig. 4.6d with equivalence classes shown in Fig. 4.6c.



(a) Equivalence classes $X/_{\sim}$ of $T/_{\sim}$ from (b).

(b) Initial quotient $T/_{\sim}$.

(c) Equivalence classes $X/_{\sim_r}$ of $T/_{\sim_r}$ from (d).

(d) Refined quotient $T/_{\sim_r}$.

Fig. 4.6 Refinement of quotient $T/_{\sim}$ (**b**) at state X_3 using function REFINE() (Algorithm 5) results in the construction of quotient $T/_{\sim_r} = \text{REFINE}(T/_{\sim}, X_3)$ (**c**). For additional details, see Example 4.7

For any states $X_i, X_j \in X/_{\sim}$, we can guarantee that the intersection $con(X_i) \cap Pre(con(X_j))$ is nonempty if and only X_j is reachable from X_i in $T/_{\sim}$. In other words,

$$X_j \in \delta_{\sim}(X_i) \Leftrightarrow con(X_i) \cap Pre(con(X_j)) \neq \emptyset. \tag{4.11}$$

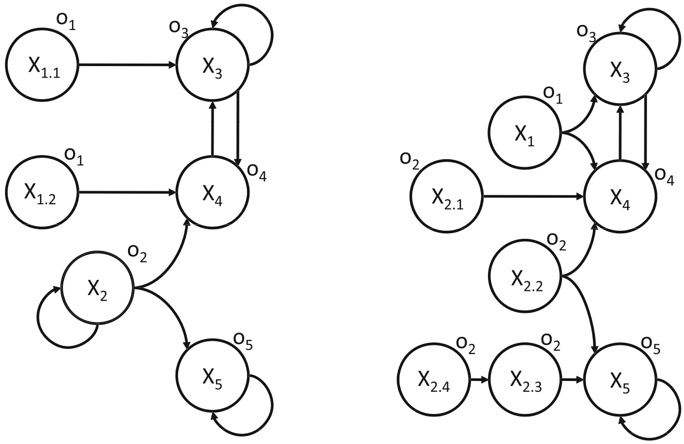
Then, all nonempty intersections $con(X_i) \cap_{X_j \in \mathbb{X}'} Pre(con(X_j)) \setminus \cup_{X_k \in \mathbb{X}''} Pre(con(X_k))$, where $\mathbb{X}' \in 2^{\delta_{\sim}(X_i)}$ and $\mathbb{X}'' = \delta_{\sim}(X_i) \setminus \mathbb{X}'$, provide a partition of X_i that satisfies the bisimulation property. Therefore, if $m = |\delta_{\sim}(X_i)|$ is the number of successors of X_i, applying REFINE($T/_{\sim}$, X_i) results in at most 2^m subsets.

When T is deterministic, given states $X_i, X_j, X_k \in X/_{\sim}$ such that X_j and X_k are reachable from X_i (i.e., $X_j, X_k \in \delta_{\sim}(X_i)$), we have $con(X_i) \cap Pre(con(X_j)) \cap Pre(con(X_k)) = \emptyset$. In other words, there are no states in $con(X_i)$ that can make a transition to both $con(X_j)$ and $con(X_k)$ in one step, which would make T nondeterministic. Then, $con(X_i) \cap Pre(con(X_j))$ of all successors $X_j \in \delta_{\sim}(X_i)$

Algorithm 5 $T/_{\sim_r}$ =REFINE($T/_{\sim}$, X_i): Refine $T/_{\sim}$ at state $X_i \in X/_{\sim}$

```
$\mathbb{X} := \{X_i\}$ and $\delta_{\sim_r}(X_i) = \emptyset$
for all $X_j \in \delta_{\sim}(X_i)$ do
    $\hat{\mathbb{X}} := \emptyset$
    for all $X_k \in \mathbb{X}$ do
        Construct state $X_1$ such that
            $con(X_1) = con(X_k) \cap Pre(con(X_j))$
            $o_{\sim_r}(X_1) = o_{\sim}(X_i)$
            $\delta_{\sim_r}(X_1) = \delta_{\sim_r}(X_k) \cup X_j$
        Construct state $X_2$ such that
            $con(X_2) = con(X_k) \setminus Pre(con(X_j))$
            $o_{\sim_r}(X_2) = o_{\sim}(X_i)$
            $\delta_{\sim_r}(X_2) = \delta_{\sim_r}(X_k)$
        $\hat{\mathbb{X}} := \hat{\mathbb{X}} \cup \{X_1, X_2\}$
    end for
    $\mathbb{X} := \hat{\mathbb{X}}$
end for
for all $X_j \in \mathbb{X}$ such that $X_i \in \delta_{\sim_r}(X_j)$ do
    for all $X_k \in \mathbb{X}$ do
        if $Post(con(X_j)) \cap con(X_k) \neq \emptyset$ then
            $\delta_{\sim_r}(X_j) := (\delta_{\sim_r}(X_j) \setminus \{X_i\}) \cup \{X_k\}$
        end if
    end for
end for
for all $X_j \in X/_{\sim} \setminus \{X_i\}$ do
    $o_{\sim_r}(X_j) := o_{\sim}(X_j)$
    if $X_i \in \delta_{\sim}(X_j)$ then
        $\delta_{\sim_r}(X_j) := \delta_{\sim}(X_j) \setminus \{X_i\}$
        for all $X_k \in \mathbb{X}$ do
            if $Post(con(X_j)) \cap con(X_k) \neq \emptyset$ then
                $\delta_{\sim_r}(X_j) := \delta_{\sim_r}(X_j) \cup \{X_k\}$
            end if
        end for
    else
        $\delta_{\sim_r}(X_j) := \delta_{\sim}(X_j)$
    end if
end for
$X/_{\sim_r} := (X/_{\sim} \setminus \{X_i\}) \cup \mathbb{X}$
return $T/_{\sim_r} = (X/_{\sim_r}, \delta_{\sim_r}, O, o_{\sim_r})$
```

provide a partition of state X_i, satisfying the bisimulation property and applying REFINE($T/_{\sim}$, X_i) on the quotient of a deterministic system T results in at most $m = |\delta_{\sim}(X_i)|$ subsets. For a deterministic T, REFINE($T/_{\sim}$, X_i) is more efficient and is summarized as Algorithm 6.



(a) Quotient $T/_{\sim_1}$ =REFINE($T/_\sim$,X_1). (b) Quotient $T/_{\sim_2}$ =REFINE($T/_\sim$,X_2).

Fig. 4.7 The quotient $T/_\sim$ (Fig. 1.11b) of transition system T (Fig. 1.11a) is refined at states X_1 or X_2. For additional information, see Example 4.8

Algorithm 6 $T/_{\sim_r}$ =REFINE($T/_\sim$, X_i): Refine $T/_\sim$ at state $X_i \in X/_\sim$ for a deterministic T

```
$X/_{\sim_r} := X/_\sim \setminus X_i$
for all $X_j \in X/_{\sim_r}$ do
    $\delta_{\sim_r}(X_j) := \delta_\sim(X_j) \setminus X_i$
    $o_{\sim_r}(X_j) = o_\sim(X_j)$
end for
for all $X_j \in \delta_\sim(X_i)$ do
    $X/_{\sim_r} := X/_{\sim_r} \cup X_k$, where $con(X_k) = con(X_i) \cap Pre(con(X_j))$
    $\delta_{\sim_r}(X_k) := \{X_j\}$ and $o_{\sim_r}(X_k) := o(X_i)$
    for all $X_l \in X/_{\sim_r}$ such that $X_i \in \delta_\sim(X_l)$ do
        if $Post(con(X_l)) \cap con(X_k) \neq \emptyset$ then
            $\delta_{\sim_r}(X_l) := \delta_{\sim_r}(X_l) \cup \{X_k\}$
        end if
    end for
end for
return $T/_{\sim_r} = (X/_{\sim_r}, \delta_{\sim_r}, O, o_{\sim_r})$
```

Example 4.8 Consider the quotient $T/_\sim$ (Fig. 1.11b) of transition system T (Fig. 1.11a). All states of T from the equivalence class $con(X_1) = \{x_1, x_2, x_3\}$ have deterministic transitions only. Therefore, refining $T/_\sim$ at state X_3 demonstrates the refinement procedure implemented by Algorithm 6 and results in the refined quotient $T/_{\sim_1}$ =REFINE($T/_\sim, X_1$). $T/_{\sim_1}$ includes the subsets $X_{1.1}$ and $X_{1.2}$ instead of X_1, where $con(X_{1.1}) = con(X_1) \cap Pre(con(X_3)) = \{x_1\}$ and $con(X_{1.2}) = con(X_1) \cap Pre(con(X_4)) = \{x_2, x_3\}$ (see Fig. 4.7a).

Since there exist states of T from the equivalence class $con(X_2) = \{x_4, \ldots, x_9\}$ with nondeterministic transitions, the more general procedure implemented by Algorithm 6 must be applied to refine $T/_\sim$ at state X_2. As a result, the refined quotient $T/_{\sim_2}$ =REFINE($T/_\sim, X_2$) is obtained, where subsets $X_{2.1}$, $X_{2.2}$, $X_{2.3}$ and $X_{2.4}$ are included instead of X_2. These subsets are computed as $con(X_{2.1}) = con(X_2) \cap Pre(X_4) \setminus Pre(X_5) = \{x_4\}$, $con(X_{2.2}) = con(X_2) \cap Pre(X_4) \cap Pre(X_5) = \{x_7, x_8\}$, $con(X_{2.3}) = con(X_2) \cap Pre(X_5) \setminus Pre(X_4) = \{x_9\}$ and $con(X_{2.4}) = con(X_2) \cap Pre(X_2) = \{x_5, x_6\}$ (see Fig. 4.7a or the equivalence classes shown in Fig. 4.8b).

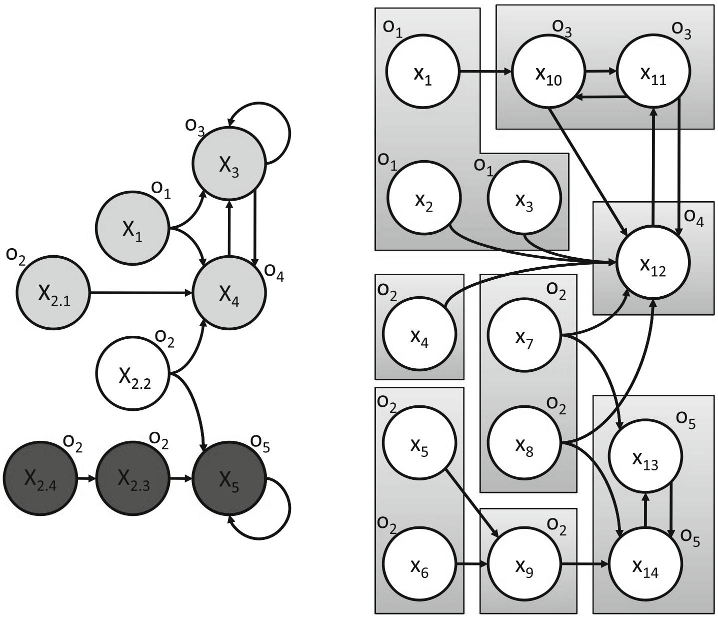
The analysis approach described in this section is summarized as Algorithm 7. The overall goal of limiting the size of the quotient $T/_{\sim_r}$ during the execution of the algorithm is achieved by targeting specific states, with the hope of improving the solution by expanding $X^{\phi}_{\hat{T}/_{\sim_r}}$ and $X^{\neg\phi}_{\hat{T}/_{\sim_r}}$. Besides the operations discussed in Sect. 1.3, this procedure does not require any additional computation. In the following, we consider the conditions guaranteeing that the quotient obtained through Algorithm 7 provides an exact solution to Problem 4.1.

Proposition 4.1 *Given transition system T with quotient $T/_{\sim_r}$, an exact solution to Problem 4.1 is obtained whenever all states of $T/_{\sim_r}$ can be partitioned between the largest satisfying and strictly violating regions $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$. In other words,*

$$X^{\phi}_{T/_{\sim_r}} \cup X^{\neg\phi}_{T/_{\sim_r}} = X/_{\sim_r} \Rightarrow con(X^{\phi}_{T/_{\sim_r}}) = X^{\phi}_T \text{ and } con(X^{\neg\phi}_{T/_{\sim_r}}) = X^{\neg\phi}_T \tag{4.12}$$

Proof By definition, all runs of T originating at a state $x \in con(X^{\phi}_{T/_{\sim_r}})$ satisfy ϕ and, therefore, $con(X^{\phi}_{T/_{\sim_r}})$ is a satisfying region of T. For any state $x \notin con(X^{\phi}_{T/_{\sim_r}})$, we have $x \in con(X_i)$ for some equivalence class $X_i \in X/_{\sim_r}$, where $X_i \notin X^{\phi}_{T/_{\sim_r}}$. Since $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$ partition the set $X/_{\sim_r}$, we can guarantee that $X_i \in X^{\neg\phi}_{T/_{\sim_r}}$ and, therefore, all runs of T originating at x violate ϕ, making $con(X^{\phi}_{T/_{\sim_r}})$ the largest satisfying region of T. The same argument can be made for the strictly violating region $con(X^{\neg\phi}_{T/_{\sim_r}})$ of T. ■

For a nondeterministic transition system T, it might be infeasible to partition the states $X/_{\sim_r}$ into $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$, due to the existence of uncertain states $x \in X^{\phi?}_T$ of



(a) Largest satisfying (light gray) and strictly violating (dark gray) regions in $T/_{\sim_r}$.

(b) Equivalence classes of T after quotient refinement.

Fig. 4.8 Iterative analysis and refinement of the quotient $T/_{\sim_r}$ reveals the largest satisfying and strictly violating regions $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$ (**a**) and allows the computation of a satisfying and violating regions in T (**b**). See Example 4.9 for additional details

T (see Definition 4.4). However, for a deterministic T the result from Proposition 4.1 becomes stronger:

Proposition 4.2 *Given a deterministic transition system T with quotient $T/_{\sim_r}$, an exact solution to Problem 4.1 is obtained only when all states of $T/_{\sim_r}$ can be partitioned between the largest satisfying and strictly violating regions $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$. In other words,*

$$X^{\phi}_{T/_{\sim_r}} \cup X^{\neg\phi}_{T/_{\sim_r}} = X/_{\sim_r} \Leftrightarrow con(X^{\phi}_{T/_{\sim_r}}) = X^{\phi}_{T} \text{ and } con(X^{\neg\phi}_{T/_{\sim_r}}) = X^{\neg\phi}_{T} \tag{4.13}$$

Proof The sufficiency of this proposition follows from Proposition 4.1. For necessity, assume by contradiction that $con(X^{\phi}_{T/_{\sim_r}}) = X^{\phi}_{T}$ and $con(X^{\neg\phi}_{T/_{\sim_r}}) = X^{\phi}_{T}$ but $X^{\phi}_{T/_{\sim_r}} \cup X^{\neg\phi}_{T/_{\sim_r}} \subset X/_{\sim_r}$. Then, there exists an equivalence class $X_i \in X/_{\sim_r}$ such that $X_i \notin X^{\phi}_{T/_{\sim_r}}$ and $X_i \notin X^{\neg\phi}_{T/_{\sim_r}}$. There must exist runs satisfying ϕ and runs satisfying $\neg\phi$ originating at states from $con(X_i)$ in T but, since no states from $con(X_i)$ can be included in either X^{ϕ}_{T} or $X^{\neg\phi}_{T}$ (i.e., $con(X^{\phi}_{T/_{\sim_r}})$ and $con(X^{\phi}_{T/_{\sim_r}})$ are maximal) both types of runs must originate at every state $x \in con(X_i)$. This implies that T is nondeterministic and contradicts the hypothesis. ■

Algorithm 7 $[X_T^{\phi},\ X_T^{\neg\phi}] = \textsc{TargetedAnalysis}(T, \phi)$: Given an LTL formula ϕ and transition system T, compute the largest satisfying and strictly violating regions of T.

```
Construct $T/_\sim$
$T/_{\sim_r} := T/_\sim$
$\mathbb{X}_r := X/_{\sim_r}$
repeat
  $X^{\phi}_{T/_{\sim_r}} := \textsc{analyze}(T/_{\sim_r}, \mathbb{X}_r, \phi)$
  $X^{\neg\phi}_{T/_{\sim_r}} := \textsc{analyze}(T/_{\sim_r}, \mathbb{X}_r, \neg\phi)$
  $\mathbb{X}_r := X/_{\sim_r} \setminus (X^{\phi}_{T/_{\sim_r}} \cup X^{\neg\phi}_{T/_{\sim_r}})$
  for all $X_i \in \mathbb{X}_r$ do
    $T/_{\sim_r} := \textsc{refine}(T/_{\sim_r}, X_i)$
  end for
until $\mathbb{X}_r = \emptyset$ or no states from $X/_{\sim_r}$ were refined
return $X_T^{\phi} = con(X^{\phi}_{T/_{\sim_r}})$, $X_T^{\neg\phi} = con(X^{\neg\phi}_{T/_{\sim_r}})$
```

Requiring that the set $\mathbb{X}_r$ becomes empty, which is one of the two possible termination conditions in Algorithm 7, guarantees that the states of $T/_\sim$ have been partitioned between sets $X^{\phi}_{T/_{\sim_r}}$ and $X^{\neg\phi}_{T/_{\sim_r}}$. From Propositions 4.1 and 4.2 this guarantees that an exact solution to Problem 4.1 was obtained. However, unless T is deterministic where, from Proposition 4.2, this condition is both necessary and sufficient, this requirement might be too strong. In fact, for a general (nondeterministic) system T it might be impossible to make $\mathbb{X}_r$ empty, due to the existence of states $X_i \in X/_{\sim_r}$ such that, for all states $x \in con(X_i)$ of T, there exist both runs satisfying ϕ and $\neg\phi$ originating in x. The possibility of such uncertain states necessitates the second termination condition in Algorithm 7 (no states from $X/_{\sim_r}$ are refined in an iteration). Based on our previous discussion, once Algorithm 7 has terminated, the set of uncertain states of T can be computed as $X_T^{\phi?} = con(X^{\phi?}_{T/_{\sim_r}})$, where $X^{\phi?}_{T/_{\sim_r}} = X/_{\sim_r} \setminus (X^{\phi}_{T/_{\sim_r}} \cup X^{\neg\phi}_{T/_{\sim_r}})$ is the set of uncertain states of $T/_{\sim_r}$. Such uncertain states will be considered in more detail in Sect. 4.5, where a computational procedure for their explicit identification during the analysis will be proposed.

Even when both termination conditions are considered, a large number of iterations might be required for Algorithm 7 to terminate when it is applied to large finite systems, while its termination cannot be guaranteed in general for the infinite systems that will be discussed in subsequent chapters. To address this issue, a limit on the number of iterations performed by the algorithm can be imposed or states of the quotient that become "too small" might not be considered for further refinement by excluding them from $\mathbb{X}_r$. Reaching such termination conditions leads to an approximate solution to Problem 4.1 but further refinement of any state $X_i \notin \mathbb{X}_r$ might not expand $X^{\phi}_{T/\sim_r}$ significantly since X_i is satisfying (i.e., $X_i \in X^{\phi}_{T/\sim_r}$), violating (i.e., $X_i \in X^{\neg\phi}_{T/\sim_r}$), uncertain (i.e., $X_i \in X^{\phi?}_{T/\sim_r}$), or small.

Even when no additional termination conditions are imposed and Algorithm 7 returns an exact solution to Problem 4.1, the quotient $T/_{\sim_r}$ is, in general, not a bisimulation quotient (e.g., see Example 4.9). This motivates us to explore the conditions required to guarantee that an exact solution to Problem 4.1 is obtained in the following section and to use them later to develop a more efficient analysis strategy.

Example 4.9 Using the procedure from Algorithm 7, we analyze transition system T introduced originally in Example 1.10 (Fig. 1.11a).

In Example 1.10, we described the construction of the simulation quotient $T/_{\sim}$ of T (Fig. 1.11b). However, in Example 4.4, we showed that simply analyzing $T/_{\sim}$ to compute its largest satisfying and strictly violating regions $X^{\phi}_{T/\sim}$ and $X^{\neg\phi}_{T/\sim}$ (Fig. 4.4a) leads only to a conservative solution to Problem 4.1 and the computation of a subset of the largest satisfying and strictly violating regions X^{ϕ}_{T} and $X^{\neg\phi}_{T}$ of T (Fig. 4.4b).

In Example 4.5, we considered an approach where the bisimulation algorithm (Algorithm 1) was used to obtain a quotient $T/_{\approx}$, bisimilar with T Fig. 4.5b with refined equivalence classes shown in Fig. 4.5a. By computing the largest satisfying and strictly violating regions of this bisimulation quotient, we were able to obtain the corresponding regions for T Fig. 4.5c (i.e., the solution to Problem 4.1 was exact). Furthermore, this allowed us to compute the uncertain region $X^{\phi?}_{T/\approx}$ of $T/_{\approx}$ and use it to obtain the uncertain region $X^{\phi?}_{T}$ of T.

By applying the procedure implemented in Algorithm 7, we can also obtain an exact solution to Problem 4.1, without constructing a bisimulation quotient. Indeed, after the initial quotient $T/_{\sim}$ is constructed and analyzed as in Example 4.4, the algorithm targets refinement to state $X_2 \in X/_{\sim}$ only (see Fig. 4.4a). Applying the refinement procedure to that state as described in Example 4.8 leads to the construction of quotient $T/_{\sim_r}$ (Fig. 4.5b). While $T/_{\sim_r}$ is not bisimilar with T, computing regions $X^{\phi}_{T/\sim_r}$ and $X^{\neg\phi}_{T/\sim_r}$ (Fig. 4.8a with equivalence classes shown in Fig. 4.8b) provides an exact solution to Problem 4.1.

4.5 Formula-Equivalence

So far in this chapter, we described methods for analyzing transition systems based on the construction of finite quotients, which led to the analysis procedure summarized as Algorithm 7. While our discussion focused primarily on the analysis of potentially large, finite transition systems, the construction of finite quotients also makes these approaches suitable for infinite systems—such applications will be considered in detail in subsequent chapters. Algorithm 7 combined state refinement inspired by the bisimulation algorithm (Algorithm 1) with model-checking-based analysis (Algorithm 3) in an iterative procedure. With a limited number of iterations, the algorithm was conservative and led to the computation of under-approximations of the largest satisfying and strictly violating regions (Definitions 4.1 and 4.3), which could be improved by performing additional iterations. When Algorithm 7 terminates, there could be states left that have not been assigned as either satisfying the specification or its negation and might be "too small" to undergo additional refinement. Even without a limit on the number of iteration or the size of regions that could undergo refinement the termination of the algorithm with an exact solution could not be guaranteed in general (e.g. when applied to an infinite system).

While the solution to Problem 4.1 returned by Algorithm 7 was, in general, only an approximation, we also encountered cases where an exact solution was obtained. In Sect. 1.3, we showed that the construction of a bisimulation quotient during refinement guarantees our solution was exact but Example 4.9 suggested that this condition was unnecessarily strong (i.e., bisimulation is a sufficient but not a necessary condition). The conditions from Propositions 4.1 and 4.2 could also guarantee that an exact solution was obtained but were too conservative unless only deterministic systems were considered. In the following, we explore more general conditions guaranteeing that an exact solution to Problem 4.1 is obtained (Eq. (4.8) holds). Specifically, we seek necessary and sufficient conditions weaker than bisimulation, guaranteeing that two transition systems (or a transition system and its quotient) are equivalent with respect to the satisfaction of a given LTL formula ϕ.

In Propositions 4.1 and 4.2 we described conditions guaranteeing that the largest satisfying and strictly violating regions of T were obtained by analyzing the quotient $T/_{\sim_r}$ (computed as part of Algorithm 7), even when the two systems are not bisimilar. While these conditions could be used to test whether such a solution to Problem 4.1 was exact, the approach could be applied only in specific cases (i.e., when T was deterministic). In the following, we generalize these results and formulate the conditions required to guarantee that two systems are equivalent with respect to the satisfaction of a specific LTL formula.

Definition 4.5 (*Formula equivalence*) Given a finite transition system T and an LTL formula ϕ, an observational equivalence relation $\sim$ is a ϕ-equivalence of T if and only if, for all states $x_1, x_2 \in X$ such that $x_1 \sim x_2$, we have

$$T(x_1) \vDash \phi \Leftrightarrow T(x_2) \vDash \phi$$

We denote a ϕ-equivalence relation as $\sim_\phi$ and refer to the quotient $T/_{\sim_\phi}$ as ϕ-equivalent quotient.

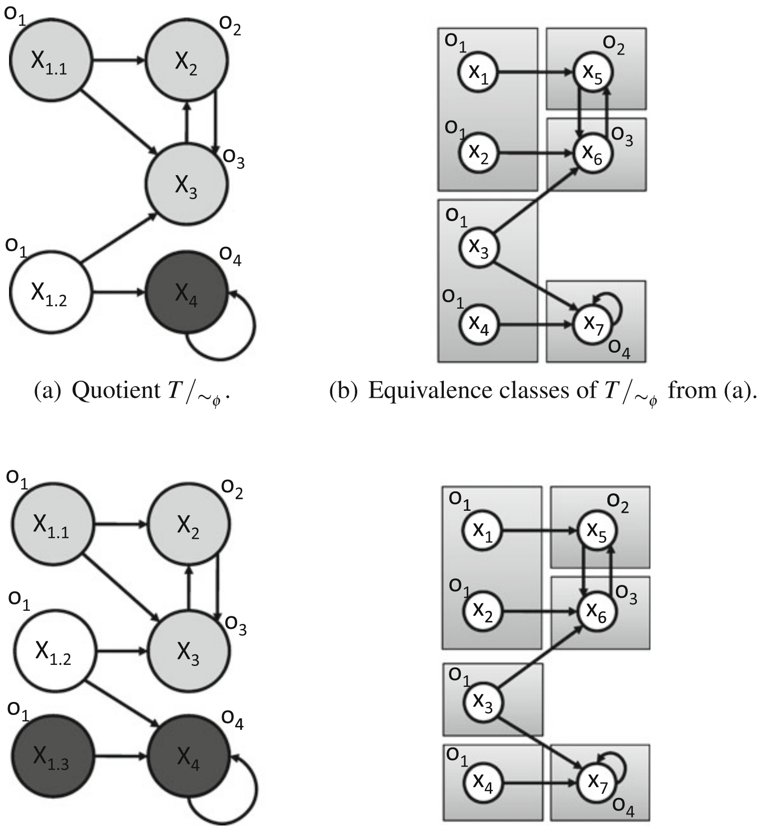
From Eq. (4.7) it follows that a bisimulation relation $\sim$ is a ϕ-equivalence for all LTL formulas ϕ. Bisimulation is a sufficient condition guaranteeing that Eq. (4.8) holds but since we are interested in the analysis of T for a specific LTL formula ϕ it can be too restrictive.

Proposition 4.3 *Given a transition system T and an LTL formula ϕ, the equation $X_T^\phi = con(X_{T/\sim}^\phi)$ is satisfied and an exact solution to Problem 4.1 is obtained by analyzing the quotient $T/_\sim$ if and only if $\sim$ is a ϕ-equivalence of T.*

Proof Assume that $\sim$ is a ϕ-equivalence. From Definition 4.5 it follows that $\forall x \in X$ such that $T(x) \vDash \phi, x \in con(X_i), X_i \in T/_\sim$ we have $T/_\sim(X_i) \vDash \phi$. Then, $\forall x \in X, x \in X_T^\phi \Leftrightarrow x \in con(X_{T/\sim}^\phi)$ and therefore $X_T^\phi = con(X_{T/\sim}^\phi)$. Assume that $\sim$ is not a ϕ-equivalence. Then, $\exists x_1, x_2 \in X$ such that $x_1 \sim x_2$, $T(x_1) \vDash \phi$ and $T(x_2) \nvDash \phi$. Considering the equivalence class $X_i \in X/_\sim$ such that $x_1, x_2 \in con(X_i)$ we have $T/_\sim(X_i) \nvDash \phi$. Then $x_1 \in X_T^\phi$ but $x_1 \notin con(X_{T/\sim}^\phi)$ and therefore $X_T^\phi \neq con(X_{T/\sim}^\phi)$. ■

Proposition 4.3 shows that ϕ-equivalence is a necessary and sufficient condition guaranteeing that the largest satisfying region of T can be computed through the computation of the largest satisfying region of the quotient $T/_{\sim_\phi}$ (i.e., $X_T^\phi = con(X_{T/\sim_\phi}^\phi)$). The condition $X_T^\phi = con(X_{T/\sim}^\phi)$ is similar to the one in Eq. (4.8) (left-hand side) but is formulated for a quotient $T/_\sim$, rather than the bisimulation quotient $T/_\approx$. Then, obtaining an exact solution to Problem 4.1 reduces to the computation of $con(X_{T/\sim_\phi}^\phi)$, where $T/_{\sim_\phi}$ is a finite, ϕ-equivalent quotient for T (e.g., see Fig. 4.9a, b). As previously discussed, in order to obtain more information about T, the strictly violating region of T might also be computed as $X_T^{\neg\phi} = con(X_{T/\sim}^{\neg\phi})$, which reduces to the construction and analysis of the $\neg\phi$-equivalent quotient for T. More generally, to guarantee the exact computation of both the largest satisfying and strictly violating regions of T (Eq. (4.8)), a quotient that is both ϕ-equivalent and $\neg\phi$-equivalent must be computed. We denote such a quotient as $T/_{\approx_\phi}$ (e.g., see Fig. 4.9c, d). Constructing $T/_{\approx_\phi}$ also allows the computation of the uncertain region $X_T^{\phi?} = con(X_{T/\approx_\phi}^{\phi?})$ of T through the computation of the uncertain region $X_{T/\approx_\phi}^{\phi?} = X/_{\approx_\phi} \setminus (X_{T/\approx}^{\neg\phi} \cup X_{T/\approx}^\phi)$ of $T/_{\approx_\phi}$.

Example 4.10 We are interested in analyzing the transition system T shown in Fig. 4.9b with specification $\phi = \Box\Diamond o_3$. The quotient $T/_{\sim_\phi}$ (shown in Fig. 4.9a with equivalence classes highlighted in Fig. 4.9b) of T is not a bisimulation quotient (the characterization from Theorem 1.1 is violated at state $X_{1.1}$). Even so, the quotient is ϕ-equivalent and therefore the largest satisfying region $X_T^\phi = \{x_1, x_2, x_5, x_6\}$ can be computed as $X_T^\phi = con(X_{T/\sim_\phi}^\phi)$, where $X_{T/\sim_\phi}^\phi = \{X_{1.1}, X_2, X_3\}$. However, region $X_{T/\sim_\phi}^{\neg\phi} = \{X_4\}$ provides only an under-approximation $con(X_{T/\sim}^{\neg\phi}) = \{x_7\}$ of the strictly violating $X_T^{\neg\phi} = \{x_4, x_7\}$ of T.



(a) Quotient $T/_{\sim_\phi}$.

(b) Equivalence classes of $T/_{\sim_\phi}$ from (a).

(c) Quotient $T/_{\approx_\phi}$.

(d) Equivalence classes of $T/_{\approx_\phi}$ from (c).

Fig. 4.9 Given LTL formula $\phi = \Box\Diamond o_3$, analysis of the ϕ-equivalent quotient $T/_{\sim_\phi}$ (**a**) of a transition system T allows the computation of the largest satisfying region (*light gray*) but only a subset of the strictly violating region (*dark gray*) of T. Analysis of the quotient $T/_{\approx_\phi}$ (**c**) that is both ϕ-equivalent and $\neg\phi$-equivalent allows the computation of both the largest satisfying (*light gray*) and strictly violating (*dark gray*) regions of T. For additional details, see Example 4.10

While quotient $T/_{\approx_\phi}$ (shown in Fig. 4.9c with equivalence classes highlighted in Fig. 4.9d) is still not a bisimulation quotient, it is both a ϕ-equivalent and an $\neg\phi$-equivalent of T. This allows the exact computation of the strictly violating region $X_T^{\neg\phi} = con(X_{T/_{\approx_\phi}}^{\neg\phi}) = \{x_4, x_7\}$ through the computation of region $X_{T/_{\approx_\phi}}^{\neg\phi} = \{X_{1.3}, X_4\}$. Constructing quotient $T/_{\approx_\phi}$ also leads to the computation of the uncertain region $X_T^{\phi?} = con(X_{T/_{\approx_\phi}}^{\phi?}) = \{x_3\}$ of T, where $X_{T/_{\approx_\phi}}^{\phi?} = X/_{\approx_\phi} \setminus (X_{T/_{\approx_\phi}}^{\neg\phi} \cup X_{T/_{\approx_\phi}}^{\phi}) = \{X_{1.2}\}$.

Motivated by the strategy for solving Problem 4.1 outlined above, in the following we develop a procedure for the computation of formula equivalent quotients. While the approach we described in Sect. 4.3 could, in principle, also lead to the construction of formula equivalent quotients through iterative analysis and refinement (as was the case for the system from Example 4.9), a large number of steps was required to achieve this. Furthermore, the optimizations introduced to control the number of states produced through refinement as part of this method resulted in an additional source of conservatism for general formulas. The method we develop next aims directly at the construction of formula equivalent quotients and is more efficient, while in addition, it overcomes some of the limitations of the previous procedure.

In the following, we develop an algorithm for the computation of ϕ-equivalent quotients of finite transitions systems, leveraging ideas from the bisimulation algorithm (Algorithm 1) and automata-based model checking. To simplify the presentation of our method, we assume that the LTL formula ϕ is translated into a deterministic Büchi automaton B_ϕ over the set of observations O—we discuss how this assumption is relaxed in Sect. 4.6.

Since the computation of ϕ-equivalent quotients is guided by formula ϕ, it is most natural to perform the computation in the product automaton $P = T/_\sim \otimes B_\phi$ (Definition 3.2), where both the structure of the system ($T/_\sim$) and the specification (B_ϕ) is captured. The computational techniques we employ in the following will be described in detail in Chap. 5 in the context of control transition systems. In this chapter we only give a brief overview, sufficient for the presentation of our analysis procedure.

Let $S_\top \subseteq S_P$ be the set of states of P, from which all runs are accepted (i.e., they visit a final state from the set F_P infinitely often). The set $S_\top$ can be computed efficiently using a method inspired by automata-theoretic model checking and games (to be discussed in detail in Chap. 5). We identify a subset $F_\top \subseteq F_P$ of accepting states of P from which infinitely many revisits to the set F_P are guaranteed. $S_\top$ is then a set of states from which a visit to $F_\top$ is guaranteed. The largest satisfying set $X^\phi_{T/_\sim}$ of $T/_\sim$ can be computed as the projection $\alpha(S_\top \cap S_{P0}) \subseteq X/_\sim$. Similarly, we can also identify a set of states $S_\perp \subseteq S_P$ of P from which no runs are accepting. The projection $\alpha(S_\perp \cap S_{P0}) \subseteq X/_\sim$ corresponds to $X^{\neg\phi}_{T/_\sim}$ (i.e., the largest set of states of $T/_\sim$ from which no runs satisfy ϕ).

The analysis approach we described in Sect. 4.3 required the construction of both the product automaton $P_\phi = T/_\sim \otimes B_\phi$ with the formula and $P_{\neg\phi} = T/_\sim \otimes B_{\neg\phi}$ with its negation in order to compute the largest satisfying and strictly violating regions $X^\phi_{T/_\sim}$ and $X^{\neg\phi}_{T/_\sim}$. Using the approach from this section, we avoid the construction of $P_{\neg\phi}$ and only perform computation on P_ϕ to find these sets.

Let $S_? = S_P \setminus (S_\top \cup S_\perp)$ be the subset of states, from which some but not all runs are accepting in P. The projection $X^{\phi?}_{T/_\sim} = \alpha(S_? \cap S_{P0}) \subseteq X/_\sim$ corresponds to the set of uncertain states of $T/_\sim$, where both runs satisfying ϕ and $\neg\phi$ originate. The ϕ-equivalence property (Definition 4.5) can only be violated at states from $X^{\phi?}_{T/_\sim}$. As previously discussed, when T is deterministic, each state $x \in X$ satisfies either ϕ or $\neg\phi$. Then, the existence of states in $X^{\phi?}_{T/_\sim}$ is only due to the construction of the abstraction $T/_{\sim}$ (e.g., for two equivalent states $x_1, x_2 \in con(X_i)$ for some $X_i \in X/_{\sim}$, if $x_1 \vDash \phi$ and $x_2 \vDash \neg\phi$ then $X_i \in X^{\phi ?}_{T/_{\sim}}$).

As in Sect. 4.3, we can attempt to separate satisfying and violating runs from a state in the set $S_?$ through refinement in order to separate the subsets of that state between sets $S_\top$ and $S_\perp$. Since the structure of P is completely determined by B_ϕ and $T/_{\sim}$ and B_ϕ is fixed and determined by the formula ϕ, states in P can only be refined through refinement of $T/_{\sim}$. We refine a state $(X_i, s) \in S_?$ by applying the procedure REFINE $(T/_{\sim}, \alpha(X_i, s))$ (Algorithm 5) Sect. 4.3.

To prevent unnecessary computation, changes made through refinement in the quotient $T/_{\sim}$ are projected to the product P by applying a function $P' =$ UPDATE $(P, T/_{\sim_r}, (X_i, s))$, rather than by recomputing it as $P' = T/_{\sim_r} \otimes B_\phi$. This further reduces the number of states in P' after refinement since, when a state X_i is refined in $T/_{\sim_r}$, not every state (X_i, s) is necessarily refined in P'. Due to the one-to-many correspondence between the states of the refined product P' and the refined quotient $T/_{\sim_r}$, after refinement the updated product automaton P' might include significantly fewer states than the recomputed product automaton $T/_{\sim_r} \otimes B_\phi$.

When T is nondeterministic, it is possible that for a state $X_i \in X/_{\sim}$, there exist both trajectories satisfying ϕ and $\neg\phi$ originating at all states $x \in con(X_i)$ of T. Then, state X_i is inherently uncertain in $T/_{\sim}$ and abstraction refinement can never separate satisfying and violating runs from X_i. We partition the set of states $S_?$ into subsets $S_\prec$ and $S_\div$, where states of $T/_{\sim}$ from the projection $\alpha(S_\prec \cap S_{P0})$ are inherently uncertain, while refinement should be targeted to states from $\alpha(S_\div \cap S_{P0})$. In the following, we provide a computational characterization of the states from $S_\prec$ (Proposition 4.4).

Proposition 4.4 *Given a state $(X_i, s) \in S_P$, we have $(X_i, s) \in S_\prec$ if and only if*

i. $(X_i, s) \in S_?$ (i.e., both satisfying and violating runs originate there),
ii. $\forall (X_j, s') \in \delta_P((X_i, s))$, $(X_j, s') \in S_\top$, $(X_j, s') \in S_\perp$ or $(X_j, s') \in S_\prec$ (i.e., all successors states of (X_i, s) have been characterized in P),
iii. $T/_{\sim}$ =REFINE$(T/_{\sim}, X_i)$ (i.e., state (X_i, s) cannot be refined further).

Proof All successors of (X_i, s) are characterized in P and, therefore, none of the successors of $X_i = \alpha((X_i, s))$ will be considered for further refinement in $T/_{\sim}$. Since applying REFINE$(T/_{\sim}, X_i)$ does not affect X_i, then for all states X_j such that $X_j \in \delta_{\sim}(X_i)$ we have $\forall x \in con(X_i)$, $\exists x' \in con(X_j)$ such that $x' \in \delta(x)$. Therefore, $(X_i, s) \in S_\prec$ and X_i is inherently uncertain. ■

Finally, in the following we derive the conditions guaranteeing that a ϕ-equivalent quotient has been computed based on the computation of sets $S_\top$, $S_\perp$, $S_\div$ and $S_\prec$ in P.

Proposition 4.5 *The equivalence relation $\sim$ is a ϕ-equivalence of T if and only if $S_\div \cap S_{P0} = \emptyset$.*

Proof For necessity, assume $S_\div \cap S_{P0} \neq \emptyset$ where (X_i, s) is a state of P such that $(X_i, s) \in S_\div$. Then, $X_i = \alpha((X_i, s))$ is a state of $T/_{\sim}$ such that $\exists x_1, x_2 \in con(X_i)$, where $T(x_1) \vDash \phi$ and $T(x_2) \vDash \neg\phi$ and, therefore, $\sim$ is not a ϕ-equivalence. For sufficiency, assume $S_\div \cap S_{P0} = \emptyset$. Then, $\forall X \in X/_{\sim}$ we have

i. $\forall x \in con(X_i), T(x) \vDash \phi$,
ii. $\forall x \in con(X_i), T(x) \vDash \neg\phi$ or
iii. $\forall x \in con(X_i), T(x) \nvDash \phi$ and $T(x) \nvDash \neg\phi$

and therefore $\sim$ is a ϕ-equivalence. ■

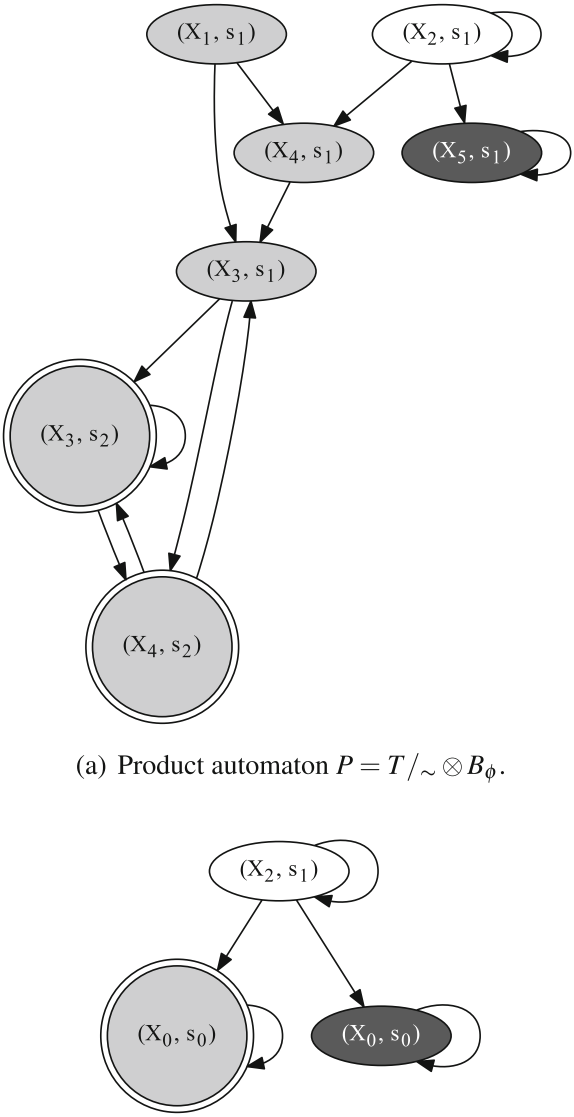
In general, the set $S_{\div}$ is nonempty but can be made empty if accepting and non-accepting runs from each state $(X_i, s) \in S_{\div}$ are separated through refinement as described above. Following from Proposition 4.5 and the discussion presented in Sect. 4.5, this provides a solution to Problem 4.1.

There are several additional optimizations that can be introduced in this analysis procedure. The product automaton P can be simplified by respectively replacing each set $S_\top$, $S_\bot$ and $S_\prec$ by a single dummy state s_1, s_2 or s_3, such that $\delta_P(s_1) = \{s_1\}$, $\delta_P(s_2) = \{s_2\}$, $\delta_P(s_3) = \{s_1, s_2\}$ and $s_1 \in F_P$. The motivation for this simplification comes from the fact that guaranteeing a visit to a state from (the previously identified) set $S_\top$ can be enforced from a state of P is equivalent to guaranteeing that the state belong to $S_\top$ and the same argument holds for set $S_\bot$. When simplifying P, for a state $s \in S_P \setminus (S_\top \cup S_\bot \cup S_\prec)$ a transition to a dummy state is included if a transition to a state from the corresponding set was present (e.g., if there existed a state $s' \in S_\top$ such that $s' \in \delta_P(s)$ then $s_1 \in \delta_P(s)$). This simplification is performed by the function $P' = \textsc{Simplify}(P)$, which reduces the number of states of P and leads to faster computation.

Example 4.11 We apply our formula-guided analysis procedure (Algorithm 8) in order to study transition system T, defined originally in Example 1.11 (Fig. 1.11a), against specification $\phi = \Box\Diamond o_3$.

i. The specification, given as the LTL formula ϕ, is translated into a deterministic Büchi automaton B_ϕ defined by $S = \{s_1, s_2\}$, $S_0 = \{s_1\}$, $O = \{o_1, \ldots, o_5\}$, $F = \{s_2\}$, $\delta(s_1, o_1) = \delta(s_1, o_2) = \delta(s_1, o_4) = \delta(s_1, o_5) = \{s_1\}$, $\delta(s_1, o_3) = \{s_2\}$, $\delta(s_2, o_3)$=$\{s_2\}$, $\delta(s_2, o_1) = \delta(s_2, o_2) = \delta(s_2, o_4) = \delta(s_2, o_5) = \{s_1\}$. Note that B_ϕ has a structure that resembles the automaton shown in Fig. 2.3d but with different transition labels and is indeed deterministic.
ii. The initial quotient $T/_\sim$ of T under the observational equivalence relation $\sim$ is constructed (the quotient $T/_\sim$ was already shown in Fig. 1.11b). Then, the product automaton $P = T/_\sim \otimes B_\phi$ shown in Fig. 4.10a is constructed (note that unreachable states in P are removed during the construction). The set of states of P, from which all runs are guaranteed to visit a final state of P infinitely often (all runs are accepted in P) are identified as $S_\top = \{(X_1, s_1), (X_4, s_1), (X_3, s_1), (X_3, s_2), (X_4, s_2)\}$. Similarly, the set of states from which no run visits a final state infinitely often is identified as $S_\bot = \{(X_5, s_1)\}$. Using this information, we can compute the largest

Fig. 4.10 Computing the set of states $S_\top$ and $S_\perp$ of the product automaton $P = T/_\sim \otimes B_\phi$, shown respectively in *light gray* and *dark gray* in (**a**), allows both the simplification of P into $P' = \mathrm{S{\small IMPLIFY}}(P)$ (**b**) and the computation of the largest satisfying and strictly violating regions of $T/_\sim$, together with a set of states $S_\div$, where refinement is required. See Example 4.11 (steps i–iii) for details

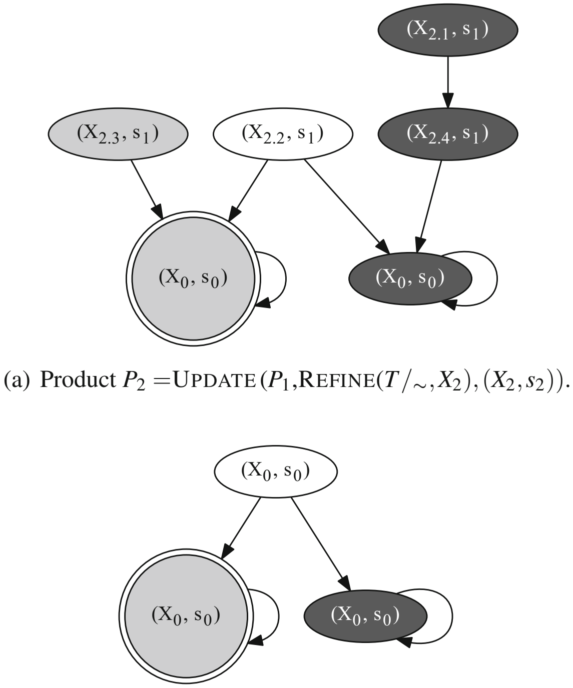


(a) Product automaton $P = T/_\sim \otimes B_\phi$.

(b) Simplified product $P_1 = \mathrm{S{\small IMPLIFY}}(P)$.

satisfying and strictly violating regions of $T/_\sim$ through the projections $X^{\phi}_{T/_\sim} = \alpha(S_\top \cap S_{P0}) \subseteq X/_\sim = \{X_1, X_3, X_4\}$ and $X^{\neg\phi}_{T/_\sim} = \alpha(S_\perp \cap S_{P0}) = \{X_5\}$, which agrees with the previously computed sets from Example 4.4

Fig. 4.11 The product P_2 is obtained by updating P_1 to capture the changes made to the quotient $T/_{\sim}$ through refinement. The sets of states $S_{\top}$ and $S_{\bot}$ of P_2 (shown respectively in *light gray* and *dark gray* in (**a**)) are computed and allows expanding the largest satisfying and strictly violating regions of T and the additional simplification of P_2 into P_3 = SIMPLIFY(P_2) (**b**). See Example 4.11 (steps iv, v) for details

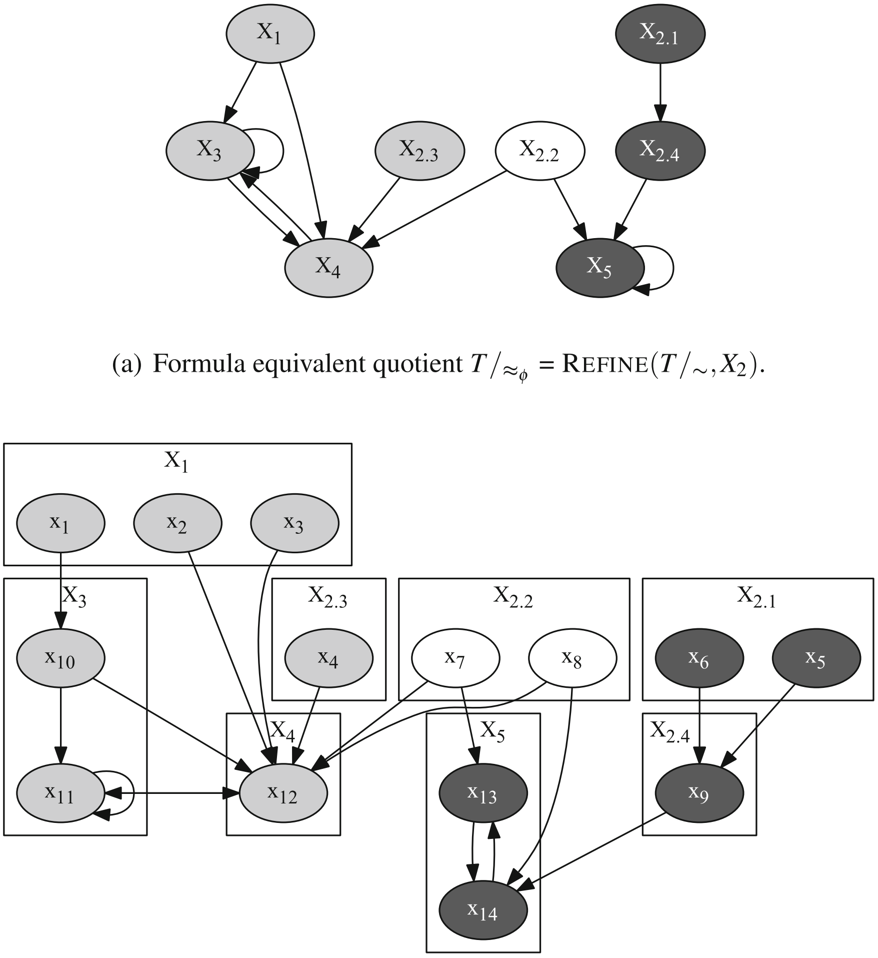


(a) Product P_2 =UPDATE (P_1,REFINE($T/_{\sim}$,X_2), (X_2,s_2)).

(b) Simplified product P_3 = SIMPLIFY (P_2).

(Fig. 4.4a). In this case, no states are inherently uncertain in $T/_{\sim}$—in the following steps we will show that state (X_2, s_1) can be refined and, therefore, it fails the characterization from Proposition 4.4. Then, the set $S_{\prec}$ is empty and refinement is targeted to the set $S_{\div} = \{(X_2, s_1)\}$.

iii. While the sets of states $S_{\top}$ and $S_{\bot}$ computed at the previous step provide only an under-approximation of the largest satisfying and strictly violating sets of T (this approximation was already shown in Fig. 4.4b), they allow us to simplify the product automaton P into P_1 = SIMPLIFY(P) shown in Fig. 4.10b. All states from the set $S_{\top}$ are replaced by a single dummy state (X_0, s_0) and the same simplification is performed for the set $S_{\bot}$ (note that one of the dummy states is accepting and the other one is not). Both dummy states have self loops and all transitions from the remaining states of P to a state from $S_{\top}$ or $S_{\bot}$ are replaced with a transition to the corresponding dummy state, thereby preserving the structure of P. Since, in this case, the set $S_{\prec}$ is empty, the third “inherently uncertain” dummy state is omitted.



(a) Formula equivalent quotient $T/_{\approx_\phi}$ = REFINE$(T/_{\sim}, X_2)$.

(b) Equivalence classes $X/_{\approx_\phi}$ of $T/_{\approx_\phi}$.

Fig. 4.12 The formula equivalent quotient $T/_{\approx_\phi}$ (**a**), constructed after refinement of $T/_{\sim}$, can be used to equivalently identify the largest satisfying, strictly violating and uncertain regions of T (**b**). See Example 4.11 (steps vi) for details

iv. The set of states from the projection $\alpha(S_{\div}) = \{X_2\}$ must be refined in $T/_{\sim}$. The refined quotient is constructed as $T/_{\sim_r}$ = REFINE$(T/_{\sim}, X_2)$ (see Fig. 4.12a), where state X_2 is partitioned into the set of subsets $\{X_{2.1}, X_{2.2}, X_{2.3}, X_{2.4}\}$ (i.e., $con(X_2) = con(X_{2.1}) \cup con(X_{2.2}) \cup con(X_{2.3}) \cup con(X_{2.4})$ in Fig. 4.12b). The refinement of $T/_{\sim}$ is projected to the product automaton by applying the procedure P_2 =UPDATE$(P_1, T/_{\sim_r}, (X_2, s_2))$, rather than recomputing it as $P_2 = T/_{\sim_r} \otimes B_\phi$, which leads to a simpler structure.

v. Once the updated product P_2 is computed, the sets $S_\top$ and $S_\perp$ can be computed again (which now include the dummy states). State $(X_{2.2}, s_1)$ satisfies the characterization from Proposition 4.4. Therefore, the projection $\alpha((X_{2.2}, s_1)) = X_{2.2}$ corresponds to an inherently uncertain region of T (i.e., there exists both runs that satisfy ϕ and that violate it (and satisfy $\neg\phi$) that originate at each state $x \in con(X_{2.2})$). Simplifying the product as P_3 = SIMPLIFY(P_2) results in an automaton containing dummy states only—now, an inherently uncertain dummy state with transitions to both the satisfying and rejecting dummy states is included.

vi. All states of P_2 have been partitioned between the sets $S_\top$, $S_\perp$ and $S_?$. Further refinement of states of $T/_{\sim_r}$ is not going to improve the solution and, in fact, a ϕ-equivalent quotient has been obtained. Furthermore, the computation from Algorithm 8 guarantees that a quotient that is both ϕ-equivalent and $\neg\phi$-equivalent has been obtained and, therefore, $T/_{\sim_r} = T/_{\approx_\phi}$ in Fig. 4.11a. This allows us to guarantee that the largest satisfying, strictly violating and inherently uncertain regions $X^{\phi}_{T/\approx_\phi}$, $X^{\neg\phi}_{T/\approx_\phi}$ and $X^{\phi?}_{T/\approx_\phi}$ of $T/_{\approx_\phi}$ (Fig. 4.12a) can be used to compute the largest satisfying, strictly violating and inherently uncertain regions $X^{\phi}_{T} = con(X^{\phi}_{\hat{T}/_\sim})$, $X^{\neg\phi}_{T} = con(X^{\neg\phi}_{\hat{T}/_\sim})$ and $X^{\phi?}_{T} = con(X^{\phi?}_{\hat{T}/_\sim})$ of T (Fig. 4.12a), which provides an exact solution to Problem 4.1. Note that a coarser quotient that is only ϕ-equivalent but not $\neg\phi$-equivalent can be constructed by combining equivalence classes $X_{2.1}$ and $X_{2.2}$ into a single equivalence class—the corresponding quotient is still observation preserving, since $X_{2.1}$ and $X_{2.2}$ share the same observation o_2.

Optimizations based on the decomposition of the product P into strongly connected components (SCCs) and the subsequent construction of an SCC quotient graph, similar to the ones described for model checking in Chap. 3 are also possible. First, we make the following observations. Given states $s, s' \in S_P$, such that s' is reachable from s but s is not reachable from s', if $s' \in S_?$ then $s \in S_?$ but characterizing s as $S_?$, $S_\top$, or $S_\perp$ does not have any influence on characterization of s'. The product P is viewed as a directed graph $G_P = (S_P, E)$, where $(s, s') \in E$ if and only if $s' \in \delta_P(s)$. This graph is partitioned into maximal strongly connected components (SSCs), inducing the directed acyclic quotient graph $\mathbb{G}_P = (\mathbb{C}, \mathbb{E})$. The following properties are then guaranteed to hold:

i. for each SCC C we have $C \subseteq S_\top$, $C \subseteq S_\perp$, or $C \subseteq S_?$ (i.e., all states $s \in C$ are equivalent with respect to such characterization) and
ii. for all SCCs C, C', such that C' is reachable from C in $\mathbb{G}_P$ it holds that $C' \subseteq S_? \Rightarrow C \subseteq S_?$.

Converting the product automaton to a SCC quotient graph and processing the SCCs in bottom up manner allows for a more efficient characterization of states within each SCC and prevents unnecessary refinement.

Algorithm 8 $[X^{\phi}_{T},\ X^{\neg\phi}_{T},\ X^{\phi?}_{T}] = \textsc{FgAnalysis}(T, \phi)$: Given an LTL formula ϕ and a large or infinite transition system T, compute of the largest satisfying and strictly violating regions of T.

```
Translate $\phi$ to deterministic Büchi automaton $B_\phi$
Construct $T/_\sim$
Construct $P = T/_\sim \otimes B_\phi$
Initialize $T/_{\sim_r} := T/_\sim$
Compute $S_\top$ and $S_\bot$ in $P$
$S_? := S_P \setminus (S_\top \cup S_\bot)$
Compute $S_\prec \subseteq S_?$, $S_\div := S_? \setminus S_\prec$
repeat
  for all $(X_i, s) \in S_\div$ do
    if $X$ not yet refined in $T/_{\sim_r}$ then
      $T/_{\sim_r} := \textsc{Refine}(T/_{\sim_r}, X_i)$
    end if
    $P := \textsc{Update}\ (P, T/_{\sim_r}, (X_i, s))$
  end for
  $P := \textsc{Simplify}(P)$
until $S_\div = \emptyset$
$X^{\phi}_{T/\approx_\phi} = \alpha(S_\top \cap S_{P0})$, $X^{\neg\phi}_{T/\approx_\phi} = \alpha(S_\bot \cap S_{P0})$, $X^{\phi?}_{T/\approx_\phi} = \alpha(S_\prec \cap S_{P0})$
return $X^{\phi}_{T} = con(X^{\phi}_{\hat{T}/\sim})$, $X^{\neg\phi}_{T} = con(X^{\neg\phi}_{\hat{T}/\sim})$, $X^{\phi?}_{T} = con(X^{\phi?}_{\hat{T}/\sim})$
```

The overall method discussed in this section is summarized in Algorithm 8. As before, this procedure cannot be guaranteed to terminate for general transitions system T returning the formula equivalent quotient $T/_\approx$, for which the set $S_\div$ is empty. Instead, termination can again be ensured by either limiting the number of iterations or by only refining states of $\hat{T}/_\sim$ that correspond to "large enough" regions of T. If computation is stopped because an iteration limit is reached or no additional "large enough" states remain, only an under-approximation of the solution to Problem 4.1 is obtained, which can be improved by adjusting these limits.

4.6 Notes

The analysis methods presented in this chapter were based on the construction, iterative refinement and verification of finite abstractions (quotients) of large or infinite systems. The verification strategy we used was based on LTL model checking [45] (discussed in Chap. 3), while our refinement procedures were inspired by bisimulation-based refinement [35, 99]. A related idea of developing iterative procedures that combine model checking and refinement was used in [42] for verification from formulas in the universal fragment ACTL of CTL.

The abstractions we used could be sufficient, providing only approximate results to analysis problems, in which case they were constructed using simulation relations [45]. On the other hand, abstractions that were equivalent for all LTL specifications were constructed using the notion of bisimulation [131]. We also described a refinement procedure aimed at the constructions of abstractions that were equivalent to large or infinite systems only with respect to given LTL specifications. The related idea of defining CTL formula-specific equivalences coarser than bisimulation has been explored in [13] in the context of finite state systems.

Relying on a temporal logic formula to guide the refinement of an abstraction is also central to verification methods based on counterexample-guided refinement (CEGAR) [44]. The CEGAR loop involves the iterative generation and invalidation of counterexamples through model checking and abstraction refinement. Such a verification procedure might terminate if no additional counterexamples can be generated (the specification is satisfied), a valid counterexample is found (the specification is violated) or the computational resources are exhausted (results are inconclusive). Instead of performing many model checking steps, our method from Sect. 4.5 aimed directly at the construction of formula equivalent quotients.

The methods from this chapter provided more informative analysis results than simple Yes/No answers (as obtained by model-checking) by identifying satisfying and violating regions (i.e., regions of initial conditions from which all trajectories of the system are guaranteed to satisfy or violate the specification). In some cases, our analysis procedure revealed regions of initial conditions for which nothing can be guaranteed (i.e., trajectories of the system originating in such states might satisfy the specification) but the satisfaction of such states could be resolved through additional computation. Labeling sets of states by *true*, *false* or *maybe* with respect to the satisfaction of the LTL specification as described above was also central to our methods and is related to the construction and refinement of 3-valued abstractions [37, 41].

The analysis procedure from Sect. 4.5 was based on the computation of attractor sets of finite control transition systems, described in [104] and inspired by automata theoretic model checking and Büchi games. Here, we applied these methods only to transition systems without inputs but in Chap. 5 we will describe the algorithms in more detail and use them for the construction of control strategies—the original context in which these methods were developed. The approach presented in this chapter was enabled by the assumption that the LTL specification can be translated into a deterministic Büchi automaton, which simplified the presentation significantly. It is known that there exist LTL formulas that can only be translated to nondeterministic Büchi automata (i.e., the expressivity of specifications used as part of our method is currently restricted to only a fragment of LTL). However, any LTL formula can also be translated into a language-equivalent deterministic Rabin automaton [153], a translation that is associated with a higher computational cost but allows an extension of this method to arbitrary LTL formulas. We will describe such an extension based on the use of deterministic Rabin automata in Chap. 5.

Chapter 5
Finite Temporal Logic Control

In this chapter, we treat the general problem of controlling non-deterministic finite transition systems from specifications given as LTL formulas over their sets of observations. We show that, in general, this control problem can be mapped to a Rabin game. For the particular case when the LTL formula translates to a deterministic Büchi automaton, we show that a more efficient solution to the control problem can be found via a Büchi game. Finally, for specifications given in the syntactically co-safe fragment of LTL, we show that the control problem maps to a simple reachability problem. For all three cases, we present all the details of the involved algorithms and several illustrative examples. In Part III, we combine these algorithms with abstractions to derive LTL control strategies for systems with infinitely many states. The problem that we consider in this chapter can be formally stated as follows:

Definition 5.1 (*Control strategy*) A (history dependent) *control function*[1] $\Omega : X^+ \to \Sigma$ for control transition system $T = (X, \Sigma, \delta, O, o)$ maps a finite, non-empty sequence of states to an input of T. A control function Ω and a set of initial states $X_0 \subseteq X$ provide a *control strategy* for T.

We denote a control strategy by (X_0, Ω), the set of all trajectories of the closed loop system T under the control strategy by $T(X_0, \Omega)$, and the set of all words produced by the closed loop T as $\mathscr{L}_T(X_0, \Omega)$. For any trajectory $x_1x_2x_3 \ldots \in T(X_0, \Omega)$ we have $x_1 \in X_0$ and $x_{k+1} \in \delta(x_k, \sigma_k)$, where $\sigma_k = \Omega(x_1, \ldots, x_k)$, for all $k \geq 1$.

Definition 5.2 (*Largest Controlled Satisfying Region*) Given a transition system $T = (X, \Sigma, \delta, O, o)$ and an LTL formula ϕ over O, the largest controlled satisfying region $X_T^{\phi} \subseteq X$ is the largest set of states for which there exists a control function $\Omega : X^+ \to \Sigma$ such that all trajectories $T(X_T^{\phi}, \Omega)$ of the closed loop system satisfy ϕ (i.e., $\mathscr{L}_T(X_T^{\phi}, \Omega) \subseteq \mathscr{L}_{\phi}$).

The LTL control problem is analogous to LTL analysis problem (Problem 4.1), and can be formulated as:

[1] In general, the control function Ω is a partial function, i.e. not every finite sequence of states is mapped to an input.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_5

Problem 5.1 (*Largest Controlled Satisfying Region Problem*) Given a finite transition system $T = (X, \Sigma, \delta, O, o)$ and an LTL formula ϕ over O, find a control strategy (X_T^{ϕ}, Ω) such that X_T^{ϕ} is the largest controlled satisfying region and $\mathscr{L}_T(X_T^{\phi}, \Omega) \subseteq \mathscr{L}_{\phi}$.

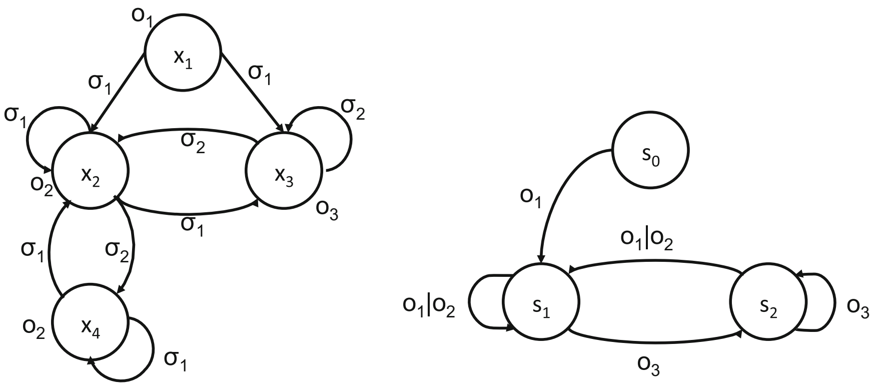
The control problem for transition systems from LTL specifications is stated in most general form in Problem 5.1, i.e., for nondeterministic transition systems and full LTL specifications. In the following section, we present an algorithm to solve this problem and discuss the related complexity. In the presented algorithm, the control synthesis problem is treated as a game played on a finite graph and approached using automata theoretic methods. Such game semantics are introduced due to the nondeterminism of the transition system and the accepting condition of a Rabin automaton. However, if the transition system is deterministic, the control problem can be solved through model checking techniques in a more efficient way. In the subsequent sections, we focus on particular cases of this problem, e.g., when the LTL formula can be translated to a deterministic Büchi automaton (a dLTL specification), and when the LTL formula can be translated to an FSA (an scLTL formula), and present more efficient solutions to the control problem and discuss the associated complexities.

5.1 Control of Transition Systems from LTL Specifications

In this section, we provide a solution to the general problem of controlling finite, nondeterministic systems from LTL specifications (Problem 5.1). The procedure involves the translation of the LTL formula into a deterministic Rabin automaton, the construction of the product automaton of the transition system and the Rabin automaton, followed by the solution of a Rabin game on this product. The solution of the Rabin game is a control strategy for the product automaton, and finally this solution is transformed into a control strategy for the transition system. The resulting control strategy takes the form of a *feedback control automaton*, which reads the current state of T and produces the control input to be applied at that state. The overall control procedure is summarized in Algorithm 9. In the rest of this section, we provide the details of this procedure.

Algorithm 9 LTL CONTROL(T, ϕ): Control strategy (X_T^{ϕ}, Ω) such that all trajectories in $T(X_T^{\phi}, \Omega)$ satisfy ϕ

```
1: Translate $\phi$ into a deterministic Rabin automaton $R = (S, S_0, O, \delta_R, F)$.
2: Build a product automaton $P = T \otimes R$
3: Transform $P$ into a Rabin game
4: Solve the Rabin game
5: Map the solution to the Rabin game into a control strategy for the original transition system $T$
```



(a) Transition system T

(b) $\phi = o_1 \wedge (\diamond\Box(o_1 \vee o_2) \vee \diamond\Box o_3)$

Fig. 5.1 Graphical representations of transition system (**a**) and the Rabin automaton (**b**) from Example 5.1. For the automaton, s_0 is the initial state and the acceptance condition is defined by $F = \{(G_1, B_1), (G_2, B_2)\}$, where $G_1 = B_2 = \{s_2\}$ and $B_1 = G_2 = \{s_1\}$

Step 1: Construction of the Rabin Automaton

The first step is to translate the LTL specification ϕ into a deterministic Rabin automaton R. Note that there are readily available off-the-shelf tools for such translations (see Sect. 5.4).

Example 5.1 Consider the nondeterministic transition system $T = (X, \Sigma, \delta, O, o)$ from Example 1.1 shown in Fig. 1.1, and reproduced for convenience in Fig. 5.1a. We consider the following specification "a trajectory of T originates at a state where o_1 is satisfied, and it eventually reaches and remains in a region where either o_1 or o_2 are satisfied, or o_3 is satisfied". The specification is formally defined as the LTL formula

$$\phi = o_1 \wedge (\Diamond\Box(o_1 \vee o_2) \vee \Diamond\Box o_3).$$

A Rabin automaton representation of the formula ϕ is shown in Fig. 5.1b.

Step 2: Construction of the Product Automaton

The second step is the construction of a product automaton between the transition system T and the Rabin automaton R, which is formally defined as:

Definition 5.3 (*Controlled Rabin Product Automaton*) The *controlled Rabin product automaton* $P = T \otimes R$ of a finite (control) transition system $T = (X, \Sigma, \delta, O, o)$ and a Rabin automaton $R = (S, S_0, O, \delta_R, F)$ is defined as $P=(S_P, S_{P0}, \Sigma, \delta_P, F_P)$, where

- $S_P = X \times S$ is the set of states,
- $S_{P0} = X \times S_0$ is the set of initial states,
- Σ is the input alphabet,
- $\delta_P : S_P \times \Sigma \to 2^{S_P}$ is the transition map, where $\delta_P((x, s), \sigma) = \{(x', s') \in S_P \mid x' \in \delta(x, \sigma),\ \text{and}\ s' = \delta_R(s, o(x))\}$, and
- $F_P = \{(X \times G_1, X \times B_1), \ldots, (X \times G_n, X \times B_n)\}$ is the Rabin acceptance condition.

This product automaton is a nondeterministic Rabin automaton with the same input alphabet Σ as T. Each accepting run $(x_1, s_1)(x_2, s_2)(x_3, s_3) \ldots$ of a product automaton $P = T \otimes R$ can be projected into a trajectory $x_1 x_2 x_3 \ldots$ of T, such that the word $o(x_1)o(x_2)o(x_3) \ldots$ is accepted by R (i.e., satisfies ϕ) and vice versa. This allows us to reduce Problem 5.1 to finding a control strategy for P. We define a control strategy for a Rabin automaton, and therefore for a product automaton constructed as in Definition 5.3, similarly as for a transition system. However, instead of history dependent control strategy, we introduce a memoryless strategy. As we will present later in this section, control strategies obtained by solving Rabin games (step 4 of Algorithm 9) are memoryless.

Definition 5.4 (*Control strategy for a Rabin automaton*) A memoryless *control function* $\pi : S \to O$ for a Rabin automaton $R = (S, S_0, O, \delta_R, F)$ maps a state of R to an input of R. A control function π and a set of initial states $W_0 \subseteq S_0$ provide a *control strategy* (W_0, π) for R.

A run $s_1 s_2 s_3 \ldots$ *under strategy* (W_0, π) is a run satisfying the following two conditions: (1) $s_1 \in W_0$ and (2) $s_{k+1} \in \delta_R(s_k, \pi(s_k))$, for all $k \geq 1$.

The product automaton P allows us to reduce Problem 5.1 to the following problem:

Problem 5.2 Given a controlled Rabin product automaton $P{=}(S_P, S_{P0}, \Sigma, \delta_P, F_P)$ find the largest set of initial states $W_{P0} \subseteq S_{P0}$ for which there exists a control function $\pi_P : S_P \to \Sigma$ such that each run of P under the strategy (W_{P0}, π_P) satisfies the Rabin acceptance condition F_P.

Example 5.2 The product automaton $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$ of the transition system and the Rabin automaton from Example 5.1 (Fig. 5.1) is shown in Fig. 5.2. Note that the blocking states that are not reachable from the non-blocking initial state $p_0 = (x_1, s_0)$ are removed from P and are not shown in Fig. 5.2.

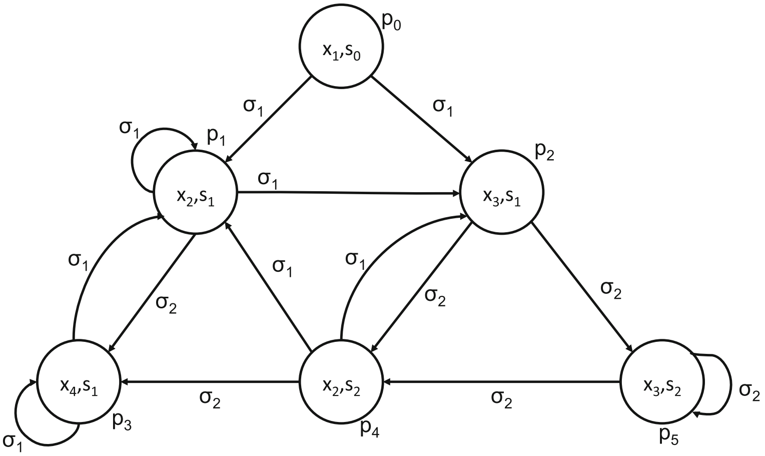


Fig. 5.2 Graphical representation of the product between the transition system from Fig. 5.1a and the Rabin automaton from Fig. 5.1b. The initial state is $p_0 = (x_1, s_0)$. The accepting condition is defined by $F_P = \{(G_1, B_1), (G_2, B_2)\}$, where $G_1 = B_2 = \{p_4, p_5\}$ and $G_2 = B_1 = \{p_1, p_2, p_3\}$

Step 3: Translation to a Rabin Game

A Rabin game consists of a finite graph (V, E) containing a token. The token is always present in one of the states and can move along the edges. There are two players: a protagonist and an adversary. V is partitioned into protagonist's states $V_{\mathbf{P}}$ and adversary's states $V_{\mathbf{A}}$. The owner of the state containing a token chooses the edge along which the token moves. A Rabin game is formally defined as:

Definition 5.5 (*Rabin Game*) A Rabin game played by two players (a protagonist and an adversary) on a graph (V, E) is a tuple $\mathbf{G} = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}})$, where

- $V_{\mathbf{P}}$ is the set of protagonist's states,
- $V_{\mathbf{A}}$ is the set of adversary's states,
- $V_{\mathbf{P}} \cup V_{\mathbf{A}} = V$, $V_{\mathbf{P}} \cap V_{\mathbf{A}} = \emptyset$,
- $E \subseteq V \times V$ is the set of possible actions,
- $F_{\mathbf{G}} = \{(G_1, B_1), \ldots, (G_n, B_n)\}$ is the winning condition for the protagonist, where $G_i, B_i \subseteq V$ for all $i \in \{1, \ldots, n\}$.

A play p is an infinite sequence of states visited by the token. Each play is winning either for the protagonist or the adversary. The protagonist wins if $inf(p) \cap G_i \neq \emptyset \wedge inf(p) \cap B_i = \emptyset$ for some $i \in \{1, \ldots, n\}$, where $\inf(p)$ denotes the set of states that appear in the play p infinitely often. The adversary wins in the rest of the cases. The winning region for the protagonist is defined as the set of states $W_{\mathbf{P}} \subseteq V$ such that there exists a control function $\pi_P : W_{\mathbf{P}} \cap V_{\mathbf{P}} \to E$, and all plays starting in the winning region and respecting the winning strategy are winning for the protagonist regardless of the adversary's choices. A solution to a Rabin game is a winning region and winning strategy for the protagonist.

The third step of Algorithm 9 is the construction of a Rabin game from the product automaton, which is performed as follows.

Definition 5.6 (*Rabin game of a Rabin automaton*) A *Rabin game* $\mathbf{G} = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}})$ of a Rabin automaton $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$ is defined as:

- $V_{\mathbf{P}} = S_P$ is the protagonist's states,
- $V_{\mathbf{A}} = S_P \times \Sigma$ is the adversary's states,
- $E \subseteq \{V_{\mathbf{P}} \times V_{\mathbf{A}} \cup V_{\mathbf{A}} \times V_{\mathbf{P}}\}$ is the set of edges, which is defined as
  - $(q_{\mathbf{P}}, q_{\mathbf{A}}) \in E$ if $q_{\mathbf{P}} \in V_{\mathbf{P}}$, $q_{\mathbf{A}} \in V_{\mathbf{A}}$, and $q_{\mathbf{A}} = (q_{\mathbf{P}}, \sigma)$, where $\sigma \in \Sigma^{q_{\mathbf{P}}}$ (i.e., if $\delta_P(q_{\mathbf{P}}, \sigma) \neq \emptyset$),
  - $(q_{\mathbf{A}}, q_{\mathbf{P}}) \in E$ if $q_{\mathbf{A}} \in V_{\mathbf{A}}$, $q_{\mathbf{P}} \in V_{\mathbf{P}}$, and $q_{\mathbf{A}} = (q'_{\mathbf{P}}, \sigma)$, and $q_{\mathbf{P}} \in \delta_P(q'_{\mathbf{P}}, \sigma)$,
- $F_{\mathbf{G}} = F_P$ is the protagonist's winning condition.

Intuitively, the protagonist chooses action σ, whereas the adversary resolves nondeterminism. Note that in a Rabin game constructed from a Rabin automaton, the protagonist's (adversary's) states can be reached in one step only from the adversary's (protagonist's) states. We will show later in this section that a solution to the Rabin game $\mathbf{G}$ can be easily transformed into a solution to Problem 5.2.

Example 5.3 The Rabin game of the product automaton from Example 5.2 is shown in Fig. 5.3, where protagonist's states are represented as circles and adversary's states are represented as rectangles.

Step 4: Solving the Rabin Game

We present Horn's algorithm for solving Rabin games. The main idea behind the algorithm is as follows. The protagonist wins if they can infinitely often visit G_i and avoid B_i for some $i \in \{1, \dots, n\}$. Conversely, the protagonist can not win if the adversary can infinitely often visit B_i for each $i \in \{1, \dots, n\}$. Since it is sufficient for the protagonist to satisfy one of the conditions (G_i, B_i) from $F_{\mathbf{G}}$, the protagonist chooses a condition and tries to avoid visits to B_i and enforce visits to G_i. In turn the adversary tries to avoid G_i. By removing the states where the protagonist (or the adversary) can enforce a visit to a desired set, a smaller game is defined and the algorithm is applied to this game recursively. If the computation ends favorably for the adversary, then the protagonist chooses a different condition (G_j, B_j) from $F_{\mathbf{G}}$ and tries to win the game by satisfying this condition. For a given set $V' \subset V$, the set of states from which the protagonist (or the adversary) can enforce a visit to V' is called an *attractor set*, which is formally defined as follows:

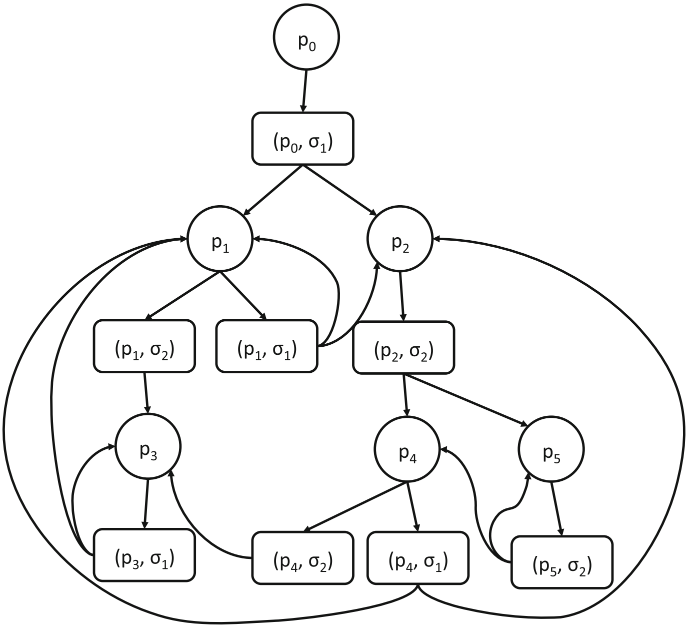


Fig. 5.3 Graphical representation of the Rabin game constructed from the Rabin automaton from Fig. 5.2. An example play is $p = p_0(p_0, \sigma_1)p_2(p_2, \sigma_2)(p_5(p_5, \sigma_2))^\omega$

Definition 5.7 (*Protagonist's direct attractor*) The protagonist's direct attractor of a set of states V', denoted by $\mathsf{A}^1_{\mathbf{P}}(V')$, is the set of all states $v_{\mathbf{P}} \in V_{\mathbf{P}}$, such that there exists an edge $(v_{\mathbf{P}}, v_{\mathbf{A}})$, where $v_{\mathbf{A}} \in V'$ together with the set of all states $v_{\mathbf{A}} \in V_{\mathbf{A}}$, such that for all $v_{\mathbf{P}} \in V_{\mathbf{P}}$ it holds that $(v_{\mathbf{A}}, v_{\mathbf{P}}) \in E$ implies $v_{\mathbf{P}} \in V'$:

$$\mathsf{A}^1_{\mathbf{P}}(V') := \{v_{\mathbf{P}} \in V_{\mathbf{P}} | (v_{\mathbf{P}}, v_{\mathbf{A}}) \in E, v_{\mathbf{A}} \in V'\} \bigcup \{v_{\mathbf{A}} \in V_{\mathbf{A}} | \{v_{\mathbf{P}} | (v_{\mathbf{A}}, v_{\mathbf{P}}) \in E\} \subseteq V'\}.$$

Definition 5.8 (*Adversary's direct attractor*) The adversary's direct attractor of V', denoted by $\mathsf{A}^1_{\mathbf{A}}(V')$, is the set of all states $v_{\mathbf{A}} \in V_{\mathbf{A}}$, such that there exists an edge $(v_{\mathbf{A}}, v_{\mathbf{P}})$, where $v_{\mathbf{P}} \in V'$ together with the set of all states $v_{\mathbf{P}} \in V_{\mathbf{P}}$, such that for all $v_{\mathbf{A}} \in V_{\mathbf{A}}$ it holds that $(v_{\mathbf{P}}, v_{\mathbf{A}}) \in E$ implies $v_{\mathbf{A}} \in V'$:

$$\mathsf{A}^1_{\mathbf{A}}(V') := \{v_{\mathbf{A}} \in V_{\mathbf{A}} | (v_{\mathbf{A}}, v_{\mathbf{P}}) \in E, v_{\mathbf{P}} \in V'\} \bigcup \{v_{\mathbf{P}} \in V_{\mathbf{P}} | \{v_{\mathbf{A}} | (v_{\mathbf{P}}, v_{\mathbf{A}}) \in E\} \subseteq V'\}.$$

In other words, the protagonist can enforce a visit to V' from each state $v \in \mathsf{A}^1_{\mathbf{P}}(V')$, regardless of the adversary's choice. Similarly, the adversary can enforce a visit to V' from each state $v \in \mathsf{A}^1_{\mathbf{A}}(V')$, regardless of the protagonist's choice.

Example 5.4 Consider the Rabin game shown in Fig. 5.3. The protagonist's direct attractor set of $\{p_5\}$, $\mathsf{A}^1_{\mathbf{P}}(\{p_5\})$ is empty, since p_5 can be reached from (p_2, σ_2) and (p_5, σ_2), and for both these states the adversary can choose an edge incident to p_4 instead of p_5. On the other hand

$$\mathsf{A}^1_{\mathbf{P}}(\{p_4, p_5\}) = \{(p_2, \sigma_2), (p_5, \sigma_2)\},$$

since at (p_2, σ_2) (and similarly at (p_5, σ_2)), the adversary can either choose the edge $((p_2, \sigma_2), p_4)$ or $((p_2, \sigma_2), p_5)$ and both lead to $\{p_4, p_5\}$.

The adversary's direct attractor set of $\{p_5\}$ is $\mathsf{A}^1_{\mathbf{A}}(\{p_5\}) = \{(p_2, \sigma_2), (p_5, \sigma_2)\}$, since the adversary can enforce a visit to $\{p_5\}$ only from (p_2, σ_2) and (p_5, σ_2). As there are no other adversary states that have an edge to a state from the set $\{p_4, p_5\}$, we have:

$$\mathsf{A}^1_{\mathbf{A}}(\{p_4\}) = \mathsf{A}^1_{\mathbf{A}}(\{p_5\}) = \mathsf{A}^1_{\mathbf{A}}(\{p_4, p_5\}) = \{(p_2, \sigma_2), (p_5, \sigma_2)\}.$$

The protagonist's attractor set $\mathsf{A}_{\mathbf{P}}(V')$ is the set of all states from which a visit to V' can be enforced by the protagonist in zero or more steps. $\mathsf{A}_{\mathbf{P}}(V')$ can be computed iteratively via computation of the converging sequence

$$\mathsf{A}^*_{\mathbf{P}0}(V') \subseteq \mathsf{A}^*_{\mathbf{P}1}(V') \subseteq \ldots,$$

where $\mathsf{A}^*_{\mathbf{P}0}(V') = V'$ and

$$\mathsf{A}^*_{\mathbf{P}i+1}(V') = \mathsf{A}^1_{\mathbf{P}}(\mathsf{A}^*_{\mathbf{P}i}(V')) \cup \mathsf{A}^*_{\mathbf{P}i}(V').$$

The sequence is indeed converging because there are at most $|V_{\mathbf{P}} \cup V_{\mathbf{A}}|$ different sets in the sequence. Intuitively $\mathsf{A}^*_{\mathbf{P}_i}(V')$ is the set from which a visit to the set V' can be enforced by the protagonist in at most i steps.

Example 5.5 Consider the Rabin game shown in Fig. 5.3. The protagonist's attractor set for $V' = \{p_4, p_5\}$ is recursively computed as follows:

$$\begin{aligned}
&\mathsf{A}^*_{\mathbf{P}1}(V') = \mathsf{A}^*_{\mathbf{P}0}(V') \cup \mathsf{A}^1_{\mathbf{P}}(\{p_4, p_5\}) = \{p_4, p_5, (p_2, \sigma_2), (p_5, \sigma_2)\},\\
&\mathsf{A}^*_{\mathbf{P}2}(V') = \mathsf{A}^*_{\mathbf{P}1}(V') \cup \mathsf{A}^1_{\mathbf{P}}(\mathsf{A}^*_{\mathbf{P}1}(V')) = \{p_2, p_4, p_5, (p_2, \sigma_2), (p_5, \sigma_2)\},\\
&\mathsf{A}^*_{\mathbf{P}}(V') = \mathsf{A}^*_{\mathbf{P}3}(V') = \mathsf{A}^*_{\mathbf{P}2}(V') \cup \mathsf{A}^1_{\mathbf{P}}(\mathsf{A}^*_{\mathbf{P}2}(V')) = \{p_2, p_4, p_5, (p_2, \sigma_2), (p_5, \sigma_2)\}.
\end{aligned}$$

The adversary's attractor set of V' is computed similarly. This computation converges at the fifth iteration, and the resulting set is

$$\mathsf{A}^*_{\mathbf{A}}(V') = \{p_0, p_2, p_4, p_5, (p_0, \sigma_1), (p_1, \sigma_1), (p_2, \sigma_2), (p_4, \sigma_1), (p_5, \sigma_2)\}. \tag{5.1}$$

Attractor strategy $\pi_{\mathsf{A}_{\mathbf{P}}(V')}$ for the protagonist's attractor set determines how to ensure a visit to set V' from attractor set $\mathsf{A}_{\mathbf{P}}(V')$. For all $v \in \mathsf{A}^*_{\mathbf{P}i+1}(V') \setminus \mathsf{A}^*_{\mathbf{P}i}(V')$, the attractor strategy is defined as $\pi_{\mathsf{A}_{\mathbf{P}}(V')}(v) = (v, v')$, where v' is an arbitrary $v' \in \mathsf{A}^*_{\mathbf{P}i}(V')$. The adversary's attractor $\mathsf{A}_{\mathbf{A}}(V')$ and attractor strategy $\pi_{\mathsf{A}_{\mathbf{A}}(V')}$ are computed analogously. The protagonist's and adversary's attractors of V' in a game $\mathbf{G}$ are denoted by $\mathsf{A}^{\mathbf{G}}_{\mathbf{P}}(V')$ and $\mathsf{A}^{\mathbf{G}}_{\mathbf{A}}(V')$, respectively.

Let (V, E) denote the graph of a Rabin game $G = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}})$, where $V = V_{\mathbf{P}} \cup V_{\mathbf{A}}$. For simplicity, for a set $Q \subseteq V$, we denote $\mathbf{G} \setminus Q$ the graph $(V \setminus Q, E \setminus E')$ (and the corresponding game), where E' is the set of all edges incident with states from Q.

Horn's algorithm is summarized in Algorithm 10. First the protagonist chooses a condition (G_i, B_i) (line 1). As the protagonist needs to avoid B_i, a sub game $\mathbf{G}^0_i$ is defined by removing the adversary's attractor set for B_i. Then, a sub game $\mathbf{G}^j_i$ is defined iteratively by removing winning regions for the adversary (line 7). The iterative process terminates when no winning region is found for the adversary, i.e., $\mathbf{G}^j_i = \mathbf{G}^{j+1}_i$. In this case, either $\mathbf{G}^j_i$ is empty, or it is winning for the protagonist. If $\mathbf{G}^j_i$ is not empty, then the protagonists attractor of $\mathbf{G}^j_i$ in game $\mathbf{G}$ (line 11) is also winning for the protagonist. By removing the winning region for the protagonist (line 14), a new smaller game is defined and the algorithm is run on this game.

Algorithm 10 RABINGAME $(G = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}}))$: Winning region $W_{\mathbf{P}} \subseteq (V_{\mathbf{P}} \cup V_{\mathbf{A}})$ and winning strategy $\pi_{\mathbf{P}}$ for the protagonist, winning region $W_{\mathbf{A}} \subseteq (V_{\mathbf{P}} \cup V_{\mathbf{A}})$ for the adversary

```
1: for all $(G_i, B_i) \in F_{\mathbf{G}}$ do
2:   $j = 0$
3:   $\mathbf{G}_i^j = \mathbf{G} \setminus \mathsf{A}_{\mathbf{A}}^{\mathbf{G}}(B_i)$   {remove all states in $\mathsf{A}_{\mathbf{A}}^{\mathbf{G}}(B_i)$ and transitions adjacent to them from $\mathbf{G}$}
4:   repeat
5:     $\mathbf{H}_i^j = \mathbf{G}_i^j \setminus \mathsf{A}_{\mathbf{P}}^{\mathbf{G}_i^j}(G_i)$   {note that $(G_i, B_i)$ is not present in $\mathbf{H}_i^j$ any more}
6:     $(W'_{\mathbf{P}}, \pi'_{\mathbf{P}}, W'_{\mathbf{A}})$ = RABINGAME($\mathbf{H}_i^j$)   {recursive call}
7:     $\mathbf{G}_i^{j+1} = \mathbf{G}_i^j \setminus \mathsf{A}_{\mathbf{A}}^{\mathbf{G}_i^j}(W'_{\mathbf{A}})$
8:     $j$++
9:   until $\mathbf{G}_i^j = \mathbf{G}_i^{j+1}$   {$\mathbf{G}_i^j$ is guaranteed to be winning for the protagonist}

10:   if $\mathbf{G}_i^j \neq \emptyset$ then
11:     $W_{\mathbf{P}} = W_{\mathbf{P}} \cup \mathsf{A}_{\mathbf{P}}^{\mathbf{G}}(\mathbf{G}_i^j)$   {The protagonist's attractor of $\mathbf{G}_i^j$ in $\mathbf{G}$ is winning}
12:     $\pi_{\mathbf{P}} = \pi_{\mathbf{P}} \cup \pi_{\mathbf{P}}^{'} \cup \pi_{\mathbf{P}}^{''}$,   {$\pi_{\mathbf{P}}^{'}$ is the protagonist's attractor strategy computed in line 6}
13:                                            {$\pi_{\mathbf{P}}^{''}$ is the protagonist's attractor strategy for $\mathsf{A}_{\mathbf{P}}^{\mathbf{G}}(\mathbf{G}_i^j)$}
14:     $\mathbf{G}^s = \mathbf{G} \setminus W_{\mathbf{P}}$
15:     $(W_{\mathbf{P}}^s, \pi_{\mathbf{P}}^s, W_{\mathbf{A}}^s)$ = RABINGAME($\mathbf{G}^s$)   {run the algorithm on a smaller graph;
                                            consider all pairs in the acceptance condition over again}
16:     $W_{\mathbf{P}} = W_{\mathbf{P}} \cup W_{\mathbf{P}}^s$
17:     $\pi_{\mathbf{P}} = \pi_{\mathbf{P}} \cup \pi_{\mathbf{P}}^s$
18:     BREAK   {break the whole for-cycle 1–18}
19:   end if
20: end for
21: $W_{\mathbf{A}} = \mathbf{G} \setminus W_{\mathbf{P}}$
```

Example 5.6 We illustrate Algorithm 10 on the Rabin game shown in Fig. 5.3. At the first iteration, we consider Rabin pair (G_1, B_1), where $G_1 = \{p_4, p_5\}$ and $B_1 = \{p_1, p_2, p_3\}$. The adversary's attractor $\mathsf{A}_{\mathbf{A}}^{\mathbf{G}}(B_1)$ is $V_{\mathbf{P}} \cup V_{\mathbf{A}}$, therefore, on line 10 of Algorithm 10, the graph $\mathbf{G}_1^0$ is empty. As we do not find any states winning for the protagonist, we continue with the next Rabin pair.

In the second iteration of Algorithm 10, we consider Rabin pair (G_2, B_2), where $G_2 = \{p_1, p_2, p_3\}$ and $B_2 = \{p_4, p_5\}$. We eliminate $A_{\mathbf{A}}^{\mathbf{G}}(B_2)$ from the graph on line 3. The remaining graph is $\mathbf{G}_2^0$. We compute $A_{\mathbf{P}}^{\mathbf{G}_2^0}(G_2)$, and find out that it is equal to $\mathbf{G}_2^0$. This means that $\mathbf{H}_2^0$ is empty, $\mathbf{G}_2^1$ is equal to $\mathbf{G}_2^0$, and $\mathbf{G}_2^0$ is guaranteed to be a part of the protagonist's winning region. $A_{\mathbf{A}}^{\mathbf{G}}(B_2)$ and $\mathbf{G}_2^0$ are shown in Fig. 5.4. The protagonist's attractor of $\mathbf{G}_2^0$ in game $\mathbf{G}$ is

$$W_{\mathbf{P}} = A_{\mathbf{P}}^{\mathbf{G}}(\mathbf{G}_2^0) = \{p_1, p_3, p_4, (p_1, \sigma_2), (p_3, \sigma_1), (p_4, \sigma_2)\},$$

and the corresponding winning strategy for the protagonist is (lines 11 and 12)

$$\begin{aligned}\pi_{\mathbf{P}}(p_1) &= (p_1, (p_1, \sigma_2)),\\ \pi_{\mathbf{P}}(p_3) &= (p_3, (p_3, \sigma_1)),\\ \pi_{\mathbf{P}}(p_4) &= (p_4, (p_4, \sigma_2)).\end{aligned}$$

As we find a winning region for the protagonist, we rerun the algorithm for a smaller game (line 15) as illustrated in Fig. 5.5. Note that the algorithm is run from the beginning on the subgame and all Rabin acceptance pairs are considered again.

At the first iteration of Algorithm 10 on the subgame $\mathbf{G}^s$ shown in Fig. 5.5, we consider Rabin pair (G_1^s, B_1^s), where $G_1^s = \{p_5\}$ and $B_1^s = \{p_2\}$. The adversary's attractor of B_1^s is $\{p_0, p_2, (p_0, \sigma_1), (p_1, \sigma_1), (p_4, \sigma_1)\}$, and the protagonist's attractor of G_1^s on $\mathbf{G}_1^0 = \mathbf{G}^s \setminus A_{\mathbf{A}}^{\mathbf{G}^s}(B_1)$ is $\mathbf{G}_1^0$. $\mathbf{H}_1^0$ is empty, and the protagonists wins everywhere in $\mathbf{G}_1^0$ and its attractor in $\mathbf{G}^s$. The attractor of $\mathbf{G}_1^0$ in $\mathbf{G}^s$ covers $\mathbf{G}^s$. Therefore, we find that the protagonist wins everywhere in $\mathbf{G}^s$ with the following strategy:

$$\begin{aligned}\pi_{\mathbf{P}}^s(p_0) &= (p_0, (p_0, \sigma_1)),\\ \pi_{\mathbf{P}}^s(p_2) &= (p_2, (p_2, \sigma_2)),\\ \pi_{\mathbf{P}}^s(p_5) &= (p_5, (p_5, \sigma_2)).\end{aligned}$$

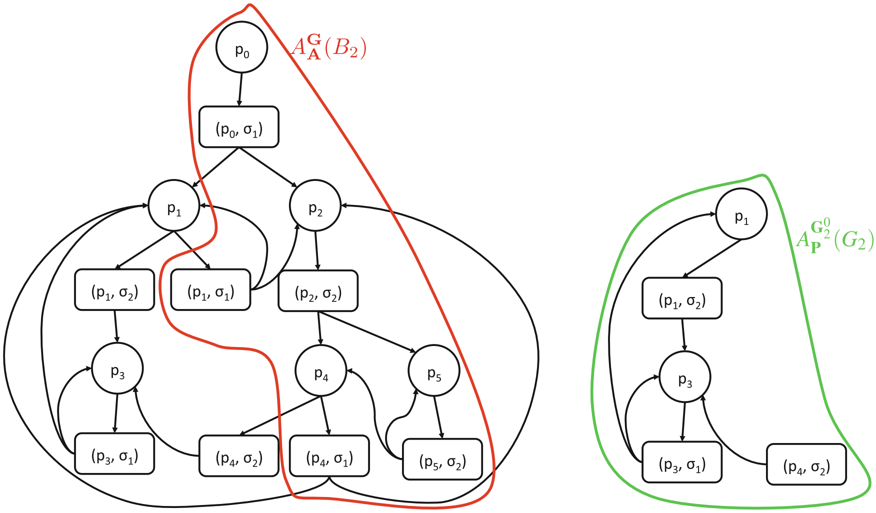
As $W_{\mathbf{P}}^s$ covers $\mathbf{G}^s$, the algorithm (recursive call on the sub-game $\mathbf{G}^s$) terminates with $W_{\mathbf{P}}^s$ and strategy $\pi_{\mathbf{P}}^s$. Finally, the winning region for the protagonist $W_{\mathbf{P}}$ on the initial game $\mathbf{G}$ covers $V_{\mathbf{P}} \cup V_{\mathbf{A}}$, and the protagonist wins everywhere in $\mathbf{G}$ with the strategy $\pi_{\mathbf{P}}$ computed in line 17.

Complexity The complexity of Algorithm 10 is $\mathscr{O}(|V|^{2n} n!)$. Intuitively, the first part ($\mathscr{O}(|V|^{2n}$) comes from the two recursions and the second part ($n!$) comes from the protagonist's ability to change the condition. For a Rabin game of a Rabin automaton, the complexity of the algorithm is $\mathscr{O}((|S_P| + |S_P||\Sigma|)^{2n} n!)$, since $V = V_{\mathbf{P}} \cup V_{\mathbf{A}}$, $V_{\mathbf{P}} = S_P$, and $V_{\mathbf{A}} = S_P \times \Sigma$.

Step 5: Mapping the Rabin Game Solution to a Control Strategy

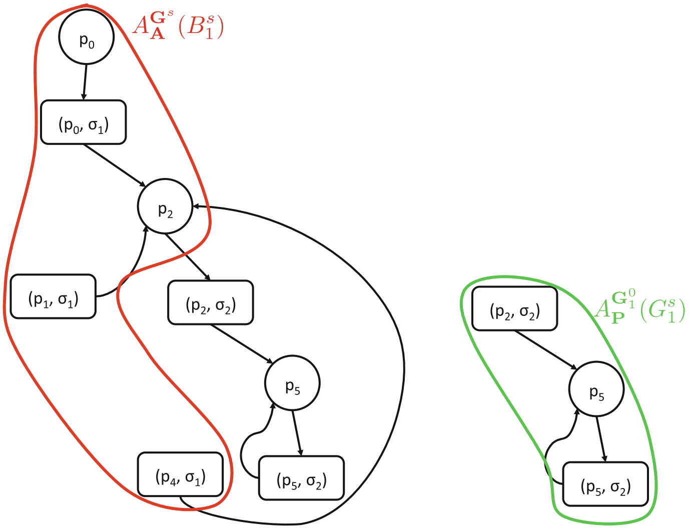
In order to complete the solution to Problem 5.1, we transform a solution to a Rabin game $\mathbf{G} = (V_{\mathbf{P}}, V_{\mathbf{A}}, E, F_{\mathbf{G}})$ of the product automaton $P = T \otimes R$ into a control strategy (X_T^{ϕ}, Ω) for T. The solution to the Rabin game is given as a winning region $W_{\mathbf{P}} \subseteq V_{\mathbf{P}}$ and a winning strategy $\pi_{\mathbf{P}} : W_{\mathbf{P}} \to E$.

We first transform the solution into a memoryless strategy for the product P, and present the solution to Problem 5.2. Clearly, the winning region for P is $W_P = W_{\mathbf{P}}$. The initial winning region is the subset of initial states that belong to W_P, i.e., $W_{P0} =$



(a) $A_{\mathbf{A}}^{\mathbf{G}}(B_2)$ (b) Sub-game G_2^0

Fig. 5.4 Adversary's attractor of B_2 in game $\mathbf{G}$ is shown with a red frame in (**a**). The sub-game $\mathbf{G}_2^0$ obtained by removing $A_{\mathbf{A}}^{\mathbf{G}}(B_2)$ from $\mathbf{G}$ is shown in (**b**). The protagonist's attractor of G_2 in game $\mathbf{G}_2^0$ covers $\mathbf{G}_2^0$



(a) $A_{\mathbf{A}}^{\mathbf{G}^s}(B_1^s)$ (b) $A_{\mathbf{P}}^{\mathbf{G}_1^0}(G_1^s)$

Fig. 5.5 Adversary's attractor set of B_1^s in game $\mathbf{G}^s$ is shown with a red frame in (**a**). The sub-game $\mathbf{G}_1^0$ is shown in (**b**). The protagonist's attractor of G_1 in game $\mathbf{G}_1^0$ covers $\mathbf{G}_1^0$

$W_P \cap S_{P0}$. The strategy π_P is obtained as follows. For all $v \in W_{\mathbf{P}}$, $\pi_P(v) = \sigma$, such that $\pi_{\mathbf{P}}(v) = (v, v')$, and $v' = (v, \sigma)$.

The remaining task is to adapt (W_{P0}, π_P) as a control strategy (X_T^{ϕ}, Ω) for T. Although the control function π_P was memoryless, Ω is history dependent and takes the form of a feedback control automaton:

Definition 5.9 Given a product automaton $P = T \otimes R$, where $T = (X, \Sigma, \delta, O, o)$ and $R = (S, S_0, O, \delta_R, F)$, a winning region W_P for P, and a control strategy (W_{P0}, π_P) for P, a feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$ is defined as

- $S_C = S$ is the set of states,
- $S_{C0} = S_0$ is the set of initial states,
- X is the set of inputs (the set of states of T),
- $\tau : S_C \times X \to S_C$ is the memory update function defined as:

$$\tau(s, x) = \delta_R(s, o(x)) \text{ if } (x, s) \in W_P, \tau(s, x) = \perp \text{ otherwise}$$

- Σ is the set of outputs (the set of inputs of T),
- $\pi : S_C \times X \to \Sigma$ is the output function:

$$\pi(s, x) = \pi_P((x, s)) \text{ if } (x, s) \in W_P, \pi(s, x) = \perp \text{ otherwise.}$$

The set of initial states X_T^{ϕ} of T is given by $\alpha(W_{P0})$, where $\alpha : S_P \to X$ is the projection from states of P to X. The control function Ω is given by C as follows: for a sequence $x_1 \ldots x_n$, $x_1 \in X_T^{\phi}$, we have $\Omega(x_1 \ldots x_n) = \sigma$, where $\sigma = \pi(s_n, x_n)$, $s_{i+1} = \tau(s_i, x_i)$, and $x_{i+1} \in \delta(x_i, \pi(s_i, x_i))$, for all $i \in \{1, \ldots, n-1\}$. It is easy to see that the product automaton of T and C will have the same states as P but contains only transitions of P closed under π_P. Then, all trajectories in $T(X_T^{\phi}, \Omega)$ satisfy ϕ and therefore (X_T^{ϕ}, Ω) is a solution to Problem 5.1. Note that if $p = (x, s) \notin W_{\mathbf{P}}$, then the adversary wins all the plays starting from p regardless of the protagonists choices, which implies that there is always a run starting from the product automaton state $(x, s) \in S_P$ that does not satisfy the Rabin acceptance condition F_P regardless of the applied control function. Therefore, Algorithm 9 finds the largest controlled satisfying region.

Example 5.7 We transform the winning region $W_{\mathbf{P}}$ and the winning strategy $\pi_{\mathbf{P}}$ found in Example 5.6 into a control strategy (X_T^{Φ}, Ω) for the transition system T and formula Φ from Example 5.1. The memoryless control strategy (W_{P0}, π_P) for the product P (Fig. 5.2) is defined as $W_{P0} = \{p_0\}$, $\pi_P(p_0) = \sigma_1, \pi_P(p_1) = \sigma_2, \pi_P(p_2) = \sigma_2, \pi_P(p_3) = \sigma_1, \pi_P(p_4) = \sigma_2$, and $\pi_P(p_5) = \sigma_2$.

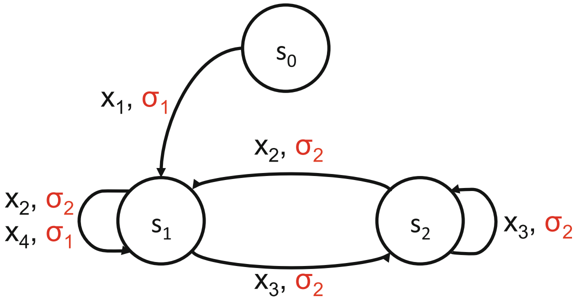


Fig. 5.6 The control automaton from Example 5.7. The initial state is s_0. The *arrows* between states are labeled with the states of the transition system depicting the memory update function. The corresponding control actions are shown in *red*

The set of initial states is $X_T^\Phi = \{x_1\}$, and the feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$, that defines the history dependent control function Ω, is constructed as in Definition 5.9. The control automaton is shown in Fig. 5.6 and is formally defined as:

$S_C = \{s_0, s_1, s_2\}$,
$S_{C0} = \{s_0\}$,
$X = \{x_1, x_2, x_3, x_4\}$,
$\tau(s_0, x_1) = s_1, \tau(s_1, x_2) = s_1, \tau(s_1, x_3) = s_2, \tau(s_1, x_4) = s_1, \tau(s_2, x_2) = s_1,$
$\tau(s_2, x_3) = s_2$,
$\Sigma = \{\sigma_1, \sigma_2\}$,
$\pi(s_0, x_1) = \sigma_1, \pi(s_1, x_2) = \sigma_2, \pi(s_1, x_3) = \sigma_2, \pi(s_1, x_4) = \sigma_1, \pi(s_2, x_2) =$
$\sigma_2, \pi(s_2, x_3) = \sigma_2$.

Example 5.8 Consider the robot transition system described in Example 1.4, and the motion planning task ϕ described in Example 2.2. The Rabin automaton representation of the formula ϕ is shown in Fig. 5.7a. The Rabin automaton, and therefore the product of the robot transition system and the Rabin automaton, has a single pair (G, B) in its accepting condition. We follow Algorithm 9 and synthesize a control strategy for the robot from the formula ϕ. The robot satisfies the motion planning task if it starts from any region except the dangerous region, i.e., $X_T^\phi = \{x_1, x_2, x_3, x_4, x_5, x_7, x_8\}$, and chooses its directions according to the control automaton C depicted in Fig. 5.7b.

When the robot starts from x_1 (B), the control automaton outputs $\pi(s_0, x_1) = W$, and updates its memory from s_0 to $\tau(s_0, x_1) = s_1$. The robot moves West and ends in x_7 (G). The next action is $\pi(s_1, x_7) = N$, and the next control automaton state is $\tau(s_1, x_7) = s_2$. The robot moves North and ends in x_4 (R). Then, the robot moves North again and ends in x_2 (I) as the control automaton outputs $\pi(s_2, x_4) = N$ and updates its memory as $\tau(s_2, x_4) = s_3$. Then, the control automaton outputs $\pi(s_3, x_2) = W$, and updates its memory as $\tau(s_3, x_2) = s_0$. The robot moves West and ends in x_0 (B). Since the robot and the control automaton both are in their initial conditions and all the applied actions are deterministic, the robot continues by applying the same series of actions, and produces the satisfying word:

$$(BGRI)^{\omega}$$

Next, we consider the second motion planning task ψ described in Example 2.2. Again, we apply Algorithm 9 and synthesize a control strategy for the specification formula ψ. The Rabin automaton representation of ψ and the control automaton generated by the algorithm are shown in Fig. 5.8. The set of satisfying initial states are $X_T^{\psi} = \{x_1, x_2, x_3, x_4, x_5, x_7, x_8\}$. When the robot starts from x_1, and chooses its directions according to the control automaton, it produces

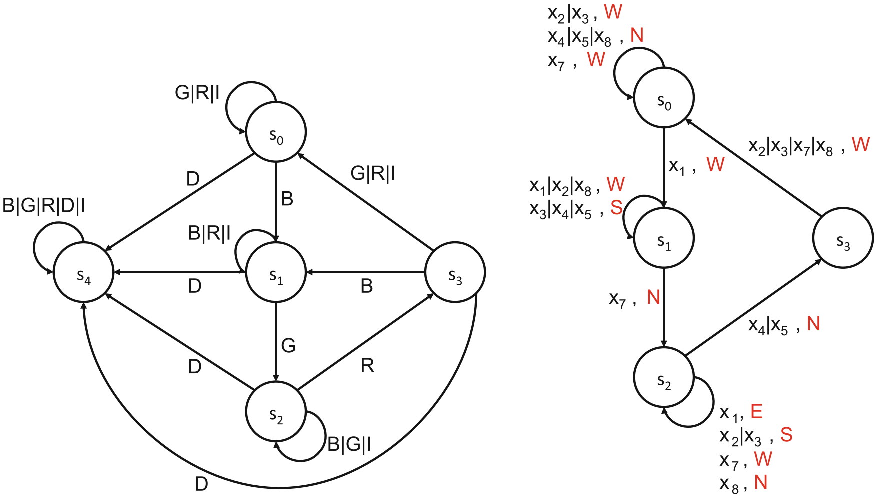
$$BIRG \text{ or } BIRIG,$$

before it returns to x_0, and the control automaton state set to s_0 again. As both the robot and the control automaton are in their initial states, the robot repeatedly produces either $BIRG$ or $BIRIG$. The corresponding word is represented as

$$(BIRG \mid BIRIG)^{\omega}.$$

5.2 Control of Transition Systems from dLTL Specifications

In this section, we present a slightly more efficient and intuitive solution to Problem 5.1 for the case when the LTL specification formula can be translated to a deterministic Büchi automaton. The solution follows the main lines of the method presented in Sect. 5.1 for arbitrary LTL specifications. Instead of the Rabin automaton, we construct a deterministic Büchi automaton, and take its product with the transition system. In this case, the product is a nondeterministic Büchi automaton. We find a control strategy for the product by solving a Büchi game and then transform it to a strategy for the original transition system. This procedure is summarized in Algorithm 11. The details are presented in the rest of this section.



(a) $\phi = \Box\Diamond B \wedge \Box\Diamond G \wedge \Box\Diamond R \wedge \Box\neg D$

(b) Control automaton C

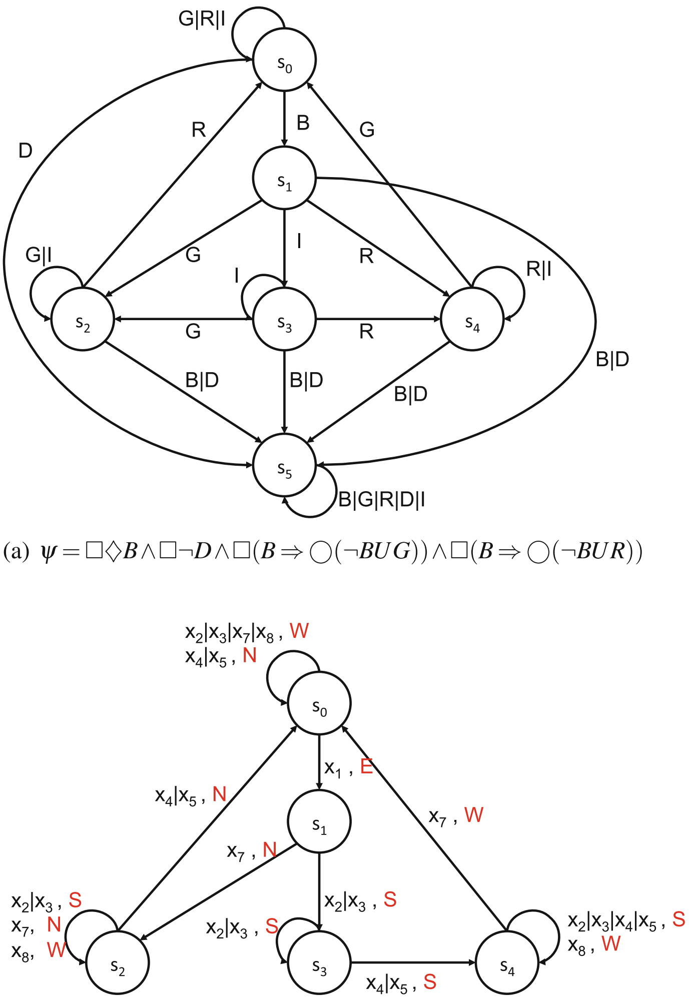
Fig. 5.7 Rabin automaton representation of the specification formula ϕ (**a**) and the control automaton (**b**) from Example 5.8. For the Rabin automaton, s_0 is the initial state. There is a single pair in the accepting condition: $F = \{(G, B)\}$, where $G = \{s_3\}$, and $B = \{s_4\}$. For the control automaton C, s_0 is the initial state. The arrows between states are labeled with the states of the robot transition system depicting the memory update function. The corresponding control actions are shown in *red*. For example $\tau(s_0, x_2) = \tau(s_0, x_3) = s_0$ and the corresponding action is defined as $\pi(s_0, x_2) = \pi(s_0, x_3) = W$. State s_4, which is not reachable from the initial state s_0, is not shown

Algorithm 11 DLTL CONTROL(T, ϕ) : Control strategy (X_T^{ϕ}, Ω) such that all trajectories in $T(X_T^{\phi}, \Omega)$ satisfy ϕ

```
Translate $\phi$ to deterministic Büchi automaton $B = (S, S_0, O, \delta_B, F)$
Build a product automaton $P = T \otimes B$
Solve a Büchi game
Map the solution to a control strategy for the original transition system $T$
```

The first step of Algorithm 11 is to translate the dLTL specification Φ into a deterministic Büchi automaton $B = (S, S_0, O, \delta_B, F)$. The second step is the construction of a product automaton P of the transition system $T = (X, \Sigma, \delta, O, o)$ and B. The product automaton $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$ is constructed as described in Definition 5.3 with the exception that the set of accepting states of P is defined as $F_P = X \times F$. The product automaton P is a nondeterministic Büchi automaton if T is nondeterministic, otherwise it is a deterministic Büchi automaton.

Each accepting run $\rho_P = (x_1, s_1)(x_2, s_2)(x_3, s_3)\ldots$ of a product automaton $P = T \otimes B$ can be projected into a trajectory $x_1x_2x_3\ldots$ of T, such that the word $o(x_1)o(x_2)o(x_3)\ldots$ is accepted by B (i.e., satisfies ϕ) and vice versa. Similar to the solution proposed in the previous section, this allows us to reduce Problem 5.1 to finding a control strategy for P.



(a) $\psi = \Box\Diamond B \wedge \Box\neg D \wedge \Box(B \Rightarrow \bigcirc(\neg B\, \mathcal{U}\, G)) \wedge \Box(B \Rightarrow \bigcirc(\neg B\, \mathcal{U}\, R))$

(b) Control automaton C

Fig. 5.8 Rabin automaton representation of the specification formula ψ (**a**) and the control automaton (**b**) from Example 5.8. For the Rabin automaton, s_0 is the initial state. There is a single pair in the accepting condition: $F = \{(G, B)\}$, where $G = \{s_1\}$, and $B = \{s_5\}$. For the control automaton C, s_0 is the initial state. The arrows between states are labeled with the states of the robot transition system depicting the memory update function. The corresponding control actions are shown in *red*. State s_5, which is not reachable from the initial state s_0, is not shown

Problem 5.3 Given a controlled Büchi product automaton $P=(S_P, S_{P0}, \Sigma, \delta_P, F_P)$, find the largest set of initial states $W_{P0} \subseteq S_{P0}$ for which there exists a control function $\pi_P : S_P \to \Sigma$ such that each run of P under strategy (W_{P0}, π_P) satisfies the Büchi accepting condition F_P.

The solution to Problem 5.3 is summarized in Algorithm 12. The main idea behind the algorithm is to first compute a subset $\overline{F}_P$ of F_P such that a visit to $\overline{F}_P$ can be enforced from $\overline{F}_P$ in a finite number of steps. Then, what remains is to compute the set of all states W_P and a control function π_P such that all runs originating from W_P in closed loop with π_P can reach $\overline{F}_P$ in a one or more steps. By the definition of $\overline{F}_P$, it holds that $\overline{F}_P \subseteq W_P$, and hence these runs satisfy the Büchi condition. To compute $\overline{F}_P$ and π_P, we first define direct and proper attractor sets of a set $S \subseteq S_P$ for a Büchi automaton P:

Definition 5.10 (*Direct attractor*) The direct attractor of a set S, denoted by $\mathsf{A}^1(S)$, is defined as the set of all $s \in S_P$ from which there can be enforced a visit to S in one step:

$$\mathsf{A}^1(S) = \{s \in S_P \mid \exists \sigma \in \Sigma, \delta_P(s, \sigma) \subseteq S\}.$$

The direct attractor set induces a strategy $\pi_P^{1,S} : \mathsf{A}^1(S) \to \Sigma$ such that

$$\delta_P(s, \pi_P^1(s)) \subseteq S.$$

Definition 5.11 (*Proper attractor*) The proper attractor of a set S, denoted by $\mathsf{A}^+(S)$, is defined as the set of all $s \in S_P$ from which there can be enforced a visit to S in one or more steps.

The proper attractor set $\mathsf{A}^+(S)$ of a set S can be computed iteratively via the converging sequence

$$\mathsf{A}^1(S) \subseteq \mathsf{A}^2(S) \subseteq \ldots,$$

where $\mathsf{A}^1(S)$ is the direct attractor of S, and

$$\mathsf{A}^{i+1}(S) = A^1(A^i(S) \cup S) \cup \mathsf{A}^i(S).$$

Intuitively, $\mathsf{A}^i(S)$ is the set from which a visit to S in at most i steps can be enforced by choosing proper controls. The *attractor strategy* $\pi_P^{+,S}$ for $\mathsf{A}^+(S)$ is defined from the direct attractor strategies computed through the converging sequence as follows:

$$\pi_P^{+,S} = \pi_P^{1,\mathsf{A}^i(S)}(s), \text{ for all} s \in \mathsf{A}^{i+1}(S) \setminus \mathsf{A}^i(S).$$

A *recurrent set* of a given set A is the set of states from which infinitely many revisits to A can be enforced. In Algorithm 12, first the recurrent set $\overline{F}_P$ of F_P is computed with an iterative process (lines 3–6). Note that we start with $\overline{F}_P = F_P$ and iteratively remove the states from which a revisit to $\overline{F}_P$ can not be guaranteed. This loop terminates after a finite number of iterations since F_P is a finite set. The

Algorithm 12 BÜCHIGAME ($P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$): Winning region $W_P \subseteq S_P$ and winning strategy π_P.

```
$\overline{F}_P = \emptyset$
$\overline{F}_P^{new} = F_P$
while $\overline{F}_P \neq \overline{F}_P^{new}$ do
    $\overline{F}_P = \overline{F}_P^{new}$
    $\overline{F}_P^{new} = \mathsf{A}^+(\overline{F}_P) \cap \overline{F}_P$
end while
$W_P = \mathsf{A}^+(\overline{F}_P)$, compute the corresponding attractor strategy $\pi_P$
```

termination guarantees $\overline{F}_P \subseteq \mathsf{A}^+(\overline{F}_P)$, and hence infinitely many revisits to $\overline{F}_P$ from $\overline{F}_P$ can be enforced. In the final step of the algorithm the proper attractor of $\overline{F}_P$ and the corresponding attractor strategy is computed. As $\overline{F}_P \subseteq \mathsf{A}^+(\overline{F}_P)$, π_P is an attractor strategy that solves the Büchi game for all $s \in W_P$.

Complexity The time complexity of Algorithm 12 is $\mathscr{O}(|S_P|^2|\Sigma|)$.

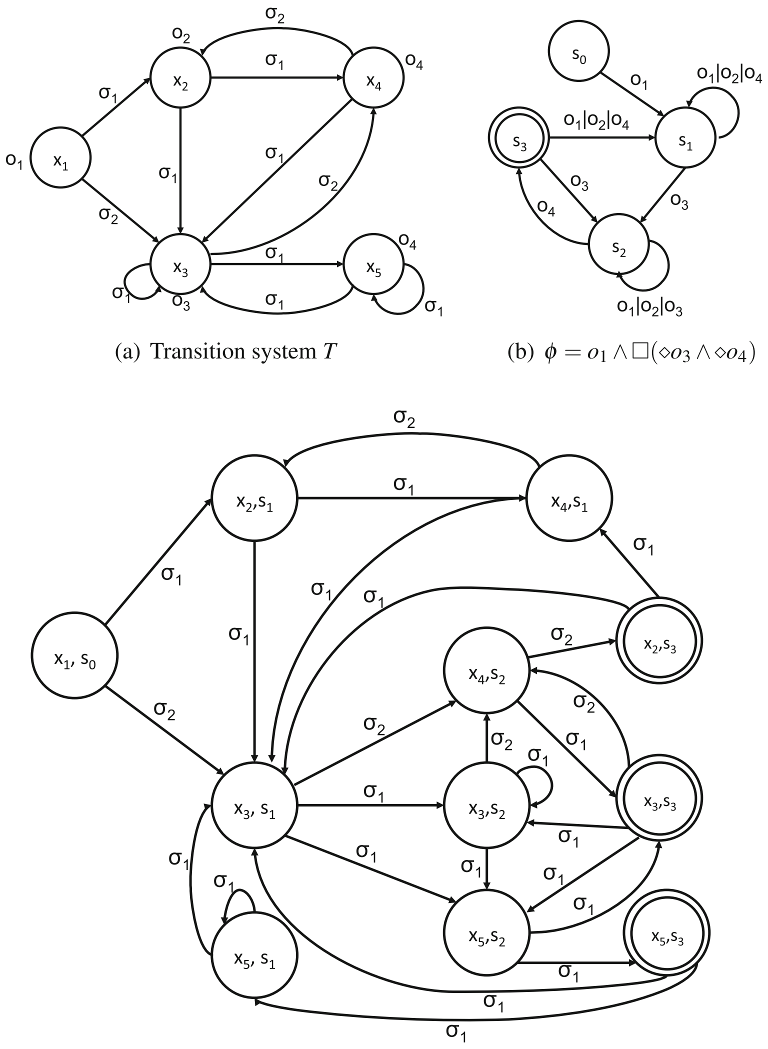
Remark 5.1 A Büchi automaton B can be interpreted as a Rabin automaton with a single pair $\{(G_1, B_1)\}$ in its accepting condition, where $G_1 = \overline{F}_P$ and $B_1 = \emptyset$. Consequently, Algorithm 10 for the Rabin game can be used for the Büchi automaton to solve Problem 5.3. In this particular case, $n = 1$ and the time complexity of Algorithm 10 is $\mathscr{O}((|S_P| + |S_P||\Sigma|)^2)$.

The final step of the dLTL control algorithm is to translate the control strategy (W_{P0}, π_P) obtained from Algorithm 12 into a control strategy (X_T^ϕ, Ω) for T, where $W_{P0} = W_P \cap S_{P0}$. As in the solution presented for LTL specifications in the previous section, although the control function π_P is memoryless, Ω is history dependent and takes the form of a feedback control automaton. The control automaton is constructed from P and π_P as in Definition 11, and the control function Ω is defined by the control automaton. Finally, the set of initial states X_T^ϕ of T is given by $\alpha(W_{P0})$, where $\alpha : S_P \to X$ is the projection from states of P to X.

Example 5.9 Consider the nondeterministic transition system T shown in Fig. 5.9a and the following LTL formula over its set of observations:

$$\phi = o_1 \wedge \Box(\Diamond o_3 \wedge \Diamond o_4).$$

We follow Algorithm 11 to find the control strategy (X_T^Φ, Ω) that solves Problem 5.1 for the transition system T and formula ϕ.



(a) Transition system T

(b) $\phi = o_1 \wedge \square(\diamond o_3 \wedge \diamond o_4)$

(c) The product of T (a) and B (b)

Fig. 5.9 Transition system (**a**), Büchi automaton (**b**), and their product (**c**) from Example 5.9. For the Büchi automaton, s_0 is the initial state and s_3 is the accepting state. For the product automaton, (x_1, s_0) is the initial state, and $\{(x_2, s_3), (x_3, s_3), (x_5, s_3)\}$ is the set of accepting states. The states that are not reachable from the non-blocking initial state (x_1, s_0) are not shown in (**c**)

We first construct a deterministic Büchi automaton B (Fig. 5.9b) that accepts the language satisfying the formula. Then, we construct the product of the system and the automaton. The product automaton P, which is shown in Fig. 5.9c, is a non-deterministic Büchi automaton since T is nondeterministic. Note that the states that are not reachable from non-blocking initial states are removed from P and are not shown in Fig. 5.9c.

To find a control strategy for P, we follow Algorithm 12. In the first iteration, $\overline{F}_P = \{(x_2, s_3), (x_3, s_3), (x_5, s_3)\}$ (F_P) and we compute the proper attractor of F_P as follows:

$$\begin{aligned}
\mathsf{A}^1(F_P) &= \{(x_4, s_2), (x_5, s_2)\},\\
\mathsf{A}^2(F_P) &= \{(x_3, s_1), (x_3, s_2), (x_3, s_3), (x_4, s_2), (x_5, s_2)\},\\
\mathsf{A}^3(F_P) &= \{(x_1, s_0), (x_4, s_1), (x_3, s_1), (x_3, s_2), (x_3, s_3), (x_4, s_2), (x_5, s_2)\}\\
\mathsf{A}^4(F_P) &= \{(x_2, s_1), (x_2, s_3), (x_1, s_0), (x_4, s_1), (x_3, s_1), (x_3, s_2),\\
&\qquad (x_3, s_3), (x_4, s_2), (x_5, s_2)\}
\end{aligned}$$

The sequence converges at iteration 4 and $\mathsf{A}^+(F_P) = \mathsf{A}^4(F_P)$. In the first iteration of the main loop of Algorithm 12, $\overline{F}_P^{new} = \{(x_2, s_3), (x_3, s_3)\}$, and (x_5, s_3) is eliminated. The main loop terminates after the second iteration as

$$\mathsf{A}^+(\overline{F}_P) \cap \overline{F}_P = \overline{F}_P, \text{ where } \overline{F}_P = \{(x_2, s_3), (x_3, s_3)\}.$$

As the last step of Algorithm 12, we compute $W_P = \mathsf{A}^+(\{(x_2, s_3), (x_3, s_3)\})$, and the corresponding attractor strategy as follows:

$$\begin{aligned}
&\mathsf{A}^1(\overline{F}_P) = \{(x_4, s_2)\}, && \pi_P((x_4, s_2)) = \sigma_1,\\
&\mathsf{A}^2(\overline{F}_P) = \{(x_3, s_3), (x_3, s_2), (x_3, s_1)\} \cup \mathsf{A}^1(\overline{F}_P), &&\\
&\qquad\qquad \pi_P((x_3, s_3)) = \sigma_2, \pi_P((x_3, s_2)) = \sigma_2, \pi_P((x_3, s_1)) = \sigma_2, &&\\
&\mathsf{A}^3(\overline{F}_P) = \{(x_1, s_0), (x_4, s_1)\} \cup \mathsf{A}^2(\overline{F}_P), && \pi_P((x_1, s_0)) = \sigma_2, \pi_P((x_4, s_1)) = \sigma_1,\\
&\mathsf{A}^4(\overline{F}_P) = \{(x_2, s_1), (x_2, s_3)\} \cup \mathsf{A}^3(\overline{F}_P), && \pi_P((x_2, s_1)) = \sigma_1, \pi_P((x_2, s_3)) = \sigma_1,\\
&\mathsf{A}^4(\overline{F}_P) = \mathsf{A}^5(\overline{F}_P) = \mathsf{A}^+(\overline{F}_P). &&
\end{aligned}$$

The control strategy (W_{P0}, π_P) solves Problem 5.3 for P, where $W_{P0} = \{(x_1, s_0)\}$ and π_P is as defined above. The final step is the transformation of (W_{P0}, π_P) into a control strategy (X_T^{Φ}, Ω) for T. The set of initial states is $X_T^{\Phi} = \{x_1\}$, and the feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$ (shown in Fig. 5.10), which defines the history dependent control function Ω, is constructed as in Definition 5.9, and formally defined as:

$S_C = \{s_0, s_1, s_2, s_3\}$,
$S_{C0} = \{s_0\}$,
$X = \{x_1, x_2, x_3, x_4, x_5\}$,
$\tau(s_0, x_1) = s_1, \tau(s_1, x_2) = s_1, \tau(s_1, x_3) = s_2, \tau(s_1, x_4) = s_1, \tau(s_2, x_3) = s_2,$
$\tau(s_2, x_4) = s_3,\ \tau(s_3, x_2) = s_1,\ \tau(s_3, x_3) = s_2$
$\Sigma = \{\sigma_1, \sigma_2\}$,
$\pi(s_0, x_1) = \sigma_2, \pi(s_1, x_2) = \sigma_1, \pi(s_1, x_3) = \sigma_2, \pi(s_1, x_4) = \sigma_1, \pi(s_2, x_3) =$
$\sigma_2, \pi(s_2, x_4) = \sigma_1, \pi(s_3, x_2) = \sigma_1, \pi(s_3, x_3) = \sigma_2$.

Remark 5.2 In a Büchi game over a product automaton $P = T \otimes B$, a state (x, s) is added to W_P only if there is a control strategy guaranteeing that all runs ρ_P of P originating from (x, s) satisfy that the projection of ρ_P onto S (Büchi automaton states) is an accepting run of B. The condition is necessary to guarantee that each run of T that originate from x is satisfying. While the product is a non-deterministic Büchi automaton, this condition is stronger than the Büchi acceptance: a word is accepted by a non-deterministic Büchi automaton if there exists an accepting run. In other words, it is not necessary that all runs are accepting. Due to this difference in the notion of the satisfying TS states and non-deterministic Büchi acceptance, an algorithm similar to Algorithm 11 cannot be used for non-deterministic Büchi automaton, which is illustrated in Example 5.10.

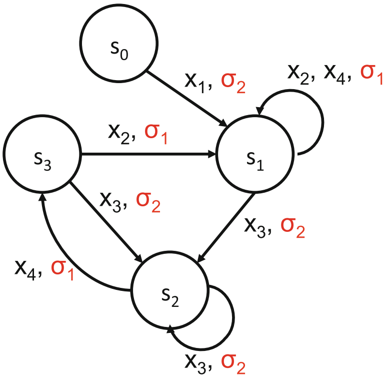
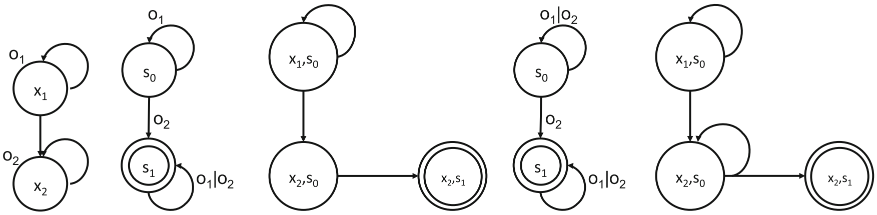


Fig. 5.10 The control automaton obtained by solving the Büchi game on the product automaton shown in Fig. 5.9c



(a) T (b) $\phi = \diamond o_2$, B_1 (c) $P_1 = T \otimes B_1$ (d) $\phi = \diamond o_2$, B_2 (e) $P_2 = T \otimes B_2$

Fig. 5.11 Transition system T (**a**), Büchi automata B_1 (**b**) and B_2 (**c**), the product of T and B_1 (**d**), and the product of T and B_2 (**e**) from Example 5.10

Example 5.10 Consider the transition system T shown in Fig. 5.11a and specification $\phi = \diamond o_2$ over its set of observations. A deterministic Büchi automaton and a non-deterministic Büchi automaton that accept the language satisfying the formula are shown in Figs. 5.11b and 5.11d, respectively. The corresponding product automata are shown in Figs. 5.11c and 5.11e. T has a single run $x_2x_2x_2\ldots$ originating from x_2 and it satisfies ϕ_1. Due to the non-determinism of T, there are multiple runs originating from x_1. The run $x_1x_1x_1\ldots$ originating from x_1 produces the word $o_1o_1o_1\ldots$ that violates the formula, and all other runs originating from x_1 satisfy the formula. Therefore, we have $X_T^{\phi} = \{x_2\}$. We can easily verify this observation by running Algorithm 12 on the product automaton P_1 with $\Sigma = \{\sigma\}$, i.e., a single control input labels all the transitions. The algorithm returns $W_P = \mathsf{A}^+(\overline{F}_P) = \{(x_2, s_0), (x_2, s_1)\}$, hence $W_{P0} = \{(x_2, s_0)\}$ and $X_T^{\phi} = \{x_2\}$.

Now, consider the product P_2 (Fig. 5.11e) of T and the non-deterministic Büchi automaton B_2 accepting the same language as B_1. It is not possible to differentiate (x_1, s_0) and (x_2, s_0) on P_2 via reachability analysis or recurrence set construction, since satisfying and violating runs originate from both (x_1, s_0) and (x_2, s_0).

5.3 Control of Transition Systems from scLTL Specifications

A solution to Problem 5.1 is found more efficiently when the specification ϕ is an scLTL formula. This is due to the simple FSA acceptance condition for scLTL formulas. The solution we present here resembles the one we presented in Sect. 5.1

for arbitrary LTL specifications. The procedure involves the construction of an FSA from the specification formula ϕ, the construction of the product automaton of the system and the FSA, solving a reachability problem on the product automaton to find a control strategy for the product automaton, and finally translation of this strategy to the transition system. While a control strategy for the product automaton was found by solving a Rabin game in Sect. 5.1 and a Büchi game in Sect. 5.2, the product of an FSA and a nondeterministic transition system is a nondeterministic finite state automaton (NFA), for which a control strategy can be found by solving a reachability problem. This step can be interpreted as finding the attractor set of the accepting states of the product automaton and the corresponding control strategy. Moreover, when T is deterministic, the product is an FSA and the largest controlled satisfying region can simply be found by traversing the graph of the product automaton starting from its set of accepting states. The procedure for determining control strategies for nondeterministic transition systems from scLTL formulas is summarized in Algorithm 13. The details are presented in the rest of this section.

Algorithm 13 SCLTL CONTROL(T, ϕ) : Control strategy (X_T^{ϕ}, Ω) such that all trajectories in $T(X_T^{\phi}, \Omega)$ satisfy scLTL formula ϕ

```
Translate $\phi$ into an FSA $A = (S, s_0, O, \delta_A, F)$
Build a product automaton $P = T \otimes R$
Solve a reachability problem on the graph of the product automaton
Map the solution to the reachability problem to a control strategy for the original transition
  system $T$
```

The first step of Algorithm 13 is to translate the scLTL specification ϕ into an FSA $A = (S, s_0, O, \delta_A, F)$. This can be done using off-the-shelf tool as discussed in Sect. 2.3. The second step is the construction of a product automaton P of the transition system $T = (X, \Sigma, \delta, O, o)$ and the FSA A. The product automaton $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$ is constructed as described in Definition 5.3 with the exception that the set of accepting states of P is defined as $F_P = X \times F$. The product automaton P is a NFA if T is nondeterministic, and it is an FSA if T is deterministic.

Each accepting run $\rho_P = (x_1, s_1)(x_2, s_2) \ldots (x_n, s_n)$ of the product automaton P can be projected into a trajectory $x_1 x_2 \ldots x_n$ of T, such that the word $o(x_1)o(x_2) \ldots o(x_n)$ is accepted by A (i.e., all words that contain the prefix $o(x_1)$ $o(x_2) \ldots o(x_n)$ satisfies ϕ) and vice versa. Analogous to the solution for arbitrary LTL specifications presented in Sect. 5.1, this allows us to reduce Problem 5.1 to finding a control strategy (W_0, π) for P, which is defined as in Definition 5.4.

Problem 5.4 Given a product NFA $P = (S_P, S_{P0}, \Sigma, \delta_P, F_P)$, find the largest set of initial states $W_{P0} \subseteq S_{P0}$ for which there exists a control function $\pi_P : S_P \to \Sigma$ such that each run of P under the strategy (W_{P0}, π_P) reaches the set of accepting states F_P.

We use W_P to denote the set of states of P from which a visit to the set of accepting states can be enforced by a control function. This set and the corresponding control strategy can easily be computed with a single attractor computation:

$$W_P = F_P \cup \mathsf{A}^+(F_P),$$

where $\mathsf{A}^+(F_P)$ is the proper attractor of F_P and π_P is the corresponding attractor strategy, which are described in Definition 5.11.

This computation results in a control strategy (W_{P0}, π_P) that solves Problem 5.4, where $W_{P0} = W_P \cap S_{P0}$. The final step of Algorithm 13 is the transformation of the control strategy (W_{P0}, π_P) for the product P into a control strategy (X_T^{Φ}, Ω) for T. The control function Ω for T is history dependent and takes the form of a feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$, which is constructed from P, T and A as described in Definition 5.9. The set of initial states X_T^{ϕ} of T is given by $\alpha(W_{P0})$, where $\alpha : S_P \rightarrow X$ is the projection from states of P to X. The control function Ω is given by C as explained in Sect. 5.1. The product automaton of T and C will have the same states as P but contains only transitions of P closed under π_P. Then, all trajectories in $T(X_T^{\phi}, \Omega)$ satisfy ϕ. Moreover, if $(x_1, s_1) \notin W_P$, then $\delta_P((x_1, s_1), \sigma) \nsubseteq W_P$ for all $\sigma \in \Sigma$, which implies that there exists a run of P that originate at (x_1, s_1) and can not reach F_P regardless of the applied control function. Therefore, in the case when ϕ is an scLTL formula, X_T^{ϕ} is the largest controlled satisfying region and the strategy (X_T^{ϕ}, Ω) obtained from Algorithm 13 is a solution to Problem 5.1.

Complexity The complexity of finding the control strategy for the product automaton P (step 3 of Algorithm 13) is $\mathscr{O}(|S_P||\Sigma|)$, since an attractor set is computed in maximum $\mathscr{O}(|S_P||\Sigma|)$ iterations.

Example 5.11 Consider the nondeterministic transition system T shown in Fig. 5.12a and the scLTL formula over its set of observations:

$$\phi = \Diamond o_4 \wedge (\neg o_3 U o_4) \wedge (\neg o_4 U o_2).$$

We follow Algorithm 13 to find the control strategy (X_T^{Φ}, Ω) that solves Problem 5.1 for transition system T and formula ϕ. We first construct an FSA A (Fig. 5.12b) that accepts the good prefixes of the formula. Then, we construct the product of the system and the FSA. The product automaton P, which is shown in Fig. 5.12c, is an NFA since T is nondeterministic. Note that the states that are not reachable from non-blocking initial states are removed from P and are not shown in Fig. 5.12c.

To find a control strategy for P, we compute the converging sequence W_P^{i*} and control function π_P:

$$\begin{array}{ll}
W_P^{0*} = \{(x_5, s_2)\}, & \pi_P((x_5, s_2)) = \sigma_1 \\
W_P^{1*} = \{(x_5, s_1)\} \cup W_P^{0*}, & \pi_P((x_5, s_1)) = \sigma_1 \\
W_P^{2*} = \{(x_2, s_0), (x_2, s_1)\} \cup W_P^{1*}, & \pi_P((x_2, s_0)) = \sigma_1, \pi_P((x_2, s_1)) = \sigma_1 \\
W_P^{3*} = \{(x_4, s_0), (x_4, s_1)\} \cup W_P^{2*}, & \pi_P((x_4, s_0)) = \sigma_1, \pi_P((x_4, s_1)) = \sigma_1 \\
W_P^{4*} = W_P^{3*}. &
\end{array}$$

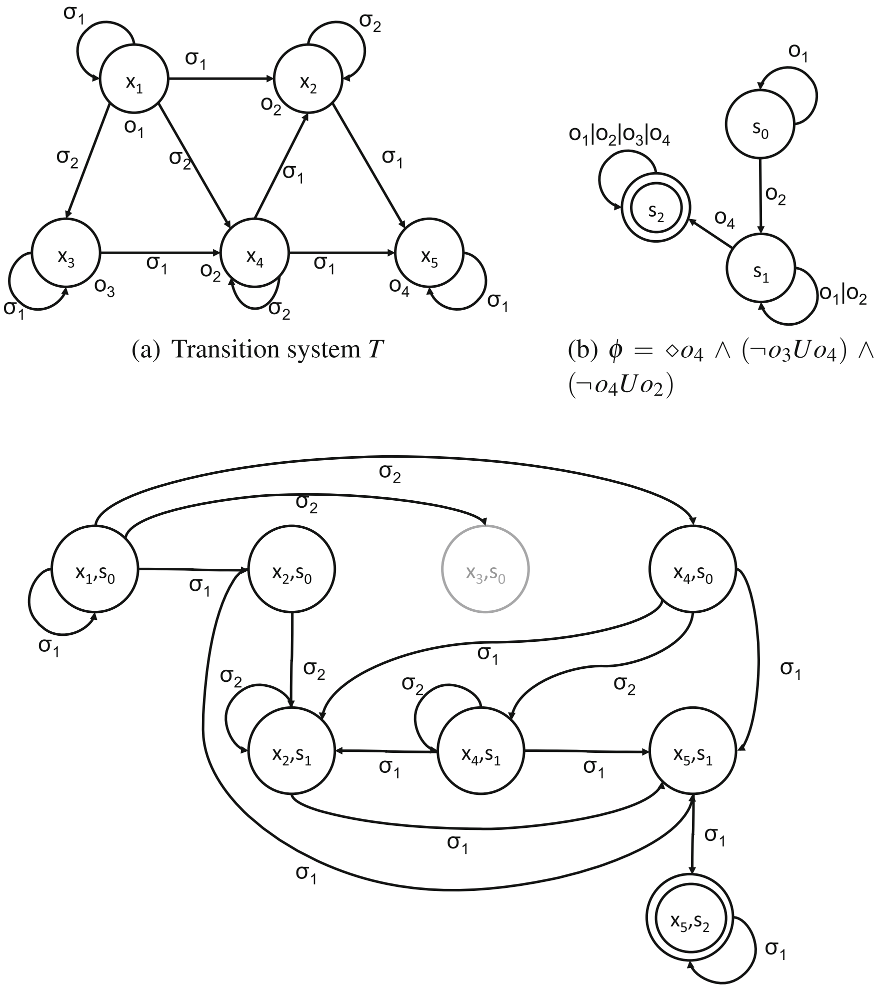
The control strategy (W_{P0}, π_P) solves Problem 5.4 for P, where $W_{P0} = \{(x_2, s_0), (x_4, s_0)\}$ and π_P is as defined above. The final step is the transformation of (W_{P0}, π_P) into a control strategy (X_T^{Φ}, Ω) for T. The set of initial states is $X_T^{\Phi} = \{x_2, x_4\}$, and the feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Sigma, \pi)$, that defines the history dependent control function Ω, is constructed as in Definition 5.9, and formally defined as:

$S_C = \{s_0, s_1, s_2\}$,
$S_{C0} = \{s_0\}$,
$X = \{x_1, x_2, x_3, x_4, x_5\}$,
$\tau(s_0, x_2) = s_1, \tau(s_0, x_4) = s_1, \tau(s_1, x_2) = s_1, \tau(s_1, x_4) = s_1, \tau(s_1, x_5) = s_2$, $\tau(s_2, x_5) = s_2$,
$\Sigma = \{\sigma_1, \sigma_2\}$,
$\pi(s_0, x_2) = \sigma_1, \pi(s_0, x_4) = \sigma_1, \pi(s_1, x_2) = \sigma_1, \pi(s_1, x_4) = \sigma_1, \pi(s_1, x_5) = \sigma_1, \pi(s_2, x_5) = \sigma_1$.

5.4 Notes

We presented a complete treatment of the LTL control problem for a finite transition system. If the transition system is deterministic, the problem can be solved through model-checking-based techniques (see Chap. 3). Indeed, an off-the-shelf model checker can be used to model check the system against the negation of the formula. If the negation of the formula is not satisfied at a state, i.e., there exists a run violating the negation of the formula, then it is returned as a certificate of violation. This run, which satisfies the formula, can be enforced in the deterministic transition system by choosing appropriate controls at the states in the run. This approach was used in [105] to develop a conservative solution to an LTL control problem for a continuous-time, continuous space linear system.

In this chapter, we focused on the case when the transition system is nondeterministic. We showed that, in the most general case, the problem can be reduced to a Rabin game [146]. There are various approaches to solve Rabin games [55, 90,



(a) Transition system T

(b) $\phi = \diamond o_4 \wedge (\neg o_3 U o_4) \wedge (\neg o_4 U o_2)$

(c) The product of T (a) and FSA (b)

Fig. 5.12 Transition system (**a**), the FSA (**b**), and the product of them (**c**) from Example 5.11. For the FSA, s_0 is the initial state and s_2 is the accepting state. For the product automaton, $\{(x_1, s_0), (x_2, s_0), (x_3, s_0), (x_4, s_0)\}$ is the set of initial states, and (x_5, s_2) is the accepting state. The blocking state (x_3, s_0) that is reachable from a non-blocking initial state (x_1, s_0) is shown in *grey*

141]. The solution we presented is based on [90]. The Rabin game based approach to the control problem from this chapter is based on [170]. For the particular case when the LTL formula can be translated to a deterministic Büchi automaton, we showed that the control problem reduced to a Büchi game [38], for which efficient solutions exist [167]. A treatment of the control problem for this case can be found in [104]. Finally, if the specification is given in the syntactically co-safe fragment of LTL, called scLTL [156], then the solution reduced to a reachability problem, for which we propose an efficient algorithm. In all three cases mentioned above, the control strategy for the original transition system takes the form of a feedback control automaton, which is easy to interpret and implement.

For simplicity of exposition, we only consider synthesis from LTL specifications. Readers interested in CTL and CTL* specifications are referred to [9, 57, 92]. There has also been some interest in combining optimality with correctness in formal synthesis. Examples include optimal LTL control for transition systems [51, 161, 171] and Markov decision processes [40, 52, 160], and optimization problems with costs formulated using the quantitative semantics of logics such as signal temporal logic (STL) and metric temporal logic (MTL) [12, 19, 54, 59, 94, 95, 107, 176].

Part III
Analysis and Control of Discrete-Time Dynamical Systems

Chapter 6
Discrete-Time Dynamical Systems

In this chapter, we introduce the two classes of discrete-time dynamical systems that we will focus on in the rest of the book: piecewise affine control systems with polytopic parameter uncertainties and switched linear systems. As particular instantiations of the first class, we define autonomous systems, fixed parameter systems, and combinations of the above. By generalizing the ideas already presented in Sect. 1.2 and Example 1.8, we define embeddings of such systems into (infinite) transition systems. This enables formal definitions for their semantics and the use of abstractions to map analysis and control problems for such systems to verification and synthesis problems for finite transition systems, which were treated in Part II.

6.1 Piecewise Affine Systems

Let L be a finite index set and $\mathbf{X}_l$, $l \in L$ be a set of open, full dimensional polytopes[1] in $\mathbb{R}^N$, such that $\mathbf{X}_{l_1} \cap \mathbf{X}_{l_2} = \emptyset$ for all $l_1, l_2 \in L$, where $l_1 \neq l_2$.

Definition 6.1 (*PWA Control System*) A discrete-time, uncertain-parameter piecewise affine (PWA) control system $\mathscr{W}$ over $\mathbf{X} = \bigcup_{l \in L} \mathbf{X}_l$ is defined as:

$$\mathscr{W} \;:\; x(k+1) = A_l x(k) + B_l u(k) + c_l,\;\; x(k) \in \mathbf{X}_l,\;\; u(k) \in \mathbf{U},\;\; l \in L \tag{6.1}$$

where, at each time step $k = 0, 1, \ldots, x(k) \in \mathbb{R}^N$ is the state of the system and $u(k)$ is the input restricted to a polytopic set $\mathbf{U} \subset \mathbb{R}^M$. Matrices $A_l \in \mathbf{P}_l^A$, $B_l \in \mathbb{R}^{N \times M}$, $c_l \in \mathbf{P}_l^c$ are the system parameters for mode $l \in L$, where parameters A_l and c_l for each $l \in L$ are restricted to polytopic sets $\mathbf{P}_l^A \subset \mathbb{R}^{N \times N}$ and $\mathbf{P}_l^c \subset \mathbb{R}^N$, respectively.

Note that, for technical reasons to become clear later, we assume that there is no parameter uncertainty in the control parameter matrix B_l.

[1] In the rest of the book, we assume polytopes are open and full dimensional, unless specifically mentioned otherwise. See Sect. A.1. This assumption is discussed in Sect. 6.3.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_6

In the following, we define several subclasses of PWA systems used later in the book, which are obtained by restricting Definition 6.1.

Definition 6.2 (*Fixed Parameter PWA Control System*) A fixed-parameter PWA control system is a PWA system (Definition 6.1) where, for all modes $l \in L$, the sets $\mathbf{P}_l^A$ and $\mathbf{P}_l^c$ are singletons, i.e., $A_l \in \mathbb{R}^{N \times N}$ and $c_l \in \mathbb{R}^N$.

In the particular case when $\mathbf{P}_l^c = \emptyset$, for all $l \in L$, the system from Definition 6.2 is called a *fixed parameter piecewise linear control system*. In other words, such a system is described by $x(k+1) = A_l x(k) + B_l u(k)$, $x(k) \in \mathbf{X}_l$, $u(k) \in \mathbf{U}$, $l \in L$.

Definition 6.3 (*Autonomous PWA System*) An autonomous PWA system is a PWA system (Definition 6.1) with input $u(k) = 0$ for all time steps $k = 0, 1, \ldots$ i.e.,

$$\mathscr{W} \;:\; x(k+1) = A_l x(k) + c_l,\;\; x(k) \in \mathbf{X}_l,\;\; l \in L. \tag{6.2}$$

Definition 6.4 (*Autonomous Fixed Parameter PWA System*) An autonomous, fixed-parameter PWA system is an autonomous PWA system (Definition 6.3) where, for all modes $l \in L$, the sets $\mathbf{P}_l^A$ and $\mathbf{P}_l^c$ are singletons, i.e., $A_l \in \mathbb{R}^{N \times N}$ and $c_l \in \mathbb{R}^N$.

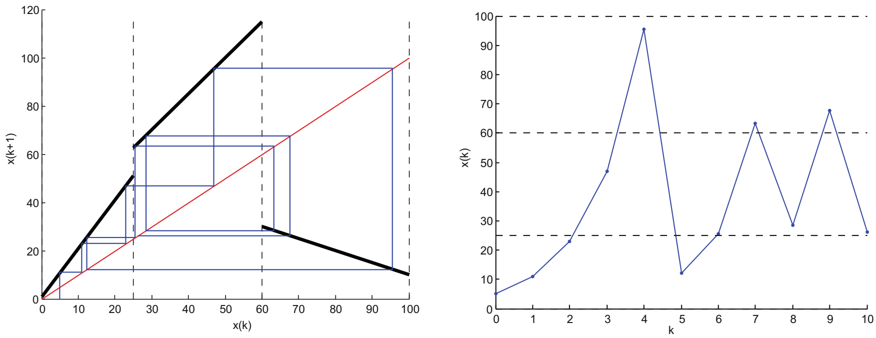
In the particular case when $\mathbf{P}_l^c = \emptyset$, for all $l \in L$, the system from Definition 6.4 is called a *fixed parameter piecewise linear system*. In other words, such a system is described by $x(k+1) = A_l x(k)$, $x(k) \in \mathbf{X}_l$, $l \in L$.

Definition 6.5 (*Autonomous Additive Uncertainty PWA System*) An autonomous, additive uncertainty PWA system is an autonomous PWA system (Definition 6.3) where, for all modes $l \in L$, the set $\mathbf{P}_l^A$ is a singleton, i.e., $A_l \in \mathbb{R}^{N \times N}$ and $\mathbf{P}_l^c \subset \mathbb{R}^N$ is a polytopic parameter set.

In other words, only the vector component c_l is uncertain in Definition 6.5, while the matrix component A_l is fixed. Both the vector and matrix parameter components are fixed in Definition 6.4.

System $\mathscr{W}$ evolves along different affine dynamics in different regions of the continuous state space $\mathbf{X}$. When $\mathscr{W}$ is in a state $x(k) \in \mathbf{X}_l$ for some $l \in L$, we say that the system is in *mode* $l \in L$. Then, the next visited state $x(k+1)$ is computed according to the affine map of Eq. (6.1) with parameters A_l and c_l, specifying the dynamics of $\mathscr{W}$ in mode l. Starting from initial conditions $x(0) \in \mathbf{X}_{l_0}$ for some $l_0 \in L$ a trajectory of system $\mathscr{W}$ can be obtained by the following (numerical simulation) procedure:

i. Start from initial conditions $x(0) \in \mathbf{X}_{l_0}$, where the system is in mode l_0.
ii. Select parameters $A \in \mathbf{P}_{l_0}^A$ and $c \in \mathbf{P}_{l_0}^c$ from the allowed parameter sets.
iii. Select an input $u \in \mathbf{U}$ from the allowed input set.
iv. Apply the affine map of Definition 6.1 to compute the next state $x(1) = Ax(0) + B_{l_0} u + c$.
v. Find the mode $l_1 \in L$ of $\mathscr{W}$ such that $x(1) \in \mathbf{X}_{l_1}$.
vi. Repeat this procedure iteratively for each subsequent step.



(a) Cobweb diagram of PWA system $\mathscr{W}$.

(b) A trajectory produced by $\mathscr{W}$.

Fig. 6.1 A one dimensional PWA system is defined by the three regions of different dynamics, separated by *dashed lines* in (**a**). The parameters of the system in each mode are represented by the *black lines* in (**a**). Applying the PWA map iteratively is represented by the cobweb diagram from (**a**) and generates the trajectory of the system shown in (**b**). See Example 6.1 for additional details

The operating regions $\mathbf{X}_l, l \in L$ from Definition 6.1 of a PWA system are also considered as regions of interests of system $\mathscr{W}$. As we are only interested in trajectories of system $\mathscr{W}$ evolving in $\mathbf{X}$, we define an additional mode Out with trivial dynamics $x(k+1) = x(k)$ and region $\mathbf{X}_{\text{Out}} = \mathbb{R}^N \setminus \mathbf{X}$. Informally, a trajectory $w_{\mathbf{X}} = w_{\mathbf{X}}(1)w_{\mathbf{X}}(2)\ldots$ produces a word $w_L = w_L(1)w_L(2)\ldots$ such that $w_L(i)$ is the index of the region visited by state $w_{\mathbf{X}}(i)$, i.e., $w_{\mathbf{X}}(i) \in \mathbf{X}_{w_L(i)}$, and $w_L(i)$ is Out if $w_{\mathbf{X}}(i) \notin \mathbf{X}$. For example, trajectory $x(0)x(1)x(2)\ldots$ satisfying $x(0), x(1) \in \mathbf{X}_{l_1}$ and $x(2) \in \mathbf{X}_{l_2}$ for some $l_1, l_2 \in L$ produces word $l_1 l_1 l_2 \ldots$. After a short example, we formalize the semantics of PWA trajectories and their satisfaction of LTL formulas through an embedding into a transition system, which generalizes the one already introduced in Sect. 1.2.

Example 6.1 We define the fixed-parameter, autonomous, one dimensional ($N = 1$) piecewise affine system $\mathscr{W}$ shown schematically in Fig. 6.1a, which has three different modes ($L = \{1, 2, 3\}$). Regions $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are open, full dimensional polytopes in $\mathbb{R}^1$, defined in V-representation (see Definition A.5) as the interiors of the convex hulls (Definition A.3) $\mathbf{X}_1 = \text{int}(\text{hull}(\{1, 25\}))$, $\mathbf{X}_2 = \text{int}(\text{hull}(\{25, 60\}))$ and $\mathbf{X}_3 = \text{int}(\text{hull}(\{60, 100\})$ (i.e., $V(\mathbf{X}_1) = \{1, 25\}$, $V(\mathbf{X}_2) = \{25, 60\}$ and $V(\mathbf{X}_3) = \{60, 100\}$ are the sets of vertices of closures of polytopes $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$, respectively). These polytopes define the regions of the state space of $\mathscr{W}$ where the system operates under different parameters (i.e., $\mathscr{W}$ is in a different mode in each region). They are represented in Fig. 6.1a, b by dashed lines, partitioning the state space $\mathbf{X} = \mathbf{X}_1 \cup \mathbf{X}_2 \cup \mathbf{X}_3$.

The parameters of system $\mathscr{W}$ are defined as $A_1 = 2$, $b_1 = 1$, $A_2 = 1.5$, $b_2 = 25$ and $A_3 = -0.5$, $b_3 = 60$ and are represented as thick black lines in Fig. 6.1a.

Selecting initial conditions $x(0) = 5$ allows us to generate a trajectory $w_{\mathbf{X}}$ of $\mathscr{W}$ by applying the update map iteratively (the map update is represented by the cobweb diagram of Fig. 6.1a). The initial state and the following 10 steps $w_{\mathbf{X}}(1) \ldots w_{\mathbf{X}}(11) = x(0) \ldots x(10)$ of this trajectory are represented in Fig. 6.1b. It is easy to see that in states $x(0), x(1), x(2), x(5) \in \mathbf{X}_1$, $x(3), x(6), x(8), x(10) \in \mathbf{X}_2$ and $x(4), x(7), x(9) \in \mathbf{X}_3$ the system is in mode 1,2 and 3, respectively. Then, the word produced by this trajectory is w_L where, for the fragment $w_L(1) \ldots w_L(11)$, we have $w_L(1) = w_L(2) = w_L(3) = w_L(6) = 1, w_L(4) = w_L(7) = w_L(9) = w_L(11) = 2$ and $w_L(5) = w_L(8) = w_L(11) = 3$ (i.e., $w_L = 1\,1\,1\,2\,3\,1\,2\,3\,2\,3\,2\,\ldots$).

Definition 6.6 (*Embedding Transition System for $\mathscr{W}$*) The embedding for PWA control system $\mathscr{W}$ (Definition 6.1) is a transition system (Definition 1.1) $T_{\mathscr{W}} = (X_{\mathscr{W}}, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W}}, O_{\mathscr{W}}, o_{\mathscr{W}})$ with:

- $X_{\mathscr{W}} = \mathbb{R}^n$,
- $\Sigma_{\mathscr{W}} = \mathbf{U}$,
- $\delta_{\mathscr{W}}(x, u) = A_l x + B_l u + c_l$, if $x \in \mathbf{X}_l$,
- $O_{\mathscr{W}} = L \cup \{\text{Out}\}$,
- $o_{\mathscr{W}}(x) = l$ if and only if there exists $l \in L$ such that $x \in \mathbf{X}_l$ and $o_{\mathscr{W}}(x) = \text{Out}$ otherwise.

The embedding transition system from Definition 6.6 has an infinite number of states and inputs and is non-blocking. Furthermore, for the general class of uncertain-parameter PWA systems, the embedding transition system is non-deterministic. Indeed, given state $x \in \mathbf{X}$ and input $u \in \mathbf{U}$, multiple states can be reached in a single step through the dynamics defined in Definition 6.1, depending on the choice of parameters $A_l \in \mathbf{P}_l^A$ and $c_l \in \mathbf{P}_l^c$—the possible non-deterministic next-state choices are captured in the set $\delta_{\mathscr{W}}(x, u)$. In contrast, a fixed-parameter PWA system $\mathscr{W}$ (Definition 6.2) leads to a deterministic embedding transition system, since for each state $x \in \mathbf{X}$ and input $u \in \mathbf{U}$, only a single state $x' = A_l x + B_l u + c_l$ satisfies the dynamics of $\mathscr{W}$ and, therefore, $\delta_{\mathscr{W}}(x, u)$ is a singleton. Since the other PWA subclasses from Definitions 6.3, 6.4, and 6.5 are particular cases of the general PWA from Definition 6.1, the embedding $T_{\mathscr{W}}$ defined above applies with minor and obvious adjustments. For example, for the autonomous PWA systems from Definitions 6.3 and 6.5, the embeddings $T_{\mathscr{W}}$ are non-deterministic transition systems with no inputs, while for the PWA from Definition 6.4, the embedding is a deterministic transition system with no inputs.

Only infinite words are produced by the embedding transition system for each of the system classes defined above and, therefore, LTL formulas over $L \cup \{\text{Out}\}$ can be interpreted over such words, leading to the following definition:

Definition 6.7 (*LTL satisfaction for $\mathscr{W}$*) Trajectories of a PWA system $\mathscr{W}$ (Definition 6.1) originating in a polytope $\mathbf{X}_0 \subseteq \mathbf{X}$ satisfy formula ϕ if and only if $T_{\mathscr{W}}(\mathbf{X}_0)$ satisfies ϕ (according to Definition 3.1).

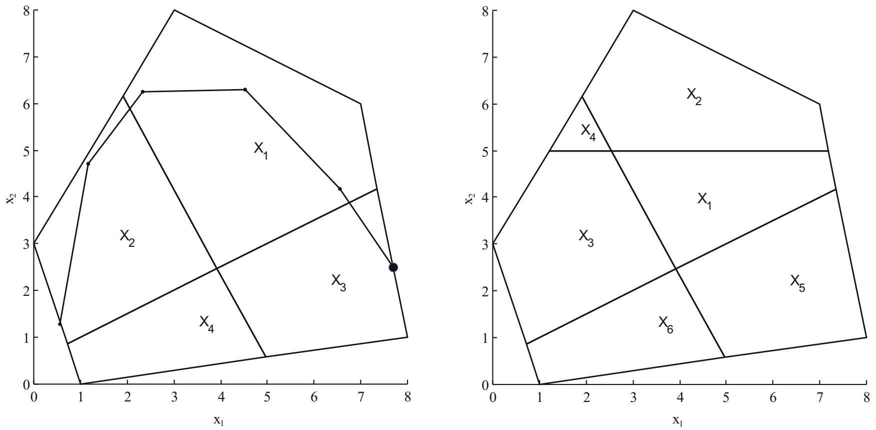


Fig. 6.2 A two dimensional PWA system $\mathscr{W}$ is defined by the four regions of different dynamics but trajectories of the system might leave the defined state space. In **a**, such a trajectory is shown evolving along the state space $\mathbf{X}$ of the system, where the initial state and subsequently visited states are represented by *circles*. The *larger circle* at the boundary of the outer polytope represents all states of index larger than 5. To formulate specifications over linear predicates that do not observe the initial set of polytopes, additional partitioning of the state space might be necessary as in (**b**). See Example 6.2 for additional details

Example 6.2 We define the two dimensional ($N = 2$) autonomous fixed-parameter PWA system $\mathscr{W}$ (Definition 6.4) shown schematically in Fig. 6.2a, which has four different modes ($L = \{1, \ldots, 4\}$). The state space of $\mathscr{W}$ is $\mathbf{X} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_4$, where $\mathbf{X}_1, \ldots, \mathbf{X}_4$ are open, full dimensional polytopes determined by cutting hull({[1, 0], [8, 1], [7, 6], [3, 8], [0, 3]}) with hyperplanes $[1 - 2]x = -1$ and $[-1.83 - 1]x = -9.65$ (Fig. 6.2a). The parameters of the system in each mode are defined as

$$\begin{aligned} A_1 &= \begin{bmatrix} 0.55 & -0.5 \\ 0.7 & 0.65 \end{bmatrix}, \; c_1 = \begin{bmatrix} 3 \\ -1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0.35 & -0.5 \\ 0.5 & 0.15 \end{bmatrix}, \; c_2 = \begin{bmatrix} 2.5 \\ 0 \end{bmatrix}, \\ A_3 &= \begin{bmatrix} 0.95 & -0.5 \\ 0.5 & 0.65 \end{bmatrix}, \; c_3 = \begin{bmatrix} 0.5 \\ -1.3 \end{bmatrix}, \quad A_4 = \begin{bmatrix} 0.95 & -0.5 \\ 0.5 & 0.65 \end{bmatrix}, \; c_4 = \begin{bmatrix} 3 \\ 0 \end{bmatrix}. \end{aligned} \tag{6.3}$$

A trajectory $w_{X_{\mathscr{W}}} = w_{X_{\mathscr{W}}}(1)w_{X_{\mathscr{W}}}(2)\ldots$ of $\mathscr{W}$ is generated, starting from initial conditions $w_{X_{\mathscr{W}}}(1) = x(0) = [7.7, 2.5]$. After five steps, the trajectory exists $\mathbf{X}$ (i.e., $w_{X_{\mathscr{W}}}(6) = x(5) = [0.5489, 1.2808] \notin \mathbf{X}$). Since PWA systems were defined with trivial dynamics in mode Out (i.e., when a state outside the defined state space is visited) the trajectory remains in state $x(5)$ for all future times (i.e., for $k = 6, 7, 8, \ldots$ we have $w_{X_{\mathscr{W}}}(k) = x(5)$). The word produced by trajectory $w_{X_{\mathscr{W}}}$ is $w_{O_{\mathscr{W}}} = w_{O_{\mathscr{W}}}(1)w_{O_{\mathscr{W}}}(2)\ldots$, where $w_{O_{\mathscr{W}}}(1) = 3$, $w_{O_{\mathscr{W}}}(2) = w_{O_{\mathscr{W}}}(3) = w_{O_{\mathscr{W}}}(4) = 1$, $w_{O_{\mathscr{W}}}(5) = 2$ and $w_{O_{\mathscr{W}}}(k) = \text{Out}$ for $k = 6, 7, 8, \ldots$ (i.e., $w_{O_{\mathscr{W}}} = 3\,1\,1\,1\,2\,\text{Out}\,\text{Out}\ldots$).

If we are interested in behaviors of the PWA system $\mathscr{W}$ where the second component of the state x reaches values above 5, we need to further partition the state of the system using hyperplane $[0, 1]x = 5$. This results in a system with 6 polytopic regions ($L = \{1, \ldots, 6\}$) denoted by $\mathbf{X}_1, \ldots, \mathbf{X}_6$ (see Fig. 6.2b). To specify that all trajectories eventually visit a state where the second component of x has a value above 5, we write the LTL formula $\Diamond(2 \vee 4)$. In other words, we require that trajectories eventually visit regions $\mathbf{X}_2$ or $\mathbf{X}_4$, where the above specification is satisfied.

6.2 Switched Linear Systems

Definition 6.8 (*Switched Linear System*) A discrete-time switched linear system over $\mathbf{X} \subset \mathbb{R}^N$ is defined as

$$\mathscr{S} \ : \ x(k+1) = A_{\gamma_k} x(k), \quad \gamma_k \in \Gamma, \tag{6.4}$$

where, at each time step $k = 0, 1, \ldots, x(k) \in \mathbb{R}^N$ is the state of the system, γ_k is the input that selects the active subsystem from a finite index set Γ, and $A_\gamma \in \mathbb{R}^{N \times N}$, for all $\gamma \in \Gamma$.

System $\mathscr{S}$ evolves along different linear dynamics, depending on the chosen value of γ_k from Γ at time k. Similar to the terminology for system $\mathscr{W}$ defined above, when $\mathscr{S}$ evolves along dynamics γ, we say that the system is in *mode* $\gamma \in \Gamma$. Starting from initial conditions $x(0) \in \mathbf{X}$ and initial mode $\gamma_0 \in \Gamma$, given a function $\gamma : \{0, 1, 2, \ldots\} \rightarrow \Gamma$, a trajectory of system $\mathscr{S}$ can be obtained by the following (numerical simulation) procedure:

i. Start from initial conditions $x(0) \in \mathbf{X}$ and initial mode $\gamma_0 \in \Gamma$.
ii. Apply the linear map from Eq. (6.4) to compute the next state $x(1) = A_{\gamma_0} x(0)$.
iii. Update the mode γ_1 according to function γ.
iv. Repeat this procedure iteratively for each subsequent step.

We are interested in studying trajectories of system $\mathscr{S}$ with respect to a finite set of semi linear sets $\mathbf{X}_l, l \in L$, where $\mathbf{X}_{l_1} \cap \mathbf{X}_{l_2} = \emptyset$, for any $l_1 \neq l_2$, and $\mathbf{X} = \cup_{l \in L} \mathbf{X}_l$ (see Appendix A.4 and Example 10.1). Informally, similar to the PWA system $\mathscr{W}$ presented above, a trajectory $w_\mathbf{X} = w_\mathbf{X}(1) w_\mathbf{X}(2) \ldots$ produces a word $w_L = w_L(1) w_L(2) \ldots$ such that $w_L(i)$ is the index of the region visited by state $w_\mathbf{X}(i)$, i.e., $w_\mathbf{X}(i) \in \mathbf{X}_{w_L(i)}$. As it will become clear in Chap. 10, because of the particular problem of interest, the trajectories of $\mathscr{S}$ always stay inside $\mathbf{X}$, and the extra mode Out is not necessary in this case. Also, as in Sect. 6.1, we informally think of $\mathbf{X}_l, l \in L$ as a partition of $\mathbf{X}$, and ignore behaviors on the boundaries of $\mathbf{X}_l$. This assumption is discussed in Sect. 6.3.

Definition 6.9 (*Embedding Transition System for $\mathscr{S}$*) The embedding for the switched control system $\mathscr{S}$ (Definition 6.8) is a transition system $T_{\mathscr{S}} = (X_{\mathscr{S}}, \Sigma_{\mathscr{S}}, \delta_{\mathscr{S}}, O_{\mathscr{S}}, o_{\mathscr{S}})$ with:

- $X_{\mathscr{S}} = \mathbf{X}$,
- $\Sigma_{\mathscr{S}} = \Gamma$,
- $\delta_{\mathscr{S}}(x, \gamma) = A_\gamma x$,
- $O_{\mathscr{S}} = L$,
- $o_{\mathscr{S}}(x) = l, l \in L$ if and only if $x \in \mathbf{X}_l$.

The embedding transition system from Definition 6.9 has an infinite number of states and finitely many inputs. It is deterministic and non-blocking. It produces infinite words, and, therefore, LTL formulas over L can be interpreted over such words, leading to the following definition:

Definition 6.10 (*LTL satisfaction for $\mathscr{S}$*) Trajectories of a switched system $\mathscr{S}$ (Definition 6.8) originating in a region $\mathbf{X}_0 \subseteq \mathbf{X}$ satisfy LTL formula ϕ over L if and only if $T_{\mathscr{S}}(\mathbf{X}_0)$ satisfies ϕ (according to Definition 3.1).

Similar to the PWA system $\mathscr{W}$ treated in Sect. 6.1, we can define different types of embeddings for $\mathscr{S}$ as particular cases of the one defined above. With particular relevance to the verification Problem 10.2 treated in Chap. 10, an autonomous embedding transition system that captures the behavior of $\mathscr{S}$ under all possible switchings can be defined as $T^A_{\mathscr{S}} = (X_{\mathscr{S}}, \delta^A_{\mathscr{S}}, O_{\mathscr{S}}, o_{\mathscr{S}})$, where $X_{\mathscr{S}}$, $O_{\mathscr{S}}$, and $o_{\mathscr{S}}$ are as defined above and $\delta^A_{\mathscr{S}}(x) = \{A_\gamma x,\ \gamma \in \Gamma\}$.

6.3 Notes

Piecewise affine systems (PWA), i.e., systems that evolve along different discrete-time affine dynamics in different polytopic regions of the (continuous) state space are widely used as models in many areas. They can approximate nonlinear dynamics with arbitrary accuracy and are equivalent with several other classes of systems, including hybrid systems [82]. In addition, there exist efficient techniques for the identification of such models from experimental data, which include Bayesian methods, bounded-error procedures, clustering-based methods, mixed-integer programming, and algebraic geometric methods (see [64, 97] for a review).

We made some simplifying assumptions in our definition of the PWA system $\mathscr{W}$ and its embedding (Definitions 6.1 and 6.6), which is inspired from [5, 7, 138, 163] (see also [72, 74, 105, 106, 162, 180, 184, 185]). First, we defined the system on a set of open full dimensional polytopes, thus ignoring states where the dynamics are ambiguous (states on the boundaries between regions). This is enough for practical purposes, since only sets of measure zero are disregarded and it is unreasonable to assume that equality constraints can be detected in real-world applications. Trajectories starting and remaining in such sets are therefore of no interest. Trajectories starting in the interior of full-dimensional polytopes also cannot “vanish” in such zero-measure sets unless the dynamics of the system satisfy some special conditions, which are easy to derive but omitted. Furthermore, if such sets are of interest, the results presented throughout the book can be extended to more general case where the state space is partitioned into polytopes with some of the facets removed. In particular, facets are simply lower dimensional polytopes and the results can be extended by induction. Second, the semantics is defined over the polytopes $\mathbf{X}_l$, which are given a priori. However, arbitrary linear inequalities can be accommodated by including additional polytopes (as long as the region $l \in L$ visited at each step can be observed), in which case the system will have the same dynamics in several modes.

Switched systems have been extensively studied for more than fifty years. They have numerous applications in mechanical systems, automotive industry, power systems, aircraft and traffic control. Switched linear systems, as defined in this chapter, attracted most of the attention. Existing works focus on analysis of stability, controllability, reachability, and observability. Excellent overviews can be found in [125, 159]. Note that, for switched linear systems $\mathscr{S}$, we make the same simplifying assumptions as for the piecewise affine systems $\mathscr{W}$, and the limitations induced by these assumptions are similar.

Chapter 7
Largest Satisfying Region

In this chapter, we develop a procedure that attempts to find the largest set of initial states from which an autonomous PWA system (Definition 6.3) satisfies an LTL formula over the set labeling the polytopes in its definition. The same problem was considered in Chap. 4 for a finite transition system and an LTL formula over its set of observations. Several methods were presented to find an exact solution to this problem. As expected, since PWA systems have infinitely many states, we are only able to find a subset of the largest satisfying region in this chapter. We formulate the problem for the general case of autonomous PWA systems with uncertain parameters, and we show that more efficient solutions can be found for the particular cases of autonomous PWA systems with fixed parameters and additive uncertainties. The problem that we consider in this chapter can be formally stated as follows:

Problem 7.1 (*Largest Satisfying Region for PWA Systems*) Given an autonomous PWA system $\mathscr{W}$ (Definition 6.3) and an LTL formula ϕ over $L \cup \{\text{Out}\}$, find the largest set of initial states from which all trajectories of $\mathscr{W}$ satisfy ϕ.

From Definition 6.7, solving Problem 7.1 involves working with the infinite embedding transition system $T_{\mathscr{W}}$ (Definition 6.6) and formula ϕ. In Chap. 3, we described LTL model checking as an algorithmic procedure for deciding whether a finite transition systems satisfies an LTL formula. Then, in Sect. 4.1, we used model checking to develop an analysis procedure for finite transition systems (Algorithm 3). Since the embedding $T_{\mathscr{W}}$ from Definition 6.6 is infinite, neither model checking, nor the analysis procedure from Algorithm 3 can be applied directly to solve Problem 7.1.

In Chap. 4, we also developed several methods for the analysis of potentially large transition systems through the construction and refinement of their quotients. In the following sections, we show that this theory can be extended to infinite transition systems such as $T_{\mathscr{W}}$, in order to address Problem 7.1.

First, we consider autonomous, fixed-parameter PWA systems (Definition 6.4) and autonomous, additive-uncertainty systems (Definition 6.5). For these systems, we show that the quotient construction and refinement procedures from Chap. 4

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_7

are implementable, and use them to solve Problem 7.1 in Sect. 7.1. For general autonomous PWA systems with uncertain parameters (Definition 6.3), we develop a conservative procedure in Sect. 7.2.

7.1 PWA Systems with Fixed and Additive Uncertain Parameters

The following discussion applies to autonomous PWA systems with additive parameter uncertainty (Definition 6.5). All the results automatically apply to the subclass of autonomous fixed-parameter PWA systems (Definition 6.4). We will discuss the differences as appropriate.

We describe the construction of quotient $T_{\mathscr{W}}/_{\sim} = (X_{\mathscr{W}}/_{\sim}, \delta_{\mathscr{W},\sim}, O_{\mathscr{W}}, o_{\mathscr{W},\sim})$ through the construction of its sets of states and observations, and observation and transition maps. From the definition of the observational equivalence relation $\sim$ (Definition 1.2), induced by observation map $o_{\mathscr{W}}$ of $T_{\mathscr{W}}$ (Definition 6.6) and the definition of the quotient $T_{\mathscr{W}}/_{\sim}$ (Definition 1.3), the set of states $X_{\mathscr{W}}/_{\sim}$ of quotient $T_{\mathscr{W}}/_{\sim}$ is simply the set of observations $X_{\mathscr{W}}/_{\sim} = O_{\mathscr{W}} = L \bigcup \{\text{Out}\}$ of $T_{\mathscr{W}}/_{\sim}$, which is inherited from $T_{\mathscr{W}}$, and the observation map is identity. Given a state $l \in X_{\mathscr{W}}/_{\sim}$, where $l \neq$ Out, the set of all equivalent states from l is

$$con(l) = \mathbf{X}_l. \tag{7.1}$$

In other words, each equivalence class is a polytope from the PWA system definition (Definition 6.1), while the explicit representation of the set $con(\text{Out}) = \mathbb{R}^N \setminus \mathbf{X}$ is not required for our methods.

In order to complete the construction of quotient $T_{\mathscr{W}}/_{\sim}$, we need to compute the transition function $\delta_{\mathscr{W},\sim}$. In Sect. 1.3, we showed that through Eq. (1.7), transitions of the quotient $T_{\mathscr{W}}/_{\sim}$ can be found by computing the set of successors of a region in $T_{\mathscr{W}}$ using the $Post()$ operation defined in Eq. (1.4)—given states $l_1, l_2 \in X_{\mathscr{W}}/_{\sim}$, there exists a transition from l_1 to l_2 (i.e., $l_2 \in \delta_{\mathscr{W},\sim}(l_1)$) if and only if the intersection $Post(con(l_1)) \cap con(l_2)$ is non-empty. From the computation of the set of equivalent states $con(l)$ for an equivalence class $l \in X_{\mathscr{W}}/_{\sim}$ given in Eq. (7.1), checking if a transition between states $l_1, l_2 \in X_{\mathscr{W}}/_{\sim}$ exists amounts to checking the non-emptiness of the intersection $Post(\mathbf{X}_{l_1}) \cap \mathbf{X}_{l_2}$. Formally, the computation of transitions in the quotient $T_{\mathscr{W}}/_{\sim}$ for any states $l_1, l_2 \in X_{\mathscr{W}}/_{\sim}$, where $l_1 \neq$ Out and $l_2 \neq$ Out is summarized as

$$l_2 \in \delta_{\mathscr{W},\sim}(l_1) \text{ if and only if } Post(\mathbf{X}_{l_1}) \cap \mathbf{X}_{l_2} \neq \emptyset. \tag{7.2}$$

Given a polytope $\mathbf{X}_l$ for some $l \in L$, the set of successor states $Post(\mathbf{X}_l)$ is another polytope computable as[1]:

$$Post(\mathbf{X}_l) = A_l \mathbf{X}_l \oplus \mathbf{P}_l^c, \tag{7.3}$$

where $A_l \mathbf{X}_l$ is the image of polytope $\mathbf{X}_l$ through matrix A_l (see Appendix A.3) and "$\oplus$" denotes the Minkowski (set) sum (Definition A.7). Since $Post(\mathbf{X}_l)$ is a polytope, for any states $l_1, l_2 \in X_{\mathcal{W}}/_\sim$ the intersection $Post(\mathbf{X}_{l_1}) \cap \mathbf{X}_{l_2}$ is also a polytope and its non-emptiness can be checked easily using polyhedral operations. Note that, for the particular case of a fixed parameter PWA system, $\mathbf{P}_l^c$ in Eq. (7.3) is a singleton $c_l \in \mathbb{R}^N$.

Given a state $l \in X_{\mathcal{W}}/_\sim$, where $l \neq \text{Out}$, a transition from state l to state Out is assigned in accordance to Definition 6.6 as

$$\text{Out} \in \delta_{\mathcal{W},\sim}(l) \text{ if and only if } Post(\mathbf{X}_l) \not\subseteq \mathbf{X}, \tag{7.4}$$

which is also checked easily, since both $Post(\mathbf{X}_l)$ and $\mathbf{X}$ are polytopic sets. To complete the construction of $\delta_{\mathcal{W},\sim}$, transitions for state Out $\in X_{\mathcal{W}}/_\sim$ must be assigned but, from Definitions 6.1 and 6.6, it only has a transition to itself (i.e., $\delta_{\mathcal{W},\sim}(\text{Out}) = \{\text{Out}\}$).

Algorithm 14 $T_{\mathcal{W}}/_\sim$ =QUOTIENT($\mathcal{W}$) : Compute the quotient $T_{\mathcal{W}}/_\sim$ of an additive uncertainty PWA system $\mathcal{W}$

```
$X_{\mathcal{W}}/_\sim := L \bigcup \{\text{Out}\}$
$O_{\mathcal{W}} := X_{\mathcal{W}}/_\sim$
for all $l \in X_{\mathcal{W}}/_\sim$ do
  $o_{\mathcal{W},\sim}(l) := l$
  $\delta_{\mathcal{W},\sim} := \emptyset$
  if $Post(\mathbf{X}_l) \not\subseteq \mathbf{X}$ then
    $\delta_{\mathcal{W},\sim}(l) := \delta_{\mathcal{W},\sim}(l) \cup \{\text{Out}\}$
  end if
  for all $l' \in X_{\mathcal{W}}/_\sim$ do
    if $Post(\mathbf{X}_l) \cap \mathbf{X}_{l'} \neq \emptyset$ then
      $\delta_{\mathcal{W},\sim}(l) := \delta_{\mathcal{W},\sim}(l) \cup \{l'\}$
    end if
  end for
end for
$\delta_{\mathcal{W},\sim}(\text{Out}) := \{\text{Out}\}$
return $T_{\mathcal{W}}/_\sim = (X_{\mathcal{W}}/_\sim, \delta_{\mathcal{W},\sim}, O_{\mathcal{W}}, o_{\mathcal{W},\sim})$
```

The transition map of quotient $T_{\mathcal{W}}/_\sim$ is constructed using the computation described above, which completes the quotient's construction. The computation of

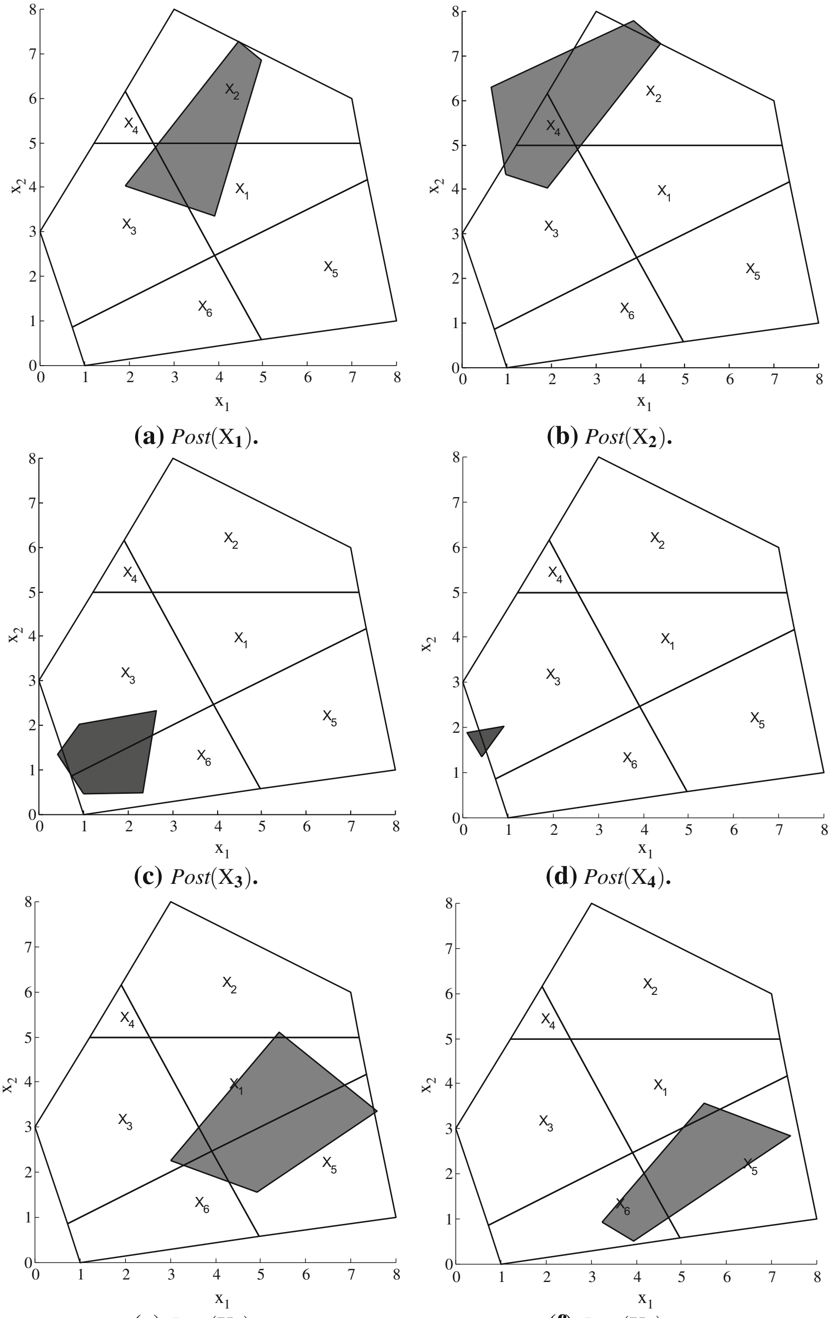
[1] In this chapter, we assume, for simplicity of presentation, that all matrices $A_l, l \in L$ are invertible. This assumption can be easily relaxed as discussed in Sect. 7.4. The technical details are included in Sects. A.3 and A.4.

$T_{\mathscr{W}}/_{\sim}$ is summarized in Algorithm 14, which is implementable using polyhedral operations on polytopes. Since the number of regions L of PWA system $\mathscr{W}$ is finite, $T_{\mathscr{W}}$ has a finite set of observations $O_{\mathscr{W}}$ and, as a result, the set of states $X_{\mathscr{W}}/_{\sim}$ of the quotient is also finite. This allows the application of model checking or analysis of $T_{\mathscr{W}}/_{\sim}$ through Algorithm 3 but the implementation of the more advanced analysis procedure from Chap. 4 requires additional operations, which will be discussed next.

Example 7.1 We apply Algorithm 14 to construct the quotient $T_{\mathscr{W}}/_{\sim}$ for the PWA system $\mathscr{W}$ defined in Example 6.2. Initially, the system had four regions (Fig. 6.2a) but additional partitioning of the state space was required to accommodate some specifications, resulting in a system with six regions denoted by $\mathbf{X}_1, \ldots, \mathbf{X}_6$ (Fig. 6.2b) with $L = \{1, \ldots, 6\}$. Therefore, the quotient $T_{\mathscr{W}}/_{\sim}$ has six states $X_{\mathscr{W}}/_{\sim} = \{1, \ldots, 6\}$ where, for each state $l \in X_{\mathscr{W}}/_{\sim}$, the set of equivalent states of $T_{\mathscr{W}}$ (and therefore $\mathscr{W}$) is given by $con(l) = \mathbf{X}_l$ as in Eq. (7.1).

To compute the transitions of $T_{\mathscr{W}}/_{\sim}$, we compute the set of successors $Post(\mathbf{X}_l)$ for each region $\mathbf{X}_l$ of $T_{\mathscr{W}}$ (see Fig. 7.1). Checking the non-emptiness of the intersection $Post(\mathbf{X}_{l_1}) \cap \mathbf{X}_{l_2}$ allows us to compute the transitions of $T_{\mathscr{W}}/_{\sim}$. Only the set of successors of region 6 is completely included within the defined state space $\mathbf{X}$ and, therefore, all other states have a transition to state Out (note that $Post(\mathbf{X}_1) \not\subset \mathbf{X}$, although this is not obvious from Fig. 7.1a). This leads to the inclusion of transitions $\delta_{\mathscr{W},\sim}(1) = \{1, 2, 3, \text{Out}\}$, $\delta_{\mathscr{W},\sim}(2) = \{2, 3, 4, \text{Out}\}$, $\delta_{\mathscr{W},\sim}(3) = \{3, 6, \text{Out}\}$, $\delta_{\mathscr{W},\sim}(4) = \{3, \text{Out}\}$, $\delta_{\mathscr{W},\sim}(5) = \{1, 2, 3, 5, 6, \text{Out}\}$ and $\delta_{\mathscr{W},\sim}(6) = \{1, 5, 6\}$. The resulting quotient $T_{\mathscr{W}}/_{\sim}$ is shown in Fig. 7.2, where the observations for each state are omitted but are clear from the state labels.

By embedding the PWA system $\mathscr{W}$ into an infinite transition system $T_{\mathscr{W}}$ (Definition 6.6), we reduced Problem 7.1 to Problem 4.1. However, since $T_{\mathscr{W}}$ was infinite, the analysis procedure outlined as Algorithm 3 in Chap. 4 could not be applied directly. So far, we showed that the quotient $T_{\mathscr{W}}/_{\sim}$ of the embedding $T_{\mathscr{W}}$ under the observational equivalence relation $\sim$ (Definition 1.2) can be constructed using polyhedral operations (Algorithm 14). Since $T_{\mathscr{W}}/_{\sim}$ is finite, this allows us to apply the analysis technique described in Sect. 1.3. However, as discussed there, such an approach leads to a conservative solution to Problem 7.1. In order to obtain less conservative results, bisimulation-based and formula-guided quotient refinement techniques were proposed in Sects. 4.3 and 4.5, respectively. Both methods were initialized by constructing a finite quotient such as $T_{\mathscr{W}}/_{\sim}$ but in addition required the implementation of a state refinement procedure.



(a) *Post*(X$_\mathbf{1}$).

(b) *Post*(X$_\mathbf{2}$).

(c) *Post*(X$_\mathbf{3}$).

(d) *Post*(X$_\mathbf{4}$).

(e) *Post*(X$_\mathbf{5}$).

(f) *Post*(X$_\mathbf{6}$).

Fig. 7.1 Successor states (*shaded gray*) of different regions of PWA system $\mathscr{W}$ defined in Example 6.2 (Fig. 6.2b). See Example 6.2 for additional details

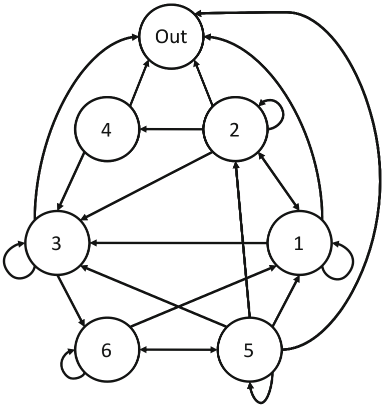


Fig. 7.2 Finite quotient $T_{\mathscr{W}}/_\sim$ of PWA system $\mathscr{W}$ defined in Example (6.2) (Fig. 6.2b). Observations of the states are omitted. See Example 7.1 for additional details

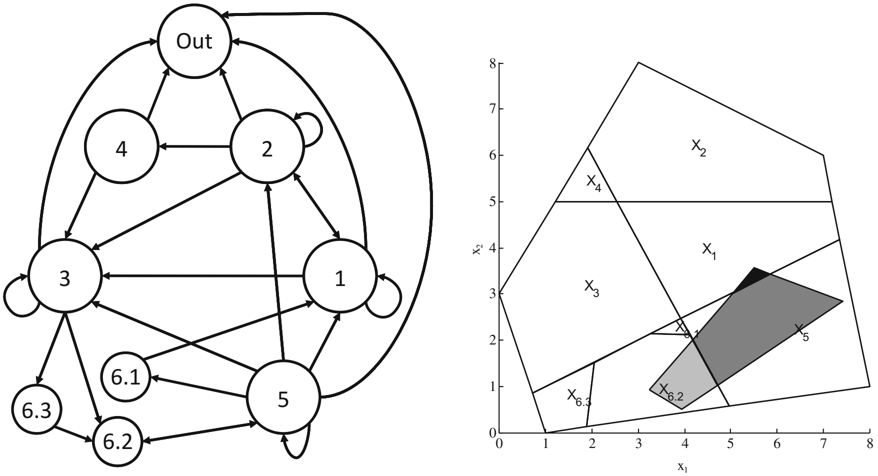
In the following, we focus on the implementation of the refinement procedure REFINE() (Algorithm 5, Chap. 4) and show that for autonomous additive uncertainty (and fixed parameter) PWA systems all its operations are computable through polyhedral operations. Specifically, as the embedding of a PWA system with additive parameter uncertainty is non-deterministic, we will refer to Algorithm 5. For the particular case of a PWA with fixed parameters, whose embedding is deterministic, the refinement procedure is described in Algorithm 6.

To implement function REFINE() for $T_{\mathscr{W}}$, given states $l_1, l_2 \in X_{\mathscr{W}}/_\sim$ such that $l_2 \in \delta_{\mathscr{W},\sim}(l_1)$ (i.e., l_2 is reachable from l_1 in $T_{\mathscr{W}}/_\sim$), we need to be able to construct a state l', such that $con(l') = con(l_1) \cap Pre(con(l_2))$ (see Algorithms 5 and 6). From Eq. (7.1), this computation reduces to the construction of a state l' where $con(l') = \mathbf{X}_{l_1} \cap Pre(\mathbf{X}_{l_2})$.

Under the invertibility assumption made earlier in this chapter, which, as stated, can be easily relaxed (see Sect. 7.4), this intersection is computable as

$$\mathbf{X}_{l_1} \cap Pre(\mathbf{X}_{l_2}) = \mathbf{X}_{l_1} \cap A_{l_1}^{-1}(\mathbf{X}_{l_2} \ominus \mathbf{P}_{l_1}^c), \tag{7.5}$$

where $\ominus$ denotes the Minkowski difference (Definition A.8). Note that while the $Pre()$ operation is applied to region $\mathbf{X}_{l_2}$, the parameters of region $\mathbf{X}_{l_1}$ are used for the computation, which is consistent with Definition 6.2. Using Eq. (7.5) to refine the states of $T_{\mathscr{W}}/_\sim$ and Eq. (7.3) to update its transitions wherever necessary (see Algorithms 5 and 6) allows the implementation of function REFINE() and all computation is performed using polyhedral operations.



(a) Refined quotient $T_{\mathscr{W}}/_{\sim_r}$ (b) Equivalence classes $X_{\mathscr{W}}/_{\sim_r}$

Fig. 7.3 Refined quotient $T_{\mathscr{W}}/_{\sim_r}$ =REFINE($T_{\mathscr{W}}/_{\sim}$, 6) (**a**) and equivalence classes $X_{\mathscr{W}}/_{\sim_r}$ (**b**) of PWA system $\mathscr{W}$ from Example 6.2 (Fig. 6.2b). The successor states $Post(con(6.1))$ (dark gray), $Post(con(6.2))$ (medium gray) and $Post(con(6.3))$ (light gray) are also shown for the refined subsets $6.1, 6.2, 6.3 \in X_{\mathscr{W}}/_{\sim_r}$, where $con(6.1) \cup con(6.2) \cup con(6.3) = con(6)$ for state $6 \in X_{\mathscr{W}}/_{\sim}$. See Example 7.2 for additional details

Example 7.2 We apply function REFINE() (Algorithm 6) to refine the quotient $T_{\mathscr{W}}/_{\sim}$ (constructed in Example 7.1 and shown in Fig. 7.2) of PWA system $\mathscr{W}$ defined in Example 6.2 (Fig. 6.2b). We target refinement to state $6 \in X_{\mathscr{W}}/_{\sim}$ and construct the refined quotient $T_{\mathscr{W}}/_{\sim_r}$ =REFINE($T_{\mathscr{W}}/_{\sim}$, 6). State 6 has three successors in $X_{\mathscr{W}}/_{\sim}$ (i.e., $\delta_{\mathscr{W},\sim} = \{1, 5, 6\}$) and, therefore, refinement results in three subsets in $X_{\mathscr{W}}/_{\sim_r}$ denoted as 6.1, 6.2 and 6.3, where $con(6.1) \cup con(6.2) \cup con(6.3) = con(6)$. Each subset has only a single outgoing transitions in $T_{\mathscr{W}}/_{\sim_r}$ (see the sets of successors shown in Fig. 7.3b), which is implicitly induced through the refinement and incoming transitions are recomputed (see Algorithm 6). This results in the construction of the refined quotient $T_{\mathscr{W}}/_{\sim_r}$ shown in Fig. 7.3a.

Note that the notation is abused in this example and in the rest of this chapter. As we assumed that all the polytopes are open, the equality $con(6.1) \cup con(6.2) \cup con(6.3) = con(6)$ does not hold precisely. Indeed, $con(6)$ contains some facets of $con(6.1)$, $con(6.2)$, and $con(6.3)$, which are not contained in $con(6.1) \cup con(6.2) \cup con(6.3)$. More discussions are included in Sect. 7.4.

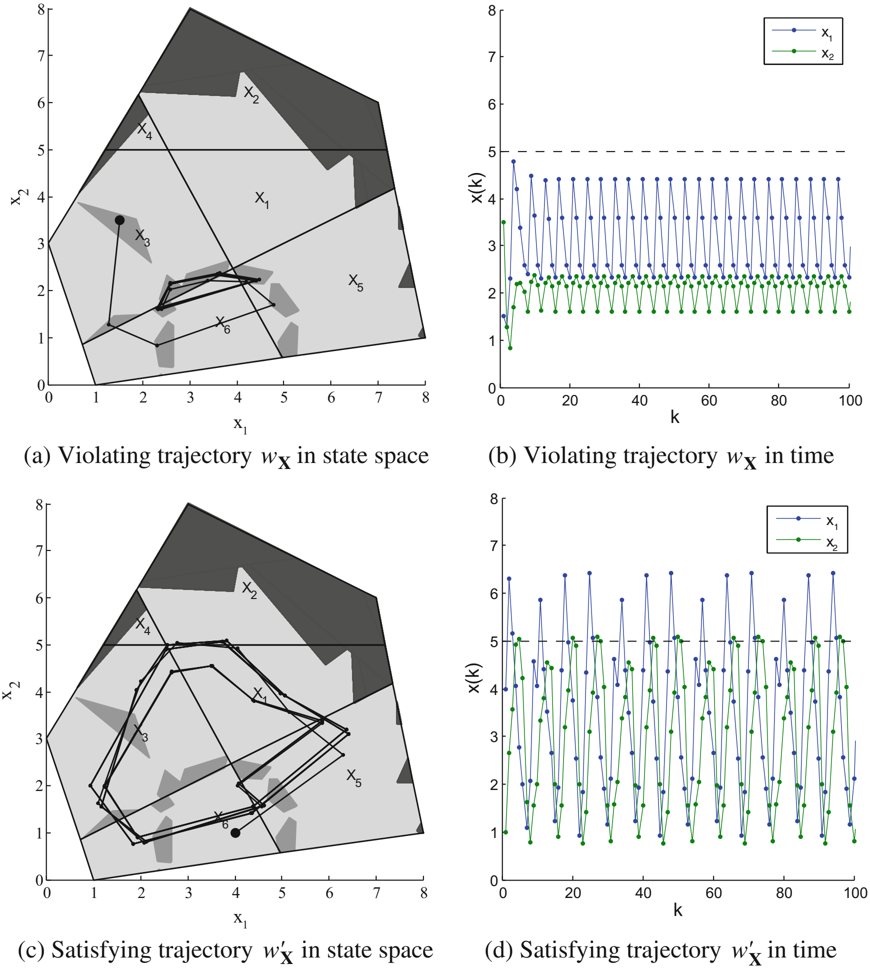
After a state $l \in X_{\mathscr{W}}/_{\sim}$ is refined into states l_1 and l_2 such that $con(l_1) \cup con(l_2) = con(l)$, the computation from Eqs. (7.3) and (7.5) can be applied to the subsets l_1 and l_2. This enables the iterative refinement of the quotient $T_{\mathscr{W}}/_{\sim}$ and allows the implementation of the bisimulation algorithm (Algorithm 1) and the analysis methods described in Sects. 4.3 (Algorithm 7) and 4.5 (Algorithm 8) for PWA systems.

A termination condition based on the sizes of equivalence classes was also proposed in Chap. 4 for the analysis procedures discussed there. To determine if a state $l \in X_{\mathscr{W}}/_{\sim}$ is "large enough" to undergo additional refinement, we compute the radius of the largest sphere inscribed in polytope $con(l)$ and apply the refinement procedure only if it is larger than a certain predefined limit ε. In other words, we apply the refinement procedure to state l only if $r(\mathbf{X}_l) > \varepsilon$, where $r(\mathbf{X}_l)$ is the radius of the Chebyshev ball of $\mathbf{X}_l$ (see Definition A.9 in the Appendix).

Example 7.3 We apply the analysis method from Sect. 4.3 summarized as Algorithm 7 to identify satisfying and violating regions of PWA system $\mathscr{W}$ defined in Example 6.2 and shown in Fig. 6.2a. We are interested in testing whether trajectories of the system keep reaching values over 5 in the second component x_2 of the system state x. Therefore, we introduce additional partitions to the states space of the system as shown in Fig. 6.2b and formulate the specification as the LTL formula $\phi = \Box\Diamond(2 \vee 4)$, requiring that states from regions $\mathbf{X}_2$ and $\mathbf{X}_4$ are visited infinitely often. Furthermore, we want to guarantee that trajectories of the system remain within the defined state space $\mathbf{X}$ and therefore augment the specification as $\phi' = \Box\Diamond(2 \vee 4) \wedge \Box\neg\mathrm{Out}$.

We set $\varepsilon = 0.1$ as the limit on the states from $X_{\mathscr{W}}/_{\sim_r}$ that can undergo refinement, which guarantees the termination of the analysis procedure. While only an under-approximation of the largest satisfying and strictly violating regions of $T_{\mathscr{W}}$ (and therefore $\mathscr{W}$) is obtained as discussed in Chap. 4, most of the system's state space is characterized as satisfying or violating (see Fig. 7.4a).

All trajectories originating in the regions shown in dark and medium gray in Fig. 7.4a violate specification ϕ'—trajectories originating in the dark gray region leave the defined state space $\mathbf{X}$, while trajectories originating in the light gray region oscillate but do not visit states where the values of the second component x_2 are above 5. However, all trajectories originating in the satisfying region shown in light gray in Fig. 7.4a are guaranteed to satisfy the specification.



(a) Violating trajectory $w_{\mathbf{X}}$ in state space

(b) Violating trajectory $w_{\mathbf{X}}$ in time

(c) Satisfying trajectory $w'_{\mathbf{X}}$ in state space

(d) Satisfying trajectory $w'_{\mathbf{X}}$ in time

Fig. 7.4 Satisfying (*light gray*) and violating (*medium gray*) regions of PWA system $\mathscr{W}$ defined in Example 6.2 (Fig. 6.2b) for specification "$\Box\Diamond(2 \vee 4) \wedge \Box\neg\text{Out}$" were identified using the analysis procedure described in Sect. 4.3 (Algorithm 7). Trajectories of $\mathscr{W}$ originating in the region shown in dark gray leave the defined state space of the system and therefore are also violating. A violating trajectory $w_{\mathbf{X}}$ (**a**) and (**b**) and a satisfying trajectory $w'_{\mathbf{X}}$ (**c**) and (**d**) were obtained by initializing $\mathscr{W}$ in the violating or satisfying regions (initial conditions are shown as large *circles*). See Example 7.3 for additional details

Note that for an autonomous, fixed-parameter PWA system $\mathscr{W}$ (Definition 6.4), the embedding $T_{\mathscr{W}}$ is deterministic, which allows the application of the more efficient refinement strategies from Algorithm 6. The computation from Eq. (7.5) is also sufficient to implement refinement strategies for autonomous, additive uncertainty PWA systems (Definition 6.5) through Algorithm 5. Refinement strategies for more general autonomous, uncertain parameters systems are discussed in the following Sect. 7.2.

7.2 PWA Systems with Uncertain Parameters

When the matrix component of the parameters is allowed to vary as in the autonomous PWA system from Definition 6.3, given a polytope $\mathbf{X}_l$, the set $Post_{T_{\mathscr{W}}}(\mathbf{X}_l)$ is not necessarily convex and, in general, there are no algorithms capable of its exact computation. Thus, the computational procedures for the construction and refinement of quotients described so far in this chapter do not apply directly to such systems. Instead, in the following we develop an analysis strategy for autonomous, uncertain parameter PWA systems based on the construction of over-approximation quotients.

Proposition 7.1 *Given a polytope $\boldsymbol{X}_l$, a convex over-approximation of $Post_{T_{\mathscr{W}}}(\boldsymbol{X}_l)$ can be computed as:*

$$\overline{Post}_{T_{\mathscr{W}}}(\boldsymbol{X}_l) = hull\{Ax \mid A \in V(\boldsymbol{P}_l^A), x \in V(\boldsymbol{X}_l)\} \oplus \boldsymbol{P}^c, \tag{7.6}$$

where $hull()$ and $V()$ denote the convex hull and set of vertices, respectively (see Sect. A.1).

Proof Let $V(\mathbf{X}_l) = \{v_1, \ldots, v_R\}$ and $V(\mathbf{P}^A) = \{w_1, \ldots, w_M\}$. Let $x \in \mathbf{X}_l$ and $A \in \mathbf{P}_l^A$. Then $x = \sum_{r=1}^{R} \lambda_r v_r$, $A = \sum_{m=1}^{M} \mu_m w_m$, and

$$Ax = (\sum_{m=1}^{M} \mu_m w_m)(\sum_{r=1}^{R} \lambda_r v_r) = \sum_{m=1}^{M} \sum_{r=1}^{R} \mu_m \lambda_r w_m v_r$$

Since $\mu_m, \lambda_r \geq 0$ and $\sum_{m=1}^{M} \mu_m = \sum_{r=1}^{R} \lambda_r = 1$ then $\mu_m \lambda_r \geq 0$ for any m, r and $\sum_{m=1}^{M} \sum_{r=1}^{R} \mu_m \lambda_r = 1$. Therefore

$$Ax \in \text{hull}\{wv, w \in V(\mathbf{P}_l^A), v \in V(\mathbf{X})\}$$

and the rest of the proof follows from Definition A.7. ■

The computation from Eq. (7.6) produces an over-approximation of the reachable states from a given region and is the smallest convex set containing $Post_{T_{\mathscr{W}}}(\mathbf{X}_l)$:

$$Post_{T_{\mathscr{W}}}(\mathbf{X}_l) \subseteq \overline{Post}_{T_{\mathscr{W}}}(\mathbf{X}_l) \tag{7.7}$$

Although a precise distance between the real set and its over-approximation is hard to quantify, the volume of $Post_{T_{\mathscr{W}}}()$ was not significantly increased by the approximation for the systems we considered.

Using the over-approximation $\overline{Post}_{T_{\mathscr{W}}}(\mathbf{X}_l)$, instead of the exact $Post_{T_{\mathscr{W}}}(\mathbf{X}_l)$ an over-approximation quotient $\overline{T_{\mathscr{W}}/_{\sim}} = (Q_{\mathscr{W}}/_{\sim}, \overline{\delta_{\mathscr{W},\sim}}, O_{\mathscr{W}}, o_{\mathscr{W},\sim})$ can be constructed. From Eq. (7.7), it follows that for all $l \in Q_{\mathscr{W}}/_{\sim}$, we have $\delta_{\mathscr{W},\sim} \subseteq \overline{\delta_{\mathscr{W},\sim}}$, which leads to

$$\mathscr{L}_{T_{\mathscr{W}}} \subseteq \mathscr{L}_{T_{\mathscr{W}}/_{\sim}} \subseteq \mathscr{L}_{\overline{T_{\mathscr{W}}/_{\sim}}}. \tag{7.8}$$

Therefore, the over-approximation quotient $\overline{T_{\mathscr{W}}/_{\sim}}$ simulates the exact quotient $T_{\mathscr{W}}/_{\sim}$ and $\overline{T_{\mathscr{W}}/_{\sim}}$ can be used instead of $T_{\mathscr{W}}/_{\sim}$ for the methods we developed in Chap. 4 but the results become more conservative. The over-approximation quotient $\overline{T_{\mathscr{W}}/_{\sim}}$ can be computed through Algorithm 14 by substituting the $Post_{T_{\mathscr{W}}}()$ operation with its over-approximation $\overline{Post}_{T_{\mathscr{W}}}()$ when the matrix component of the parameters of $\mathscr{W}$ is uncertain. This leads to the computation of $X^{\phi}_{\overline{T_{\mathscr{W}}/_{\sim}}}$ where $X^{\phi}_{\overline{T_{\mathscr{W}}/_{\sim}}} \subseteq X^{\phi}_{T_{\mathscr{W}}/_{\sim}} \subseteq X^{\phi}_{T_{\mathscr{W}}}$, following from Eq. (7.8). While this is sufficient to apply some of the analysis strategies developed previously, additional refinement strategies are required to obtain less conservative analysis results but for autonomous, uncertain parameter PWA systems, the $Pre()$ operation is not easily computable. Instead, we apply a refinement approach, where a polytope $\mathbf{X}_l$ is split along each dimension e.g., through the center of the Chebyshev ball of $\mathbf{X}_l$ (Definition A.9). While this strategy is less efficient since the dynamics of the system are not taken into account during refinement, it allows an implementation through quad-tree data structures and their extensions into higher dimensions.

By constructing the over-approximation quotient in Algorithm 7 and using it within the analysis methods from Chap. 4 (e.g., Algorithm 8) together with the refinements strategy described above, a (more conservative) solution to Problem 7.1 is obtained even for autonomous, uncertain parameter PWA systems.

Example 7.4 Consider a two dimensional ($N = 2$) autonomous PWA system that has a total of nine rectangular regions $\mathbf{X}_1, \ldots, \mathbf{X}_9$ labeled by $L = \{1, 2, \ldots, 9\}$. The parameters for each region for an initial fixed parameter model (where $\mathbf{P}_l^A$, $\mathbf{P}_l^c$ are singletons A_l, c_l, respectively) are as follows:

$$A_1 = A_3 = A_9 = \begin{bmatrix} 0.82 & 0.00 \\ 0.00 & 0.67 \end{bmatrix}, \quad A_2 = A_8 = \begin{bmatrix} 0.82 & -0.37 \\ 0.00 & 0.67 \end{bmatrix},$$

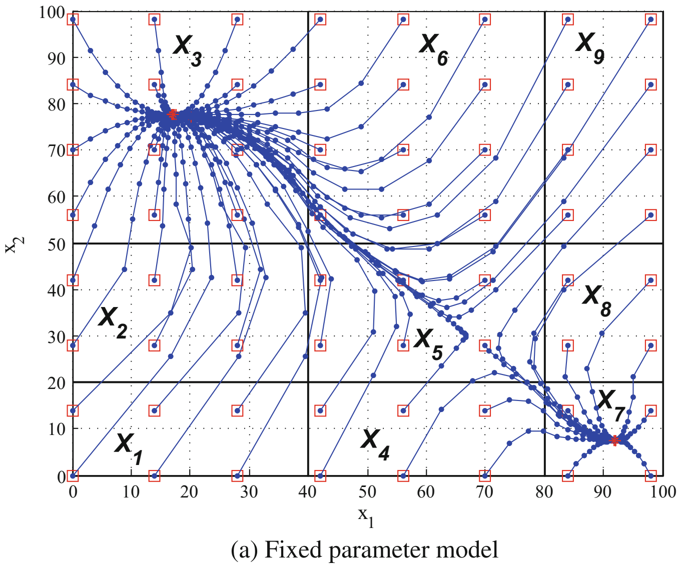
$$A_4 = A_6 = \begin{bmatrix} 0.82 & 0.00 \\ -0.52 & 0.67 \end{bmatrix}, \quad A_5 = \begin{bmatrix} 0.96 & -0.39 \\ -0.55 & 0.80 \end{bmatrix}, \quad A_7 = \begin{bmatrix} 0.82 & 0.00 \\ 0.00 & 0.67 \end{bmatrix},$$

$$c_1 = \begin{bmatrix} 16.68 \\ 25.55 \end{bmatrix}, \quad c_2 = \begin{bmatrix} 19.37 \\ 25.55 \end{bmatrix}, \quad c_3 = \begin{bmatrix} 3.08 \\ 25.55 \end{bmatrix},$$

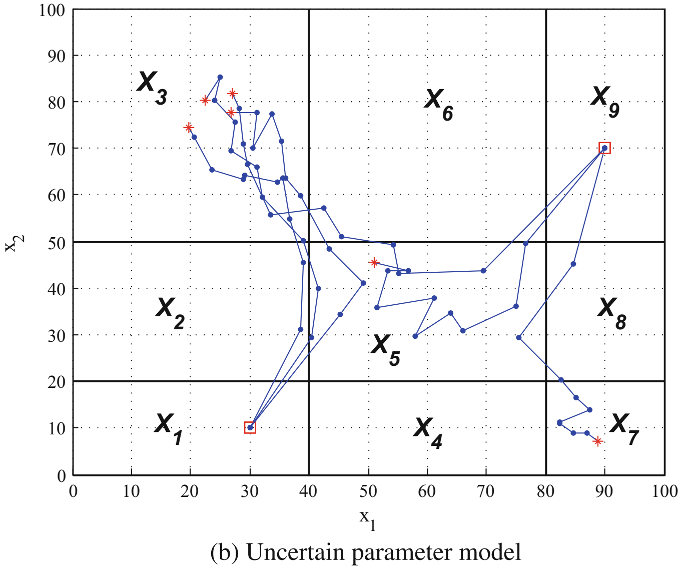
$$c_4 = \begin{bmatrix} 16.68 \\ 43.34 \end{bmatrix}, \quad c_5 = \begin{bmatrix} 14.66 \\ 42.97 \end{bmatrix}, \quad c_6 = \begin{bmatrix} 3.08 \\ 47.65 \end{bmatrix},$$

$$c_7 = \begin{bmatrix} 16.68 \\ 2.47 \end{bmatrix}, \quad c_8 = \begin{bmatrix} 25.12 \\ 2.47 \end{bmatrix}, \quad c_9 = \begin{bmatrix} 3.08 \\ 2.47 \end{bmatrix}$$

Under the fixed parameters, dynamics 3 and 7 have unique, asymptotically stable equilibria inside rectangles $\mathbf{X}_3$ and $\mathbf{X}_7$ (see Fig. 7.5). An interesting problem is finding the regions of attraction for the two equilibria and exploring how those regions change when parameter uncertainty is introduced. By exploiting convexity properties of affine functions on polytopes, it can be easily proved that under the fixed parameters, $\mathbf{X}_3$ and $\mathbf{X}_7$ are invariants for dynamics 3 and 7, respectively. From this, we can immediately conclude that $\mathbf{X}_3$ and $\mathbf{X}_7$ are regions of attraction for the two equilibria. Therefore, our problem reduces to



(a) Fixed parameter model



(b) Uncertain parameter model

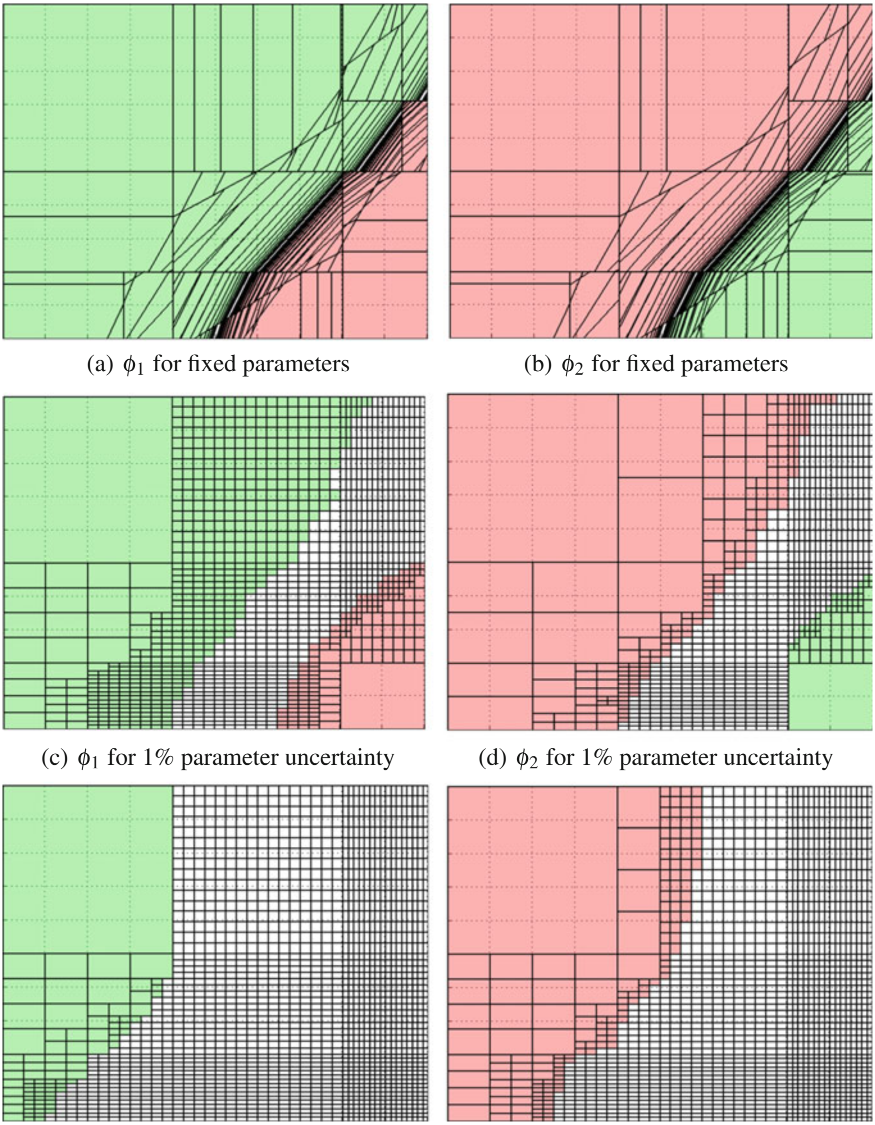
Fig. 7.5 Simulated trajectories of the autonomous PWA system from Example 7.4. Initial conditions are denoted by *red squares*

finding maximal regions satisfying LTL formulas $\phi_1 = \Diamond\Box 3$ and $\phi_2 = \Diamond\Box 7$. In other words, we want to find maximal sets of initial conditions, from which trajectories will eventually reach regions $\mathbf{X}_3$ or $\mathbf{X}_7$ and stay there forever.

To explore how the sizes of the attractor regions change, hyper-rectangular parameter uncertainty was introduced in the model by allowing each component of the parameters A_l and c_l for region $l \in L$ to vary in a range of size specified as a percentage of the fixed parameter value and centered around it (parameter components equal to 0 were also allowed to vary in a small range). Results from the computation with various levels of uncertainty are compared with the ones obtained under fixed parameters (Fig. 7.6). Because of the rectangular initial partition of the state space, 2^N-trees could be used as an efficient splitting strategy for the uncertain parameter case. Our method identifies only an attracting region for the equilibria at $\mathbf{X}_3$ for 5% uncertainty. As expected, increasing the level of uncertainty in the parameters decreases the size of the identified regions of state space (but a region identified at higher uncertainty is always a subset of the one identified at lower uncertainty).

Even if smaller limit ε is used, under parameter uncertainty it is possible that a subset of the state space is never included in the identified regions—a property resulting from nondeterminism introduced in the embedding transitions system. Even though complete partitioning of the state space might not be possible, decreasing ε provides further refinement and greater detail of the identified regions (initial iterations attempt to capture large satisfying (or violating) regions, while subsequent ones expand the solution less but provide greater resolution on its boundaries).

Example 7.5 Consider the PWA system from Example 7.4, where additive uncertainty is introduced by allowing each parameter c_l for region $l \in L$ to vary in a range of 10% of its original value (as before, parameter components equal to 0 were also allowed to vary in a small range). This corresponds to the PWA from Definition 6.5. As in Example 7.4, we are interested in identifying maximal regions satisfying LTL formulas $\phi_1 = \Diamond\Box 3$ and $\phi_2 = \Diamond\Box 7$. This problem can be approached by applying the refinement strategy for uncertain parameter systems as in Example 7.4, leading to the identification of the satisfying and violating regions presented in Fig. 7.7a, b. Alternatively, the specialized refinement strategy for additive uncertainty systems can be applied, resulting in the identification of the regions presented in Fig. 7.7c, d. Such a strategy leads to a finer partition, since at each step a region is refined by considering all combinations of regions reachable from it. However, this allows the identification of larger satisfying and violating regions for the same limit $\varepsilon = 5$, compared to the more general refinement strategy.



(a) ϕ_1 for fixed parameters

(b) ϕ_2 for fixed parameters

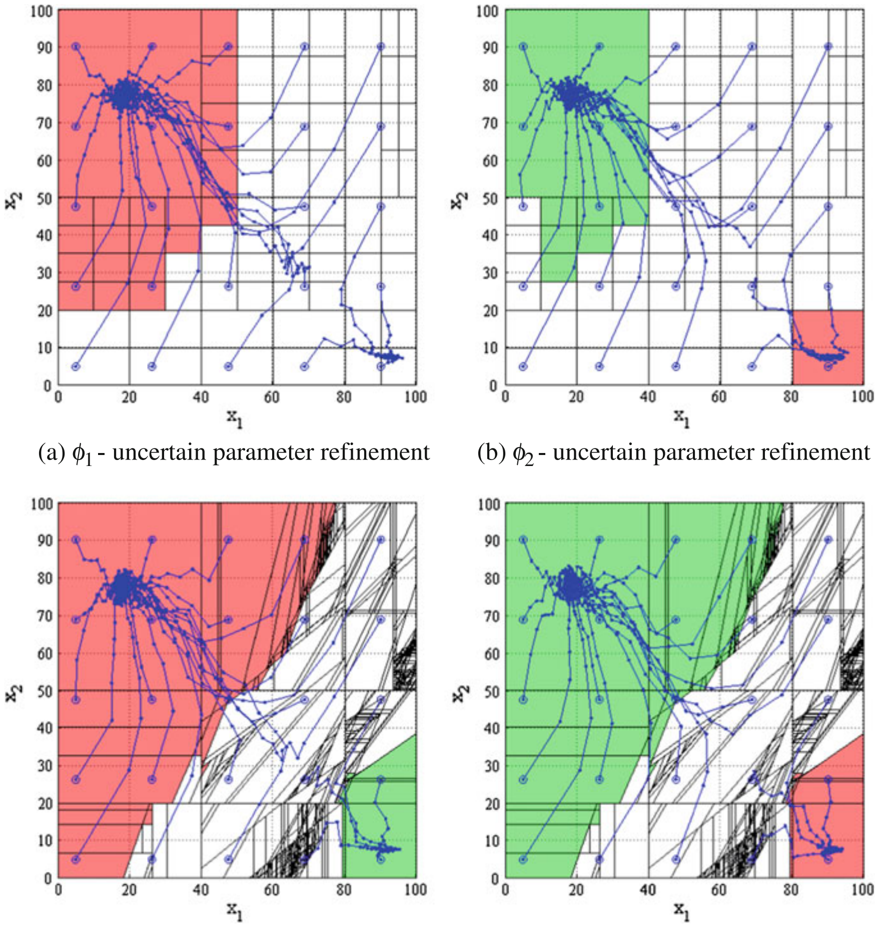
(c) ϕ_1 for 1% parameter uncertainty

(d) ϕ_2 for 1% parameter uncertainty

(e) ϕ_1 for 5% parameter uncertainty

(f) ϕ_2 for 5% parameter uncertainty

Fig. 7.6 Results for Example 7.4. Regions satisfying the formula are shown in *green*, while regions satisfying the negation of the formula are shown in *red*. The refinement exploits the system dynamics (using Pre) for the case when the parameters are fixed ((**a**) and (**b**)) and it uses a generic rectangular partitions when the parameters are uncertain ((**c**), (**d**), (**e**), and (**f**))



(a) ϕ_1 - uncertain parameter refinement

(b) ϕ_2 - uncertain parameter refinement

(c) ϕ_1 - additive uncertainty refinement

(d) ϕ_2 - additive uncertainty refinement

Fig. 7.7 The general refinement strategy for uncertain parameter systems and the specific refinement strategy for additive uncertainty systems were applied to the PWA system from Example 7.4 with 10% additive uncertainty. Regions satisfying the formula are shown in green, while regions satisfying the negation of the formula are shown in red. While the general strategy leads to fewer regions after refinement with the given limit $\varepsilon = 5$, the additive uncertainty refinement allows the identification of larger satisfying and violating regions

Example 7.6 Consider a three dimensional ($N = 3$) autonomous PWA system that has a total of 27 rectangular regions $\mathbf{X}_1, \ldots, \mathbf{X}_{27}$ labeled by $L = \{1, 2, \ldots, 27\}$. The parameters for each region for a fixed parameter model (where $\mathbf{P}_l^A, \mathbf{P}_l^c$ are singletons A_l, c_l, respectively) are as follows:

$$A_{1,3,7,9,19,21,25,27} = \begin{bmatrix} 0.67 & 0 & 0 \\ 0 & 0.67 & 0 \\ 0 & 0 & 0.67 \end{bmatrix}$$

$$A_{2,8,20,26} = \begin{bmatrix} 0.67 & 0 & 0 \\ 0 & 0.67 & 0 \\ 0 & -0.63 & 0.67 \end{bmatrix}$$

$$A_{4,6,22,24} = \begin{bmatrix} 0.67 & 0 & 0 \\ -0.63 & 0.67 & 0 \\ 0 & 0 & 0.67 \end{bmatrix}$$

$$A_{5,23} = \begin{bmatrix} 0.67 & 0 & 0 \\ -0.63 & 0.67 & 0 \\ 0.29 & -0.63 & 0.67 \end{bmatrix}$$

$$A_{10,12,16,18} = \begin{bmatrix} 0.67 & 0 & -0.63 \\ 0 & 0.67 & 0 \\ 0 & 0 & 0.67 \end{bmatrix}$$

$$A_{11,17} = \begin{bmatrix} 0.67 & 0.29 & -0.63 \\ 0 & 0.67 & 0 \\ 0 & -0.63 & 0.67 \end{bmatrix}$$

$$A_{13,15} = \begin{bmatrix} 0.67 & 0 & -0.63 \\ -0.63 & 0.67 & 0.29 \\ 0 & 0 & 0.67 \end{bmatrix}$$

$$A_{14} = \begin{bmatrix} 0.58 & 0.29 & -0.6 \\ -0.6 & 0.58 & 0.29 \\ 0.29 & -0.6 & 0.58 \end{bmatrix}$$

$$c_1 = \begin{bmatrix} 26 \\ 26 \\ 26 \end{bmatrix} \quad c_2 = \begin{bmatrix} 26 \\ 26 \\ 38 \end{bmatrix} \quad c_3 = \begin{bmatrix} 26 \\ 26 \\ 3.3 \end{bmatrix} \quad c_4 = \begin{bmatrix} 26 \\ 38 \\ 26 \end{bmatrix} \quad c_5 = \begin{bmatrix} 26 \\ 38 \\ 31 \end{bmatrix} \quad c_6 = \begin{bmatrix} 26 \\ 38 \\ 3.3 \end{bmatrix}$$

$$c_7 = \begin{bmatrix} 26 \\ 3.3 \\ 26 \end{bmatrix} \quad c_8 = \begin{bmatrix} 26 \\ 3.3 \\ 48 \end{bmatrix} \quad c_9 = \begin{bmatrix} 26 \\ 3.3 \\ 3.3 \end{bmatrix} \quad c_{10} = \begin{bmatrix} 38 \\ 26 \\ 26 \end{bmatrix} \quad c_{11} = \begin{bmatrix} 31 \\ 26 \\ 38 \end{bmatrix} \quad c_{12} = \begin{bmatrix} 48 \\ 26 \\ 3.3 \end{bmatrix}$$

$$c_{13} = \begin{bmatrix} 38 \\ 31 \\ 26 \end{bmatrix} \quad c_{14} = \begin{bmatrix} 33 \\ 33 \\ 33 \end{bmatrix} \quad c_{15} = \begin{bmatrix} 48 \\ 28 \\ 3.3 \end{bmatrix} \quad c_{16} = \begin{bmatrix} 38 \\ 3.3 \\ 26 \end{bmatrix} \quad c_{17} = \begin{bmatrix} 28 \\ 3.3 \\ 48 \end{bmatrix} \quad c_{18} = \begin{bmatrix} 48 \\ 3.3 \\ 3.3 \end{bmatrix}$$

$$c_{19} = \begin{bmatrix} 3.3 \\ 26 \\ 26 \end{bmatrix} \quad c_{20} = \begin{bmatrix} 3.3 \\ 26 \\ 38 \end{bmatrix} \quad c_{21} = \begin{bmatrix} 3.3 \\ 26 \\ 3.3 \end{bmatrix} \quad c_{22} = \begin{bmatrix} 3.3 \\ 48 \\ 26 \end{bmatrix} \quad c_{23} = \begin{bmatrix} 3.3 \\ 48 \\ 28 \end{bmatrix} \quad c_{24} = \begin{bmatrix} 3.3 \\ 48 \\ 3.3 \end{bmatrix}$$

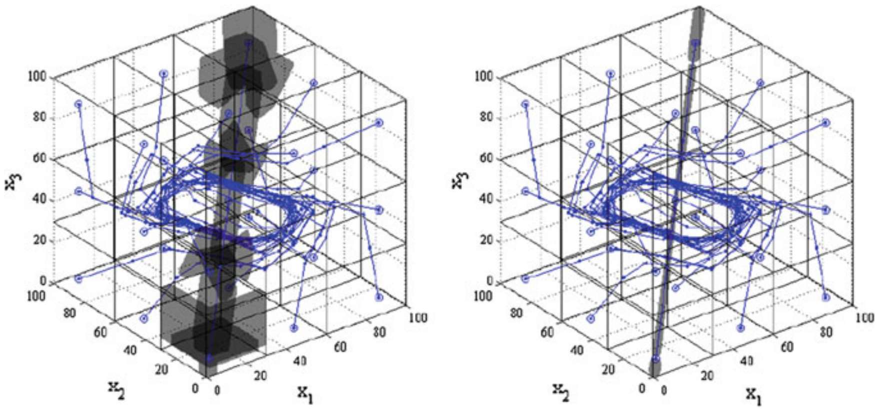
$$c_{25} = \begin{bmatrix} 3.3 \\ 3.3 \\ 26 \end{bmatrix} \quad c_{26} = \begin{bmatrix} 3.3 \\ 3.3 \\ 48 \end{bmatrix} \quad c_{27} = \begin{bmatrix} 3.3 \\ 3.3 \\ 3.3 \end{bmatrix}$$

The dynamics of the system are illustrated by simulated trajectories in Fig. 7.8. We are interested in testing whether all initial conditions lead to oscillatory behavior and define sub-formula ϕ, which is satisfied when the value of state variable 1 is above a certain threshold (i.e., ϕ is the disjunction of all regions $\mathbf{X}_l$ such that $\forall x \in \mathbf{X}_l$, $[1\ 0\ 0]x > 60$). By analyzing the system with LTL formula $\Box(\Diamond\phi \wedge \Diamond\neg\phi)$ we search for the maximal set of initial conditions guaranteeing that trajectories of the system keep oscillating between low and high values of the first component of the state. No violating region was identifying using the analysis approach, while most of the region $[0, 100]^3$ was identified as satisfying, with the exception of a small uncertain region shown in Fig. 7.8.

7.3 Formula-Guided Refinement

The analysis approach for PWA systems described in Sect. 7.1 is based on the construction and refinement of finite quotients inspired by bisimulation algorithms. As described in Sect. 4.5, an alternative approach to the analysis of large or infinite transition systems is to perform quotient refinement based on the exact specification for a particular problem. The goal in this case is to construct a formula-equivalent quotient (instead of a bisimilar one), which could be used for analysis instead of the original system, where all results are equivalent for the specific property of interest.

The approach from Sect. 4.5 can be implemented using polyhedral operations, utilizing the computations already described in Sect. 7.1, together with computational techniques described in Chap. 5. Using such a *formula-guided refinement* strategy,



(a) Refinement limit $\varepsilon = 5$

(b) Refinement limit $\varepsilon = 1$

Fig. 7.8 Results of the analysis in Example 7.6. Initial states of simulated trajectories are shown as circles. The specification is the system oscillates such that the value of the first component of the state goes above and under 60. Results for two refinement limits are shown. No violating region was identified for this property, while the largest satisfying region that was identified includes most of the state space $[0, 100]^3$, with the exception of the uncertain region shown in gray

the computation is performed on the product automaton and could lead to improved efficiency, since all refinement is guided by the specification, while the resulting refined system is simplified at each step. To illustrate this, the two approaches are compared in Example 7.7.

Example 7.7 To illustrate the differences between the analysis strategies described in Sects. 7.1 and 7.3, we apply these approaches to the two-dimensional ($N = 2$) PWA system introduced in Example 7.4 (denoted as $\mathscr{W}_2$ in the following) and the three-dimensional ($N = 3$) PWA system introduced in Example 7.6 (denoted as $\mathscr{W}_3$).

For $\mathscr{W}_2$, we seek the maximal sets of initial conditions guaranteeing that all trajectories of the system eventually reach regions $\mathbf{X}_3$ or $\mathbf{X}_7$, respectively, specified using LTL formulas $\Diamond 3$ and $\Diamond 7$.

For $\mathscr{W}_3$, we are interested in testing whether all initial conditions lead to oscillatory behavior and define sub-formulas ϕ_1 and ϕ_2, which are satisfied when state variable 3 is respectively low and high (i.e., ϕ_1 is the disjunction of all regions $\mathbf{X}_l$ such that $\forall x \in \mathbf{X}_l$, $[0\ 0\ 1]x < 30$ and, similarly, ϕ_2 is the disjunction of all regions $\mathbf{X}_l$ such that $\forall x \in \mathbf{X}_l$, $[0\ 0\ 1]x > 60$). By analyzing the system with LTL formula $\Box(\Diamond\phi_1 \wedge \Diamond\phi_2)$ we search for the maximal set of initial conditions guaranteeing that trajectories of the system keep oscillating between low and high values of state variable 3.

The results obtained by applying the analysis methods described in Sects. 7.1 and 7.3 to both systems are summarized in the following table (the numbers corresponding to the approach from Sect. 7.1 are given in parentheses). The reported computation times correspond to a 20-iteration limit. The relative volumes (as a percentage of the total state space) of the identified satisfying and violating regions are reported as "% Satisfaction" and "% Violation", respectively. The number of states in the initial quotient and the quotient after all refinement steps are reported as $|X/_\sim|$ and $|\hat{X}/_\sim|$, respectively. The number of states in the product automaton after all refinement, updating and simplification is reported as $|\hat{S}_P|$ (no product simplification is involved for the method described in Sect. 7.1 and the number of states of the product automaton with the specification and the negation of the specification are reported separately). The approach from Sect. 7.3 allows us to consider only a single product automaton, unlike the one from Sect. 7.1, where both the product automaton with the formula and its negation are constructed. In addition, the minimizations of this product automaton keep the number of the states to be considered low. As a result, the computation times are improved significantly by using the formula-guided approach, while the results (in terms of identified satisfying and violating regions) are comparable between the two methods.

System	$\mathscr{W}_2$		$\mathscr{W}_3$
Specification	$\diamond 3$	$\diamond 7$	$\square(\diamond\phi_1 \wedge \diamond\phi_2)$
Computation time	29 (97) s	28.9 (96) s	17 (153) min
% Satisfaction	79.7 (79.8)	20.1 (20.1)	99.62 (99.76)
% Violation	20.1 (20.1)	79.7 (79.8)	0 (0)
$\|X/_\sim\|$	9	9	27
$\|\hat{X}/_\sim\|$	423 (463)	419 (459)	2845 (2970)
$\|\hat{S}_P\|$	7 (926,463)	7 (918,459)	17 (8910,8910)

7.4 Notes

In this chapter, which is based on [177, 181–183, 185], we showed that the methods developed in Chap. 4 for finite transition systems can be extended to autonomous discrete-time PWA systems with parameter uncertainties. We developed a method that attempts to find the largest region of initial states from which such a system satisfies an LTL formula. Motivated by the fact that finite bisimulations only exist for very limited classes of dynamical systems [83–86], central to our approach is the notion of simulation. Related, probabilistic versions of this method can be found in [1, 96, 115].

There are several simplifying assumptions that we made for simplicity of computation and presentation. First, we assumed that the LTL specification is given over the set of symbols labeling the polytopes from the definition of the PWA system (Definition 6.3). Note that, as suggested in Example 7.1, this can be easily relaxed to allow for formulas over arbitrary predicates in the state variables of the system. Indeed, given a set of such predicates, a finer partition can be constructed by adding polytopes and labeling them according to the satisfaction of the predicates (see also Sect. 1.2).

Second, both in the definition of the system (Definitions 6.3, 6.4, and 6.5, which are particular cases of 6.1) and its semantics (Definitions 6.6 and 6.7), we considered only open, full dimensional polytopes as discussed in Sect. 6.3. Third, we assumed that the A_l-matrices of the system were non-singular. While seemingly restrictive, this assumption was made purely for simplicity of presentation. The implementation of the $Post$ and Pre operators can be easily extended to semi-linear sets and singular affine functions as discussed in Sect. A.4.

While in this chapter we focused on the problem of finding largest sets of satisfying initial states, the results presented here can be used to perform "classical" LTL model checking of PWA using standard tools such as SPIN [89], NuSMV [43], PRISM [114], or DiVinE [18]. Given a PWA system $\mathscr{W}$, the exact finite quotient $T_{\mathscr{W}}/_\sim$ or the over-approximation finite quotient $\overline{T_{\mathscr{W}}/_\sim}$ can be constructed as described in Sects. 7.1 and 7.2. Then, $T_{\mathscr{W}}/_\sim$ or $\overline{T_{\mathscr{W}}/_\sim}$ can be checked against an LTL formula ϕ over the index set L of $\mathscr{W}$. In addition, the special observation Out can be used as an atomic proposition in ϕ. As a simple example, consider the problem of guaranteeing that region $\mathbf{X}$ is an invariant for all trajectories of a PWA system $\mathscr{W}$. We formulate the specification $\phi = \square\neg$Out requiring that trajectories of the system never visit the region labeled by Out. In other words, satisfying trajectories of $\mathscr{W}$ will never leave $\mathbf{X}$. We can model check the quotient $T_{\mathscr{W}}/_{\sim}$ against ϕ using standard tools and if $T_{\mathscr{W}}/_{\sim}$ satisfies ϕ we can guarantee that all trajectories of $\mathscr{W}$ satisfy the specification (the same is true for the over-approximation quotient $\overline{T_{\mathscr{W}}/_{\sim}}$). In subsequent chapters we will use this strategy to guarantee that trajectories of $\mathscr{W}$ do not leave the defined state space of the system.

It is important to note that simply model checking the quotient $T_{\mathscr{W}}/_{\sim}$ (and especially the over-approximation $\overline{T_{\mathscr{W}}/_{\sim}}$) is restrictive as discussed in Chap. 4, which motivated the development of additional refinement strategies. While the satisfaction of the formula can be guaranteed for all trajectories of $\mathscr{W}$ when the quotient satisfies the formula, nothing can be guaranteed if the formula is violated. Since, in general, $T_{\mathscr{W}}/_{\sim}$ is coarse (it contains few states), positive verification results can be rarely obtained. In Chap. 4, we extended the standard model checking methods in order to obtain more informative results. In this chapter, we showed that the construction of quotients is possible for PWA systems and the techniques from Chap. 4 can, therefore, be extended to such systems.

For the computation of an over-approximation of the set of states reachable from a certain region in a PWA system with parameter uncertainty, we defined the $\overline{Post}$ operator. Intuitively, in this treatment we assume that the parameter uncertainty is inherent in the dynamics of the system and must be handled as such as part of the analysis problem. An alternative formulation, where such uncertainty is considered as an allowed range in which the system's parameters can be tuned (corresponding to a parameter synthesis problem) is presented in Chap. 8. A treatment related to Eq. (7.6) can be found in [16], where it is shown that this over-approximation is the smallest convex set containing the states reachable from a given region.

The methods presented in this chapter involve model checking of the finite quotient $T_{\mathscr{W}}/_{\sim}$ at each step of the iterative procedure. Even though the worst case complexity of LTL model checking is exponential in the size of the formula, this upper limit is rarely reached in practice. We use an in-house model checker, which allows us to model check $T_{\mathscr{W}}/_{\sim}$ from specific states only and perform computation (such as the construction of Büchi automata) only once instead of recomputing at each step.

The construction and refinement of finite quotients used in our approach is based on polyhedral operations, which also have an exponential upper bound. Therefore, the applicability of the method depends on controlling the number of states as refinement progresses. When applied to a state X, the refinement procedure $\textsc{Refine}(T_{\mathscr{W}}/_{\sim}, X)$ can, in general, produce a maximum of 2^k subsets, where $k = |Post_{T_{\mathscr{W}}/_{\sim}}(X)|$ is the number of states reachable from state X. In the particular case when the parameters of the PWA system are fixed, only k subsets can be produced. To limit the explosion in the number of states in the quotient, we only refine states when this can improve the solution. Even so, due to its inherent complexity, this method is not suitable for the analysis of systems in high dimensions or when many iterations are required to find a solution. As expected, the method performs best if large portions of the state space can be characterized as satisfying the formula or its negation during earlier iterations.

The formula-guided refinement method for PWA systems presented in Sect. 7.3 is based on the approach for constructing formula-equivalent quotients from Sect. 4.5. The main advantage of this approach is that the specification is incorporated directly as part of the refinement procedure and the computation is performed on the product automaton, which allows for certain optimizations. Compared to the iterative refinement approaches presented in Sect. 7.1, this procedure aims at constructing formula-equivalent abstractions that might be coarser than bisimulation for certain systems (conditions under which the analysis results from the formula-guided refinement approach are exact are described in Sect. 4.5 although, in general, the overall procedure is still conservative). Additional notes on the formula-guided refinement approach (in the context of transition systems rather than PWA systems) are available in Sect. 4.6.

The analysis of PWA systems for properties such as stability, invariance and reachability has also been considered elsewhere (e.g., [28]) but the approaches presented in this chapter allow greater expressivity through LTL specifications.

The algorithms presented in this chapter were implemented in Matlab and made available for download at http://sites.bu.edu/hyness/fapas/. The tools, called FAPAS (short for “Formal analysis of Piecewise Affine Systems”) and FFAPAS (short for “Formula-guided Formal analysis of Piecewise Affine Systems”), use the MPT Toolbox [113] for polyhedral operations and the LTL2BA package [65] for the construction of Büchi automata from LTL formulas. For the two-dimensional ($N = 2$) case study presented in Example 7.4, the computation using FAPAS required under 20 s for the fixed parameter case, and under 10 min for all the uncertain parameter cases, where the limit on refinement was set to $\varepsilon = 1$ and $\varepsilon = 5$. For the three-dimensional ($N = 3$) case study presented in Example 7.6, the computation required under 20 min where $\varepsilon = 5$. This computation was performed on a 3.4 GHz machine with 1 GB of memory. The evaluation presented in Example 7.7, where the iterative and formula-guided refinement analysis approaches (described in Sects. 7.1 and 7.3) were compared, was performed on a 3.0 GHz machine with 4GB of memory.

Chapter 8
Parameter Synthesis

To study the satisfaction of LTL formulas by trajectories of a PWA system $\mathscr{W}$, in Chap. 6 we defined the embedding transition system $T_{\mathscr{W}}$, which was infinite. We showed that finite quotients of $T_{\mathscr{W}}$ can be computed through polyhedral operations and in Chap. 7 we used such quotients to develop an analysis procedure for PWA systems. We discussed how this procedure can be used to find a region of initial conditions of $\mathscr{W}$, from which all trajectories are guaranteed to satisfy the specification. Our procedure was also capable of handling PWA systems with uncertain parameters restricted to polytopic ranges. We assumed that parameter uncertainty was inherent in the system and, in order to guarantee satisfaction, trajectories must satisfy the specification regardless of the (nondeterministic) choice of parameters from the allowed range.

In this chapter, we take a different approach to study autonomous PWA systems with uncertain parameters. The parameters of the system are allowed to vary in predefined polytopic ranges as before, but in this chapter we assume that those ranges can be restricted further. In other words, we treat the parameter ranges not as an uncertainty inherent in the system, but rather as allowed ranges in which the system parameters must be tuned. Our goal is to find subsets of the allowed parameters for each region, such that the satisfaction of a specification can be guaranteed. The problem that we consider in this chapter can be formally stated as follows:

Problem 8.1 (*Parameter synthesis*) Given an autonomous, uncertain parameter, discrete-time piecewise affine system $\mathscr{W}$ (Definition 6.3) and an LTL formula ϕ over $L \cup \{\text{Out}\}$, find subsets of system parameters $\mathbf{P}^A_{l,\phi} \subseteq \mathbf{P}^A_l$ and $\mathbf{P}^c_{l,\phi} \subseteq \mathbf{P}^c_l$ for each region $l \in L$ and a non-empty set of initial states $\mathbf{X}_{0,\phi}$, such that all trajectories of the system originating there satisfy the formula under all identified parameters.

In other words, we are interested in excluding parameters from the allowed sets $\mathbf{P}^A_l$ and $\mathbf{P}^c_l$ for each region $l \in L$, for which the formula is not satisfied. As it will become clear later, for each region $l \in L$, our solution will be in the form of a union of disjoint open polytopes, which are subsets of the allowed polytopes $\mathbf{P}^A_l$ and $\mathbf{P}^c_l$. In general, it is possible that for some states, no allowed parameters can be found such that the satisfaction of the specification can be guaranteed. Such states are excluded from the

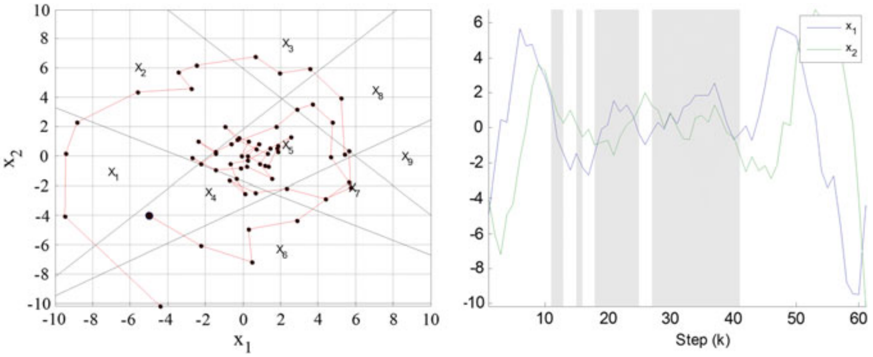
C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_8

allowed initial states of the system $\mathbf{X}_{0,\phi}$ and, therefore, the overall problem involves searching for both parameter ranges and initial states from which the satisfaction of the specification can be guaranteed, leading to the formulation of Problem 8.1.

Our approach to Problem 8.1 involves the construction of discrete abstractions in the form of finite transition systems as described in Chap. 7 and a counterexample-guided strategy allowing the identification and elimination of parameters leading to violating trajectories in the system. We first embed the autonomous PWA system $\mathscr{W}$ into $T_{\mathscr{W}}$ (see Definition 6.6 in Chap. 6), which has infinitely many states and is, in general, non-deterministic: given a state of $T_{\mathscr{W}}$ several states might be reachable under different parameters from the allowed sets. Thus, there is a correspondence between the transitions of $T_{\mathscr{W}}$ and the parameters of $\mathscr{W}$ and a possible approach to Problem 8.1 involves iteratively model checking $T_{\mathscr{W}}$ with LTL formula ϕ in order to generate counterexamples as described in Chap. 3 and eliminating such violating behavior by restricting the allowed parameters of $\mathscr{W}$ to appropriate subsets. When no additional counterexamples can be generated, the satisfaction of the specification by the system is guaranteed. Such an approach resembles iterative, counterexample-guided "debugging" of system $T_{\mathscr{W}}$, corresponding to a parameter synthesis procedure for $\mathscr{W}$. However, since $T_{\mathscr{W}}$ is infinite and model checking cannot be applied directly, our parameter synthesis strategy relies on the construction of the finite over-approximation quotient $\overline{T_{\mathscr{W}}/_{\sim}}$ (or the quotient $T_{\mathscr{W}}/_{\sim}$ for autonomous, additive uncertainty PWA systems), whose language includes the language of $T_{\mathscr{W}}$ as described in Chap. 7. Model checking can then be used to cut transitions from $\overline{T_{\mathscr{W}}/_{\sim}}$ and, correspondingly, cut sets of parameters from $\mathscr{W}$ until all system trajectories satisfy the formula.

To provide a solution to Problem 8.1, in Sect. 8.1 we discuss the problem of identifying a subset of the transitions for a finite transition system (such as $\overline{T_{\mathscr{W}}/_{\sim}}$) in order to satisfy an LTL formula. In Sects. 8.2 and 8.3, we characterize sets of parameters associated to transitions in quotients of uncertain parameter PWA systems.Then, in Sect. 8.4, we present a solution to Problem 8.1. Alternatively, in Sect. 8.5, we propose a method for the direct construction of a bisimulation quotient.

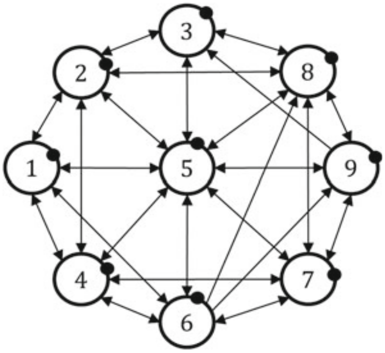
Example 8.1 A 2-dimensional ($N = 2$) PWA system is defined on the set of polytopes $\mathbf{X}_1 \ldots \mathbf{X}_9$ (L = {1…9}) shown in Fig. 8.1a. This system is similar to the one defined in Example 1.8 but is autonomous and has uncertain parameters. For each mode $l \in L$ the parameters are restricted to the ranges $\mathbf{P}_l^A$ and $\mathbf{P}_l^c$. Initially, all modes have identical allowed parameter ranges (i.e., $\forall l \in L$, $\mathbf{P}_l^A = \mathbf{P}^A$, $\mathbf{P}_l^c = \mathbf{P}^c$), which are defined by restricting the individual parameter components of $A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$ and $c = [c_1, c_2]^T$ to the ranges $0.8 \leq a_1 \leq 1$, $-0.55 \leq a_2 \leq -0.05$, $0.05 \leq a_3 \leq 0.55$, $0.8 \leq a_4 \leq 1$, $-1 \leq c_1 \leq 1$ and $-1 \leq c_2 \leq 1$.



(a) Simulated trajectory in state space (b) Simulated trajectory over time

Fig. 8.1 A simulation of an uncertain parameter PWA system in state space (**a**) and over time (**b**). The trajectory's initial state is marked by a disk in **a**, while visits to the "unsafe" region $\mathbf{X}_5$ are highlighted in *gray* in **b**. See Example 8.1 for details

Fig. 8.2 Quotient of the example PWA system from Fig. 8.1 described in Example 8.1. A dot next to a state indicates a self-loop transition, while the Out state as well as transitions to it from all other states are omitted



A simulated trajectory of the system visits region $\mathbf{X}_5$ (see Fig. 8.1a, b), which is considered "unsafe" in this example, and eventually leaves the defined state space $\mathbf{X}$. Through the rest of this chapter, we will develop a framework that allows us to formulate requirements such as "stay inside $\mathbf{X}$ and never visit $\mathbf{X}_5$", or richer behaviors such as permanent oscillations, and to enforce such specifications by restricting the allowed parameters in each mode of the system to different subsets of the allowed parameters. The method is based on the construction of finite quotients as in Fig. 8.2.

8.1 Counterexample-Guided Pruning of Finite Systems

In this section, we focus on finite transition systems with no inputs and consider the following synthesis problem:

Problem 8.2 (*Transition system synthesis*) Given a finite transition system $T = (X, \delta, O, o)$ and an LTL formula ϕ over its set of observations O, find a subset of transitions $\delta_\phi \subseteq \delta$ and a region $X_{0,\phi} \subseteq X$ such that transition system $T_\phi = (X, \delta_\phi, O, o)$ satisfies ϕ from $X_{0,\phi}$ (i.e., $T_\phi(X_{0,\phi}) \vDash \phi$).

Our goal in Problem 8.2 is to obtain a transition system T_ϕ that satisfies specification ϕ but preserves the states of T. One possible approach to Problem 8.2 is based on the idea of counterexample guided "debugging". Using the LTL model checking procedures described in Chap. 3 we generate a counterexample—a run of T satisfying the negation of the formula $\neg\phi$. If such a run exists, then we eliminate it by removing one of the traversed transitions. Then, we reiterate the process until no additional counterexamples are found, in which case we obtain the transition system T_ϕ satisfying ϕ.

In general, several different transitions are taken during the generation of a counterexample and removing any one of them will remove the counterexample from the language of the quotient. Selecting the best transition to remove at each step is nontrivial and, in general, it is not clear if removing a particular transition will lead to a solution or to the "best" solution when several solutions exist. In order to obtain more general results, we exhaustively generate all solutions by testing all transitions taken by a counterexample. This process can be seen as generating a tree (see Fig. 8.3), having as its root the initial system T, together with a set of states to be considered as initial (e.g. $X_0 = X$). Each child node in the tree represents a transition system that has the same set of states as the parent, but only a subset of its transitions. To construct the children of a node, model checking is applied to the system represented by it and a counterexample is generated. Each child represents the system obtained by removing one of the transitions traversed by the counterexample from the parent system. The exact order in which counterexamples are generated is not important because the entire tree is explored by this procedure.

When transitions are removed, a state might become blocking, resulting in the appearance of finite words in the system's language. Since the semantics of LTL are defined only over infinite words, we recursively make all blocking states unreachable by removing all their incoming transitions. This allows us to guarantee that blocking states are never reached. We must also guarantee that the system is not initialized in a blocking state and, therefore, we remove any blocking states from the initial set X_0. If the initial set X_0 becomes empty at a node of the tree, further removal of transitions will not lead to a solution and such systems are ignored.

A leaf node in the tree constructed as described above represents a transition system for which computation has stopped and no additional counterexamples can be generated. Such systems either include empty initial sets (since blocking states are removed) and are ignored, or otherwise, do not include any blocking states and therefore their languages are non-empty and contain infinite words only. Furthermore, since no additional counterexamples can be generated, all words in the languages of such transition systems are guaranteed to satisfy the LTL formula.

Some additional optimizations to the procedure described above can be performed. It is possible that the same transition system is obtained through different branches of the tree (i.e., through a different sequence of counterexamples). This might lead to unnecessary computation and therefore, the tree is always pruned so that repeated nodes are never added. Furthermore, once a satisfying transition system is found, the tree is pruned to remove all its subsets, thus preserving the richest system with most transitions. Finally, if the negation of the LTL formula is satisfied at any step of the procedure, then no satisfying runs exist in the system and further computation will not lead to solutions, so the tree is pruned accordingly.

The procedure described so far is summarized as Algorithm 15. Since T is finite and therefore contains a finite number of transitions, the termination of Algorithm 15 is guaranteed—in the worst case, all transitions of T will be eliminated. Since the entire computation tree of transition systems described above is explored, the solution to Problem 8.2 generated by this procedure is complete. When $X_0 = \emptyset$ for all leaf nodes of T_{tr} then no T_ϕ and $X_{0,\phi}$ that solve Problem 8.2 exist. In general, several satisfying transition systems (represented by leaf nodes in the computation tree) might be obtained. In this case, selecting the "best" solution (line 15) might involve additional metrics, for example by ranking all transition systems providing a solution to Problem 8.2 based on the number of transitions they include or the expressivity of their languages.

Algorithm 15 Given a finite $T = (X, \delta, O, o)$ and an LTL formula ϕ over O, find $T_\phi = (X, \delta_\phi, O, o)$ and $X_{0,\phi} \subseteq X$, such that $\delta_\phi \subseteq \delta$ and $T_\phi(X_{0,\phi}) \models \phi$

```
1: $T_{tr} := \{(T, X)\}$
2: for all leaf nodes $(T', X'_0)$, $T' = (X, \delta', O, o)$ of $T_{tr}$ where $X'_0 \neq \emptyset$ do
3:   Generate a counterexample $w \in \mathscr{L}_{T'} \cap \mathscr{L}_{\neg\phi}$
4:   for all transitions $x' \in \delta'(x)$ from $w$ do
5:     Construct $\delta''$ by removing $x' \in \delta'(x)$ from $\delta'$
6:     Construct $T'' = (X, \delta'', O, o)$
7:     Construct $X''_0 \subseteq X'_0$ and adjust $\delta''$ to ensure that $T''$ is non-blocking
8:     if $T'' \notin T_{tr}$ and $T'' \not\models \neg\phi$ then
9:       Add $(T'', X''_0)$ as a child of $(T', X'_0)$ in $T_{tr}$
10:    end if
11:  end for
12: end for
13: Select a leaf node $(T_\phi, X_{0,\phi})$ of $T_{tr}$ where $X_{0,\phi} \neq \emptyset$
14: return $(T_\phi, X_{0,\phi})$
```

Example 8.2 To illustrate the counterexample-guided pruning of finite systems, we apply Algorithm 15 to the simple system shown as the root of the tree in Fig. 8.3. The observations of the system, where each state $X = \{X_1 \dots X_5\}$ has an unique observations $o(X_i) = o_i$, are omitted in the illustration. We are interested in pruning the transitions of the system such that all trajectories keep visiting states with observation o_2 (i.e., state X_2). Therefore, we apply Algorithm 15 using the specification $\Box\Diamond o_2$. To simplify the illustration, we

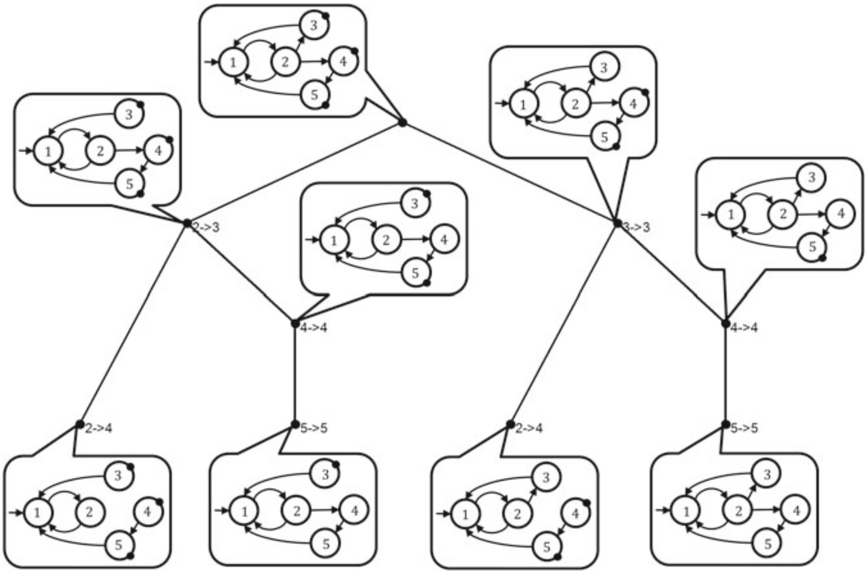


Fig. 8.3 The computation tree generated by Algorithm 15 when applied to a simple, finite transition system (illustrated at the root of the tree) using the specification $\Box\Diamond o_2$. In other words, we require that trajectories of the system (originating in state X_1) always keep visiting states with observation o_2, which in this case is only state X_2 (the unique observations $o(X_i) = o_i$ are omitted from the illustrations). At each node, the indicated transition is removed from the system and the resulting transition systems are illustrated next to each node. A dot next to a state indicates a self-transition. See Example 8.2 for additional details

consider only state X_1 as initial, instead of identifying the set of satisfying initial states while pruning the transitions.

The first counter-example generated by the algorithm is $o_1 o_2 (o_3)^{\omega}$, which can be removed by pruning the transition from state X_1 to X_2, the one from X_2 to X_3 or the self-transition at X_3. Pruning the transition between states X_1 and X_2 leads to a blocking system that does not satisfy the specification. The pruning of each one of the other two transitions is explored separately and additional counterexamples are generated.

When Algorithm 15 terminates, the entire computation tree illustrated in Fig. 8.3 is explored. The leaf nodes represent transition systems that have been pruned in such a way that the specification is satisfied.

8.2 Parameter Sets and Transitions

For simplicity of presentation, in the rest of this chapter we use a slightly different notation from the one introduced in Sect. 6.1. There, the parameter ranges for an autonomous, uncertain-parameter PWA system were defined as $\mathbf{P}_l^A$ and $\mathbf{P}_l^c$ for each region $l \in L$. For the following discussion, it is convenient to consider a notation with only a single parameter set for each region. We define the set of parameters $\mathbf{P}_l$ for each region $l \in L$ as a polytope in $\mathbb{R}^{(N^2+N)}$ that combines $\mathbf{P}_l^A$ and $\mathbf{P}_l^c$. The linear functions $A : \mathbb{R}^{(N^2+N)} \to \mathbb{R}^{N^2}$ and $c : \mathbb{R}^{(N^2+N)} \to \mathbb{R}^N$ take the first N^2 and the last N components of $p \in \mathbb{R}^{(N^2+N)}$ and form a $N \times N$ matrix and $N \times 1$ vector, respectively. The dynamics of the autonomous, uncertain parameter PWA system from Definition 6.3 are then described by

$$\mathscr{W} \ : \ x(k+1) = A(p)x(k) + c(p), \ x(k) \in \mathbf{X}_l, \ p \in \mathbf{P}_l, \ l \in L, \ k = 0, 1, \ldots \tag{8.1}$$

Remark 8.1 For the case when the parameter set $\mathbf{P}_l$ is given as an union of polytopes $\mathbf{P}_l = \bigcup_{i=1}^{n_l} \mathbf{P}_l^i$, the formulas for the calculation of $Post$ and $\overline{Post}$ of $\mathbf{X}_l$ from Eqs. (1.4) and (7.6) can be extended to $\bigcup_{i=1}^{n_l} \text{hull}(\{A(p)v + c(p), v \in V(\mathbf{X}_l), p \in V(\mathbf{P}_l^i)\}$.

In Sect. 7.2, we described the construction of the over-approximation quotient $\overline{T_{\mathscr{W}}/_{\sim}}$ of an uncertain parameter PWA system $\mathscr{W}$. In this section, we use LTL model checking to "cut" transitions from $\overline{T_{\mathscr{W}}/_{\sim}}$ until we obtain a transition system $\overline{T_{\mathscr{W}}/_{\sim}}_{\phi}$ satisfying the formula ϕ. Once a satisfying transition system is obtained, we modify the original PWA system $\mathscr{W}$ by removing parameter values in such a way that the language of the embedding transition $T_{\mathscr{W}\phi}$ system of the modified $\mathscr{W}_\phi$ is included in the language of $\overline{T_{\mathscr{W}}/_{\sim}}_{\phi}$. In other words, $\overline{T_{\mathscr{W}}/_{\sim}}_{\phi}$ becomes a quotient of the modified embedding $T_{\mathscr{W}\phi}$, which guarantees the satisfaction of the formula by the system.

The finite quotient $T_{\mathscr{W}}/_{\sim} = (X_{\mathscr{W}}/_{\sim}, \delta_{\mathscr{W},\sim}, O_{\mathscr{W}}, o_{\mathscr{W},\sim})$ is constructed so that it captures all possible transitions of the embedding $T_{\mathscr{W}} = (X_{\mathscr{W}}, \delta_{\mathscr{W}}, O_{\mathscr{W}}, o_{\mathscr{W}})$. By the definition of the embedding, transitions are included in the embedding if and only if appropriate parameters for such a transition are allowed. Therefore, we can relate the transitions present in the finite quotient to sets of allowed parameters for the PWA system. These relationships are formalized in the following and are used as part of the parameter synthesis procedures.

Definition 8.1 Given states $l, l' \in X_{\mathscr{W}}/_{\sim}$, let

$$\mathbf{P}^{l \to l'} = \{p \in \mathbb{R}^{N^2+N} \mid Post(\mathbf{X}_l, p) \cap \mathbf{X}_{l'} \neq \emptyset\} \tag{8.2}$$

denote the set of parameters for which a state from region $\mathbf{X}_l$ makes a transition into region $\mathbf{X}_{l'}$ in $\mathscr{W}$.

In other words

$$p \in \mathbf{P}^{l \to l'} \Leftrightarrow \exists x \in \mathbf{X}_l \text{ such that } A(p)x + c(p) \in \mathbf{X}_{l'} \tag{8.3}$$

and the transition $l \to l'$ is included in the quotient $T_{\mathscr{W}}/_{\sim}$ (or its over-approximation $\overline{T_{\mathscr{W}}/_{\sim}}$) if and only if some parameters from the set $\mathbf{P}^{l\to l'}$ are allowed in mode $l \in L$ of PWA system $\mathscr{W}$. This relates the transitions of the quotient with the set of parameters allowed at a given region as

$$l' \in \delta_{\mathscr{W},\sim}(l) \Leftrightarrow \mathbf{P}_l \cap \mathbf{P}^{l\to l'} \neq \emptyset. \tag{8.4}$$

Equivalently, if the set of parameters allowed for $\mathscr{W}$ in mode $l \in L$ is restricted to any subset of set

$$\mathbf{P}^{l\nrightarrow l'} = \mathbf{P}_c^{l\to l'} = \{p \in \mathbb{R}^{N^2+N} \mid Post(\mathbf{X}_l) \cap \mathbf{X}_{l'} = \emptyset\}, \tag{8.5}$$

where $\mathbf{P}_c^{l\to l'}$ denotes the complement of $\mathbf{P}^{l\to l'}$, then a transition between states l and l' is impossible in $T_{\mathscr{W}}/_{\sim}$. In other words

$$l' \notin \delta_{\mathscr{W},\sim}(l) \Leftrightarrow \mathbf{P}_l \subseteq \mathbf{P}^{l\nrightarrow l'}. \tag{8.6}$$

The equivalence from Eq. (8.6) provides a strategy for eliminating transitions of $T_{\mathscr{W}}/_{\sim}$ (or $\overline{T_{\mathscr{W}}/_{\sim}}$) by restricting the parameters of $\mathscr{W}$ to appropriate sets. This allows the implementation of the algorithm developed in Sect. 8.1 to the quotient of $\mathscr{W}$, provided that the set $\mathbf{P}^{l\nrightarrow l'}$ can be computed.

We first consider the computation of the sets from Definition 8.1 and Eq. (8.5) for autonomous, additive-uncertainty PWA systems, where the following computation is possible:

Proposition 8.1 *Given states $l, l' \in X_{\mathscr{W}}/_{\sim}$, the V-representation of set $\boldsymbol{P}^{l\to l'}$ from Definition 8.1 can be computed from the V-representations of $\boldsymbol{X}_l$ and $\boldsymbol{X}_{l'}$ as*

$$\boldsymbol{P}^{l\to l'} = \{p \in \mathbb{R}^{N^2+N} \mid A(p) = A_l, c(p) \in B\} \text{ where} \tag{8.7}$$

$$B = hull(\{v' - A_l v, v \in V(\boldsymbol{X}_l), v' \in V(\boldsymbol{X}_{l'})\})\}. \tag{8.8}$$

Proof We need to show that

$$p \in \mathbf{P}^{l\to l'} \Leftrightarrow A(p) = A_l, c(p) \in B \overset{?}{\Leftrightarrow} \exists x \in \mathbf{X}_l \text{ such that } A_l x + c(p) \in \mathbf{X}_{l'}$$

($\Leftarrow$) Let $\exists x \in \mathbf{X}_l$ such that $A_l x + c(p) \in \mathbf{X}_{l'}$. Let $m = |V(\mathbf{X}_l)|$ and $x = \Sigma_{i=1}^m \lambda_i v_i$, where $0 < \lambda_i < 1$ for all $i = 1, \ldots, m$ and $\Sigma_{i=1}^m \lambda_i = 1$. Let $n = |V(\mathbf{X}_{l'})|$ and $x' = \Sigma_{j=1}^n \mu_j v_j'$, where $0 < \mu_j < 1$ for all $j = 1, \ldots, n$ and $\Sigma_{j=1}^n \mu_j = 1$. Then,

$$A_l \Sigma_{i=1}^m \lambda_i v_i + c(p) = \Sigma_{j=1}^n \mu_j v_j' \Rightarrow c(p) = \Sigma_{j=1}^n \mu_j v_j' - A_l \Sigma_{i=1}^m \lambda_i v_i =$$
$$= \Sigma_{i=1}^m \Sigma_{j=1}^n \lambda_i \mu_j (v_j' - A_l v_i) \Rightarrow c(p) \in hull(\{v' - A_l v, v \in V(\mathbf{X}_l),\ v' \in V(\mathbf{X}_{l'})\})$$

($\Rightarrow$) Let $c(p) \in hull(\{v' - A_l v, v \in V(\mathbf{X}_l),\ v' \in V(\mathbf{X}_{l'})\})$, $c(p) = \Sigma_{i=1}^m \Sigma_{j=1}^n \nu_{ij}$ $(v_j' - A_l v_i)$, where $0 < \nu_{ij} < 1$, $i = 1, \ldots, m$, $j = 1, \ldots, n$ and $\Sigma_{i=1}^m \Sigma_{j=1}^n \nu_{ij} = 1$.

Let $\lambda_i = \Sigma_{j=1}^{m} \nu_{ij}$ and $\mu_j = \Sigma_{i=1}^{n} \nu_{ij}$. Of course, $0 < \lambda_i < 1$ for all $i = 1, \ldots, m$, $0 < \mu_j < 1$ for all $j = 1, \ldots, n$ and $\Sigma_{i=1}^{m} \lambda_i = \Sigma_{j=1}^{n} \mu_j = \Sigma_{i=1}^{m} \Sigma_{j=1}^{n} \nu_{ij} = 1$. Then, for $x = \Sigma_{i=1}^{m} \lambda_i v_i$ and $x' = \Sigma_{j=1}^{n} \mu_j v'_j$ we have $A_l x + c(p) = x'$ and therefore $\exists x \in \mathbf{X}_l$ such that $A_l x + c(p) \in \mathbf{X}_{l'}$. ■

In the equations above, A_l represents the (singleton) parameter value of the matrix corresponding to mode $l \in L$. Following from Proposition 8.1, the removal of a transition between states l and l' in $T_{\mathscr{W}}/_\sim$ amounts to the restriction of the allowed parameters in mode l of $\mathscr{W}$ to any set

$$\mathbf{P}'_l \subseteq \mathbf{P}_l \setminus \mathbf{P}^{l \to l'}, \tag{8.9}$$

and all computation can be performed using polyhedral operations.

For autonomous PWA systems where the matrix component of the parameters is allowed to vary, the set $\mathbf{P}^{l \to l'}$ cannot be computed easily. Instead, we focus directly on the computation of a subset

$$\underline{\mathbf{P}}^{l \nrightarrow l'} \subseteq \mathbf{P}^{l \nrightarrow l'}, \tag{8.10}$$

guaranteeing that a transition from all states in region $\mathbf{X}_l$ to any state in region $\mathbf{X}_{l'}$ is impossible in $T_{\mathscr{W}}$. This allows the removal of transition $l \to_{e\sim} l'$ in $T_{\mathscr{W}}/_\sim$ by restricting the parameters of $\mathscr{W}$ to a set $\mathbf{P}'_l \subseteq (\mathbf{P}_l \cap \underline{\mathbf{P}}^{l \nrightarrow l'})$. In the following, we develop two different strategies leading to the the construction of $\underline{\mathbf{P}}^{l \nrightarrow l'}$, which offer a tradeoff between computational complexity and the accuracy of the obtained approximation.

Our first computational strategy is based on the following observation: a transition from l to l' is impossible in $T_{\mathscr{W}}/_\sim$ under parameters $p \in \mathbf{P}_l$ (i.e., $Post(\mathbf{X}_l, p) \cap \mathbf{X}_{l'} = \emptyset$) when one of the inequalities defining the H-representation of region $\mathbf{X}_{l'}$ is violated by all successor states $A(p)x + c(p) \in Post(\mathbf{X}_l)$, $x \in \mathbf{X}_l$.

Proposition 8.2 *Given states $l, l' \in X_{\mathscr{W}}/_\sim$, the H-representation of a conservative under-approximation of set $\boldsymbol{P}^{l \nrightarrow l'}$ from Eq. (8.5) can be computed from the V-representation of $\boldsymbol{X}_l$ and the H-representation of $\boldsymbol{X}_{l'} = \{x \in \mathbb{R}^N \mid h_i^T x \leq k_i, i = 1, \ldots, n\}$ as*

$$\underline{\boldsymbol{P}}^{l \nrightarrow l'} = \bigcup_{i=1}^{n} \{p \in \mathbb{R}^{(N^2+N)} \mid h_i^T \hat{v} p > k_i, \forall v \in V(\boldsymbol{X}_l), \forall i = 1, \ldots, n\} \subseteq \boldsymbol{P}^{l \nrightarrow l'}, \tag{8.11}$$

where, for any state $x \in \mathbb{R}^N$ and parameters $p \in \mathbb{R}^{N^2+N}$, the $\hat{x}$ operator reshapes vector x so that $\hat{x} p = A(p)x + c(p)$.

Proof

$$\text{Let } p \in \underline{\mathbf{P}}^{l \nrightarrow l'} = \bigcup_{i=1}^{n} \{p \in \mathbb{R}^{(N^2+N)} \mid h_i^T \hat{v} p > k_i, \forall v \in V(\mathbf{X}_l), \forall i = 1, \ldots, n\} =$$

$$
\begin{aligned}
&= \bigcup_{i=1}^{n} \{p \in \mathbb{R}^{(N^2+N)} \mid h_i^T (A(p)v + c(p)) > k_i, \forall v \in V(\mathbf{X}_l)\} \Rightarrow \\
&\Rightarrow \exists i, 1 \leq i \leq n, \forall v \in V(\mathbf{X}), h_i^T (A(p)v + c(p)) > k_i \Leftrightarrow \\
&\Leftrightarrow \forall x \in \mathbf{X}, h_i^T (A(p)v + c(p)) > k_i \Leftrightarrow \\
&\Leftrightarrow \nexists x \in \mathbf{X}, A(p)x + c(p) \in \mathbf{X}_{l'} \Leftrightarrow \\
&\Leftrightarrow Post(\mathbf{X}_l, p) \cap \mathbf{X}_{l'} = \emptyset
\end{aligned}
$$

■

Using the computation from Proposition 8.2, the under-approximation $\underline{\mathbf{P}}^{l \nrightarrow l'}$ is obtained as a union of polyhedral sets and allows the computation of a set of allowed parameters $\mathbf{P}'_l = (\mathbf{P}_l \cap \underline{\mathbf{P}}^{l \nrightarrow l'})$ for mode l of $\mathscr{W}$, guaranteeing the removal of the transition between states l and l' in $T_{\mathscr{W}}/_{\sim}$.

A second strategy for the computation of an under-approximation to $\mathbf{P}^{l \nrightarrow l'}$ for some $l, l' \in X_{\mathscr{W}}/_{\sim}$ is based on the set difference operator $\ominus$ introduced in Definition A.8 and the observation that, given parameters $p \in \mathbf{P}_l$,

$$
Post(\mathbf{X}_l, p) \cap \mathbf{X}_{l'} \neq \emptyset \Leftrightarrow \mathbf{0} \in Post(\mathbf{X}_l, p) \ominus \mathbf{X}_{l'} \tag{8.12}
$$

Proposition 8.3 *Given states $l, l' \in X_{\mathscr{W}}/_{\sim}$ and the V-representations of $\boldsymbol{X}_l$ and $\boldsymbol{X}_{l'}$, the V-representation of a conservative under-approximation of the set from Eq. (8.5) can be computed as*

$$
\underline{\boldsymbol{P}}^{l \nrightarrow l'} = \{p \in \mathbb{R}^{N^2+N} \mid h^T \hat{v} p < h^T v', \forall v \in V(\boldsymbol{X}_l), \forall v' \in V(\boldsymbol{X}_{l'})\} \subseteq \boldsymbol{P}^{l \nrightarrow l'} \tag{8.13}
$$

for any $h \in \mathbb{R}^N$. Furthermore, a less conservative under-approximation can be computed as

$$
\underline{\boldsymbol{P}}^{l \nrightarrow l'} = \bigcup_{h \in H} \{p \in \mathbb{R}^{N^2+N} \mid h^T \hat{v} p < h^T v', \forall v \in V(\boldsymbol{X}_l), \forall v' \in V(\boldsymbol{X}_{l'})\} \subseteq \boldsymbol{P}^{l \nrightarrow l'} \tag{8.14}
$$

where $H \subseteq \mathbb{R}^N$.

Proof To guarantee that all points from a polytope $x \in \mathbf{X}_l$ satisfy a linear inequality $h^T x \leq k$ it is necessary and sufficient to guarantee that the inequality is satisfied at all vertices $v_x \in V(\mathbf{X}_l)$

$$
\forall v \in V(\mathbf{X}_l), h^T v \leq k \Leftrightarrow \forall x \in \mathbf{X}_l, h^T x \leq k \tag{8.15}
$$

Let $p \in \underline{\mathbf{P}}^{l \nrightarrow' l}$. Then, for some $h \in \mathbb{R}^N$ we have

$$
\begin{aligned}
&\forall v \in V(\mathbf{X}_l), \forall v \in V(\mathbf{X}_l), h^T \hat{v} p < h^T v' \Rightarrow \\
&\Rightarrow \forall v \in V(\mathbf{X}_l), \forall v \in V(\mathbf{X}_l), h^T (A(p)v + c(p) - v') < 0,
\end{aligned}
$$

which, from Eq. (8.15), implies that

$$\forall x \in Post(\mathbf{X}_l, p) \ominus \mathbf{X}_{l'}, h^T x < 0 \Rightarrow \mathbf{0} \notin Post(\mathbf{X}_l, p) \ominus \mathbf{X}_{l'}.$$

From Eq. (8.12) this guarantees that $Post(\mathbf{X}_l, p) \cap \mathbf{X}_{l'} = \emptyset$. Extending this computation to a union of sets as in Eq. (8.14) is straightforward, since the result holds for any $h \in H$, where $H \subset \mathbb{R}^N$ is a set of samples. ■

The quality of the under-approximation $\underline{\mathbf{P}}^{l \nrightarrow l'}$ computed through Proposition 8.3 improves as the number of samples in set H from Eq. (8.14) is increased. To obtain better coverage, the set H can be obtained through uniform sampling of rotation groups. While, in terms of computational complexity, this approach could be more costly than obtaining the under-approximation $\underline{\mathbf{P}}^{l \nrightarrow l'}$ through Proposition 8.2, it allows control over the quality of the approximation by selecting the number of samples in H.

8.3 Transient Parameters

In this section, we consider the particular case when a self loop (i.e., a transition $l \to l$ for some $l \in X_{\mathscr{W}}/_\sim$) must be removed during the application of the procedure described in Sect. 8.1 to the quotient $T_{\mathscr{W}}/_\sim$. Although such transitions can be removed by restricting the parameters of PWA system $\mathscr{W}$ to appropriate subsets using the computation from Proposition 8.2 or Proposition 8.3, this might lead to very conservative results. In fact, such an approach would require that whenever a self loop at a state $l \in X_{\mathscr{W}}/_\sim$ is removed, all trajectories of $\mathscr{W}$ leave region $\mathbf{X}_l$ in a single step, which is hard to enforce. Instead, in the following we derive conditions guaranteeing that if the parameters are restricted appropriately, trajectories of the system leave the region eventually, but not necessarily in a single step, which leads to a less conservative parameter synthesis procedure.

Definition 8.2 A subset of parameters $\mathbf{P}'_l \subseteq \mathbf{P}_l$ is *transient* at mode $l \in L$ of PWA system $\mathscr{W}$ if and only if, for all trajectories $x(0)x(1)x(2)\ldots$ such that $x(0) \in \mathbf{X}_l$, there exists a finite $k > 1$ such that $x(0), \ldots, x(k) \in \mathbf{X}_l$ and $x(k+1) \notin \mathbf{X}_l$.

Remark 8.2 The above definition of transient parameters is related to the definition of stuttering inputs from Definition 9.2 in Sect. 9.3 from Chap. 9. Propositions 8.4 and 8.5 are stated in more general forms as Propositions 9.5 and 9.6, respectively, and their proofs are therefore postponed to Chap. 9.

Restricting the parameters of a PWA system $\mathscr{W}$ to a transient subset $\mathbf{P}'_l \subseteq \mathbf{P}_l$ guarantees that region $\mathbf{X}_l$ becomes a transient region for all trajectories of the system. While this does not eliminate the self loop at state $l \in X_{\mathscr{W}}/_\sim$ in the quotient $T_{\mathscr{W}}/_\sim$, it guarantees that this loop cannot be followed infinitely often along any trajectory. In the particular case when a specification expressed as a formula ϕ from the LTL$\setminus\bigcirc$ fragment is considered, this allows us to safely ignore transition $l \to l$ without violating the inclusion of the system language within the quotient language. Formally, this result follows from the stutter equivalence (i.e., an equivalence with respect to the order of observations visited along all trajectories but not the exact number of repetitions of each observation) of the quotients constructed with and without stutter transitions such as the self loop at state l. In the following, we derive a necessary and sufficient condition characterizing a subset of parameters as transient and use it to develop a computational procedure based on polyhedral operations for restricting the parameters of a PWA system to transient subsets.

Proposition 8.4 *A subset of parameters $\boldsymbol{P}'_l \subseteq \boldsymbol{P}_l$ is transient at mode $l \in L$ of PWA system $\mathscr{W}$ if and only if*

$$\mathbf{0} \notin hull(\{(A(p) - I)v + c(p), \forall v \in V(\boldsymbol{X}_l), \forall p \in V(\boldsymbol{P}'_l)\}) \tag{8.16}$$

where $I \in \mathbb{R}^{N \times N}$ denotes the identity matrix.

The characterization from Proposition 8.4 provides a computational procedure that allows us to check if a set of parameters is transient. Since this condition is necessary and sufficient, in the following we use it to restrict a set of parameters to a transient subset.

Proposition 8.5 *Given state $l \in X_{\mathscr{W}}/_{\sim}$ and the V-representation $V(\boldsymbol{X}_l)$ of $\boldsymbol{X}_l$, the subset of parameters with the following H-representation*

$$\boldsymbol{P}'_l = \{p \in \mathbb{R}^{N^2+N} \mid h^T \hat{v} p < h^T v, \forall v \in V(\boldsymbol{X}_l)\} \tag{8.17}$$

is transient at mode $l \in L$ for any $h \in \mathbb{R}^N$.

Note that the computation from Proposition 8.4 is similar to the one from Proposition 8.3 but does not allow a larger transient set of parameters to be computed as a union of smaller ones as in Eq. (8.14). However, sampling might still be beneficial in order to find larger subsets of parameters that are transient. In other words, given the set of samples H, generated by uniformly sampling rotation groups as in Proposition 8.3, we seek to find an $h \in H$ that maximizes the volume of the intersection

$$\mathbf{P}'_l = \mathbf{P}_l \cap \{p \in \mathbb{R}^{N^2+N} \mid h^T \hat{v} p < h^T v, \forall v \in V(\mathbf{X}_l)\}). \tag{8.18}$$

8.4 Parameter Synthesis for PWA Systems

Using the results described in Sects. 8.3 and 8.2 and the method discussed in Sect. 8.1, a solution to Problem 8.1 can be obtained as follows. Given PWA system $\mathscr{W}$ with embedding $T_{\mathscr{W}}$ and LTL formula ϕ, the quotient $T_{\mathscr{W}}/_{\sim}$ (or its over-approximation $\overline{T_{\mathscr{W}}/_{\sim}}$) is constructed as described in Chap. 7 and Algorithm 15 is applied to it. Whenever a transition between states l and l' is removed as part of the computation, the set of system parameters of $\mathscr{W}$ is restricted using Propositions 8.2 or 8.3. In the particular case when $l = l'$ and ϕ is from the $\text{LTL}\backslash\bigcirc$ fragment, the self loop $l \rightarrow l$ is removed through Proposition 8.5 in order to obtain less conservative results.

The computation from Algorithm 15 is guaranteed to terminate returning a tree, where the root node represents the quotient $T_{\mathscr{W}}/_{\sim}$ (or $\overline{T_{\mathscr{W}}/_{\sim}}$) and every other node represents a transition system that has the same states as $T_{\mathscr{W}}/_{\sim}$ (and $\overline{T_{\mathscr{W}}/_{\sim}}$) but only a subset of its transitions (see Sect. 8.1). Each transition system from this tree is, in fact, a quotient of a PWA system where parameters have been restricted to appropriate subsets using the computation described so far. In other words, each node in the tree represents a triple $(T'_{\mathscr{W}}/_{\sim}, \mathscr{W}', \mathbf{X}'_0)$ where

i. $X'_0 \subseteq X'_{\mathscr{W}}/_{\sim}$ is a set of initial states from the quotient $T'_{\mathscr{W}}/_{\sim}$,
ii. the difference between $\mathscr{W}$ and $\mathscr{W}'$ is in the parameter sets where, for each mode $l \in L$, the parameter set of $\mathscr{W}'$ is a subset of the allowed parameter set for $\mathscr{W}$ (i.e., $\mathbf{P}'_l \subseteq \mathbf{P}_l$ for some $l \in L$),
iii. the difference between $T'_{\mathscr{W}}/_{\sim}$ and $T_{\mathscr{W}}/_{\sim}$ is the set of transitions where, for each state $l \in X_{\mathscr{W}}/_{\sim}$, the set of reachable states from l in $T'_{\mathscr{W}}/_{\sim}$ is a subset of the set of reachable states in $T_{\mathscr{W}}/_{\sim}$, and
iv. T'_W is the embedding of $\mathscr{W}'$, while $T'_{\mathscr{W}}/_{\sim}$ is the quotient of T'_e induced by the observational equivalence relation $\sim$.

Therefore, $T'_{\mathscr{W}}/_{\sim}$ simulates $T'_{\mathscr{W}}$ and the language inclusion $\mathscr{L}_{T'_{\mathscr{W}}} \subseteq \mathscr{L}_{T'_{\mathscr{W}}/_{\sim}}$ is guaranteed. When a leaf node in the tree returned by Algorithm 15 represents the triple $(T'_{\mathscr{W}}/_{\sim}, \mathscr{W}', X'_0)$, we know that $T'_{\mathscr{W}}/_{\sim}$ satisfies ϕ from region X_0 (see Sect. 8.1) and, therefore, $\mathscr{W}'$ satisfies ϕ from region $con(X_0)$, which provides a solution to Problem 8.1.

The PWA system structure allows different regions to share the same set of parameters, for example, whenever the initial partitioning of X is refined to accommodate a specification. Therefore, it is possible that additional transitions besides the target one are removed at each step of the computation described above and, to account for this, we reconstruct the quotient every time parameters are cut. If, during the computation, the set of allowed parameters for some mode $l \in L$ of $\mathscr{W}$ becomes empty, then we consider state l and all states from region $con(l)$ as blocking in $T_{\mathscr{W}}/_{\sim}$ and $T_{\mathscr{W}}$, respectively, and we make them unreachable by restricting the parameters of all other regions accordingly. Whenever an over-approximation quotient is constructed, a spurious transition might appear in place of one that was already eliminated but we prevent this by enforcing that once a transition is removed it never reappears in the quotient.

The computation described so far in this section is summarized as Algorithm 16, which covers the most general case of parameter uncertainty in $\mathscr{W}$. This procedure is guaranteed to terminate but, while our solution to the purely discrete problem discussed in Sect. 8.1 was complete, Algorithm 16 might not find a solution to Problem 8.1 even when one exists. Due to the construction and use of (over-approximation) quotients to guide the removal of parameters and the computation of under-approximate parameter sets for the removal of transitions, the overall method becomes conservative, but the correctness of the solution (when one is found) is guaranteed. As for Algorithm 16, the procedure summarized in Algorithm 16 returns a tree, where several solutions to Problem 8.1 might be possible and are represented by its leaf nodes. Selecting the "best" solution is a non-trivial problem, and might depend on the application. It is possible to introduce additional constraints (such as requiring that a particular transition is present) or compare total number of transitions of the solutions, since more reachable states from the initial one with more transitions result in a richer language.

Both the number of states and transitions in the embedding $\overline{T_{\mathscr{W}}/_\sim}$ contribute to the complexity of Algorithm 16. A high dimensional system with many regions of different dynamics would be embedded in a transition system with a large number of states. This, together with the size of the LTL formula affects the time required to perform model checking on the system. The number of transitions in the original embedding, on the other hand, depends on the dynamics of the system and determines how many iterative model checking steps must be performed during the generation of the computation tree. As a result, Algorithm 16 can perform well even on high dimensional systems, as long as the total number of transitions is low or only few transitions must be removed to reach a solution.

Algorithm 16 Given a PWA system $\mathscr{W}$ (embedded in $T_{\mathscr{W}}$) and an LTL formula ϕ, identify a region $\mathbf{X}_{0,\phi}$ and construct a system $\mathscr{W}^{\phi}$ that satisfies ϕ from $\mathbf{X}_{0,\phi}$ and, for each mode $l \in L$, its parameters are a subset of the allowed parameters for $\mathscr{W}$.

```
Construct quotient $\overline{T_{\mathscr{W}}/_\sim}$ from $T_{\mathscr{W}}$
Let $X_0 := X_{\mathscr{W}}/_\sim \setminus \{\text{Out}\}$
Initialize $T_{tr} := \{(\overline{T_{\mathscr{W}}/_\sim}, \mathscr{W}, X_0)\}$
for all leaf nodes $(\bar{T}'_{\mathscr{W}}/_\sim, \mathscr{W}', X'_0)$ of $T_{tr}$ where $X'_0 \neq \emptyset$ do
  Generate a counterexample $w \in \mathscr{L}_{\bar{T}'_{\mathscr{W}}/_\sim} \cap \mathscr{L}_{\neg\phi}$
  for all transitions $l$ to $l'$ from $w$ do
    if $l = l'$ and $\phi \in \text{LTL}\backslash\bigcirc$ then
      Construct $\mathbf{P}''_l$ as a transient subset of $\mathbf{P}'_l$
    else
      $\mathbf{P}''_l := (\mathbf{P}'_l \cap \underline{\mathbf{P}}^{l \nrightarrow l'})$
    end if
    Construct PWA system $\mathscr{W}''$ (the parameter set for mode $l \in L$ is $\mathbf{P}''_l$)
    Construct quotient $\bar{T}''_{\mathscr{W}}/_\sim$ of $\mathscr{W}''$
    Enforce no transition between $l$ and $l'$ in $\bar{T}''_{\mathscr{W}}/_\sim$
    Recursively make all blocking states of $\bar{T}''_{\mathscr{W}}/_\sim$ unreachable and restrict the parameters of
      $\mathscr{W}''$ appropriately
    Construct initial set $X''_0 \subseteq X'_0$ by excluding blocking initial states
    if no node from $T_{tr}$ has $\bar{T}''_{\mathscr{W}}/_\sim$ as its quotient then
      add $(\bar{T}''_{\mathscr{W}}/_\sim, \mathscr{W}'', X''_0)$ as a child of $(\bar{T}'_{\mathscr{W}}/_\sim, \mathscr{W}', X_0)$ in $T_{tr}$
    end if
  end for
end for
select a leaf node $(\bar{T}_{\mathscr{W}^{\phi}}/_\sim, \mathscr{W}^{\phi}, X_{0,\phi})$ of $T_{tr}$ where $X_{0,\phi} \neq \emptyset$
return $\mathscr{W}^{\phi}$ and $\mathbf{X}_{0,\phi} = con(X_{0,\phi})$
```

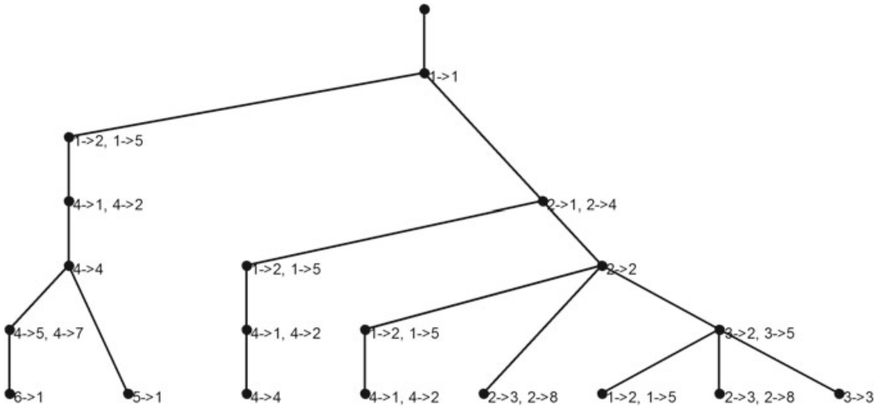


Fig. 8.4 First 20 nodes of the computation tree generated by Algorithm 16 when applied to the PWA system from Example 8.1 (simulated trajectory in Fig. 8.1 and quotient in Fig. 8.2). In contrast to the transition system synthesis algorithm application illustrated in Fig. 8.3, here multiple transitions are often removed at each node as a result of restricting the parameters of the system as explained in Sect. 8.2

Remark 8.3 In order to prevent unnecessary computation, we can first model-check the quotient $T_{\mathscr{W}}/_{\sim}$ from each initial state against $\neg\phi$. The satisfaction of this negation from an initial state implies that there are no satisfying trajectories originating there under any of the allowed parameters and a solution to Problem 8.1 will not be found by our parameter synthesis procedure.

Example 8.3 To illustrate the parameter synthesis for PWA systems approach presented in this chapter, we apply Algorithm 16 to the PWA system defined in Example 8.1. The initial system has uncertain parameters and, for some parameter values, trajectories of the system visit region $\mathbf{X}_5$, which we consider unsafe. In addition, it is possible that trajectories of the initial system leave $\mathbf{X}$, which is required to be invariant. Finally, we require that trajectories of the system oscillate, visiting regions $\mathbf{X}_1$ and $\mathbf{X}_9$, while avoiding region $\mathbf{X}_5$. To capture all these properties, we use the specification $\Box(\neg 5 \wedge \neg \text{Out} \wedge \Diamond 1 \wedge \Diamond 9)$. The invariant $\mathbf{X}$ is enforced as a pre-processing step by restricting the parameters of the system to suitable sub-sets and therefore the specification reduces to $\Box(\neg 5 \wedge \Diamond 1 \wedge \Diamond 9)$.

The first 20 nodes (corresponding to different quotients) of the computation tree generated by Algorithm 16 are shown in Fig. 8.4. Compared to the counterexample guided pruning of transitions in a finite system (illustrated in Fig. 8.3), multiple transitions are removed in this case since the parameters of the PWA system are restricted at each step and the quotient is recomputed.

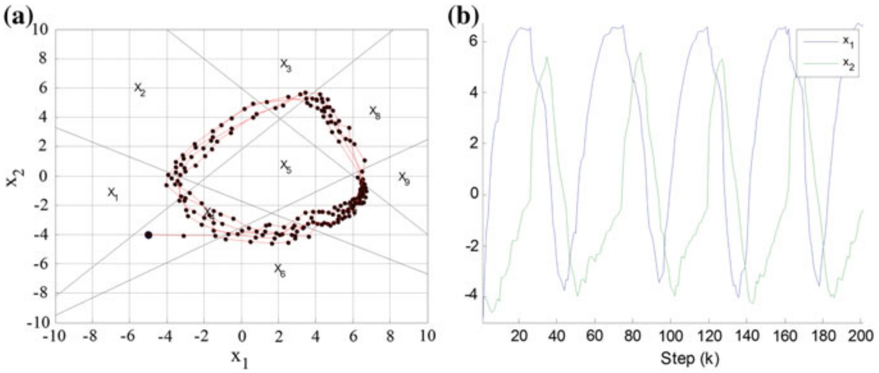
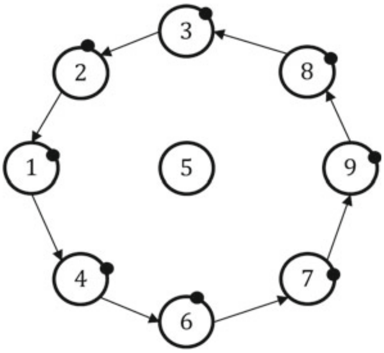


Fig. 8.5 A satisfying trajectory in state space (**a**) and time (**b**) for the system synthesized in Example 8.3. The initial state is shown as a disk in **a**

Fig. 8.6 Resulting quotient after parameter synthesis for the system in Example 8.3. All transitions to state Out are also removed to enforce **X** as an invariant, while self-transitions are preserved but the parameters are restricted as stuttering to enforce the progress of trajectories of the system



Only a single satisfying PWA system is identified in this case after the algorithm terminates and a simulated trajectory of the resulting system is shown in Fig. 8.5. This illustrates that the system satisfies the specification and the trajectory oscillates without visiting region $\mathbf{X}_5$ or leaving the invariant **X**. The quotient of the resulting system is shown in Fig. 8.6, which illustrates that a number of transitions were removed in order to satisfy the specification compared to the quotient of the initial system shown in Fig. 8.2.

Interestingly, for this example no solutions are identified unless the computation of transient parameters described in Sect. 8.3 is utilized. Indeed, the quotient shown in Fig. 8.6 retains self-transitions at a number of the states and the trajectory illustrated in Fig. 8.5 makes several steps within the same region before making a transition to an adjacent region along the oscillations.

8.5 Parameter Synthesis Using Bisimulations

The construction of finite bisimulation quotients for PWA systems is attractive due to their equivalence with respect to model checking. One possible approach to the computation of bisimulation quotients involves the construction and subsequent refinement of simulation quotients as discussed in Chap. 7. However, this procedure is not guaranteed to terminate since, in general, a PWA system might not admit a finite bisimulation quotient. Here, we consider an orthogonal approach where, instead of attempting to construct a bisimulation quotient for a given PWA system directly (which might be infeasible), we first restrict the system's behavior by restricting its sets of parameters appropriately. While, in doing so, we sacrifice possible behavior and restrict the richness of the system's language, we obtain a system for which a bisimulation quotient can be computed trivially.

A bisimulation quotient constructed using the approach outlined above is not directly guaranteed to satisfy a given specification (Problem 8.1). However, as it will become clear in the following, each transition of such a bisimulation quotient corresponds to a specific set of parameters. Then, a particular property can be easily enforced (for example by pruning transitions and the corresponding parameter sets using the approach described in Sect. 8.1). In addition, the bisimulation quotient can be used interchangeably with the restricted-parameter PWA system for model checking or analysis.

As before, we begin by studying the relation between the set of parameters of $\mathscr{W}$ and transitions in the quotient $T_{\mathscr{W}}/_{\sim}$.

Definition 8.3 Given states $l, l' \in X_{\mathscr{W}}/_{\sim}$, let

$$\mathbf{P}^{l\Rightarrow l'} = \{p \in \mathbb{R}^{(N^2+N)} \mid Post(\mathbf{X}_l, p) \subseteq \mathbf{X}_{l'}\} \tag{8.19}$$

denote the set of parameters for which every state $x \in \mathbf{X}_l$ makes transitions to states in $\mathbf{X}_{l'}$ only.

In other words,

$$p \in \mathbf{P}^{l\Rightarrow l'} \Leftrightarrow \forall x \in \mathbf{X}_l,\, A(p)x + c(p) \in \mathbf{X}_{l'}. \tag{8.20}$$

From Eq. (8.19) it follows that restricting the allowed parameters in mode $l \in L$ of PWA system $\mathscr{W}$ to any subset $\mathbf{P}'_l \subseteq (\mathbf{P}_l \cap \mathbf{P}^{l\Rightarrow l'})$ guarantees that only the deterministic transition $l \to l'$ is possible in $T_{\mathscr{W}}/_{\sim}$.

Proposition 8.6 *If, for each mode $l \in L$, the parameters of PWA system $\mathscr{W}$ are restricted to the subset $\boldsymbol{P}'_l = \boldsymbol{P}_l \cap (\bigcup_{l' \in L} \boldsymbol{P}^{X_l \Rightarrow X_{l'}})$, then the quotient $T_{\mathscr{W}}/_{\sim}$ is a bisimulation quotient.*

Proof The proof follows immediately from Definitions 8.3, 1.4 (bisimulation) and 6.6 (embedding of PWA systems). ■

After the parameters of $\mathscr{W}$ are restricted as described in Proposition 8.6, the quotient $T_{\mathscr{W}}/_{\sim}$ is still computable as before but its transitions are implicitly induced by

$l \to l' \Leftrightarrow \mathbf{P}'_l \cap \mathbf{P}^{l \Rightarrow l'} \neq \emptyset$. Note that $T_{\mathscr{W}}/_{\sim}$ is in general nondeterministic since the parameter set $\mathbf{P}_l \cap \mathbf{P}^{\mathbf{X}_l \Rightarrow \mathbf{X}_{l'}}$ might be nonempty for several different $l' \in L$.

Proposition 8.7 *Given states $l, l' \in X_{\mathscr{W}}/_{\sim}$, the H-representation of parameter set $\boldsymbol{P}^{l \Rightarrow l'}$ from Definition 8.3 can be computed from the V-representation $con(l) = \boldsymbol{X}_l = hull(V(\boldsymbol{X}_l))$ and the H-representation $con(l') = \boldsymbol{X}_{l'} = \{x \in \mathbb{R}^N \mid h_i^T x \leq k_i, i = 1, \ldots, n\}$ as*

$$\boldsymbol{P}^{l \Rightarrow l'} = \{p \in \mathbb{R}^{(N^2+N)} \mid h_i^T(A(p)v + c(p)) < k_i, \forall v \in V(\boldsymbol{X}_l), \forall i = 1, \ldots, n\}$$

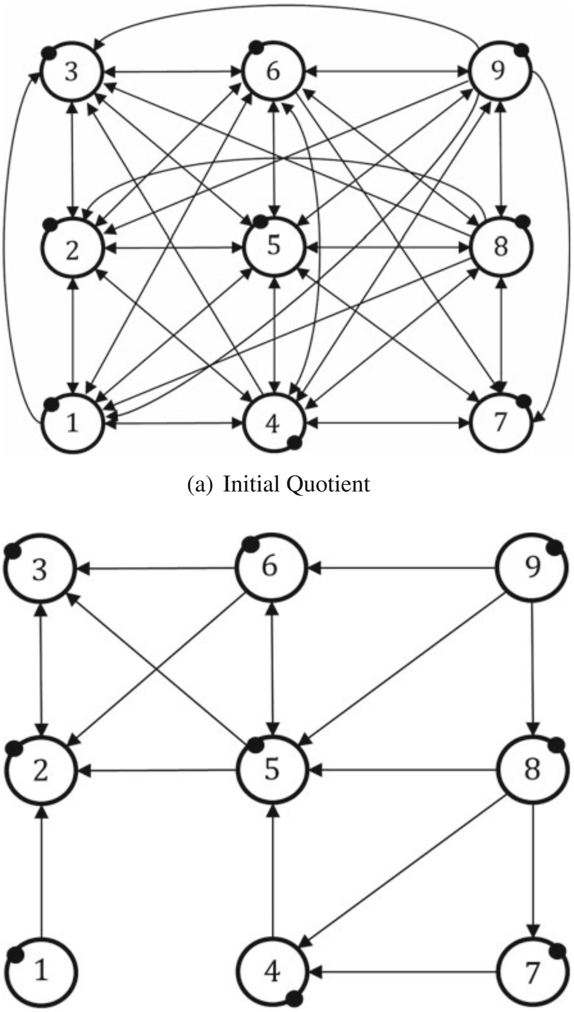
Proof ($\Rightarrow$) Let $p \in \mathbb{R}^{N^2+N}$ such that $\forall v \in V(\mathbf{X}_l), \forall i = 1, \ldots, n, h_i^T(A(p)v + c(p)) < k_i$. From Eq. (8.15), this implies that $\forall i = 1, \ldots, n,\ \forall x \in \mathbf{X}_l, h_i^T(A(p)x + c(p)) < k_i$ or, equivalently, $\forall x \in \mathbf{X}_l, A(p)x + c(p) \in \mathbf{X}_{l'}$ and therefore $Post(\mathbf{X}_l, p) \subseteq \mathbf{X}_{l'}$.

($\Leftarrow$) Let $p \in \mathbb{R}^{N^2+N}$ such that $Post(\mathbf{X}_l, p) \subseteq \mathbf{X}_{l'}$. Then, $hull(\{A(p)v + c(p), v \in V(\mathbf{X}_l)\}) \subseteq \mathbf{X}_{l'}$, which implies that $\forall i = 1, \ldots, n,\ \forall v \in V(\mathbf{X}_l), h_i^T(A(p)v + c(p)) < k_i$. ■

The computation from Proposition 8.7 allows us to restrict the parameters of $\mathscr{W}$ as described in Proposition 8.6, which guarantees that the quotient $T_{\mathscr{W}}/_{\sim}$ is, in fact, a bisimulation quotient of $T_{\mathscr{W}}$.

Example 8.4 We illustrate the parameter synthesis using bisimulations method presented in this chapter on the two-dimensional ($N = 2$) PWA system defined in Example 7.4. As before, we assume hyper-rectangular parameter sets, where each parameter is restricted to a range defined by ±50% of the fixed parameter values from Example 7.4 (parameters equal to 0 are restricted to the range $[-1.0E^{-2}, 1.0E^{-2}]$). Furthermore, the parameters for each region $l \in L$ are restricted to ensure that $\mathbf{X}$ is an invariant of all trajectories of the system. This can be accomplished using a computation similar to the one described in Proposition 8.7, where the entire $\mathbf{X}$ is used as the target (instead of a specific $\mathbf{X}_{l'}$) to restrict the parameters for each region $\mathbf{X}_l$. The quotient of this system under the uncertain parameters is illustrated in Fig. 8.7a.

We apply the method described in Sect. 8.5 in order to modify the parameters of the system and obtain a bisimulation quotient directly. A graphical representation of the resulting bisimulation quotient is shown in Fig. 8.7. As expected, a number of transitions of the system are lost when parameters are restricted to the smaller sets guaranteeing bisimulation. For example, the parameter set for region $\mathbf{X}_7$ retains around 50% of its original volume, while the one for region $\mathbf{X}_5$ retains only 0.017% when only parameters guaranteeing bisimulation are preserved. However, due to the language equivalence between the bisimulation quotient and the initial PWA system, the two systems can be used equivalently for model checking.



(a) Initial Quotient

(b) Bisimulation Quotient (after parameter synthesis)

Fig. 8.7 **a** Quotient of the PWA system from Example 7.4 with uncertain parameters restricted to ±50% of the fixed-parameter values. **b** Bisimulation quotient after the application of the parameter synthesis approach described in Sect. 8.5. Self-loop transitions at a state are indicated by a dot next to it. See Example 8.4 for details

8.6 Notes

In this chapter, which is based on [178], we showed that an iterative procedure can be used to obtain subsets of parameters for a PWA system, such that an LTL formula is satisfied (Problem 8.1).

The counterexample-guided pruning method presented in Sect. 8.1 focuses on the problem of enforcing that a finite transition system satisfies an LTL specification. The approach involves iteratively generating counterexamples through model-checking (Chap. 3) and removing these violating behaviours by pruning the transitions of the system. This approach is similar to counterexample-guided abstraction refinement (CEGAR) [44] in the sense that counterexamples are generated iteratively. However, while with CEGAR counterexamples are removed from the language of the system by refining the abstraction (unless a counterexamples is not spurious, in which case the analysis terminates), here we directly remove each counterexample by pruning transitions. More generally, enforcing a particular property for a finite system could be considered as an instance of the supervisory control problem (e.g., where a separate controller is synthesized to disable unwanted transitions and behaviors of the system) but here we consider autonomous systems, while the removal of transitions corresponds to a memoryless control strategy. The counterexample-guided pruning approach from this chapter is also different from other synthesis methods, since the states of the system remain unchanged and only its transitions are modified. This restriction is required in order to apply the approach for synthesis of PWA systems, where only the system's parameters (and the transitions they induced) can be restricted.

To apply this approach to PWA systems, in Sect. 8.2 we derived conditions guaranteeing that a given transition is present or not present in the quotient. For systems with additive parameter uncertainty only these conditions are exact, while otherwise the results are conservative (e.g., the sets of parameters preventing a transition between two states of the quotient are under-approximated). Two strategies were proposed for computing an under-approximation of parameter subsets guaranteeing that a transition between two regions (states in the quotient) is not possible. The computation from Proposition 8.3 could be more costly than the one from Proposition 8.2 but it allows some control over the quality of the approximation (e.g., by sampling of rotation groups, for example using the methods described in [133]). Using the properties from Sect. 8.2, the removal of a transition in the finite quotient can then be enforced in the PWA system by restricting the parameter sets to a suitable subset. In general, several transitions are removed at each step as a result of restricting the system's parameters, which was illustrated in Fig. 8.4.

In addition, weaker conditions were derived for restricting the parameters of a system to subsets that enforce transitions eventually rather than in a single step or, equivalently, enforcing that unwanted self-transitions at specific states of the quotient cannot be taken infinitely. While this computation can be applied only for a subset of specifications (LTL formulas without the next operator), this enables the synthesis of satisfying PWA systems in the cases where no subset of parameters can be found otherwise as in Example 8.4. The computation of such parameter sets is related to approaches for dealing with liveness properties (e.g., [24]) and stuttering behaviour in the quotient, which is discussed in more detail in Sect. 9.3.

In Sect. 8.5, we also derived conditions guaranteeing that only deterministic transitions exist in the quotient for certain subsets of parameters. This allowed us to restrict the parameters of a PWA system to subsets guaranteeing that a finite bisimulation quotient is constructed. While this restricts the possible behaviour of the system as illustrated in Example 8.4, it provides a strategy for obtaining finite bisimulation quotients, which can be used equivalently with the modified, concrete PWA systems (with restricted parameters) for analysis or model checking.

In the controls and formal methods communities, the parameter synthesis problem has been considered for a variety of systems. For example, in [86] parametric constraints were derived for guaranteeing the correctness of hybrid automaton models with respect to safety properties and a counterexample-guided parameter synthesis approach was developed for linear hybrid automata in [62]. In [53] a parameter synthesis approach was developed for dynamical systems modeled as nonlinear differential equations based on sensitivity analysis and search over initial conditions. Parameter synthesis has also been explored for other continuous time hybrid models such as piecewise multi-affine systems [22–25]. These approaches share many similarities with the methods for discrete time systems presented in this chapter, including the construction of finite abstractions as well as the identification of parameter sets inducing particular transitions or transient behaviors in such quotients.

The counterexample-guided pruning approach employed in this chapter for finite transition systems to drive the parameter synthesis in Sect. 8.4 is complete (even though our solution to the overall PWA parameter synthesis method is conservative) but computationally intensive as it explores the entire computation tree (where each node corresponded to a PWA system with different parameter sets and quotient) exhaustively. As formulated, the method performs best when the system has “little” violating behavior consisting of short executions, which can be removed after the generation of a small number of counterexamples. However, the conditions relating the parameter ranges of a PWA systems to transitions in the resulting quotient can also serve as a foundation for more efficient approaches, based on improved strategies for enforcing the satisfaction of an LTL property in a finite transition system or various heuristics. For example, in [24] a parameter synthesis approach similar to a binary search was developed. However, this approach cannot be applied directly here, since the set of parameters cannot always be partitioned into a subset inducing a given transition and one that does not due to the use of under-approximations. Also, the parameter sets for each state region were independent rather than linked through a common parameter. Alternatively, parallel model checking methods could improve the scalability of parameter synthesis approaches as demonstrated in [17].

The algorithms presented in this chapter were implemented as a software tool for Parameter Synthesis for Piecewise Affine Systems ParSyPAS, which is freely downloadable at http://hyness.bu.edu/software. The tool is built under MATLAB, and uses our in-house LTL model checker described in [103], LTL2BA [65] for the conversion of an LTL formula to a Büchi automaton, and the MPT toolbox [113] for polyhedral operations. The evaluation presented in Example 8.4 was performed on a 3.6 GHz machine with 32 GB of memory and required around 70 min for the synthesis of a satisfying PWA system.

Chapter 9
Temporal Logic Control

In Chaps. 7 and 8 we considered autonomous PWA systems. We showed that finite quotients of such systems can be computed and, based on this, developed methods for analysis and parameter synthesis from temporal logic specifications. Unlike the systems we discussed in previous chapters, which evolved autonomously, in this chapter we consider PWA control systems, which can be affected externally by applying a control signal. Then, it is possible to guarantee the satisfaction of a specification by trajectories of a PWA control system if an appropriate control signal is applied. More specifically, in this chapter we focus on fixed-parameter, PWA control systems and consider the following problem:

Problem 9.1 (*PWA Control*) Given a fixed-parameter PWA control system $\mathscr{W}$ (Definition 6.2) and an LTL formula ϕ over $L \cup \{\text{Out}\}$, find a control strategy such that all trajectories of the closed loop system satisfy ϕ.

Using Definition 6.6, Problem 9.1 becomes an LTL control problem, where we seek a feedback control strategy $(\mathbf{X}_0, \Omega)$ for the infinite, deterministic embedding transition system $T_{\mathscr{W}}$. In this chapter, we assume that the state of the system cannot be measured precisely or the applied inputs are corrupted by noise and, therefore, seek control strategies that are robust both with respect to measured state and applied input. As a result, the set of initial states $\mathbf{X}_0$ and control function Ω from the feedback control strategy $(\mathbf{X}_0, \Omega)$ (see Definition 5.1) are defined in terms of the initial regions of $\mathscr{W}$ and no state refinement is performed (such approaches will be considered in subsequent chapters). In Chap. 5 we discussed the problem of controlling a finite, possibly nondeterministic transition system from LTL specifications. To apply the methods presented there to Problem 9.1, we develop an approach based on the construction of a finite abstraction for $T_{\mathscr{W}}$, which we refer to as the control transition system T_c such that a control strategy generated for T_c using the algorithms from Chap. 5 can be adapted for $T_{\mathscr{W}}$.

More specifically, we construct T_c through a two step process. In the first step, by using the state equivalence relation induced by the polytopes from the definition of the PWA system, we construct a quotient $T_{\mathscr{W}}/_{\sim}$, which has finitely many states but an infinite set of inputs. This part of the procedure is similar to the methods we used

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_9

in Chaps. 7 and 8, with the exception that fixed-parameter, PWA control systems are considered, resulting in the construction of quotient transition systems with inputs. Thus, these results provide an extension of the methods for the construction of finite abstraction of autonomous PWA systems from Chap. 7 to a control framework. Once the quotient $T_{\mathscr{W}}/_{\sim}$ is constructed, we then define an equivalence relation in the control space, which leads to the construction of the finite control transition system T_c, which is suitable for the methods from Chap. 5. The solution to Problem 9.1 is obtained by implementing the control strategy for T_c as a feedback control automaton for the initial PWA system that reads the index of the region visited at each step and supplies the next input. As it will become clear later, our approach is robust in the sense that the closed loop system is guaranteed to satisfy the specification, even when state measurements and applied inputs are perturbed.

The remainder of this chapter is organized as follows. In Sect. 9.1 we define the control transition system T_c and outline an algorithm for its computation. In Sect. 9.2 we show how the methods from Chap. 5 are applied to T_c to formulate a solution to Problem 9.1. In Sect. 9.3 we discuss a strategy for reducing the conservatism of the overall method by characterizing the *stuttering behavior* (self transitions at a state of T_c that can be taken infinitely in T_c but do not correspond to real trajectories of $T_{\mathscr{W}}$) inherent in the construction of the control transition system. This approach is related to the characterization of transient regions and parameters from Chap. 8 as well as the well known Zeno behavior and addresses a major source of conservatism with the abstraction procedure from Chap. 7.

9.1 Control Abstraction

In this section, we define the finite control transition system $T_c = (X_c, \Sigma_c, \delta_c, O_c, o_c)$ for the (infinite) embedding $T_{\mathscr{W}} = (X_{\mathscr{W}}, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W}}, O_{\mathscr{W}}, o_{\mathscr{W}})$ (Definition 6.6) and present an algorithm for its computation. In Sect. 9.2 we will show how the control methods from Chap. 5 are applied to T_c to solve Problem 9.1.

9.1.1 Definition

As in Chap. 7, the observation map $o_{\mathscr{W}}$ of $T_{\mathscr{W}}$ induces an observational equivalence relation $\sim$ over the set of states $X_{\mathscr{W}}$. However, the systems we considered before were autonomous and here, the quotient transition system $T_{\mathscr{W}}/_{\sim} = (X_{\mathscr{W}}/_{\sim}, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W},\sim}, O_{\mathscr{W}}, o_{\mathscr{W},\sim})$ induced by $\sim$ has an infinite set of inputs $\Sigma_{\mathscr{W}} = \mathbf{U}$, which is preserved from $T_{\mathscr{W}}$. The set of transitions of $T_{\mathscr{W}}/_{\sim}$ is defined as $l' \in \delta_{\mathscr{W},\sim}(l, u)$ if and only if there exist $u \in \mathbf{U}$, $x \in \mathbf{X}_l$ and $x' \in \mathbf{X}'$ such that $x' = \delta_{\mathscr{W}}(x, u)$. Note that in general $T_{\mathscr{W}}/_{\sim}$ is nondeterministic, even though $T_{\mathscr{W}}$ is deterministic. Indeed, for a state of the quotient $l \in X_{\mathscr{W}}/_{\sim}$ it is possible that different states $x, x' \in \mathbf{X}_l$ have transitions in $T_{\mathscr{W}}$ to states from different equivalence classes under the same input. The set of observations $O_{\mathcal{W}} = L$ of $T_{\mathcal{W}}/_{\sim}$ is preserved from $T_{\mathcal{W}}$ and the observation map $o_{\mathcal{W},\sim}$ is identity.

The transition map $\delta_{\mathcal{W},\sim}$ can be related to the transitions of $T_{\mathcal{W}}$ by using the $Post$ operator defined in Eq. (7.3):

$$\delta_{\mathcal{W},\sim}(l, u) = \{l' \in X_{\mathcal{W}}/_{\sim} \mid Post(\mathbf{X}_l, \{u\}) \cap \mathbf{X}_{l'} \neq \emptyset\}, \tag{9.1}$$

for all $l \in X_{\mathcal{W}}/_{\sim}$ and $u \in \Sigma_{\mathcal{W}}$. For each state $l \in X_{\mathcal{W}}/_{\sim}$, we define an equivalence relation $\approx_l$ over the set of inputs $\Sigma_{\mathcal{W}}$ as

$$(u_1, u_2) \in \approx_l \text{ iff } \delta_{\mathcal{W},\sim}(l, u_1) = \delta_{\mathcal{W},\sim}(l, u_2). \tag{9.2}$$

In other words, inputs u_1 and u_2 are equivalent at state l if they produce the same set of transitions in $T_{\mathcal{W}}/_{\sim}$. Let U_l^C, $l \in L$, $C \in 2^{X_{\mathcal{W}}/\sim}$ denote the equivalence classes of $\Sigma_{\mathcal{W}}$ in the partition induced by the equivalence relation $\approx_l$:

$$U_l^C = \{u \in \Sigma_{\mathcal{W}} \mid \delta_{\mathcal{W},\sim}(l, u) = C\} \tag{9.3}$$

Let $c(U_l^C)$ be an input in U_l^C such that

$$\forall u \in \Sigma_{\mathcal{W}}, d(c(U_l^C), u) < \varepsilon \Rightarrow u \in U_l^C, \tag{9.4}$$

where $d(u, u')$ denotes the distance between inputs $u, u' \in \Sigma_{\mathcal{W}}$ and ε is a predefined parameter specifying the robustness of the control strategy. As it will become clear in Sect. 9.1.2, $d(u, u')$ is the Euclidian distance in $\mathbb{R}^M$ and $c(U_l^C)$ can be computed as the center of a sphere inscribed in U_l^C.

Initially, the states of T_c are the observations of $T_{\mathcal{W}}$ (i.e., $X_c = L$). The set of inputs available at a state $l \in L$ is $\Sigma_c^l = \{c(U_l^C) \mid C \in 2^{X_{\mathcal{W}}/\sim}\}$ and the transition map is $\delta_c(l, c(U_l^C)) = C$. In general, it is possible that at a given state l, $\Sigma_c^l = \emptyset$, in which case state l is blocking. As it will become clear in Sect. 9.1.2, such states are removed from the system in a recursive procedure together with their incoming transitions and therefore $X_c \subseteq L$. The set of X_c observations and observation map of T_c are preserved from $T_{\mathcal{W}}/_{\sim}$, which completes the construction of the control transition system.

Following from the construction described so far, the control transition system T_c is a finite transition system and a control strategy for it can be generated using the methods presented in Chap. 5. In Sect. 9.2, we will show that a control strategy for T_c can be adapted as a robust control strategy (with respect to knowledge of exact state and applied input) for the infinite $T_{\mathcal{W}}$. This will allow us to use T_c as part of our solution to Problem 9.1.

9.1.2 Computation

Initially, the states of the control transition system T_c are simply the labels L of the polytopes from the definition of the PWA system (Definition 6.2). To complete its construction, we need to be able to compute the set of inputs Σ_c^l available at each state $l \in X_c$ and the transition map δ_c, while eliminating the states that are unreachable in order to guarantee that T_c remains non-blocking.

Given a polytope $\mathbf{X}_l$ from the definition of the PWA system, let

$$\Sigma^l = \{u \in \Sigma_{\mathscr{W}} \mid Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \subseteq \mathbf{X}\} \tag{9.5}$$

be the set of all inputs guaranteeing that all states from $\mathbf{X}_l$ transit inside $\mathbf{X}$ (i.e., Σ^l is the set of all inputs allowed at l). In other words, regardless which $u \in \Sigma^l$ and $x \in \mathbf{X}_l$ are selected, x will transit inside $\mathbf{X}$ under u in $T_{\mathscr{W}}$. Then, in order to guarantee that $\mathbf{X}$ is an invariant for all trajectories of the system (an assumption that we made in the formulation of Problem 9.1) it is sufficient to restrict the set of inputs Σ_c^l available at each state $l \in X_c$ to $\Sigma_c^l \subseteq \Sigma^l$.

Proposition 9.1 *Let $\boldsymbol{X} = \{x \in \mathbb{R}^N \mid Hx < K\}$ be the H-representation of the polytope $\boldsymbol{X}$ from the definition of the PWA system (Definition 6.2). Then, Σ^l is a polytope with the following H-representation:*

$$\Sigma^l = \{u \in \boldsymbol{U} \mid \forall v \in V(\boldsymbol{X}_l), HB_l u < K - H(A_l v + c_l)\}, \tag{9.6}$$

where $V(\boldsymbol{X}_l)$ denotes the set of vertices of $\boldsymbol{X}_l$.

Proof Note that the set defined in Eq. (9.5) can be equivalently written as

$$\Sigma^l = \{u \in \mathbf{U} \mid \forall x \in \mathbf{X}_l, A_l x + B_l u + c_l \in \mathbf{X}\} \tag{9.7}$$

Let $u \in \mathbf{U}$ such that $\forall x \in \mathbf{X}_l.\ A_l x + B_l u + c_l \in \mathbf{X}$. Then,

$$\begin{aligned} \forall x \in \mathbf{X}_l.\ H(A_l x + B_l u + c_l) < K \Rightarrow \forall x \in \mathbf{X}_l.\ HB_l u < K - H(A_l x + c_l) \Rightarrow \\ \Rightarrow \forall v \in V(X_l).\ HB_l u < K - H(A_l v + c_l) \end{aligned}$$

Let $u \in \mathbf{U}$ such that $\forall v \in V(\mathbf{X}_l).\ HB_l u < K - H(A_l v + c_l)$. Then, $\forall v \in V(\mathbf{X}_l).\ A_l v + B_l u + c_l \in \mathbf{X}$. Let $m = |V(\mathbf{X}_l)|$, $x = \Sigma_{i=1}^m \lambda_i v_i$, where $v_i \in V(\mathbf{X}_l)$, $0 < \lambda_i < 1$ for all $i = 1, \ldots, m$ and $\Sigma_{i=1}^m \lambda_i = 1$. Then,

$$\begin{aligned} A_l x + B_l u + c_l = A_l \Sigma_{i=1}^m \lambda_i v_i + B_l u + c_l \ = \ \Sigma_{i=1}^m \lambda_i (A_l v_i + B_l u + c_l) \in \mathbf{X} \Rightarrow \\ \Rightarrow \forall x \in \mathbf{X}_l.\ A_l x + B_l u + c_l \in \mathbf{X} \end{aligned}$$

■

The set of states reachable from state l in $T_{\mathscr{W}}/_\sim$ under the allowed inputs is

$$Post_{T_{\mathscr{W}}/_\sim}(l, \Sigma^l) = \{l' \in X_{\mathscr{W}}/_\sim \mid Post_{T_{\mathscr{W}}}(\mathbf{X}_l, \Sigma^l) \cap \mathbf{X}_{l'} \neq \emptyset\} \tag{9.8}$$

and can be computed using polyhedral operations, since

$$Post_{T_{\mathscr{W}}}(\mathbf{X}_l, \Sigma^l) = A_l\mathbf{X}_l + B_l\Sigma^l + c_l. \tag{9.9}$$

Given a polytope $\mathbf{X}_l$ from the definition of the PWA system (Definition 6.2) and an arbitrary polytope $\mathbf{X}'$, let

$$U^{\mathbf{X}_l \to \mathbf{X}'} = \{u \in \Sigma_{\mathscr{W}} \mid Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \cap \mathbf{X}' \neq \emptyset\} \tag{9.10}$$

denote the set of all inputs under which $T_{\mathscr{W}}$ can make a transition from a state in $\mathbf{X}_l$ to a state inside $\mathbf{X}'$. Equivalently, applying any input $u \in \mathbf{U}$, $u \notin U^{\mathbf{X}_l \to \mathbf{X}'}$ guarantees that $T_{\mathscr{W}}$ will not make a transition inside $\mathbf{X}'$, from any state in $\mathbf{X}_l$. The following proposition states that $U^{\mathbf{X}_l \to \mathbf{X}'}$ is a polyhedral set that can be computed from the V- (vertex) and H- (hyperplane) representations of $\mathbf{X}_l$ and $\mathbf{X}'$:

Proposition 9.2 *Let H and K be the matrices in the H-representation of the following polytope:*

$$\{\hat{x} \in \mathbb{R}^N \mid \exists x \in \boldsymbol{X}_l,\ A_l x + \hat{x} + c_l \in \boldsymbol{X}'\} \tag{9.11}$$

Then $U^{\boldsymbol{X}_l \to \boldsymbol{X}'}$ is a polytope with the following H-representation:

$$U^{\boldsymbol{X}_l \to \boldsymbol{X}'} = \{u \in \boldsymbol{U} \mid HB_l u < K\} \tag{9.12}$$

Proof The set defined in Eq. (9.11) is a polytope with the following V-representation:

$$\text{hull}\{v' - (Av + c) \mid v \in V(\mathbf{X}_l),\ \ v' \in V(\mathbf{X}')\} \tag{9.13}$$

Let $x \in \mathbf{X}_l$ such that $A_l x + \hat{x} + c_l \in \mathbf{X}'$. Let $m = |V(\mathbf{X}_l)|$ and $x = \Sigma_{i=1}^m \lambda_i v_i$, where $0 < \lambda_i < 1$ for all $i = 1, \ldots, m$ and $\Sigma_{i=1}^m \lambda_i = 1$. Let $n = |V(\mathbf{X}')|$ and $x' = \Sigma_{j=1}^n \mu_j v'_j$, where $0 < \mu_j < 1$ for all $j = 1, \ldots, n$ and $\Sigma_{j=1}^n \mu_j = 1$. Then,

$$A_l \Sigma_{i=1}^m \lambda_i v_i + \hat{x} + c_l = \Sigma_{j=1}^n \mu_j v'_j \Rightarrow \hat{x} = \Sigma_{j=1}^n \mu_j v'_j - A_l \Sigma_{i=1}^m \lambda_i v_i - c_l =$$
$$= \Sigma_{i=1}^m \Sigma_{j=1}^n \lambda_i \mu_j (v'_j - (A_l v_i + c_l)) \Rightarrow \hat{x} \in \text{hull}\{v' - (Av + c) | v \in V(\mathbf{X}_l),\ \ v' \in V(\mathbf{X}')\}$$

Let $\hat{x} = \Sigma_{i=1}^m \Sigma_{j=1}^n \nu_{ij}(v'_j - (A_l v_i + c_l))$, where $0 < \nu_{ij} < 1, i = 1, \ldots, m, j = 1, \ldots, n$ and $\Sigma_{i=1}^m \Sigma_{j=1}^n \nu_{ij} = 1$. Let $\lambda_i = \Sigma_{j=1}^m \nu_{ij}$ and $\mu_j = \Sigma_{i=1}^n \nu_{ij}$. Of course, $0 < \lambda_i < 1$ for all $i = 1, \ldots, m$, $0 < \mu_j < 1$ for all $j = 1, \ldots, n$ and $\Sigma_{i=1}^m \lambda_i = \Sigma_{j=1}^n \mu_j = \Sigma_{i=1}^m \Sigma_{j=1}^n \nu_{ij} = 1$. Then, for $x = \Sigma_{i=1}^m \lambda_i v_i$ and $x' = \Sigma_{j=1}^n \mu_j v'_j$ we have $A_l x + \hat{x} + c_l = x'$ and therefore $\exists x \in \mathbf{X}_l$ such that $A_l x + \hat{x} + c_l \in \mathbf{X}'$. To conclude the proof of Proposition 9.2, let H, K be the matrices in the H-representation of the set defined in Eq. (9.11) and note that the set defined in Eq. (9.10) can be equivalently written as

$$U^{\mathbf{X}_l \to \mathbf{X}'} = \{u \in \mathbf{U} \mid \exists x \in \mathbf{X}_l,\, A_l x + B_l u + c_l \in \mathbf{X}'\} \tag{9.14}$$

■

Proposition 9.3 *Given a state $l \in X_c$ and a set of states $C \in 2^{X_c}$, the set U_l^C from Eq. (9.3) can be computed as follows:*

$$U_l^C = \bigcap_{l' \in C} U^{\mathbf{X}_l \to \mathbf{X}_{l'}} \setminus \bigcup_{l'' \notin C} U^{\mathbf{X}_l \to \mathbf{X}_{l''}} \tag{9.15}$$

Proof From Eqs. (9.1) and (9.3) we have

$$\begin{aligned} U_l^C &= \{u \in \Sigma_{\mathscr{W}} \mid \forall l' \in C,\, Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \cap \mathbf{X}_{l'} \neq \emptyset, \\ &\quad \forall l'' \notin C,\, Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \cap \mathbf{X}_{l''} = \emptyset\} = \\ &= \{u \in \Sigma_{\mathscr{W}} \mid \forall l' \in C,\, Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \cap \mathbf{X}_{l'} \neq \emptyset\} \setminus \\ &\quad \{u \in \Sigma_{\mathscr{W}} \mid \exists l'' \notin C,\, Post_{T_{\mathscr{W}}}(\mathbf{X}_l, u) \cap \mathbf{X}_{l''} \neq \emptyset\} = \\ &= \bigcap_{l' \in C} U^{\mathbf{X}_l \to \mathbf{X}_{l'}} \setminus \bigcup_{l'' \notin C} U^{\mathbf{X}_l \to \mathbf{X}_{l''}} \end{aligned}$$

■

We can guarantee that if a state l' is not reachable from state l in $T_{\mathscr{W}}/_{\sim}$ (i.e., $l' \notin Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l)$) then $U^{\mathbf{X}_l \to \mathbf{X}_{l'}} = \emptyset$ and therefore, $U_l^C = \emptyset$ if $C \nsubseteq Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l)$ and otherwise the computation in Eq. (9.15) reduces to

$$U_l^C = \bigcap_{l' \in C} U^{\mathbf{X}_l \to \mathbf{X}_{l'}} \setminus \bigcup_{l'' \in Post_{T_{\mathscr{W}}/_{\sim}}(l, \Sigma^l) \setminus C} U^{\mathbf{X}_l \to \mathbf{X}_{l''}} \tag{9.16}$$

A non-empty input region U_l^C is in general nonconvex but can always be represented as a finite union of open polytopes (see Eq. (9.16)). In order to guarantee the robustness of the control strategy (as described in Sect. 9.1.1) we only include input sets that are "large enough" (i.e., $r(U_l^C) > \varepsilon$, where ε is a predefined robustness parameter and $r()$ is the radius of the Chebyshev ball (Definition A.9). Note that in general this approach might be conservative, since a sphere inscribed in a union of polytopes from U_l^C might have a larger radius. Following from the results presented in this section, the control transition system T_c can be computed using polyhedral operations only (the computation is summarized in Algorithm 17).

Remark 9.1 It is possible to reduce the size of T_c after it is initially constructed without sacrificing solutions. More "nondeterminism" available at a state does not result in more winning strategies for Algorithm 17, while at the same time unnecessarily increases the complexity of the method. Formally, let $u_1 = c(U_l^{C_1})$ and $u_2 = c(U_l^{C_2})$ where $C_1, C_2 \in 2_c^X$, $C_1 \subseteq C_2$ be inputs of T_c available at state $l \in X_c$ (i.e., $\{u_1, u_2\} \subseteq \Sigma_c^l$). If input u_2 is used in a control strategy, then the specification

Algorithm 17 T_c = CONTROL- TS($\mathscr{W}$, ε): Construct control transition system T_c

```
$X_c := L$
for each $l \in X_c$ do
  $\Sigma_c^l := \Sigma^l$ [Eq. (9.5)]
  compute $Post_{T_{\mathscr{W}}/\sim}(l, \Sigma_c^l)$ [Eq. (9.8)]
  for each $C \subseteq Post_{T_{\mathscr{W}}/\sim}(l, \Sigma_c^l)$ do
    compute $U_l^C$ [Eq. (9.16)]
    if $r(U_l^C) > \varepsilon$ then
      include input $c(U_l^C)$ in $\Sigma_c^l$
      include transition $\delta_c(l, c(U_l^C)) = C$
    end if
  end for
  if $\Sigma_c^l = \emptyset$ then
    recursively make state $l$ unreachable and set $X_c := X_c \setminus l$
  end if
end for
$\Sigma_c = \bigcup_{l \in X_c} \Sigma_c^l$
return $T_c$
```

is satisfied regardless of which state $l' \in C_2$ is visited in the next step. Clearly, the same holds for input u_1 since C_1 is a subset of C_2 but keeping both inputs is unnecessary. Therefore, at each state $l \in X_c$ we set $\Sigma_c^{ls} = \Sigma_c^{ls} \setminus u_2$ if $u_1, u_2 \in \Sigma_c^{ls}$ or $\Sigma_c^{lu} = \Sigma_c^{lu} \setminus u_2$ if $u_1, u_2 \in \Sigma_c^{lu}$ when the property described above holds.

9.2 LTL Control of PWA Systems

In Sect. 9.1 we defined the control transition system T_c as a finite abstraction of the infinite $T_{\mathscr{W}}$ and showed that it can be computed using polyhedral operations. In Sect. 5.1 of Chap. 5, we presented an approach for controlling finite transition systems (such as T_c) from specifications given as LTL formulas. In this section, we show that a control strategy generated for T_c can be adapted to the infinite $T_{\mathscr{W}}$, while the satisfaction of LTL formulas by the closed loop systems is preserved, which completes the solution to Problem 9.1.

Definition 9.1 (*PWA control strategy*) A control strategy (X_0^c, Ω^c) for T_c can be translated into a control strategy (X_0, Ω) for $T_{\mathscr{W}}$ as follows. The initial set $X_0^c \subseteq X_c$ gives the initial set $X_0 = \bigcup_{l \in X_0^c} \mathbf{X}_l \subseteq X_{\mathscr{W}}$. Given a finite sequence of states $x(0) \ldots x(k)$ where $x(0) \in X_0$, the control function is defined as $\Omega(x(0) \ldots x(k)) = \Omega^c(o_{\mathscr{W}}(x(0)) \ldots o_{\mathscr{W}}(x(k)))$.

Proposition 9.4 *Given a control strategy (X_0^c, Ω^c) for T_c translated as a control strategy (X_0, Ω) for $T_{\mathscr{W}}$, $\mathcal{L}_{T_{\mathscr{W}}}(X_0, \Omega) \subseteq \mathcal{L}_{T_c}(X_0^c, \Omega^c)$, which implies that if $T_c(Q_0^c, \Omega^c)$ satisfies an arbitrary LTL formula ϕ, then so does $T_{\mathscr{W}}(X_0, \Omega)$.*

Proof Transition systems T_c and $T_{\mathscr{W}}$ have the same set of observations and therefore the same LTL formula can be interpreted over both systems. Let $x(0) \in X_0$ be an initial state for $T_{\mathscr{W}}$. Its observation is $x^c(0) = o_{\mathscr{W}}(x(0))$, which is a satisfying initial state for T_c (i.e., $x^c(0) \in X_0^c$). The next input to be applied in T_c is given by the control function ($u^c(0) = \Omega(x^c(0)) \in \Sigma_c \subset \Sigma_{\mathscr{W}}$) and we can guarantee that regardless which input $u(0) \in \Sigma_{\mathscr{W}}$ such that $d(u(0), u^c(0)) < \varepsilon$ is applied in $T_{\mathscr{W}}$, we have $\delta_{\mathscr{W}}(x(0), u(0)) \in \delta_c(x^c(0), u^c(0))$. This shows that a finite fragment of a word in $T_{\mathscr{W}}(Q_0, \Omega)$ is also a finite fragment of a word in $T_c(Q_0^c, \Omega^c)$ and the rest of the proof follows by induction. ■

The overall solution to Problem 9.1 consists of constructing the control transition system T_c (Sect. 9.1), finding a satisfying control strategy for T_c and adapting it to the original $T_{\mathscr{W}}$, or equivalently PWA system (Definition 9.1), which from Proposition 9.4 guarantees the correctness of the solution. It is important to note that a control strategy generated using this approach is robust with respect to knowledge of the exact state of the system (i.e., the control strategy depends on the observation of a state rather than the state itself). In addition, the control strategy is robust with respect to perturbations in the applied input bounded by ε, which can be used as a tuning parameter.

9.3 Conservatism and Stuttering Behavior

In Chap. 5 we described a solution to the problem of controlling a finite and possibly nondeterministic transition system from LTL specifications. In order to generate a control strategy for an infinite transition system such as $T_{\mathscr{W}}$ (Problem 9.1) we described the construction of a finite control abstraction T_c in Sect. 9.1. However, due to *spurious trajectories* (i.e., trajectories of T_c not present in $T_{\mathscr{W}}$) we cannot guarantee that a control strategy will be found for T_c even if one exists for $T_{\mathscr{W}}$ and therefore, the overall method is conservative. In Chap. 7, we eliminated spurious trajectories through state refinement but the states of T_c cannot be refined. Indeed, the control strategies we consider in this chapter cannot differentiate between states $x_1, x_2 \in X_{\mathscr{W}}$ when $o(x_1) = o(x_2)$ and therefore the states of T_c must satisfy $o_c(l_1) = o_c(l_2)$ if and only if $l_1 = l_2$ for all $l_1, l_2 \in X_c$. In the following, we present an alternative approach for reducing this conservatism.

Similar to the approach for the characterization of transient parameters from Sect. 8.3, here we characterize *stutter* steps as a specific class of spurious trajectories, which we introduce through Example 9.1 and Fig. 9.1.

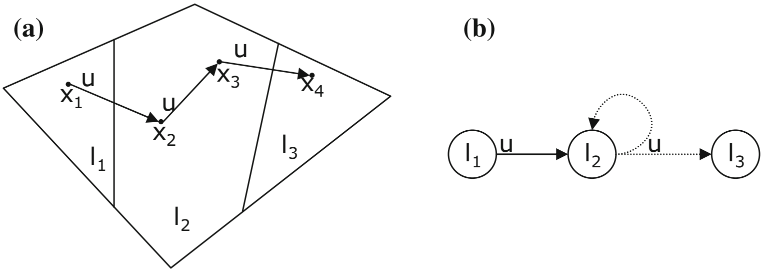


Fig. 9.1 A trajectory remaining forever in state l_2 exists in the finite abstraction (**b**), although such a behavior is not necessarily possible in the concrete system (**a**)

Example 9.1 Assume that a constant input $uuu\ldots$ produces a trajectory $x_1x_2x_3x_4\ldots$ in $T_{\mathscr{W}}$ where $o(x_1) = l_1, o(x_2) = o(x_3) = l_2, o(x_4) = l_3$ (Fig. 9.1a). The corresponding word $l_1l_2l_2l_3\ldots$ is a trajectory of T_c (i.e., $l_1, l_2, l_3 \in X_c$) and from the construction described in Sect. 9.1 it follows that $l_2 \in \delta_c(l_1, u)$ and $\{l_2, l_3\} \subseteq \delta_c(l_2, u)$ (Fig. 9.1b). Then, there exists a trajectory of T_c that remains infinitely in state $l_2 \in X_c$ under input u, which is not necessarily true for $T_{\mathscr{W}}$. Such spurious trajectories do not affect the correctness of a control strategy but increase the overall conservativeness of the method. We address this by characterizing *stuttering inputs*, which guarantee that the system will leave a state eventually, rather than in a single step, and using this additional information during the construction of the control strategy for T_c.

Definition 9.2 (*Stuttering inputs*) Given a state $l \in X_c$ and a set of states $C \in 2^{X_c}$, the set of inputs U_l^C is *stuttering* if and only if $l \in C$ and for all input words $u(0)u(1)\ldots$, where $u(i) \in U_l^C$, there exists a finite $k > 1$ such that the trajectory $x(0)x(1)\ldots$ produced in $T_{\mathscr{W}}$ by the input word satisfies $o(x(i)) = l$ for $i = 0, \ldots, k-1$ and $o(x(k)) = l' \in C, l' \neq l$.

Using Definition 9.2 we identify a stuttering subset $\Sigma_c^{ls} \subseteq \Sigma_c^l$ of the inputs available at a state $l \in X_c$. Let $u = c(U_l^C) \in \Sigma_c^l$ for some $C \in 2^{X_c}$ be an input of T_c computed as described in Sect. 9.1. Then $u \in \Sigma_c^{ls}$ if and only if U_l^C is stuttering. Note that a transition $\delta_c(l, u) = C$ from a state $l \in X_c$ where u is stuttering is always nondeterministic (i.e., $|C| > 1$) and contains a self loop (*i.e.*, $l \in C$) but the self loop cannot be taken infinitely in a row (i.e., a trajectory of $T_{\mathscr{W}}$ cannot remain infinitely in region $\mathbf{X}_l$ under input word $uuu\ldots$). An input $u \in \Sigma_c^{lu} = \Sigma_c^l \setminus \Sigma_c^{ls}$ induces a transition $\delta_c(l, u) = C$ where: (1) when $C = \{l\}$ trajectories of T_c and $T_{\mathscr{W}}$ produced by input word $uuu\ldots$ remain infinitely in state l and region $\mathbf{X}_l$, respectively, (2) when $l \notin C$ trajectories of T_c and $T_{\mathscr{W}}$ leave state l and region $\mathbf{X}_l$, respectively in one step under input u, (3) when $\{l\} \subset C$ trajectories of $T_{\mathscr{W}}$ produced by input word $uuu\ldots$ can potentially remain in region $\mathbf{X}_l$ infinitely. Although in case (3) it is also possible that trajectories of $T_{\mathscr{W}}$ produced by input word $uuu\ldots$ leave region $\mathbf{X}_l$ in finite time, we have to be conservative in order to guarantee the correctness of the control strategy.

Note that our definition of stuttering in Definition 9.2 requires that T_c leaves a state after a finite number of transitions are taken under the same stuttering input and therefore an infinite stutter cycle is never possible. Second, we identify a set of stuttering inputs rather than constructing T_c as a time abstract system. While we only characterize spurious infinite self loops (i.e., cycles of length 1), in general, it is possible that cycles of arbitrary length are spurious in T_c. Considering higher order cycles is computationally challenging and decreases the conservativeness of the approach only for very specific cases, while spurious self loops are commonly produced during the construction of T_c and can be identified or constructed through polyhedral operations as described in the following (Propositions 9.5 and 9.6).

Proposition 9.5 *Given a state $l \in X_c$ and a set of states $C \in 2^{X_c}$, input region U_l^C is stuttering if and only if $l \in C$ and $\mathbf{0} \notin hull\{(A_l - I)v_x + B_l v_u + c_l \mid \forall v_x \in V(\boldsymbol{X}_l), \forall v_u \in V(U_l^C)\}$, where hull denotes the convex hull, $V(.)$ is the set of vertices and I is the identity matrix.*

Proof ($\Rightarrow$) Let $\mathbf{0} \in \text{hull}\{(A_l - I)v_x + B_l v_u + c_l \mid \forall v_x \in V(\mathbf{X}_l), \forall v_u \in V(U_l^C)\}$. Then, there exists $x \in \mathbf{X}_l$, $u \in U_l^C$ such that $A_l x + B_l u + c_l = x$ and a trajectory of the system produced by applying input sequence $uuu\ldots$ and starting at x remains forever inside $\mathbf{X}_l$. Therefore, from Definition 9.2, U_l^C is not stuttering.

($\Leftarrow$) Let $\mathbf{0} \notin \text{hull}\{(A_l - I)v_x + B_l v_u + c_l \mid \forall v_x \in V(\mathbf{X}_l), \forall v_u \in V(U_l^C)\}$. From the separating hyperplane theorem it follows that there exists $a \in \mathbb{R}^N$ such that, for all $z \in \text{hull}\{(A_l - I)v_x + B_l v_u + c_l \mid \forall v_x \in V(\mathbf{X}_l)), a^T z > 0$. Then, any trajectory of the system originating in $\mathbf{X}_l$ and produced by input word $u_1 u_2 u_3\ldots$, where $u_i \in U_l^C$ will have a positive displacement along the direction of a^T at every step. Since $\mathbf{X}_l$ is bounded, all trajectories will leave it in a finite number of steps and, therefore, U_l^C is stuttering. ■

The strategy from Proposition 9.5 provides a computational characterization of stuttering input regions. In general, however, it is possible that an input region U_l^C cannot be identified as stuttering but a stuttering subset $\hat{U}_l^C \subset U_l^C$ can be identified. Then, if such a subset is "large enough" (i.e., $r(\hat{U}_l^C) > \varepsilon$) it can be used in T_c and allow more general control strategies. In Proposition 9.6 we describe the computation of such stuttering subsets.

Proposition 9.6 *Given an arbitrary $a \in \mathbb{R}^N$, the input region $\hat{U}_l^C = \{u \in U_l^C \mid \forall v \in V(\boldsymbol{X}_l), a^T B_l u > -a^T(A_l - I_N)v - c_l\}$, where $l \in C$ is always stuttering.*

Proof

$$\begin{aligned}\hat{U}_l^C &= \{u \in U_l^C \mid \forall v_x \in V(\mathbf{X}_l), a^T B_l u > -a^T(A_l - I)v_x - c_l\} = \\ &= \{u \in U_l^C \mid \forall v_x \in V(\mathbf{X}_l), a^T((A_l - I)v_x + B_l u + c_l) > 0\} \Rightarrow \\ &\Rightarrow \mathbf{0} \notin \text{hull}\{(A_l - I)v_x + B_l v_u + c_l \mid \forall v_x \in V(\mathbf{X}_l), \forall v_u \in V(\hat{U}_l^C)\},\end{aligned}$$

which, from Proposition 9.5, guarantees that $\hat{U}_l^C$ is stuttering. ■

Although Proposition 9.6 is valid for an arbitrary $a \in \mathbb{R}^N$, the volume of the stuttering subset $\hat{U}_l^C \subset U_l^C$ depends on a. Since only "large enough" input regions are considered in T_c (see Algorithm 17), a should be chosen in such a way that the radius $r(\hat{U}_l^C)$ is maximized.

The control algorithms we discussed in Sect. 5.1 can be adapted to handle the additional information about stuttering inputs captured in T_c, while the correctness and completeness of the control strategy computation for the product automaton P is still guaranteed. P is constructed as in Sect. 5.1 and therefore it naturally inherits the partitioned input set $\Sigma_c^l = \Sigma_c^{ls} \cup \Sigma_c^{lu}$ for each state $l \in X_c$. Going back to the Rabin game interpretation of the control problem discussed in Sect. 5.1, we need to account for the fact that the adversary cannot take transitions under the same stuttering input infinitely many times in a row. As a result, the construction of the control strategy is still performed using the same algorithm and only the computations of the direct attractors from Definitions 5.7 and 5.8 are modified as follows (the notation used in the rest of this section was introduced in Chap. 5).

Let $l \in X_c$ and $u \in \Sigma_c^{ls}$ be a state and a stuttering input of T_c (Definition 9.2). We are interested in *edge* (s, u, s') of transition $\delta_P(s, u) = S'$, where $\alpha(s) = l$ and $s' \in S'$ (here $\alpha()$ is the projection of states from P to states of T_c—see Sect. 4.5). Edge (s, u, s') is called *u-nontransient* edge if $\alpha(s) = \alpha(s') = l$ and *transient* otherwise. Note that, even though (l, u, l) is a self loop in T_c, (s, u, s') is not necessarily a self loop in P. In addition, since there is at most one self loop at a state $l \in X_c$ and R is deterministic, there is at most one u-nontransient edge leaving state s.

We refer to a sequence of edges $(s_1, u_1, s_2)(s_2, u_2, s_3) \dots (s_{n-1}, u_{n-1}, s_n)$, where $s_i \neq s_j$ for any $i, j \in \{1, \dots, n\}$ as a *simple path*, and to a simple path $(s_1, u_1, s_2) \dots (s_{n-1}, u_{n-1}, s_n)$ followed by (s_n, u_n, s_1) as a *cycle*. We can observe that any sequence of u-nontransient edges (i.e., a run of the product automaton, or its finite fragment) is of one of the following shapes: a cycle (called a *u-nontransient cycle*), a lasso shape (a simple path leading to a u-nontransient cycle), or a simple path ending at a state where the input u is not available at all. Informally, the existence of a stuttering self loop in a state l under input u in T_c means that this self loop cannot be followed infinitely many times in a row. Similarly, any u-nontransient cycle in the product graph cannot be followed infinitely many times in a row without leaving it. This leads us to the new definitions of protagonist's and adversary's direct attractor.

Definition 9.3 (*Modified protagonist's direct attractor*) The protagonist's direct attractor of S', denoted by $\mathsf{A}_P^1(S')$, is the set of all states $s \in S_P$, such that there exists an input u satisfying

(1) $\delta_P(s, u) \subseteq S'$, or
(2) s lies on a u-nontransient cycle, such that each state s' of the cycle satisfies that $s'' \in S'$ for all transient edges (s', u, s'').

In other words, the protagonist can enforce a visit to S' also by following a u-nontransient cycle finitely many times and eventually leaving it to S'.

Definition 9.4 (*Modified adversary's direct attractor*) The adversary's direct attractor of S', denoted by $\mathsf{A}^1_S(S')$, is the set of all states $s \in S_P$, such that for each input u there exists a state s' such that

(1) $s' \in \delta_P(s, u) \cap S'$, and
(2) s' does not lie on a u-nontransient cycle.

In other words, the adversary cannot enforce a visit to S' via an edge of a u-nontransient cycle. This edge can be taken only finitely many times in a row and eventually a different edge under input u has to be chosen.

By identifying stuttering inputs during the construction of the control transition system T_c (Propositions 9.5 and 9.6) and modifying the approach from Sect. 5.1 to handle this additional information during the construction of a control strategy for T_c (Definitions 9.3 and 9.4), we can reduce the conservatism associated with the overall method. Even so, our solution to Problem 9.1 remains conservative but it is important to note that the only source of conservativeness is the construction of T_c—the solution to the LTL control problem for T_c is complete.

Example 9.2 We consider the problem of controlling the two-tank system shown in Fig. 9.2. The system has two state variables ($N = 2$) that represent the water levels in the two tanks and range in $(0, 0.7)$. It has one control dimension ($M = 1$) representing the inflow rate, which ranges in $(0, 5e^{-4})$. The state space of the system is partitioned into 49 rectangular regions (i.e., $L = 1, \ldots, 49$) by 7 evenly spaced thresholds along each dimension. These thresholds signify that we can only detect whether the water level in each tank is above or below the marks at $0.1, 0.2, \ldots, 0.7$ (see Fig. 9.3a). Valve v_1 is opened only if submerged (i.e., if the water level in either tank is above 0.2—the height of the valve) and, therefore, the valve is closed when the system is in regions 1, 2, 8, and 9 and opened otherwise (see Figs. 9.2 and 9.3a). Discrete-time, linear equations describing the dynamics of the system in each mode are derived using a 5 sec. time step and $1.54e^{-2}\, m^2$, $1e^{-4}\, m^2$, and $2.125e^{-5}\, m^2$ as the cross sectional areas of the two tanks, valve v_1, and valve v_2, respectively. The dynamics of the system for each region $l \in L$ are given by

$$A_l = \begin{cases} \begin{bmatrix} 1 & 0 \\ 0 & 0.9635 \end{bmatrix} & \text{for } l = 1, 2, 8, 9 \\ \begin{bmatrix} 0.8281 & 0.1719 \\ 0.1719 & 0.7916 \end{bmatrix} & \text{otherwise,} \end{cases}$$

$$B_l = [324.6753, 0]^T, c_l = [0, 0]^T \text{ for } l = 1, \ldots, 49.$$

We seek a control strategy for the system guaranteeing the satisfaction of a specification, expressed informally as "whenever tank 2 is empty, it will eventually get filled up". To formalize this specification we define the subformulas ϕ_1 = "the level of tank 2 is below 0.1" (i.e., "tank 2 is empty") and ϕ_2 = "the level of tank 2 is above 0.4" (i.e., "tank 2 is full"), which can be expressed as disjunctions of regions from L as $\phi_1 = 1 \vee \ldots \vee 7$ and $\phi_2 = 29 \vee \ldots \vee 49$. The above specification translates to the following LTL formula:

$$\phi = \Box(\phi_1 \Rightarrow \Diamond\phi_2).$$

Satisfying control strategies were found from all regions expect 49 when the required robustness was set to $\varepsilon = 5e^{-6}$ (Fig. 9.3a). If stuttering inputs are not characterized as described in Sect. 9.3, satisfying control strategies are identified for regions 29, ..., 48 only. A simulated trajectory of the closed loop system is shown in Fig. 9.3b.

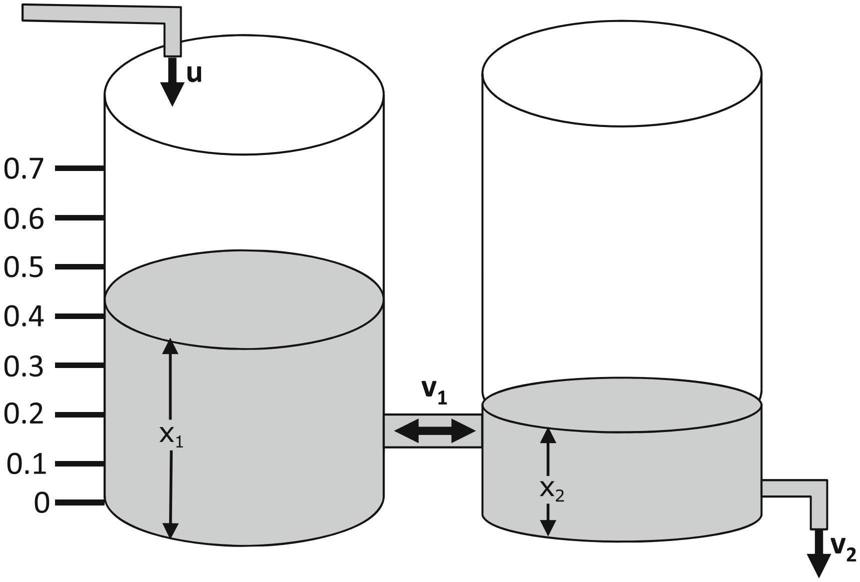
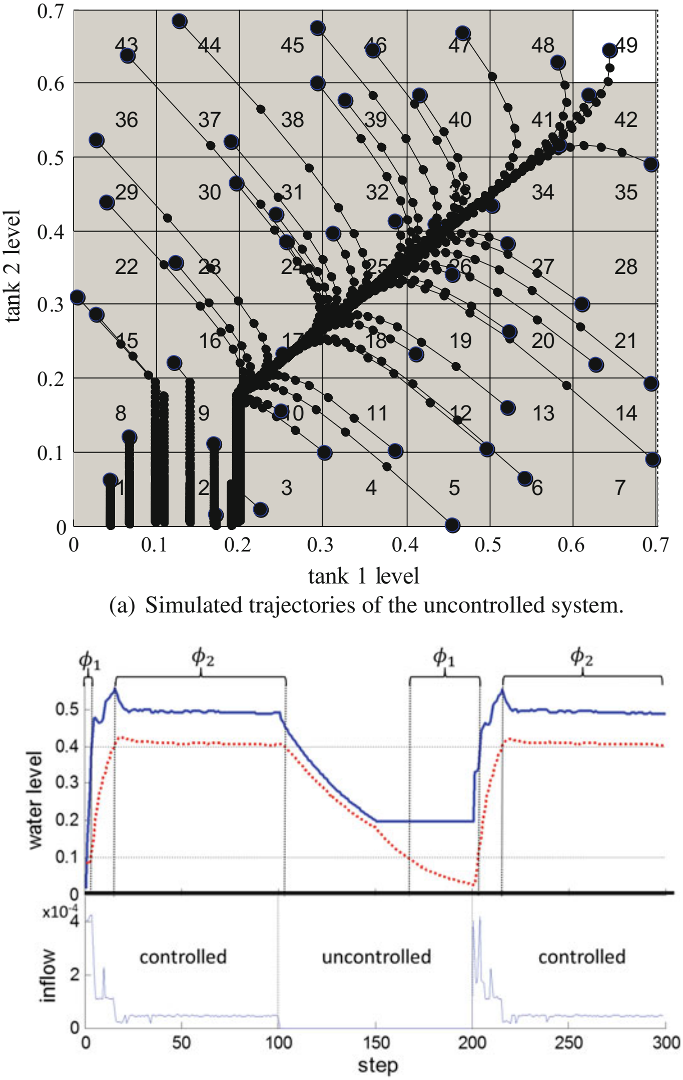


Fig. 9.2 A two-tank system. Water is drained at a constant rate from tank 2 though valve v_2, while tank 1 is filled at a rate that is controlled externally. Water can also flow in either direction, from the tank with more water to the one with less, through valve v_1, which is opened only if submerged



(a) Simulated trajectories of the uncontrolled system.

(b) A simulated trajectory of the closed-loop system.

Fig. 9.3 Simulated trajectories of the uncontrolled and closed-loop water tank system from Fig. 9.2. Initial conditions are shown as *blue circles* and regions are labeled only by their indexes in (**a**). Control strategies guaranteeing the satisfaction of specification $\phi = \Box(\phi_1 \Rightarrow \Diamond\phi_2)$ are found from all shaded regions in (**a**). The water levels of tanks 1 and 2 are shown respectively as a *blue (solid)* and a *red (dashed) line* in (**b**) and the trajectory is guaranteed to satisfy specification ϕ. See Example 9.2 for additional details

Example 9.3 We apply the method from this chapter to generate control strategies for a simple PWA system with two state variables ($N = 2$) ranging in $(0, 100)$. The state space is partitioned into 36 rectangular regions (i.e., $L = 1, \dots, 36$). The system has two control inputs ($M = 2$) ranging in $(-15, 15) \times (-16, 16)$. The dynamics of the system are defined as follows:

$$A_{1\ldots4,9\ldots12,25\ldots28,33\ldots36} = \begin{bmatrix} 0.9900 & 0.0 \\ 0.0 & 0.9800 \end{bmatrix} \quad A_{5\ldots8,29\ldots32} = \begin{bmatrix} 0.9900 & 0.0 \\ -0.0200 & 0.9800 \end{bmatrix}$$

$$A_{13\ldots16,21\ldots24} = \begin{bmatrix} 0.9900 & -0.0300 \\ 0.0 & 0.9800 \end{bmatrix} \quad A_{17\ldots20} = \begin{bmatrix} 0.9900 & -0.0300 \\ -0.0200 & 0.9800 \end{bmatrix}$$

$$B_{1\ldots36} = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

$$c_{1\ldots4} = \begin{bmatrix} 0.9200 \\ 1.5300 \end{bmatrix} \quad c_{5\ldots8} = \begin{bmatrix} 1.3900 \\ 1.5300 \end{bmatrix} \quad c_{9\ldots12} = \begin{bmatrix} 0.1700 \\ 1.5300 \end{bmatrix}$$

$$c_{13\ldots16} = \begin{bmatrix} 0.9200 \\ 2.9000 \end{bmatrix} \quad c_{17\ldots20} = \begin{bmatrix} 1.3800 \\ 2.9000 \end{bmatrix} \quad c_{21\ldots24} = \begin{bmatrix} 0.1700 \\ 2.9200 \end{bmatrix}$$

$$c_{25\ldots28} = \begin{bmatrix} 0.9200 \\ 0.1500 \end{bmatrix} \quad c_{29\ldots32} = \begin{bmatrix} 1.4100 \\ 0.1500 \end{bmatrix} \quad c_{33\ldots36} = \begin{bmatrix} 0.1700 \\ 0.1500 \end{bmatrix}$$

The PWA model captures the characteristic bistability of the system, where trajectories of the uncontrolled system go towards one of two possible stable equilibria located in regions $\mathbf{X}_{10}$ and $\mathbf{X}_{27}$ (see Fig. 9.4).

We seek a control strategy that drives the system to low values of state variable 1 and high values of state variable 2, while avoiding intermediate values of both state variables. We define sub-formulas ϕ_1 = "state variable 1 is below 20 and state variable 2 is above 75" and ϕ_2 = "state variable 1 is above 40 and below 80 and state variable 2 is above 20 and below 50" which can be expressed as disjunctions of regions from L as $\phi_1 = 10$ and $\phi_2 := 17 \vee \ldots \vee 20$. The control specification translates to the following LTL formula:

$$\phi = \Diamond \Box \phi_1 \wedge \Box \neg \phi_2,$$

which cannot be translated into a deterministic Büchi automaton (see Chap. 2).

A control transition system T_c with 36 states was constructed. Out of the total 949 nonempty input regions found (denoted by U_l^C in Sect. 9.1), 684 were "large enough" (the radii of their inscribed spheres were larger than $\varepsilon = 5e^{-2}$) to be considered for a robust control strategy and included in T_c. After reducing the size of T_c (i.e., removing unnecessary nondeterministic transitions as explained in Remark 9.1) only 222 input regions were included out of which 42 were deterministic and 109 were identified as stuttering. The specification ϕ translates into a deterministic Rabin automaton with 3 states and 1 pair in its acceptance condition.

Satisfying control strategies with the required robustness ($\varepsilon = 5e^{-2}$) were found from all regions expect for $13, \ldots, 20$ (shown in light gray in Fig. 9.5a). For this particular problem, the target region $\mathbf{X}_{10}$ of specification ϕ is reachable only through transitions under stuttering inputs in T_c. Therefore, a satisfying control strategy can be identified only from region $\mathbf{X}_{10}$, unless stuttering behavior is considered. The additional computation described in Sect. 9.3 allowed us to expand the satisfying initial set from $\mathbf{X}_{10}$ only to the entire region highlighted in Fig. 9.5.

Starting from random initial conditions, trajectories of the closed loop system were simulated (Fig. 9.5), where at each step applied inputs were corrupted by noise bounded by ε. All simulated trajectories avoid the unsafe regions $\mathbf{X}_{17} \ldots \mathbf{X}_{20}$ and satisfy the specification, thereby demonstrating the correctness and robustness of the control strategy.

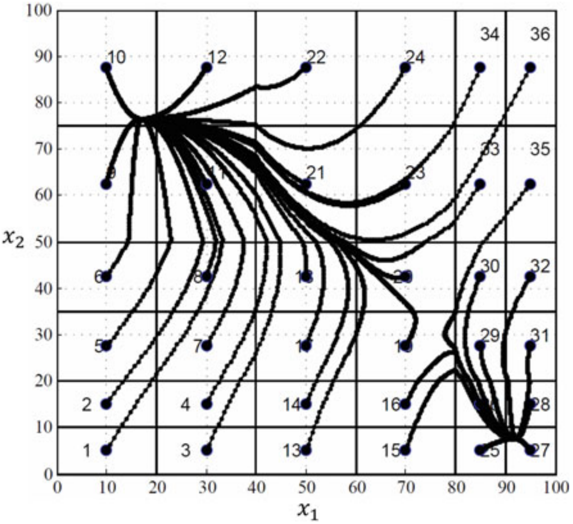
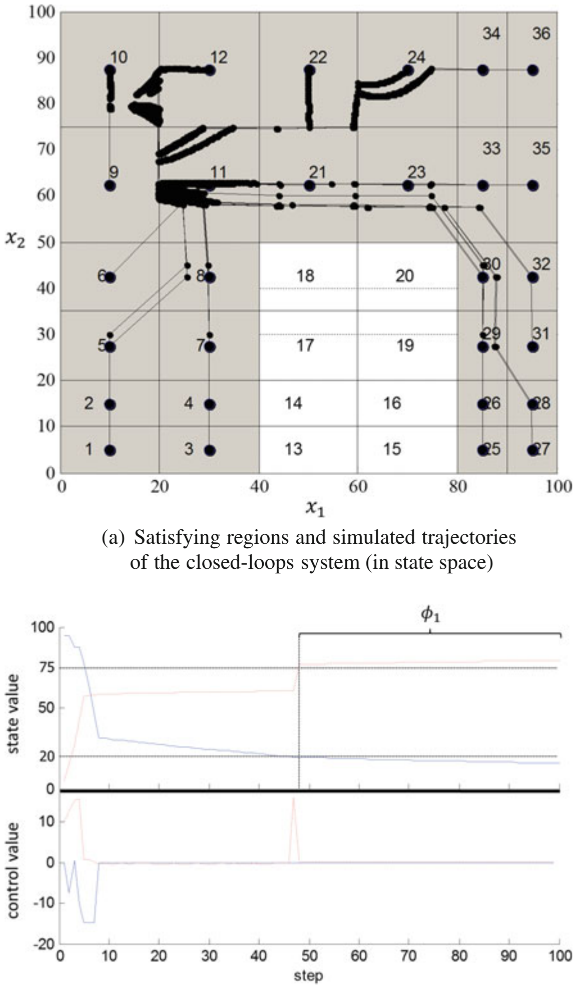


Fig. 9.4 Trajectories of the uncontrolled PWA system go towards one of two possible stable equilibria located in regions $\mathbf{X}_{10}$ and $\mathbf{X}_{27}$ (initial states are shown as *small circles* and regions are labeled only by their indexes). See Example 9.3 for additional details

Fig. 9.5 Control strategies guaranteeing the satisfaction of specification $\phi = \Diamond\Box\phi_1 \wedge \Box\neg\phi_2$ are found from all shaded regions (regions are labeled only by their indexes). Simulated trajectories from different regions satisfy the specification. A simulated trajectory of the closed loop system is guaranteed to satisfy specification ϕ—the values of state variables 1 and 2 and the respective external control values are shown as a *blue (solid)* and a *red (dashed) line*. See Example 9.3 for additional details



(a) Satisfying regions and simulated trajectories of the closed-loops system (in state space)

(b) Simulated trajectory and control values of the closed-loop system (over time)

Example 9.4 We seek a control strategy for the PWA system defined in Example 9.3 that forces the system to oscillate between states where the values of one of the state variables is high and the other is low and vice versa, while states where the values of both state variables are high or low are never reached. We define sub-formulas ϕ_1 = "state variable 1 is below 40 and state variable 2 is above 50", ϕ_2 := "state variable 1 is above 80 and state variable 2 is below 20", ϕ_3 := "state variable 1 is below 40 and state variable 2 is below 20", and ϕ_4 := "state variable 1 is above 80 and state variable 2 is above 50", which can be expressed as disjunctions of regions from L as $\phi_1 = 9 \vee \ldots \vee 12$, $\phi_2 = 25 \vee \ldots \vee 28$, $\phi_3 = 1 \vee \ldots \vee 4$, and $\phi_4 = 33 \vee \ldots \vee 36$. The above specification translated to the following LTL formula:

$$\phi = \Box(\Diamond\phi_1 \wedge \Diamond\phi_2 \wedge \neg(\phi_3 \vee \phi_4)).$$

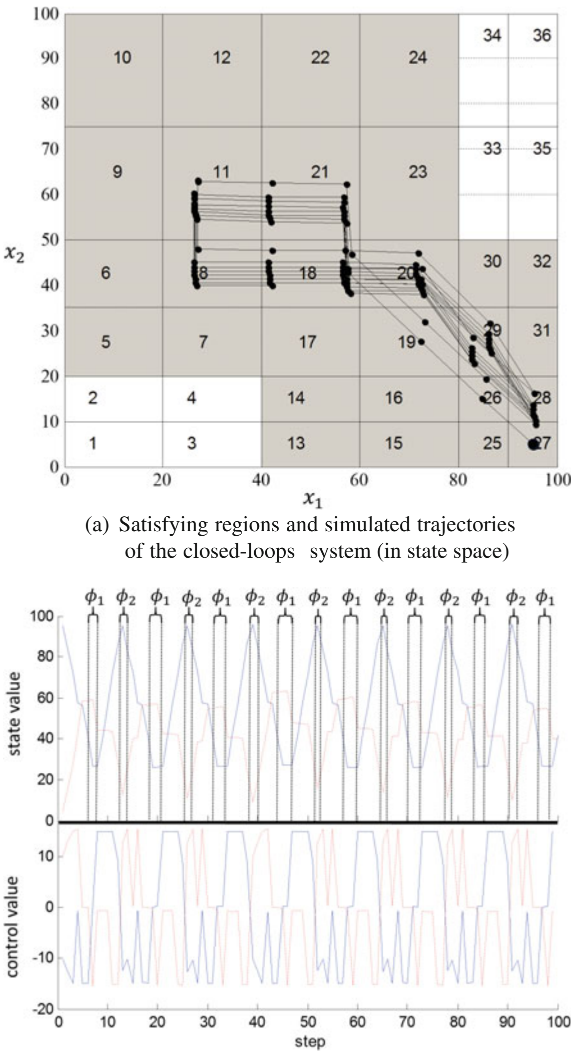
Satisfying control strategies where found from all regions expect $1, \ldots, 4$ and $33, \ldots, 36$ when the required robustness was set to $\varepsilon = 5e^{-2}$ (Fig. 9.6a). If stuttering inputs are not characterized as described in Sect. 9.3, no satisfying control strategies are identified. A simulated trajectory of the closed loop system is shown in Fig. 9.6b.

9.4 Notes

The material presented in this chapter is based on [170, 179, 184]. Related approaches can be found in [138, 164], where the existence of equivalent (bisimulation) quotients and control strategies under the assumption of controllability for discrete-time linear systems is characterized. The monotonicity property of a discrete-time piecewise system is exploited in [46] to develop an efficient abstraction algorithm. Algorithmic procedures for controlling continuous-time linear systems are given in [104, 105], where the constructed deterministic (nondeterministic) abstractions are not equivalent to the original system but instead capture only a restricted but controllable subset of its behavior. An alternative abstraction technique based on quantifier elimination for real closed fields and theorem proving has been proposed in [168].

To solve Rabin games, we implemented an algorithm from [90] (see details in Sect. 5.1) but extended it to deal with stuttering behavior, which leads to additional winning strategies. The concept of stuttering (see Sect. 9.3) has been established previously [15] and related work has focused on determining if a specification is closed under stuttering [137], in which case it can be expressed in the LTL$\backslash\bigcirc$ fragment (LTL without the next operator) [139] and less conservative stutter bisimulation quotients can be constructed [15]. The approach from this chapter does not require any special structure from either the specification or the quotient. Instead, we charac-

Fig. 9.6 Control strategies guaranteeing the satisfaction of specification $\phi = \square(\Diamond\phi_1 \wedge \Diamond\phi_2 \wedge \neg(\phi_3 \vee \phi_4))$ are found from all shaded regions (regions are labeled only by their indexes). A sample satisfying simulated trajectory is shown. A simulated trajectory of the closed loop toggle switch system is guaranteed to satisfy specification ϕ—state and control variables are labeled as in Fig. 9.5b. See Example 9.4 for additional details



(a) Satisfying regions and simulated trajectories of the closed-loops system (in state space)

(b) Simulated trajectory and control values of the closed-loop system (over time)

terize individual transitions as stuttering while constructing the abstraction and use this additional information during the solution of the Rabin game, which reduces the conservatism of the overall method but does not restrict the expressivity of the specification. Another related approach is based on characterizing the sets of transient states (instead of an individual transition) and augmenting the transition relation with the corresponding transitions, which was applied to switched systems in [136]. While considering transient sets reduces conservatism, it is computationally expensive to characterize such sets.

As our proposed solution to Problem 9.1 consists of (1) the construction of the control transition system T_c and (2) the generation of a control strategy for T_c, the overall computational complexity is the cumulative complexity of the two parts. The computation of T_c involves enumerating all subsets of L at any element of L, which gives $\mathscr{O}(|L| \cdot 2^{|L|})$ iterations in the worst case, although in practice this can be reduced (see Eq. (9.16)). At each iteration, polyhedral operations are performed, which scale exponentially with N, the size of the continuous state space. The characterization of stuttering inputs described in this chapter checks each element from Σ_c through polyhedral operations.

The overall complexity of the control strategy synthesis for T_c is $\mathscr{O}(k!n^k)$, where n is the size of the product automaton and k is the number of pairs in the Rabin condition of the product automaton (see Chap. 5). The modifications in the computation of the direct attractor we made in order to adapt the algorithm to deal with stuttering behavior do not change the overall complexity. Note that, in general, Rabin games are NP-complete, so the exponential complexity with respect to k is not surprising. However, LTL formulas are usually translated into Rabin automata with very few tuples in their acceptance condition.

The method described in this chapter was implemented in MATLAB as the software package CONPAS2, where all polyhedral operations were performed using the MPT toolbox [113]. The tool takes as input a PWA system (Definition 6.2) and an LTL formula and produces a set of satisfying initial regions and a feedback control strategy for the system (Definition 9.1). The tool is freely downloadable from our web site at http://sites.bu.edu/hyness/conpas2/. As an alternative to CONPAS2, the MPT TOOLBOX [113] and the HYBRID TOOLBOX [27] for MATLAB can also be used to design piecewise affine control laws but neither can handle the richness of LTL specifications directly. The problem of controlling Mixed Logical Dynamical (MLD) systems from LTL specifications has been considered in [100] and is related to this problem, due to the equivalence between MLD and PWA systems [82]. Rather than relying on the construction of finite quotients as in this chapter, the approach taken in [100] involves representing LTL specifications as mixed-integer linear constraints but a finite horizon assumption is imposed.

Other temporal logic control tools include PESSOA [128], LTLMOP [61], TULIP [174] (http://tulip-control.sourceforge.net), LTLCON [105], and its extension BÜCON [104]. PESSOA is capable of generating control strategies for linear, nonlinear, and switched systems from temporal logic specifications. Abstractions based on the notions of approximate simulation and bisimulation are constructed. LTLMOP uses a fragment of LTL called GR(1) [109, 142], which accounts for the adversarial nature of the environment. Similar to LTLMOP, TULIP models the environment as an adversary and uses a receding horizon framework to alleviate the computational complexity of synthesis. Similar to CONPAS2, TULIP can handle discrete-time affine control systems. LTLCON addresses the control problem for linear systems but only deterministic abstractions are allowed, which leads to very conservative results. Its extension, BüCon, relaxes this but restricts the specification language to the fragment of LTL generated by deterministic Büchi automata (see Sect. 5.2). In contrast, conPAS2 constructs nondeterministic abstractions of piecewise affine systems and generates control strategies from full LTL specifications, while conservatism is further reduced by identifying stuttering behavior.

Chapter 10
Finite Bisimulations

In this chapter, we focus on verification and control of switched linear systems (Definition 6.8). We show that a finite bisimulation quotient (see Sect. 1.3) of a stable switched linear system exists and can be constructed by performing a finite number of basic polyhedral operations. To construct the quotient system, we use a polyhedral Lyapunov function. The existence of such a function is a necessary condition for the stability of a switched linear system. Once a bisimulation quotient is constructed, the verification and synthesis problems can be solved by using techniques presented in Chaps. 4 and 5.

Throughout this chapter, we assume that, in Definition 6.8, all matrices $A_\gamma \in \mathbb{R}^{N\times N}$, $\gamma \in \varGamma$, are strictly stable (i.e., Schur), for all $\gamma \in \varGamma$. We assume also that the system is asymptotically stable in $\mathbf{X}$ under arbitrary switching. Specifically, we assume that a common infinity norm Lyapunov function (LF) $V : \mathbb{R}^N \to \mathbb{R}^+$ of the form (A.13) with contraction rate $\rho \in (0, 1)$ is known for system 6.4 and $\mathbf{X}$ is a sublevel set of V. Furthermore, we assume that, among the regions of interest $\mathbf{X}_l$, $l \in L$, there exist one (denoted by $\mathbf{X}_{l_o}$, $l_o \in L$) that is a sublevel set of V.

Remark 10.1 The asymptotic stability assumption implies existence of a common infinity norm Lyapunov function. An overview of Lyapunov stability and polyhedral Lyapunov functions is presented in Appendix A.5. The sublevel set assumption on $\mathbf{X}$ is made to simplify the presentation of technical results in this chapter. As discussed later, this assumption is not necessary.

Example 10.1 Consider a switched linear system $\mathscr{S}$ (Definition 6.8) with two subsystems:

$$A_1 = \begin{bmatrix} -0.65 & 0.32 \\ -0.42 & -0.92 \end{bmatrix},\ A_2 = \begin{bmatrix} 0.65 & 0.32 \\ -0.42 & -0.92 \end{bmatrix},\ \varGamma = \{1, 2\}. \tag{10.1}$$

The system is globally asymptotically stable, and $V(x) = \|\mathrm{L}x\|_\infty$ is an infinity norm Lyapunov function with contraction rate $\rho = 0.94$, where

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_10

$$\mathrm{L} = \begin{bmatrix} -0.0625 & 0.6815 & 0.9947 & 0.9947 \\ 1 & 1 & 0.6868 & -0.0678 \end{bmatrix}^\top . \tag{10.2}$$

We define $\mathbf{X}$ and $\mathbf{X}_0$ as sublevel sets of V with bounds 10 and 5.063, i.e.,

$$\mathbf{X} = \{x \in \mathbb{R}^n \mid V(x) \leq 10\}, \text{ and}$$

$$\mathbf{X}_0 = \{x \in \mathbb{R}^n \mid V(x) \leq 5.063\}.$$

The regions of interests $\{\mathbf{X}_l\}_{l \in L}$, $L = \{0, \ldots, 4\}$ are shown in Fig. 10.1. In particular, $\mathbf{X}_0$ is a sublevel set, $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are non-intersecting polytopes included in $\mathbf{X} \backslash \mathbf{X}_0$, and $\mathbf{X}_4$ is the remaining region, i.e.

$$\mathbf{X}_4 = \mathbf{X} \setminus \bigcup_{l=0,\ldots,3} \mathbf{X}_l.$$

Note that each $\mathbf{X}_l$ is a semi-linear set.

We consider both temporal logic control and verification problems for system $\mathscr{S}$. In the control problem, we assume that we can choose the dynamics A_γ, $\gamma \in \Gamma$ to be applied at each step k. Our goal is to find the largest set of initial states and a control strategy such that all the corresponding trajectories of $\mathscr{S}$ satisfy a temporal logic specification. In the verification problem, we assume that we do not have control over the switches, i.e., at every time step a subsystem (mode) is arbitrarily chosen

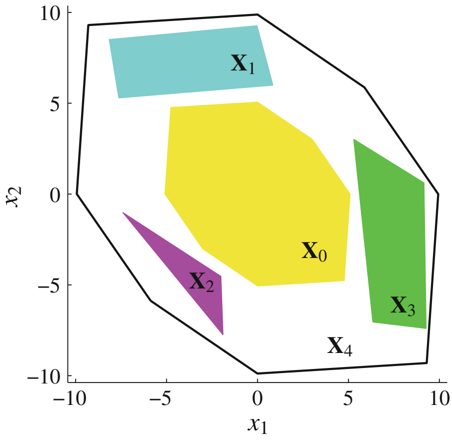


Fig. 10.1 The state space $\mathbf{X}$ and the regions of interests of the switched system defined in Example 10.1. The boundary of $\mathbf{X}$ is shown with *black lines*. The sublevel set $\mathbf{X}_0$ is shown in *yellow*, $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are shown in *cyan*, *magenta* and *green*, respectively. $\mathbf{X}_4$ is shown in *white*

from the set Γ. A switched system under arbitrary switching is autonomous and nondeterministic. Our goal is to find the largest set of initial states such that all trajectories under all possible switching scenarios satisfy a temporal logic specification.

The asymptotic stability assumption implies that all trajectories of $\mathscr{S}$ eventually sink in $\mathbf{X}_{l_o}$, i.e. reach $\mathbf{X}_{l_o}$ and stay in there. For this reason, we will focus on the syntactically co-safe fragment of LTL, which includes all specifications of LTL where satisfactions of trajectories can be determined by a finite prefix (see Definition 2.3). Since we are interested in the behavior of $\mathscr{S}$ until $\mathbf{X}_{l_o}$ is reached, scLTL is a sufficiently rich specification language.

The above control and verification problems can be formally stated by using embedding $T_{\mathscr{S}} = (X_{\mathscr{S}}, \Sigma_{\mathscr{S}}, \delta_{\mathscr{S}}, O_{\mathscr{S}}, o_{\mathscr{S}})$ of $\mathscr{S}$ and its autonomous version $T^A_{\mathscr{S}} = (X_{\mathscr{S}}, \delta^A_{\mathscr{S}}, O_{\mathscr{S}}, o_{\mathscr{S}})$, respectively (see Definition 6.9 in Chap. 6).

Problem 10.1 (*Largest controlled satisfying region for* $\mathscr{S}$) Given an embedding $T_{\mathscr{S}}$ of a switched linear system $\mathscr{S}$ and an scLTL formula ϕ over L, find a control strategy $(X^{\phi}_{T_{\mathscr{S}}}, \Omega)$ (Definition 5.1) such that $X^{\phi}_{T_{\mathscr{S}}}$ is the largest controlled satisfying region (Definition 5.2) of the embedding transition system $T_{\mathscr{S}}$.

The embedding transition system $T_{\mathscr{S}}$ has a finite set of controls (Γ from Eq. (6.4)). However, it has infinitely many states. Therefore, the methods developed in Chap. 5 on synthesis of control strategies for finite transition systems can not directly be applied to solve Problem 10.1. We develop a computational procedure to construct a finite bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ for the embedding transition system $T_{\mathscr{S}}$ of an asymptotically stable switched linear system. Since the bisimulation quotient preserves all properties that are expressible in LTL (and scLTL), and is finite, the methods developed in Chap. 5 can be used to find a control strategy $(X^{\phi}_{T_{\mathscr{S}}/_{\approx}}, \Omega/_{\approx})$ for $T_{\mathscr{S}}/_{\approx}$, which directly maps to a strategy $(X^{\phi}_{T_{\mathscr{S}}}, \Omega)$ for $T_{\mathscr{S}}$ solving Problem 10.1.

Problem 10.2 (*Largest satisfying region under arbitrary switching for* $\mathscr{S}$) Given an embedding $T^A_{\mathscr{S}}$ of a switched linear system $\mathscr{S}$ under arbitrary switching and an scLTL formula ϕ over L, find the largest set of states $X^{\phi}_{T^A_{\mathscr{S}}}$ from which ϕ is satisfied.

To solve the verification Problem 10.2, we construct the finite bisimulation quotient $T^A_{\mathscr{S}}/_{\approx}$ of $T^A_{\mathscr{S}}$ from the bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ of $T_{\mathscr{S}}$, then employ analysis techniques presented in Chap. 4.

Example 10.2 Consider the switched system defined in Example 10.1, and the specification “Never visit $\mathbf{X}_2$ and eventually visit $\mathbf{X}_1$. Moreover, if $\mathbf{X}_3$ is visited, then $\mathbf{X}_1$ must not be visited at the next time step.” This specification can be translated to the scLTL formula over $L = \{0, 1, 2, 3, 4\}$:

$$\phi := (\neg 2 \; \mathcal{U} \; 0) \wedge \Diamond 1 \wedge ((3 \Rightarrow \bigcirc \neg 1) \; \mathcal{U} \; 0) \tag{10.3}$$

Note that all system trajectories reach $\mathbf{X}_0$ in finite time and sink in there since the system is globally asymptotically stable. Therefore, "never visit $\mathbf{X}_2$", represented as $\Box\neg 2$ in LTL, can be written as $\neg 2\ U\ 0$ in scLTL for the stable system.

10.1 Bisimulation Quotient

In this section, we present an algorithm to construct a bisimulation $\approx\subseteq X_{\mathscr{S}} \times X_{\mathscr{S}}$ of the embedding transition system $T_{\mathscr{S}} = (X_{\mathscr{S}}, \Gamma, \delta_{\mathscr{S}}, L, o_{\mathscr{S}})$ of system $\mathscr{S}$ (Eq. 6.4). The proposed algorithm is similar to the bisimulation algorithm (Algorithm 1) presented in Chap. 1 in the sense that it starts with the observational equivalence relation $\sim$, and the corresponding equivalence classes $X/_\sim$, then iteratively refines these classes. While in Algorithm 1, an equivalence class that violates the bisimulation property is refined at each iteration, here we exploit the fact that the system is asymptotically stable, and guide the refinement procedure in such a way that starting from a set $X_0 \subset X_{\mathscr{S}}$, with $\mathbf{0} \in int(X_0)$, we incrementally construct a finite bisimulation for a larger subset of $X_{\mathscr{S}}$ as the algorithm iterates.

10.1.1 Level Sets and Slices

The proposed abstraction algorithm builds upon existence of a sequence of sets $P_{\Theta_0}, \ldots, P_{\Theta_M}$ satisfying properties i–iv:

i. $\mathbf{0} \in int(P_{\Theta_0})$,
ii. $P_{\Theta_M} = X_{\mathscr{S}}$,
iii. $P_{\Theta_i} \subset P_{\Theta_{i+1}}$, for all $i = 0, \ldots, M-1$
iv. $Post_{T_{\mathscr{S}}}(P_{\Theta_i}, \Gamma) \subset P_{\Theta_{i-1}}$ for all $i = 1, \ldots, M$ and $Post_{T_{\mathscr{S}}}(P_{\Theta_0}, \Gamma) \subset P_{\Theta_0}$.

We define the sequence of sets from the Lyapunov function V and its contraction rate ρ. For any $\Theta > 0$,

$$P_{\Theta} = \{x \in \mathbb{R}^N \mid V(x) \leq \Theta\}$$

is a sublevel set of V. We define a sequence of real positive numbers $\overline{\Theta} := \Theta_0, \ldots, \Theta_M$ such that the induced sequence of sublevel sets $P_{\Theta_0}, \ldots, P_{\Theta_M}$ of V ($P_{\Theta_i} = \{x \in \mathbb{R}^N \mid V(x) \leq \Theta_i\}$) satisfies properties i-iv.

It is assumed that $X_{\mathscr{S}}$ ($= \mathbf{X}$) and $\mathbf{X}_{l_0}$, for some $l_0 \in L$, are sublevel sets of V. Let $\Theta_{\mathbf{X}}, \Theta_{l_0} > 0$, such that $X_{\mathscr{S}} = P_{\Theta_{\mathbf{X}}}$ and $\mathbf{X}_{l_0} = P_{\Theta_{l_0}}$. We define the sequence $\overline{\Theta} := \Theta_0, \ldots, \Theta_M$ as

$$\Theta_{i+1} = \rho^{-1}\Theta_i, \quad i = 0, \ldots, M-2, \tag{10.4}$$

$\Theta_0 := \Theta_{l_0}$, $\Theta_M := \Theta_{\mathbf{X}}$, and

$$M := \arg\min_M \{\rho^{-M}\Theta_0 \mid \rho^{-M}\Theta_0 \geq \Theta_{\mathbf{X}}.\} \tag{10.5}$$

Next, we define a *slice* of the state space $\mathbf{X}$ as the region between two consecutive sets from the induced sequence of sublevel sets $\overline{P} = P_{\Theta_0}, \ldots, P_{\Theta_M}$:

$$S_i = P_{\Theta_i} \setminus P_{\Theta_{i-1}}, \qquad i = 1, \ldots, M. \tag{10.6}$$

For convenience, we also denote $S_0 := P_{\Theta_0}$ (although S_0 is not a slice in between two level sets). We immediately see that the sets $\{S_i\}_{i=0,\ldots,M}$ form a partition of $X_{\mathscr{S}}$.

Remark 10.2 The slices are bounded semi-linear sets since the sublevel sets are polytopes. In particular, the slices are always bounded semi-linear sets if the sublevel sets are also bounded semi-linear sets.

Example 10.3 Consider the switched system given in Example 10.1. The sequence $\overline{\Theta}$ is computed from $\Theta_{\mathbf{X}}$, Θ_{l_0} and ρ as described above, which resulted in $M = 11$. The polytopic sublevel sets $\overline{P} := P_{\Theta_0}, \ldots, P_{\Theta_{11}}$ are shown in Fig. 10.2.

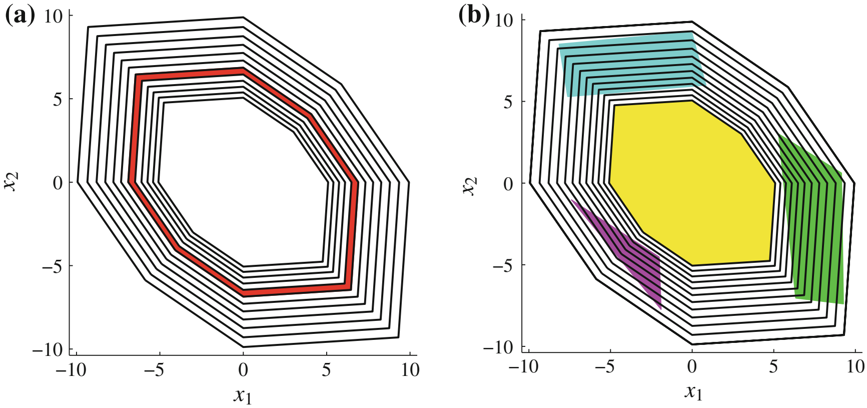


Fig. 10.2 The boundaries of the sublevel sets from Example 10.3 are shown in (**a**), where S_5 is shown in red. The sublevel sets, and implicitly the slices, are shown together with the regions of interest in (**b**)

Proposition 10.1 *Assume that the set of slices* $\{S_i\}_{i=0,\ldots,M}$ *is obtained from a sequence* $\overline{\Theta}$ *satisfying Eq.* (10.4). *Given a state x in the i-th slice, i.e.,* $x \in S_i$, *where* $1 \leq i \leq M$, *its successor state* $x' \in \{\delta_{\mathscr{S}}(x, \gamma) \mid \gamma \in \Gamma\}$ *satisfies* $x' \in S_j$ *for some* $j < i$.

Proof Each sublevel set P_{Θ_i} is ρ-contractive (see Appendix A.5). By the definition of a ρ-contractive set (Definition A.12), we have that

$$x' = A_\gamma x \in \rho P_{\Theta_i} = \{x \in \mathbb{R}^N \mid V(x) \leq \rho\Theta_i\} \text{ for all } \gamma \in \Gamma, x \in P_{\Theta_i}.$$

Hence, it holds that $\delta_{\mathscr{S}}(x, \sigma) \in \rho P_{\Theta_i}$ for all $\sigma \in \Sigma, x \in P_{\Theta_i}$. From (10.4), we have $\rho\Theta_i = \Theta_{i-1}$. Therefore $P_{\Theta_{i-1}} = \{x \in \mathbb{R}^N \mid V(x) \leq \Theta_{i-1}\}$ implies that $P_{\Theta_{i-1}} = \{x \in \mathbb{R}^N \mid V(x) \leq \rho\Theta_i\}$ and hence $P_{\Theta_{i-1}} = \rho P_{\Theta_i}$ and $x' \in P_{\Theta_{i-1}}$. From the definition of slices (10.6), $x' \in S_j$ for some $j < i$. ■

10.1.2 Abstraction Algorithm

We denote by $\sim$ the observational equivalence relation (Definition 1.2) induced by $\mathbf{X}_l, l \in L$, i.e., $x_1 \sim x_2$ if $x_1, x_2 \in \mathbf{X}_l$ for some $l \in L$. The concretization map of the equivalence classes in $\sim$ induces a partition of $X_{\mathscr{S}}$, i.e., $\{\mathbf{X}_l\}_{l \in L}$. Equivalently, an observation preserving partition $\mathbf{P} = \{P_i\}_{i=1,\ldots,m}$ of $X_{\mathscr{S}}$ (i.e., $P_i \cap P_j = \emptyset$ if $i \neq j$, $X = \cup_{i=1,\ldots,m} P_i$ and $P_i \subseteq \mathbf{X}_l$ for some $l \in L$, for each $i = 1, \ldots, m$), induces a relation $\sim_{\mathbf{P}} \subset X \times X$ such that $x_1 \sim_{\mathbf{P}} x_2$ if $x_1, x_2 \in P_i$ for some $i = 1, \ldots, m$.

The proposed abstraction algorithm computes the bisimulation quotient by iteratively refining an observation preserving partition with respect to one step controllable sets. We first explain two procedures, `ComputePre` and `RefineUpdate`, which are used by the main abstraction algorithm.

The procedure `ComputePre`(P, γ) takes as input P, which is a bounded semi-linear set (e.g., a slice), and $\gamma \in \Sigma_{\mathscr{S}}$, which is the switching input, and returns the set $Pre_{T_{\mathscr{S}}}(P, \gamma)$. In general, if P is a semi-linear set, then $Pre_{T_{\mathscr{S}}}(P, \gamma)$ is also a semi-linear set and it can be computed via quantifier elimination. In particular, $Pre_{T_{\mathscr{S}}}$ of a bounded semi-linear set P can be implemented by convex decompositions and repeated applications of (A.7) ($Pre_{T_{\mathscr{S}}}$ of a polytope) as shown in Appendix A.4.

The procedure `RefineUpdate`$(\mathbf{P}, T, \sigma, q)$ is described in Algorithm 18 and is formulated for a general transition system $T = (X, \Sigma, \delta, O, o)$. It takes as input a partition $\mathbf{P}$, a transition system T, a control input $\sigma \in \Sigma$ of T, and a state $q \in X$ of T. The procedure refines $\mathbf{P}$ with respect to $Pre_{T_{\mathscr{S}}}(con(q), \sigma)$ and updates T. If $\mathbf{P}$ consists of only bounded semi-linear sets, then the resulting refinement $\mathbf{P}^+$ consists of only bounded semi-linear sets. This fact allows us to compute $Pre_{T_{\mathscr{S}}}(P, \gamma)$ with `ComputePre`(P, γ) only taking bounded semi-linear sets as inputs.

Algorithm 18 $[\mathbf{P}^+, T^+] =$ `RefineUpdate`$(\mathbf{P}, T, \sigma, q)$

```
Require: A TS $T = (X, \Sigma, \delta, O, o)$, a partition $\mathbf{P}$ where $con(q') \in \mathbf{P}$ for all $q' \in X$, $\sigma \in \Sigma$ and $q \in Q$.
Ensure: $\mathbf{P}^+$ is a finite refinement of $\mathbf{P}$ with respect to ComputePre$(con(q), \sigma)$, $T^+$ is a TS updated from $T$.
Set $\tilde{P} =$ ComputePre$(con(q), \sigma)$.
Set $\mathbf{P}^+ = \mathbf{P}$ and $T^+ = T$.
for all $q' \in X^+$ such that $con(q') \cap \tilde{P} \neq \emptyset$ do
  Replace $q'$ in $Q^+$ by $\{q_1, q_2\}$ and set $con(q_1) = con(q') \cap \tilde{P}$, $con(q_2) = con(q') \backslash \tilde{P}$.
  Replace $con(q')$ in $\mathbf{P}^+$ by $\{con(q_1), con(q_2)\}$.
  Replace each $\delta^+(q', \sigma') = q''$ by $\delta^+(q_1, \sigma') = q''$ and $\delta^+(q_2, \sigma') = q''$.
  Set $\delta^+(q_1, \sigma) = q$
end for
```

We now present the abstraction algorithm (see Algorithm 19) that computes the bisimulation quotient. The main idea is to start with $\{\mathbf{X}_l\}_{l \in L}$, refine the partition according to $\{S_i\}_{i=0,\ldots,M}$:

$$\mathbf{P}_0 = \{\mathbf{X}_l \cap S_i \mid l \in L, i \in \{0, \ldots, M\}\} \tag{10.7}$$

such that $\mathbf{P}_0$ is a refinement to both $\{\mathbf{X}_l\}_{l \in L}$ and $\{S_i\}_{i=0,\ldots,M}$, and then iteratively refine $\mathbf{P}_0$ using the *Pre* operator until the bisimulation quotient is obtained. Starting with $\{\mathbf{X}_l\}_{l \in L}$ is necessary to guarantee that the partition is observation preserving. The second step guarantees that each element in the partition is included in a slice. The third step allows us to ensure that at iteration i of the algorithm, the bisimulation quotient for states within P_{Θ_i} is completed.

Algorithm 19 Abstraction algorithm

```
Require: System $\mathscr{S}$ with dynamics in Eq. 6.4, polyhedral LF $V$ with a contraction rate $\rho$, sets $\mathbf{X}$ and $\{\mathbf{X}_i\}_{i \in L}$.
Ensure: Bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ of the embedding transition system $T_{\mathscr{S}}$ and the corresponding observation preserving partition $\mathbf{P}$.
Generate the sequence of sublevel sets $\bar{P}_\Theta = P_{\Theta_0}, \ldots, P_{\Theta_M}$ and slices $S_0, \ldots, S_M$ as defined in Eq. (10.6).
Set $\mathbf{P}_0 = \{\mathbf{X}_l \cap S_i \mid l \in L, i \in \{0, \ldots, M\}\}$.
Initialize $T_{\mathscr{S}}/_{\sim_0}$ by setting $X_{\mathscr{S}}/_{\sim_0}$ as the set labeling $\mathbf{P}_0$. Set transitions only for the state $q_{l_0} \in X_{\mathscr{S}}/_{\sim_0}$ where $con(q_{l_0}) = S_0 = \mathbf{X}_{l_0}$ with $\delta/_{\sim_0}(q_{l_0}, \gamma) = q_{l_0}$ for all $\gamma \in \Gamma$.
for each $i = 0, \ldots, M - 1$ do
  Set $T_{\mathscr{S}}/_{\sim_{i+1}} = T_{\mathscr{S}}/_{\sim_i}$ and $\mathbf{P}_{i+1} = \mathbf{P}_i$.
  for each $q \in X_{\mathscr{S}}/_{\sim_i}$ where $con(q) \subseteq S_i$ do
    for each $\gamma \in \Gamma$ do
      Set $[\mathbf{P}_{i+1}, T_{\mathscr{S}}/_{\sim_{i+1}}] =$ RefineUpdate $(\mathbf{P}_{i+1}, T_{\mathscr{S}}/_{\sim_{i+1}}, \gamma, q)$.
    end for
  end for
end for
$T_{\mathscr{S}}/_{\approx} = T_{\mathscr{S}}/_{\sim_M}$, and $\mathbf{P} = \mathbf{P}_M$.
```

The correctness of Algorithm 19 will be shown by an inductive argument. Given a sublevel set P_{Θ_i} and a partition $\mathbf{P}_i$ as obtained in Algorithm 17, we define $\tilde{\mathbf{P}}_i$ as

$$\tilde{\mathbf{P}}_i := \{P \in \mathbf{P}_i \mid P \subseteq P_{\Theta_i}\}. \tag{10.8}$$

From Algorithm 19, we see that $\mathbf{P}_0$ partitions all the slices, and since $\mathbf{P}_i$ is a finite refinement of $\mathbf{P}_0$, we can directly see that $\tilde{\mathbf{P}}_i$ is a partition of P_{Θ_i}. Consider an embedding transition system $T_{\mathscr{S}}(i)$ as a subset of $T_{\mathscr{S}}$ with set of states $\{x \in X_{\mathscr{S}} \mid x \in P_{\Theta_i}\}$. Then the following result holds:

Proposition 10.2 *At the completion of the i-th iteration (of the outer loop) of Algorithm 19 (where $\mathbf{P}_{i+1}$ is obtained), if $\sim_i$ induced by $\tilde{\mathbf{P}}_i$ as defined in (10.8) is a bisimulation of $T_{\mathscr{S}}(i)$, then $\sim_{i+1}$ induced by $\mathbf{P}_{i+1}$ is a bisimulation of $T_{\mathscr{S}}(i+1)$.*

Proof We show that at the end of i-th iteration, each transition originating at a state $q \in X_{\mathscr{S}}/_{\sim_{i+1}}$ with $con(q) \subseteq P_{\Theta_{i+1}}$ satisfies the bisimulation requirement (Definition 1.4).

By Proposition 10.1, for each $x \in S_{i+1}$ and $\gamma \in \Gamma$, $x' = A_\gamma x$ must be in a slice with a lower index and thus $x' \in T_{\mathscr{S}}(i)$. Let $x \in con(q) \in \mathbf{P}_i$. If $x' \in S_i$, then we have $x \in \texttt{ComputePre}(con(q'), \gamma)$ (from step 1 of Algorithm 18) for some $q' \in X_{\mathscr{S}}/_{\sim_i}$. The `RefineUpdate` procedure replaces $con(q)$ with $con(q_1) = con(q) \cap P$ and $con(q_2) = con(q) \setminus P$, and updates $T_{\mathscr{S}}/_{\sim_{i+1}}$. We note from the definition of *Pre* operator that for any $x \in con(q_1), x' = A_\gamma x \in con(q')$, thus for any $x_1, x_2 \in con(q_1)$, $x_1 \sim x_2, A_\gamma x_1 \sim A_\gamma x_2$. Moreover, the same argument holds for any subset of $con(q_1)$. Therefore, the transitions given in steps 6 and 7 of Algorithm 1 satisfy the bisimulation requirement.

On the other hand, if $x' \notin S_i$, then $x' \in S_j$ for some $j < i$ and x is already in a set $con(q)$, where $\delta_{\mathscr{S}}/_{\sim_{i+1}}(q, \gamma) = q'$ for some q' satisfying the bisimulation requirement. Therefore, step 8 of Algorithm 19 provides exactly the transitions needed for all states in S_{i+1} and thus, $\sim_{i+1}$ induced by $\tilde{\mathbf{P}}_{i+1}$ is a bisimulation of $T_{\mathscr{S}}(i+1)$. ■

Theorem 10.1 *The quotient transition system $T_{\mathscr{S}}/_{\approx}$ obtained from Algorithm 19 is a bisimulation of the embedding transition system $T_{\mathscr{S}}$.*

Proof From Algorithm 1, we have that $\mathbf{P}_i$ is a refinement of $\{\mathbf{X}_l\}_{l \in L}$ for any $i = 0, \ldots, M$. Therefore, $\mathbf{P}_M$ and its induced relation $\sim_M$ are observation preserving.

At step 3 of Algorithm 19, we set $\delta/_{\sim_0}(q_{l_0}, \gamma) = q_{l_0}, \forall \gamma \in \Gamma$ where $con(q_{l_0}) = P_{\Theta_0}$. From the definition of $T_{\mathscr{S}}$, we see that since q_{l_0} is the only state, $\sim_0$ induced by $\tilde{\mathbf{P}}_0$ is a bisimulation of $T_{\mathscr{S}}(0)$. Using Proposition 10.2 and induction, at iteration $M-1$, we have that $\sim_M$ induced by $\tilde{\mathbf{P}}_M$ is a bisimulation of $T_{\mathscr{S}}(M)$. Note that $\tilde{\mathbf{P}}_M$ is exactly $\mathbf{P}_M$, P_{Θ_M} is exactly $X_{\mathscr{S}}$ and $T_{\mathscr{S}}(M)$ is exactly $T_{\mathscr{S}}$. Therefore $\sim_M$ induced by $\mathbf{P}_M$ is a bisimulation of $T_{\mathscr{S}}$. ■

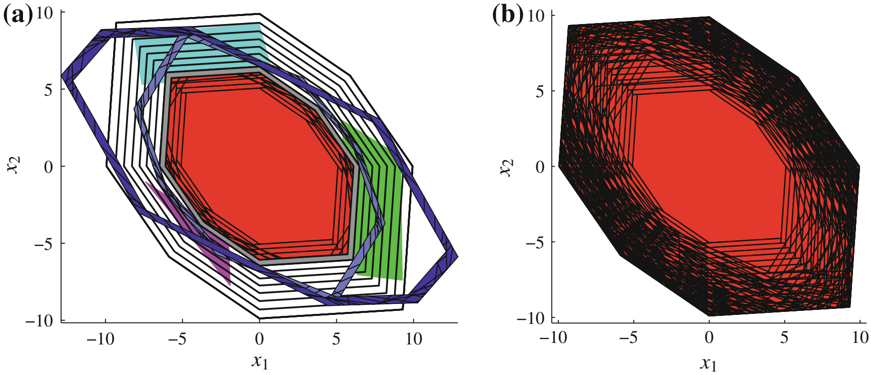


Fig. 10.3 **a** At the end of the forth iteration ($i = 3$), the bisimulation quotient for states within P_{Θ_4} is completed, which are shown in *red* (P_{Θ_3}) and *grey* (S_4). In the forth iteration, the states within $P_{\Theta_{11}} \setminus P_{\Theta_4}$ are partitioned according to $Pre_{T_{\mathscr{S}}}(P, \gamma), P \in S_4$. $Pre_{T_{\mathscr{S}}}(S_4, 1)$ and $Pre_{T_{\mathscr{S}}}(S_4, 2)$ are shown in *dark* and *light blue*, respectively. **b** At the last iteration where $i = 10$, the algorithm is completed. The state space covered by the bisimulation quotient is shown in *red*, covering all of $X_{\mathscr{S}}$

At each iteration i, the number of updated sets is finite as the partition $\mathbf{P}_i$ and the set of inputs Γ are finite, and therefore, we have:

Corollary 10.1 *The bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ can be generated in a finite number of steps, which is determined by the contraction rate of the Lyapunov function.*

Example 10.4 Algorithm 19 is applied on the same setting as in Example 10.3 to compute the bisimulation quotient. $\tilde{\mathbf{P}}_4$ and $\mathbf{P}_{11}$ are shown in Fig. 10.3.

Remark 10.3 The assumptions on sets $\mathbf{X}$, and $\mathbf{X}_{l_0}$ are made for simplicity of presentation. The problem formulation and the approach described in this chapter can be easily extended to arbitrary positively invariant sets X and $\mathbf{X}_{l_0}$ with $\mathbf{X}_{l_0} \subseteq X$, i.e., not obtained as the sublevel sets of a Lyapunov function, by considering the largest sublevel set that is included in $\mathbf{X}_{l_0}$ and the smallest sublevel set that includes $\mathbf{X}$ (Θ_0 and $\Theta_{\mathbf{X}}$ can be made arbitrarily small and arbitrary large, respectively, so as to capture any compact subset of $\mathbb{R}^N$).

10.1.3 Extensions

Although the focus of this chapter is on switched linear systems with polyhedral Lyapunov functions, the presented approach for construction of a bisimulation quotient can also be applied to other classes of discrete-time systems with different Lyapunov functions, if

i. the sublevel sets of the Lyapunov function are semi-linear sets,
ii. the pre-image of a bounded semi-linear set is computable and is also a semi-linear set, and
iii. the dynamical system has a finite set of controls.

The first condition guarantees that the slices are semi-linear sets, and therefore the initial partition is composed of semi-linear sets. The second condition allows us to compute pre-images throughout the algorithm. Finally the last condition is necessary since the pre-images of the partition elements are computed for each control input (line 7 of Algorithm 19).

For example, consider fixed parameter piecewise linear systems $\mathscr{W}$ (see Definition 6.4 and the accompanying text) described by $x(k+1) = A_l x(k)$, $x(k) \in \mathbf{X}_l$, $l \in L$. The sets $\mathbf{X}_l$, $l \in L$ provide a partition of $\mathbf{X}$ and are assumed semi-linear. As discussed in Sect. 10.3, for such systems Lyapunov functions with piecewise polyhedral sublevel sets can be constructed. A fixed parameter piecewise linear system with a piecewise polyhedral Lyapunov function satisfies the three properties stated above. The extension of the method presented in this chapter for such systems requires to refine the initial partition according to $\mathbf{X}_l$, $l \in L$. Then, the proposed algorithms can be used to construct a quotient transition system $T_{\mathscr{W}}/_{\sim}$. In this case, in step 3 of Algorithm 18, it is sufficient to refine a state q' only if $con(q') \subset \mathbf{X}_l$ since only mode l can be active in $con(q')$. By eliminating some of the transitions of $T_{\mathscr{W}}/_{\sim}$ according to $\mathbf{X}_l$, i.e., $\delta_{\mathscr{W}}(q, l) = q'$ only if $con(q) \subseteq \mathbf{X}_l$, we obtain a bisimulation quotient $T_{\mathscr{W}}/_{\approx}$ for the corresponding embedding transition system.

Example 10.5 In this example, we show how the proposed method to construct a bisimulation quotient can be applied to an autonomous fixed-parameter piecewise linear system (Definition 6.4).

We define a two-dimensional system $\mathscr{W}$ with 24 modes ($L = 0, \ldots, 23$). The operating regions $\{\mathbf{X}_l\}_{l \in L}$ are shown in Fig. 10.4. The dynamics of the system are defined by

$$\begin{aligned}
A_i &= \begin{bmatrix} -0.0546 & -0.7764 \\ 0.0212 & -0.8521 \end{bmatrix}, i = 0, 1, 2, 12, 13, 14, \\
A_i &= \begin{bmatrix} -0.0700 & -0.8150 \\ 0.0700 & -0.7300 \end{bmatrix}, i = 3, 4, 5, 15, 16, 17, \\
A_i &= \begin{bmatrix} -0.9200 & -0.0200 \\ 0.7580 & -0.0200 \end{bmatrix}, i = 6, 7, 8, 18, 19, 20 \\
A_i &= \begin{bmatrix} -0.9200 & 0.0200 \\ 0.7580 & -0.0200 \end{bmatrix}, i = 9, 10, 11, 21, 22, 23.
\end{aligned} \tag{10.9}$$

The system is asymptotically stable and admits a piecewise linear Lyapunov function

$$V(x) = \|\mathrm{L}_\gamma x\|_\infty, \quad \text{if } x \in \Omega_\gamma, \quad \mathrm{L}_\gamma \in \mathbb{R}^{l \times N}, l \geq N, l \in \mathbb{N}, \tag{10.10}$$

where $\{\Omega_\gamma\}_{\gamma\in\Gamma}$ is a conic partition of $\mathbb{R}^N$, and

$$\begin{aligned} \mathrm{L}_1 = \mathrm{L}_5 &= \begin{bmatrix} 0.2997 & 4.8651 \\ 4.8651 & -0.2997 \end{bmatrix}, \ \mathrm{L}_2 = \mathrm{L}_6 = \begin{bmatrix} 4.7802 & 0.3846 \\ 0.3846 & -4.0110 \end{bmatrix}, \\ \mathrm{L}_3 = \mathrm{L}_7 &= \begin{bmatrix} -5.0549 & 0.1099 \\ 0.1099 & 5.0549 \end{bmatrix}, \ \mathrm{L}_4 = \mathrm{L}_8 = \begin{bmatrix} -0.1099 & 5.0549 \\ 5.0549 & 0.1099 \end{bmatrix}. \end{aligned} \tag{10.11}$$

In particular, the operating regions are defined with respect to the conic partition as follows:

$$\mathbf{X}_i \subset \Omega_{\lfloor i/3 \rfloor+1} \forall i = 0, \dots, 23.$$

$\mathbf{X}$ and the yellow region ($\bigcup_{i=0}^{7} \mathbf{X}_{3i}$) are defined as sublevel sets of V with bounds 19.75 and 10, i.e.,

$$\mathbf{X} = \{x \in \mathbb{R}^N \mid V(x) \leq 19.75\}, \text{ and}$$

$$\bigcup_{i=0}^{7} \mathbf{X}_{3i} = \{x \in \mathbb{R}^N \mid V(x) \leq 10\}.$$

Note that $\mathbf{X} = \bigcup_{i=0}^{23} \mathbf{X}_i$, and each operating region $\mathbf{X}_i$ are semi-linear sets. In Fig. 10.4, operating regions $\mathbf{X}_{3i-1}, \mathbf{X}_{3i-2}, \mathbf{X}_{3i-3}$ that are in the same cone Ω_i are shown with different colors. For example regions $\mathbf{X}_{12}, \mathbf{X}_{13}$ and $\mathbf{X}_{14}$ in $\Omega_3 \cap \mathbf{X}$ are shown with yellow, white and green, respectively.

The sublevel sets of V (10.10) are not polytopic, however, the slices are still bounded-semi linear sets and can be computed as explained in Sect. 10.1.1 ($M = 11$). These slices and sublevelsets are shown in Fig. 10.5a. Algorithm 19 is used to compute a quotient transition system $T_{\mathscr{W}}/_\sim$. The resulting partition is shown in Fig. 10.5b. By eliminating some of the transitions according to $\{\mathbf{X}_l\}_{l\in L}$, i.e., $\delta_{\mathscr{W}}/_\sim(q,\gamma) = q'$ only if $con(q) \subset \mathbf{X}_l$, we obtain a bisimulation quotient $T_{\mathscr{W}}/_\approx$ for the piecewise linear system. Note that each state q of $T_{\mathscr{W}}/_\approx$ has a single outgoing transition, and the system is not controlled.

10.1.4 Complexity

The proposed algorithm for construction of a bisimulation quotient involves computations of pre-images of bounded semi-linear sets through linear dynamics, intersections and set differences for semi-linear sets at each iteration. The number of operations performed, and hence the complexity of the algorithm, scale linearly with

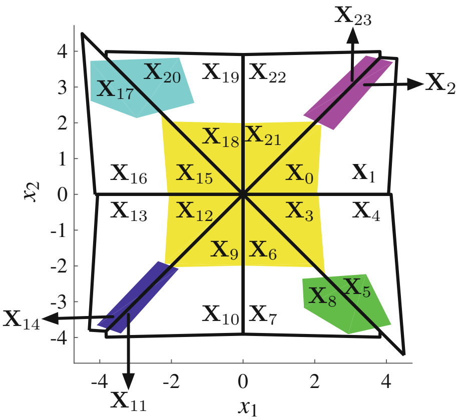


Fig. 10.4 The operating regions of the system from Example 10.5. The inner sublevel set is shown in *yellow*. The intersections of cones Ω_i with $\mathbf{X}$ are shown with *black* boundaries. There are exactly 3 regions in each intersection ($\Omega_i \cap \mathbf{X}$), and these are shown with different colors

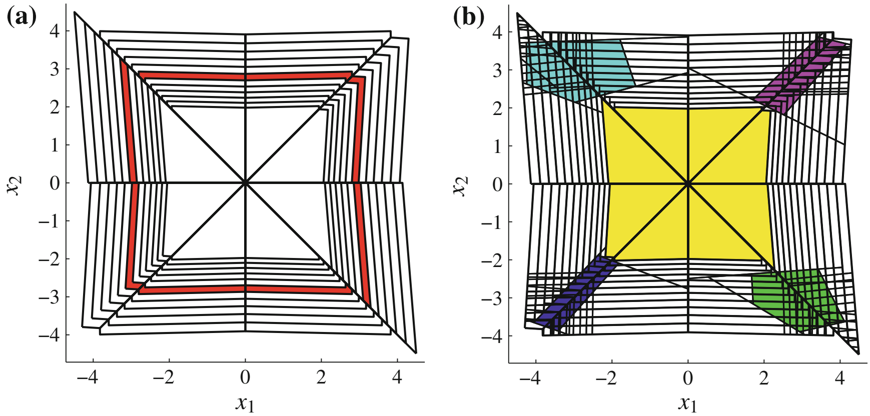


Fig. 10.5 The boundaries of the sublevel sets from Example 10.5 are shown in (**a**), where S_6 is shown in *red*. The boundaries of the semi-linear sets from the resulting partition $\mathbf{P}_{11}$ are shown in (**b**)

the size of the resulting partition $|\mathbf{P}_M|$. Therefore, it is enough to derive an upper bound on $|\mathbf{P}_M|$ with respect to the number of slices, observations and controls.

Let s_i be the number of sets in partition $\mathbf{P}_M$ that are included in slice S_i, i.e.,

$$s_i := |\{P \in \mathbf{P}_M | P \subseteq S_i\}|.$$

Similarly, s_i^0 denotes the number of sets in the initial partition $\mathbf{P}_0$ that are included in slice S_i. In the subsequent analysis, r is used to denote the number of observations within $X_{\mathscr{S}} \setminus \mathbf{X}_{l_0}$ ($r = |L| - 1$), and e is used to denote the number of input symbols ($e = |\Gamma|$).

Lemma 10.1 *The number s_i of sets in the resulting partition $\mathbf{P}_M$ that are included in slice $i \geq 1$ is less than or equal to*

$$\bar{s}_i = r \left(\sum_{k=0}^{i-1} s_k \right)^e . \tag{10.12}$$

Proof A set $P \in \mathbf{P}_0$ with $P \subseteq S_i$ is partitioned only if there exists $\gamma \in \Gamma$ and $P' \in S_j$ for some $j < i$ such that $P \cap Pre_{T_{\mathscr{S}}}(P', \gamma) \neq \emptyset$ and $P \backslash Pre_{T_{\mathscr{S}}}(P', \gamma) \neq \emptyset$ (step 3 of Algorithm 18). Therefore,

(a) for any two states $q_1, q_2 \in X_{\mathscr{S}}/_{\approx}$ with $con(q_1), con(q_2) \subset S_i$ if $o(q_1) = o(q_2)$, there exists $\gamma \in \Gamma$ such that $\delta_{\mathscr{S}}/_{\approx}(q_1, \gamma) = q_1'$, $\delta_{\mathscr{S}}/_{\approx}(q_2, \gamma) = q_2'$ and $q_1' \neq q_2'$.

From Proposition 10.1 and the bisimulation requirement (Definition 1.4), we have that

(b) for each $q \in X_{\mathscr{S}}/_{\approx}$ with $con(q) \subseteq S_i$ and for each $\gamma \in \Gamma$, there exists a state q' with $con(q') \subseteq S_j$ for some $j < i$ such that $\delta_{\mathscr{S}}/_{\approx}(q, \gamma) = q'$.

Given properties *(a)* and *(b)*, the number of sets obtained from partitioning a set $P \in \mathbf{P}_0$ with $P \subseteq S_i$ is bounded by the number of permutations of size e, with unrestricted repetitions, taken from a set of size $\sum_{k=0}^{i-1} s_k$.

The given bound is obtained by observing that $s_i^0 \leq r$ for all $i = 1, \ldots, M$, since the initial partition $\mathbf{P}_0$ is obtained by refining the coarsest observation preserving partition $\{\mathbf{X}_l\}_{l \in L}$ (see (10.7)) according to slice partition. ■

Remark 10.4 The bound on s_i (10.12) is attained when the following conditions are satisfied.

i. $s_i^0 = r$.
ii. Let $\gamma_1 \ldots \gamma_e$ be a sequence of controls such that $\gamma_j \neq \gamma_k$ if $j \neq k$. For each sequence of states $q_1 \ldots q_e$ from $X_{\mathscr{S}}/_{\sim_{i-1}}$ (with unrestricted repetitions), i.e., $con(q_j) \subseteq P_{\Theta_{i-1}}, j = 1, \ldots, e$, there exists a state $q \in X_{\mathscr{S}}/_{\approx}$ with $con(q) \subseteq S_i$, such that $\delta_{\mathscr{S}}/_{\approx}(q, \gamma_j) = q_j$ for all $j = 1, \ldots, e$.

The bound is computed through a combinatorial perspective by utilizing the contractive property of the system. Even though the bound is attainable, the underlying dynamics is not considered explicitly. Therefore, in many cases the bound is not attained (see Example 10.6).

Remark 10.5 $\mathbf{P}_M$ is the coarsest refinement of $\mathbf{P}_0$ satisfying the bisimulation requirement. This claim follows from statement-(a) of the proof of Lemma 10.1, and can easily be shown by an inductive argument on the partitions of the sublevel sets, i.e., $\tilde{\mathbf{P}}_i$ as defined in (10.8).

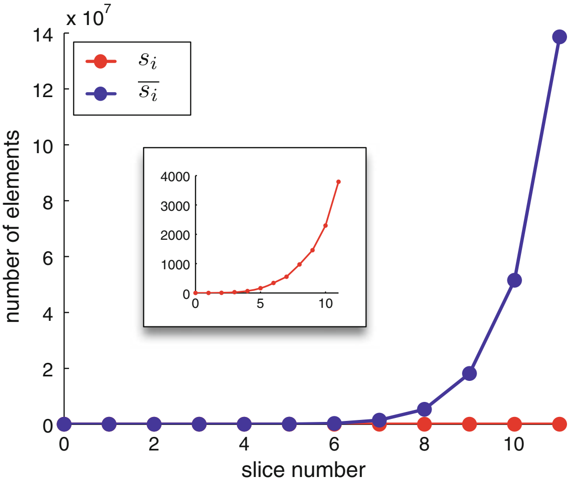


Fig. 10.6 Comparison between the number s_i of elements in a slice and the corresponding bound $\overline{s_i}$ computed as in Eq. (10.12)

Example 10.6 The number of sets in partition $\mathbf{P}_{11}$ from Example 10.4 is illustrated in Fig. 10.6, where a comparison is made between the slice numbers and their bounds computed as in (10.12).

Theorem 10.2 *Let* $p_i = |\{P \mid P \in \mathbf{P}_M, P \subseteq P_{\Theta_i}\}|$ *for all* $i = 0, \ldots, M$. *Then* $p_0 = 1$ *and*

$$p_i \leq (r+1)^{\sum_{j=0}^{i-1} e^j}, \qquad i = 1, \ldots, M. \tag{10.13}$$

Proof As P_{Θ_0} is not partitioned, the claim holds for $i = 0$, i.e., $p_0 = 1$. We prove the claim for $i \geq 1$ by induction. The definitions of p_i, s_i and $\overline{s_i}$ imply that

$$p_{i+1} = p_i + s_{i+1} \leq p_i + \overline{s_{i+1}}.$$

From (10.12) $\overline{s}_1 = r$, and hence the claim holds for $i = 1$ as $p_1 \leq 1 + r$. Assume that inequality (10.13) holds for p_k for some $k \geq 1$. By Lemma 10.1, we have that $\overline{s}_{k+1} = rp_k^e$. Therefore,

$$p_{k+1} \leq p_k + rp_k^e < (r+1)p_k^e.$$

Using the inductive hypothesis on p_k, the left hand side can be rewritten as

$$\begin{aligned}
p_{k+1} &< (r+1)\left((r+1)^{\sum_{j=0}^{k-1} e^j}\right)^e \\
p_{k+1} &< (r+1)\left((r+1)^{\sum_{j=1}^{k} e^j}\right) \\
p_{k+1} &< (r+1)^{\sum_{j=0}^{k} e^j}.
\end{aligned}$$

Thus, inequality (10.13) holds for p_{k+1}, and by induction we conclude that (10.13) holds for all $i = 1, \ldots, M$. ■

The size p_M of the resulting partition of the working set $\mathbf{X}$ is double exponential in M when $e > 1$ (switched systems). Therefore when $e > 1$ the number of *Pre* operations performed, p_{M-1}, is double exponential in $M - 1$. It is easy to verify from (10.13) that the bound is exponential in M for linear systems, i.e., $e = 1$. Note that the derived bound is an upper bound for the worst case, i.e., when $\bar{s}_i = s_i$ for all $i = 0, \ldots, M$.

Remark 10.6 The computational complexity increases with the number of sublevel sets M, which is computed from the working set $\mathbf{X}$, the target set $\mathbf{X}_{l_0}$, and the contraction rate ρ of the Lyapunov function. Therefore, the amount of computation can be adjusted by the choice of the working set $\mathbf{X}$ and the target set $\mathbf{X}_{l_0}$ for a given Lyapunov function. For example, the computation time can be decreased by choosing $\mathbf{X}_{l_0}$ as the largest sublevel set that does not intersect with the regions of observations. In addition, using a Lyapunov function with a minimal contraction rate can decrease the computation time.

10.2 Synthesis and Verification

In this section, we present solutions to Problem 10.1 and Problem 10.2. In particular, our solutions proceed by finding a bisimulation quotient of the embedding transition system as described above. Then, we use techniques presented in Chap. 5 for finite transition systems.

10.2.1 Synthesis

Problem 10.1 aims at finding the largest controlled satisfying region of the embedding transition system $T_{\mathscr{S}}$ of a switched linear system $\mathscr{S}$. To solve this problem, we first construct a bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ of $T_{\mathscr{S}}$ (see Sect. 10.1). Then, we apply the SCLTL CONTROL algorithm (Algorithm 13) presented in Sect. 5.3 on the quotient $T_{\mathscr{S}}/_{\approx}$ and specification ϕ. The algorithm outputs a control strategy $(X^{\phi}_{\mathscr{S}/_{\approx}}, \Omega)$ for the bisimulation quotient. The final step of the proposed solution for Problem 10.1

is the transformation of the control strategy $(X^{\phi}_{\mathscr{S}/\approx}, \Omega)$ for the bisimulation quotient to a control strategy $(X^{\phi}_{\mathscr{S}}, \Omega_{\mathscr{S}})$ for the embedding transition system, and hence for the original dynamical system. The transformation of the set of satisfying states is straightforward:

$$X^{\phi}_{\mathscr{S}} = \bigcup_{q \in X^{\phi}_{\mathscr{S}/\approx}} con(q).$$

The control function Ω is history dependent and takes the form of a feedback control automaton $C = (S_C, S_{C0}, X, \tau, \Gamma, \pi)$, which is constructed from the specification automaton A (an FSA constructed from ϕ), and transition system $T_{\mathscr{S}}/_{\approx}$. The control function Ω can easily be transformed to a control function for $\mathscr{S}$ through the bisimulation relation $\approx$:

$$\Omega_{\mathscr{S}}(x_0 \ldots x_M) := \Omega(q_0 \ldots q_M), \text{ where } x_i \in con(q_i), i = 0, \ldots, M.$$

Example 10.7 Consider the switched linear system given in Example 10.1 and scLTL formula ϕ given in Example 10.2. We first apply Algorithm 19 to construct a bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ of the embedding transition system (see Example 10.4). The quotient system has 9677 states. Then, we apply Algorithm 13 on $T_{\mathscr{S}}/_{\approx}$ and ϕ to obtain a control strategy $(X^{\phi}_{\mathscr{S}/\approx}, \Omega)$ for $T_{\mathscr{S}}/_{\approx}$. The FSA constructed from ϕ, and therefore the control automaton, have 6 states. Finally, we transform the strategy to a strategy $(X^{\phi}_{\mathscr{S}}, \Omega_{\mathscr{S}})$ for the embedding transition system as explained above. The set of satisfying initial states $X^{\phi}_{\mathscr{S}}$ is shown in Fig. 10.7.

Two trajectories of the switched system originating from $X^{\phi}_{\mathscr{S}}$ in closed-loop with $\Omega_{\mathscr{S}}$ are shown in Fig. 10.7. The first trajectory (shown in light blue and labeled with numbers) is generated from $\begin{bmatrix} -4.8676 \\ -5.7102 \end{bmatrix}$ by the switching sequence 2, 2, 1. The second trajectory (shown in dark blue and labeled with letters) is generated from $\begin{bmatrix} 4.9276 \\ 6.2883 \end{bmatrix}$ by the switching sequence 1, 2, 1, 1. Note that both trajectories are satisfying.

10.2.2 Verification

In this section, we consider the problem of verifying the switched system under arbitrary switching, i.e., at every time-step a subsystem is arbitrarily chosen from the set Γ. Note that the system under arbitrary switching is uncontrolled and non-deterministic. Therefore, we define an embedding transition system

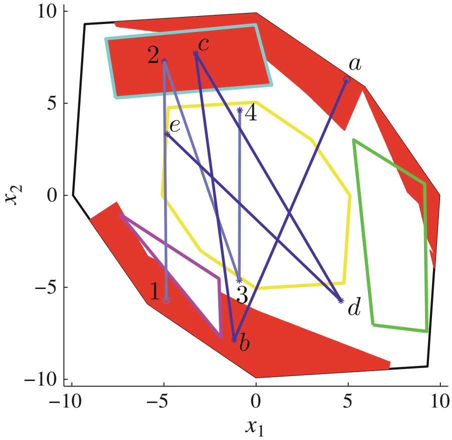


Fig. 10.7 $X_{\mathscr{S}}^{\phi}$ from Example 10.7 is shown in *red*. The color code used in Fig. 10.1 is used for the boundaries of the regions of interests. Two simulated trajectories are indicated by their labels

$T_{\mathscr{S}}^{A} = (X_{\mathscr{S}}, \delta_{\mathscr{S}}^{A}, L, o_{\mathscr{S}})$ that captures arbitrary switching (see Sect. 6.2) from the embedding transition system $T_{\mathscr{S}} = (X_{\mathscr{S}}, \Gamma, \delta_{\mathscr{S}}, L, o_{\mathscr{S}})$ by removing the input set (uncontrolled) and adapting the transitions as follows:

$$x' \in \delta_{\mathscr{S}}^{A}(x) \text{ if there exists } \gamma \in \Gamma \text{ and } \delta_{\mathscr{S}}(x,\gamma) = x'.$$

Note that $T_{\mathscr{S}}^{A}$ is infinite and non-deterministic.

The verification Problem 10.2 aims at finding the largest satisfying region of the embedding transition system $T_{\mathscr{S}}^{A}$. To solve this problem, we first construct a bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ of $T_{\mathscr{S}}$ (see Sect. 10.1). Then, we convert the bisimulation quotient $T_{\mathscr{S}}/_{\approx} = (X_{\mathscr{S}}/_{\approx}, \Gamma, \delta_{\mathscr{S}}/_{\approx}, L, o_{\mathscr{S}}/_{\approx})$ of $T_{\mathscr{S}}$ to $T_{\mathscr{S}}^{A}/_{\approx} = (X_{\mathscr{S}}/_{\approx}, \delta_{\mathscr{S}}^{A}/_{\approx}, L, o_{\mathscr{S}}/_{\approx})$ as follows:

$$q' \in \delta_{\mathscr{S}}^{A}/_{\approx}(q) \text{ if there exists } \gamma \in \Gamma \text{ and } \delta_{\mathscr{S}}/_{\approx}(q,\gamma) = q'.$$

In this case, we have a particular bisimulation relation. We consider the embedding and the quotient transition as systems with a single input labeling all the transitions. Then, we apply the SCLTL CONTROL algorithm (Algorithm 13) presented in Sect. 5.3 to the quotient $T_{\mathscr{S}}^{A}/_{\approx}$ and specification ϕ. The algorithm outputs the set of satisfying initial states ($X_{\mathscr{S}/_{\approx}}^{\phi}$) for the bisimulation quotient. Note that we omit the feedback control function as we have a single "dummy" control. The final step of the proposed solution for Problem 10.2 is the transformation of the set of satisfying states:

$$X_{\mathscr{S}}^{\phi} = \bigcup_{q \in X_{\mathscr{S}/_{\approx}}^{\phi}} con(q). \tag{10.14}$$

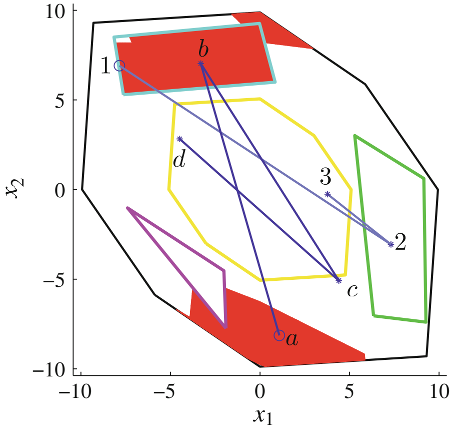


Fig. 10.8 $X_{\mathscr{S}}^{\phi}$ from Example 10.8 is shown in *red*. The color code used in Fig. 10.1 is used for the boundaries of the regions of interests. Two simulated trajectories are indicated by their labels

Example 10.8 Consider again the switched linear system from Example 10.1 and scLTL formula ϕ given in Example 10.2. We first apply Algorithm 19 to construct a bisimulation quotient $T_{\mathscr{S}}/_{\approx}$ of the embedding transition system (see Example 10.4). Then, we construct the bisimulation quotient $T_{\mathscr{S}}^{A}/_{\approx}$ under arbitrary switching from $T_{\mathscr{S}}/_{\approx}$ as explained above. Finally, we apply Algorithm 13 on $T_{\mathscr{S}}^{A}/_{\approx}$ and ϕ to obtain a the set of satisfying states $X_{\mathscr{S}}^{\phi}/_{\approx}$ of $T_{\mathscr{S}}/_{\approx}$. The set of satisfying initial states $X_{\mathscr{S}}^{\phi}$ of the switched system is defined as in (10.14) and is shown in Fig. 10.8. Note that, by definition, this is a subset of the set of initial states found for the synthesis problem (see Fig. 10.7).

Two trajectories of the switched system originating from $X_{\mathscr{S}}^{\phi}$ under arbitrary switching are shown in Fig. 10.8. The first trajectory (shown in light blue and labeled with numbers) is generated from $\begin{bmatrix} -7.8522 \\ 6.9122 \end{bmatrix}$, and the second trajectory (shown in dark blue and labeled with letters) is generated from $\begin{bmatrix} 1.0822 \\ -8.1389 \end{bmatrix}$. At each time step, the switching input is chosen randomly from the set $\Gamma = \{1, 2\}$ for the simulations. Note that both trajectories are satisfying.

Example 10.9 We consider the piecewise linear system given in Example 10.5 and a specification over its set of observations $L = 0, \dots, 23$. The specification is *"A system trajectory never visits* $\mathbf{X}_2$, $\mathbf{X}_{23}$ *(magenta regions),* $\mathbf{X}_{11}$ *and* $\mathbf{X}_{14}$ *(dark blue regions) and eventually visits* $\mathbf{X}_{17}$, $\mathbf{X}_{20}$ *(cyan regions),* $\mathbf{X}_5$ or $\mathbf{X}_8$ *(green regions)"*, which translates to the following scLTL formula:

$$\phi := (\neg(2 \vee 23 \vee 11 \vee 14)\,U\,(0 \vee 3 \vee 6 \vee 9 \vee 12 \vee 15 \vee 18 \vee 21)) \wedge \Diamond(17 \vee 20 \vee 5 \vee 8).$$

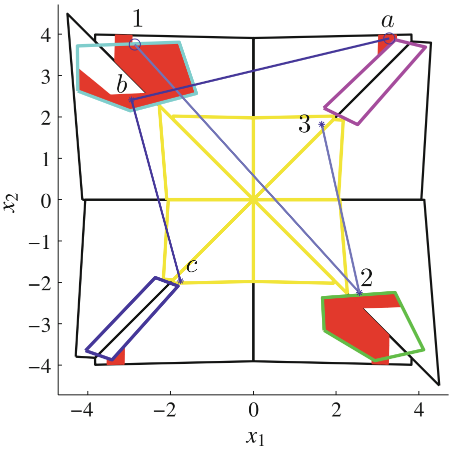


Fig. 10.9 $X_{\mathscr{W}}^{\phi}$ from Example 10.9 is shown in *red*. The color code used in Fig. 10.1 is used for the boundaries of the regions of interests. Two simulated trajectories are indicated by their labels

To verify the system against ϕ, we first construct a bisimulation quotient $T_{\mathscr{W}}/_{\approx}$ as explained in Sect. 10.1.3 (see Example 10.5). Note that the bisimulation quotient is finite and deterministic, and is not controlled. Then, we apply Algorithm 13 to $T_{\mathscr{W}}/_{\approx}$ and ϕ to obtain the set of satisfying states $X_{\mathscr{W}}^{\phi}/_{\approx}$ of $T_{\mathscr{W}}/_{\approx}$. Finally, we compute $X_{\mathscr{W}}^{\phi}$ as in (10.14). $X_{\mathscr{W}}^{\phi}$ and two simulated trajectories are shown in Fig. 10.9.

10.3 Notes

In this chapter, which is based on our work [71, 72], we presented an algorithm for constructing a finite quotient of a stable discrete-time switched linear system in a bounded subset of its state space with the aid of a polyhedral Lyapunov function. In particular, infinity norm Lyapunov functions are used in the examples. However, as explained in Sect. 10.1.3, the proposed methods can be applied if the sublevel sets of the Lyapunov functions are semi-linear. Furthermore, the use of infinity norm Lyapunov functions is not limiting since the existence of a common infinity norm Lyapunov function is a necessary condition for the stability of switched linear systems [124].

The assumption on the existence of a common Lyapunov function (i.e., the assumption on the stability under arbitrary switching) can be relaxed by considering restricted, state dependent switching. In particular, the system from Example 10.5 is adapted from [122], where stabilizing static feedback control laws for discrete-time piecewise affine (PWA) systems are synthesized. The synthesis framework presented in [122] involves computation of piecewise linear Lyapunov functions that admit piecewise polytopic sublevel sets. Here, we considered the stable closed-loop system, i.e., state dependent switching.

The construction of finite abstractions for dynamical systems based on stability properties and Lyapunov functions was studied in [69, 157]. Specifically, approximately bisimilar finite abstractions for continuous-time switched systems were constructed under incremental stability assumptions in [69], where sublevel sets of a common Lyapunov function (or multiple Lyapunov functions with additional assumptions) were used. The abstract model was defined by quantizing the state space of the switched system, and sampling the trajectories originating from the quantized state space. The approximate bisimulation relation guarantees that the trajectories of the abstract model and the original system are close to each other [68]. In [69], the accuracy of the abstraction was defined by the quantization parameter and the Lyapunov function. The method developed in [69] is not limited to linear systems. However, the abstraction is approximate and the case studies presented in [69] highlight that a considerable accuracy requires a large abstract model. The authors of [69] relaxed the incremental stability assumption in their work on construction of approximate simulations [187]. A related approach for stochastic systems can be found in [186]. Another conceptually related work is [157], where N Lyapunov functions were used for the abstraction of N-dimensional continuous-time Morse-Smale systems (e.g., hyperbolic linear systems) to timed automata. The abstraction proposed therein is weaker than bisimulation, but it can be used to verify safety properties.

The method presented in this chapter was implemented in Matlab as a tool called called FBSLS (Finite Bisimulations for Switched Linear Systems), and is available for download at http://sites.bu.edu/hyness/switchedpolybis/. Given a set of observations over polytopic subsets of the state space, a switched linear system with stable subsystems, and a polyhedral Lyapunov function, the tool generates the bisimulation quotient in a finite number of steps. Starting from a sublevel set that includes the origin in its interior, the tool iteratively constructs the bisimulation quotient for any larger sublevel set. The tool also implements synthesis of the switching law and system verification with respect to specifications given as syntactically co-safe Linear Temporal Logic (scLTL) formulas over the observed polytopic subsets, which are also provided as inputs by the user.

Chapter 11
Language Guided Controller Synthesis

In Chap. 9, we treated the temporal logic control problem for a PWA system $\mathscr{W}$. We presented an abstraction-based solution consisting of three main steps. First, we constructed an abstract model of the system based on the partition induced by the polytopic from the definition of $\mathscr{W}$. Second, we synthesized a control strategy for the abstraction from the LTL specification formula. Finally, we refined the strategy back to the original system $\mathscr{W}$.

While being able to accommodate full LTL specifications, the method from Chap. 9 was conservative, mainly because the original (rough) partition of the state space was not refined if a control strategy was not found. In this chapter, we address this limitation. We restrict our attention to specifications given in the co-safe fragment of LTL (scLTL) and present a language-guided synthesis method. Central to our approach is the construction and refinement of an automaton that restricts the search for initial states and control strategies in such a way that the satisfaction of the specification is guaranteed at all times. We focus on fixed-parameter control PWA systems $\mathscr{W}$ (Definition 6.2). Formally, the problem can be stated by using the embedding $T_{\mathscr{W}} = (X_{\mathscr{W}}, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W}}, O_{\mathscr{W}}, o_{\mathscr{W}})$ (Definition 6.6) of $\mathscr{W}$ as follows:

Problem 11.1 *(scLTL Control)* Given a fixed-parameter PWA control system $\mathscr{W}$ (Definition 6.2), and an scLTL formula ϕ over $L \cup \{\text{Out}\}$, find a control strategy $(X^{\phi}_{T_{\mathscr{W}}}, \Omega)$ (Definition 5.1) such that $X^{\phi}_{T_{\mathscr{W}}}$ is the largest controlled satisfying region (Definition 5.2) of the embedding transition system $T_{\mathscr{W}}$ and $\mathscr{L}_{T_{\mathscr{W}}}(X^{\phi}_{T_{\mathscr{W}}}, \Omega) \subseteq \mathscr{L}_{\phi}$.

We propose a solution to Problem 11.1 by relating the control synthesis problem with the construction of a dual automaton from the FSA that accepts the good prefixes of the specification formula ϕ ($\mathscr{L}_{pref,\phi}$). The states of the dual automaton will correspond to polyhedral subsets of $X_{\mathscr{W}}$, and therefore $\mathbf{X}$, and its transitions will be mapped to state feedback controllers. First, the transitions and the states of the dual automaton will be removed by checking their feasibility with respect to the transitions of $T_{\mathscr{W}}$, and hence the particular dynamics of $\mathscr{W}$. Then, its states will be refined

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_11

until feasible region-to-region state feedback controllers are obtained. The refined dual automaton and the controllers labeling its transitions will provide a solution to Problem 11.1 in the form of a feedback control automaton, which will be mapped to a control strategy as given in Definition 5.1. This approach reduces the controller synthesis part of Problem 11.1 to solving a finite number of region-to-region control problems. The outlined control procedure is summarized in Algorithm 20. Through the rest of this section, we provide details of this procedure.

Algorithm 20 SCLTL CONTROL($\mathscr{W}, \phi$): Control strategy $(X^{\phi}_{T_{\mathscr{W}}}, \Omega)$ such that all trajectories in $T_{\mathscr{W}}(X^{\phi}_{T_{\mathscr{W}}}, \Omega)$ satisfy ϕ

```
Construct $T_{\mathscr{W}}$ from $\mathscr{W}$
Translate $\phi$ into an FSA $A = (S, s_0, L \cup \{\text{Out}\}, \delta_A, F)$
Build a dual automaton $A_D$ from $A$
Prune the states and the transitions of $A_D$ according to $T_{\mathscr{W}}$
Refine $A_D$ and synthesize transition controllers
Transform the refined automaton and its transition controllers into a control strategy for $\mathscr{W}$
```

Example 11.1 We define a two-dimensional fixed-parameter PWA control system $\mathscr{W}$ (Definition 6.2) with four modes ($L = \{1, 2, 3, 4\}$). The operating regions are shown in Fig. 11.1a. The dynamics of the system are defined by

$$\begin{aligned}
A_1 &= \begin{bmatrix} 0.92 & 0 \\ 0 & 0.98 \end{bmatrix}, B_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, c_1 = \begin{bmatrix} 0.1 \\ 0 \end{bmatrix}, \\
A_2 &= \begin{bmatrix} 0.95 & 0.1 \\ 0 & 0.98 \end{bmatrix}, B_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, c_2 = \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}, \\
A_3 &= \begin{bmatrix} 0.99 & 0 \\ 0 & 0.98 \end{bmatrix}, B_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, c_3 = \begin{bmatrix} 0.1 \\ 0 \end{bmatrix}, \\
A_4 &= \begin{bmatrix} 0.99 & 0 \\ 0 & 0.98 \end{bmatrix}, B_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, c_4 = \begin{bmatrix} -0.2 \\ 0.2 \end{bmatrix},
\end{aligned} \tag{11.1}$$

and the controls are constrained to set $\mathbf{U} = \{u \mid -1 \leq u_i \leq 1, i = 1, 2\}$. We consider the specification "Start in $\mathbf{X}_1$ and eventually visit $\mathbf{X}_4$. Moreover, until visiting $\mathbf{X}_4$, do not leave $\mathbf{X}$, do not visit $\mathbf{X}_3$, and if $\mathbf{X}_2$ is visited, then $\mathbf{X}_4$ should be visited at the next time step." The specification is formally defined as the scLTL formula

$$\phi = 1 \wedge (\neg \text{Out} \ \wedge \neg 3 \ \wedge \ (\neg 2 \ \vee \ \bigcirc 4) \ \mathcal{U} \ 4)$$

over $L \cup \{\text{Out}\}$. The FSA that accepts the good prefixes of the formula ϕ is shown in Fig. 11.1b. We define the embedding transition system $T_{\mathscr{W}}$ according to Definition 6.6. A run

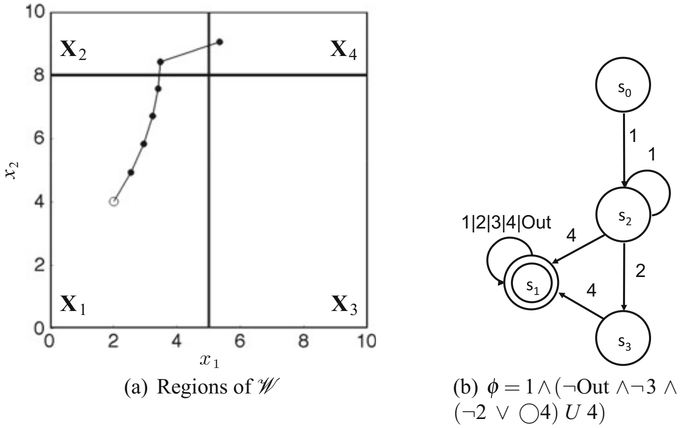
$$w_{X_{\mathscr{W}}} = \begin{bmatrix} 2.0000 \\ 4.0000 \end{bmatrix}, \begin{bmatrix} 2.5400 \\ 4.9200 \end{bmatrix}, \begin{bmatrix} 2.9448 \\ 5.8216 \end{bmatrix}, \begin{bmatrix} 3.2271 \\ 6.7052 \end{bmatrix}, \begin{bmatrix} 3.3984 \\ 7.5711 \end{bmatrix}, \begin{bmatrix} 3.4694 \\ 8.4196 \end{bmatrix}, \begin{bmatrix} 5.3379 \\ 9.0513 \end{bmatrix},$$

of $T_{\mathscr{W}}$ originating at $\begin{bmatrix} 2.0000 \\ 4.0000 \end{bmatrix} \in X_{\mathscr{W}}$ and generated by the input word $w_{\Sigma_{\mathscr{W}}} = w_{\Sigma_{\mathscr{W}}}(1) w_{\Sigma_{\mathscr{W}}}(2) w_{\Sigma_{\mathscr{W}}}(3) w_{\Sigma_{\mathscr{W}}}(4) w_{\Sigma_{\mathscr{W}}}(5) w_{\Sigma_{\mathscr{W}}}(6)$ with $w_{\Sigma_{\mathscr{W}}}(k) = \begin{bmatrix} 1.0000 \\ 1.0000 \end{bmatrix}$ for $k = 1, \ldots, 6$ is shown in Fig. 11.1a. The run produces the output word 1, 1, 1, 1, 1, 2, 4. The run satisfies the specification, since the word produced by the run is accepted by the FSA shown in Fig. 11.1b.

11.1 Dual Automaton Construction and Simplification

In this section, we present the dual automaton definition and construction for a general FSA $A = (S, s_0, \Sigma, \delta_A, F)$. We then consider the particular case of automata with input alphabet $O_{\mathscr{W}}$ and discuss the connection with the control Problem 11.1.

As explained in Sect. 2.2, the good prefixes of an scLTL formula ϕ over Σ are accepted by an FSA $A = (S, s_0, \Sigma, \delta_A, F)$. The dual of an FSA A is constructed by interchanging its states and transitions:



(a) Regions of $\mathscr{W}$

(b) $\phi = 1 \wedge (\neg\text{Out} \wedge \neg 3 \wedge (\neg 2 \vee \bigcirc 4)\ \mathcal{U}\ 4)$

Fig. 11.1 The regions of the PWA system and a simulated trajectory (**a**) and the FSA (**b**) from Example 11.1. For the FSA, s_0 is the initial state and s_1 is the accepting state. For simplicity of the representation, if several transitions exist between two states, only one transition labeled by the set of all inputs (separated by the symbol |) is shown

Definition 11.1 Given an FSA $A = (S, s_0, \Sigma, \delta_A, F)$, its dual automaton is a tuple $A_D = (S_D, S_{D_0}, \Sigma, \delta_D, \tau_D, F_D)$ where

- $S_D = \{(s,l) \mid \delta_A(s,l) \neq \emptyset\}$ is the set of states,
- $S_{D0} = \{(s_0,l) \mid \delta_A(s_0,l) \neq \emptyset\}$ is the set of initial states,
- Σ is the output alphabet,
- $\delta_D : S_D \to 2^{S_D}$ is the transition map, where

$$\delta_D((s_1,l_1)) = \{(s_2,l_2) \mid s_2 = \delta_A(s_1,l_1), \delta_A(s_2,l_2) \neq \emptyset\},$$

- $\tau_D : S_D \to \Sigma$ is the output function, where $\tau_D((s,l)) = l$, and
- $F_D = \{(s,l) \mid \delta_A(s,l) \in F\}$ is the set of accepting (final) states.

Informally, the states of the dual automaton A_D are the transitions of the automaton A. There is a transition between two states of A_D if the corresponding transitions are connected by a state in A. The output alphabet of A_D is the same as the input alphabet of A, i.e., Σ. τ_D is an output function. For a state of A_D, τ_D produces the symbol that enables the transition in A. The set of initial states S_{D_0} of A_D is the set of all transitions that leave the initial state in A. Similarly, the set of final states F_D of A_D is the set of transitions that end in a final state of A.

An accepting run of a dual automaton is a sequence $w_{S_D} = w_{S_D}(1) \dots w_{S_D}(n) \in \Sigma^*$, where $w_{S_D}(1) \in S_{D_0}$, $w_{S_D}(n) \in F_D$ and $w_{S_D}(i+1) \in \delta_D(w_{S_D}(i))$ for all $i = 1, \dots, n-1$. An accepting run w_{S_D} produces a word $w_\Sigma = w_\Sigma(1) \dots w_\Sigma(n)$, where $w_\Sigma(i) = \tau(w_{S_D}(i))$, for all $i = 1, \dots, n$. The output language $\mathscr{L}_{A_D}$ of a dual automaton A_D is the set of all words that are generated by accepting runs of A_D. The construction of a dual automaton A_D from an FSA A guarantees that any word produced by A_D is accepted by A, and any word accepted by A can be produced by A_D:

Proposition 11.1 *The output language of the dual automaton A_D coincides with the language accepted by the automaton A, i.e., $\mathscr{L}_A = \mathscr{L}_{A_D}$.*

Proof $\Rightarrow$: For every word $w_\Sigma = w_\Sigma(1) \dots w_\Sigma(n)$ accepted by automaton A, there exists a run $w_S = w_S(1) \dots w_S(n+1)$ such that $w_S(1)$ is s_0, $w_S(n+1) \in F$ and $\delta_A(w_S(i), w_\Sigma(i)) = w_S(i+1)$ for all $i = 1, \dots, n$. The transition sequence of this run corresponds to an accepting run

$$w_{S_D} = w_{S_D}(1) \dots w_{S_D}(n) = (w_S(1), w_\Sigma(1)) \dots (w_S(n), w_\Sigma(n))$$

of the dual automaton which generates the word w_Σ since $w_{S_D}(1) = (w_S(1), w_\Sigma(1)) \in S_{D_0}$, $w_{S_D}(n) = (w_S(n), w_\Sigma(n)) \in F_D$, $w_{S_D}(i+1) \in \delta_D(w_{S_D}(i))$ for all $i = 1, \dots, n-1$ and $\tau_D(w_{S_D}(i)) = w_\Sigma(i)$ for all $i = 1, \dots, n$.

$\Leftarrow$: Similarly, a run $w_{S_D}(n) = (w_S(1), w_\Sigma(1)) \dots (w_S(n), w_\Sigma(n))$ of A_D that produces the word $w_\Sigma = w_\Sigma(1) \dots w_\Sigma(n)$ yields a run $w_S = w_S(1) \dots w_S(n+1)$ of A which accepts the word w_Σ. ■

The final states of the dual automaton A_D which correspond to the transitions that leave a final state of A can only be reached from the initial states of A_D through other final states of A_D. In other words, if $w_{S_D} = w_{S_D}(1) \ldots w_{S_D}(n)$ is an accepting run of A_D, and $w_{S_D}(n)$ corresponds to a transition that leaves a final state of A, then $w_{S_D} = w_{S_D}(1) \ldots w_{S_D}(n-1)$ is also an accepting run of A_D. For simplicity, we construct a simpler dual automaton $A_{D'}$ by removing such states and the transitions adjacent to them from A_D. Note that any accepting run of $A_{D'}$ is also accepted by A_D, and any accepting run of A_D contains a prefix that is accepted by $A_{D'}$. In the following, for simplicity and without the risk of confusion, we denote the simplified dual automaton by A_D.

As summarized in Algorithm 20, after we construct the embedding transition system $T_{\mathscr{W}} = (X_{\mathscr{W}}, \Sigma_{\mathscr{W}}, \delta_{\mathscr{W}}, O_{\mathscr{W}}, o_{\mathscr{W}})$ of the PWA control system $\mathscr{W}$, we translate the scLTL specification ϕ over $O_{\mathscr{W}}$ into an FSA $A = (S, s_0, O_{\mathscr{W}}, \delta_A, F)$, and then take the dual $A_D = (S_D, S_{D0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$ of the FSA A. A word

$$w_{O_{\mathscr{W}}} = w_{O_{\mathscr{W}}}(1) \ldots w_{O_{\mathscr{W}}}(n)$$

produced by an accepting run

$$w_{S_D} = w_{S_D}(1) \ldots w_{S_D}(n)$$

of A_D, i.e., $\tau_D(w_{S_D}(i)) = w_{O_{\mathscr{W}}}(i)$ for all $i = 1, \ldots, n$, defines a sequence of sets

$$X_{w_{O_{\mathscr{W}}}(1)} \ldots X_{w_{O_{\mathscr{W}}}(n)}$$

in the state space of system $T_{\mathscr{W}}$. Any run

$$w_{X_{\mathscr{W}}} = w_{X_{\mathscr{W}}}(1) \ldots w_{X_{\mathscr{W}}}(n)$$

of $T_{\mathscr{W}}$ with $w_{X_{\mathscr{W}}}(i) \in X_{w_{O_{\mathscr{W}}}(i)}$, $i = 1, \ldots, n$ satisfies the specification by Proposition 11.1. Consequently, the dual automaton construction reduces the controller synthesis part of Problem 11.1 to solving a finite number of region to region control problems defined by the transitions of A_D. Note that these regions correspond to polyhedral sets in the state space of the PWA control system $\mathscr{W}$.

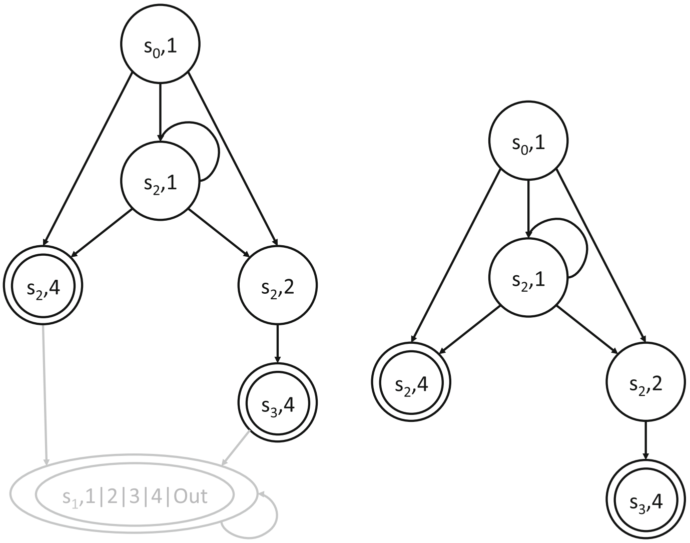
For a dual automaton state $s \in S_D$, we use $R_s \subseteq X_{\mathscr{W}}$ to denote the corresponding region of system $T_{\mathscr{W}}$, e.g., for a state s of the initial dual automaton A_D constructed from A, $R_s := X_{\tau_D(s)}$. For a transition map δ_D, if $s' \in \delta_D(s)$ and $s \neq s'$, we denote the transition from s to s' by (s, s'), and the set of all transitions between two different states of A_D by

$$\Delta_D = \cup_{s \in S_D} \{(s, s') \mid s' \in \delta_D(s), s \neq s'\}.$$

Example 11.2 The dual A_D of the FSA A from Example 11.1 (Fig. 11.1b) is shown in Fig. 11.2a. A possible accepting run $w_{S_D} = (s_0, 1)(s_2, 1)(s_2, 2)(s_2, 4)(s_1, 4)$ of A_D defines a sequence of regions $\mathbf{X}_1\mathbf{X}_1\mathbf{X}_2\mathbf{X}_4\mathbf{X}_4$, and any run $w_{X_\mathscr{W}} = w_{X_\mathscr{W}}(1)w_{X_\mathscr{W}}(2)w_{X_\mathscr{W}}(3)w_{X_\mathscr{W}}(4)w_{X_\mathscr{W}}(5)$ of $T_\mathscr{W}$ from Example 11.1 that follows the sequence of regions, i.e., $w_{X_\mathscr{W}}(1) \in \mathbf{X}_1$, $w_{X_\mathscr{W}}(2) \in \mathbf{X}_1$, $w_{X_\mathscr{W}}(3) \in \mathbf{X}_2$, $w_{X_\mathscr{W}}(4) \in \mathbf{X}_4$, $w_{X_\mathscr{W}}(5) \in \mathbf{X}_4$, satisfies the specification ϕ. The run of the embedding transition system $T_\mathscr{W}$ shown in Fig. 11.1a follows the sequence of regions defined by the accepting run

$$w_{S_D} = (s_0, 1)(s_2, 1)(s_2, 1)(s_2, 1)(s_2, 1)(s_2, 2)(s_3, 4).$$

The final states of the dual automaton A_D which correspond to the transitions that leave a final state of the FSA A are shown in gray, and the dual automaton $A_{D'}$ obtained by removing these states and transitions is shown in Fig. 11.2b. The run $w_{S_{D'}} = (s_0, 1)(s_2, 1)(s_2, 2)(s_3, 4)$ is accepted by both of the dual automata.



(a) A_D (b) $A_{D'}$

Fig. 11.2 The dual of the FSA (Fig. 11.1b) from Example 11.1 (**a**) and the simplified version of it (**b**). For the automaton A_D shown in (**a**), $(s_0, 1)$ is the initial state, and $\{(s_2, 4), (s_3, 4), (s_1, 1), (s_1, 2), (s_1, 3), (s_1, 4), (s_1, \text{Out})\}$ is the set of final states. For simplicity of representation, the states $\{(s_1, 1), (s_1, 2), (s_1, 3), (s_1, 4), (s_1, \text{Out})\}$ are shown together as $(s_1, 1|2|3|4|\text{Out})$. The dual automaton $A_{D'}$ obtained by removing the states $\{(s_1, 1), (s_1, 2), (s_1, 3), (s_1, 4), (s_1, \text{Out})\}$ (represented by $(s_1, 1|2|3|4|\text{Out})$) and the transitions adjacent to them (shown in *gray* in (**a**)) from A_D is shown in (**b**). For automaton $A_{D'}$, $(s_0, 1)$ is the initial state, and $\{(s_2, 4), (s_3, 4)\}$ is the set of final states

A transition $(s, s') \in \Delta_D$ induce a region to region, $R_s - to - R_{s'}$ controller synthesis problem. We say that a transition $(s, s') \in \Delta_D$ is *enabled* if there exists an admissible control law that achieves the transition for all $x \in R_s$. Two conditions are introduced for constructing admissible controllers according to existence of a self transition of the source state s:

- When the source state has a self loop, $s \in \delta_D(s)$, a controller *enables* a transition (s, s') if the corresponding closed-loop trajectories originating in R_s reach $R_{s'}$ in finite time and remain within R_s until they reach $R_{s'}$.
- When the source state does not have a self loop, $s \notin \delta_D(s)$, a transition (s, s') is only enabled if there exists a controller such that the resulting closed-loop trajectory originating in R_s reaches $R_{s'}$ at the next discrete-time instant.

For every transition from the set Δ_D, if a controller that enables the transition can be constructed, then every resulting closed-loop trajectory originating in $\cup_{s \in S_{D0}} R_s$ will satisfy the specifications by Proposition 11.1. However, existence of such controllers is not guaranteed for all the states of $T_{\mathscr{W}}$ within $X_{\mathscr{W}}$.

Problem 11.1 aims at finding the largest subset of $X_{\mathscr{W}}$ for which the region-to-region control problems induced by the dual automaton are feasible. To this end, first, the dual automaton is pruned by checking the feasibility of transitions and states for the embedding transition system $T_{\mathscr{W}}$.

The feasibility of the transitions of the dual automaton is first checked by considering the transitions of $T_{\mathscr{W}}$, i.e., the particular dynamics of the PWA system $\mathscr{W}$ and the set $\mathbf{U}$ where the control input takes values. The set of states that can be reached from the region of state $s \in S_D$ in one step is defined as $Post(R_s, \Sigma_{\mathscr{W}})$, where $Post()$ is defined in Eq. (1.2).

For a transition (s, s') of the dual automaton, if $Post(R_s, \Sigma_{\mathscr{W}}) \cap R_{s'} = \emptyset$, then this transition is considered *infeasible*, since there is no admissible controller that enables this transition. As R_s and $\Sigma_{\mathscr{W}}$ (the control set $\mathbf{U}$) are polyhedral sets, $Post(R_s)$ can be computed by considering the PWA dynamics:

$$Post(R_s, \mathbf{U}) = hull(\{A_{\tau_D(s)} v + B_{\tau_D(s)} u + c_{\tau_D(s)} \mid v \in V(R_s), u \in V(\mathbf{U})\}). \quad (11.2)$$

Algorithm 21 summarizes the pruning procedure. Once the infeasible transitions are removed as in line 2, the following feasibility tests are performed. A state and all of its adjacent transitions are removed from the dual automaton either if it does not have an outgoing transition to another state and it is not a final state or if it does not have an incoming transition from another state and it is not an initial state (line 6). Removing such states and transitions does not reduce the solution space since such states cannot be part of any satisfying trajectory.

Algorithm 21 PRUNING($T_{\mathscr{W}}, A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$): Pruned dual automaton $A_{D'} = (S_{D'}, S_{D'_0}, O_{\mathscr{W}}, \delta_{D'}, \tau_D, F_D)$

```
1: $S_{D'} = S_D$
2: $\delta_{D'}(s) = \{s' \mid s' \in \delta_D(s), Post(R_s, \Sigma_{\mathscr{W}}) \cap R_{s'} \neq \emptyset\}$
3: while $S_D \neq \emptyset$ do
4:   for all $s \in S_D$ do
5:     $S_D = S_D \setminus \{s\}$
6:     if $(s \notin F_D$ AND $\{s' \mid (s, s') \in \Delta_{D'}\} = \emptyset)$ OR $(s \notin S_{D_0}$ AND $\{s' \mid (s', s) \in \Delta_{D'}\} = \emptyset)$ then
7:       $S_{D'} = S_{D0} \setminus \{s\}$
8:       $S_D = S_D \cup \{s' \mid (s, s') \in \Delta_{D'}\} \cup \{s' \mid (s', s) \in \Delta_{D'}\}$
9:     end if
10:   end for
11: end while
12: $S_{D'_0} = S_{D_0} \cap S_{D'}$
```

Example 11.3 The states of the simplified dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$ shown in Fig. 11.2b from Example 11.2 are relabeled and shown in Fig. 11.3a. The regions of the states of the dual automaton A_D are

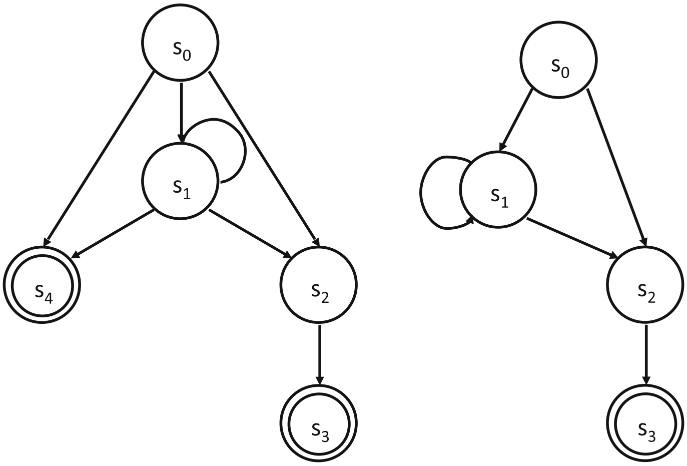
$$R_{s_0} = R_{s_1} = \mathbf{X}_1, R_{s_2} = \mathbf{X}_2, R_{s_3} = R_{s_4} = \mathbf{X}_4,$$

where $\mathbf{X}_i$, $i = 1, 2, 3, 4$ are the operating regions of $\mathscr{W}$ (shown in Fig. 11.1a).

We apply Algorithm 21 to the transition system $T_{\mathscr{W}}$ and automaton A_D, and obtain the pruned initial dual automaton denoted by $A_{D0} = (S_{D0}, S_{D0_0}, O_{\mathscr{W}}, \delta_{D0}, \tau_{D0}, F_{D0})$. We first check the feasibility of the transitions of A_D (line 2).

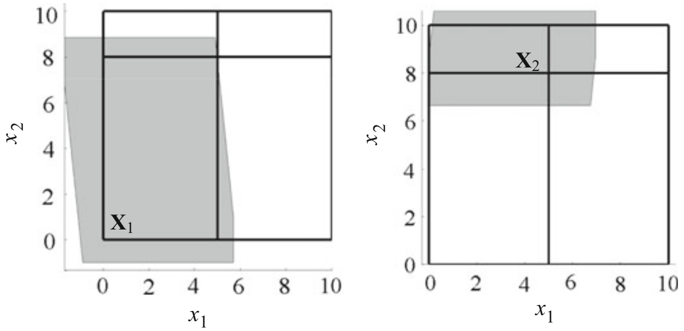
$Post(R_{s_0}, \Sigma_{\mathscr{W}})(= Post(R_{s_1}, \Sigma_{\mathscr{W}}))$ and $Post(R_{s_2}, \Sigma_{\mathscr{W}})$ are shown in Fig. 11.4. The transitions leaving s_0 in the pruned automaton A_{D0} are set as $\delta_{D0}(s_0) = \{s_1, s_2\}$ since $\delta_D(s_0) = \{s_1, s_2, s_4\}$ and $Post(R_{s_0}, \Sigma_{\mathscr{W}}) \cap R_{s_1} \neq \emptyset$, $Post(R_{s_0}, \Sigma_{\mathscr{W}}) \cap R_{s_2} \neq \emptyset$, and $Post(R_{s_0}, \Sigma_{\mathscr{W}}) \cap R_{s_4} = \emptyset$. Note that the transition (s_0, s_4) is not added to the pruned dual automaton since R_{s_4} is not reachable from R_{s_0}. Similarly, the set of transitions leaving s_1 is $\delta_{D0}(s_1) = \{s_1, s_2\}$, since $\delta_D(s_1) = \{s_1, s_2, s_4\}$, $Post(R_{s_1}, \Sigma_{\mathscr{W}}) \cap R_{s_1} \neq \emptyset$, $Post(R_{s_1}, \Sigma_{\mathscr{W}}) \cap R_{s_2} \neq \emptyset$ and $Post(R_{s_1}, \Sigma_{\mathscr{W}}) \cap R_{s_4} = \emptyset$. Finally, $\delta_{D0}(s_2) = \{s_3\}$, since $\delta_D(s_2) = \{s_3\}$ and $Post(R_{s_2}, \Sigma_{\mathscr{W}}) \cap R_{s_3} \neq \emptyset$.

When transitions (s_0, s_4) and (s_1, s_4) are removed, state s_4 becomes infeasible since it is not reachable from the initial state s_0. s_4 satisfies the second condition given in line 6, and hence s_4 and all transitions adjacent to it are removed. All the remaining states are feasible, and the resulting pruned automaton A_{D0} is shown in Fig. 11.3b.



(a) A_D (b) A_{D0}

Fig. 11.3 The dual automaton from Fig. 11.2b (**a**), and the dual automaton A_{D0} obtained by pruning A_D (**b**)



(a) $Post(R_{s_0})(=Post(R_{s_1}))$ (b) $Post\ (R_{s_2})$

Fig. 11.4 One step reachable sets from $\mathbf{X}_1$ (**a**) and $\mathbf{X}_2$ (**b**) are shown in *grey*

11.2 Dual Automaton Refinement

Algorithm 21 guarantees that a non-empty subset of a source region R_s is one-step controllable to the target region $R_{s'}$ corresponding to the transition $(s, s') \in \Delta_{D0}$. However, this does not imply the feasibility of the corresponding region-to-region control problem, which is formally defined next.

11.2.1 Transition Controllers

Problem 11.2 *(Region-to-region control)* For a given transition $(s, s') \in \Delta_D$ of a dual automaton A_D and transition system $T_{\mathscr{W}}$, construct a control function $g : R_s \to \Sigma_{\mathscr{W}}$ such that for all $x \in R_s$ there exists $N_x \in \mathbb{N}_+$, $N_x < \infty$ and

$$\begin{aligned} & x(0) = x, & \\ & x(k+1) = \delta_{\mathscr{W}}(x(k), g(x(k))), & k = 0, \ldots, N_x - 1, \\ & x(k) \in R_s & k = 0, \ldots, N_x - 1, \\ & x(N_x) \in R_{s'}. & \end{aligned}$$

Enabling a transition (s, s') requires a control function that solves Problem 11.2 for that transition if $s \in \delta_D(s)$. If, however, $s \notin \delta_D(s)$, the transition (s, s') is enabled only if there is a solution to the control problem with $N_x = 1$ for all $x \in R_s$.

For a transition $(s, s') \in \Delta_D$, the set of states in R_s that can reach $R_{s'}$ in one step is called a *beacon*. We use $B_{ss'}$ to denote the beacon corresponding to transition (s, s'), which is defined as

$$B_{ss'} := R_s \cap Pre(R_{s'}, \Sigma_{\mathscr{W}}),$$

where $Pre(R_{s'}, \Sigma_{\mathscr{W}})$ is the set of states that can reach $R_{s'}$ in one step as defined in Eq. (1.3). In particular, for an embedding transition system $T_{\mathscr{W}}$, $B_{ss'}$ is defined according to the corresponding affine dynamics:

$$B_{ss'} := \{x \in R_s \mid \exists u \in \mathbf{U} : A_{\tau_D(s)}x + B_{\tau_D(s)}u + c_{\tau_D(s)} \in R_{s'}\}. \tag{11.3}$$

If R_s, $R_{s'}$ and $\mathbf{U}$ are polytopes, then $B_{ss'}$ is also a polytope and computed via orthogonal projection (see Appendix A.3).

By the definition of a beacon $B_{ss'}$, Problem 11.2 can be decomposed in two subproblems. The first problem concerns the computation of a control function which generates a closed-loop trajectory, for all $x \in B_{ss'}$, that reaches $R_{s'}$ in one step. The second problem concerns the construction of a control function which generates a closed-loop trajectory, for all $x \in R_s$, that reaches $B_{ss'}$ in a finite number of steps. These synthesis problems are formally stated next.

Problem 11.3 For a transitions system $T_{\mathscr{W}}$, let B and R be subsets of $X_{\mathscr{W}}$ with $B \subseteq Pre(R)$. Find a control function $g : B \to \Sigma_{\mathscr{W}}$ such that $\delta_{\mathscr{W}}(x, g(x)) \in R$ for all $x \in B$.

Problem 11.4 For a transitions system $T_{\mathscr{W}}$, let B and R be subsets of $X_{\mathscr{W}}$ with $B \subseteq R$. Find a control function $g : R \to \Sigma_{\mathscr{W}}$ such that for all $x \in R$, there exists $N_x \in \mathbb{N}_+$, $N_x < \infty$ and:

$$\begin{aligned}
&x(0) = x, && \\
&x(k+1) = \delta_{\mathscr{W}}(x(k), g(x(k))), && k = 0, \dots, N_x - 1, \\
&x(k) \in R && k = 0, \dots, N_x, \\
&x(N_x) \in B. &&
\end{aligned}$$

For a transition $(s, s') \in \Delta_D$, while Problem 11.3 is always feasible since $B_{ss'} \subseteq Pre(R_{s'})$, Problem 11.4 is not necessarily feasible for any set R_s and corresponding beacon $B_{ss'}$.

Remark 11.1 For the considered PWA dynamics, and polyhedral sets R_s, $R_{s'}$ and $B_{ss'}$, Problems 11.3 and 11.4 can be solved via linear programming. We present two methods for solving these control problems in Appendix A.6. The first solution is based on vertex interpolation and requires iteratively solving a finite number of linear programs. The second solution is based on contractive sets. While more conservative, the latter only requires solving a single linear program.

If a control function $g_{ss'}(\cdot)$ that solves Problem 11.2 for the transition (s, s') is found, we use $\mathbf{W}(s, s')$ to denote the minimum number of steps necessary for all states in R_s to reach $R_{s'}$ in closed-loop with $g_{ss'}(\cdot)$, i.e.,

$$\mathbf{W}(s, s') = \max_{x \in R_s} N_x,$$

where N_x is as defined in Problem 11.2. If no controller can be found, then the weight $\mathbf{W}(s, s')$ is set to infinity.

11.2.2 Refinement

An iterative algorithm is developed to refine the states of the dual automaton, and hence the corresponding regions, until the largest set of satisfying initial states of $T_{\mathscr{W}}$ and the corresponding control strategy is found. Algorithm 22 refines the automaton at each iteration by partitioning the states for which there does not exist an admissible sequence of control actions with respect to reaching a final state. The algorithm does not affect the states of $T_{\mathscr{W}}$ that can reach a final state region and as such, it results in a monotonically increasing, with respect to set inclusion, set of states of system $T_{\mathscr{W}}$ for which there exists an admissible control strategy.

Algorithm 22 REFINEMENT($T_{\mathscr{W}}, A_{D0} = (S_{D0}, S_{D0_0}, O_{\mathscr{W}}, \delta_{D0}, \tau_{D0}, F_{D0})$): Refined dual automaton $A_{DR} = (S_{DR}, S_{DR_0}, O_{\mathscr{W}}, \delta_{DR}, \tau_{DR}, F_{DR})$

```
1: $\mathbf{W}(s, s') = FeasibilityTest(s, s')$, for each $(s, s') \in \Delta_{D0}$
2: $\mathbf{W}^P = ShortestPath(\mathbf{W}, F_{D0})$
3: $CandidateSet = \{(s, s') \mid (s, s') \in \Delta_{D0}, \mathbf{W}^P(s) = \infty, \mathbf{W}^P(s') \neq \infty\}$
4: $k := 0$
5: while $CandidateSet \neq \emptyset$ do
6:    $(s_p, s_t) := \min_{\mathbf{W}^P(s')}\{(s, s') \mid (s, s') \in CandidateSet\}$
7:    $A_{Dk'} = Partitioning(A_{Dk}, s_p, s_t)$
8:    $A_{Dk+1} = Pruning(T_{\mathscr{W}}, A_{Dk'})$
9:    $\mathbf{W}(s, s') = FeasibilityTest(s, s')$, for each $(s, s') \in \Delta_{Dk+1}$
10:   $\mathbf{W}^P = ShortestPath(\mathbf{W}, F_{Dk+1})$
11:   $CandidateSet = \{(s, s') \mid (s, s') \in \Delta_{Dk+1}, \mathbf{W}^P(s) = \infty, \mathbf{W}^P(s') \neq \infty\}$
12:   $k := k + 1$
13: end while
```

At each iteration k of Algorithm 22, for each transition (s, s') from the transition set Δ_{Dk}, Problem 11.2 is solved and a weight $\mathbf{W}(s, s')$ is assigned according to the solution. Then, a weight is computed for each state $s \in S_{Dk}$. The weight $\mathbf{W}^P(s)$ of a state s is defined as the shortest path cost from s to a final state on the graph of the automaton weighted by $\mathbf{W}(\cdot)$. The *ShortestPath* procedure computes a shortest path cost for every state of A_{Dk} using Dijkstra's algorithm [47]. Then, the set of candidate states for partitioning is constructed as follows. A state s that has an infinite weight ($\mathbf{W}^P(s) = \infty$) and a transition ($(s, s') \in \Delta_{Dk}$) to a state that has a finite weight ($\mathbf{W}^P(s') < \infty$) is chosen as a candidate state for partitioning (lines 3 and 11). Then, a transition (s_p, s_t) is selected from the set of candidate set by considering the path costs in line 6. The state s_p with infinite weight is partitioned according to the transition (s_p, s_t). This partitioning procedure is explained next. The partitioned automaton $A_{Dk'}$ is pruned to remove infeasible states and transitions in line 8.

11.2.3 Partitioning

A state s_p is partitioned into a set of states $S_P = \{s_1, \ldots, s_d\}$ via a polyhedral partition of R_{s_p}. The transitions of the new states are inherited from the state s_p and new states are set as initial states if $s_p \in S_{D_0}$ to preserve the automaton language. The partitioning procedure is summarized in Algorithm 23.

Algorithm 23 PARTITIONING($A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D), s_p$): Partitioned dual automaton $A_{D'} = (S_{D'}, S_{D'_0}, O_{\mathscr{W}}, \delta_{D'}, \tau_{D'}, F_D)$

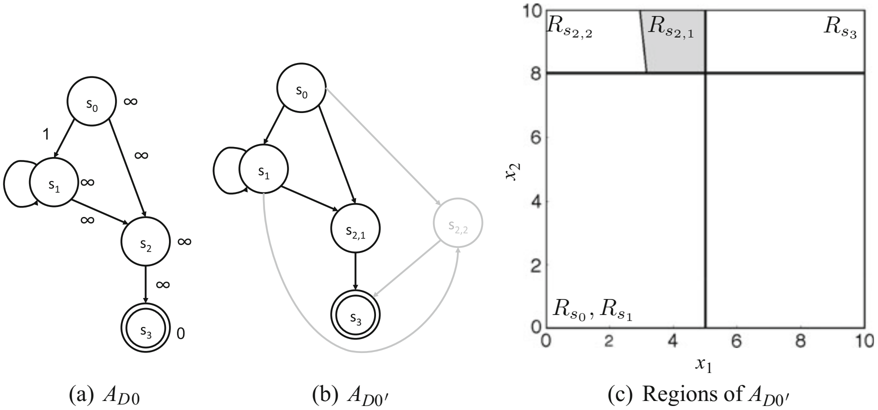
```
1: $S_P = \{s_1, \ldots, s_d\}$ such that $R_{s_p} = \cup_{i=1,\ldots,d} R_{s_i}$ and $R_{s_i} \cap R_{s_j} = \emptyset$ for any $i \neq j$
2: $S_{D'} = (S_D \setminus \{s_p\}) \cup S_P$
3: $S_{D'_0} = S_{D'} \cap S_{D_0}$
4: $S_{D'_0} = S_{D'_0} \cup S_P$ if $s_p \in S_{D_0}$
5: $\delta_{D'}(s) = \delta_D(s) \setminus \{s_p\}$ for each $s \in S_{D'} \setminus S_P$
6: if $s_p \in \delta_D(s)$ then
7:    $\delta_{D'}(s) = \delta_{D'}(s) \cup S_P$
8: end if
9: $\delta_{D'}(s_i) = \delta_D(s_p) \setminus \{s_p\}$ for each $s_i \in S_P$
10: if $s_p \in \delta_D(s_p)$ then
11:    $\delta_{D'}(s_i) = \delta_{D'}(s_i) \cup S_P$
12: end if
13: $\tau_{D'}(s) = \tau_D(s)$ for each $s \in S_{D'} \setminus S_P$
14: $\tau_{D'}(s_i) = \tau_D(s_p)$ for each $s \in S_P$
```

While Algorithm 23 is designed for partitioning a state s_p via any polytopic partition of R_{s_p} (line 1), here, we follow a heuristic partitioning strategy guided by a transition (s_p, s_t). Region R_{s_p} is partitioned into two subregions using a hyperplane of the beacon $B_{s_p s_t}$. The hyperplane that maximizes the radius of the Chebyshev ball that can fit in any of the resulting regions is chosen as the partitioning criterion. Choosing a hyperplane of the beacon ensures that only one of the resulting states can have a transition to s_t. Even if a controller that enables the transition to s_t does not exist for this state, after further partitioning the beacon becomes a state itself and the transition is enabled for it. The employed maximal radius criterion is likely to result in a less-complex partition, as opposed to iteratively computing one-step controllable sets to $B_{s_p s_t}$.

The dual automaton at the k-th iteration of Algorithm 22 is denoted by $A_{Dk} = (S_{Dk}, S_{Dk_0}, O_{\mathscr{W}}, \delta_{Dk}, \tau_{Dk}, F_{Dk})$. For the dual automaton A_{Dk}, the set $\mathbf{X}_k^{\phi}$ is defined as the union of the regions corresponding to initial states with finite weights:

$$\mathbf{X}_k^{\phi} := \bigcup_{s \in \{s' \in S_{Dk_0} | \mathbf{W}^P(s') < \infty\}} R_s. \qquad (11.4)$$

The refinement algorithm (Algorithm 22) stops when there are no transitions from states with infinite weights to states with finite weights, i.e., when the candidate set for partitioning is empty. Note that when a state is partitioned, the new states inherit only the transitions that satisfy a reachability condition, as the infeasible transitions and states are removed by the pruning algorithm in line 8 of Algorithm 22. Therefore, when the algorithm stops all states have finite weights. Moreover, the feasibility check of the transitions (line 2 of Algorithm 21) and the feasibility check of the states (line 6 of Algorithm 21) guarantees that either



(a) A_{D0} (b) $A_{D0'}$ (c) Regions of $A_{D0'}$

Fig. 11.5 The initial dual automaton (**a**), the partitioned dual automaton (**b**), and the state regions (**c**) in the first iteration of the refinement algorithm. The weights of the states and the transitions are shown in (**a**). The infeasible states and infeasible transitions of $A_{D0'}$ are shown in *gray* in (**b**). The state regions of $A_{D0'}$ and the beacon $B_{s_2 s_3}$ (= $R_{s_{2,1}}$, in *gray*) are shown in (**c**)

- the regions of the states removed in line 8 of Algorithm 22 are not reachable from initial states with finite weights through automaton transitions, or
- the regions of the final states are not reachable from the regions of the states removed in line 8 of Algorithm 22 through automaton transitions.

Example 11.4 We apply Algorithm 22 to the transition system $T_{\mathscr{W}}$ and the pruned automaton A_{D0} (shown in Fig. 11.3b) from Example 11.3. We explain and illustrate schematically the first three iterations of the algorithm. First, for each transition (s, s') from the set Δ_{D0}, a region-to-region control problem (Problem 11.2) is solved and a weight $\mathbf{W}(s, s')$ is assigned (line 1). A control function is found only for transition (s_0, s_1), and the weights of the other transitions are set to infinity. Then, a weight is computed for each state. The transition and the state weights for A_{D0} are shown in Fig. 11.5a.

The candidate set for A_{D0} is $\{(s_2, s_3)\}$. s_2 is partitioned into $\{s_{2,1}, s_{2,2}\}$ according to the beacon $B_{s_2 s_3}$, which is shown in Fig. 11.5c. The refined automaton obtained by partitioning s_2 is shown in Fig. 11.5b, where the infeasible states and the infeasible transitions that are removed by the pruning algorithm are shown in gray. While the transition $(s_{2,2}, s_3)$ is removed via the feasibility check performed in line 2 of Algorithm 21, state $s_{2,2}$ and transitions adjacent to it are removed in line 6 of Algorithm 21.

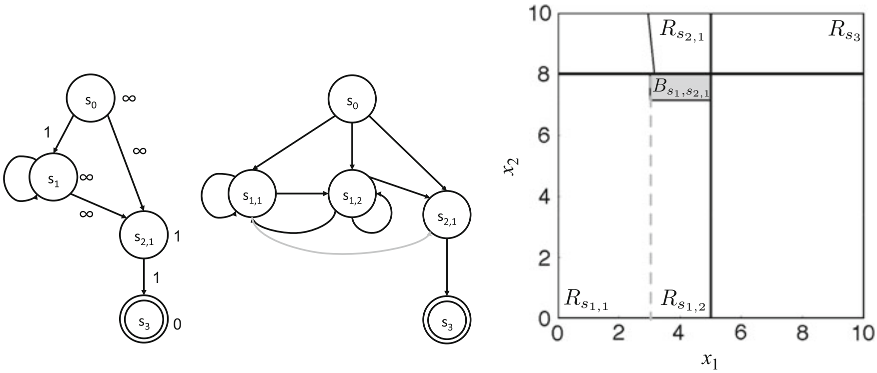
The automaton A_{D1} generated in the first iteration is shown in Fig. 11.6a. In the second iteration, the transition $(s_1, s_{2,1})$ is chosen from the candidate set $\{(s_0, s_{2,1}), (s_1, s_{2,1})\}$ for partitioning. The hyperplane of the beacon $B_{s_1 s_{2,1}}$ that maximizes the radius of the Chebyshev ball that can fit in $R_{s_{1,1}}$ and $R_{s_{1,1}}$ is used to partition s_1 into $\{s_{1,1}, s_{1,2}\}$. The second iteration is illustrated schematically in Fig. 11.6.

Automaton A_{D2}, the weights of its transitions and states are shown in Fig. 11.7a. In the third iteration, the transition $(s_{1,2}, s_{2,1})$ is chosen from the candidate set $\{(s_0, s_{2,1}), (s_{1,2}, s_{2,1})\}$, and $s_{1,2}$ is partitioned into $\{s_{1,2,1}, s_{1,2,2}\}$. After this refinement step, the beacon $B_{s_{1,2}, s_{2,1}}$ becomes the region of state $s_{1,2,1}$, and the transition $(s_{1,2,1}, s_{2,1})$ is feasible for $s_{1,2,1}$ since $R_{1,2,1} \subseteq Pre(s_{2,1})$. The regions of automaton $A_{D2'}$ are shown in Fig. 11.7b.

The algorithm partitions a state at each iteration and terminates at the 13-th iteration. For a dual automaton A_{Dk} obtained at the k-th iteration, the set of satisfying initial states $\mathbf{X}_k^{\phi}$ of $T_{\mathscr{W}}$ is computed as the union of the regions of the initial states with finite weights (see Eq. 11.4). These regions are shown in Fig. 11.8.

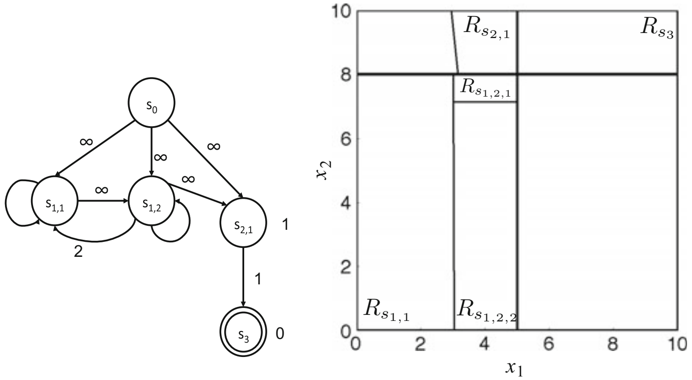
Proposition 11.2 *Given an arbitrary iteration $k \geq 1$ of Algorithm 22, the set $\mathbf{X}_k^{\phi}$ as defined in Eq.* (11.4) *has the following properties:*

(i) *There exists a sequence of admissible control actions such that every closed-loop trajectory of system $T_{\mathscr{W}}$ originating in $\mathbf{X}_k^{\phi}$ satisfies the formula ϕ, and*



(a) A_{D1} (b) $A_{D1'}$ (c) Regions of $A_{D1'}$

Fig. 11.6 Automaton A_{D1} (**a**), partitioned automaton $A_{D1'}$ (**b**), and the state regions (**c**) in the second iteration of the refinement algorithm. The weights of the states and the transitions are shown in (**a**), and the infeasible transition is shown in *gray* in (**b**). The beacon $B_{s_1 s_{2,1}}$ is shown in *gray*, and the hyperplane of $B_{s_1 s_{2,1}}$ that is used to partition R_1 is shown with a *dashed gray line* in (**c**). As s_0 is not partitioned, $R_{s_0} = R_{s_{1,1}} \cup R_{s_{1,2}}$



(a) A_{D2} (b) Regions of $A_{D2'}$

Fig. 11.7 Automaton A_{D2} (**a**) and the state regions (**b**) in the third iteration of the refinement algorithm. The weights of the states and the transitions are shown in (**a**)

(ii) $\boldsymbol{X}_{k-1}^{\phi} \subseteq \boldsymbol{X}_{k}^{\phi}$.

Proof (i) A finite path weight for a state $s \in S_{Dk_0}$ implies that there exists an accepting automaton run $w_{S_{Dk}} = w_{S_{Dk}}(1), \ldots, w_{S_{Dk}}(n)$ such that $s = w_{S_{Dk}}(1)$, $\mathbf{W}(w_{S_{Dk}}(i), w_{S_{Dk}}(i+1)) < \infty$ for all $i = 1, \ldots, n-1$, and $\mathbf{W}^P(s) = \sum_{i=1}^{n-1} \mathbf{W}(w_{S_{Dk}}(i), w_{S_{Dk}}(i+1))$. As a transition weight is assigned according to the existence of the controller that enables the transition, there exists a control sequence that ensures that every closed-loop trajectory originating in $R_{w_{S_{Dk}}(1)}$ reaches $R_{w_{S_{Dk}}(n)}$ by following the automaton path. Considering that removing states and transitions only reduces the language of the automaton, by Proposition 11.1 it follows that $\mathscr{L}_{A_{D0}} \subseteq \mathscr{L}_{pref,\phi}$. Since the proposed partitioning procedure preserves the language, we have $\mathscr{L}_{A_{Dk}} \subseteq \mathscr{L}_{A_{Dk-1}}$. Consequently, $\mathscr{L}_{A_{Dk}} \subseteq \mathscr{L}_{pref,\phi}$ and the resulting trajectories satisfy the formula.

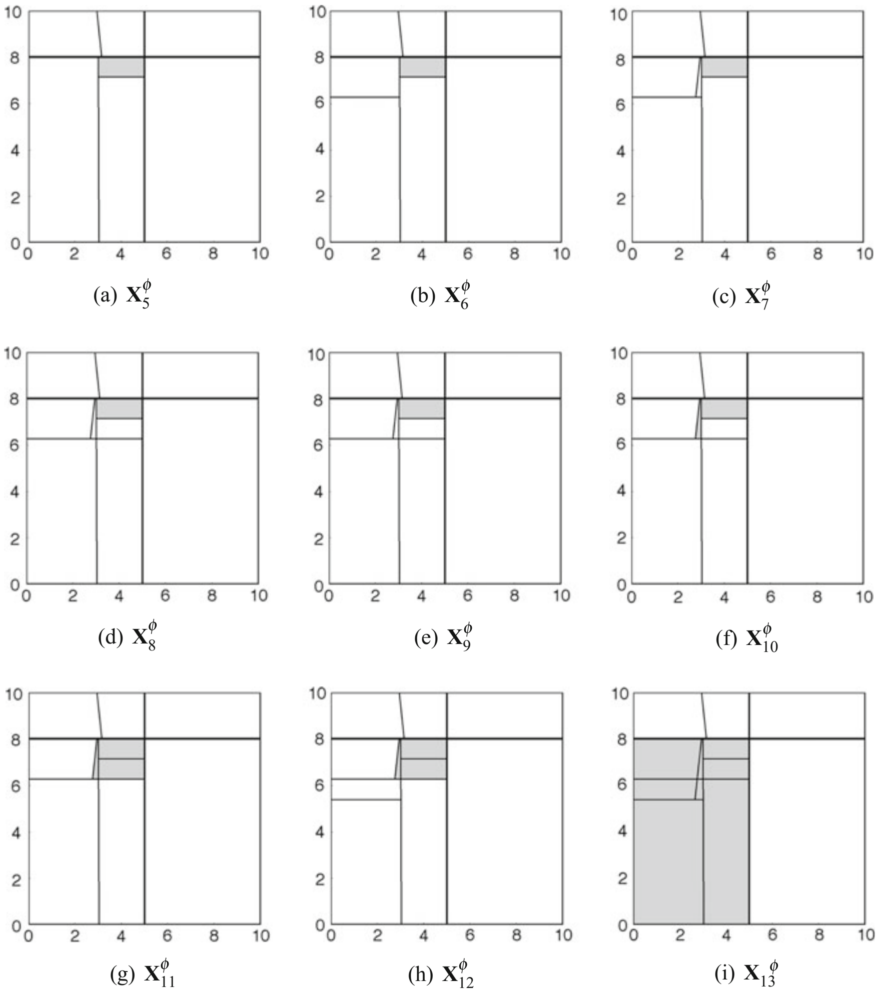
(ii) For any $x \in \mathbf{X}_{k-1}^{\phi}$, there exists an accepting automaton run

$$w_{S_{Dk-1}} = w_{S_{Dk-1}}(1), \ldots, w_{S_{Dk-1}}(n)$$

with $x \in R_{w_{S_{Dk-1}}(1)}$ and $\mathbf{W}^P(w_{S_{Dk-1}}(1)) = \sum_{i=1}^{n-1} \mathbf{W}(w_{S_{Dk-1}}(i), w_{S_{Dk-1}}(i+1))$. Let s_p be the state chosen for partitioning at iteration k. Then, $\mathbf{W}^P(s_p) = \infty$ and s_p can not appear in $w_{S_{Dk-1}}$, i.e., $s_p \neq w_{S_{Dk-1}}(i)$, $i = 1, \ldots, n$, since $\mathbf{W}^P(w_{S_{Dk-1}}(i)) < \infty$ for all $i = 1, \ldots, n$. As only the transitions adjacent to s_p, and the states with infinite weights are affected by partitioning, $w_{S_{Dk-1}}(i) \in S_{Dk}$ for all $i = 1, \ldots, n$, and $w_{S_{Dk-1}}(i+1) \in \delta_{Dk}(w_{S_{Dk-1}}(i))$ for all $i = 1, \ldots, n-1$. Therefore, $x \in R_{w_{S_{Dk}}(1)}$,

$$w_{S_{Dk}} = w_{S_{Dk}}(1), \ldots, w_{S_{Dk}}(n), \text{ where } w_{S_{Dk}}(i) = w_{S_{Dk-1}}(i), \text{ for all } i = 1, \ldots, n,$$

is an accepting run of A_{Dk} with finite weight and thus, $x \in \mathbf{X}_{k}^{\phi}$. Observing that $x \in \mathbf{X}_{k-1}^{\phi}$ was chosen arbitrarily completes the proof. ■



(a) $\mathbf{X}_5^{\phi}$ (b) $\mathbf{X}_6^{\phi}$ (c) $\mathbf{X}_7^{\phi}$

(d) $\mathbf{X}_8^{\phi}$ (e) $\mathbf{X}_9^{\phi}$ (f) $\mathbf{X}_{10}^{\phi}$

(g) $\mathbf{X}_{11}^{\phi}$ (h) $\mathbf{X}_{12}^{\phi}$ (i) $\mathbf{X}_{13}^{\phi}$

Fig. 11.8 The set of satisfying initial states found at iterations $k = 5, \ldots, 13$ are shown in *gray*

11.3 Control Strategy

In this section, we provide a solution to Problem 11.1 by using the refined dual automaton $A_{DR} = (S_{DR}, S_{DR_0}, O_{\mathscr{W}}, \delta_{DR}, \tau_{DR}, F_{DR})$ and its transition controllers synthesized during the refinement, e.g., $\{g_{ss'}(\cdot) \mid \mathbf{W}(s, s') < \infty\}$. We first define a successor function, which determines the strategy for a given refined dual automaton.

Definition 11.2 A function $\Gamma : S_D \to S_D$ is called a *successor function* for a dual automaton $A_D = (S_D, S_{D_0}, \Sigma_D, \delta_D, \tau_D, F_D)$ (Definition 11.1), and transition weight function $\mathbf{W} : S_D \times S_D \to \mathbb{N}_+$ if it satisfies:

i. $\Gamma(s) = s$ if and only if $s \in F_D$.

ii. $\Gamma(s) \in \delta_D(s)$ and $\mathbf{W}(s, \Gamma(s)) \neq \infty$ for all $s \in S_D \setminus F_D$.
iii. For each $s \in S_D$, there exists a finite $n \in \mathbb{N}$ such that $\Gamma^n(s) \in F_D$, where $\Gamma^n(s) = \Gamma(\Gamma^{n-1}(s))$ and $\Gamma^0(s) = s$.

A successor function $\Gamma(\cdot)$ of a dual automaton A_D induces a partial order $\preceq_\Gamma$ over S_D such that $\Gamma^n(s) \preceq_\Gamma s$ for all $n \in \mathbb{N}$. Moreover final states of A_D are fixed points of the function $\Gamma(\cdot)$.

Definition 11.3 Given a successor function $\Gamma(\cdot)$ for a dual automaton A_D, the *automaton potential* of a state $s \in S_D$ is defined by the function $\mathbf{W}_\Gamma : S_D \to \mathbb{N}$ as follows:

$$\mathbf{W}_\Gamma(s) = \begin{cases} 0 & \text{if } s \in F_D \\ \mathbf{W}(s, \Gamma(s)) + \mathbf{W}_\Gamma(\Gamma(s)) & \text{otherwise.} \end{cases} \tag{11.5}$$

For any $s, s' \in S_D$, $s \preceq_\Gamma s'$ implies that $\mathbf{W}_\Gamma(s) \leq \mathbf{W}_\Gamma(s')$.

A successor function Γ according to Definition 11.2 can easily be constructed by traversing the graph of the automaton staring from the final states. Let S_T denote the states that are traversed. Initially, set $\Gamma(s_f) := s_f$, $s_f \in F_{DR}$ and $S_T := S_{DR}$. Then iteratively choose a state s from the set $S_{DR} \setminus S_T$ such that there exists a state $s' \in S_T$ and $\mathbf{W}(s, s') < \infty$, and set $\Gamma(s) := s'$ and $S_T := S_T \cup \{s\}$. Note that, while constructing a successor function Γ_{SP} as outlined above, if s' and the candidate state s are chosen such that

$$(s, s') = \arg \min_{(s,s') \in \{S_{DR} \setminus S_T\} \times S_T \text{ and } s' \in \delta_{DR}(s)} \mathbf{W}(s, s') + \mathbf{W}_{\Gamma_{SP}}(s'), \tag{11.6}$$

then $\mathbf{W}_{\Gamma_{SP}}(s)$ is the cost of the shortest path from s to a final automaton state on the graph of the dual automaton weighted by $\mathbf{W}$. The successor function $\mathbf{W}_{\Gamma_{SP}}$ can equivalently be defined as follows:

$$\Gamma_{SP}(s) = \begin{cases} s & s \in F_{DR} \\ \arg\min_{\mathbf{W}^P(s')}\{s' \in \delta_{DR}(s)\} & s \notin F_{DR} \end{cases} \tag{11.7}$$

In particular, $\Gamma_{SP}(s)$ is the state that succeeds s in the shortest path from s to a final automaton state. The pruning step and the termination condition ($\mathit{CandidateSet} = \emptyset$) guarantee that $\mathbf{W}^P(s) < \infty$ for each $s \in S_{DR}$. Therefore, the function $\Gamma_{SP}(\cdot)$ satisfies all the properties of Definition 11.2, and is a successor function for A_{DR}. It is clear that the automaton potential function induced by $\Gamma_{SP}(\cdot)$ equals to $\mathbf{W}^P(\cdot)$, i.e., $\mathbf{W}_{\Gamma_{SP}}(s) = \mathbf{W}^P(s)$ for each $s \in S_{DR}$.

We define a feedback control automaton from the refined dual automaton A_{DR} and a successor function. For a given state x_0 of $T_{\mathscr{W}}$, there might be more than one accepting automaton run originating from an initial state $s \in S_{DR_0}$ with $x_0 \in R_s$. Essentially, the successor function determines an automaton run among those, and the corresponding controller sequence.

Definition 11.4 Given a dual automaton $A_{DR} = (S_{DR}, S_{DR_0}, O_{\mathscr{W}}, \delta_{DR}, \tau_{DR}, F_{DR})$, its transition controllers, the corresponding transition weight function $\mathbf{W} : S_{DR} \times S_{DR} \rightarrow \mathbb{N}_+$, and a successor function $\Gamma : S_{DR} \rightarrow S_{DR}$, a feedback control automaton $C = (S_C, S_{C0}, \mathbf{X}_{\mathscr{W}}, \tau_C, \Sigma_{\mathscr{W}}, \pi)$ is defined as

- $S_C = S_{DR}$ is the set of states,
- $S_{C0} = S_{DR_0}$ is the set of initial states,
- $X_{\mathscr{W}}$ is the set of inputs (the set of states of $T_{\mathscr{W}}$),
- $\tau_C : S_C \times X_{\mathscr{W}} \rightarrow S_C$ is the memory update function defined as:

$$\tau_C(s, x) = \begin{cases} \Gamma(s) & \text{if } x \in R_{\Gamma(s)} \\ s & \text{if } x \in R_s \setminus R_{\Gamma(s)} \\ \perp & \text{otherwise} \end{cases}$$

- $\Sigma_{\mathscr{W}}$ is the set of outputs (the set of inputs of $T_{\mathscr{W}}$),
- $\pi : S_C \times X_{\mathscr{W}} \rightarrow \Sigma_{\mathscr{W}}$ is the output function:

$$\pi(s, x) = g_{s\Gamma(s)}(x),$$

  where $g_{s\Gamma(s)} : R_s \rightarrow \Sigma_{\mathscr{W}}$ is the feedback control function that solves Problem 11.2 for transition $(s, \Gamma(s))$.

The set of initial states $X^{\phi}_{T_{\mathscr{W}}}$ of $T_{\mathscr{W}}$ is defined as the union of the regions of the initial states of C:

$$X^{\phi}_{T_{\mathscr{W}}} = \bigcup_{s \in S_{C0}} R_s. \tag{11.8}$$

The feedback control automaton C reads the current state of $T_{\mathscr{W}}$, produces the control input to be applied at that state of the transition system, and updates its internal state. The control function Ω is given by C as follows: for a sequence $x(1) \ldots x(n)$, $x(1) \in X^{\phi}_{T_{\mathscr{W}}}$, we have $\Omega(x(1) \ldots x(n)) = \sigma$, where $\sigma = \pi(s(n), x(n))$, $s(i+1) = \tau_C(s(i), x(i))$, and $x(i+1) = \delta_{\mathscr{W}}(x(i), \pi(s(i), x(i)))$, for all $i \in \{1, \ldots, n-1\}$. As the controls and the states of the transition system $T_{\mathscr{W}}$ and the PWA control system $\mathscr{W}$ are the same, the control function Ω directly maps to a control function for $\mathscr{W}$.

For transition system $T_{\mathscr{W}}$ and a control automaton C as constructed in Definition 11.4, the time required to satisfy the specification for trajectories of $T_{\mathscr{W}}$ originating in R_s, $s \in S_{C0}$ is upper bounded by $\mathbf{W}_{\Gamma}(s)$. Consequently, the time required to satisfy the specification starting from any state $x \in X^{\phi}_{T_{\mathscr{W}}}$ of system $T_{\mathscr{W}}$ is upper bounded by

$$\max_{s \in S_{C0}} \mathbf{W}_{\Gamma}(s).$$

The successor function defined in Eq. (11.7) minimizes this bound.

Proposition 11.3 *Every closed-loop trajectory produced by the control strategy $(X^{\phi}_{T_{\mathscr{W}}}, \Omega)$ satisfies formula ϕ.*

Proof The set of states, the set of initial states and the set of final states of the control automaton C and the refined dual automaton A_{DR} are the same. Moreover, if $\tau_C(s, x) = s'$, then $s' \in \delta_{DR}(s)$. Consequently, an accepting run of C corresponds to an accepting run of A_{DR}. The proof that all the trajectories of the closed loop system satisfy the formula follows immediately from Proposition 11.2 since $X^{\phi}_{T_{\mathscr{W}}} = X^{\phi}_{R}$. ■

The following theorem states that when the refinement algorithm terminates, the proposed solution to Problem 11.1 is complete, i.e., $X^{\phi}_{T_{\mathscr{W}}}$ is the largest satisfying region.

Theorem 11.1 *Suppose Algorithm 22 terminates. Then any trajectory of the embedding transition system $T_{\mathscr{W}}$ that produces a word $w_{X_{\mathscr{W}}} \in \mathscr{L}_{\phi}$ originates in $X^{\phi}_{T_{\mathscr{W}}}$ (Eq. 11.8).*

Proof To show that any satisfying trajectory originates in $X^{\phi}_{T_{\mathscr{W}}}$, assume by contradiction that there exists a trajectory

$$w_{X_{\mathscr{W}}} = w_{X_{\mathscr{W}}}(1), \ldots, w_{X_{\mathscr{W}}}(n)$$

such that $w_{X_{\mathscr{W}}}(1) \notin X^{\phi}_{T_{\mathscr{W}}}$ and the trajectory satisfies the specification, i.e., $w_{O_{\mathscr{W}}} = w_{O_{\mathscr{W}}}(1), \ldots, w_{O_{\mathscr{W}}}(n) \in \mathscr{L}_{pref,\phi}$ where $w_{O_{\mathscr{W}}}(i) = o_{\mathscr{W}}(w_{X_{\mathscr{W}}}(i))$ for all $i = 1, \ldots, n$. Then by Proposition 11.1, there exists an accepting run

$$w_{S_{D0}} = w_{S_{D0}}(1), \ldots, w_{S_{D0}}(n)$$

of the initial dual automaton A_{D0} such that

$$w_{X_{\mathscr{W}}}(i) \in R_{w_{S_{D0}}(i)}, \text{ for all } i = 1, \ldots, n.$$

$w_{X_{\mathscr{W}}}(1) \notin X^{\phi}_{T_{\mathscr{W}}}$ indicates that a state s with $w_{X_{\mathscr{W}}}(1) \in R_s$ obtained by partitioning $w_{S_{D0}}(1)$ is removed during the pruning step. Assume that the pruning step is not employed. Then run $w_{S_{D0}}$ induces a unique refined dual automaton run

$$w_{S_{DR}} = w_{S_{DR}}(1), \ldots, w_{S_{DR}}(n),$$

where $w_{S_{DR}}(i)$ and $w_{S_{D0}}(i)$ coincide or $w_{S_{DR}}(i)$ is obtained from $w_{S_{D0}}(i)$ through partitioning and $w_{X_{\mathscr{W}}}(i) \in R_{w_{S_{DR}}(i)} \subseteq R_{w_{S_{D0}}(i)}$ for all $i = 1, \ldots, n$.

Let $w'_{S_{DR}} = w'_{S_{DR}}(n_1), \ldots, w'_{S_{DR}}(n_{n'})$ be obtained by eliminating consecutive duplicates in $w_{S_{DR}}$, i.e., for each $i = 1, \ldots, n'$:

$$n_i \leq i, \text{ and } w_{X_{\mathscr{W}}}(k) \in R_{w'_{S_{DR}}(n_i)}, \text{ for each } k = n_i, \ldots, i-1.$$

Then, $w_{X_{\mathscr{W}}}(1) \notin X^{\phi}_{T_{\mathscr{W}}}$ indicates that $\mathbf{W}^P(w'_{S_{DR}}(n_1)) = \infty$. Hence, either

$$Post(R_{w'_{S_{DR}}(n_i)}) \cap R_{w'_{S_{DR}}(n_{i+1})} = \emptyset \text{ or } \mathbf{W}(w'_{S_{DR}}(n_i), w'_{S_{DR}}(n_{i+1})) = \infty$$

for some $i = 1, \ldots, n' - 1$. Let n_i be the maximal index where either one of the equalities given above holds. Therefore, $\mathbf{W}(w'_{S_{DR}}(n_k), w'_{S_{DR}}(n_{k+1})) < \infty$ for all $k = i+1, \ldots, n'-1$ and $\mathbf{W}^P(w'_{S_{DR}}(n_k)) < \infty$ for all $k = i+1, \ldots, n'$. As Algorithm 22 terminates, it holds that

$$Post(R_s) \bigcap \left\{ \bigcup_{\mathbf{W}^P(s')<\infty,\, (s,s')\in \Delta_{DR}} R_{s'} \right\} = \emptyset \tag{11.9}$$

for all $s \in S_{DR}$ with $\mathbf{W}^P(s) = \infty$. Consequently, $Post(R_{w'_{S_{DR}}(n_i)}) \cap R_{w'_{S_{DR}}(n_{i+1})} = \emptyset$, there is no control $\sigma \in \Sigma_{\mathscr{W}}$ that satisfies $w_{X_{\mathscr{W}}}(n_{i+1}) = \delta_{\mathscr{W}}(w_{X_{\mathscr{W}}}(n_{i+1} - 1), \sigma)$. Therefore $w_{X_{\mathscr{W}}}$ is not a trajectory of $T_{\mathscr{W}}$ and thus, we reached a contradiction. ■

Example 11.5 We consider the double integrator dynamics with sampling time of 1 second. We assume that the controls are constrained to set $\mathbf{U} = [-2, 2]$, and the states are constrained to set $\mathbf{X} = [-10, 2]^2$. There are 13 regions of interest $\mathbf{X}_l$, $l \in L$, $L = \{1, 2, \ldots, 13\}$, which are shown in Fig. 11.9a (see also Example 2.4). This system can be written as a two—dimensional fixed-parameter PWA control system $\mathscr{W}$ (Definition 6.2):

$$A_{1,\ldots,13} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad B_{1,\ldots,13} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, c_{1,\ldots,13} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{11.10}$$

We consider the specification described in Example 2.4, which is to visit regions $\mathbf{X}_2$ or $\mathbf{X}_9$ and then the target region $\mathbf{X}_7$, while avoiding $\mathbf{X}_{11}$ and $\mathbf{X}_{12}$, and staying inside $\mathbf{X}$ until the target region is reached. This specification translates to the following scLTL formula over $L \cup \{\text{Out}\}$:

$$\phi = ((\neg 11 \ \wedge \ \neg 12 \ \wedge \ \neg \text{Out}) \ \mathcal{U} \ 7) \wedge (\neg 7 \ \mathcal{U} \ (2 \ \vee \ 9))$$

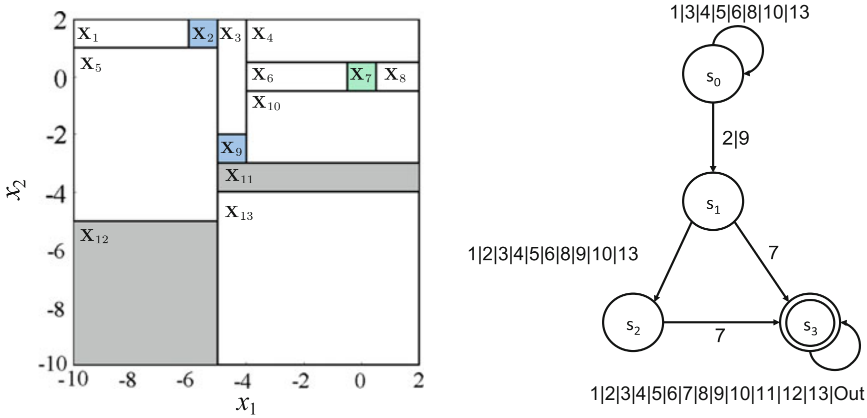
We apply Algorithm 20 to find the largest set of satisfying initial states of the system, and the corresponding control strategy. The FSA A that accepts $\mathscr{L}_{pref,\phi}$ is shown in Fig. 11.9b. The initial dual automaton A_D constructed according to Definition 11.1 from FSA A has 21 states and 194 transitions; 116 of the transitions are removed via the pruning algorithm applied in line 4 of Algorithm 20. The refinement algorithm (Algorithm 22) terminates after 108 iterations, thus the largest set of satisfying initial states $T^{\phi}_{\mathscr{W}}$ is found. We define a control automaton C, which has 116 states, according to the successor function $\Gamma_{SP}(\cdot)$ defined in Eq. (11.7). When the control strategy $(T^{\phi}_{\mathscr{W}}, \Omega)$ defined by C is used, all trajectories originating from $T^{\phi}_{\mathscr{W}}$ satisfy the specification within 15 time steps. The set of satisfying initial states found at iterations $k = 20, 50, 80$ of the refinement algorithm, the set $T^{\phi}_{\mathscr{W}}$, and some satisfying trajectories of the closed loop system are shown in Fig. 11.10. In this example, the transition controllers and the corresponding transition weights are computed using the vertex interpolation method presented in Appendix A.6.2.

Example 11.6 We consider the PWA system with 36 polytopes defined in Example 9.3 with a slight modification of the control set, $\mathbf{U} = \{u \in \mathbb{R}^2 \mid -2.5 \leq u_i \leq 2.5, i = 1, 2\}$.

The specification considered in Example 9.3 was to reach $\mathbf{X}_{10}$ and remain in there forever, while staying in $\mathbf{X}$ and avoiding regions $\mathbf{X}_{17}$, $\mathbf{X}_{18}$, $\mathbf{X}_{19}$ and $\mathbf{X}_{20}$. Note that the satisfaction of "*always remain in* $\mathbf{X}_{10}$" can not be guaranteed in finite time. Therefore, it can not be expressed in scLTL. Here, we consider a slightly different specification expressed as the following scLTL formula

$$\phi = (\neg 17 \ \wedge \ \neg 18 \ \wedge \ \neg 19 \ \wedge \ \neg 20 \ \wedge \ \neg Out) \ \mathcal{U} \ 10.$$

The trajectories satisfying formula ϕ reaches $\mathbf{X}_{10}$ while staying in $\mathbf{X}$ and avoiding regions $\mathbf{X}_{17}$, $\mathbf{X}_{18}$, $\mathbf{X}_{19}$ and $\mathbf{X}_{20}$. The operating regions of the PWA system are shown in Fig. 9.4.

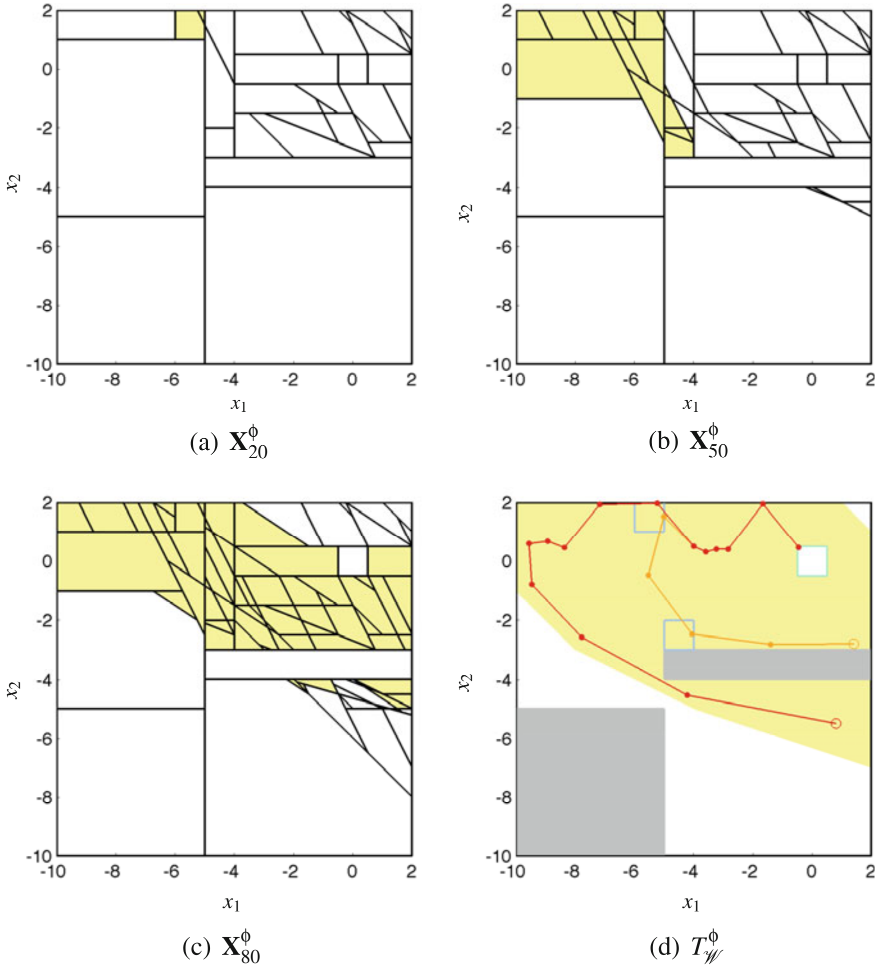


(a) Regions of $\mathscr{W}$

(b) $\phi = ((\neg 11 \wedge \neg 12 \wedge \neg \text{Out}) \ \mathcal{U} \ 7) \wedge (\neg 7 \ \mathcal{U} \ (2 \vee 9))$

Fig. 11.9 The regions of the PWA control system from Example 11.5 (**a**) and the FSA constructed from ϕ (**b**). For the FSA, s_0 is the initial state and s_3 is the accepting state. As before, if several transitions exist between two states, only one transition labeled by the set of all inputs (separated by the symbol |) is shown

We apply Algorithm 20 to find the largest set of satisfying initial states of the system, and the corresponding control strategy. The FSA A that accepts $\mathscr{L}_{pref,\phi}$ has two states and three transitions. The initial dual automaton A_D constructed according to Definition 11.1 from FSA A has 32 states and 961 transitions (each of the states correspond to an operating region of $\mathscr{W}$, and the states associated with $\mathbf{X}_{17}, \mathbf{X}_{18}, \mathbf{X}_{19}$ and $\mathbf{X}_{20}$ are not represented); 796 of the transitions are removed via the pruning algorithm applied in line 4 of Algorithm 20. We apply two controller synthesis methods presented in A.6,



(a) $\mathbf{X}_{20}^{\phi}$

(b) $\mathbf{X}_{50}^{\phi}$

(c) $\mathbf{X}_{80}^{\phi}$

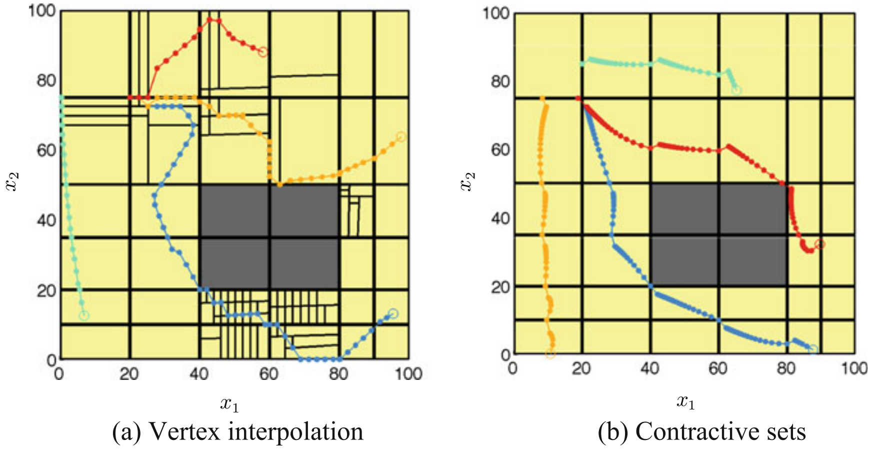
(d) $T_{\mathscr{W}}^{\phi}$

Fig. 11.10 The set of satisfying initial states found at iterations $k = 20, 50, 80, 108$ are shown in *yellow*. Sample trajectories of the closed loop system are shown in (**d**), where the initial states are marked by *circles*. The trajectories coincide in the last 6 steps before they reach the target region

namely vertex interpolation (see A.6.2) and polyhedral LFs (see A.6.3), to synthesize transition controllers, i.e., to solve Problem 11.2. In both cases, we used the successor function $\Gamma_{SP}(\cdot)$ defined in Eq. 11.7 to define control automata from the corresponding refined dual automata.

When the vertex interpolation method is used with a maximal transition cost 5 ($S = 5$, see A.6.2), the refinement algorithm terminates after 75 iterations. The refined dual automata has 107 states and 418 transitions. The corresponding control automaton guarantees that any trajectory originating in $T^{\phi}_{\mathscr{W}} = \mathbf{X} \setminus \cup_{i=17,18,19,20}\mathbf{X}_i$ satisfies the specification within 54 time steps. When the contractive sets method is employed, the dual automaton is not refined since all the states have finite costs. Hence, the corresponding control automaton has 32 states. In this case any trajectory originating in $T^{\phi}_{\mathscr{W}}$ satisfies the specification within 175 steps. Note that the sets of satisfying initial states found in both of the cases are the same, since the termination of the refinement algorithm guarantees that the resulting set $\mathbf{X}^{\phi}_{R}$ (see (11.4)) is the largest set of satisfying initial states. The regions of the refined dual automata and some trajectories of the closed loop systems are shown in Fig. 11.11.

Although a smaller control set is used in this example compared to the example from Example 9.3, the set of satisfying initial states found in Example 9.3 is smaller than $\mathbf{X}^{\phi}_{R}$. This is due to the refinement procedure and the differences between control strategies, i.e., a constant control input is assigned to each partition set in Example 9.3, while, here, a feedback control law is synthesized for each transition.



(a) Vertex interpolation (b) Contractive sets

Fig. 11.11 The sets of satisfying initial states are shown in *yellow*. The regions of the refined dual automata and sample trajectories of the closed loop systems described in Example 11.6 are shown in (**a**) and (**b**). The initial states of the trajectories are marked by *circles*

11.4 Notes

This chapter is based on our work [75, 76]. Related approaches include [165, 184], in which an abstraction is first constructed through the design of polytope-to-polytope feedback controllers, and then controlled by solving a temporal logic game on the abstraction. In this chapter, the exploration of the state-space is "guided" by the specification. In this sense, it is related to [63, 152]. By combining the abstraction and the automaton control processes, the method proposed in this chapter avoids regions of the state-space that do not contain satisfying initial states, and is, as a result, efficient. In addition, it naturally induces an iterative refinement and enlargement of the set of initial conditions, which was not possible in Chap. 9 (see also [184]) and was not formula-guided in [165].

This chapter provides two solutions based on linear programming for solving polytope-to-polytope control problems. The first solution is based on vertex interpolation and requires iteratively solving a finite number of linear programs. The second solution is based on contractive sets. While more conservative, it only requires solving a single linear program. Related approaches to solving polytope-to-polytope control problems for discrete-time systems are based on iterative computations of one-step controllable sets, e.g., [147]. For continuous-time systems the problem of controlling a linear system from one polytope to another is defined as a facet reachability problem [36, 80]. Similar to the proposed vertex interpolation method, to solve the facet reachability problem, first controls for the vertices are computed via linear programming, and then these controls are used to define a state feedback control law. While facet reachability is enforced by a flow constraint in the linear program [80], in the vertex interpolation method reachability of the polytope is enforced by direct constraints on the trajectories originating at the vertices.

This chapter can also be seen in the context of results on obstacle avoidance [33, 147, 148]. It provides a systematic way to explore the feasible state-space from "rich" temporal logic specifications that are not limited to going to a target while avoiding a set of obstacles. Furthermore, it does not necessarily involve paths characterized by unions of overlapping polytopes and the existence of artificial closed-loop equilibria. As a byproduct, the approach developed in this chapter provides an upper bound for the time necessary to satisfy the temporal logic specifications by all the trajectories originating from the constructed set of initial states.

The approach presented in this chapter can be extended from scLTL to more general LTL specifications. The main challenge is to extend the language-guided approach from the acceptance condition of an FSA to the more complicated acceptance condition of a Büchi automaton. The extension involves the solution of a backward reachability problem, which enforces the Büchi acceptance condition. Briefly, any run accepted by a Büchi automaton has a prefix-suffix structure [158]. The suffix is an infinite Büchi automaton run that periodically visits some accepting state. Starting from this observation, the scLTL approach can be extended to full LTL in two steps. First, the refinement algorithm can be used to find a cycle originating from an accepting state of the Büchi automaton. When a cycle is found, the algorithm can be used to find the set of states that can reach the corresponding accepting state. A limitation of this extension is that completeness cannot be guaranteed. Since only a single cycle is used, the initial states that can reach another cycle are not taken into consideration. The proposed solution provides the maximum time to reach the accepting state from the cycle (for prefix), and the maximum time between consecutive visits of an accepting state (for suffix).

The complexity of the proposed language-guided approach is characterized with respect to the size of the specification formula and the number of iterations of the refinement algorithm. The construction of an FSA from an scLTL formula ϕ leads to a double exponential blowup [111], therefore the initial dual automaton has $2^{2^{\mathcal{O}(|\phi|)}}$ transitions. However, this theoretical bound is not usually achieved in practice [117]. Moreover, the simplification due to the considered observation set (O instead of 2^O), and the pruning algorithm reduce the automaton size. In Algorithm 21, one step reachable sets are computed for each state to check the feasibility of transitions, which requires $|S_D| + |\Delta_D|$ basic polyhedral operations. Then, the feasibility of the states are checked by traversing the underlying graph, which is polynomial in $|S_D|$.

The complexity of Algorithm 22 is dictated by the number of iterations $R \in \mathbb{Z}_+$, which depends on the control system, the specification, and the employed region-to-region controller synthesis method. Initially, transition ($\mathbf{W}$) and state ($\mathbf{W}^P$) costs are computed via *FeasibilityCheck* and Dijkstra's algorithm, respectively. The run time of Dijkstra's algorithm is $\mathcal{O}(|S_{D0}|^2)$. The *FeasibilityCheck* procedure involves basic polyhedral operations and solving region-to-region controller synthesis problems (Problems 11.3 and 11.4). Linear programming based solutions to these problems are given in the Appendix. At each iteration of Algorithm 22, a candidate state is partitioned into two states, and the *FeasibilityCheck* procedure is used to compute the costs of the new transitions. Let d be the maximum number of transitions incident to a state $s \in S_{Di}$ for all iterations $i = 0, \ldots, R$. Then, the total number of *FeasibilityCheck* procedure runs is upper bounded by $|\Delta_{D0}| + 2dR$. Note that, at each iteration it is sufficient to compute costs ($\mathbf{W}^P$) for the new states and the states that can reach these states.

The computational framework presented in this chapter was implemented as a Matlab software package, called LanGuiCS (Language Guided Control Synthesis), which is freely downloadable from http://sites.bu.edu/hyness/languics/. The toolbox takes as input an scLTL formula over a set of linear predicates, the matrices of a PWA system, the control constraint sets, and the operating regions. It outputs the maximal set of initial states and feedback controllers such that the trajectories of the closed loop system originating from the set of initial states satisfy the formula. The tool uses scheck2 [117] to construct a finite state automaton from an scLTL formula and the MPT toolbox [113] for polyhedral operations. Both the interpolation and the set contraction approaches are implemented to compute the polytope-to-polytope controllers. The toolbox allows for displaying the set of initial states and for simulating the trajectories of the closed-loop system for 2D and 3D examples.

Chapter 12
Optimal Temporal Logic Control

In this chapter, we focus on synthesis of an optimal control strategy for a PWA system constrained to satisfy a temporal logic specification. The specification is a formula of syntactically co-safe Linear Temporal Logic (scLTL). The cost is a quadratic function that penalizes the distance from desired state and control trajectories, which are called reference trajectories. To incorporate dynamic environments, we assume that the reference trajectories are only available over a finite horizon. The goal is to find a control strategy such that the trajectory of the closed-loop system originating from a given initial state satisfies the formula and minimizes the cost. We treat the temporal logic specifications as constraints in an optimal control problem and propose a model predictive control (MPC) solution, as the natural approach for such constrained problems.

Specifically, we consider fixed-parameter PWA control systems $\mathscr{W}$ (Definition 6.2). The specifications are scLTL formulas over arbitrary linear predicates in the state of the system. The optimization objective is to minimize a quadratic cost penalizing the distance from reference state and control trajectories denoted by $x^r(0), x^r(1) \ldots$ and $u^r(0), u^r(1), \ldots$, respectively. The stage cost at time k is defined with respect to $x^r(k)$ and $u^r(k)$ by $\mathrm{L} : \mathbf{X} \times \mathbf{U} \to \mathbb{R}_+$:

$$\begin{aligned}\mathrm{L}(x(k), u(k)) = {} & (x(k) - x^r(k))^\top \mathrm{Q}(x(k) - x^r(k)) \\ & + (u(k) - u^r(k))^\top \mathrm{R}(u(k) - u^r(k)),\end{aligned} \tag{12.1}$$

where $\mathrm{Q} \in \mathbb{R}^{N\times N}$ and $\mathrm{R} \in \mathbb{R}^{M\times M}$ are positive definite matrices. We assume that, for some K, at time k the reference state and control trajectories of length K are known. At time k, the cost of a finite trajectory $x(k), \ldots, x(k + \mathrm{K} - 1)$ originating at x_k and generated by the control sequence $\mathbf{u}(k) = u(k), \ldots, u(k + \mathrm{K} - 1)$ is defined as

$$\mathrm{C}(x(k), \mathbf{u}(k)) = \sum_{i=0}^{\mathrm{K}-1} \mathrm{L}(x(i + k), u(i + k)). \tag{12.2}$$

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7_12

Problem 12.1 (*scLTL Optimal Control*) Given a fixed-parameter PWA control system $\mathscr{W}$ (Definition 6.2), an initial state $x(0) \in \mathbf{X}$, and an scLTL formula ϕ over $L \cup \{\text{Out}\}$, find a feedback control strategy such that the closed-loop trajectory originating at x_0 satisfies ϕ while minimizing the cost defined in (12.2).

Remark 12.1 Note that, as before, restricting the scLTL formula over $L \cup \{\text{Out}\}$ is made for simplicity of presentation, and is not limiting. Arbitrary linear predicates can be accommodated by refining the initial partition and updating L and $\{\text{Out}\}$.

We propose a two-step solution to Problem 12.1. In the first step, by using the framework developed in Chap. 11, we construct an automaton from the specification formula and the embedding transition system $T_{\mathscr{W}}$, i.e., the refined dual automaton obtained from Algorithm 22. The states of the automaton correspond to polyhedral subsets of $X_{\mathscr{W}}$ (the state space of the embedding transition system), and any satisfying trajectory of $T_{\mathscr{W}}$ follows a sequence of polyhedral sets defined by an accepting run of the automaton. In the second step, we design an MPC controller that minimizes the cost over the available reference trajectory, while ensuring that the resulting trajectory satisfies the specification. While the automaton is constructed "offline", at each stage, the MPC controller solves an optimization problem "online" and produces the control action. The constraints of the optimization problem ensure that the produced trajectory lies within an automaton path. We propose two methods to ensure that the produced trajectory reaches a final automaton state, and therefore satisfies the specification. First, we present an MPC scheme with a terminal constraint which guarantees that the the produced trajectory makes progress towards a final state. Second, we present an MPC scheme with a terminal cost function, i.e., a distance measure to a final automaton state. The weight of the terminal cost function increases at each time step, which guarantees that the produced trajectory reaches a final automaton state.

As it will be established, the designed controllers are recursively feasible, meaning that if the MPC optimization problem is feasible for the initial state at the initial time instant, then it remains feasible until the specification is satisfied, which is guaranteed by the constraints of the MPC optimization problem. The correctness of the solution will therefore be guaranteed. As correctness is independent from the reference sequences and the cost (Eq. 12.2), these sequences can be chosen freely. For example, i) a desired control sequence and the corresponding state trajectory can be used, ii) the reference control sequence can be chosen to minimize the amount of the applied control ($u^r(i) = \mathbf{0}, \forall i \in \mathbb{Z}_+$), iii) the reference trajectory can be used to steer the trajectory towards a desired region ($x^r(i) = x_c, \forall i \in \mathbb{Z}_+$, where x_c is the center of the desired region).

12.1 Automaton Generation

As the first step of the approach proposed for solving Problem 12.1, we use the language-guided framework presented in Chap. 11. We apply Algorithm 22 to system $T_{\mathscr{W}}$ and formula ϕ. We denote the refined dual automaton by

$$A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D), \tag{12.3}$$

and the corresponding transition weight function by $\mathbf{W} : S_D \times S_D \to \mathbb{N}_+$. We denote the region of a state $s \in S_D$ by R_s. Note that the weight of a transition is computed with respect to a feedback control strategy such that all states in R_s can reach $R_{s'}$ within $\mathbf{W}(s, s')$ steps.

Remark 12.2 As our goal is to find a control strategy for a given initial state x_0, we can terminate the refinement algorithm at the ith iteration if $x_0 \in X_k^{\phi}$ (11.4). However, if we do not stop the refinement algorithm, and it terminates in finite time, then we obtain $X_{\mathscr{W}}^{\phi}$, and the method developed in this chapter provides a solution to Problem 12.1 for each initial state from the set $X_{\mathscr{W}}^{\phi}$.

12.2 Lyapunov-Type Functions for Dual Automaton

In this section, we define Lyapunov-type functions over the state spaces of the automaton (Eq. 12.3) and $T_{\mathscr{W}}$. These functions are used to enforce the satisfaction of the specification by the MPC controller.

In control theory, control Lyapunov functions (CLFs) are used to enforce closed-loop stability of an equilibrium point. An overview of Lyapunov functions is presented in Appendix A.5. To enforce the satisfaction condition of a dual automaton, we define a real positive function that we call a *potential function* (Sect. 12.2.1). We also introduce a particular type of potential functions, which we call *contractive* potential functions (Sect. 12.2.2). These potential functions resemble control Lyapunov functions. In the remainder of this section, we formally define these functions, and present candidate functions.

12.2.1 Potential Function

Definition 12.1 A function

$$V : \bigcup_{s \in S_D} \{\{s\} \times R_s\} \to \mathbb{N}_+$$

is called a *potential function* for a transition system $T = (X, \Sigma, \delta, O, o)$ and a dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$ with $R_s \subseteq X, \forall s \in S_D$, if it satisfies:

(i) $V(s, x) = 0$ for all $s \in F_D$.
(ii) For each $(s, x) \in \bigcup_{s \in S_D} \{\{s\} \times R_s\}$, if $V(s, x) \neq 0$, then there exists a control $u \in \Sigma$ such that $x' = \delta(x, u)$, $x' \in R_{s'}$, $s' \in \delta_D(s)$, and $V(s', x') < V(s, x)$.

The definition of a potential function implies that a trajectory of T originating from $x \in R_s$, for some $s \in S_{D_0}$ can satisfy the specification within $V(s, x)$ steps, i.e., follows a sequence of polyhedral sets defined by an accepting run of A_D. We define a class of potential functions that satisfies properties (i) and (ii) of Definition 12.1 by using a successor function (Definition 11.2), and a control potential function:

Definition 12.2 A function

$$V_{con} : \bigcup_{\{(s,s') \mid s' \in \delta_D(s), s \neq s'\}} \big\{\{(s,s')\} \times R_s\big\} \longrightarrow \mathbb{N}_+$$

is called a *control potential function* for a transition system $T = (X, \Sigma, \delta, O, o)$ and a dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$ with $R_s \subseteq X, \forall s \in S_D$ and transition weight function $\mathbf{W} : S_D \times S_D \to \mathbb{N}_+$, if it satisfies:

(i) $V_{con}((s,s'),x) \leq \mathbf{W}(s,s')$.
(ii) If $V_{con}((s,s'),x) \neq \infty$, then there exists a control $u \in \Sigma$ such that either $\delta(x,u) \in R_{s'}$ or $\delta(x,u) \in R_s$ and $V_{con}((s,s'),\delta(x,u)) < V_{con}((s,s'),x)$.

The definition of a control potential function implies that for a transition of dual automaton $s' \in \delta_D(s)$, a trajectory of $T_{\mathscr{W}}$ originating from $x \in R_s$ can reach $R_{s'}$ within $V_{con}((s,s'),x)$ steps, while it stays in R_s until it reaches $R_{s'}$. We define a set $\mathscr{R}^{k,ss'}_{V_{con}} \subseteq R_s, k \in \mathbb{N}_+$, for a given control potential function V_{con} and a transition $s' = \delta_D(s)$ as

$$\mathscr{R}^{k,ss'}_{V_{con}} = \{x \in R_s \mid V_{con}((s,s'),x) \leq k\}. \tag{12.4}$$

We assume that the set from Eq. 12.4 is described by unions of polytopes, which will be instrumental for the MPC controller design. We present candidate control potential functions that satisfy properties of Definition 12.2 in Appendix A.7. In particular, we define a control potential function based on one step controllable sets, i.e., *Pre* operator, and control potential functions based on the feedback controllers that are used to compute the transition weight function $\mathbf{W}$ during the automaton refinement step, i.e., feedback controllers solving Problem 11.2. For each of the candidate control potential function V_{con} given in Appendix A.7, the set $\mathscr{R}^{k,ss'}_{V_{con}}$ is defined as a polytope or union of polytopes.

We define the *potential* at (s,x) for given successor function Γ (see Definition 11.2), control potential function V_{con}, and automaton potential function $\mathbf{W}_\Gamma$ (see Definition 11.2) as

$$V_\Gamma(s,x) = \begin{cases} 0 & \text{if } s \in F_D, \\ V_{con}((s,\Gamma(s)),x) + \mathbf{W}_\Gamma(\Gamma(s)) & \text{otherwise.} \end{cases} \tag{12.5}$$

Informally, the candidate potential function (12.5) at (s,x), $s \in S_D, x \in R_s$ is defined as an upper bound for the time required to reach $R_{\Gamma^d(s)}$ from x by applying the corresponding polytope-to-polytope feedback controllers along an automaton path defined by the successor function $s\Gamma(s)\ldots\Gamma^d(s)$, where $d \in \mathbb{N}_+$ and $\Gamma^d(s)$ is a final state of the automaton.

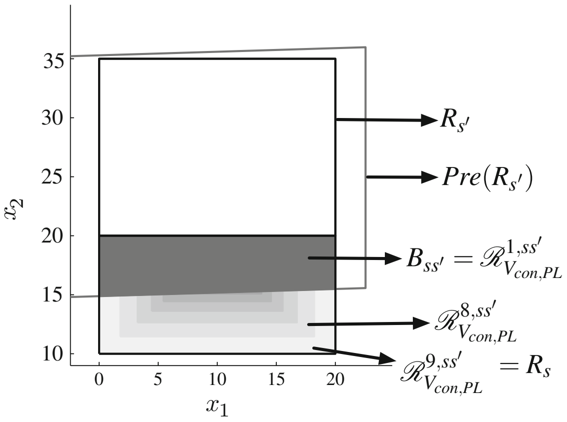


Fig. 12.1 The constraints set for a transition $s \in \delta_D(s')$. The borders of the regions R_s and $R_{s'}$ are shown with *thick black lines*. $Pre(R_{s'})$ and the beacon $B_{ss'}$ of the transition are labeled. The constraint sets are highlighted with shades of *grey*, and two of them are labeled. Note that $\mathscr{R}^{k-1,ss'}_{V_{con,PL}} \subseteq \mathscr{R}^{k,ss'}_{V_{con,PL}}$ for $k = 1, \ldots, 9$

Example 12.1 We consider the PWA system and the refined dual automaton from Example 11.6. We define a control potential function $V_{con,PL}$ based on the feedback controllers synthesized by using the polyhedral LFs method during the automaton refinement. The definitions of the functions and the corresponding constraint sets $\mathscr{R}^{k,ss'}_{V_{con,PL}}$ can be found in A.7.2.2. The constraint sets for a transition $s' \in \delta_D(s)$ with $\mathbf{W}(s, s') = 9$ are shown in Fig. 12.1. Each constraint set of control potential function $V_{con,PL}$ is represented as a union of two polytopes, i.e., the beacon $B_{ss'}$ and a sublevel set of a Lyapunov function.

Proposition 12.1 *The function defined in (12.5) is a potential function according to Definition 12.1.*

Proof Property *(i)* of Definition 12.1 is satisfied trivially by the function V_Γ (12.5). To prove that the function satisfies Defintion 12.1-*(ii)*, we consider two cases: $V_{con}((s, \Gamma(s)), x) = 1$ and $V_{con}((s, \Gamma(s)), x) > 1$.

Note that $V_{con}((s, \Gamma(s)), x) = 1$ and property Defintion 12.2-*(ii)* imply that there exists $u \in \Sigma$ such that $\delta(x, u) \in R_{\Gamma(s)}$. As such, in the case $V_{con}((s, \Gamma(s)), x) = 1$, the claim holds as

$$V_\Gamma(\Gamma(s), x') \leq \mathbf{W}_\Gamma(\Gamma(s)) \text{ for all } x' \in R_{\Gamma(s)}.$$

In the case when $V_{con}((s, \Gamma(s)), x) > 1$, from Defintion 11.2-*(i)* and Defintion 12.2-*(i)*, it holds that $V_{con}((s, \Gamma(s)), x) \neq \infty$. By property Defintion 12.2-*(ii)*, there exists $u \in \Sigma$ such that one of the following holds:

(a) $\delta(x, u) \in R_{\Gamma(s)}$,
(b) $\delta(x, u) \in R_s$ and $V_{con}((s, \Gamma(s)), \delta(x, u)) < V_{con}((s, \Gamma(s)), x)$.

The claim is true for *(a)* by the same argument given for the $V_{con}((s, \Gamma(s)), x) = 1$ case. For *(b)*, the claim holds trivially by the definition of the function V_Γ (12.5). ■

12.2.2 Contractive Potential Function

Definition 12.3 A function $V^c : \bigcup_{s \in S_D} \{\{s\} \times R_s\} \to \mathbb{R}_+$ is called a *contractive potential function* with contraction rate $\rho \in [0, 1)$ for a transition system $T = (X, \Sigma, \delta, O, o)$ and a dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$ with $R_s \subseteq X, \forall s \in S_D$, if it satisfies:

(i) $V^c(s, x) = 0$ for all $s \in F_D$.
(ii) For each $(s, x) \in \bigcup_{s \in S_D} \{\{s\} \times R_s\}$, if $V^c(s, x) \neq 0$, then there exists a control $u \in \Sigma$ such that $x' = \delta(x, u)$, $x' \in R_{s'}$, $s' \in \delta_D(s)$, and $V^c(s', x') \leq \rho V^c(s, x)$.

We define a class of contractive potential functions that satisfy properties *(i)* and *(ii)* of Definition 12.3 by using a successor function (Definition 11.2), and the transition controllers synthesized during the refinement step. In particular, the proposed contractive potential function is based on the polyhedral LF polytope-to-beacon synthesis method presented in Appendix A.6.3. Therefore, the contractive potential function can only be used if the weight function **W** is defined according to such controllers, i.e., when the polyhedral LF method is used to solve Problem 11.4 during the automaton refinement step.

First, we define a function ${V_{con}}^{LF} : \bigcup_{\{(s,s') | s' \in \delta_D(s), s \neq s'\}} \big\{\{(s, s')\} \times R_s\big\} \longrightarrow \mathbb{R}_+$ with respect to the feedback control laws synthesized using the polyhedral LFs method. Consider a transition $s' \in \delta_D(s)$, and let a feedback control law solving Problem 11.4 for R_s and $B_{ss'}$ be synthesized by using the polyhedral LFs method. Furthermore, let $\rho_{ss'}$ and $x_{ss'}$ be the corresponding contraction rate and equilibrium point, respectively (see A.6.3 for details). We define the potential of a state $x \in R_s$ with respect to the transition $s' \in \delta_D(s)$ as

$$ {V_{con}}^{LF}((s, s'), x) = (\mathbf{W}(q, q') - 1)\mathscr{M}(x) + 1, \tag{12.6} $$

where $\mathscr{M}$ is as defined in Eq. (A.23) with respect to R_s and $x_{ss'}$ (R_s is the source region **S** and $x_{ss'}$ is the equilibrium point x^s in Appendix A.6.3).

Finally, we define the *potential* at (s, x) for a given successor function Γ and automaton potential function $\mathbf{W}_\Gamma$ as

$$ V^c(s, x) = \begin{cases} 0 & \text{if } s \in F_D, \\ {V_{con}}^{LF}((s, \Gamma(s)), x) + \mathbf{W}_\Gamma(\Gamma(s)) & \text{otherwise.} \end{cases} \tag{12.7} $$

Proposition 12.2 *According to Definition 12.3, the function defined in (12.7) is a contractive potential function with contraction rate*

$$\rho = \max \left\{ \max_{s \in S_D} \frac{\mathbf{W}_\Gamma(s)}{\mathbf{W}_\Gamma(s)+1}, \max_{s \in S_D \setminus F_D} \frac{w(s)\rho_{s\Gamma(s)}^{w(s)+1} + 1 + \mathbf{W}_\Gamma(\Gamma(s))}{w(s)\rho_{s\Gamma(s)}^{w(s)} + 1 + \mathbf{W}_\Gamma(\Gamma(s))} \right\}, \tag{12.8}$$

where $w(s) = \boldsymbol{W}(s, \Gamma(s)) - 1$.

Proof Property *(i)* of Definition 12.3 is satisfied trivially by the function $V^c(s, x)$ (12.7). To prove that the function satisfies property *(ii)* with contraction rate ρ (12.8), we consider two cases:

(a) ${V_{con}}^{LF}((s, \Gamma(s)), x) \leq w(s)\rho_{ss'}^{w(s)} + 1$, and
(b) ${V_{con}}^{LF}((s, \Gamma(s)), x) > w\rho_{ss}^{w(s)} + 1$.

Notice that for any $s \in S_D \setminus F_D$ and $x \in R_s$, if the case-*(a)* holds, then

$$\mathscr{M}(x) \leq \rho_{s\Gamma(s)}^{w(s)}. \tag{12.9}$$

From the solution of the R_s to $B_{s\Gamma(s)}$ control problem, it holds that

$$\rho_{s\Gamma(s)}^{w(s)}(R_s \oplus \{-x_{s\Gamma(s)}\}) \subseteq (B_{s\Gamma(s)} \oplus \{-x_{s\Gamma(s)}\}) \tag{12.10}$$

Equations (12.9) and (12.10) imply that $x \in B_{s\Gamma(s)}$. By the definition of $B_{s\Gamma(s)}$, there exists $u \in \Sigma$ such that $\delta(x, u) \in R_{\Gamma(s)}$. Moreover, by the definitions of $\mathbf{W}$ and ${V_{con}}^{LF}$, we have that ${V_{con}}^{LF}((s, \Gamma(s)), x) \geq 1$ and ${V_{con}}^{LF}((s, \Gamma(s)), x) \leq \mathbf{W}(s, \Gamma(s))$. Consequently, it holds that $\frac{V^c(\Gamma(s), \delta(x,u))}{V^c(s,x)} \leq \frac{\mathbf{W}_\Gamma(\Gamma(s))}{\mathbf{W}_\Gamma(\Gamma(s))+1}$. Hence, if the condition *(a)* is satisfied by s and x, then there exists $u \in \Sigma$ such that $\delta(x, u) \in R_{\Gamma(s)}$ and

$$V^c(\Gamma(s), \delta(x, u)) \leq \rho_a V^c(s, x), \text{ where } \rho_a = \max_{s \in S_D} \frac{\mathbf{W}_\Gamma(s)}{\mathbf{W}_\Gamma(s)+1}. \tag{12.11}$$

Assume that case *(b)* holds for some $s \in S_D$ and $x \in R_s$. We first note that if *(b)* holds, then we have $\mathbf{W}(s, \Gamma(s)) > 1$, since ${V_{con}}^{LF}((s, \Gamma(s)), x) \leq \mathbf{W}(s, \Gamma(s))$. The feedback control law g solving the R_s to $B_{s\Gamma(s)}$ guarantees that $\mathscr{M}(\delta(x, g(x))) \leq \rho_{s\Gamma(s)}\mathscr{M}(x)$. Therefore, by the definitions of the functions V^c (12.7) and ${V_{con}}^{LF}$ (12.6), we have that

$$\frac{V^c(s, \delta(x, g(x)))}{V^c(s, x)} \leq \frac{\rho_{s\Gamma(s)} w(s)\mathscr{M}(x) + 1 + \mathbf{W}_\Gamma(\Gamma(s))}{w(s)\mathscr{M}(x) + 1 + \mathbf{W}_\Gamma(\Gamma(s))} \tag{12.12}$$

The right hand side of (12.12) increases as $\mathscr{M}(x)$ decreases. If the condition *(b)* holds for $x \in R_s$, then it follows from (12.6) that $\mathscr{M}(x) > \rho_{sS(s)}^{w(s)}$. Hence,

$$\frac{V^c(s, \delta(x, g(x)))}{V^c(s, x)} \leq \frac{w(s)\rho_{s\Gamma(s)}^{w(s)+1} + 1 + \mathbf{W}_\Gamma(\Gamma(s))}{w(s)\rho_{s\Gamma(s)}^{w(s)} + 1 + \mathbf{W}_\Gamma(\Gamma(s))}. \tag{12.13}$$

Therefore, for all $s \in S_D \setminus F_D$ and $x \in R_s$, if *(b)* is satisfied,

$$V^c(s, \delta(x, g(x))) \leq \rho_b V^c(s, x), \text{ where} \rho_b = \max_{s \in S_D \setminus F_D} \frac{w(s)\rho_{s\Gamma(s)}^{w(s)+1} + 1 + \mathbf{W}_\Gamma(\Gamma(s))}{w(s)\rho_{s\Gamma(s)}^{w(s)} + 1 + \mathbf{W}_\Gamma(\Gamma(s))}. \tag{12.14}$$

Equation (12.8) implies that $\rho = \max\{\rho_a, \rho_b\}$. Notice that $\rho < 1$, since $\rho_a < 1$ and $\rho_b < 1$ by (12.11) and (12.14), respectively. From (12.11) and (12.14), we conclude that the function V^c is a ρ contractive potential function. ■

12.3 MPC Strategies

In this section, we present two Model Predictive Control (MPC) schemes to solve Problem 12.1 for a given dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$, and a transition weight function $\mathbf{W} : S_D \times S_D \longrightarrow \mathbb{N}_+$. In both cases, we formulate MPC optimization problems over $\bigcup_{s \in S_D} \{\{s\} \times R_s\}$ to be solved "online" at each time step. The constraints of the optimization problems guarantee that the resulting closed-loop trajectory of the embedding transition system $T_{\mathscr{W}}$ follows an automaton path. To guarantee that the trajectory reaches a final automaton state, we propose terminal constraint and terminal cost techniques, which are implemented by using the potential functions defined in Sect. 12.2.

Definition 12.4 An automaton-enabled finite trajectory

$$\mathbf{T} = (s(0), x(0)) \ldots (s(\mathrm{K}), x(\mathrm{K}))$$

is a sequence of automaton (Eq. 12.3) and embedding transition system $T_{\mathscr{W}}$ state pairs such that

(i) for each $k = 0, \ldots, \mathrm{K} - 1$ there exists $u \in \Sigma_{\mathscr{W}}$ such that $x(k+1) = \delta_{\mathscr{W}}(x(k), u)$,
(ii) $x(k) \in R_{s(k)}$, for all $k = 0, \ldots, \mathrm{K}$,
(iii) $s(k+1) \in \delta_D(s(k))$, for all $k = 0, \ldots, \mathrm{K} - 1$.

The definition of an automaton-enabled trajectory implies that the projection $\gamma_{S_D}(\mathbf{T}) = s(0) \ldots s(\mathrm{K})$ of the trajectory onto the automaton states is an automaton path and the projection $\gamma_{X_{\mathscr{W}}}(\mathbf{T}) = x(0) \ldots x(\mathrm{K})$ onto the transition system states is a trajectory of $T_{\mathscr{W}}$ that follows the sequence of polyhedra defined by the automaton path.

The construction of the dual automaton A_D from Chap. 11 and Definition 12.4 guarantees that for any satisfying trajectory $\mathbf{x} = x(0) \ldots x(d), d \in \mathbb{N}$ of system $T_{\mathscr{W}}$ originating at $x(0)$, there exists an automaton-enabled trajectory $\mathbf{T}$ such that $\gamma_{X_{\mathscr{W}}}(\mathbf{T}) = \mathbf{x}$ and $\gamma_{S_D}(\mathbf{T})$ is an accepting run of A_D. Therefore, in the MPC controller design, we restrict our attention to the control sequences that generate automaton-enabled trajectories. We use $\mathbf{U}_\mathrm{K}(s, x)$ to denote the set of all control sequences of length K that produce automaton-enabled trajectories starting from (s, x) as characterized in Definition 12.4. By following the standard MPC notation, we use

$$\mathbf{T}_k = (s(0|k), x(0|k)) \ldots (s(\mathrm{K}|k), x(\mathrm{K}|k)),$$

to denote a *predicted* automaton-enabled trajectory originating at $(s(k), x(k))$, i.e., $s(0|k) = s(k), x(0|k) = x(k)$, at time $k \in \mathbb{N}$.

Consider the following optimization problem over $\mathbf{U}_{\mathrm{K}}(s(k), x(k))$:

$$\min_{\mathbf{u}(k)=u(0|k),\ldots,s(\mathrm{K}-1|k)\in\mathbf{U}_{\mathrm{K}}(s(k),x(k))} C(x(k), \mathbf{u}(k))$$

$$\text{subject to} \quad x(i+1|k) = \delta(x(i|k), u(i|k)), \quad i = 0, \ldots, \mathrm{K}-1 \tag{12.15a}$$

$$x(0|k) = x(k). \tag{12.15b}$$

We denote the optimal automaton-enabled trajectory

$$\mathbf{T}^*(k) = (s^*(0|k), x^*(0|k)) \ldots (s^*(\mathrm{K}|k), x^*(\mathrm{K}|k))$$

generated by the optimal control sequence $\mathbf{u}^*(k) \in \mathbf{U}_{\mathrm{K}}(s(k), x(k))$, until a final automaton state is reached, i.e., $s(k) \in F_D$. As the first control of the optimal control sequence is applied, we have:

$$x(k+1) = x^*(1|k), \qquad s(k+1) = s^*(1|k), \qquad k = 0, 1, 2, \ldots. \tag{12.16}$$

The constraints of the optimization problem (12.15) guarantee that the controlled trajectory follows an automaton path. However, the satisfaction of the specification is not guaranteed since the controlled trajectory might not reach a final automaton state. For example, assume that $s \in \delta_D(s)$, $x = \delta_{\mathscr{W}}(x, u)$ for some $x \in R_s$, $x^r(i) = x$ and $u^r(i) = u$ for all i. In this particular case, if the controlled trajectory reaches (s, x) at time k, then it remains there as $u^*(0|\bar{k}) \ldots u(\mathrm{K}-1|\bar{k})$ with $u(i|\bar{k}) = u$, $i = 0, \ldots, \mathrm{K}-1$ is the optimal solution of optimization problem (12.15) for all future time steps $\bar{k} > k$. To guarantee the satisfaction of the specification, we propose two techniques. First, we introduce a progress constraint over the terminal state of the predicted trajectory $(s(\mathrm{K}|k), x(\mathrm{K}|k))$ by using the potential functions presented in Sect. 12.2. Second, we define the objective of the optimization as the weighted sum of the cost given in (12.2), and the cost of the terminal state $(s(\mathrm{K}|k), x(\mathrm{K}|k))$ defined by the contractive potential function presented in Sect. 12.2.

12.3.1 MPC with Terminal Constraints

Suppose that V_{con} is a control potential function (Definition 12.2), and $\underline{\Gamma}$, $\overline{\Gamma}$ are successor functions (Definition 11.2) such that the corresponding automaton potential functions $\mathbf{W}_{\underline{\Gamma}}$ and $\mathbf{W}_{\overline{\Gamma}}$ (Definition 11.3) satisfy:

$$\mathbf{W}_{\underline{\Gamma}}(s) \leq \mathbf{W}_{\overline{\Gamma}}(s), \text{ for all } s \in S_D. \tag{12.17}$$

Furthermore, let $V_{\underline{\Gamma}}$ be the potential function defined by $\underline{\Gamma}(\cdot)$ and V_{con} as given in (12.5). Given these functions, the MPC optimization problem is formulated as follows:

$$\min_{\mathbf{u}(k)=u(k|0),\dots,s(K-1|k)\in\mathbf{U}_{\mathrm{K}}(s(k),x(k))} C(x(k),\mathbf{u}(k))$$

$$\text{subject to} \quad V_{\underline{\Gamma}}(s(\mathrm{K}|k),x(\mathrm{K}|k)) < v(k), \tag{12.18a}$$

$$x(i+1|k) = \delta(x(i|k),u(i|k)), \quad i=0,\dots,\mathrm{K}-1, \tag{12.18b}$$

where $x(i \mid k) = x(k)$, $v(k) \in \mathbb{N}$, $v(0) = \mathbf{W}_{\overline{\Gamma}}(s(0))$ and for $k \geq 1$, $v(k)$ is defined by

$$v(k) = \min\{v(k-1)-1, \mathbf{W}_{\overline{\Gamma}}(s^*(K|k-1))\}. \tag{12.19}$$

For $k \geq 1$, the optimal predicted trajectory obtained at the previous time step $(s^*(\mathrm{K}|k-1))$ and $v(k-1)$ are used to enforce the satisfaction of the specification. Specifically, the predicted trajectory at time k must end in a state, for which there exists a control sequence guaranteeing that the trajectory originating from that state reaches a final automaton state within $v(k)$ steps by following the sequence of polyhedra defined by the successor function $\underline{\Gamma}$. It is important to note that the bound $v(k)$ is computed according to $\overline{\Gamma}$, and by (12.17) and (12.5), $\mathbf{W}_{\overline{\Gamma}}(s^*(\mathrm{K}|k-1))$ is an upper bound on $V_{\underline{\Gamma}}(s^*(\mathrm{K}|k-1), x^*(\mathrm{K}|k-1))$. The key property of the progress constraint is that the bound of the progress constraint decreases at each time step, i.e., $v(k+1) < v(k)$. This property guarantees that the resulting trajectory eventually reaches a final automaton state. By the definition of the potential function given in (12.5), the progress constraint given in (12.18a) at time $k \geq 0$ takes the following from:

$$V_{con}((s(\mathrm{K}|k), \underline{\Gamma}(s(\mathrm{K}|k))), x(\mathrm{K}|k)) < v(k) - \mathbf{W}_{\underline{\Gamma}}(\underline{\Gamma}(s(\mathrm{K}|k))). \tag{12.20}$$

Let $\bar{k} := v(k) - 1 - \mathbf{W}_{\underline{\Gamma}}(\underline{\Gamma}(s(\mathrm{K}|k)))$. If $\bar{k} \geq \mathbf{W}(s(\mathrm{K}|k), \underline{\Gamma}(s(\mathrm{K}|k)))$, then the inequality given in (12.20) is trivially satisfied for all $x(\mathrm{K}|k) \in R_{s(\mathrm{K}|k)}$. If, however $\bar{k} < \mathbf{W}(s(\mathrm{K}|k), \underline{\Gamma}(s(\mathrm{K}|k)))$, then the inequality is satisfied only if

$$x(\mathrm{K}|k) \in \mathscr{R}_{V_{con}}^{\bar{k},s(\mathrm{K}|k)\underline{\Gamma}(s(\mathrm{K}|k))}. \tag{12.21}$$

As such, if the set $\mathscr{R}_{V_{con}}^{\bar{k},s(\mathrm{K}|k)\underline{\Gamma}(s(\mathrm{K}|k))}$ (see Eq. 12.4) is a polytope or union of polytopes, then the set of states of $T_{\mathscr{W}}$ that satisfy the terminal constraint can be represented as a union of polytopes for a given automaton state $s(\mathrm{K}|k)$.

Next, we show that the optimal solution of the MPC problem from Eq. 12.18 can be found by solving a finite number of convex optimization problems. First, consider the set of automaton paths of length K that originate from $s(k)$:

$$\begin{aligned}\mathbf{P}_{s(k)}^{\mathrm{K}} =& \{s(0|k)s(1|k)\dots s(\mathrm{K}|k) \mid s(0|k) := s(k),\\ & s(i+1|k) \in \delta_D(s(i|k)), i=0,\dots,\mathrm{K}-1\}.\end{aligned} \tag{12.22}$$

Since S_D is a finite set, $\mathbf{P}_{s(k)}^{\mathrm{K}}$ is a finite set. The definition of an automaton-enabled trajectory $\mathbf{T}_k$ of horizon K (Definition 12.4) implies that $\gamma_{S_D}(\mathbf{T}_k) \in \mathbf{P}_{s(k)}^{\mathrm{K}}$ for any trajectory that can be produced by a control sequence from the set $\mathbf{U}_{\mathrm{K}}(s(k), x(k))$.

Note that the weights of the transitions are not considered while constructing the set $\mathbf{P}^{\mathrm{K}}_{s(k)}$. A transition $s' \in \delta_D(s)$ with infinite weight means that the synthesis method failed to find a feedback control law that solves the corresponding region to region control problem. However, the problem might have a solution for subsets of R_s and $R_{s'}$.

Given a finite automaton path $\mathbf{s}(k) \in \mathbf{P}^{\mathrm{K}}_{s(k)}$, let $\mathbf{U}^{\mathbf{s}(k)}_{\mathrm{K}}(s(k), x(k))$ denote the set of all control sequences that produce an automaton-enabled trajectory $\mathbf{T}_k$ with $\gamma_{S_D}(\mathbf{T}_k) = \mathbf{s}(k)$. Essentially, $\mathbf{U}^{\mathbf{s}(k)}_{\mathrm{K}}(s(k), x(k))$ is the set of all control sequences that produce trajectories of $T_{\mathscr{W}}$ that originate at $x(k)$ and follow the sequence of polyhedra defined by $\mathbf{s}(k)$. Then, it is straightforward to see that

$$\mathbf{U}_{\mathrm{K}}(s(k), x(k)) = \bigcup_{\mathbf{s}(k) \in \mathbf{P}^{\mathrm{K}}_{s(k)}} \mathbf{U}^{\mathbf{s}(k)}_{\mathrm{K}}(s(k), x(k)). \tag{12.23}$$

Consider a path $\mathbf{s}(k) = s(0|k) \dots s(\mathrm{K}|k) \in \mathbf{P}^{\mathrm{K}}_{s(k)}$ and the following constraints in the variables $\mathbf{u}(k) = u(0|k) \dots u(\mathrm{K}-1|k)$:

$$x(i+1|k) = \delta(x(i|k), u(i|k)), \qquad i = 0, \dots, \mathrm{K}-1, \tag{12.24a}$$

$$x(i|k) \in R_{s(i|k)}, \qquad i = 1, \dots, \mathrm{K}, \tag{12.24b}$$

$$u(i|k) \in \Sigma_{\mathscr{W}}, \qquad i = 0, \dots, \mathrm{K}-1. \tag{12.24c}$$

The set of control sequences that satisfy constraints (12.24) is $\mathbf{U}^{\mathbf{s}(k)}_{\mathrm{K}}(s(k), x(k))$. Therefore, the MPC problem given in (12.18) can be restated as:

$$\begin{aligned} \min_{\mathbf{s}(k) \in \mathbf{P}^{\mathrm{K}}_{s(k)}} \min_{\mathbf{u}(k)} \quad & C(x(k), \mathbf{u}(k)) \\ \text{subject to} \quad & V_{\underline{\Gamma}}(s(\mathrm{K}|k), x(\mathrm{K}|k)) < v(k), \\ & \text{(12.24a), (12.24b), and (12.24c).} \end{aligned} \tag{12.25}$$

As the set of states $T_{\mathscr{W}}$ that satisfies the progress constraint can be represented as unions of polytopes, the optimal solution of the MPC problem (12.18) can be found by solving a set of convex optimization problems, i.e., quadratic programming (QP) problems. Note that constraint (12.24a) takes the form of a linear equality according to the PWA dynamics $(A_{\tau_D(s(i|k))}, B_{\tau_D(s(i|k))}, c_{\tau_D(s(i|k))})$ active in $R_{s(i|k)}$:

$$x(i+1|k) = A_{\tau_D(s(i|k))} x(i|k) + B_{\tau_D(s(i|k))} u(i|k) + c_{\tau_D(s(i|k))}. \tag{12.26}$$

To guarantee that the resulting closed-loop trajectory of $T_{\mathscr{W}}$ reaches a region R_{s_f}, where $s_f \in F_D$, we proceed as follows: at each time-step k the prediction horizon, denoted as I_k, is determined with respect to the predicted trajectory obtained at the previous step. Specifically, the length of the observed reference trajectory, K, is used as the initial prediction horizon I_0 at time-step $k = 0$. Then, for time-step $k \geq 1$, if the predicted trajectory obtained at the previous step visits a final state at position j for the first time, $j - 1$ is used as the prediction horizon I_k. Otherwise, the same prediction horizon as in the previous time-step, I_{k-1}, is used. The following function is used to determine the prediction horizon for a given trajectory $\mathbf{T}_k = (s(0|k), x(0|k)) \ldots (s(I_k|k), x(I_k|k))$:

$$I(\mathbf{T}_k) = \begin{cases} I_k & \text{if } s(i|k) \notin F_D, \forall i = 0, \ldots, I_k \\ j-1 & \text{if } s(i|k) \notin F_D, \forall i = 0, \ldots, j-1, \quad s(j|k) \in F_D. \end{cases} \tag{12.27}$$

Adapting the prediction horizon according to function I (12.27) allows us to optimize the cost until the specification is satisfied, i.e., until a final automaton state is reached.

Algorithm 24 `AutomatonGuidedMPC` $-$ `TCS` $(A_D, \mathbf{W}, x(0), \mathrm{K}, C, V_{con}, \overline{\Gamma}, \underline{\Gamma})$

```
Require: Dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$, transition weight function $\mathbf{W} : S_D \times S_D \to \mathbb{N}_+$, an initial condition $x(0) \in R_{s(0)}$ for some $s(0) \in S_{D_0}$, MPC horizon K, cost function $C$, and functions $V_{con}$, $\underline{\Gamma}$ and $\overline{\Gamma}$.
1: Set $k = 0$, $I_k = \mathrm{K}$, $v(0) = \mathbf{W}_{\overline{\Gamma}}(s(0))$. (Initialization)
2: while $s(k) \notin F_D$ do
3:   $OptCost = \infty$, $\mathbf{u}^*(k) = \emptyset$, $\mathbf{T}_k^* = \emptyset$.
4:   Compute $\mathbf{P}_{s(k)}^{I_k}$ .
5:   for all $\mathbf{s} = s(0|k) \ldots s(I_k|k) \in \mathbf{P}_{s(k)}^{I_k}$ do
6:     if $\mathbf{W}_{\underline{\Gamma}}(\underline{\Gamma}(s(I_k|k))) < v(k) - 1$ then
7:       $c = \min_{\mathbf{u}(k)=u(0|k)\ldots u(I_k-1|k)} C(x(k), \mathbf{u}(k))$ subject to
             (12.18a), (12.24a), (12.24b), and (12.24c)
8:       if $c < OptCost$ then
9:         $OptCost := c$, set $\mathbf{u}^*(k)$ and $\mathbf{T}_k^*$ with respect to the solution of the optimization problem (line 7).
10:      end if
11:    end if
12:  end for
13:  Apply $\mathbf{u}^*(0|k)$, set $s(k+1) := s^*(1|k)$, $x(k+1) := x^*(1|k)$.
14:  $I_{k+1} = I(\mathbf{T}_k^*)$.
15:  $v(k+1) = \mathbf{W}_{\overline{\Gamma}}(s^*(I_{k+1}|k))$.
16:  if $I_{k+1} == I_k$ then
17:    $v(k+1) = \min\{v(k) - 1, v(k+1)\}$.
18:  end if
19:  $k = k + 1$.
20: end while
```

The proposed MPC controller with a terminal state constraint is summarized in Algorithm 24, where a set of optimization problems is solved at each time step until a final automaton state is reached (line 2). At each time step, the linear quadratic optimization problem given in line 7 is solved for each automaton path $\mathbf{s} \in \mathbf{P}_{s(k)}^{I_k}$ (12.22), which satisfies the condition given in line 6, and each polytope from the set $\mathscr{R}_{V_{con}}^{\bar{k},s(S|k)\underline{\Gamma}(s(S|k))}$ (see Eq. 12.21). Note that if an automaton path does not satisfy the condition given in line 6, the problem given in line 7 becomes infeasible. When the loop over $\mathbf{P}_{s(k)}^{I_k}$ (line 5) is terminated, the first element of the optimal control sequence $\mathbf{u}^*(k)$ is applied and the state $(s(k+1), x(k+1))$ is computed. Notice that at each time step, either the prediction horizon or the bound used in the terminal constraint (12.18a) is reduced (lines 14–19).

Property 12.1 *The length of any satisfying trajectory of $T_{\mathscr{W}}$ originating at $x(0)$ is lower bounded by* K.

We assume that $T_{\mathscr{W}}$, $x(0)$ and K satisfy Property 12.1. This assumption is made to simplify the presentation of the following results. The length, $\overline{\mathrm{K}}$, of the shortest satisfying trajectory of $T_{\mathscr{W}}$ originating at $x(0)$ can be found by solving a set of optimization problems of the form given in Eq. 12.25. Then, the assumption is not necessary if $\overline{\mathrm{K}}$ is used as the initial prediction horizon in the case when $\overline{\mathrm{K}} < \mathrm{K}$.

Lemma 12.1 *Suppose that Property 12.1 holds, and there exists $s(0) \in S_{D_0}$ such that $x(0) \in R_{s(0)}$. Then, the optimization problem given in line 7 of Algorithm 24 is feasible for some $\mathbf{s}(0) \in \boldsymbol{P}^{\mathrm{K}}_{s(0)}$ at the initial condition $(s(0), x(0))$.*

Proof Definition 11.3 imply that $v(0) = \mathbf{W}_{\overline{\Gamma}}(s(0)) < \infty$. Since $V_{\underline{\Gamma}}(s(0), x(0)) \leq \mathbf{W}_{\underline{\Gamma}}(s(0))$ and $\mathbf{W}_{\underline{\Gamma}}(s(0)) \leq \mathbf{W}_{\overline{\Gamma}}(s(0))$ for all $s(0) \in S_D$, it holds that $V_{\underline{\Gamma}}(s(0), x(0)) \leq v(0)$. By Proposition 12.1 and Assumption 12.1, there exists a control sequence $\mathbf{u} = u(0) \ldots u(\mathrm{K} - 1)$ and an automaton path $\mathbf{s} = s(0) \ldots s(\mathrm{K})$ such that the value of the potential function $V_{\underline{\Gamma}}$ strictly decreases along the trajectory $\mathbf{T} = (s(0), x(0)) \ldots (s(\mathrm{K}), x(\mathrm{K}))$ generated by $\mathbf{u}$ from the initial condition $(s(0), x(0))$, and hence, $V_{\underline{\Gamma}}(s(\mathrm{K}), x(\mathrm{K})) < V_{\underline{\Gamma}}(s(0), x(0)) \leq v(0)$. As such, the optimization problem is feasible for $\mathbf{s}$. ■

If the MPC optimization problem (12.18) is feasible for the initial state at the initial time step, then it remains feasible until the specification is satisfied. In other words, the proposed MPC controller is recursively feasible, which is formally stated as:

Theorem 12.1 *Suppose that Property 12.1 holds, and there exists $s(0) \in S_{D_0}$ such that $x(0) \in R_{s(0)}$. Then:*

(i) If the optimization problem given in line 7 of Algorithm 24 is feasible for some $\mathbf{s}(k) \in \boldsymbol{P}^{I_k}_{s(k)}$ at time k for state $(s(k), x(k))$, and $s(k+1) \notin F_D$, then there exists $\mathbf{s}(k+1) \in \boldsymbol{P}^{I_{k+1}}_{s(k+1)}$ such that the problem is feasible for $\mathbf{s}(k+1)$ and state $(s(k+1), x(k+1))$.

(ii) The trajectory of $T_{\mathscr{W}}$ produced by the closed-loop system satisfies the specification.

Proof *(i)* Let $\mathbf{T}^*_k = (s^*(0|k), x^*(0|k)) \ldots (s^*(I_k|k), x^*(I_k|k))$ be the trajectory generated by the optimal control sequence $\mathbf{u}^*(k) = u^*(0|k) \ldots u^*(I_k|k)$ at step k. From (12.27), we have that $I_{k+1} \leq I_k$. By Proposition 12.1, there exists a control $u \in \mathbf{U}$ and a state $s' \in S_D$ such that $x' = \delta_{\mathscr{W}}(x(I_{k+1}|k), u) \in R_{s'}$, $s' \in \delta_D(s(I_{k+1}|k))$ and

$$V_{\underline{\Gamma}}(s', x') < V_{\underline{\Gamma}}(s^*(I_{k+1}|k), x^*(I_{k+1}|k)). \tag{12.28}$$

By (12.5) and (12.17), it holds that

$$V_{\underline{\Gamma}}(s^*(I_{k+1}|k), x^*(I_{k+1}|k)) \leq \mathbf{W}_{\underline{\Gamma}}(s^*(I_{k+1}|k)) \leq \mathbf{W}_{\overline{\Gamma}}(s^*(I_{k+1}|k)) \tag{12.29}$$

In the case when $I_k = I_{k+1}$, by constraint (12.18a), we have that $V_{\underline{\Gamma}}(s^*(I_{k+1}|k), x^*(I_{k+1}|k)) < v(k)$, and as such, by (12.28) and (12.29) it holds that

$$V_{\underline{\Gamma}}(s', x') < v(k) - 1. \tag{12.30}$$

Equations (12.28–12.30) imply that $V_{\underline{\Gamma}}(s', x') < v(k+1)$. As such, the control sequence $u^*(1|k) \ldots u^*(I_k|k), u'$ is a feasible solution of the optimization problem at step $k+1$ for $s^*(1|k) \ldots s^*(I_{k+1}|k)s'$.

(ii) We first show that the produced trajectory reaches a final automaton state in finite time. By Lemma 12.1, the optimization problem is feasible at time $k = 0$ for some $\mathbf{s}(0) \in \mathbf{P}_{s(0)}$. From Theorem 12.1-*(i)* it follows that an optimal predicted trajectory $\mathbf{T}_k^*$ exists at each time step until a final state is reached. Let $v^* = \max_{s \in S_D} \mathbf{W}_{\overline{\Gamma}}(s)$. By Definition 11.3, $v^* < \infty$. From lines 1 and 15 of Algorithm 24, it holds that $v(k) \leq v^*$. From line 17 of Algorithm 24, we have that $v(k+1) < v(k)$ until a final automaton state appears in a trajectory. The strict decrease and the progress constraint (12.18a) imply that there exists $k' \leq v(0)$ such that the trajectory $\mathbf{T}_{k'}^*$ visits a final automaton state. The optimization horizon at step $k'+1$ satisfies $I_{k'+1} < \mathrm{K}$, and $v(k'+1) = \mathbf{W}_{\overline{\Gamma}}(s^*(I_{k'+1}|k')) \leq v^*$.

By applying the same argument iteratively, we conclude that there exists a time $k'' < \mathrm{K}v^*$ such that $I_{k''} = 1$ and the predicted trajectory $\mathbf{T}_{k''}^*$ ends in a final automaton state. Therefore, the trajectory produced by the closed-loop system reaches a final automaton state within $\mathrm{K}v^*$ steps.

As shown above, the proposed MPC controller produces a finite trajectory $(s(0), x(0)) \ldots (s(l), x(l)), l \leq \mathrm{K}v^*$. Next, we show that the projected system trajectory $x(0) \ldots x(l)$ satisfies ϕ. It is assumed that $x(0) \in R_{s(0)}$ and $s(0) \in S_{D_0}$. By the definition of $\mathbf{P}_s^{I_k}$, $s(i+1) \in \delta_D(s(i))$ for all $i = 0, \ldots, l-1$ and by the termination condition $s(l) \in F_D$. Consequently, $s(0) \ldots s(l)$ is an accepting automaton run. The constraints of the optimization problem given in line 7 ensure that $x(i) \in R_{s(i)}$ for all $i = 0, \ldots, l$. Hence, the system trajectory $x(0) \ldots x(l)$ satisfies the specification. ■

Example 12.2 We consider the double integrator dynamics and the refined dual automaton from Example 11.5. We define the cost function as in (12.2) with

$$\mathrm{Q} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad \mathrm{R} = 0.2. \tag{12.31}$$

We use the MPC controller with terminal constraints to minimize the cost with respect to reference trajectories.

The successor function Γ_{SP} as defined in Equation 11.7, i.e., $\Gamma_{SP}(s)$ is the state that succeeds s in the shortest path from s to F_D, and the control potential function $V_{con,CS}$, which is defined in Appendix A.7.1, are used in Algorithm 24 for $\underline{\Gamma}$ and V_{con}. The successor function $\overline{\Gamma}$ is constructed by traversing the graph of the automaton A_D as explained in Sect. 11.3 with a slight modification of the rule given in Eq. 11.6, i.e., the states s, s', with $\mathbf{W}(s, s') < \infty$ that maximize the right hand side of Eq. 11.6 are chosen.

We apply the MPC controller on satisfying and violating reference trajectories. The reference trajectories $\mathbf{x}^{r_i}$, $i = 1, 2, 3$ are generated by the reference control sequences $\mathbf{u}^{r_i}$, $i = 1, 2, 3$ from the initial conditions $x^{r_1}(0) = \begin{bmatrix} 1 \\ -6 \end{bmatrix}$, $x^{r_2}(0) = \begin{bmatrix} 1.4 \\ -2.8 \end{bmatrix}$ and $x^{r_3}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, respectively, where

$$\begin{aligned}
\mathbf{u}^{r_1} &= 1.8, 1.4, 1.8, 1.8, 1, 0, -0.7, 0, 0, 0, 0.1, -1.2, 0.\\
\mathbf{u}^{r_2} &= 0.6, 0.6, 0.8, 0.8, 1, 0.4, 0.4, -0.6, -0.6, -0.6, -0.4, -0.6, 1.\\
\mathbf{u}^{r_3} &= -0.6, -1, -0.3, -0.3, -0.4, -0.5, 0.2, 1, 1, 1, 0.4, -0.8, -0.5, -0.5, -0.5, -0.5, 0, 0.5,\\
&\quad 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, 0.2, 0.2, -0.5, -0.5, -1.2, 0.2, 0.2667, 0.1333.
\end{aligned}$$

The reference trajectories $\mathbf{x}^{r_1}$ and $\mathbf{x}^{r_3}$ satisfy the specification, while $\mathbf{x}^{r_2}$ violates it. The reference trajectories and the corresponding simulated trajectories generated by Algorithm 24 are shown in Fig. 12.2. As shown in Fig. 12.2a, the controlled trajectory $\mathbf{x}^1$ follows $\mathbf{x}^{r_1}$ until the specification is satisfied. However, as shown in Fig. 12.2b, the controlled trajectory $\mathbf{x}^2$ follows the reference trajectory $\mathbf{x}^{r_2}$ only for the first 12 steps, then the distance between them increases as the controlled trajectory visits region B to satisfy the specification. Even though reference trajectory $\mathbf{x}^{r_3}$ satisfies the specification, as shown in Fig. 12.2e the controlled trajectories for optimization horizons $\mathrm{K} = 2$ and $\mathrm{K} = 5$ can not exactly follow $\mathbf{x}^{r_3}$ due to the progress constraint.

12.3.2 MPC with Terminal Cost

Suppose that Γ is a successor function (Definition 11.3), and V^c is the corresponding contractive control potential function (Definition 12.3) defined as in (12.7), and $\alpha \in \mathbb{R}_+$ and $\gamma \in (0, 1)$ are given constants. The MPC optimization problem with additive cost is formulated as follows:

$$\min_{\mathbf{u}(k)=u(k|0),\ldots,u(\mathrm{K}-1|k)\in \mathbf{U}_{\mathrm{K}}(s(k),x(k))} \gamma^k C(x(k), \mathbf{u}(k)) + (1-\gamma^k)\alpha V^c(s(\mathrm{K}|k), x(\mathrm{K}|k)), \tag{12.32}$$

$$\begin{aligned}
\text{subject to}\quad & x(i+1|k) = \delta(x(i|k), u(i|k)), \quad i = 0, \ldots, \mathrm{K}-1.\\
& x(0|k) = x(k).
\end{aligned}$$

The search space ($\mathbf{U}_{\mathrm{K}}(s(k), x(k))$) and the constraints of optimization problems in Eqs. (12.32) and (12.18) are the same. However, the optimization objectives are

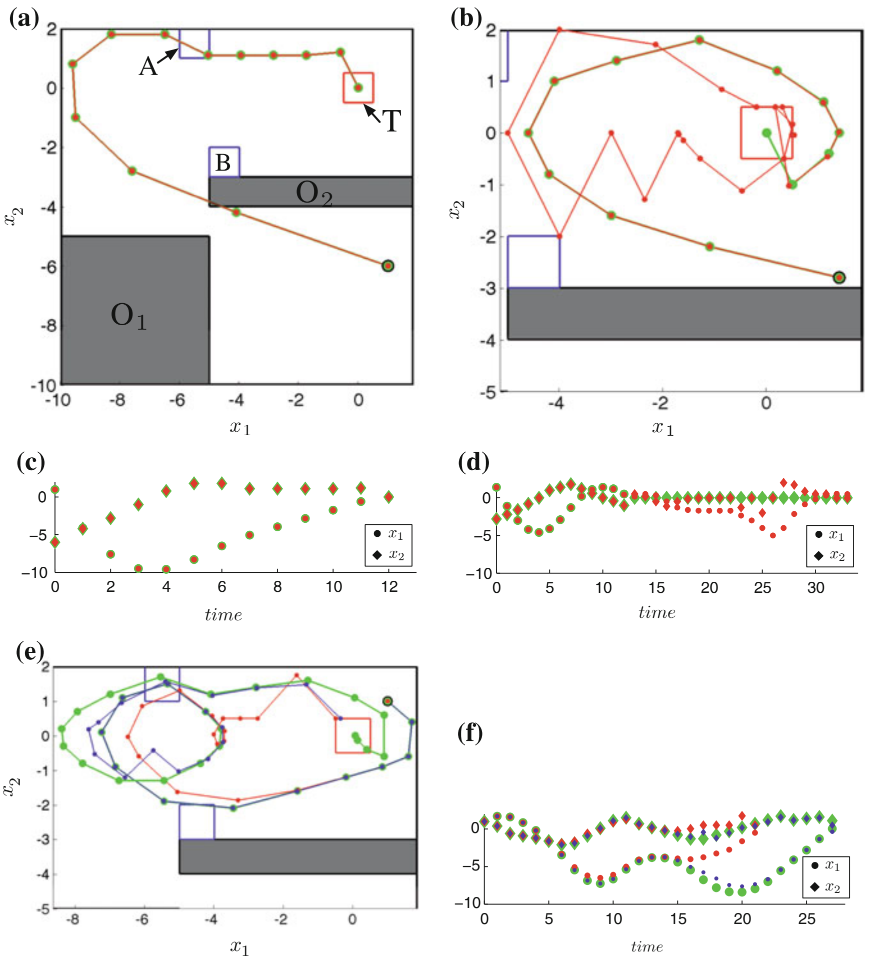


Fig. 12.2 The reference trajectory (*green*, satisfying in **a** and **e**, and violating in **b**) and the trajectory of the controlled system (*red* and *blue*). The initial states are marked by *black circles* in **a**, **b**, and **e**. The projections of the trajectories shown in (a), (b), and (e) over the first and second dimensions with respect to time are shown in **c**, **d**, and **f**, respectively. **a**, **c** K = 2, total cost = 0. **b**, **d** K = 4, total cost = 58.53. **e**, **f** K = 2, total cost = 61.78 (*red*) and K = 5, total cost = 3.44 (*blue*). The *color* codes in **a**, **b**, **e** and **c**, **d**, **f** are the same

different. In (12.32), the objective of the optimization is to minimize the cost with respect to the available reference state and control trajectories, while guaranteeing that the resulting trajectory reaches an accepting state. To enforce the latter part, the contractive potential function V^c (12.7) is used as the terminal cost. As time progresses, the weight of the terminal cost, i.e., $1 - \gamma^k$, increases, which in turn guarantees that the resulting trajectory steers towards an accepting state. The value of the potential function is scaled by a constant factor $\alpha \in \mathbb{R}_+$, since the objective is to minimize the potential and trajectory cost together. A candidate scaling factor is

$$\alpha = \frac{\mathrm{K}\max_{s\in S_D, x\in R_s} V^c(s,x)}{L^*}, \tag{12.33}$$

where L^* is the maximal value of the function L, i.e.,

$$L^* = \max_{x,x^r\in X, u, u^r\in \mathbf{U}} (x-x^r)^\top \mathrm{Q}(x-x^r) + (u-u^r)^\top \mathrm{R}(u-u^r),$$

and Q and R are as defined in the stage cost Eq. (12.1).

As shown in Sect. 12.3.1, the set of admissible control sequences $\mathbf{U}_\mathrm{K}(s(k), x(k))$ for state $(s(k), x(k))$ is the union of admissible control sequences $\mathbf{U}_\mathrm{K}^{\mathbf{s}(k)}(s(k), x(k))$ for each automaton path $\mathbf{s}(k)$ from the set $\mathbf{P}^\mathrm{K}_{s(k)}$ (see (12.22) and (12.23)). Therefore, the optimal solution of the MPC problem given in (12.32) equals to the optimal solution of:

$$\begin{aligned} \min_{\mathbf{s}(k)\in\mathbf{P}^\mathrm{K}_{s(k)}} \min_{\mathbf{u}(k)} \quad & \gamma^k C(x(k), \mathbf{u}(k)) + (1-\gamma^k)\alpha V^c(s(\mathrm{K}|k), x(\mathrm{K}|k)) \\ \text{subject to} \quad & (12.24\text{a}), (12.24\text{b}), \text{ and } (12.24\text{c}). \end{aligned} \tag{12.34}$$

Therefore, the optimal solution to the MPC problem given in Eq. (12.32) can be found by solving a QP for each $\mathbf{s}(k) \in \mathbf{P}^\mathrm{K}_{s(k)}$.

Algorithm 25 `AutomatonGuidedMPC` − `TCF` $(\mathsf{A}^D, \mathbf{W}, x(0), N, L, V^c, \alpha, \gamma)$

```
Require: Dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$, transition weight function $\mathbf{W}\colon S_D \times S_D \to \mathbb{N}_+$, an initial condition $x(0) \in R_{s(0)}$ for some $s(0) \in S_{D_0}$, MPC horizon K, cost function $C$, contractive potential function $V^c$, scaling factor $\alpha$ and discount factor $\gamma$
1: Set $k = 0, I_k = \mathrm{K}$ (Initialization)
2: while $s(k) \notin F_D$ do
3:    $OptCost = \infty$, $\mathbf{u}^*(k) = \emptyset$, $\mathbf{T}^*(k) = \emptyset$.
4:    Compute $\mathbf{P}^{I_k}_{s(k)}$ .
5:    for all $\mathbf{s} = s(0|k)\ldots s(I_k|k) \in \mathbf{P}^{I_k}_{s(k)}$ do
6:       $c = \min_{\mathbf{u}(k)=u(0|k)\ldots u(I_k-1|k)} \gamma^k C(x(k), \mathbf{u}(k)) + (1-\gamma^k)\alpha V^c(s(I_k|k), x(I_k|k))$ subject to
            (12.24a), (12.24b), and (12.24c)
7:       if $c < OptCost$ then
8:          $OptCost := c$, set $\mathbf{u}^*(k)$ and $\mathbf{T}^*_k$ with respect to the solution of the optimization problem
            (line 6)
9:       end if
10:   end for
11:   Apply $\mathbf{u}^*(0|k)$, $s(k+1) = s^*(1|k)$, $x(k+1) = x^*(1|k)$
12:   $I_{k+1} = I(\mathbf{T}^*_k)$
13:   $k = k + 1$
14: end while
```

The proposed MPC controller with a terminal cost is summarized in Algorithm 24, where a set of optimization problems is solved at each time step until a final automaton state is reached (line 2). At each time step, the linear quadratic optimization problem given in line 6 is solved for each automaton path $\mathbf{s} \in \mathbf{P}^{I_k}_{s(k)}$ (Eq. 12.22). When the loop over $\mathbf{P}^{I_k}_{s(k)}$ (line 5) is terminated, the first element of the optimal control sequence

$\mathbf{u}^*(k)$ is applied and the state $(s(k+1), x(k+1))$ is computed. As in Algorithm 24, at each time step, the prediction horizon is updated according to function I (12.27).

Next, we show that the closed-loop trajectory generated by the proposed MPC controller (Algorithm 25) satisfies the specification. As in Sect. 12.3.1, we first show that the MPC optimization problem is feasible at the first time step for the initial state, then we show that it remains feasible until the specification is satisfied.

Lemma 12.2 *Suppose that Property 12.1 holds, and there exists $s(0) \in S_{D_0}$ such that $x(0) \in R_{s(0)}$. Then, the optimization problem given in line 6 of Algorithm 25 is feasible for some $\mathbf{s}(0) \in \boldsymbol{P}^{\mathrm{K}}_{s(0)}$ at the initial condition $(s(0), x(0))$.*

Proof By construction of the dual automaton, there exists $\mathbf{u} = u(0), \ldots, u(d-1)$, $d \in \mathbb{Z}_+$ such that the trajectory $\mathbf{T} = (s(0), x(0)) \ldots (s(d), x(d))$ generated by $\mathbf{u}$ from the initial condition $(s(0), x(0))$ satisfies $s(d) \in F_D$, and hence $s(0), \ldots, s(d)$ is an accepting automaton run. By Assumption 12.1, $\mathrm{K} \leq d$. As such $\mathbf{s}(0) = s(0), \ldots, s(\mathrm{K}) \in \mathbf{P}^{\mathrm{K}}_{s(0)}$, and the optimization problem is feasible for $\mathbf{s}(0)$. ■

Theorem 12.2 *Suppose that Property 12.1 holds, and there exists $s(0) \in S_{D_0}$ such that $x(0) \in R_{s(0)}$. Then:*

(i) If the optimization problem given in line 6 of Algorithm 25 is feasible for some $\mathbf{s}(k) \in \boldsymbol{P}^{I_k}_{s(k)}$ at time k for state $(s(k), x(k))$, and $s(k+1) \notin F_D$, then there exists $\mathbf{s}(k+1) \in \boldsymbol{P}^{I_{k+1}}_{s(k+1)}$ such that the problem is feasible for $\mathbf{s}(k+1)$ and state $(s(k+1), x(k+1))$.

(ii) The trajectory of $T_{\mathscr{W}}$ produced by the closed-loop system satisfies the specification.

Proof (i) Let $\mathbf{T}^*_k = (s^*(0|k), x^*(0|k)) \ldots (s^*(I_k|k), x^*(I_k|k))$ be the trajectory generated by the optimal control sequence $\mathbf{u}^*(k) = u^*(0|k) \ldots u^*(I_k|k)$ at step k. If $s(k+1) \notin F_D$ then $I_{k+1} \geq 1$ and from Eq. (12.27), it follows that $I_{k+1} \leq I_k$ and $V^c(s^*(I_{k+1}|k), x^*(I_{k+1}|k)) > 0$. By Proposition 12.2, there exists a control $u \in \mathbf{U}$ and a state $s' \in S_D$ such that $x' = \delta_{\mathscr{W}}(x^*(I_{k+1}|k), u)$ and $s' \in \delta_D(s^*(I_{k+1}|k))$. As such, the control sequence $u^*(1|k) \ldots u^*(I_k|k)u'$ is a feasible solution of the optimization problem at step $k+1$ for $s^*(1|k) \ldots s^*(I_{k+1}|k)s'$.

(ii) We first show that the produced trajectory reaches a final automaton state in finite time. By Lemma 12.2, the optimization problem is feasible at time $k = 0$ for some $\mathbf{s}(0) \in \mathbf{P}^{\mathrm{K}}_{s(0)}$. From Theorem 12.2-*(i)* it follows that an optimal predicted trajectory exists at each time step until a final state is reached. For time $k \in \mathbb{Z}_+$ $\mathbf{T}^*_k = (s^*(0|k), x^*(0|k)) \ldots (s^*(I_k|k), x^*(I_k|k))$ denotes the trajectory generated by the optimal control sequence $\mathbf{u}^*(k) = u^*(0|k) \ldots u^*(I_k|k)$. To prove the claim that there exists $k^* \in \mathbb{Z}_+$ such that $V^c(s(k^*), x(k^*)) = 0$, we will show that the prediction horizon eventually decreases, i.e., a final automaton state appears in the optimal predicted trajectory. Let $\bar{k} \in \mathbb{Z}_+$ and $I_{\bar{k}} = I_{\bar{k}+1}$. By Proposition 12.2, there exists $u' \in \mathbf{U}$ such that $x' = \delta_{\mathscr{W}}(x^*(I_{\bar{k}} \mid \bar{k}), u') \in R_{s'}$, $s' \in \delta_D(s^*(I_{\bar{k}} \mid \bar{k}))$ and

$$V^c(s', x') \leq \rho V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k})), \tag{12.35}$$

where ρ is the contraction rate of V^c. Let $\mathbf{u}(\bar{k}+1) = u^*(1|\bar{k}) \dots u^*(I_{\bar{k}}|\bar{k})u'$, and consider:

$$
\begin{aligned}
C_{\bar{k}} = \big(&\gamma^{\bar{k}+1} C(x(\bar{k}+1), \mathbf{u}(\bar{k}+1)) + (1-\gamma^{\bar{k}+1})\alpha V^c(s', x')\big) \\
&- \big(\gamma^{\bar{k}} C(x(\bar{k}), \mathbf{u}^*(\bar{k})) + (1-\gamma^{\bar{k}})\alpha V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k}))\big) \qquad (12.36) \\
= \gamma^{\bar{k}+1} &\left(L(x^*(I_{\bar{k}} \mid \bar{k}), u') + \sum_{i=1}^{I_{\bar{k}}} L(x^*(i \mid \bar{k}), u^*(i \mid \bar{k})) \right) + (1-\gamma^{\bar{k}+1})\alpha V^c(s', x') \\
&- \gamma^{\bar{k}} \left(L(x^*(0 \mid \bar{k}), u^*(0 \mid \bar{k})) + \sum_{i=1}^{I_{\bar{k}}} L(x^*(i \mid \bar{k}), u^*(i \mid \bar{k})) \right) \\
&+ (1-\gamma^{\bar{k}})\alpha V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k})).
\end{aligned}
$$

Consider L^* as in (12.33). By (12.35) and (12.36), it follows that

$$
\begin{aligned}
C_{\bar{k}} &< \gamma^{\bar{k}+1} L^* + (1-\gamma^{k+1})\alpha\rho V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k})) \\
&\qquad - (1-\gamma^{\bar{k}})\alpha V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k})) \\
&< \gamma^{\bar{k}+1} L^* + \alpha V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k})) \left(\rho - 1 + \gamma^{\bar{k}}\right). \qquad (12.37)
\end{aligned}
$$

Note that the equality $I_{\bar{k}} = I_{\bar{k}+1}$ and the definition of function V^c (12.7) imply that $V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k})) \geq 1$. Since $\gamma < 1$, from (12.37) we reach that $C_{\bar{k}} < 0$ for a sufficiently large $\bar{k}$. The bound is found for $\mathbf{u}(\bar{k}+1)$, which is a feasible solution of Eq. (12.32) at time step $\bar{k}+1$. As it is not necessarily the optimal solution, we have that

$$
\begin{aligned}
\big(\gamma^{\bar{k}+1} C(x(\bar{k}+1), \mathbf{u}^*(\bar{k}+1)) + (1-\gamma^{\bar{k}+1})\alpha V^c(s^*(I_{\bar{k}+1} \mid \bar{k}+1), x^*(I_{\bar{k}+1} \mid \bar{k}+1))\big)& \\
- \big(\gamma^{\bar{k}} C(x(\bar{k}), \mathbf{u}^*(\bar{k})) + (1-\gamma^{\bar{k}})\alpha V^c(s^*(I_{\bar{k}} \mid \bar{k}), x^*(I_{\bar{k}} \mid \bar{k}))\big)& \\
\leq C_{\bar{k}} < 0.& \qquad (12.38)
\end{aligned}
$$

Notice that for a sufficiently large $\bar{k}$, if (12.38) holds, then it holds for all $k' \geq \bar{k}$ until $I_{k'+1} \neq I_{k'}$. In the case when $I_{k'+1} \neq I_{k'}$, by (12.27) we have that $I_{k'+1} < I_{k'}$ and a final automaton state appears in the optimal trajectory $\mathbf{T}_{k'}$ at time step k'. Repeated applications of the derivation above implies that there exists $k^* \in \mathbb{Z}_+$ such that at time k^*-1, the equality $I_{k^*-1} = 1$ holds and $s(k^*) \in F_D$.

As shown above the proposed MPC controller produces a finite trajectory

$$(s(0), x(0)), \dots, (s(k^*), x(k^*)).$$

Next, we show that the projected system trajectory $x(0) \dots x(k^*)$ satisfies ϕ. It is assumed that $x(0) \in R_{s(0)}$ and $s(0) \in S_{D_0}$. By the definition of $\mathbf{P}_s^{I_k}$, $s(i+1) \in \delta_D(s(i))$

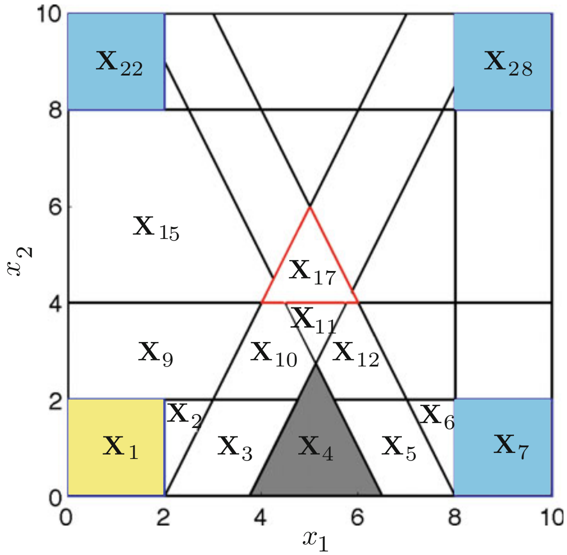


Fig. 12.3 The regions of the control system from Example 12.3. The regions used in the specification are highlighted

for all $i = 0, \ldots, k^* - 1$ and by the termination condition $s(k^*) \in F_D$. Consequently, $s(0) \ldots s(k^*)$ is an accepting automaton run. The constraints of the optimization problem given in line 6 ensure that $x(i) \in R_{s(i)}$ for all $i = 0, \ldots, k^*$. Hence, the system trajectory $x(0) \ldots x(k^*)$ satisfies the specification. ■

Example 12.3 We consider a discrete-time, planar linear control system with controls constrained to $\mathbf{U} = [-2, 2]^2$, states constrained to $X = [0, 10]^2$, and initial state $x(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. The regions of interest, which are shown in Fig. 12.3, induce a partition of $\mathbf{X}$ into 28 regions $\mathbf{X}_l$, $l \in L$, $L = \{1, 2, \ldots, 28\}$. This system can be written as a two-dimensional fixed-parameter PWA control system $\mathscr{W}$ (Definition 6.2) with:

$$A_{1,\ldots,28} = \begin{bmatrix} 0.99 & 0 \\ 0 & 0.98 \end{bmatrix}, \quad B_{1,\ldots,28} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, c_{1,\ldots,28} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{12.39}$$

We consider the following specification: "*The system originates in* $\mathbf{X}_1$, *eventually visits* $\mathbf{X}_{17}$, *and before visiting* $\mathbf{X}_{17}$ *it either visits* $\mathbf{X}_{22}$ *and* $\mathbf{X}_{28}$ *(in this order), or* $\mathbf{X}_7$. *Moreover, it does not visit* $\mathbf{X}_4$ *before it reaches* $\mathbf{X}_{17}$". The specification is translated to the following scLTL formula over $L \cup \{\text{Out}\}$:

$$\begin{aligned} \phi =& (1 \ \wedge \ \Diamond 17) \ \wedge \ (\neg 4 \ U \ 17) \ \wedge \ (\neg 17 \ U (7 \ \vee \ 28)) \ \wedge \\ & ((\neg 28 \ U \ 17) \ \vee \ (\neg 28 \ U \ 22)). \end{aligned} \tag{12.40}$$

First, we apply Algorithm 20 to find the largest set of satisfying initial states of the system, and the corresponding control strategy. During the refinement, we use polyhedral LFs method to synthesize transition controllers and transition weight function $\mathbf{W}$. The refinement algorithm (Algorithm 22) terminates at the first iteration with $X^{\phi}_{T,\mathscr{W}} = \mathbf{X}_1$, hence there exists a sequence of controllers such that all trajectories that originate from $\mathbf{X}_1$ and generated by these controllers satisfy the specification. The refined dual automaton has 101 finite cost states and 569 finite cost transitions.

We define a cost function as in Eq. (12.2) with

$$\mathrm{Q} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad \mathrm{R} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}. \tag{12.41}$$

We define reference trajectories according to a sequence of automaton states. In particular, for a given sequence of automaton states $s(0), \ldots, s(d)$, we define the first $\mathrm{K} - 1$ states of the reference trajectory as $x^r(i) := x^{c,0}$, $i = 0, \ldots, K - 2$, where $x^{c,0}$ is the center of the polytope $R_{s(0)}$. Then, we keep an index variable j (initialized to $j = 0$), and at each time step $k \geq 0$, we generate $x^r(k + \mathrm{K} - 1)$ and update j according to the state $x(k)$ of the controlled system as follows:

$$[x^r(k + \mathrm{K} - 1), j] := \begin{cases} [x^{c,j+1}, j + 1] & \text{if } x(k) \in R_{s(j)} \\ [x^r(k + \mathrm{K} - 2), j] & \text{otherwise.} \end{cases} \tag{12.42}$$

Essentially, we define a sequence of "target" regions $R_{s(0)}, \ldots, R_{s(d)}$ and design the reference trajectory such that the cost is minimized when the controlled trajectory visits these regions in the given order. To achieve this, at each time step, we pick the "target" region $R_{s(j)}$ from the sequence, and we add the center of this region $x^{c,j}$ to the reference trajectory. Once the controlled trajectory visits $R_{s(j)}$, the next region $R_{s(j+1)}$ from the sequence is defined as the target region. Consequently, the cost function penalizes the distance to the target region $R_{s(j)}$ until the controlled trajectory visits $R_{s(j)}$.

The sequence of target regions can be considered as a "soft constraint". As opposed to the scLTL specification, the controlled trajectory is allowed to violate it (e.g., it does not visit a region from the sequence). However, the violation is penalized with the cost function. Furthermore, while it is not possible to update the scLTL formula, the sequence of target regions can be changed "online" in response to the environmental changes.

We apply the MPC controller with terminal cost (Algorithm 25) to the refined dual automaton A_D. Two system trajectories generated by the MPC controller are shown in Fig. 12.4a, b, where the reference trajectories are generated as explained above according to the sequences of automaton states s_{99}, s_4, s_2 and s_{92}, s_{100}, s_2, respectively. For both experiments, the reference control sequences are defined as $u^r(i) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $i \in \mathbb{N}$, the prediction horizon K is 5, the scaling factor α is 1.45 (computed as in Eq. (12.33)), the weight constant γ is 0.95, and the successor function is Γ_{SP} as defined in Sect. 11.7.

Note that both trajectories from Fig. 12.4 satisfy specification ϕ (12.40). The experiments show that we can use the reference trajectories to steer the closed-loop trajectory towards the desired regions, which is not possible for the MPC controller with terminal constraints.

Next, we use a reference trajectory that does not reach a region of a final automaton state, and compare the controlled trajectories generated by the MPC controller initialized with different weight constants, i.e., $\gamma = 0.995$ and $\gamma = 0.99$. The reference control and state trajectories, the scaling constant and the prediction horizon are defined as $x^r(i) = \begin{bmatrix} 1 \\ 9 \end{bmatrix}$, $i \in \mathbb{N}$, and $u^r(i) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $i \in \mathbb{N}$, $\alpha = 1.45$ and $\mathrm{K} = 5$, respectively. As shown in Fig. 12.5, both trajectories satisfy ϕ (12.40). Due to the exponential increase of the weight of the terminal cost, the trajectory generated with $\gamma = 0.99$ reaches a final automaton region in 38 time steps, while the trajectory generated with $\gamma = 0.995$ reaches a final automaton region in 239 time steps.

Example 12.4 We consider the PWA system and the specification ϕ from Example 11.6. We define the matrices of cost function (12.2) as

$$\mathrm{Q} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, \quad \mathrm{R} = \begin{bmatrix} 0.8 & 0 \\ 0 & 0.8 \end{bmatrix}. \tag{12.43}$$

We apply both the proposed MPC controllers, i.e., MPC with a terminal constraint and MPC with a terminal cost function, to the refined dual automaton A_D. As in Example 12.2, the successor function Γ_{SP} (shortest path), Γ_{LP} (constructed by choosing the state maximizing the right hand side of (11.6)) and the control potential function $V_{con,CS}$, which is defined in Appendix A.7.1, are used in Algorithm 24 for $\underline{\Gamma}$, $\overline{\Gamma}$ and V_{con}, respectively. For Algorithm 25, the scaling factor α is 148 (computed as in (12.33)), the weight constant γ is 0.95, and the successor function is Γ_{SP}.

The reference control sequence **u** and the reference trajectory **x** are defined as

$$\mathbf{u}^{r_3}(i) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mathbf{x}^{r_3}(i) = \begin{bmatrix} 10 \\ 90 \end{bmatrix}, \quad \text{for all } i = 0, \ldots, 99. \tag{12.44}$$

Note that the reference trajectory corresponds to a point in the interior of the target region. The reference control sequence and the reference trajectory allow us to minimize the magnitude of the control applied and the distance to the target region.

As shown in Fig. 12.6, both MPC controllers generate similar trajectories, which advance towards the target region diagonally for the first steps, and then follow the edge of the grey region, and finally reach the target region. To prevent trajectories from reaching the boundaries of the grey region, standard techniques of motion planning such as obstacle inflation can be used. Note that both terminal cost and the stage cost penalize the distance from the target region. The average amount of time spent at each iteration of Algorithm 24 and Algorithm 25 are 3.55 sec. and 0.31 sec., respectively. The difference is due to the amount of time spent to compute the terminal constraint sets, and additional QPs solved in Algorithm 24.

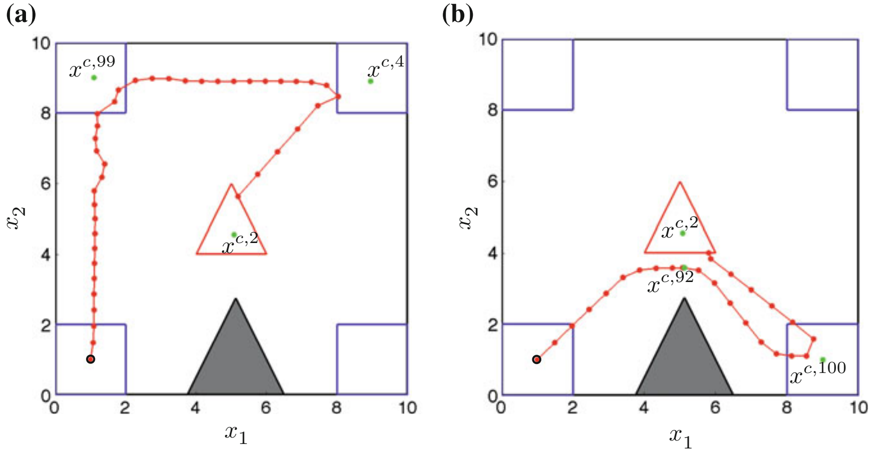


Fig. 12.4 The trajectories of the controlled system from Example 12.3. **a**, **b** The reference trajectories are generated from automaton sequences s_{99}, s_4, s_2 and s_{92}, s_{100}, s_2, respectively. The center points of the corresponding polytopes are marked with *green dots*. The initial states are marked by *black circles*. **a** The controlled trajectory reaches $R_{s_{99}}$, R_{s_4} and R_{s_2} at time steps 17, 33 and 38, respectively, i.e., $x_{17} \in R_{s_{99}}$, $x_{33} \in R_{s_4}$, and $x_{38} \in R_{s_2}$. The corresponding reference trajectory generated as defined in Eq. (12.42) is $x^r(i) = x^{c,99}$, $i = 0, \ldots, 20$, $x^r(i) = x^{c,4}$, $i = 21, \ldots, 36$ and $x^r(i) = x^{c,2}$, $i = 37, 38$. **b** The controlled trajectory reaches $R_{s_{92}}$, $R_{s_{100}}$ and R_{s_2} at time steps 9, 16 and 24. The corresponding reference trajectory is $x^r(i) = x^{c,92}$, $i = 0, \ldots, 12$, $x^r(i) = x^{c,4}$, $i = 13, \ldots, 19$ and $x^r(i) = x^{c,2}$, $i = 20, \ldots, 24$. Note that prediction horizons are shortened in the last steps for (**a**, **b**) as defined in Eq. (12.27)

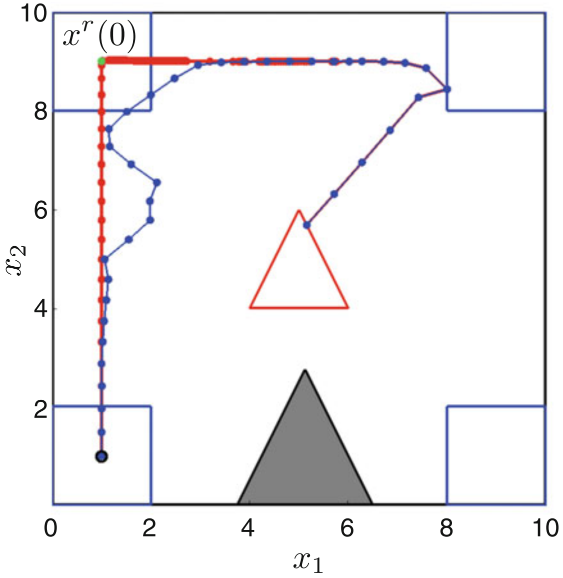


Fig. 12.5 The trajectories generated with $\gamma = 0.995$ and $\gamma = 0.99$ are shown in *red* and *blue*, respectively. The initial state is marked by a *black circle*

12.4 Notes

This chapter is based on our work [70, 73, 77], and can be integrated in the general area of model predictive control (MPC). This has been shown to be an efficient and successful method in constrained control, and MPC of linear systems has been studied quite extensively [150]. With a few exceptions [60, 149, 175], in the MPC

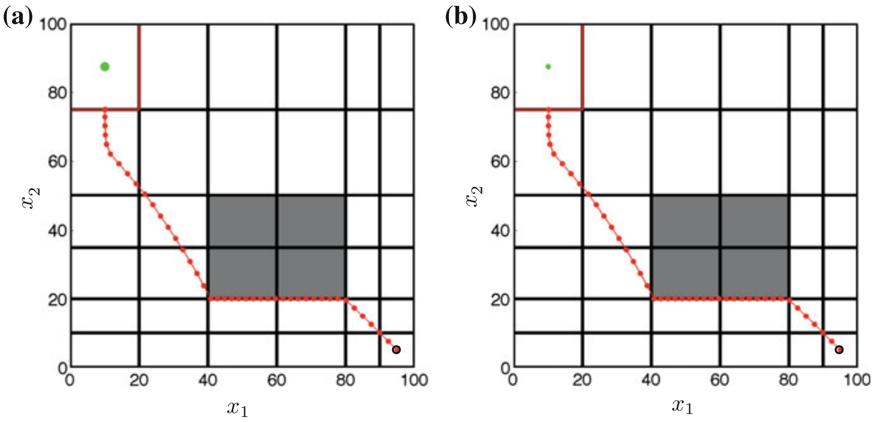


Fig. 12.6 **a** The trajectory generated by the MPC controller with terminal constraints (K = 5, sum of the stage costs 193572). **b** The trajectory generated by the MPC controller with terminal cost (K = 5, sum of the stage costs 193447)

literature, only classical control specifications were considered, such as stability and safety. A safety specification, i.e., the system trajectory does not leave a safe set, can be efficiently encoded in the optimization problem through constraints, e.g., linear constraints for polyhedral safe sets. For a stability specification, terminal cost and terminal constraint methods are used for stabilizing model predictive controllers (see [127] for a review). Several methods were proposed for designing a terminal constraint set and a terminal cost function to ensure stability [127]. The terminal constraint and terminal cost methods presented in this chapter resemble the so-called techniques from MPC literature, while they are set to guarantee the satisfaction of a terminal logic specification.

The automaton-enabled trajectory is related to the hybrid trajectory of a hybrid system [87], in the sense that the trajectory is defined over a hybrid state space, i.e., $\bigcup_{s \in S_D} \{\{s\} \times R_s\}$. MPC control schemes were also developed for hybrid systems such as piece-wise affine systems [121], hybrid dynamical systems with continuous and discrete states [49, 50], and mixed logical dynamical systems, which include hybrid automata [29]. However, as in MPC for linear systems, only classical control specifications were considered in these works. In [49, 50], a hybrid CLF for a hybrid system with continuous and discrete dynamics was designed to enforce the closed-loop stability of an equilibrium point. The hybrid CLF combined a Lyapunov-type function based on a graph distance over the discrete states of the system and a classical CLF over the continuous states of the system. While the potential functions from this chapter also combine two distance measures, they are designed to enforce the satisfaction of a temporal logic formula.

The complexities of the presented MPC controllers are characterized by the number of quadratic programs solved at each iteration. The optimal solution of the MPC problem with terminal constraint given in Eq. (12.18) is found by solving an optimization problem for each $\mathbf{s}(k) \in \mathbf{P}^{\mathrm{K}}_{s(k)}$ as shown in Eq. (12.25). The number of paths of length d originating at a node of a graph is upper bounded by b^d, where b is the branching factor of the graph, i.e., the maximum number of outgoing edges from a node. Therefore, the size of $\mathbf{P}^{\mathrm{K}}_{s(k)}$ is upper bounded by b^{K}. The number of quadratic optimization problems solved to find the optimal solution for a given path $\mathbf{s}(k)$ depends on the representation of the terminal constraint set $\mathscr{R}^{\bar{k},s(\mathrm{K}|k)\underline{\Gamma}(s(\mathrm{K}|k))}_{V_{con}}$(see Eq. (12.21)), and therefore the choice of the control potential function. Let $w^* = \max_{(s,s')|s' \in \delta_D(s), s \neq s'} \{\mathbf{W}(s, s') < \infty\}$. The constraint set $\mathscr{R}^{\bar{k},s(\mathrm{K}|k)\underline{\Gamma}(s(\mathrm{K}|k))}_{V_{con}}$ is represented as unions of at most $w^* - 1$ polytopes for the control potential functions given in Appendix A.7. Therefore, when one of these functions is used, the optimal solution of the MPC problem with terminal constraints given in Eq. (12.18) can be found by solving at most $b^{\mathrm{K}} w^*$ quadratic programs. Similarly, the optimal solution of the MPC problem with terminal cost given in Eq. (12.32) is found by solving an optimization problem for each $\mathbf{s}(k) \in \mathbf{P}^{\mathrm{K}}_{s(k)}$ as shown in Eq. (12.34). In this case, the optimal solution of a given path $\mathbf{s}(k)$ is found by solving a single quadratic program, since all the constraints given in Eq. (12.34) are linear. Therefore, the optimal solution of the MPC problem with terminal cost (12.32) can be found by solving at most b^{K} quadratic programs.

The computational framework presented in this chapter was implemented as a Matlab software package, called LanGuiMPC (Language Guided Model Predictive Control), which is freely downloadable from http://sites.bu.edu/hyness/languics/. This toolbox is an extension of the LanGuiCS toolbox, which is described in Sect. 11.4. The main Matlab function implements an MPC controller for a discrete time piece-wise affine system constrained to satisfy an scLTL formula over a set of linear predicates in its state variables. In addition to the input necessary for LanGuiCS, LanGuiMPC requires a quadratic cost function that penalizes the distance from desired state and control trajectories, which are assumed to be observable for a finite time horizon. The toolbox allows to plot the closed loop trajectories for 2D and 3D examples.

Appendix A
Background

In this appendix, we provide some background necessary for the material presented in Part III of the book. Specifically, in Sects. A.1 and A.2 we introduce polytopes and polyhedral operations. In Sect. A.3, we discuss the computation of images and pre-images of polytopes through affine functions, under the assumption that the map is non-singular. We relax this assumption and generalize polytopes to semi-linear sets in Sect. A.4. An overview of discrete-time Lyapunov stability and polyhedral Lyapunov functions, which are used in Chap. 10, is presented in Sect. A.5. The details of the vertex interpolation and contractive sets methods used in Chap. 11 to design polytope-to-polytope controllers are shown in Sect. A.6. Finally, candidate control potential functions, which are defined and used in Chap. 12, are presented in Sect. A.7.

A.1 Polytopes

Definition A.1 (*Convex Set*) A set $C \subset \mathbb{R}^N$ is *convex* if the line segment between any two points in C lies in C. In other words, for all $x_1, x_2 \in C$ and $0 \leq \lambda \leq 1$, we have $\lambda x_1 + (1 - \lambda) x_2 \in C$.

Definition A.2 (*Convex Combination*) A point $x = \sum_{i=1}^{n} \lambda_i x_i$, where $\sum_{i=1}^{n} \lambda_i = 1$ and $\lambda_i \geq 0$ for all $i = 1, \ldots, n$ is a *convex combination* of points $x_1, \ldots, x_n$.

Similarly, a point $x = \sum_{i=1}^{n} \lambda_i x_i$, where $\sum_{i=1}^{n} \lambda_i = 1$ is an *affine combination* of points $x_1, \ldots, x_n$. Points $x_1, \ldots, x_n$ are called *affinely independent* if there does not exist an $1 \leq i \leq n$ such that point x_i is an affine combination of points $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n$.

Definition A.3 (*Convex Hull*) The *convex hull* of a set C, denoted as $hull(C)$, is the set of all convex combinations from points in C:

$$hull(C) = \{x \in \mathbb{R}^N \mid x = \sum_{i=1}^{n} \lambda_i x_i,\ \lambda_i \geq 0,\ x_i \in C,\ i = 1, \ldots, n,\ \sum_{i=1}^{n} \lambda_i = 1\}.$$

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7

The *convex hull* of a set C is the smallest convex set containing C. A *hyperplane* is a set of the form

$$\{x \in \mathbb{R}^N \mid h^\top x = k,\ h \in \mathbb{R}^N,\ h \neq 0,\ k \in \mathbb{R}\}.$$

A hyperplane divides $\mathbb{R}^N$ into two *half-spaces*.

Definition A.4 (*Half-space*) A closed half-space is a set of the form

$$\{x \in \mathbb{R}^N \mid h^\top x \leq k,\ h \in \mathbb{R}^N,\ h \neq 0,\ k \in \mathbb{R}\}.$$

A *supporting hyperplane* of a set C is a hyperplane $\{h^\top x = k\}$, such that C is entirely contained in one of the two closed half-spaces defined by the hyperplane and C has at least one point on the hyperplane.

Definition A.5 (*Polytope*) A closed full dimensional *polytope* $\mathbf{X} \subset \mathbb{R}^N$ is defined as the convex hull of at least $N + 1$ affinely independent points in $\mathbb{R}^N$.

The set of points $v_1, \ldots, v_n \in \mathbb{R}^N$ whose convex hull gives $\mathbf{X}$ and with the property that for all $i = 1, \ldots, n$, point v_i is not contained in the convex hull of $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$ is called the set of *vertices* of $\mathbf{X}$ and is denoted by $V(\mathbf{X})$. A polytope is completely described by its set of vertices:

$$\mathbf{X} = hull(V(\mathbf{X})). \tag{A.1}$$

Alternatively, a polytope $\mathbf{X}$ can be described as the intersection of at least $N + 1$ closed half spaces. In other words, there exists a $n \geq N + 1$ and $h_i \in \mathbb{R}^N$, $k_i \in \mathbb{R}$, $i = 1, \ldots, n$ such that

$$\mathbf{X} = \bigcap_{i=1}^{n} \{x \in \mathbb{R}^N \mid h_i^\top x \leq k_i\}. \tag{A.2}$$

Equivalently, by constructing a matrix $H \in \mathbb{R}^{n \times N}$, where $H = [h_1^\top; \ldots; h_n^\top]$ and vector $K \in \mathbb{R}^n$, where $K = [k_1; \ldots; k_n]$: [1]:

$$\mathbf{X} = \{x \in \mathbb{R}^N \mid Hx \leq K\}, \tag{A.3}$$

where the comparison "$\leq$" is interpreted element-wise.

Forms (A.1) and (A.2) (or equivalently (A.3)) are referred to as the V- and H-representations of a polytope, respectively. Given a polytope, there exist algorithms for translation between its V- and H-representations [113, 134]. A *facet* of a polytope $\mathbf{X}$ is the intersection of $\mathbf{X}$ with one of its supporting hyperplanes. A polytope without its facets is called an *open polytope* and we use $int(\mathbf{X})$ to denote $\mathbf{X}$ without its facets (i.e., the interior of $\mathbf{X}$). Given an open polytope $\mathbf{X}$, we use $cl(\mathbf{X})$ to denote its closure (i.e., the union of $\mathbf{X}$ and its facets).

[1] We use standard Matlab notation for constructing concatenations of matrices.

A.2 Operations on Polytopes

Given polytopes $\mathbf{X}_1, \mathbf{X}_2 \subset \mathbb{R}^N$, we define the following operations:

Definition A.6 (*Set Difference*) The *set difference* of $\mathbf{X}_1$ and $\mathbf{X}_2$ is defined as:

$$\mathbf{X}_1 \backslash \mathbf{X}_2 = \{x \in \mathbb{R}^N \mid x \in \mathbf{X}_1, x \notin \mathbf{X}_2\}.$$

Note that convex polytopes are not closed under the set difference operation (i.e., $\mathbf{X}_1 \backslash \mathbf{X}_2$ is not necessarily convex, even if $\mathbf{X}_1$ and $\mathbf{X}_2$ are).

Definition A.7 (*Minkowski Sum*) The *Minkowski sum* of $\mathbf{X}_1$ and $\mathbf{X}_2$ is defined as:

$$\mathbf{X}_1 \oplus \mathbf{X}_2 = \{x_1 + x_2 \in \mathbb{R}^N \mid x_1 \in \mathbf{X}_1, x_2 \in \mathbf{X}_2\}.$$

Definition A.8 (*Minkowski Difference*) The *Minkowski difference* of $\mathbf{X}_1$ and $\mathbf{X}_2$ is defined as:

$$\mathbf{X}_1 \ominus \mathbf{X}_2 = \{x_1 - x_2 \in \mathbb{R}^N \mid x_1 \in \mathbf{X}_1, x_2 \in \mathbf{X}_2\}.$$

The Minkowski difference $\mathbf{X}_1 \ominus \mathbf{X}_2$ can also be computed as the Minkowski sum $\mathbf{X}_1 \oplus (-\mathbf{X}_2)$, where $(-\mathbf{X}_2) = \{x \in \mathbb{R}^N \mid -x \in \mathbf{X}_2\}$ is the mirror image of $\mathbf{X}_2$ around the origin. Note that our definition of Minkowski difference follows [118] and is different from the Pontryagin (Minkowski) difference from [78, 112].

Definition A.9 (*Chebyshev Ball*) The *Chebyshev ball* of a polytope $\mathbf{X} \subset \mathbb{R}^N$ is the largest radius ball $B_r(x_c) = \{x \in \mathbb{R}^N \mid ||x - x_c||_2 \leq r\}$ such that $B_r(x_c) \subset \mathbf{X}$. We use $c(\mathbf{X})$ to denote the center and $r(\mathbf{X})$ to denote the radius of the Chebyshev ball of $\mathbf{X}$.

A.3 Affine Functions on Polytopes

Definition A.10 (*Affine function*) A function $f : \mathbb{R}^N \to \mathbb{R}^M$ is called affine if it can be written as $f(x) = Ax + b$, $A \in \mathbb{R}^{M \times N}$, $b \in \mathbb{R}^M$, for all $x \in \mathbb{R}^N$.

If $\mathbf{X}$ is a full dimensional polytope in $\mathbb{R}^N$ with set of vertices $V(\mathbf{X}) = \{v_1, \ldots, v_n\}$ and $f : \mathbb{R}^N \to \mathbb{R}^M$ is an affine function, then

$$f(\mathbf{X}) = hull\{f(v_1), \ldots, f(v_n)\}, \tag{A.4}$$

i.e., the image of a polytope through an affine function is the convex hull of the vertex images through the affine function.

In the particular case $N = M$, if matrix A is nonsingular, then the vertices, facets, and interior of the polytope map through the affine transformation to the vertices, facets, and interior of the image of the polytope, respectively. Therefore

$$f(cl(\mathbf{X})) = cl(f(\mathbf{X})). \tag{A.5}$$

The pre-image of a polytope $\mathbf{X}$ in $\mathbb{R}^M$ through an affine function $f : \mathbb{R}^N \to \mathbb{R}^M$ is a convex set in $\mathbb{R}^N$ and is defined as

$$f^{-1}(\mathbf{X}) = \{x \in \mathbb{R}^N \mid Ax + b \in \mathbf{X}\}. \tag{A.6}$$

For a polytope $\mathbf{X}$ in $\mathbb{R}^M$ represented as the intersection of $m \geq M$ half-spaces as in Eq. (A.2), the pre-image is given by

$$f^{-1}(\mathbf{X}) = \bigcap_{i=1}^{m} \{x \in \mathbb{R}^N \mid h_i^\top Ax \leq k_i - h_i^\top b\}. \tag{A.7}$$

Note that if there are $N + 1$ linearly independent vectors in the set $\{h_1^\top A, \ldots, h_m^\top A\}$, then $f^{-1}(\mathbf{X})$ is a polytope in $\mathbb{R}^N$.

Consider a function $g : \mathbb{R}^N \times \mathbb{R}^L \to \mathbb{R}^M$ defined as $g(x, u) = Ax + Bu + b$, where $A \in \mathbb{R}^{M\times N}$, $B \in \mathbb{R}^{M\times L}$, $b \in \mathbb{R}^M$. The function g can be represented as summation of two affine functions $g_1 : \mathbb{R}^N \to \mathbb{R}^M$ and $g_2 : \mathbb{R}^L \to \mathbb{R}^M$ such that $g_1(x) = Ax + b$ and $g_2(u) = Bu$, i.e., $g(x, u) = g_1(x) + g_2(u)$ for all $x \in \mathbb{R}^N$, $u \in \mathbb{R}^L$. Then, the image of polytopes $\mathbf{X} \subset \mathbb{R}^N$ and $\mathbf{U} \subset \mathbb{R}^L$ through function g can be computed as

$$g(\mathbf{X}, \mathbf{U}) = g_1(\mathbf{X}) \oplus g_2(\mathbf{U}),$$

where $\oplus$ denoted Minkowski summation. Also note that function $g : \mathbb{R}^N \times \mathbb{R}^L \to \mathbb{R}^M$ can equivalently be represented as an affine function $f : \mathbb{R}^{N+L} \to \mathbb{R}^M$ as given in Definition A.10:

$$f(x) = \bar{A}\bar{x} + b, \text{ where } \bar{x} \in \mathbb{R}^{N+L} \text{ and } \bar{A} = [A, B] \in \mathbb{R}^{M\times(N+L)}. \tag{A.8}$$

Then, $g(\mathbf{X}, \mathbf{U}) = f(\mathbf{X} \times \mathbf{U})$. Moreover, the pre-image of the function g over $\mathbb{R}^N$ can be computed through orthogonal projection of $f^{-1}(\mathbf{X} \times \mathbf{U})$ to $\mathbb{R}^N$.

A.4 Semi-linear Sets and Affine Functions

Definition A.11 (*Semi-linear set*) A *semi-linear set* $\mathbf{S}$ in $\mathbb{R}^N$ is defined as unions, intersections, and complements of sets $\{x \in \mathbb{R}^N \mid a^\top x \sim b, \sim \in \{=, <\}\}$, for some $a \in \mathbb{R}^N$ and $b \in \mathbb{R}$.

A semi-linear set is also called a *polyhedron*. A closed convex bounded semi-linear set is a polytope, and a convex bounded semi-linear set is a polytope with some of its facets removed.

Note that the basic set operation definitions, such as Minkowski sum and difference, and set difference defined above, also apply to semi-linear sets. Furthermore, the image (or pre-image) of a semi-linear set through an affine function can be computed by performing the computation on a finite number of polytopes since a semi-linear set can be represented as a union of a finite number of convex sets. In particular, a semi-linear set is either a polytope, or a union of polytopes, or a convex and bounded set, or a general non-convex set. Therefore, the computation of the image (or pre-image) of a semi-linear set $\mathbf{S}$ through an affine function f falls into one of the following case:

i. If $\mathbf{S}$ is a closed polytope, then $f(\mathbf{S})$ and $f^{-1}(\mathbf{S})$ are computed as explained in Sect. A.3. Note that the computation applies to a polytope of any dimension.

ii. If $\mathbf{S}$ is a union of polytopes, one can use a standard convex decomposition method to decompose $\mathbf{S}$ into a set of polytopes $\{\mathbf{P}_i\}_{i\in I}$ (see, e.g., [79]), i.e.,

$$\mathbf{S} = \bigcup_{i\in I} \mathbf{P}_i, \text{ where for any } i \neq j, \mathbf{P}_i \cap \mathbf{P}_j = \emptyset,$$

and compute the image as

$$f(\mathbf{S}) = \bigcup_{i\in I} f(\mathbf{P}_i)$$

using case Sect. A.4 (similarly $f^{-1}(\mathbf{S}) = \bigcup_{i\in I} f^{-1}(\mathbf{P}_i)$).

iii. If $\mathbf{S}$ is a convex and bounded semi-linear set, then, in general,

$$\mathbf{S} = \mathbf{P} \setminus \bigcup_{i\in I} \mathbf{F}_i,$$

for some polytope $\mathbf{P}$ and a subset of its facets $\{\mathbf{F}_i\}_{i\in I}$. The image is computed as

$$f(\mathbf{S}) = f(\mathbf{P}) \setminus \left(\bigcup_{i\in I} f(\mathbf{F}_i) \setminus f(\mathbf{P}) \right),$$

where the right hand side can be computed as described in case Sect. A.4. Similarly, the pre-image can be computed as

$$f^{-1}(\mathbf{S}) = f^{-1}(\mathbf{P}) \setminus \left(\bigcup_{i\in I} f^{-1}(\mathbf{F}_i) \setminus f^{-1}(\mathbf{P}) \right).$$

iv. If $\mathbf{S}$ is a general (non-convex) bounded semi-linear set, then again it can be decomposed into convex and bounded semi-linear sets $\mathbf{S} = \bigcup_{i\in I} \mathbf{S}_i$. Then,

$$f(\mathbf{S}) = \bigcup_{i\in I} f(\mathbf{S}_i),$$

and each $f(\mathbf{S})$ can be computed as described in case Sect. A.4.

As summarized above, the image and the pre-image of a semi-linear set through an affine function can always be implemented by convex decompositions and repeated applications of Eqs. (A.4) and (A.7), respectively.

A.5 Lyapunov Theory

An autonomous discrete-time system is defined by

$$x(k+1) = \Phi(x(k)), \quad k \in \mathbb{N}, \tag{A.9}$$

where $x(k) \in \mathbb{R}^N$ is the state at time k and $\Phi : \mathbb{R}^N \to \mathbb{R}^N$ is an arbitrary map with $\Phi(0) = 0$. Given a state $x \in \mathbb{R}^N$, $x' := \Phi(x)$ is called a *successor* state of x.

Definition A.12 (*Contractive set and positively invariant set*) Let $\lambda \in [0, 1]$. A set $\mathbf{P} \subseteq \mathbb{R}^N$ is called λ*-contractive* (shortly, contractive) for system (A.9) if for all $x \in \mathbf{P}$ it holds that $\Phi(x) \in \lambda\mathbf{P}$. For $\lambda = 1$, $\mathbf{P}$ is called a *positively invariant* set.

Definition A.13 (*Class $\mathcal{K}_\infty$ function*) A function $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ belongs to class $\mathcal{K}_\infty$ if it is continuous, strictly increasing, $\alpha(0) = 0$, and $\lim_{s\to\infty} \alpha(s) = \infty$.

Theorem A.1 *Let* $\mathbf{P}$ *be a positively invariant set for* (A.9) *with* $0 \in int(\mathbf{P})$. *Furthermore, let* $\alpha_1, \alpha_2 \in \mathcal{K}_\infty$, $\rho \in (0, 1)$ *and* $V : \mathbb{R}^N \mapsto \mathbb{R}_+$ *such that:*

$$\alpha_1(\|x\|) \le V(x) \le \alpha_2(\|x\|), \forall x \in \mathbf{P}, \tag{A.10}$$

$$V(\Phi(x)) \le \rho V(x), \forall x \in \mathbf{P}. \tag{A.11}$$

Then (the origin of) system (A.9) *is asymptotically stable in* $\mathbf{P}$ *[93, 119].*

Definition A.14 (*Lyapunov function*) A function $V : \mathbb{R}^N \mapsto \mathbb{R}_+$ is called a *Lyapunov function* (LF) in $\mathbf{P}$ if it satisfies (A.10) and (A.11). If $\mathbf{P} = \mathbb{R}^N$, then V is called a *global Lyapunov function*.

The parameter ρ is called the *contraction rate* of V. For any $\Gamma > 0$,

$$\mathbf{P}_\Gamma := \{x \in \mathbb{R}^N \mid V(x) \le \Gamma\}$$

is called a *sublevel set* of V.

A difference inclusion system is defined as

$$x(k+1) \in \Phi(x(k)), \quad k \in \mathbb{N}, \tag{A.12}$$

where $x(k) \in \mathbb{R}^N$ is the state at time k and $\Phi : \mathbb{R}^N \to 2^{\mathbb{R}^N}$ is an arbitrary map with $\Phi(0) = 0$. Note that the system is non-deterministic, and the successor of state x takes values from a set $\Phi(x)$.

The invariant set, contractive set, and Lyapunov function definitions apply directly to difference inclusions in the absolute sense, i.e., given x, the corresponding conditions must hold for all $x' \in \Phi(x)$. For example, a set $\mathbf{P}$ is positively invariant if $\Phi(x) \subseteq \mathbf{P}$ for all $x \in \mathbf{P}$.

An infinity norm Lyapunov function is in the form

$$V(x) = \|Lx\|_{\infty}, \quad L \in \mathbb{R}^{l\times N}, l \geq N, l \in \mathbb{N}, \tag{A.13}$$

where L has full-column rank. Infinity norm Lyapunov functions are particular types of polyhedral Lyapunov functions.

If $L \in \mathbb{R}^{l\times N}$ has full-column rank and V as defined in (A.13) is a global Lyapunov function for system (A.9) with contraction rate $\rho \in (0, 1)$, then for all $\Gamma > 0$, the induced sublevel set $\mathbf{P}_{\Gamma}$ is a polytope and $0 \in int(\mathbf{P}_{\Gamma})$. Moreover, if $\Phi(x) = Ax$ for some $A \in \mathbb{R}^{N\times N}$, then for all $\Gamma > 0$, the induced sublevel set $\mathbf{P}_{\Gamma}$ is a ρ-contractive polytope for (A.9) [32, 120].

A.6 Reach Control Problems on Polytopes

Consider an affine control system of the form

$$x(k+1) = Ax(k) + Bu(k) + c, x(k) \in \mathbb{R}^N, \; u(k) \in \mathbf{U}, \tag{A.14}$$

where $A \in \mathbb{R}^{N\times N}$, $B \in \mathbb{R}^{N\times M}$ and $c \in \mathbb{R}^N$ describe the system dynamics, at each time step $k = 0, 1, \ldots, x(k) \in \mathbb{R}^N$ is the state of the system and $u(k)$ is the control restricted to a polytopic set $\mathbf{U} \subset \mathbb{R}^M$.

Given two polytopes $\mathbf{S}$ and $\mathbf{T}$ in $\mathbb{R}^N$ with $\mathbf{T} \subseteq \mathbf{S}$, and a control system (A.14), the reach control problem concerns synthesis of a feedback control law $g : \mathbb{R}^N \to \mathbb{R}^M$ such that for all $x \in \mathbf{S}$ there exists a $k_x \in \mathbb{N}$, $k_x < \infty$ and:

$$\begin{aligned}
&x(0) = x, && \\
&x(k+1) = Ax(k) + Bg(x(k)) + c, && k = 0, \ldots, k_x - 1, \\
&x(k) \in \mathbf{S} && k = 0, \ldots, k_x, \\
&x(k_x) \in \mathbf{T}. &&
\end{aligned}$$

The reach control problem might not be feasible, i.e., such a control law g might not exists. In the remainder of this section, several approaches for synthesis of a feedback control law solving the reach control problem are provided. For convenience of notation, a H-representation of a polytope $\mathbf{P}$ is denoted by matrices $H_{\mathbf{P}}$,$K_{\mathbf{P}}$, where $\mathbf{P} = \{x \mid H_{\mathbf{P}}x \leq K_{\mathbf{P}}\}$.

A.6.1 Iterative *Pre*-computation

The set of states of system (A.14) that can reach $\mathbf{T}$ in one-step can be computed via Pre operator and orthogonal projection as explained in Sect. A.3:

$$Pre(\mathbf{T} \times \mathbf{U})|_N,$$

where, for a set $\mathbf{P}$, $\mathbf{P}|_N$ denotes the orthogonal projection of $\mathbf{P}$ to $\mathbb{R}^N$. Simply $Pre(\mathbf{T} \times \mathbf{U})|_N$ is denoted by $Pre(\mathbf{T})$. Then, the set of states $\mathbf{T}^1 \subseteq \mathbf{S}$ that can reach $\mathbf{T}$ in one-step is the intersection of $Pre(\mathbf{T})$ with $\mathbf{S}$, i.e.,

$$\mathbf{T}^1 = \mathbf{S} \cap Pre(\mathbf{T}). \tag{A.15}$$

By iteratively applying (A.15), one can compute the set of states $\mathbf{T}^k \subseteq \mathbf{S}$ that can reach $\mathbf{T}$ *exactly* at step $k \in \mathbf{N}_+$, $k \geq 1$:

$$\mathbf{T}^k = \mathbf{S} \cap Pre(\mathbf{T}^{k-1}), \qquad \text{where } \mathbf{T}^0 = \mathbf{T}. \tag{A.16}$$

The set of states $\mathbf{T}^{\leq k} \subseteq \mathbf{S}$ that can reach $\mathbf{T}$ *within* $k \in \mathbb{N}$ steps is simply the union of $\mathbf{T}^0, \ldots, \mathbf{T}^k$:

$$\mathbf{T}^{\leq k} = \bigcup_{i=0,\ldots,k} \mathbf{T}^k. \tag{A.17}$$

Finally, the maximal set of states $\mathbf{T}^{\leq} \subseteq \mathbf{S}$ that can reach $\mathbf{T}$ in a finite number of steps is defined as

$$\mathbf{T}^{\leq} = \mathbf{T}^{\leq k} \text{ where } \mathbf{T}^{\leq k} = \mathbf{T}^{\leq k-1}. \tag{A.18}$$

Note that if $\mathbf{T}^{\leq} \neq \mathbf{S}$, then the reach-control problem is infeasible.

The above explained steps allows us to compute the maximal set $\mathbf{T}^{\leq}$. This method also induces a control strategy such that the trajectories originating from $\mathbf{T}^{\leq k}$ reach $\mathbf{S}$ with in k steps.

Consider the sets $\mathbf{T}^k$ and $\mathbf{T}^{k-1}$ defined above, and let $V(\mathbf{T}^k) = \{v^1, \ldots, v^n\}$. Consider the following set of linear inequalities in the variables $u^1, \ldots, u^n$:

$$\begin{aligned} H_{\mathbf{T}^{k-1}}(Av^i + Bu^i + c) &\leq K_{\mathbf{T}^{k-1}}, \\ H_{\mathbf{U}} u^i &\leq K_{\mathbf{U}}. \end{aligned} \tag{A.19}$$

A solution to (A.19) can be obtained by solving a feasibility linear program (LP). Note that the LP is always feasible since $\mathbf{T}^{\leq k} \subseteq Pre(\mathbf{T}^{\leq k-1})$. The control law

$$g_k(x) := \sum_{i=1}^{n} \lambda^i u^i, \tag{A.20}$$

where $\lambda^i \in \mathbb{R}$, $0 \leq \lambda^i \leq 1$, are such that $x = \sum_{i=1}^{n} \lambda^i v^i$, guarantees that the trajectories originating from $\mathbf{T}^k$ reach $\mathbf{T}^{k-1}$ in the next time instant. The evaluation of the control law (A.20) requires calculation of the coefficients $\lambda^1, \ldots, \lambda^n$, which amounts to solving a system of linear equations and can also be formulated as a feasibility LP.

Alternatively, an explicit PWA form of g_k can be obtained by a simplicial partition of $\mathbf{T}^k$. Then, the evaluation of g_k requires solving a point location problem [169], which consists of checking a finite number of linear inequalities. Although efficient ways to solve point location problems exist, depending on the complexity of the partition (number of simplices), the point location problem may be more computationally expensive than calculating the coefficients $\lambda^1, \ldots, \lambda^n$.

For a given state $x(0) \in \mathbf{S}$, the control strategy solving the reach control problem (under the assumption that the problem is feasible, i.e., $\mathbf{T}^{\leq} = \mathbf{S}$) amounts to finding $\mathbf{T}^k$ such that $x(0) \in \mathbf{T}^k$, and applying the control sequence $g_k(x(0)), g_{k-1}(x(1)), \ldots, g_1(x(k-1))$, where $x(i+1) = Ax(i) + Bg_{k-i}(x(i)) + c$, $i = 0, \ldots, k-1$.

Note that this method is guaranteed to find the solution to the reach-control problem if one exists. However, it is computationally expensive. Next, computationally more efficient, but conservative methods are presented.

A.6.2 Vertex Interpolation

Let $V(\mathbf{S}) = \{v^1, \ldots, v^n\}$ be the vertices of $\mathbf{S}$, and let $\{\mathbf{u}^1, \ldots, \mathbf{u}^n\}$ denote a corresponding set of finite sequences of control actions, where $\mathbf{u}^i := u_0^i, \ldots, u_{S-1}^i$ for all $i = 1, \ldots, n$, and $S \geq 1$. For each v^i define the following set of linear equality and inequality constraints in the variables $u_0^i, \ldots, u_{S-1}^i$:

$$\begin{aligned} x^i(0) &:= v^i, \\ x^i(k+1) &= Ax^i(k) + Bu^i(k) + c, \quad \forall k = 0, \ldots, S-1, \\ H_{\mathbf{S}} x^i(k) &\leq K_{\mathbf{S}}, \quad \forall k = 0, \ldots, S-1, \\ H_{\mathbf{U}} u^i(k) &\leq K_{\mathbf{U}}, \quad \forall k = 0, \ldots, S-1, \\ H_{\mathbf{U}} x^i(S) &\leq K_{\mathbf{U}}. \end{aligned} \tag{A.21}$$

A solution to the set of problems (A.21) can be searched by solving repeatedly a corresponding set of feasibility LPs starting with $S = 1$, for all $i = 1, \ldots, n$, and increasing S until a feasible solution is obtained for all LPs and the same value of S. Let $S^* \geq 1$ denote the minimal S for which a feasible solution was found. Then, it is straightforward to establish that for any $x \in \mathbf{S}$, the control law

$$g(x(k)) := \sum_{i=1}^{n} \lambda^i u^i(k), \quad k = 0, \ldots, S^* - 1, \tag{A.22}$$

where $x(0) = x$ and $\lambda^i \in \mathbb{R}$, $0 \leq \lambda^i \leq 1$, are such that $x = \sum_{i=1}^{n} \lambda^i v^i$, solves the reach control problem for **S** and **T**, and yields that closed-loop trajectories that reach **T** in at most S^* discrete-time instants.

Evaluation of the control law g of (A.22) at time $k = 0$ requires calculation of the coefficients $\lambda^1, \dots, \lambda^n$, which is an LP, while at every $k = 1, \dots, S^* - 1$ the analytic expression of g is implemented. However, a faster convergence to **T** can be obtained by taking $\lambda^i \in \mathbb{R}$, $0 \leq \lambda^i \leq 1$, such that $x = \sum_{i=1}^{n} \lambda^i x^i(j^*)$, where

$$j^* := \arg\max\{j \in \{0, \dots, S^*\} \mid x \in hull(x^1(j), \dots, x^n(j)).$$

Then, the resulting closed-loop trajectories will reach **T** in at most $S^* - j^*$ discrete-time instants.

Similar to the case of the control law from Eq. (A.20), simplicial decompositions of **S** can be employed to obtain an explicit PWA form of the control law $g(x(k))$, $k = 0, \dots, S^* - 1$, both for its standard and faster variants presented above.

A.6.3 Contractive Sets

In this section, first a Lyapunov function based solution to the reach control problem is presented, which is referred as "polyhedral LFs" method. In this method, the synthesized feedback controller guarantees that a point $x^s \in int(\mathbf{T})$ is an equilibrium point of the closed loop system and **P** is a sublevel set of a Lyapunov function. The contractive property of the sublevel set guarantees that the trajectories reach the set **T** in finite time. Then, these conditions are relaxed, and a less conservative solution, referred to as "contractive sets", is presented.

The set of equilibrium points in the interior of **T** is defined as

$$\mathbf{E_T} := \{x^s \in int(\mathbf{T}) \mid \exists u^s \in \mathbf{U} \,:\, x^s = Ax^s + Bu^s + c\}.$$

Polyhedral LFs

If $\mathbf{E_T} \neq \emptyset$, an explicit PWA solution to the reach control problem can be obtained via polyhedral LFs, see, e.g., [33, 120], as follows. Let

$$\mathscr{W}(x) := \max_{i=1,\dots,w} W_i^\top (x - x^s) \tag{A.23}$$

denote a function induced by the polytope **S** and a point $x^s \in \mathbf{E_T}$, where $w \geq N + 1$ is the number of lines of the matrix $W = [W_1^\top; \dots; W_w^\top]$, which is such that $\mathbf{S} = \{x \in \mathbb{R}^N \mid \mathscr{W}(x) \leq 1\}$. Next, consider the conic polytopic partition $\mathbf{C}_1, \dots, \mathbf{C}_w$ of **S** induced by x^s, which is constructed as follows:

$$\mathbf{C}_i := \{x \in \mathbf{S} | (W_i^\top - W_j^\top)(x - x^s) \geq 0,\ j = 1, \dots, w\} \cup \{x^s\}. \tag{A.24}$$

Notice that $\cup_{i=0,\dots,w}\mathbf{C}_i = \mathbf{S}$ and $int(\mathbf{C}_i) \cap int(\mathbf{C}_j) = \emptyset$ for all $i \neq j$. Let $\rho \in \mathbb{R}$ with $0 \leq \rho < 1$ denote a desired convergence rate. Consider the PWA control law

$$g(x) := F_i x + f_i \quad \text{if} \quad x \in \mathbf{C}_i \tag{A.25}$$

and the following feasibility LP in the variables $(F_1, f_1), \dots, (F_w, f_w)$:

$$\begin{aligned}
&\rho W_i^\top (x - x^s) - W_j^\top (Ax + Bg(x) + c - x^s) \geq 0,\\
&\qquad\qquad\qquad\qquad \forall x \in V(\mathbf{C}_i), \forall j = 1, \dots, w,\\
&F_i x + f_i \in \mathbf{U}, \quad \forall i = 1, \dots, w,\\
&(A + BF_i)x^s + Ba_i + c = x^s, \quad \forall i = 1, \dots, w.
\end{aligned} \tag{A.26}$$

Note that ρ can be minimized to obtain an optimal convergence rate and a different ρ_i can be assigned to each cone $\mathbf{C}_i$, while (A.26) remains an LP.

Proposition A.1 *Suppose that the LP* (A.26) *is feasible. Then the function* $\mathscr{W}$ *is a Lyapunov function and* $\mathbf{S}$ *is a* ρ*-contractive set for system* (A.14) *in closed-loop with the PWA control law* (A.25)*, with respect to the equilibrium* $x^s \in int(\mathbf{T})$.

The proof of Proposition A.1 follows from Theorem III.6 from [120], i.e., the constraints given in the LP (A.26) are the necessary conditions for constructing a Lyapunov function in Theorem III.6 from [120].

Let $k^* := \arg\min\{k \geq 1 \mid \rho^k(\mathbf{S} \oplus \{-x^s\}) \subseteq (\mathbf{T} \oplus \{-x^s\})\}$. All trajectories of system (A.14) in closed-loop with (A.25) that start in $\mathbf{S}$ reach $\mathbf{T}$ in at most k^* discrete-time instants. Thus, the PWA control law (A.25) solves the reach control problem for $\mathbf{S}$ and $\mathbf{T}$. The evaluation of (A.25) reduces to a point location problem that can be solved in logarithmic time due to the specific conic partition.

Next, we present a less conservative solution by relaxing the constraints of (A.26).

Contractive Sets

Pick any $\bar{x} \in int(\mathbf{T})$ (not necessarily an equilibrium point) and let $\mathscr{W}(x)$ denote the function induced by $\mathbf{S}$ and the point $\bar{x}$ as defined in (A.23). Moreover, let

$$\alpha^\star_{\bar{x}} = \max_{\alpha > 0}\{\alpha(\mathbf{S} \oplus \{-\bar{x}\}) \subseteq (\mathbf{T} \oplus \{-\bar{x}\})\}.$$

Note that $\alpha^\star_{\bar{x}} < 1$ whenever $\mathbf{T} \subseteq \mathbf{S}$ and $\mathbf{T} \neq \mathbf{S}$; pick any $\alpha_{\bar{x}}$ such that $0 < \alpha_{\bar{x}} < \alpha^\star_{\bar{x}}$. Let $\{\mathbf{C}_i\}_{i=1,\dots,w}$ be the conic partition of $\mathbf{S}$ induced by $\bar{x}$ as defined in (A.24), and let $\rho \in \mathbb{R}$ with $0 \leq \rho < 1$ denote a desired convergence rate. Consider the PWA control law (A.25), and the following feasibility LP in the variables $(F_1, f_1), \dots, (F_w, f_w)$:

$$\begin{aligned}
&\rho(W_i^\top (x - \bar{x}) - \alpha_{\bar{x}}) - (W_j^\top (Ax + Bg(x) + c - \bar{x}) - \alpha_{\bar{x}}) \geq 0,\\
&\qquad\qquad\qquad \forall x \in V(\mathbf{C}_i), \forall j = 1, \dots, w, \forall i = 1, \dots, w,\\
&F_i x + f_i \in \mathbf{U}, \quad \forall x \in V(\mathbf{C}_i), \forall i = 1, \dots, w.
\end{aligned} \tag{A.27}$$

Similar to the polyhedral LFs method, ρ can be minimized to obtain an optimal convergence rate and a different ρ_i can be assigned to each cone $\mathbf{C}_i$, while (A.27) remains an LP. If the LP (A.27) is feasible, then by construction and the definition of the function $\mathscr{W}$, for all $x \in \mathbf{S}$ it holds that

$$\rho \mathscr{W}(x) - \mathscr{W}(Ax + Bg(x) + c) \geq \alpha_{\bar{x}}(\rho - 1). \tag{A.28}$$

The recursive application of (A.28) implies that for a closed-loop trajectory $x(0), x(1), \ldots$, the following inequality holds for all $k \geq 0$:

$$\mathscr{W}(x(k)) \leq \rho^k \mathscr{W}(x(0)) - \alpha_{\bar{x}} \rho^k + \alpha_{\bar{x}}. \tag{A.29}$$

The above property can be exploited to establish the following result.

Lemma A.1 *Suppose that the LP (A.27) is feasible and let*

$$k^\star := \left\lceil \frac{\ln\left(\frac{\alpha^\star_{\bar{x}} - \alpha_{\bar{x}}}{1 - \alpha_{\bar{x}}}\right)}{\ln(\rho)} \right\rceil. \tag{A.30}$$

Then for all trajectories $x(0), x(1), \ldots$ with $x(0) \in \mathbf{S}$ of system (A.14) in closed-loop with (A.25) there exists a $k \geq 0$ such that $x(k) \in \mathbf{T}$ and, moreover, $k \leq k^\star$.

Proof Let $x(0) \in \mathbf{S}$. To prove the claim consider two cases, i.e., $\mathscr{W}(x(0)) \leq \alpha^\star_{\bar{x}}$ and $\mathscr{W}(x(0)) > \alpha^\star_{\bar{x}}$. Notice that for any $x \in \mathbf{S}$, $\mathscr{W}(x) \leq \alpha^\star_{\bar{x}}$ implies that $x \in (\alpha^\star_{\bar{x}}(\mathbf{S} \oplus \{-\bar{x}\}) \oplus \{\bar{x}\})$ and thus, it implies that $x \in \mathbf{T}$. As such, in the case when $\mathscr{W}(x(0)) \leq \alpha^\star_{\bar{x}}$, the claim holds by the definition of $k^\star$ (A.30).

Next, consider the case when $\mathscr{W}(x(0)) > \alpha^\star_{\bar{x}}$ and suppose that the inequality

$$\alpha^\star_{\bar{x}} < \rho^k \mathscr{W}(x(0)) - \alpha_{\bar{x}} \rho^k + \alpha_{\bar{x}} \tag{A.31}$$

holds for all $k \geq 0$. By taking the limit when k tends to infinity in the above inequality yields that $\alpha^\star_{\bar{x}} < \alpha_{\bar{x}}$ and thus, a contradiction was reached. As such, there must exist a $k \geq 0$ such that

$$\rho^k \mathscr{W}(x(0)) - \alpha_{\bar{x}} \rho^k + \alpha_{\bar{x}} \leq \alpha^\star_{\bar{x}} \tag{A.32}$$

holds. Equations (A.29) and (A.32) imply that $\mathscr{W}(x(k)) \leq \alpha^\star_{\bar{x}}$ and hence, $x(k) \in \mathbf{T}$. Furthermore, reordering the terms and taking the logarithm of both sides in (A.32) yields

$$k \leq \frac{\ln\left(\frac{\alpha^\star_{\bar{x}} - \alpha_{\bar{x}}}{\mathscr{W}(x(0)) - \alpha_{\bar{x}}}\right)}{\ln(\rho)}. \tag{A.33}$$

Noticing that $\mathscr{W}(x) \leq 1$ for all $x \in \mathbf{S}$, the fact that $k \leq k^\star$ follows directly from the definition of $k^\star$ (A.30). As the point $x(0) \in \mathbf{S}$ was chosen arbitrarily, the claim is established. ■

If the LP (A.27) is feasible, the PWA control law (A.25) is an admissible solution to the reach control problem as shown in Lemma A.1. As in polyhedral LFs solution, the evaluation of (A.25) reduces to a point location problem that can be solved in logarithmic time due to the specific conic partition.

The contractive sets method is a relaxation of the Lyapunov function based method. First, $\bar{x}$ is no longer required to be an equilibrium point. Second, the contraction condition of (A.27) is a relaxation of the contraction condition of (A.26). In particular, if $\bar{x}$ is an equilibrium point, then by augmenting the constraints of the LP (A.27) with the equilibrium point constraints, i.e., $(A + BF_i)\bar{x} + Bf_i + c = \bar{x}$, for all $i = 1, \ldots, w$, and setting $\alpha_{\bar{x}}$ to 0, the polyhedral LFs solution is recovered and the function $\mathscr{W}$ becomes a standard Lyapunov function with respect to the equilibrium $\bar{x}$.

A.7 Control Potential Functions

Control potential functions are formally defined in Definition 12.2 in Chap. 12. In this appendix, candidate control potential functions for a transition system $T = (X, \Sigma, \delta, O, o)$ and a dual automaton $A_D = (S_D, S_{D_0}, O_{\mathscr{W}}, \delta_D, \tau_D, F_D)$ with $R_s \subseteq X, \forall s \in S_D$ and transition weight function $\mathbf{W} : S_D \times S_D \rightarrow \mathbb{N}_+$ are presented.

A.7.1 Control Potential Function Based on One Step Controllable Sets

Consider control potential function $V_{con,CS} : \bigcup_{\{(s,s') | s' \in \delta_D(s), s \neq s'\}} \big\{\{(s, s')\} \times R_s\big\} \longrightarrow \mathbb{N}_+$:

$$V_{con,CS}((s, s'), x) = \arg\min\{k \in \{1, \ldots, \mathbf{W}(s, s')\} \mid x \in B_{ss'}^{\leq k-1}\}, \qquad \text{(A.34)}$$

where $B_{ss'}^{k-1}$ denotes the set of states in R_s that can reach the beacon $B_{ss'}$ of (s, s') in $k - 1$ steps, hence $R_{s'}$ in k steps. The set $B_{ss'}^{\leq k-1}$ is computed shown in Sect. A.6.1. Therefore, for $V_{con,CS}$, the constraint set $\mathscr{R}_{V_{con}}^{k,ss'}$ (see (12.4)) is defined as

$$\mathscr{R}_{V_{con,CS}}^{k,ss'} = B_{ss'}^{\leq k-1}.$$

A.7.2 Control Potential Function Based on Feedback Controllers

In this section, a control potential function is presented for each of the control synthesis methods given in Sect. A.6 for solving reach control problem.

A.7.2.1 Vertex Interpolation

Let $g : R_s \rightarrow \mathbf{U}$ be a feedback control law synthesized by using the vertex interpolation method to solve the reach control problem from R_s to $B_{ss'}$, and $\mathbf{W}(s, s')$ be the corresponding time bound. Let $x^i(0), \ldots, x^i(\mathbf{W}(s, s') - 1)$ be the trajectory generated by the feedback control g from the vertex $x^i(0) \in V(R_s)$. Note that $x^i(\mathbf{W}(s, s') - 1) \in B_{ss'}$ for each $x^i(0) \in V(R_s)$.

Consider control potential function $V_{con,VI} : \bigcup_{s \in S_D, s' \in \delta_D(s)} \big\{\{(s, s')\} \times R_s\big\} \longrightarrow \mathbb{N}_+$:

$$V_{con,VI}((s, s'), x) = \mathbf{W}(s, s') - \arg\max\{j \in \{0, \ldots, \mathbf{W}(s, s') - 1\} \mid \\ x \in hull(\{x^i(j)\}_{i=1,\ldots,|V(R_s)|})\}. \tag{A.35}$$

From (12.4) and (A.35), the constraint set for $V_{con,VI}$ is defined as

$$\mathscr{R}^{k,ss'}_{V_{con,VI}} = \bigcup_{j=1,\ldots,k} hull(\{x^i(\mathbf{W}(s, s') - j)\}_{i=1,\ldots,|V(R_s)|}) \tag{A.36}$$

A.7.2.2 Polyhedral LFs

Let a feedback control law be synthesized by using the polyhedral LFs method to solve the reach control problem from R_s to $B_{ss'}$, and let ρ and x^s be the corresponding contraction rate and equilibrium point, respectively.

Consider control potential function $V_{con,PL} : \bigcup_{s \in S_D, s' \in \delta_D(s)} \big\{\{(s, s')\} \times R_s\big\} \longrightarrow \mathbb{N}_+$:

$$V_{con,PL}((s, s'), x) = \mathbf{W}(s, s') - \arg\max\{j \in \{0, \ldots, \mathbf{W}(s, s') - 1\} \mid \\ x \in (\rho^k(R_s \oplus \{-x^s\}) \oplus x^s)\}. \tag{A.37}$$

For $V_{con,PL}$, the constraint set is defined as

$$\mathscr{R}^{k,ss'}_{V_{con,PL}} = \rho^{\mathbf{W}(s,s')-k}(R_s \oplus \{-x^s\}) \oplus x^s \cup B_{ss'}. \tag{A.38}$$

A.7.2.3 Contractive Sets

Let a feedback control law be synthesized by using the contractive sets method to solve the reach control problem from R_s to $B_{ss'}$, and let $\bar{x}$, ρ, $\alpha_{\bar{x}}$, $\alpha^\star_{\bar{x}}$ and the function $\mathscr{W}$ be defined as in (A.28) with respect to $\bar{x}$. For the contractive sets method, a control potential function $V_{con,C} : \bigcup_{s \in S_D, s' \in \delta_D(s)} \big\{\{(s, s')\} \times R_s\big\} \longrightarrow \mathbb{N}_+$ is defined as:

$$V_{con,C}((s,s'),x) = \left\lceil \frac{\ln\left(\frac{\alpha^{\star}_{\bar{x}} - \alpha_{\bar{x}}}{\mathscr{W}(x_0) - \alpha_{\bar{x}}}\right)}{\ln(\rho)} \right\rceil, \tag{A.39}$$

where $\lceil \cdot \rceil$ denotes the ceiling operation. For $V_{con,C}$, the constraint set is defined as

$$\mathscr{R}^{k,ss'}_{V_{con,C}} = \frac{\alpha^{\star}_{\bar{x}} - \alpha_{\bar{x}} + \alpha_{\bar{x}}\rho^{\mathbf{W}(s,s')-k}}{\rho^{\mathbf{W}(s,s')-k}}(R_s \oplus \{-\bar{x}\}) \oplus \bar{x} \cup B_{ss'}. \tag{A.40}$$

References

1. A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 16(6):624–641, 2010.
2. P. Abdulla, K. Cerans, B. Jonsson, and Y. K. Tsay. General decidability theorems for infinite-state systems. In *LICS*, page 313–321. 1996.
3. D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *IEEE Conference on Decision and Control*, Las Vegas, NV, 2016.
4. L. D. Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, LNCS, pages 499–513. Springer-Verlag, 1995.
5. R. Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015.
6. R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *International Conference on Concurrency Theory*, pages 163–178. Springer, 1998.
7. R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88:971–984, 2000.
8. E. Amir and A. Chang. Learning partially observable deterministic action models. *Journal of Articial Intelligence Research*, 33:349–402, 2008.
9. M. Antoniotti and B. Mishra. The supervisor synthesis problem for unrestricted ctl is np-complete, Technical Report, New York University, 1995.
10. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using smt solvers instead of sat solvers. volume 3925 of *Lecture Notes in Computer Science*, pages 46–162, 2006.
11. A. Arnold. *Finite transition systems: semantics of communicating systems*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.
12. E. Asarin, A. Donzé, O. Maler, and D. Nickovic. Parametric identification of temporal properties. In *Runtime Verification*, pages 147–160. Springer, 2012.
13. A. Aziz, T. Shiple, V. Singhal, R. Brayton, and A. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional CTL model checking. *Formal Methods in System Design*, 21:193–224, 2002.
14. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Softw. Eng.*, 29(6):524–541, 2003.
15. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
16. B. Barmish and J. Sankaran. The propagation of parametric uncertainty via polytopes. *IEEE Transactions on Automatic Control*, 24:346–249, 1979.

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7

17. J. Barnat, L. Brim, A. Krejci, A. Streck, D. Safranek, M. Vejnar, and T. Vejpustek. On parameter synthesis by parallel model checking. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99(PrePrints), 2011.
18. J. Barnat, L. Brim, and P. Ročkai. DiVinE 2.0: High-Performance Model Checking. In *Proceedings of the International Workshop on High Performance Computational Systems Biology (HiBi)*, pages 31–32. IEEE Computer Society Press, 2009.
19. E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti. On the robustness of temporal properties for stochastic models. In *Proceedings Second International Workshop on Hybrid Systems and Biology*, volume 125, pages 3–19, 2013.
20. E. Bartocci, L. Bortolussi, L. Nenzi, and G. Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theoretical Computer Science*, 587:3–25, 2015.
21. E. Bartocci, L. Bortolussi, and G. Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Proc. of FORMATS 2014: the 12th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 8711 of *LNCS*, pages 23–37. Springer, 2014.
22. G. Batt, C. Belta, and R. Weiss. Model checking genetic regulatory networks with parameter uncertainty. *In A. Bemporad, A. Bicchi, and G. Butazzo, editors, Tenth International Workshop on Hybrid Systems: Computation and Control (HSCC*, 07(volume4416):61–75.
23. G. Batt, C. Belta, and R. Weiss. Model checking liveness properties of genetic regulatory networks. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 323–338. Springer Berlin/Heidelberg, 2007.
24. G. Batt, C. Belta, and R. Weiss. Temporal logic analysis of gene networks under parameter uncertainty. *IEEE Transactions on Circuits and Systems and IEEE Transactions on Automacit Control (joint special issue on Systems Biology)*, 53:215–229, 2008.
25. G. Batt, B. Yordanov, R. Weiss, and C. Belta. Robustness Analysis and Tuning of Synthetic Gene Networks. *Bioinformatics*, 23(18):2415–2422, 2007.
26. L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.
27. A. Bemporad. Hybrid Toolbox—User's Guide, 2004. http://www.ing.unitn.it/~bemporad/hybrid/toolbox.
28. A. Bemporad, L. Giovanardi, and F. Torrisi. Performance driven reachability analysis for optimal scheduling and control of hybrid systems. In *Proceedings of the 39th IEEE Conference Decision and Control*, pages 969–974, 2000.
29. A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35:407–427, 1999.
30. A. Bhatia, M. Maly, E. Kavraki, and M. Vardi. Motion planning with complex goals. *IEEE Robotics Automation Magazine*, 18(3):55–64, 2011.
31. A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, DAC '99, pages 317–320, New York, NY, USA, 1999. ACM.
32. F. Blanchini. Ultimate boundedness control for uncertain discrete-time systems via set-induced Lyapunov functions. *IEEE Transactions on Automatic Control*, 39(2):428–433, 1994.
33. F. Blanchini, S. Miani, F. Pellegrino, and B. van Arkel. Enhancing controller performance for robot positioning in a constrained environment. *IEEE Transactions on Control Systems Technology*, 16(5):1066–1074, 2008.
34. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. CONCUR*, volume 1243, pages 135–150. Lecture notes in computer Science, 1997.
35. A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In E. Clarke and R. Kurshan, editors, *Computer-Aided Verification*, volume 531 of *Lecture Notes in Computer Science*, pages 197–203. Springer Berlin/Heidelberg, 1991.
36. M. Broucke. Reach control on simplices by continuous state feedback. *SIAM Journal on Control and Optimization*, 48(5):3482–3500, 2010.

37. G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In *Computer Aided Verification (CAV)*, volume 1633 of *LNCS*, page 684. Springer Berlin/Heidelberg, 1999.
38. J. R. Büchi. On a decision method in restricted second order arithmetic. In E. N. et al., editor, *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science 1960*, pages 1–12, Stanford, CA, 1962. Stanford University Press.
39. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2008.
40. K. Chatterjee, A. Gaiser, and J. Kretinsky. Automata with generalized rabin pairs for probabilistic model checking and ltl synthesis. In *Proc. of CAV*, Saint Petersburg, Russia, 2013.
41. M. Chechik and W. Ding. Lightweight Reasoning about Program Correctness. *Information Systems Frontiers*, 4(4):363–377, Dec. 2002.
42. A. Chutinan and B. H. Krogh. Verification of infinite-state dynamic systems using approximate quotient transition systems. *IEEE Transactions on Automatic Control*, 46(9):1401–1410, 2001.
43. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 241–268. Springer Berlin/Heidelberg, 2002.
44. E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(4):583–604, 2003.
45. E. M. M. Clarke, D. Peled, and O. Grumberg. *Model checking*. MIT Press, 1999.
46. S. Coogan, E. A. Gol, M. Arcak, and C. Belta. Traffic network control from temporal logic specifications. *IEEE Transactions on Control of Network Systems*, 3(2):162–171, 2016.
47. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, 2001.
48. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1(2):275–288, 1992.
49. S. Di Cairano, W. P. M. H. Heemels, M. Lazar, and A. Bemporad. Stabilizing dynamic controllers for hybrid systems: A hybrid control lyapunov function approach. *IEEE Transactions on Automatic Control*, 59(10):2629 –2643, 2014.
50. S. Di Cairano, M. Lazar, A. Bemporad, and W. P. M. H. Heemels. A control Lyapunov approach to predictive control of hybrid systems. In *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 130–143. Springer Verlag, 2008.
51. X. Ding, M. Lazar, and C. Belta. Ltl receding horizon control for finite deterministic systems. *Automatica*, 50(2):399–408, 2014.
52. X. Ding, S. L. Smith, C. Belta, and D. Rus. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, 2014.
53. A. Donzè, G. Clermont, A. Legay, and C. Langmead. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. In S. Batzoglou, editor, *Research in Computational Molecular Biology*, volume 5541 of *Lecture Notes in Computer Science*, pages 155–169. Springer Berlin/Heidelberg, 2009.
54. A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.
55. E. A. Emerson. Automata, tableaux and temporal logics (extended abstract). In *Proceedings of the Conference on Logic of Programs*, pages 79–88, 1985.
56. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, pages 995–1072. North-Holland Pub. Co./MIT Press, 1990.
57. E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

58. J. Esparza, A. Kucera, and S. Schwoon. Model checking ltl with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355–376, 2003.
59. G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.
60. M. Falt, V. Raman, and R. M. Murray. Variable elimination for scalable receding horizon temporal logic planning. In *American Control Conference*, Chicago, IL, 2015.
61. C. Finucane, G. Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, 2010.
62. G. Frehse, S. Jha, and B. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In M. Egerstedt and B. Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 187–200. Springer Berlin/Heidelberg, 2008.
63. M. D. B. G Pola, A Borri. Integrated design of symbolic controllers for nonlinear systems. *IEEE Transactions on Automatic Control*, 57(2):534–539, 2012.
64. A. Garulli, S. Paoletti, and A. Vicino. A survey on switched and piecewise affine system identification. *IFAC Proceedings Volumes*, 45(16):344–355, 2012.
65. P. Gastin and D. Oddoux. Fast ltl to büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin/Heidelberg, 2001.
66. R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, UK, 1996. Chapman & Hall, Ltd.
67. A. Girard and G. Pappas. Approximate bisimulation relations for constrained linear systems. *Automatica*, 43:1307–1317, 2007.
68. A. Girard and G. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.
69. A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2010.
70. E. A. Gol and C. Belta. An additive cost approach to optimal temporal logic control. In *American Control Conference (ACC)*, Portland, USA, 2014.
71. E. A. Gol, X. Ding, M. Lazar, and C. Belta. Finite bisimulations for switched linear systems. *IEEE Transactions on Automatic Control*, 59(12):3122–3134, 2014.
72. E. A. Gol, X. C. Ding, M. Lazar, and C. Belta. Finite bisimulations for switched linear systems. In *IEEE Conference on Decision and Control (CDC) 2012*, Maui, Hawaii, 2012.
73. E. A. Gol and M. Lazar. Temporal logic model predictive control for discrete-time systems. In *Hybrid Systems: Computation and Control (HSCC)*, Philadelphia, PA, 2013.
74. E. A. Gol, M. Lazar, and C. Belta. Language-guided controller synthesis for discrete-time linear systems. In *15th International Conference on Hybrid Systems: Computation and Control (HSCC)*, Beijing, China, 2012.
75. E. A. Gol, M. Lazar, and C. Belta. Language-guided controller synthesis for discrete-time linear systems. In *Hybrid Systems: Computation and Control*, pages 95–104. ACM, 2012.
76. E. A. Gol, M. Lazar, and C. Belta. Language-guided controller design for linear systems. *IEEE Transactions on Automatic Control*, 59(5):1163–1176, 2014.
77. E. A. Gol, M. Lazar, and C. Belta. Temporal logic model predictive control. *Automatica*, 56:78–85, 2015.
78. P. Grieder. *Efficient computation of feedback controllers for constrained systems*. Ph.d. dissertation, ETH Zürich, 2004.
79. B. Grünbaum. *Convex polytopes*. Springer Verlag, 2003.
80. L. Habets and J. van Schuppen. A control problem for affine dynamical systems on a full-dimensional polytope. *Automatica*, 40(1):21–35, 2004.

81. D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8:231–274, June 1987.
82. W. P. M. H. Heemels, B. D. Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001.
83. T. A. Henzinger. Hybrid automata with finite bisimulations. volume 944 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 1995.
84. T. A. Henzinger and P.-H. Ho. Hytech: The cornell hybrid technology tool. volume 999 of *Lecture Notes in Computer Science*, pages 265–294. Springer, 1995.
85. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. *Journal of Computer and System Sciences*, 57:94–124, 1998.
86. T. A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the book grow out of a Dagstuhl Seminar, June 1995).*, pages 265–282, London, UK, 1996. Springer-Verlag.
87. J. Hespanha. Modelling and analysis of stochastic hybrid systems. *IEEE Proceedings on Control Theory and Applications*, 153(5):520–535, 2006.
88. C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.
89. G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 25(5):279–295, May 1997.
90. F. Horn. Streett Games on Finite Graphs. In the *2nd Workshop on Games in Design and Verification (GDV)*, 2005.
91. S. Jacksic, E. Bartocci, R. Grosu, and D. Nickovic. Quantitative monitoring of STL with edit distance. In *Proc. of RV 2016: the 7th International Conference on Runtime Verification*, LNCS, page to appear. Springer, 2016.
92. S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specifications. *SIAM Journal on Control and Optimization*, 44(6):2079–2103, 2006.
93. Z.-P. Jiang and Y. Wang. A converse Lyapunov theorem for discrete-time systems with disturbances. *Systems & Control Letters*, 45:49–58, 2002.
94. X. Jin, A. Donze, J. Deshmukh, and S. Seshia. Mining requirements from closed-loop control models. In *Hybrid Systems: Computation and Control (HSCC)*, 2013.
95. A. Jones, Z. Kong, and C. Belta. Anomaly detection in cyber-physical systems: A formal methods approach. In *IEEE Conference on Decision and Control 2014*.
96. A. A. Julius and G. J. Pappas. Approximations of stochastic hybrid systems. *IEEE Transactions on Automatic Control*, 54(6), 2009.
97. A. L. Juloski, W. P. M. H. Heemels, G. Ferrari-Trecate, R. Vidal, S. Paoletti, and J. H. G. Niessen. Comparison of four procedures for the identification of hybrid systems. In *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control (HSCC)*, LNCS, pages 354–369, 2005.
98. L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.
99. P. C. Kanellakis and S. A. Smolka. CCS expressions, finite-state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.
100. S. Karaman, R. G. Sanfelice, and E. Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *Proceedings of the 47th IEEE Conference on Decision and Control (CDC)*, pages 2117–2122, 2008.
101. S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.
102. J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theoretical Computer Science*, 363(2):182–195, 2006.
103. M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from ltl specifications. In J. Hespanha and A. Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *Lecture Notes in Computer Science*, pages 333–347. Springer Berlin/Heidelberg, 2006.

104. M. Kloetzer and C. Belta. Dealing with non-determinism in symbolic control. In *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 287–300. Springer Berlin/Heidelberg, 2008.
105. M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, feb. 2008.
106. M. Kloetzer and C. Belta. Reachability analysis of multi-affine systems. *Transactions of the Institute of Measurement and Control, special issue on Hybrid Systems*, 2009.
107. Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta. Temporal logic inference for classification and prediction from data. In *The 17th International Conference on Hybrid Systems: Computation and Control (HSCC)*, Berlin, Germany, 2014.
108. R. Koymans. Specifying real-time properties with metric temporal logic. *Real- Time Systems*, 2(4):255–299, 1990.
109. H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.
110. T. Kropf. *Introduction to Formal Hardware Verification: Methods and Tools for Designing Correct Circuits and Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1999.
111. O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19:291–314, 2001.
112. M. Kvasnica. *Efficient Software Tools for Control and Analysis of Hybrid Systems*. PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, 2008.
113. M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004.
114. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic realtime systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
115. M. Lahijanian, S. B. Andersson, and C. Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Transactions in Automatic Control*, 6(8):2031–2045, 2015.
116. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
117. T. Latvala. Efficient model checking of safety properties. In *In Model Checking Software. 10th International SPIN Workshop*, pages 74–88. Springer, 2003.
118. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
119. M. Lazar. *Model predictive control of hybrid systems: Stability and robustness*. PhD thesis, Eindhoven University of Technology, The Netherlands, 2006.
120. M. Lazar. On infinity norms as Lyapunov functions: Alternative necessary and sufficient conditions. In *IEEE Conf. on Decision and Control*, pages 5936 –5942, 2010.
121. M. Lazar, W. P. M. H. Heemels, S. Weiland, and A. Bemporad. Stabilizing model predictive control of hybrid systems. *IEEE Transactions on Automatic Control*, 51(11):1813–1818, 2006.
122. M. Lazar and A. Jokić. On infinity norms as Lyapunov functions for piecewise affine systems. In *Hybrid Systems: Computation and Control*, pages 131–140, USA, 2010. ACM.
123. E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. http://leeseshia.org, second edition, 2015.
124. H. Lin and P. Antsaklis. Stability and stabilizability of switched linear systems: A survey of recent results. *IEEE Transactions on Automatic Control*, 54(2):308–322, 2009.
125. H. Lin and P. J. Antsaklis. Stability and stabilizability of switched linear systems: A survey of recent results. *IEEE Transactions on Automatic Control*, 54(2):308–322, 2009.
126. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 71–76, 2004.
127. D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.
128. M. Mazo, A. Davitian, and P. Tabuada. PESSOA: A Tool for Embedded Controller Synthesis. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 566–569. Springer Berlin/Heidelberg, 2010.

129. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
130. M. A. Memon. Computational logic: Linear-time vs. branching-time logics. Graduate Course Notes, Simon Fraser University, Burnaby, Canada, April 2003.
131. R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
132. R. Milner. *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.
133. J. C. Mitchell. Discrete uniform sampling of rotation groups using orthogonal images. *SIAM Journal of Scientific Computing*, 30(1):525–547, 2007.
134. T. Motzkin, H. Raiffa, G. Thompson, and R.M. Thrall. The double description method. In H. Kuhn and A. Tucker, editors, *Contributions to Theory of Games*, volume 2. Princeton University Press, 1953.
135. S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee. POMDPs for robotic tasks with mixed observability. In *Robotics: Science and Systems*, 2009.
136. N. Ozay, J. Liu, P. Prabhakar, and R. M. Murray. Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems. In *American Control Conference*, pages 6237–6244, June 2013.
137. O. Pãun and M. Chechik. On Closure Under Stuttering. *Formal Aspects of Computing*, 14(4):342–368, Apr. 2003.
138. G. J. Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, 2003.
139. D. A. Peled and T. Wilke. Stutter-invariant temporal properties are expressible without the next-time operator. *Information Processing Letters*, 63(5):243–246, Sept. 1997.
140. J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.
141. N. Piterman and A. Pnueli. Faster solutions of rabin and streett games. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 275–284, Washington, DC, USA, 2006. IEEE Computer Society.
142. N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Proc. VMCAI*, pages 364–380. 2006.
143. N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.
144. A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010.
145. G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008.
146. M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.
147. S. V. Rakovic, F. Blanchini, E. Cruck, and M. Morari. Robust obstacle avoidance for constrained linear discrete time systems: A set-theoretic approach. In *IEEE Conf. on Decision and Control*, pages 188–193, 2007.
148. S. V. Rakovic and D. Mayne. Robust Model Predictive Control for Obstacle Avoidance: Discrete Time Case. *Lecture Notes in Control and Information Sciences (LNCIS)*, 358:617–627, Sept. 2007.
149. V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC 2015)*, pages 239–248, April 2015.
150. J. B. Rawlings and D. Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.
151. A. Rodionova, E. Bartocci, D. Nickovic, and R. Grosu. Temporal logic as filtering. In *Proc. of HSCC'16: the 19th International Conference on Hybrid Systems: Computation and Control*, pages 11–20. ACM, 2016.
152. M. Rungger, M. Mazo, and P. Tabuada. Specification-guided controller synthesis for linear systems and safe linear-time temporal logic. In *16th International Conference on Hybrid Systems: Computation and Control*, pages 333–342, Philadelphia, PA, 2013.

153. S. Safra. On the complexity of omega-automata. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 319–327, 1988.
154. M. Schaub, T. Henzinger, and J. Fisher. Qualitative networks: a symbolic approach to analyze biological signaling networks. *BMC Systems Biology*, 1(1):4, 2007.
155. T. Schlipf, T. Buechner, R. Fritz, M. Helms, and J. Koehl. Formal verification made easy. *IBM Journal of Research and Development*, 41(4-5):567–576, 1997.
156. A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.
157. C. Sloth and R. Wisniewski. Verification of continuous dynamical systems by timed automata. *Formal Methods in System Design*, 39:47–82, 2011.
158. S. L. Smith, J. Tumova, C. Belta, and D. Rus. Optimal path planning for surveillance with temporal-logic constraints. *Int. J. Rob. Res.*, 30(14):1695–1708, 2011.
159. Z. Sun. *Switched Linear Systems: Control and Design*. Springer, 2005.
160. M. Svorenova, I. Cerna, and C. Belta. Optimal control of mdps with temporal logic constraints. In *52nd IEEE Conference on Decision and Control (CDC)*, Firenze, Italy, 2013.
161. M. Svorenova, I. Cerna, and C. Belta. Optimal temporal logic control for deterministic transition systems with probabilistic penalties. *IEEE Transactions on Automatic Control*, 60(6):1528–1541, 2015.
162. P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.
163. P. Tabuada and G. Pappas. Model checking LTL over controllable linear systems is decidable. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control*, volume 2623 of *Lecture Notes in Computer Science*, pages 498–513. Springer-Verlag, 2003.
164. P. Tabuada and G. Pappas. Linear time logic control of discrete-time linear systems. 51(12):1862–1877, 2006.
165. P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.
166. R. Thomas. Regulatory networks seen as asynchronous automata: a logical description. *Journal of Theoretical Biology*, 153:1–23, 1991.
167. W. Thomas. Infinite games and verification. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 58–65. Springer Berlin/Heidelberg, 2002.
168. A. Tiwari and G. Khanna. Series of Abstractions for Hybrid Automata. In C. Tomlin and M. Greenstreet, editors, *Hybrid Systems: Computation and Control*, volume 2289 of *Lecture Notes in Computer Science*, pages 425–438. Springer Berlin/Heidelberg, 2002.
169. P. Tøndel, T. A. Johansen, and A. Bemporad. Evaluation of piecewise affine control via binary search tree. *Automatica*, 39(5):945–950, 2003.
170. J. Tůmová, B. Yordanov, C. Belta, I. Černá, and J. Barnat. A symbolic approach to controlling piecewise affine systems. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 4230–4235, Atlanta, GA, Dec. 2010.
171. A. Ulusoy and C. Belta. Receding horizon temporal logic control in dynamic environments. *The International Journal of Robotics Research*, 33(12):1593–1607, 2014.
172. M. Vardi. Branching vs. linear time: Final showdown. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, London, UK, 2001.
173. P. Wolper, M. Vardi, and A. Sistla. Reasoning about infinite computation paths. In E. N. et al., editor, *IEEE Symposium on Foundations of Computer Science*, pages 185–194, Tucson, AZ, 1983.
174. T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray. Tulip: Software toolbox for receding horizon temporal logic planning. In *International Conference on Hybrid Systems: Computation and Control*, Chicago, Illinois, 2011.
175. T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon temporal logic planning for dynamical systems. In *IEEE conf. on decision and control*, Shanghai, China, 2009.
176. H. Yang, B. Hoxha, and G. Fainekos. Querying parametric temporal logic properties on embedded systems. In *Testing Software and Systems*, pages 136–151. Springer, 2012.

177. B. Yordanov and C. Belta. Formal analyis of piecewise affine systems under parameter uncertainty with application to gene networks. In *American Control Conference*, Seattle, Washington, 2008.

178. B. Yordanov and C. Belta. Parameter synthesis for piecewise affine systems from temporal logic specifications. In M. Egerstedt and B. Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 542–555. Springer Berlin/Heidelberg, 2008.

179. B. Yordanov and C. Belta. Temporal logic control of discrete-time piecewise affine systems. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 3182–3187, Shanghai, China, Dec. 2009.

180. B. Yordanov and C. Belta. Formal analysis of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 55(12):2834–2840, 2010.

181. B. Yordanov and C. Belta. Formal analysis of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 55(12):2834–2841, December 2010.

182. B. Yordanov, C. Belta, and G. Batt. Model checking discrete time piesewise affine systems: application to gene networks. In *European Control Conference*, Kos, Greece, 2007.

183. B. Yordanov, J. Tůmová, C. Belta, I. Černá, and J. Barnat. Formal analysis of piecewise affine systems through formula-guided refinement. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 5899–5904, Atlanta, GA, Dec 2010.

184. B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta. Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 57:1491–1504, 2012.

185. B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta. Formal analysis of piecewise affine systems through formula-guided refinement. *Automatica*, 49(1):261–266, 2013.

186. M. Zamani, P. M. Esfahani, R. Majumdar, A. Abate, and J. Lygeros. Symbolic control of stochastic systems via approximately bisimilar finite abstractions. *IEEE Transactions on Automatic Control*, 59(12):3135–3150, 2014.

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

188. P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian Statistical Model Checking with Application to Simulink/Stateflow Verification. In *International Conference on Hybrid systems: Computation and Control (HSCC) 2010*, pages 243–252, 2010.

Index

C. Belta et al., *Formal Methods for Discrete-Time Dynamical Systems*,
Studies in Systems, Decision and Control 89, DOI 10.1007/978-3-319-50763-7