UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

MASTER DEGREE IN CONTROL SYSTEMS ENGINEERING

ABSTRACTION-BASED DATA-DRIVEN CONTROL

Candidate
Davide De Lazzari

Supervisor
Prof. Ruggero Carli

Co-supervisor
Prof. Raphaël Jungers
(UCLouvain)

ACADEMIC YEAR 2021/2022
Graduation Date: 5th September 2022

This thesis has been conducted at the Université catholique de Louvain under the supervision of Prof. Raphaël Jungers while collaborating to the development of the Dionysos software [1].

Hereby, I would like to thank Raphaël and all his team for their support during my stay at UClouvain.

Abstract

Our world is living a paradigm shift in technology policy, often referred to as the Cyber-Physical Revolution or Industry 4.0.

Nowadays, Cyber-Physical Systems are ubiquitous in modern control engineering, including automobiles, aircraft, building control systems, chemical plants, transportation systems, and so on. The interactions of the physical processes with the machines that control them are becoming increasingly complex, and in a growing number of situations either the model of the system is unavailable, or it is too difficult to describe accurately. Therefore, embedded computers need to *learn* the optimal way to control the systems by the mere observation of data.

What seems the best approach to control these complex systems is often by discretizing the different variables, thus transforming the model into a combinatorial problem on a finite-state automaton, which is called an abstraction of the real system.

Until now, this approach, often referred to as *abstraction-based control* or *symbolic control*, has not been proved useful beyond small academic examples.

In this project I aim to show the potential of this approach by implementing a novel data-driven approach based on a probabilistic interpretation of the discretization error.

I have developed a toolbox (github.com/davidedl-ucl/master-thesis) implementing this kind of control with the aim of integrating it in the Dionysos software (github.com/dionysos-dev).

With this software, I succeeded in efficiently solving problems for non-linear control systems such as a path planning for an autonomous vehicle and a cart-pole balancing problem.

The long-term objective of this project is to improve the methods implemented in my current software by employing a variable discretization of the state space and to consider complex specifications such as LTL formulas.

Contents

List of Figures

List of Algorithms

List of Definitions

1

Introduction

Our world is living a paradigm shift in technology policy, often referred to as the Cyber-Physical Revolution or Industry 4.0.

Nowadays, *cyber-physical systems* (CPSs) are ubiquitous in modern control engineering, including automobiles, aircraft, building control systems, chemical plants, transportation systems, and so on. The interactions of the physical processes with the machines that control them are becoming increasingly complex, and in a growing number of situations either the model of the system is unavailable, or it is too difficult to describe accurately. However, at the same time, modern electronic devices can sense, process, and store a huge amount of data. Therefore, embedded computers need to *learn* the optimal way to control the systems by the mere observation of data.

Moreover, many CPSs are safety critical or mission critical: it must ensure that the system operates correctly meeting the satisfaction of safety or some desired specifications. Formal methods are known to provide essential tools for the design of CPSs, as they give theoretical or rigorous mathematical proofs that the system works correctly meeting the desired specification [1]. While the formal methods have been originally developed in software engineering that aims at finding bugs or security vulnerabilities in the software, the methodologies have been recently recognized to be useful in other applications, including the control design of CPSs. In particular, one of the most successful methods that interface the formal methods and the control design of CPSs is the so-called *symbolic control*, see, e.g., [[2]].

As a matter of fact, generalizing classical control techniques based on frequency analysis or convex optimization to hybrid systems is challenging. In the literature, some works achieve this for specific classes of hybrid systems but this is usually limited to systems for which the discrete part is not too complex. On the other hand, symbolic control relies on the relation between the system and an automaton to leverage algorithms on graphs or hyper-graphs. In such approach, if the control system (or part of it) is represented in a continuous-state (and input) space, a symbolic model

for the system can be obtained by discretizing the different variables, thus transforming the original model into a combinatorial problem on a finite-state automaton, which is called an *abstraction* of the real system. Even though symbolic controllers (also known as *abstraction-based controllers*) suffer from the curse of dimensionality, their complexity is less affected by the complexity of the discrete part. Therefore, the main objective of abstraction-based control is to design controllers for CPSs with logic specifications.

This research project was conducted at the Université Catholique de Louvain (UCLouvain) under the supervision of Prof. Raphaël Jungers while collaborating to the development of the *Dionysos* software [1]. The software is part of the European Research Council (ERC) project *Learning to Control – Smart and Data-Driven Formal Methods for Cyber-Physical Systems Control* (L2C). Therefore, my research project was carried on by working in close contact with Prof. Jungers's team (and in particular with the Dionysos dev team) with the final aim of developing a toolbox able to control non-linear systems via abstraction-based techniques. More specifically, Dionysos's objective is to learn the optimal control CPSs from data, whether harvested from the physical system or generated synthetically. The software will rely on a novel methodology, combining the efficiency of several modern optimization, control-theoretic, and machine-learning techniques with the theoretical power of the abstraction approach. All the pieces of the architecture are chosen to foster black-box and data-driven analysis, thereby matching rising and unresolved challenges.

This paper will be structured as follows:

- Firstly, I will introduce the most common abstraction-based strategies to design a controller for non-linear systems with a continuous-state (and input) space;
- Then, I will present a novel way to obtain an abstraction using heuristics, namely replacing the non-determinism resulting from the discretization by adding probabilities on the transitions in order to build a probabilistic abstraction.
- Thus, I will describe the methods that I have employed in my software and its structure.
- Finally, I will introduce the main examples on which I tested my toolbox and present my experimental results.

2

The Typical Abstraction-Based Control

As previously mentioned, abstraction-based control, a.k.a. symbolic control, can be an effective approach for the control design of CPSs. CPSs are usually modeled as hybrid systems and therefore present continuous state and input spaces. Roughly speaking, those variables are discretized to obtain a symbolic model (*abstraction*) based on the original (*concrete*) control system. The abstraction is constructed preserving the behavior of the concrete system, therefore a symbolic controller is designed based on the abstract system and then refined as a controller for the concrete system.

Abstraction-based control is known to be a powerful tool in the following three ways:

- it allows us to synthesize controllers for general non-linear dynamical systems (that may also present hybrid behavior) with state and input constraints;
- by constructing the symbolic model, we can take into account the constraints that are imposed on the cyber part with regard to the digital platform, such as a quantization effect;
- it allows us to synthesize controllers under various control specifications, including safety, reachability, or more complex ones such as those expressed by linear temporal logic formulas or automata on infinite strings. In particular temporal logics have well-defined syntax and semantics, which can be easily used to specify complex behavior. It has been shown [3, 4] that it is possible to convert an LTL formula into an equivalent automaton. Therefore, thanks to this, the automaton could be combined with another one representing an abstract system.

Roughly speaking, an abstraction-based feedback system usually works as follows:

1. first, the state of the real system is estimated
2. then, the state of the real system (*concrete* state) is mapped into a symbolic one (*abstract* state) thanks to an existing relation between the systems
3. then, a symbolic controller is derived based solely on the abstraction
4. finally, the output of the abstract controller is refined into an input for the concrete system.

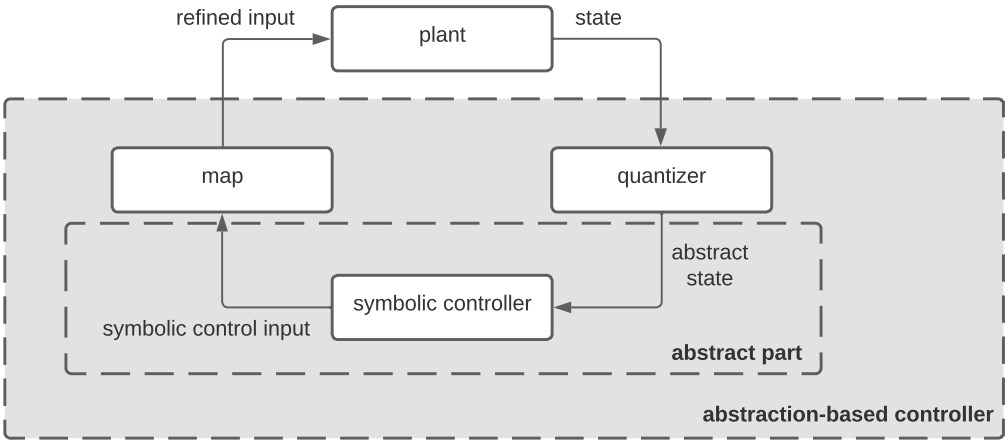

Figure 2.1: Basic abstraction-based feedback control scheme

The mapping procedure from a real to a symbolic state usually consists of a quantization, however, behavioural relationships, such as the concept of *approximate (bi-)simulation relation* [5], are used to relate the behaviours of the original control system and its symbolic model. Moreover, in order to obtain abstractions and relations with formal guarantees, a mathematical description of the system is required.

Most of the times, the computation of abstractions is essentially reduced to the over-approximation of the reachable sets of the original system, i.e., the computation of a bounding set, encompassing the attainable states, from each particular *cell*[1]. A large number of over-approximation methods have been proposed, e.g. [6], [7], [8].

[1] A set of concrete states (usually consisting of a hyper-interval) that map to the same symbolic state

However, there is a trade-off between computing a high precision over-approximation of the reachable set, leading to a less conservative abstraction, and the computation time. A method proposed in [4] is based on the computation of a growth-bound function, which is a bound on the discretization error based on continuity arguments. While this function has the advantage of being a local estimate and of depending on the input used, it can also be very conservative. It requires narrow bounds on the partial derivatives on an a-priori enclosure of the trajectory. Moreover, it provides an over-approximation component-wise, which can in some situations lead to a bad approximation as shown in Fig. 2.2.

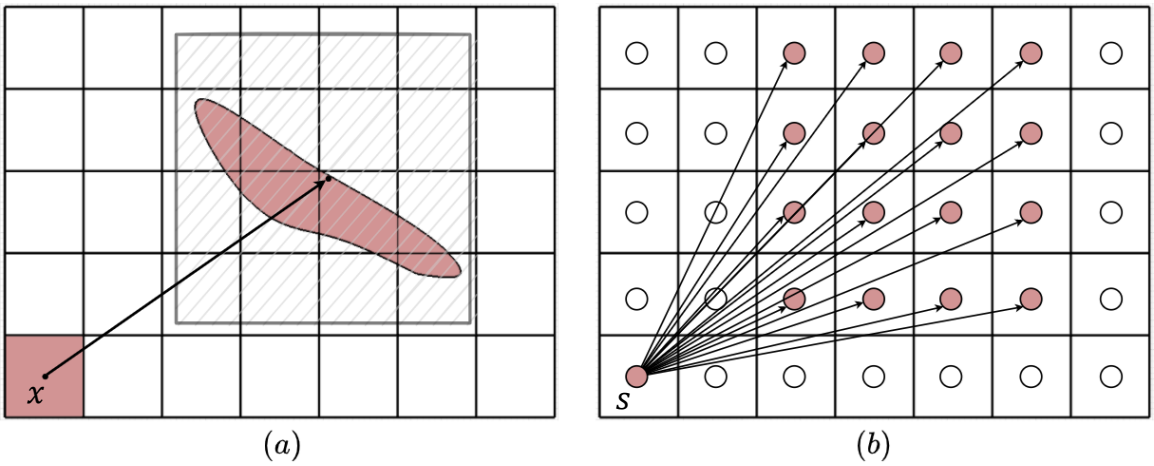

Figure 2.2: Classical abstraction approach. In this example an autonomous system with a 2-dimensional state-space is partitioned into cells (Fig. a). The cell containing the concrete state x transitions to the pink-colored part. However, due to a component-wise over-approximation, the reachable state space is considered as the grey hyper-rectangle. Therefore, in Fig. b, we can notice that the corresponding symbolic state s has a larger amount of successors compared to an abstraction obtained with an exact-computation of the reachable set, leading to a high degree of conservatism.

3

The Data-Driven Approach

3.1 Preliminaries

The interactions of the physical processes with the machines that control them are becoming increasingly complex, and in a growing number of situations either the model of the system is unavailable or it is too difficult to describe accurately. In view of thee specificities, one needs to learn how to control the systems by the mere observation of data.

3.1.1 Systems

Among the many different mathematical models used to describe dynamical phenomena we are especially interested in models with states belonging to finite sets, infinite sets, and combinations thereof. By a *finite-state system* we mean a system described by finitely many states (a.k.a. symbolic). We also consider *infinite-state systems* described by difference or differential equations with solutions evolving in infinite sets such as $\mathbb{R}^n$.

The notion of *transition system* [5] allows us to describe the concrete dynamical system and its symbolic abstraction in a unified framework. Hybrid systems, combining aspects o finite-state and infinite-state systems, consist of another class of systems that can still be described by this notion.

Definition 3.1.1 (Transition system)

A transition system (or simply a system) $\mathcal{S}$ is a sextuple $(X, X_0, U, \rightarrow, Y, H)$ consisting of:

- *a set of states X;*
- *a set of initial states $X_0 \subseteq X$;*

- *a set of inputs U;*
- *a transition relation $\rightarrow \subseteq X \times U \times X$;*
- *a set of outputs Y;*
- *an output map $H : X \rightarrow Y$.*

The evolution of a system is captured by the transition relation. We will denote a transition $(x, u, x') \in \rightarrow$ as $x \xrightarrow{u} x'$. For such a transition, state x' is called a *u-successor*, or simply *successor*, of state x. Similarly, x is called a *u-predecessor*, or *predecessor*, of state x'. Note that, since $\rightarrow \subseteq X \times U \times X$ is a relation, for any state and any input $u \in U$ there may be: no u-successors, one u-successor, or many u-successors. For conciseness, we denote the set of u-successors of a state x by $Post_u(x)$.

Definition 3.1.2 (Finite system)

A transition system is said to be finite (or symbolic), if sets X and U are finite.

Definition 3.1.3 (Deterministic system)

A transition system is said to be deterministic, if there exists at most one u-successor of x, for any $x \in X$ and $u \in U$

Moreover, in this project we will study only *simple* systems, namely the state is directly measurable and it coincides with the output of the system.

Definition 3.1.4 (Simple system)

A transition system is said to be simple if $X = Y$, $H = id$, and all states are admissible as initial states, i.e., $X = X_0$

3.1.2 Controllers

Definition 3.1.5 (Control policy)

A control policy (or controller) for the transition system $(X, X_0, U, \rightarrow, Y, H)$ is a pair $(X_{\mathcal{C}}^0, \mathcal{C})$ consisting of:

- *a set of initial states $X_{\mathcal{C}}^0$;*
- *a control law $\mathcal{C} : X^{\leq N} \rightarrow U$ that takes a finite state sequence $x(0), x(1), \ldots, x(k)$ and outputs a control $u(k)$ for all time $k \in \mathbb{N}$.*[1]

[1] Given $N \in \mathbb{N}_{>0}$ and a metric space X, $X^{\leq N}$ denotes the set of finite sequences of elements of X with length bounded by N

In this project we will consider only *static* controllers, namely the control input depends solely on the current state.

Definition 3.1.6 (Static control policy)

A static controller $(X_{\mathcal{C}}^0, \mathcal{C})$ *is a control policy such that the control law* $\mathcal{C} : X \to U$ *takes the current state* $x(k)$ *and outputs a control* $u(k)$*.*

3.1.3 Relations

As previously mentioned, in order to couple the behavior of two systems we need to resort to some notion of relation.

Given a relation $\mathcal{R} \subseteq X_1 \times X_2$ and $x_1 \in X_1$ we define $\mathcal{R}(x_1) = \{x_2 \in X_2 \mid (x_1, x_2) \in \mathcal{R}\}$. Similarly, for $x_2 \in X_2$ we define $\mathcal{R}^{-1}(x_2) = \{x_1 \in X_1 \mid (x_1, x_2) \in \mathcal{R}\}$.

The most common relation between systems used in abstraction-based control is the one of *alternating simulation relation* [5].

Definition 3.1.7 (Alternating Simulation Relation)

Let S_1 *and* S_2 *be systems with* $Y_1 = Y_2$*. A relation* $\mathcal{R} \subseteq X_1 \times X_2$ *is an alternating simulation relation from* S_1 *to* S_2 *if the following three conditions are satisfied:*

1. *for every* $x_{10} \in X_{1,0}$ *there exists* $x_{20} \in X_{2,0}$ *with* $(x_{10}, x_{20}) \in \mathcal{R}$*;*
2. *for every* $(x_1, x_2) \in R$ *we have* $H_1(x_1) = H_2(x_2)$*;*
3. *for every* $(x_1, x_2) \in R$ *and for every* $u_1 \in U_1(x_1)$ *there exists* $u_2 \in U_2(x_2)$ *such that for every* $x_2' \in \mathrm{Post}_{u_2}(x_2)$ *there exists* $x_1' \in \mathrm{Post}_{u_1}(x_1)$ *satisfying* $(x_1', x_2') \in R$*.*

Definition 3.1.8 (Alternating Simulation)

Given two systems $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *with* $Y_1 = Y_2$*, we say that* $\mathcal{S}_1$ *is alternatingly simulated by* $\mathcal{S}_2$ *or that* $\mathcal{S}_2$ *alternatingly simulates* $\mathcal{S}_1$*, denoted by* $\mathcal{S}_1 \preceq_{AS} \mathcal{S}_2$*, if there exists an alternating simulation relation (Def. 3.1.7) from* $\mathcal{S}_1$ *to* $\mathcal{S}_2$*.*

3.2 A Basic Data-Driven Approach

Suppose we aim to control a simple continuous-state discrete-time system

$$\begin{gathered} \mathcal{S}_1 = (X_1, X_{1,0}, U_1, \rightarrow_1, Y_1, H_1) \\ \text{where} \\ X_1 \subseteq \mathbb{R}^N, \quad X_{1,0} = \{x_0\}, \quad U_1 \subseteq \mathbb{R}^M, \\ (x, u, x') \in \rightarrow \quad \Longleftrightarrow \quad x' = F(x), \\ Y_1 = X_1, \quad H_1 = id \end{gathered} \tag{3.1}$$

Moreover assume X and U bounded. For simplicity, we can also consider them to be hyper-intervals.

The abstraction is constructed as follows:

1. First, we need to partition the domains. Therefore, X_1 and U_1 are divided into smaller hyper-rectangles of the same length, called *cells*.

2. Then a symbolic domain X_2 is constructed by mapping each cell $X_{1,i}$ of X_1 into an element $x_{2,i}$ of X_2. The same is done with $X_{1,0}$, thus obtaining a symbolic set $X_{2,0}$, and U_1 to construct the abstract domain U_2.

   By doing so, we have defined the maps

$$\mathcal{R}_X(x) \doteq x_{2,i} \quad \forall\ x \in X_{1,i} \tag{3.2}$$

$$\mathcal{R}_U(u) \doteq u_{2,j} \quad \forall\ u \in U_{1,j} \tag{3.3}$$

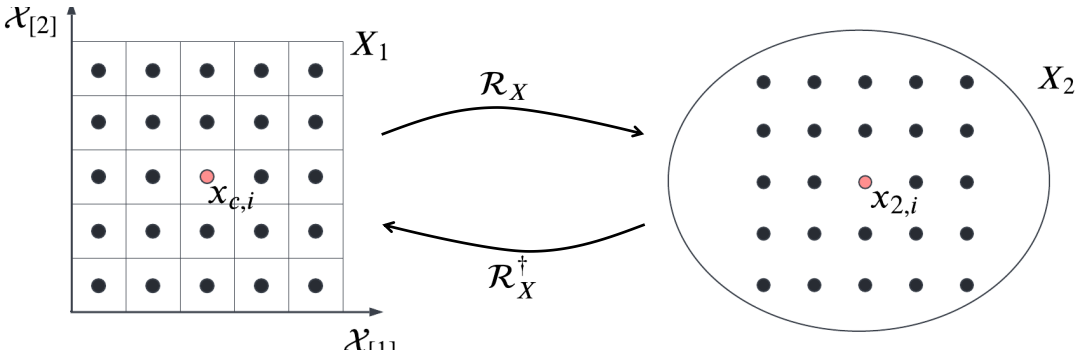

Figure 3.1: Concrete-to-abstract domain mapping (and vice-versa). An example of a 2-dimensional state space X_1 divided into cells $X_{1,i}$, each of which is related to a symbolic state $x_{2,i}$ according to the map $\mathcal{R}_X$. Another map $\mathcal{R}_X^\dagger$ maps each symbol into the center of the corresponding cell.

3. Once constructed the domains we need to define a transition relation $\rightarrow_2 \subseteq X_2 \times U_2 \times X_2$.

   We call $x_{c,i}$ the center of the hyper-rectangle $X_{1,i}$ and $u_{c,j}$ the center of the hyper-rectangle $U_{1,j}$, thus we can define the maps

$$\mathcal{R}_X^\dagger(x) \doteq x_{c,i} \quad \forall\, x \quad \text{s.t.} \quad x = R_X(x_{c,i}) \tag{3.4}$$

$$\mathcal{R}_U^\dagger(u) \doteq u_{c,j} \quad \forall\, u \quad \text{s.t.} \quad u = R_U(u_{c,j}) \tag{3.5}$$

   Then we construct the transition relation $\rightarrow_2$:

$$\forall\,(x_{c,i}, u_{c,j}, x_1') \in \rightarrow_1, \tag{3.6}$$

$$(x_2, u_2, x_2') \in \rightarrow_2 \iff \{x_2 = \mathcal{R}_X(x_{c,i}),\ u_2 = \mathcal{R}_U(u_{c,j}),\ x_2' = \mathcal{R}_X(x_1')\}$$

4. Finally, we can define the abstraction as

$$\begin{gathered}\mathcal{S}_2 = (X_2, X_{2,0}, U_2, \rightarrow_1, Y_2, H_2) \text{ where} \\ Y_2 = Y_1 \text{ and } H_2 = \mathcal{R}_X^\dagger\end{gathered} \tag{3.7}$$

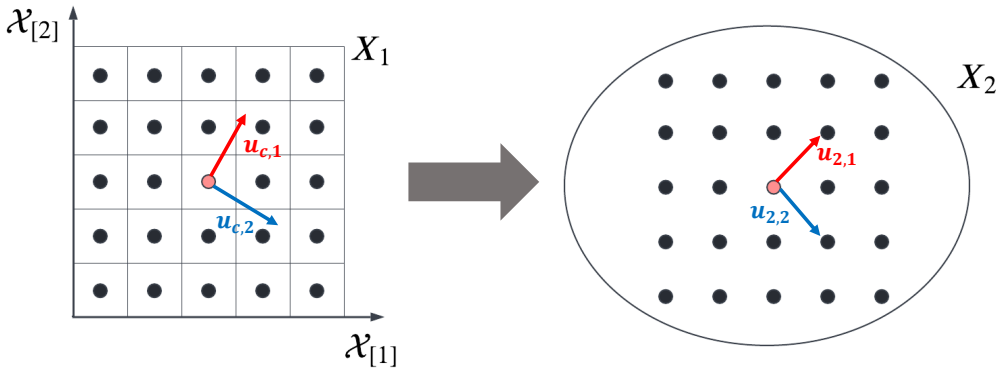

Figure 3.2: Construction of the abstract transitions. For each discretized input ($u_{c,1}$ and $u_{c,2}$ in the figure) the red state $x_{c,i}$ transition to two states in different cells. Those transitions $x_{c,i} \xrightarrow{u_{c,1}} x'$ and $x_{c,i} \xrightarrow{u_{c,2}} x''$ are used to construct two analog ones of the symbolic system.

Roughly speaking, in this four steps we have discretized the state and input spaces. We have associated a symbolic state to the center of each cell, which can now be seen as discrete states and inputs. Moreover, we can notice that the behavior of each abstract state corresponds to the one of the center of the corresponding cell.

At this point, since the inputs $u \in U$ are defined by the controller, we can restrict the behavior of the original system to the one that accepts only the inputs corresponding to the ones of the symbolic model. We define the discrete input set as

$$U_1' = \{u_1 \mid u_1 = \mathcal{R}_U(u_2)\, \forall u_2 \in U_2\} \tag{3.8}$$

Therefore, it is straightforward to notice that the system

$$\mathcal{S}_1' = (X_1, X_{1,0}, U_1', \rightarrow_1, Y_1, H_1) \tag{3.9}$$

is alternatingly simulated (see Def. 4.1.1) by $\mathcal{S}_1$.

Furthermore, since all transitions of $\mathcal{S}_2$ have a correspondent one in $\mathcal{S}_1'$, it holds that $\mathcal{S}_2 \preceq_{AS} \mathcal{S}_1'$ and therefore $\mathcal{S}_2 \preceq_{AS} \mathcal{S}_1$. However, the converse does not necessarily hold.

As mentioned in Section 2, a mathematical description of the system is required to derive formal guarantees on the behavior of the closed-loop symbolic controller. For this reason, few works attempted to implement data-driven abstraction-based controllers. Nevertheless, some of them succeeded to derive some probabilistic bounds. In [9], the authors proposed a data-driven approach based on a black-box model to

construct an abstraction relying on the notion of *PAC approximate alternating simulation relation*. The state space is uniformly discretized into cells and each of them is mapped into a state of the symbolic model. Transitions in the abstraction are computed by simulating trajectories starting in a cell. In contrast to the classical approach, this technique does not suffer from the conservatism resulting from the over-approximation of the reachable set (see Fig. 3.3). However, this method does not provide strong guarantees of correctness for the original problem but PAC bounds since some behaviors of the original system may not be simulated by the probabilistic abstraction.

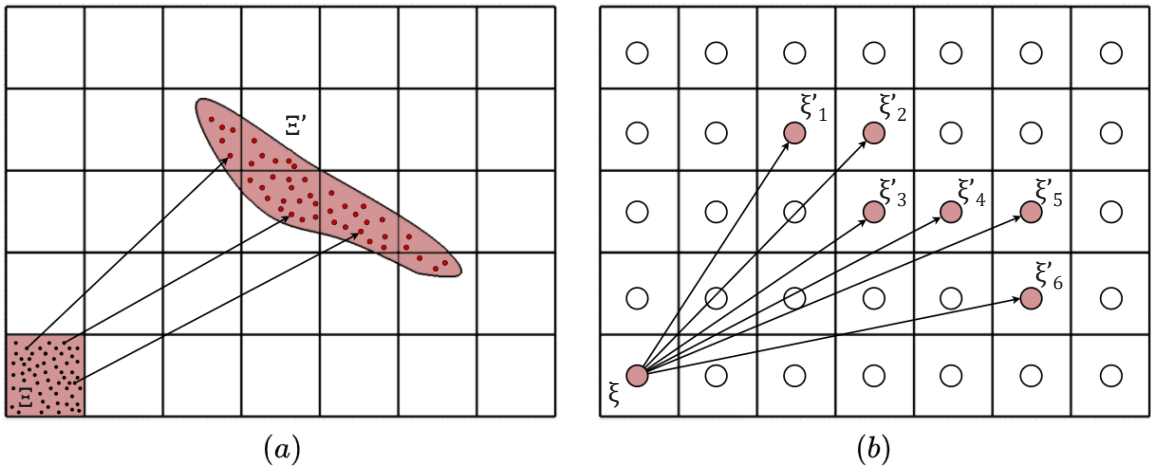

Figure 3.3: Data-driven abstraction approach based on multiple samplings per cell. In this example an autonomous system with a 2-dimensional state-space is partitioned into cells (Fig. a). Multiple trajectories are simulated for each cell in order to estimate the reachable set. The states represented by black points in Ξ transition to the red ones contained in Ξ' (i.e., the actual image of Ξ). As can be observed by comparing this figure with Fig. 2.2, the successors of xi are only 6, compared to the 12 of Fig. 2.2 (where the abstraction is computed with a model-based approach).

In the same vein, the authors of [10] also proposed a method to provide a PAC bound on a reachability control problem by computing bounds on transitions with a desired confidence level based on scenario-based approach methods. Work [11] proposed to use Gaussian process (GP) regression [12] and abstraction to compute a strategy that maximizes the probability of satisfying an LTL formula [13] of a partially-known stochastic system from a given data set. In [14], the authors have proposed a procedure to compute bounds on the probability of satisfying a specification for the particular class of mixed monotone stochastic systems. In both cases,

the resulting abstract system is nondeterministic, even though the original system is deterministic, due to the uncertain location of the continuous state in a cell.

Moreover, without assumptions on a system behavior that are usually derived with the knowledge of a model, it is not possible to derive tight deterministic bounds on the system behavior. Indeed even if a huge amount of trajectories are simulated for each cell, some of them leading to different regions of the state space might not be simulated. A graphical demonstration of this phenomenon is shown in Fig. 3.4. If none of the trajectories starting from the small blue region are simulated, the abstraction (Fig. 3.4 (a)) does not alternatingly simulate the original system (Fig. 3.4 (b)).

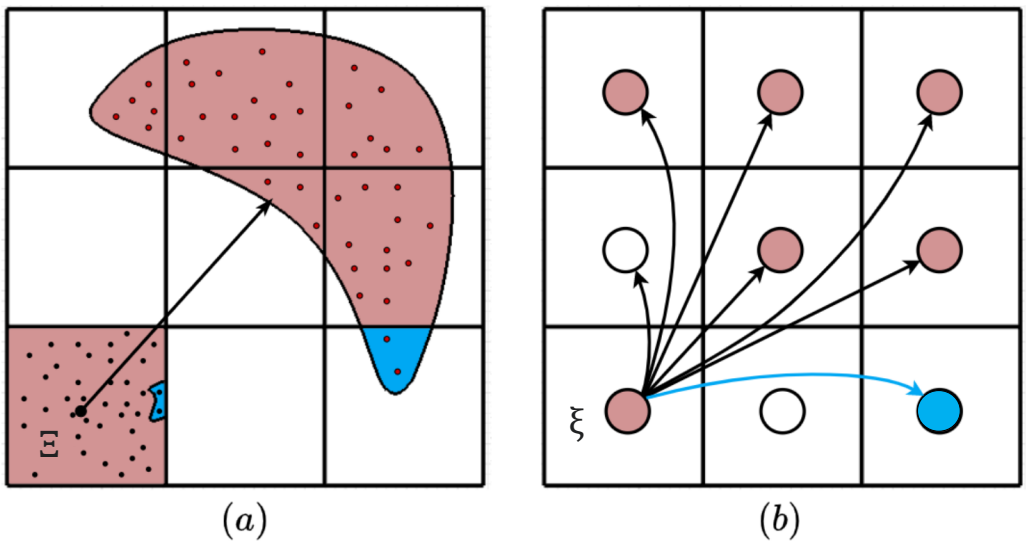

Figure 3.4: In this example a symbolic abstraction is constructed as in Fig. 3.3. From this example is possible to see how only a small part of the cell (in blue) transitions to a cell different from the others. This can lead to non-determinism even if the cell behavior is quasi-deterministic. Moreover, if none of the points in the blue region is simulated the abstraction (right) is not an alternating simulation of the concrete system (left).

3.3 A Data-Driven Probabilistic Abstraction

As previously mentioned, abstractions allow us to synthesize controllers for general nonlinear dynamical systems taking into account different constraints of cyber-physical systems. However, in a growing number of situations either the model of the system is unavailable, or it is too difficult to describe accurately, therefore, a data-driven approach is the only sensible solution to effectively control these systems. In this subsection I propose an approach to leverage the information coming from heuristics in order to decrease the conservatism inherent with discretization. To be more specific, I will present a Monte-Carlo approach based on a probabilistic interpretation of the abstraction error leading to the construction of a probabilistic Markov Decision Process [15].

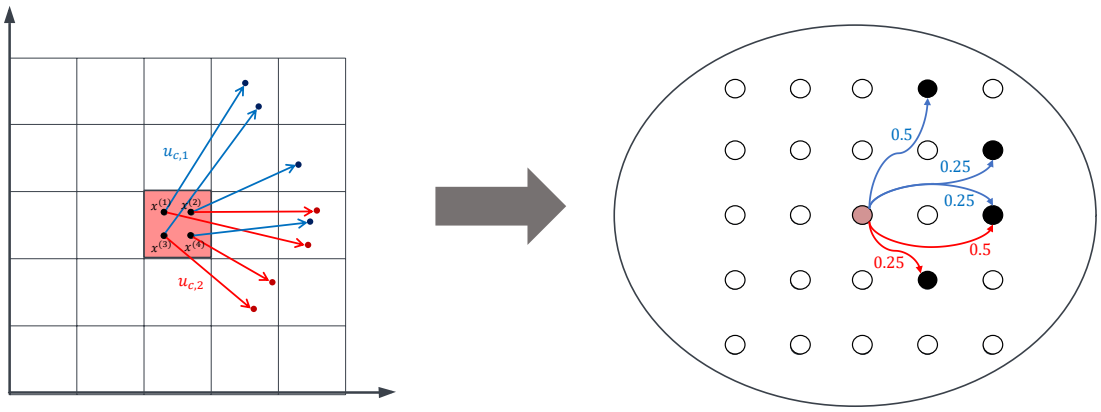

Figure 3.5: Construction of a probabilistic abstraction via Monte-Carlo methods. In the concrete system (left) $M = 4$ points $x^{(i)}$ are sampled from the pink cell and their trajectories simulated for each discrete input $(u_{c,1}, u_{c,2})$. The corresponding transitions are mapped in the abstraction (right). To each transition is assigned a probability proportional to the number of points that transition to the corresponding cell.

Suppose to be working in the same framework described in Sec. 3.2, i.e., wanting to control a simple transition system S_1 defined as in (3.1). Similarly of the procedure presented in Sec. 3.2, the state and space is partitioned into cells and each of them is mapped to a symbolic state. Then, we sample k samples per cell and simulate their trajectories for each discrete input. Finally we construct the abstraction so that each

state-input pair (x, u) transitions to a state x' with a probability proportional to the number of points that map to the corresponding cell. These probabilities have no real meaning with respect to the concrete dynamics but can be used as a proxy for the non-determinism caused by the discretization. This procedure is presented in detail in the Algorithm 1 and a graphical intuition is shown in Fig. 3.5.

Algorithm 1 Construction of a Monte-Carlo Probabilistic Abstraction

Input:

Concrete system $\mathcal{S}_1 = (X_1, X_{1,0}, U_1, \rightarrow_1, Y_1, H_1)$ as in (3.1)
(with X_1, $X_{1,0}$, and U_1 hyper-intervals)

Outputs:

Abstract system $\mathcal{S}_2 = (X_2, X_{2,0}, U_2, \rightarrow_2, Y_2, H_2)$

Domain relations $\mathcal{R}_X : X_1 \rightarrow X_2$ and $\mathcal{R}_U : U_1 \rightarrow U_2$

Inverse domain relations $\mathcal{R}_X^\dagger : X_2 \rightarrow X_1$ and $\mathcal{R}_U^\dagger : U_2 \rightarrow U_1$

Probability map $P : X_2 \times U_2 \times X_2 \rightarrow [0, 1]$

Partition X_1 and U_1 into hyper-rectangles of the same length ($\{X_{1,i}\}_{i=1\ldots m}$ and $\{U_{1,j}\}_{j=i\ldots n}$ respectively, with m and n finite)[a]

$X_2 \leftarrow \{x_{2,i}, \text{ for } i = 1, 2, \ldots, m\}$

for all $x \in X_1$ **do**

  if $\exists\, i$ s.t. $x \in X_{1,i}$ **then**

    $\mathcal{R}_X(x) \leftarrow x_{2,i}$

  end if

end for

$X_{2,0} \leftarrow \{x_{2,i} \mid X_{1,i} \subseteq X_{1,0}\}$

$U_2 \leftarrow \{u_{2,j}, \text{ for } j = 1, 2, \ldots, n\}$

for all $u \in U_1$ **do**

  if $\exists\, j$ s.t. $u \in U_{1,j}$ **then**

    $\mathcal{R}_U(u) \leftarrow u_{2,i}$

  end if

end for

---

[a] Assume also that $X_{1,0}$ corresponds exactly to the union of some cells $X_{1,i}$

Algorithm 1 (cont'd)

for $i \in \{1, 2, \ldots, n\}$ **do**

$x_{c,i} \leftarrow$ center of the cell $X_{1,i}$

$\tilde{x} = \mathcal{R}_X(x_{c,i})$

$\mathcal{R}_X^\dagger(\tilde{x}) \leftarrow x_{c,i}$

end for

for $j \in \{1, 2, \ldots, m\}$ **do**

$u_{c,i} \leftarrow$ center of the cell $U_{1,i}$

$\tilde{u} = \mathcal{R}_U(u_{c,i})$

$\mathcal{R}_U^\dagger(\tilde{u}) \leftarrow u_{c,i}$

end for

for $i \in \{1, 2, \ldots, n\}$ **do**

Select the number of samples K

Select a distribution p_i such that $\forall\, x \in X$, $p_i(x) > 0 \iff x \in X_{1,i}$

for $j \in \{1, 2 \ldots, m\}$ **do**

Draw K samples $x^{(1)}, \ldots, x^{(K)}$ from $X_{1,i}$ according to the distribution p_i, and compute the $u_{c,j}$ -successors $x^{(1)\prime}, \ldots, x^{(K)\prime}$

for $k \in \{1, 2 \ldots, K\}$ **do**

Define $\rightarrow_2 (x_{2,i}, u_{2,j})$ as

$\left\{x_{2,i} \in X_2 \mid \exists k \in \{1, \ldots, K\} \text{ s.t } x^{(k)\prime} \in X_{1,i}\right\}$

end for

for all $(x_{2,i}, u_{2,j}, x') \in \rightarrow_2$ **do**

Define $P_{\rightarrow_2} (x_{2,i}, u_{2,j}, _2\, x')$ equal to the amount of times x' occurs

in $\left\{x^{(1)\prime}, \ldots, x^{(K)\prime}\right\}$ divided by K

end for

end for

end for

$Y_2 \leftarrow Y_1$

$H_2 \leftarrow \mathcal{R}_X^\dagger$

$\mathcal{S}_2 \leftarrow (X_2, X_{2,0}, U_2, \rightarrow_2, Y_2, H_2)$

return $\mathcal{S}_2, \mathcal{R}_X, \mathcal{R}_U, \mathcal{R}_X^\dagger, \mathcal{R}_U^\dagger$

3.4 Controller Refinement

Once the abstraction has been constructed and we have defined the maps R the symbolic model, a control policy 3.1.5 must to be designed in order to control the symbolic model according to the given specifications.

Therefore, first the specifications for the concrete system have to be translated as abstract ones. In order to do so, we must rely on the relations $\mathcal{R}_X : X_1 \to X_2$, $\mathcal{R}_X^\dagger : X_2 \to X_1$, $\mathcal{R}_U : U_1 \to U_2$, $\mathcal{R}_U^\dagger : U_2 \to U_1$ previously defined.

A typical specification is the reachability of a target set $X_{1,f} \subseteq X_1$ by performing the minimum number of steps. The analog abstract problem can be defined simply by mapping the elements of $X_{1,f}$ to symbolic states $\in X_2$ according to the map $\mathcal{R}_X$, thus obtaining an abstract target set $X_{2,f}$.

Once defined the abstract specifications, a policy can be designed by employing suitable state-of-the-art algorithms to control the specific transition system. The algorithms employed in this project are presented in the next section (Sec. 4).

Algorithm 2 Control Policy Refinement

Inputs:

Concrete system $\mathcal{S}_1 = (X_1, X_{1,0}, U_1, \to_1, Y_1, H_1)$

Abstract system $\mathcal{S}_2 = (X_2, X_{2,0}, U_2, \to_2, Y_2, H_2)$

Abstract static control law $\mathcal{C}_2 : X_2 \to U_2$

State relation $\mathcal{R}_X : X_1 \to X_2$

Input relation $\mathcal{R}_U^\dagger : U_2 \to X_1$

Outputs:

Concrete static control law $\mathcal{C}_1 : X_1 \to U_1$

for all $x_1 \in X_1$ **do**

$x_2 \leftarrow \mathcal{R}_X(x)$

$u_2 \leftarrow \mathcal{C}_2(x_2)$

$u_1 \leftarrow \mathcal{R}_U^\dagger(u_2)$

$\mathcal{C}_1(x_1) \leftarrow u_1$

end for

return $\mathcal{C}_1$

4

Control Policy Design

4.1 Abstractions as Markov Decision Processes

As previously mentioned, the abstract transition systems introduced in Sections 3.2 and 5.2.2 can be actually be translated to Markov Decision Processes [16] in order to implement state-of-the-art dynamic programming algorithms[17, 18].

Definition 4.1.1 (Markov Decision Process)

A Finite-Horizon Markov Decision Process (MDP) is a 4-tuple (S, A, P, r, R)*, where:*

- S *is a finite set of states*
- A *is a finite set of actions*
- $P : S \times A \times S \to [0, 1]$ *is the probability that action* $a \in A$ *in state* $s \in S$ *at time* t *will lead to state* s' *at time* $t{+}1$*, namely,* $P(s, a, s') = \mathbb{P}\{s_{t+1} = s' \mid s_t = s, a_t = a\}$
- $r : S \times A \times S \to \mathbb{R}$ *is the reward function*
- $R : S \to \mathbb{R}$ *is the final reward function*

With a slight abuse of notation we define

$$r(s, a) \doteq \mathbb{E}_{s'}\{r(s, a, s') \mid (s, a)\} = \sum_{s'} [r(s, a, s')\ P(s, a, s')] \tag{4.1}$$

Given this definition of Markov Decision Process we can observe that by defining the reward functions r and R we can construct one MDP starting from the symbolic

model (S_2 as obtained in Alg. 1).

To be specific:

- $S \leftarrow X_2$
- $A \leftarrow U_2$
- $P \leftarrow P$

The same holds for the basic data-driven abstraction S_2 obtained in Sec. 3.2. Since the transition system is deterministic order to obtain a probability map we simply compute

$$P(s, a, s') = \mathbb{P}\{s_{t+1} = s' \mid s_t = s, a_t = a\} = 1 \tag{4.2}$$

4.2 Reachability problem

Defining the reward functions allows us to solve different kind of problems such as safety and reachability.

This project will be focused mainly on solving reachability problems, namely, reaching a state $s \in T \subseteq S$ in the minimum amount of time.

In order to solve this problem we set:

- $r(s, a, s') = -\varepsilon \quad \forall (s, a, s') \in S \times A \times S \quad \text{with } \varepsilon > 0$
- $R(s) = L \quad \forall s \in T \quad \text{with } L >> \varepsilon$
- $R(s) = 0 \quad \forall s \in S \setminus T$

4.3 MDP control algorithms

In order to design a symbolic controller for the abstraction we have translated the problem to the control of a Markov Decision Process.

With respect to the two abstraction proposed, we have come up with two distinct Markov Decision Processes. The one derived with the basic approach is indeed deterministic (see Eq. 4.2) while the other present stochastic behavior.

For this reason many algorithms suitable for a deterministic system are not applicable to a probabilistic control process.

Controlling a deterministic MDP with the rewards presented in Sec. 4.2 is equivalent to solve a shortest path problem where all the edges have the same weights,

therefore we can employ a Dijkstra's-like algorithm [18, 19]. The dynamic programming algorithm performs a backward propagation starting from the target sets until it visits all the reachable sets and select the action leading to the shortest path for each visited state. This algorithm – that we will call *Backward Induction*[1] – is presented below 3.

Algorithm 3 Backward Induction

Inputs:

S set of states

A set of actions

T set of target states

Outputs

$W : S \to \mathbb{R} \cup \{-\infty, +\infty\}$ ▷ steps to reach T

Control policy $\mathcal{C} : S \to A$

---

for all $s \in S \setminus T$ **do**

  $W(s) \leftarrow \infty$

end for

for all $s \in T$ **do**

  $W(s) \leftarrow 0$

end for

$Q \leftarrow T$ ▷ set of visited states

for all $s' \in Q$ **do**

  for all $s \in \text{Prev}(s') \setminus Q$ **do** [a]

    $W(s) \in \min_{a \in A} \{W(\text{Post}_a(s) + 1)\}$

    $\mathcal{C}(s) \in \arg\min_{a \in A} \{W(\text{Post}_a(s))\}$

  end for

end for

return $W, \mathcal{C}$

---

[a] With $Prev(s')$ we indicate all the predecessors of s', namely, all $s \in S$ s.t. $\exists a \in A$ s.t. $P(s, a, s') > 0$

[1] In the Reinforcement Learning community this name is often used as a synonym of *Value Iteration*, nevertheless in this project I will make a distinction between this two.

A backpropagation algorithm as 3 cannot be applied to a probabilistic MDP since multiple edges exits from each node and optimal trajectories may loop. However, in this scenario we can apply state-of-the-art algorithms (usually coming from Model-Based Reinforcement Learning [20]) such as *Value Iteration* [16–18], *Policy Iteration* [17, 18], and variants thereof.

In this project, we will rely solely on value-based algorithms since the value function can provide important information as it will be explained later on.

Algorithm 4 Value Iteration

Inputs:

S set of states

A set of actions

T terminal states

$P : S \times A \times S \to [0, 1]$ probability map

$r : S \times A \times S \to \mathbb{R}$ transition rewards

$R : S \to \mathbb{R}$ terminal rewards

$\theta > 0$ threshold

Outputs

$V : S \to \mathbb{R} \cup \{-\infty, +\infty\}$ ▷ steps to reach T

Control policy $\mathcal{C} : S \to A$

for all $s \in T$ **do**

$V^+(s) \leftarrow V(s) \leftarrow 0$

end for

while $\Delta < \theta$ **do**

for all $s \in S \setminus T$ **do**

$V^+(s) \leftarrow \max_{a \in A} \sum_{s'} \{P(s, a, s') \ [r(s, a, s') + V(s') + R(s')]\}$

$\mathcal{C}(s) \leftarrow \arg\max_{a \in A} \sum_{s'} \{P(s, a, s') \ [r(s, a, s') + V(s') + R(s')]\}$

$\Delta \leftarrow \max(\Delta, |V^+(s) - V(s)|)$

end for

$V \leftarrow V^+$

end while

return $V^+, \mathcal{C}$

5

Julia Implementation

5.1 Overview and objectives

This research project was conducted while collaborating to the development of the *Dionysos* software [1]. The software has the final aim of developing a toolbox able to control non-linear systems via abstraction-based techniques. More specifically, Dionysos's objective is to learn the optimal control CPSs from data, whether harvested from the physical system or generated synthetically.

The software has been developed using the Julia Programming Language [21]. Julia has been chosen mainly because of its high performances and its reproducibility. Indeed, Julia programs compile to efficient native code for multiple platforms via LLVM. Moreover, reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Until now, data-driven abstraction-based controllers have not been proved useful to control high-dimensional non-linear systems except small theoretic examples.

As a matter of fact, the classical abstraction method [22] suffers from having to compute the abstraction on the complete state space. In [23] the authors propose to adapt the size of the abstraction gradually but uniformly over the whole state space. While [24] propose to co-design the abstraction and the controller guided by the optimal control problem in order to reduce the computed part of the abstraction.

This project aims to show the potential of this approach deriving a symbolic model based on a probabilistic interpretation of the abstraction error.

Constructing an abstraction requires simulating the system a significant number of times, which usually consists of a small number of simple operations. If so, parallelizing these computations could dramatically improve the effectiveness of this approach. Indeed, parallel computing has become one of the most effective tools to

speed up many algorithms, and the evolution of computer architectures towards a higher number of cores can only confirm the effectiveness of this method.

Therefore, the Julia toolbox developed for this thesis [25], I also aim to build the foundation for a final software implementing a new agile approach to build the abstraction, while parallelizing most operations and complying with temporal logic constraints.

A smart dynamic way to build the abstraction can be combined with the approach described described in Sec. 5.2.2 in order to cope with the curse of dimensionality. An initial abstraction could be built offline, and then modified online while computing the control policy. Possible methods include refining certain regions of the state space, building the abstraction "lazily", sampling a variable number of points for each cell, dynamically considering more or fewer possible inputs, etc. In particular my software has been developed with the future intent to use an adaptive discretization of the state-space. The abstraction could be refined both based on local measures, such as the entropy of each node or the value function, and non-local factors, such as the variance of the system behavior among close cells, the likelihood for the feedback system to end up in each state (especially when the initial or target sets are known), or trying to understand if changing the abstraction in a specific region can influence the value function also on other areas of the state-space.

5.2 Structure

In this subsection I will discuss the main features of my Julia toolbox [25] while giving an overview on its structure and implementation.

The whole software has been implemented from scratch, namely only relying on packages of the `Base` Julia library plus some to handle vectors efficiently and plot figures (`StaticArrays.jl` [26] and `Plots.jl` [27]).

The main module (named *Dionysos*) has been divided into 6 sub-modules, each of them dealing with an aspect of abstraction based-control (plus one with *utils*). Those are:

- `Domain`
- `Map`
- `System`
- `Relation`

- `Control`
- `Simulation`
- `Utils`

5.2.1 The Domain Implementation

This module contains the function to define domains, namely, both contiuous and symbolic domains X_1, X_2, U_1, U_2 (as in Sec.).

In particular the file `nested_domain.jl` contains the methods to discretize a continuous bounded space into cells.

Roughly speaking, these nested domains consist of a variable-step grid implemented in a "nested" way (see Fig. 5.1). For example if we decide to use a uniform discretization of the space, each cell of the grid is marked as 1 if it's "inside" the domain (green-colored), 0 if it's not (e.g. there is an obstacle, red-colored) or -1 if it's "refined" (yellow-colored). By refined we mean that inside the domain it is stored another domain (e.g., see orange arrow). This domain itself presents the same properties of its "parent" and each of its cells can be additionally refined.

This implementation is very versatile. For example, it allows to remove an obstacle that has arbitrary bounds (e.g., see the black arrow in Fig. 5.1). Moreover it also takes into account that some dimensions may be periodic.

It is also very efficient since it is implemented with Julia Dictionaries. For example, finding in which cell a certain coordinate x lies has linear complexity with respect to the depth of the nested domain, $O(\text{depth})$.

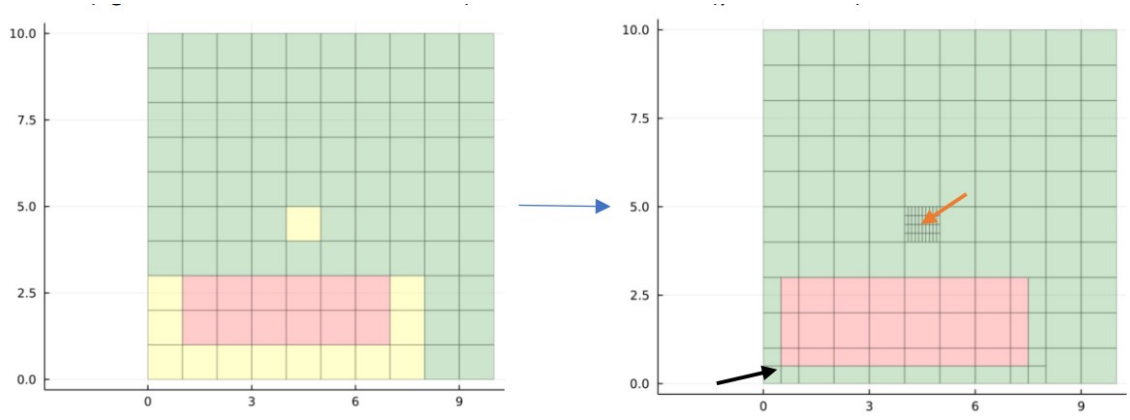

Figure 5.1: Nested domain

5.2.2 The Map Module

The Map module is used to relate one domain to another, in particular it maps a continuous domain to a symbolic one as the maps $\mathcal{R}_X, \mathcal{R}_U, \mathcal{R}_X^\dagger, \mathcal{R}_U\dagger$ in Sec. . Also this module mainly rely on the use of dictionaries.

5.2.3 The System Module

The System module contains the methods to implement both continuous-state dynamic systems both symbolic ones by relying on the Domain module.

In particular, probabilistic transition systems are implemented as automata where each node store information about its predecessors and successors and the transition probabilities.

5.2.4 The Relation Module

This module contains the methods responsible of creating a symbolic model starting from a continuous-state system following a similar procedure to the one described in Algorithm 1. For this reason this module relies on the previous ones and relates a concrete to an abstract system and vice-versa.

Constructing an abstraction requires simulating the system a significant number of times, thus these operations are parallelized on different threads which improves the effectiveness of the algorithm.

5.2.5 The Control Module

This module contains the methods to design control policies for the systems (thus relies on the System module). In particular it implements the algorithms 3 and 4 presented in Sec. 4.

The execution of the Value Iteration algorithm is parallelized on multiple threads, while it is not possible to do the same with the Backward Induction.

5.2.6 The Simulation Module

This module finally combines everything together and simulates a system given a controller. In particular if only an abstract control policy is given it also refines the controller in order to suite the concrete system as presented in Sec. 3.4 (Alg. 2).

6

Examples

In this section, I present some examples that I used to demonstrate the practicality of my approach on control problems for non-linear systems.

The simulations are runned on a CPU AMD Ryzen 7 4700U (8 cores running at 2GHz) with a RAM of 16 GB.

Both examples are expressed as continuous-time models with a measurement noise, namely with a function is given. More specifically

$$\dot{x} \in f(x, u) + W \tag{6.1}$$

where

$$f : X \times U \to \mathbb{R}^n, X \subseteq \mathbb{R}^n, U \subseteq \mathbb{R}^m \text{ and } W \subseteq \mathbb{R}^n$$

In order to obtain its discretized version F we employ a 4^{th}-order Runge-Kutta method (RK4)[28].

6.1 Path planning for an autonomous vehicle

We consider an autonomous vehicle whose dynamics are given by the bicycle model in [29]. More concretely, $f : \mathbb{R}^3 \times U \to \mathbb{R}^3$ is given by

$$f\left(x, (u_1, u_2)\right) = \begin{pmatrix} u_1 \cos\left(\alpha + x_3\right) \cos(\alpha)^{-1} \\ u_1 \sin\left(\alpha + x_3\right) \cos(\alpha)^{-1} \\ u_1 \tan\left(u_2\right) \end{pmatrix} \tag{6.2}$$

with $U = [-1, 1] \times [-1, 1]$ and $\alpha = \arctan\left(\tan\left(u_2\right)/2\right)$. Here, (x_1, x_2) is the position and x_3 is the orientation of the vehicle in the 2-dimensional plane. The control inputs u_1 and u_2 are the rear wheel velocity and the steering angle.

The concrete control problem is formulated with respect to the concrete system of Eq. 6.2. The control objective is to enforce the reaching of the rectangle $[9, 9.6] \times [0.2, 0.8]$ on the vehicle which is situated in a maze of dimension $[0, 10] \times [0, 10]$ (see Fig. 6.1) starting from any point of the domain.

The system is discretized with a time-step $\tau = 0.3$. The input space is partitioned with a step of $[0.4, 0.5]$. The state space is partitioned with a step of $[0.4, 0.4, \pi/5]$. However, it can be noticed that the discretization of the state domain into cells is not necessary uniform. Indeed the obstacles and the target set are considered with their exact dimensions. This discretization leads to 6680 symbolic states.

We construct the abstraction using K samples per cell uniformly distributed and the simulations are started on the opposite site of the maze with respect to the target set, namely in $x_0 = [0.2, 0.2, 0.0]$.

During the computation of the policy it has been considered that if a state $\tilde{x}$ of the abstract system has a non-zero probability of hitting an obstacle or going out of bounds that state is considered as "prohibited". Combining this aspect with the definition of *PAC approximate alternating simulation relation* [9] previously mentioned, a PAC-bound can be derived in order to obtain a probabilistic guarantee that the system will not end up in a prohibited state.

First, the system has been simulated considering a single sample per cell (i.e., $K = 1$) and therefore relying on Backward Induction. The resulting controller is not able to reach the target set since the reachable set of each cell are too small with respect to the actual ones. The controller almost always leads the system to a prohibited state or to end up in an infinite loop. Indeed, as previously mentioned, this data-driven approach does not provide formal guarantees on the correctness of the resulting controller since the abstraction does not alternatingly simulate the original system. The computation of this deterministic abstraction takes $6\,s$ and the controller design $0.1\ s$. The result is shown in Fig. 6.1.

The second experiment has been conducted by considering a probabilistic automaton with $K = 8$. However, even this approach does not provide formal guarantees and since the simulated trajectories are probably not enough the resulting controller is still not sufficient to bring the system from x_0 to $x_f \in T = [9, 9.6] \times [0.2, 0.8] \times \mathbb{R}$.

Nevertheless $K = 8$ can be a sufficient number of samples for this problem if we try also to simulate some trajectories starting from points on the border of each cell. In fact, if we consider as prohibited the cells which borders may lead the state into an obstacle we are able to solve the task as shown in Fig. 6.2. With this procedure the allowed states are now 6572 (compared to the previous 6680). The computation of the abstraction takes $6\,s$ and the control policy design $15\,s$. The result is shown in Fig. 6.1, where the target is reached in $66.3\ s$.

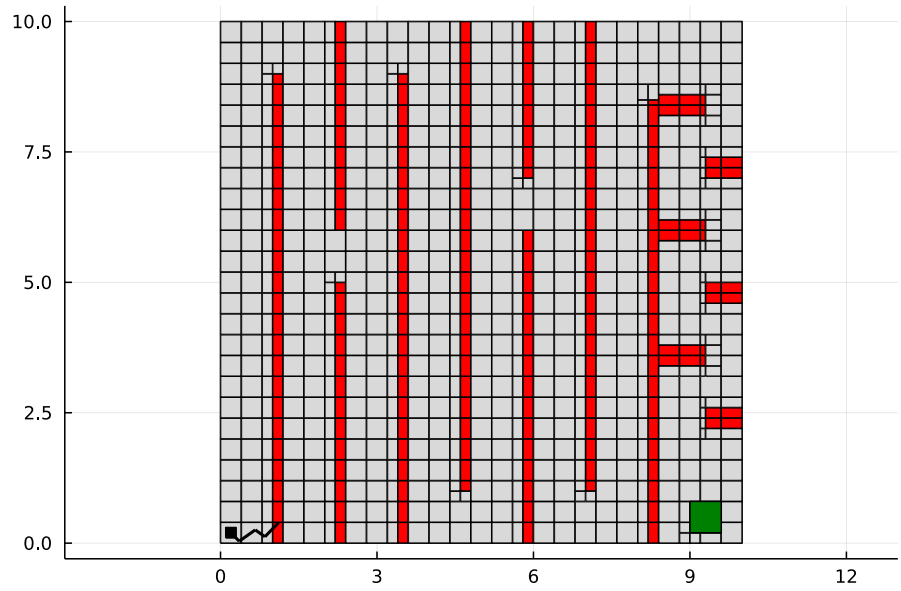

Figure 6.1: Path planning for an autonomous vehicle: basic data-driven abstraction ($K = 1$). This is a 2D projection of the 3D domain. The black square is the starting point, the red rectangles are the obstacles of the maze, while the green square is the target set.

Fig. 6.3 shows the result of the experiment when sampling 3 points per dimension (per cell), i.e., $K = 27$. The computation of the abstraction takes $14\,s$ and the control policy design $22\,s$. The target is reached in $57.6\,s$.

The resulting controller is also very robust to noise. In fact, if we consider a white Gaussian noise on the measurements with standard deviation equal to $\sigma = (0.1, 0.1, \pi/20)$ (namely σ is equal to the 25% of an average cell) the controller reaches the target set in 58.2 seconds.

The same system with a similar problem has been used as an example also in [30]. There the authors derive formal guarantees on the system behavior by using a model-based approach. Even though their approach guarantees the correctness of the refined controller by computing a Lyapunov function and relying on the definition of *approximate alternating simulation relation* [5], their abstraction takes 13509 seconds to compute (compared with the 14 seconds of Fig. 6.3) and the controller synthesis 535. They ran their simulation on a CPU Intel Core Duo @ 2.4GHz.

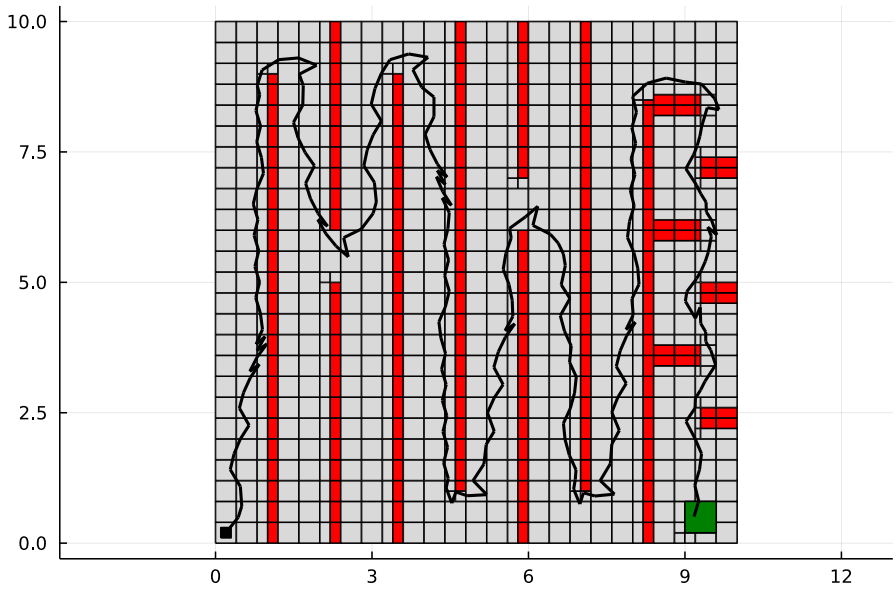

Figure 6.2: Path planning for an autonomous vehicle: basic data-driven abstraction with $K = 8$. The cells which borders may lead the state into an obstacle have been avoided.The target is reached in 66.3 s.

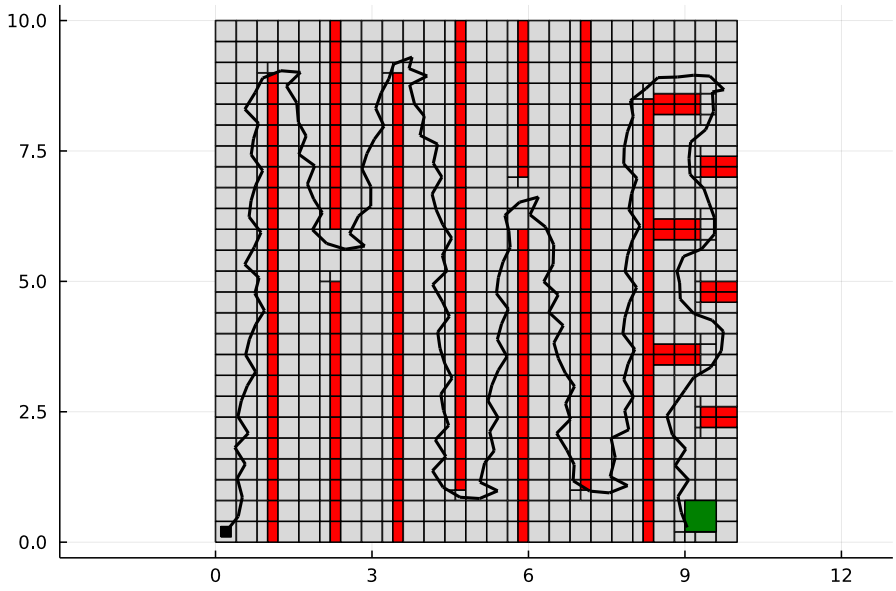

Figure 6.3: Path planning for an autonomous vehicle: basic data-driven abstraction with $K = 27$. The target is reached in 57.6 s.

6.2 Cart-pole balancing problem

The second system considered comes from a widely common example among the Reinforcement Learning community, namely, the cart-pole (a.k.a. inverted pendulum on a cart). Its model is represented in Fig. 6.4 with its linearized equations in Eq. 6.3.

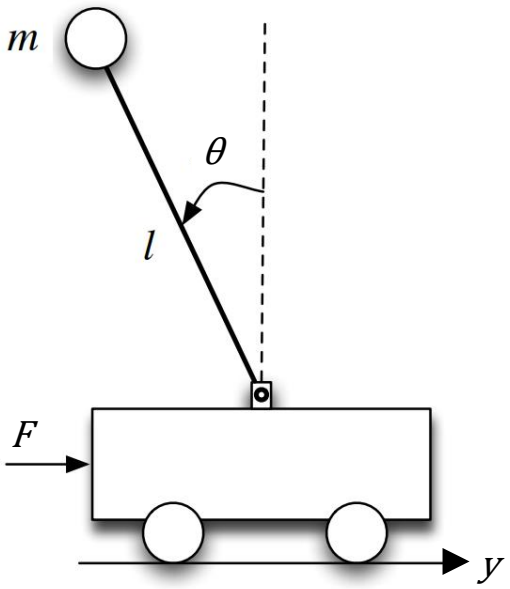

Figure 6.4: Cart-pole model

$$\begin{aligned} \ddot{\theta} &= \frac{g\sin\theta + \cos\theta\left[\frac{-F-ml\dot{\theta}^2\sin\theta+}{m_c+m}\right]}{l\left[\frac{4}{3}-\frac{m\cos^2\theta}{m_c+m}\right]} \\ \ddot{y} &= \frac{F+ml\left[\dot{\theta}^2\sin\theta-\ddot{\theta}\cos\theta\right]}{m_c+m} \end{aligned} \tag{6.3}$$

where

- $g = 9.81$ is the gravity acceleration
- $m = 0.1$ is the mass at the end of the pole
- $m_c = 1.0$ is the cart mass

- $l = 0.5$ is the pole length
- F is the input force
- θ is the pole angular position
- y is the cart position

Therefore this problem will be represented with the 4D state

$$x = \begin{pmatrix} y \\ \theta \cdot 180/\pi \\ \dot{y} \\ \dot{\theta} \cdot 180/\pi \end{pmatrix} \tag{6.4}$$

and

$$u = \frac{F}{10} \tag{6.5}$$

The control problem consists in bringing the cart-pole in a vertical position in the center y domain (i.e., $\theta = 0, y = 0$) by applying a control input $u = \pm 1$.

The domain has been considered as $[-4, 4] \times [-30, 30] \times [-6, 6] \times [-4 \cdot 180/\pi, 4 \cdot 180/\pi]$.

The target set is $y \in [-0.2, 0.2]$, $\theta \in [-1.5 \cdot 180/\pi, 1.5 \cdot 180/\pi]$.

The system has been controlled using the probabilistic Monte-Carlo abstraction with $K = 16$. The system has been discretized with a time-step $\tau = 0.04$, while the state space has been partitioned into cells of dimension $[0.4, 3 \cdot 180/\pi, 0.6, 0.4 \cdot 180/pi]$. This leads on defining a total of 144806 symbolic states.

The abstraction takes 24 seconds to construct, while the controller synthesis is performed in 50. The refined controller is always able to reach the target when starting from state not too close to the border of the domain. An example of a trajectory starting from $y = -2.5$, $\theta = -10 \cdot 180/pi$ is shown in Fig. 6.5

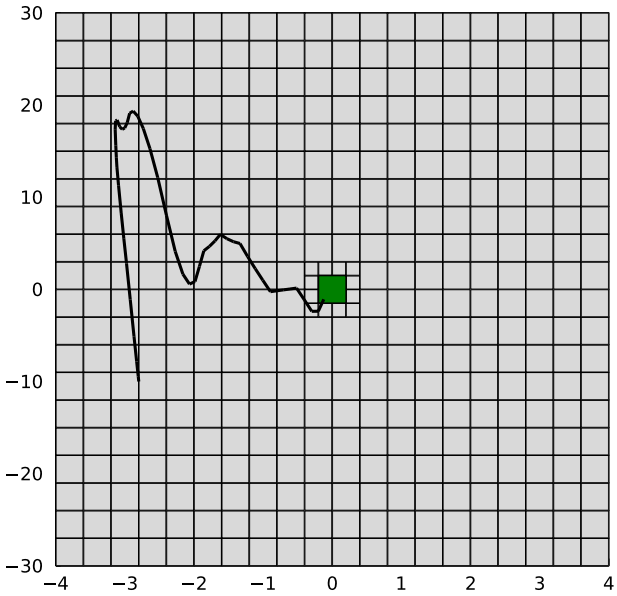

Figure 6.5: Cart-pole balancing problem: Monte-Carlo probabilistic abstraction with $K = 16$. The angular position θ is shown in deg on the vertical dimension, while the horizontal dimension represents the position y. A trajectory starts from $y = -2.5$, $\theta = -10 \cdot 180/pi$ and reaches the target is in 1.96 s.

Also the basic data-driven approach has been tried in order to solve this reachability problem. Nevertheless it achieves *almost* satisfactory results only with a very dense grid. A simpler problem has been defined in order to derive a controller in a relatively small amount of time. Namely the dimension of the y domain has been reduced and since state space has been partitioned in smaller cells also the simulation time-step has been reduced. More specifically, the following changes have been applied

- $y \in [-1.2, 1.2]$
- each cell dimension is half of the previous one, namely, $[0.2, 1.5 \cdot 180/\pi, 0.4, 0.2 \cdot 180/pi]$
- $\tau = 0.02$

The abstraction takes 303 seconds to construct, while the controller synthesis is performed in 38. However, as shown in Fig. 6.6, even with this denser grid the refined controller is not able to successfully reach the target.

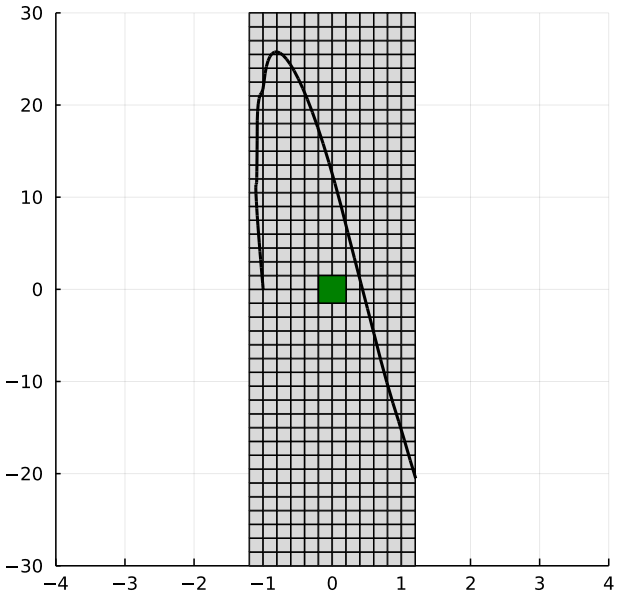

Figure 6.6: Cart-pole balancing problem: basic data-driven abstraction abstraction ($K = 1$). The angular position θ is shown in deg on the vertical dimension, while the horizontal dimension represents the position y. A trajectory starts from $y = -1.0$, $\theta = 0$ but is not able to reach the target.

7

Conclusions

The main objective of this project was to show the potential of abstraction-base controllers approach by implementing a novel data-driven approach based on a probabilistic interpretation of the discretization error.

Moreover this research project was conducted while collaborating to the development of the *Dionysos* software [1] at UCLouvain. Therefore, my research project was carried on by working in close contact with Prof. Jungers's team (and in particular with the Dionysos dev team) with the aim of developing a toolbox able to control non-linear systems via abstraction-based techniques.

In this paper I have presented the main advantages and limits of abstraction-based controllers. Firstly, I introduced the most common abstraction-based strategies relying on a mathematical description of the system. Secondly, I presented the most significant data-driven attempts to control non-linear systems. Then, I explained the procedure to derive a basic data-driven abstraction and afterwards proposed a novel technique to obtain a probabilistic symbolic abstraction relying solely on heuristics. Finally, I gave an overview on the implementation of my software which implemented these strategies and presented some examples validating the benefits of this approach.

Based on what was presented throughout this paper, we can state that abstraction-based control is a powerful tool that allows us to synthesize controllers for general non-linear dynamical systems. Moreover this systems can also present hybrid behavior and the symbolic controller can also take into account of state and input constraints, as well as constraints that are imposed on the cyber part of the cyber-physical system. Moreover it allows us to synthesize controllers under various control specifications, including safety, reachability, or more complex ones such as those expressed by temporal logic formulas or automata on infinite strings [3].

Nevertheless the classical symbolic control [5] relies on a uniform partition of the

state space and on an over approximation of the reachable sets. For these reasons, this approach is typically model-based and scales poorly with respect to the number of dimensions. In this project we have shown the effectiveness of a data-driven approach and tried to cope with the curse of dimensionality by parallelizing the computation of the abstraction and the synthesis of the control policy. Nevertheless, the strategies presented could be improved by employing an adaptive discretization of the state space as well as adapting the number of samples per cell, building the abstraction lazily, dynamically considering more or fewer inputs, etc. In particular the state-space partition could be refined by considering both local measures, such as the entropy of each node and the value function, and non-local factors like the variance of the system behavior among close cells and the likelihood for the feedback system to end up in each state.

However, as explained in Sec. 3, this approach does not provide formal guarantees. A possible direction to cope with this issue is to combine this strategy with other formal methods in order to improve their performances while keeping the formal guarantees.

Nevertheless, despite these limitations, in Sec. 6 we demonstrated how this method is more effective compared to the classical one in term of performances. Moreover, by translating the notion of PAC approximate alternating simulation relation [9] to our Monte-Carlo method we can derive probabilistic bound on the system behavior.

References

[1] *Dionysos software*. https://github.com/dionysos-dev/Dionysos.jl. Last accessed 08/2022.

[2] Giordano Pola and Maria Domenica Di Benedetto. "Control of Cyber-Physical-Systems with logic specifications: A formal methods approach". In: *Annual Reviews in Control* 47 (2019), pp. 178–192. ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2019.03.010. URL: https://www.sciencedirect.com/science/article/pii/S1367578818302153.

[3] Xu Ding, Stephen Smith, Calin Belta, and Daniela Rus. "MDP Optimal Control under Temporal Logic Constraints". In: *Proceedings of the IEEE Conference on Decision and Control* (March 2011). DOI: 10.1109/CDC.2011.6161122.

[4] Xu Chu Ding, Stephen L. Smith, Calin Belta, and Daniela Rus. *MDP Optimal Control under Temporal Logic Constraints*. 2011. DOI: 10.48550/ARXIV.1103.4342. URL: https://arxiv.org/abs/1103.4342.

[5] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. New York: Springer, 2009.

[6] Gunther Reissig. "Computing Abstractions of Nonlinear Systems". In: *IEEE Transactions on Automatic Control* 56.11 (2011), pp. 2583–2598. DOI: 10.1109/TAC.2011.2118950.

[7] Alexander Weber and Gunther Reissig. "Classical and Strong Convexity of Sublevel Sets and Application to Attainable Sets of Nonlinear Systems". In: *SIAM Journal on Control and Optimization* 52 (September 2014), pp. 2857–2876. DOI: 10.1137/130945983.

[8] Matthias Althoff. "Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars". PhD thesis. July 2010.

[9] Alex Devonport, Adnane Saoud, and Murat Arcak. "Symbolic Abstractions From Data: A PAC Learning Approach". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. 2021, pp. 599–604. DOI: 10.1109/CDC45484.2021.9683316.

[10] Thom Badings, Alessandro Abate, Nils Jansen, David Parker, Hasan Poonawala, and Mariëlle Stoelinga. *Sampling-Based Robust Control of Autonomous Systems with Non-Gaussian Noise*. October 2021.

[11] John Jackson, Luca Laurenti, Eric Frew, and Morteza Lahijanian. "Strategy synthesis for partially-known switched stochastic systems". In: *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*. ACM, May 2021. DOI: 10.1145/3447928.3456649. URL: https://doi.org/10.1145%2F3447928.3456649.

[12] Carl Edward Rasmussen. "Gaussian Processes in Machine Learning". In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_4. URL: https://doi.org/10.1007/978-3-540-28650-9_4.

[13] Giuseppe De Giacomo and Moshe Y. Vardi. "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI '13. Beijing, China: AAAI Press, 2013, pp. 854–860. ISBN: 9781577356332.

[14] Maxence Dutreix and Samuel Coogan. "Specification-Guided Verification and Abstraction Refinement of Mixed Monotone Stochastic Systems". In: *IEEE Transactions on Automatic Control* 66.7 (2021), pp. 2975–2990. DOI: 10.1109/TAC.2020.3014142.

[15] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, 1994.

[16] Richard Bellman. "Dynamic programming and stochastic control processes". In: *Information and Control* 1.3 (1958), pp. 228–239. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(58)80003-0. URL: https://www.sciencedirect.com/science/article/pii/S0019995858800030.

[17] Dimitri Bertsekas. "Dynamic Programming and Optimal Control". In: vol. 1. January 1995.

[18] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley & Sons, Inc., 1994. ISBN: 0471619779.

[19] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[20] Fan-Ming Luo, Tian Xu, Hang Lai, Xiong-Hui Chen, Weinan Zhang, and Yang Yu. *A Survey on Model-based Reinforcement Learning*. 2022. DOI: 10.48550/ARXIV.2206.09328. URL: https://arxiv.org/abs/2206.09328.

[21] *Julia Programming Language*. https://julialang.org.

[22] Gunther Reissig, Alexander Weber, and Matthias Rungger. "Feedback Refinement Relations for the Synthesis of Symbolic Controllers". In: *IEEE Transactions on Automatic Control* 62.4 (2017), pp. 1781–1796. DOI: 10.1109/TAC.2016.2593947.

[23] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin Schmuck. "Multi-Layered Abstraction-Based Controller Synthesis for Continuous-Time Systems". In: *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*. HSCC '18. Porto, Portugal: Association for Computing Machinery, 2018, pp. 120–129. ISBN: 9781450356428. DOI: 10.1145/3178126.3178143. URL: https://doi.org/10.1145/3178126.3178143.

[24] Julien Calbert, Benoît Legat, Lucas N. Egidio, and Raphaël Jungers. "Alternating Simulation on Hierarchical Abstractions". In: *2021 60th IEEE Conference on Decision and Control (CDC)*. 2021, pp. 593–598. DOI: 10.1109/CDC45484.2021.9683448.

[25] Davide De Lazzari. *Abstraction-Based Data-Driven Control Software*. https://github.com/davidedl-ucl/master-thesis. Last updated 08/2022.

[26] *StaticArrays.jl*. https://github.com/JuliaArrays/StaticArrays.jl.

[27] *StaticArrays.jl*. https://docs.juliaplots.org.

[28] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. USA: Cambridge University Press, 1992. ISBN: 0521431085.

[29] Karl Johan Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers, Second Edition*. Princeton, NJ: Princeton University Press, 2008.

[30] Majid Zamani, Giordano Pola, Manuel Mazo, and Paulo Tabuada. "Symbolic Models for Nonlinear Control Systems Without Stability Assumptions". In: *IEEE Transactions on Automatic Control* 57.7 (2012), pp. 1804–1809. DOI: 10.1109/TAC.2011.2176409.