juliacon

Dionysos.jl: a Modular Platform for Smart Symbolic Control

Julien Calbert[1], Adrien Banse[1], Benoît Legat[2], and Raphaël M. Jungers[1]

[1]ICTEAM, UCLouvain
[2]STADIUS, KU Leuven

ABSTRACT

We introduce `Dionysos.jl`, a modular package for solving optimal control problems for complex dynamical systems using state-of-the-art and experimental techniques from symbolic control, optimization, and learning. More often than not with Cyber-Physical systems, the only sensible way of developing a controller is by discretizing the different variables, thus transforming the control task into a purely combinatorial problem on a finite-state mathematical object, called an abstraction of this system. Although this approach offers a safety-critical framework, the available techniques suffer important scalability issues. In order to render these techniques practical, it is necessary to construct smarter abstractions that differ from classical techniques by partitioning the state-space in a non trivial way.

Keywords

Systems and Control, Symbolic Models, Smart abstractions

1. Introduction

In our modern world, the control systems are increasingly complex (think of smart grids, autonomous cars, robots and the internet of things). These systems are at the center of a paradigm shift coined as *the cyber-physical revolution* by the industrial and academic communities [25, 2, 27]. The industry currently applies classical control techniques even if their requirements are no longer met for these systems. This causes a loss of efficiency and a lack of guarantees that are crucial in view of the importance of safety in such systems [5]. As a consequence, many key technological applications run nowadays at sub-optimal regimes.

Formal verification frameworks have been developed using barrier certificates [36, 37] that system trajectories cannot cross, or using reachability analysis [1] to identify potential states that a system can reach. Reachability analysis involves computing an over-approximation of the set of reachable states, which can be done using the toolbox `JuliaReach` [10], for example. However, these techniques are mainly focused on verification, and are restricted to some classes of dynamical systems. In contrast, a renowned approach to address the correct-by-design synthesis relies on *abstractions* (a.k.a. symbolic control) [41], whereby a finite-state machine (also known as "symbolic model") approximates the behavior of the original (a.k.a. "concrete") system that, instead, evolves in a continuous (or even hybrid) state space. This is achieved by defining mathematical relations between the finite state machine and the original dynamics [3, 38]. Several open-source toolboxes [34, 39, 40] have already implemented abstraction-based controllers, including recent toolboxes for analysis and synthesis of abstractions for stochastic systems [33, 43], illustrating the need in industry for such techniques offering safety guarantees. Nevertheless, abstraction suffers from the *curse of dimensionality*, and none of these toolboxes can solve non-academic problems of dimension, say, larger than 3. These limitations of scale explain why abstraction-based techniques have not yet been successful, particularly in the field of robotics.

To address this issue, we introduce `Dionysos.jl`[1], a modular Julia [9] package for solving optimal control problems using state-of-the-art control techniques, from control, optimization, and machine learning, for complex systems. It is built on top of different Julia packages such as `JuMP.jl` and `MathOptInterface.jl`, and features optimal control problem definitions and several abstraction-based methods to solve them. `Dionysos.jl` is the software of the ERC project Learning to control (L2C)[2]. It implements new fundamental techniques that have been designed for smart abstraction [16, 30, 20, 6, 15] as part of the L2C project, with the goal of designing control techniques that would come with guarantees of safety and efficiency, while at the same time being able to take into account non-standard constraints or information, e.g. coming from first principles in physics, a logical specification translating some legal regulations, or some recommendation from a human being.

In recent years, a few groups have proposed ideas to alleviate the computational burdens of the abstraction-based approach, such as for instance the concept of *lazy abstractions* [18, 21, 42, 24]. `Dionysos.jl` provides a general purpose platform allowing to implement such non-standard approaches in a common environment. In particular, it features:

—abstractions that are covers of the concrete space (i.e., not only partitions);

—discretization templates, such as hyperrectangles and ellipsoids;

—controller templates, including piecewise constant controllers or piecewise affine controllers;

—different and novel types of abstraction relations such as alternating simulation relation (ASR), feedback refinement relation (FRR), or memoryless concretization relation (MCR).

[†]Julien Calbert and Adrien Banse are co-first authors with equal contribution and importance.

[1]The package is open source and available on GitHub at: `https://github.com/dionysos-dev/Dionysos.jl`.

[2]European Reasearch Council (ERC) under the European Union's Horizon 2020 research and innovation program under grant agreement No 864017 - L2C.

Our techniques, as well as our software solution, have been validated on academic examples from [22, 35, 38, 23], accessible in the documentation.

This paper is structured as follows. Section 2 introduces the concept of abstraction-based control. Section 3 is devoted to the description of the features of `Dionysos.jl`. Section 4 describes the package structure and provides a description of its main modules. In Sections 5 and 6, we present numerical examples and benchmark comparisons with existing toolboxes, respectively.

Notation: The sets $\mathbb{R}, \mathbb{Z}, \mathbb{Z}_+$ denote respectively the sets of real numbers, integers and non-negative integer numbers. Given two sets A, B, we define a *single-valued map* as $f : A \to B$, while a *set-valued map* is defined as $f : A \to 2^B$, where 2^B is the power set of B, i.e., the set of all subsets of B. The image of a subset $\Omega \subseteq A$ under $f : A \to 2^B$ is denoted $f(\Omega)$. We identify a binary *relation* $R \subseteq A \times B$ with set-valued maps, i.e., $R(a) = \{b \mid (a, b) \in R\}$ and $R^{-1}(b) = \{a \mid (a, b) \in R\}$. A relation $R \subseteq A \times B$ is *strict* (resp. *single-valued)* if for every $a \in A$ the set $R(a) \neq \varnothing$ (resp. $R(a)$ is a singleton).

2. Abstraction-based control

In this section, we provide a concise overview of abstraction-based control, the control approach implemented in `Dionysos.jl`. For a more detailed explanation, please refer to [41, 38].

2.1 Control framework

In this section, we start by defining the considered control framework, i.e., the systems, the controllers and the specifications.

We consider dynamical systems of the following form.

DEFINITION 1. *A* transition control system *is a tuple* $\mathcal{S} = (\mathcal{X}, \mathcal{U}, F)$ *such that*

$$x(k+1) \in F(x(k), u(k)),$$

where $\mathcal{X}$ *and* $\mathcal{U}$ *are respectively the set of states and inputs and the set-valued map* $F : \mathcal{X} \times \mathcal{U} \to 2^{\mathcal{X}}$.

The use of a set-valued map to describe the transition map of a system allows to model perturbations and diverse kinds of non-determinism in a common formalism. In particular, it can be used to model bounded disturbances w, i.e.

$$F(x, u) = \{f(x, u, w) \mid w \in \mathcal{W}\} \tag{1}$$

where $f : \mathcal{X} \times \mathcal{U} \times \mathcal{W} \to \mathcal{X}$ and $\mathcal{W} \subseteq \mathbb{R}^{n_w}$ is a bounded set.

We introduce the set-valued operator of *available inputs*, defined as $\mathcal{U}_F(x) = \{u \in \mathcal{U} \mid F(x, u) \neq \varnothing\}$, which represents the set of inputs u available at a given state x. When it is clear from the context which system it refers to, we simply write the available inputs operator $\mathcal{U}(x)$.

We say that a transition control system is *deterministic* if for every state $x \in \mathcal{X}$ and control input $u \in \mathcal{U}$, $F(x, u)$ is either empty or a singleton. Otherwise, we say that it is *non-deterministic*. A *finite-state* system, in contrast to an *infinite-state* system, refers to a system characterized by finitely many states and inputs.

A tuple $(\boldsymbol{x}, \boldsymbol{u}) \in \mathcal{X}^{T+1} \times \mathcal{U}^T$ is a *trajectory* of length T of the system $\mathcal{S} = (\mathcal{X}, \mathcal{U}, F)$ starting at $x(0)$ if $T \in \mathbb{N} \cup \{\infty\}$, $x(0) \in \mathcal{X}$, $\forall k \in \{0, \ldots, T\} : u(k) \in \mathcal{U}(x(k))$ and $x(k+1) \in F(x(k), u(k))$. The set of trajectories of $\mathcal{S}$ is called the *behavior* of $\mathcal{S}$, denoted $\mathcal{B}(\mathcal{S})$.

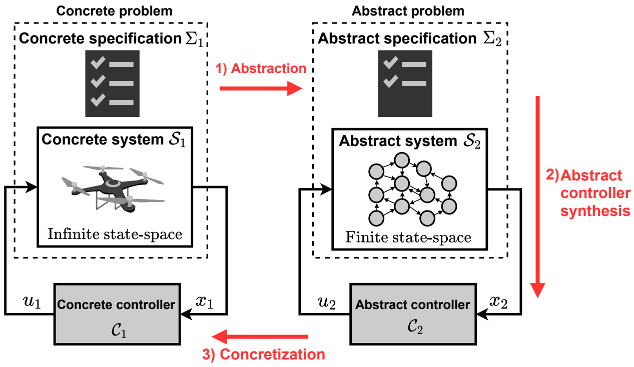

Fig. 1: The three steps of abstraction-based control.

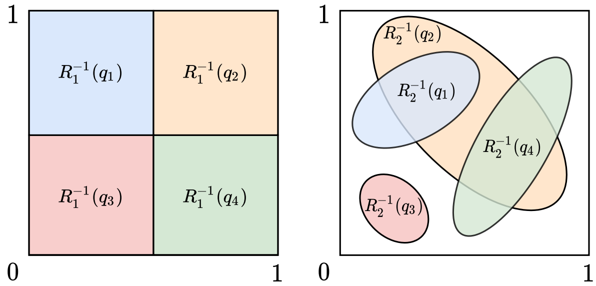

Fig. 2: Types of discretization of the concrete state space. Let $\mathcal{S}_1 = (\mathcal{X}_1, \mathcal{U}_1, F_1)$ with $\mathcal{X}_1 = [0, 1]^2$, $\mathcal{S}_2 = (\mathcal{X}_2, \mathcal{U}_2, F_2)$ with $\mathcal{X}_2 = \{q_1, q_2, q_3, q_4\}$, $R_1 \subseteq \mathcal{X}_1 \times \mathcal{X}_2$ and $R_2 \subseteq \mathcal{X}_1 \times \mathcal{X}_2$ are explicit from the figure. Left: R_1 is a strict single-valued map, i.e., it induces a full partition of $\mathcal{X}_1$. Right: R_2 is a non-strict set-valued map, i.e., it induces a partial cover of $\mathcal{X}_1$.

We now define *static* controllers, which are characterized by the fact that the set of control inputs that the controller can take is determined solely by the current state of the system.

DEFINITION 2. *We define a* static controller *for a system* $\mathcal{S} = (\mathcal{X}, \mathcal{U}, F)$ *as a set-valued map* $\mathcal{C} : \mathcal{X} \to 2^{\mathcal{U}}$ *such that* $\forall x \in \mathcal{X}$: $\mathcal{C}(x) \subseteq \mathcal{U}(x)$, $\mathcal{C}(x) \neq \varnothing$. *We define the* controlled system, *denoted as* $\mathcal{C} \times \mathcal{S}$, *as the transition system characterized by the tuple* $(\mathcal{X}, \mathcal{U}, F_{\mathcal{C}})$ *where* $x' \in F_{\mathcal{C}}(x, u) \Leftrightarrow (u \in \mathcal{C}(x) \wedge x' \in F(x, u))$.

We now define the control problem.

DEFINITION 3. *Consider a system* $\mathcal{S} = (\mathcal{X}, \mathcal{U}, F)$. *A specification* Σ *for* $\mathcal{S}$ *is defined as any subset* $\Sigma \subseteq (\mathcal{X} \times \mathcal{U})^{\infty}$. *It is said that system* $\mathcal{S}$ satisfies *the specification* Σ *if* $\mathcal{B}(\mathcal{S}) \subseteq \Sigma$. *A system* $\mathcal{S}$ *together with a specification* Σ *constitute a* control problem $(\mathcal{S}, \Sigma)$. *Additionally, a controller* $\mathcal{C}$ *is said to* solve *the control problem* $(\mathcal{S}, \Sigma)$ *if* $\mathcal{C} \times \mathcal{S}$ *satisfies the specification* Σ.

2.2 Classical abstraction

Given a mathematical description of the system dynamics and the specifications describing the desired closed-loop behavior of the system, abstraction-based control techniques involve synthesizing a correct-by-construction controller through a systematic three-step procedure illustrated on Figure 1. First, both the original system $\mathcal{S}_1 = (\mathcal{X}_1, \mathcal{U}_1, F_1)$ and the specifications Σ_1 are transposed into an *abstract* domain, resulting in an abstract system $\mathcal{S}_2 = (\mathcal{X}_2, \mathcal{U}_2, F_2)$ and corresponding abstract specifications Σ_2. We refer to the original system as the *concrete* system as opposed to the abstract system. Next, an abstract controller $\mathcal{C}_2$ is synthesized to solve this abstract control problem $(\mathcal{S}_2, \Sigma_2)$. Finally, in the third step, called *concretization* as opposed to abstraction, a controller $\mathcal{C}_1$ for the original control problem is derived from the abstract controller.

The effectiveness of this approach stems from replacing the concrete system, often characterized by an infinite number of states, with a finite state system. This substitution enables the use of powerful control tools in the second step (see [7, 26]), such as those derived from graph theory, including methods like Dijkstra or the A-star algorithm. This facilitates the design of controllers that are correct by construction, often accompanied by rigorous guarantees in terms of safety or performance.

In practice, the abstract domain $\mathcal{X}_2$ of $\mathcal{S}_2$ is constructed by discretizing the concrete state space $\mathcal{X}_1$ of $\mathcal{S}_1$ into subsets (called *cells*). The discretization is induced by a relation $R \subseteq \mathcal{X}_1 \times \mathcal{X}_2$, i.e., the cell associated with the abstract state $x_2 \in \mathcal{X}_2$ is $R^{-1}(x_2) \subseteq \mathcal{X}_1$. Note that in this context, we refer to the set-valued map $R(x_1) = \{x_2 \mid (x_1, x_2) \in R\}$ as the *quantizer*. When R is a single-valued map, we refer to it as defining a *partition* of $\mathcal{X}_1$, in contrast to the case of set-valued maps where we say that it defines a *cover* of $\mathcal{X}_1$, see Figure 2 for clarity. Notably, the condition that R is a strict relation is equivalent to ensuring that the discretization completely covers $\mathcal{X}_1$.

In order to be able to reconstruct a concrete controller $\mathcal{C}_1$ from the abstract controller $\mathcal{C}_2$, the relation R must satisfy some properties. It is shown in [17, Theorem 1] that if R is an *alternating simulation relation* (ASR) [41, Definition 4.19], then it is possible to construct a (possibly non-static) controller $\mathcal{C}_1$ for $\mathcal{S}_1$ from the abstract controller $\mathcal{C}_2$. However, the complexity of the concretization algorithm, and the fact that a non-static controller may have several implementation drawbacks, have motivated researchers to refine the notion of ASR. In particular, most algorithms in the literature [40, 11, 16] rely on the *feedback refinement relation* [38, Def. V.2], which we now define.

DEFINITION 4. *A strict relation R is a* feedback refinement relation *between the systems $\mathcal{S}_1$ and $\mathcal{S}_2$, if for each $(x_1, x_2) \in R$*

(i) $\mathcal{U}_2(x_2) \subseteq \mathcal{U}_1(x_1)$ *;*

(ii) for every $u \in \mathcal{U}_2(x_2)$, $x_1' \in F_1(x_1, u)$, $x_2' \in R(x_1')$: $x_2' \in F_2(x_2, u)$.

This specific relation allows a simple concretization scheme (see [38, V.4 Theorem])

$$\mathcal{C}_1(x_1) = \mathcal{C}_2(R(x_1)). \tag{2}$$

Observe that the requirements (i) and (ii) in Definition 4 restrict the class of concrete controllers to piecewise constant controllers since the control input only depends on the abstract state (2).

2.3 Smart abstraction

The classical abstraction-based approach consists in constructing a feedback refinement relation R based on a predefined partition of the entire state space. The limitation to piecewise constant controllers, combined with the use of a rigid/predefined partition of the entire concrete state space, could result in an intractable, or even unsolvable, abstract problem $(\mathcal{S}_2, \Sigma_2)$. Indeed, if the system does not exhibit local incremental stability [4, Definition 2.1][31] around a cell, meaning that trajectories move away from each other, the use of piecewise constant controllers introduces a significant amount of

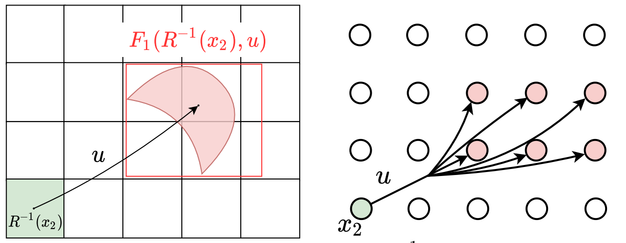

Fig. 3: Piecewise constant controller $\forall x_1 \in R^{-1}(x_2) : \mathcal{C}_1(x_1) = \{u\}$ where $u \in \mathcal{U}_2 \subseteq \mathcal{U}_1$. Left: The two-dimensional concrete system with its state space discretization. Right: The abstract system.

non-determinism into the abstraction as illustrated in Figure 3. Indeed, this results in a high cardinality of the set of outputs of the transition map $F_2(x_2, u)$ of the abstraction.

To address these issues, `Dionysos.jl` provides a framework that generalizes the classical approach by allowing the use of overlapping cells and state-dependent controllers which are defined differently from one cell to another, in a piecewise manner. The design of low-level controllers within cells, in combination with high-level abstraction-based controllers, opens up new possibilities when co-creating the abstraction and the controller, as is done in so-called *lazy abstractions* (i.e., postponing heavier numerical operations).

More precisely, `Dionysos.jl` allows to construct a (non-strict) relation R that is a cover. For this purpose, it computes a *memoryless concretization relation* (MCR) [17, Definition 8] between $\mathcal{S}_1$ and $\mathcal{S}_2$.

DEFINITION 5. *A relation R is a* memoryless concretization relation *between $\mathcal{S}_1$ and $\mathcal{S}_2$, if for each $(x_1, x_2) \in R$*

$$\begin{aligned} &\textit{for every } u_2 \in \mathcal{U}_2(x_2) \textit{ there exists } u_1 \in \mathcal{U}_1(x_1) \textit{ such that} \\ &\quad \textit{for every } x_1' \in F_1(x_1, u_1) : R(x_1') \subseteq F_2(x_2, u_2). \end{aligned} \tag{3}$$

We also define the associated *extended relation* $R_e \subseteq \mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{U}_1 \times \mathcal{U}_2$, which is defined by the set of (x_1, x_2, u_1, u_2) satisfying the condition (3), and an *interface* $I_R : \mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{U}_2 \to 2^{\mathcal{U}_1}$ which maps abstract inputs to concrete ones, i.e.,

$$I_R(x_1, x_2, u_2) = \{u_1 \mid (x_1, x_2, u_1, u_2) \in R_e\}.$$

This relation provides a simple concretization scheme, even in the presence of overlapping cells (see [17] for a complete discussion), which allows the use of state-dependent local controllers within a cell

$$\mathcal{C}_1(x_1) = (\mathcal{C}_2 \circ_{I_R} R)(x_1) = \bigcup_{x_2 \in R(x_1)} I_R(x_1, x_2, \mathcal{C}_2(x_2)).$$

By designing these local state-dependent controllers, for example by solving an optimization problem (e.g., [14, Section V]), we can ensure deterministic transitions in the abstraction, thereby eliminating the non-determinism imposed by the discretization of the concrete system (see Figure 4). In addition, contrary to the classical approach, this technique avoids discretizing the input-space and uses all the available inputs, making it possible to design, given some cost metric, better control solutions.

In order to reduce the number of cells in the abstraction, `Dionysos.jl` computes a goal-specific abstraction by co-designing the abstraction and the controller. This can be done by optimizing the shape of the cells during the construction of the abstraction. For example, the combined use of ellipsoid-based covering and affine local feedback controllers can leverage the power

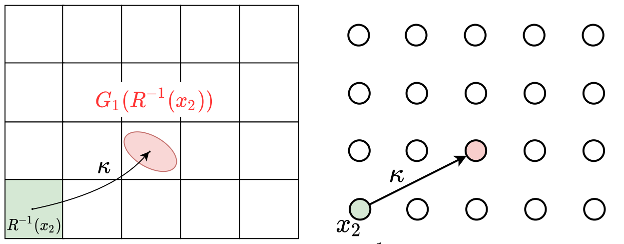

Fig. 4: Piecewise affine controller $\forall x_1 \in R^{-1}(x_2) : \mathcal{C}_1(x_1) = \{\kappa(x_1)\}$ where $\kappa \in \mathcal{U}_2$ and $\kappa(x_1) = K_\kappa x_1 + \ell_\kappa \in \mathcal{U}_1$. The map $G_1(x) = F_1(x, \kappa(x))$ provides the closed-loop transition under the controller κ. Left: The two-dimensional concrete system with its state space discretization. Right: The abstract system.

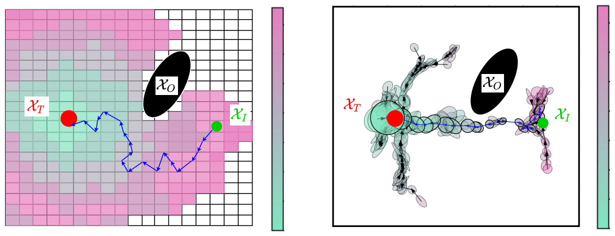

Fig. 5: Comparison between classical and smart abstractions for a planar system with state trajectory (blue line) and value function (color map) obtained for the optimal control problem of departing from $\mathcal{X}_I$ and reaching $\mathcal{X}_T$ while avoiding obstacles $\mathcal{X}_O$. Left: Abstraction covering the entire state space with a naive grid-based partition. Non-colored represents a region where no controller could be designed. Right: Abstraction partially covering the state space with ellipsoidal cells and a local feedback controller.

of linear matrix inequalities (or LMI) and convex optimization to create larger/non-standard cells (see Figure 5).

3. `Dionysos.jl` features

In this section, we present the features currently supported by `Dionysos.jl`.

Systems: `Dionysos.jl` supports bounded disturbances as described in (1) and returns static controllers (see Definition 2) for both the abstract and concrete problems.

Specifications: `Dionysos.jl` supports either *reach-avoid* or *invariance* (safety) specifications. Given a system $\mathcal{S} = (\mathcal{X}, \mathcal{U}, F)$ and sets $\mathcal{X}_I, \mathcal{X}_T, \mathcal{X}_O \subseteq \mathcal{X}$, a *reach-avoid* specification is defined as

$$\Sigma^{\text{Reach}} = \{(\boldsymbol{x}, \boldsymbol{u}) \in (\mathcal{X} \times \mathcal{U})^\infty \mid x(0) \in \mathcal{X}_I \Rightarrow \exists k \in \mathbb{Z}_+ : (x(k) \in \mathcal{X}_T \wedge \forall k' \in [0;k) : x(k') \notin \mathcal{X}_O)\}, \tag{4}$$

which enforces that all states in the initial set $\mathcal{X}_I$ will reach the target $\mathcal{X}_T$ in finite time while avoiding obstacles in $\mathcal{X}_O$. We use the abbreviated notation $\Sigma^{\text{Reach}} = [\mathcal{X}_I, \mathcal{X}_T, \mathcal{X}_O]$ to denote the specification (4). Given sets $\mathcal{X}_I, \mathcal{X}_S \subseteq \mathcal{X}$, an *invariance* specification is defined as

$$\Sigma^{\text{safe}} = \{(\boldsymbol{x}, \boldsymbol{u}) \in (\mathcal{X} \times \mathcal{U})^\infty \mid x(0) \in \mathcal{X}_I \Rightarrow \forall k \in \mathbb{Z}_+ : x(k) \in \mathcal{X}_S\}, \tag{5}$$

which requires that all states in the initial set $\mathcal{X}_I$ remain (safely) in the set $\mathcal{X}_S$ forever. In addition to the specified specifications, a *state cost function* $c : \mathcal{X} \to \mathbb{R}_+$ evaluating the cost of being in a state x, and a *transition cost function* $t : \mathcal{X} \times \mathcal{U} \to \mathbb{R}_+$ quantifying the cost of transitioning from one state x to another with control input u, can be supplied to the control problem. The objective is then to design a controller that satisfies the specification while minimizing the cumulative cost. :

Discretization templates: The *quantizer* R can either be a (partial or not) partition or cover of the concrete state space. In addition, given a continuous state space $\mathcal{X} \subseteq \mathbb{R}^n$, `Dionysos.jl` supports two types of sets for the quantizer:

—a *hyperrectangle* of *center* $c \in \mathbb{R}^n$ and *half-lengths* $h \in \mathbb{R}^n_+$: $\mathrm{H}(c, h) = \{x \in \mathbb{R}^n \mid |x_i - c_i| \leq h_i \text{ for } i = 1, \dots, n\}$;

—an *ellipsoid* with *center* $c \in \mathbb{R}^n$ and *shape* defined by $P \in \mathbb{S}^n_+$ is denoted as $\mathrm{E}(c, P) = \{x \in \mathbb{R}^n \mid (x - c)^\top P(x - c) \leq 1\}$.

4. Package structure

In this section we describe the architecture of `Dionysos.jl`. It is composed of seven root modules. The first three modules, namely `System`, `Problem` and `Optim`, will be described in this paper. For the sake of conciseness, the four other principal modules, namely `Domain`, `Mapping`, `Symbolic` and `Utils`, are skipped but can be found in the package documentation. A summary of this section is given in Figure 6.

4.1 The `System` module

The `System` module contains mathematical descriptions of controlled systems (see Definition 1), controllers (see Definition 2) and trajectories. It is an extension of `MathematicalSystems.jl` and `HybridSystems.jl` [29], it complements the latter with more specific system definitions.

The control systems are described as structures implementing the abstract type `ControlSystem{N, T}`, where `N` and `T` are respectively the dimension and type of the state-space (e.g. `N = 3` and `T = Float64` for a three-dimensional continuous state-space). For example, `ControlSystemLinearized{N, T, F1, F2, F3, F4} <: ControlSystem{N, T}`, where `F1`, `F2`, `F3` and `F4` are subtypes of `Function`, implements a control system whose transition function has been linearized. It has the form

$$\dot{x}(t) \in \tilde{F}(x, u), \tag{6}$$

where $\tilde{F}(x, u)$ is such as in (1) with an additive noise, that is

$$\tilde{F}(x, u) = \left\{ \tilde{f}(x, u) + w \,|\, w \in \mathcal{W} \right\}, \tag{7}$$

where $\mathcal{W} = [-W, W]^{n_x}$, and where $\tilde{f}$ is the linearized version of some possibly nonlinear function f. The system is considered to be sampled with a given time-step, and the corresponding discrete-time transition function is computed with a numerical derivation scheme such as the fourth-order Runge Kutta method, implemented as `RungeKutta4` in `Dionsysos.jl`. The structure `ControlSystemLinearized{N,T,F1,F2,F3,F4}` contains the fields

—`tstep::Float64`, the sampling time-step,

—`measnoise::SVector{N,T}`, the bound W on the disturbance,

—`sys_map::F1`, the sampled possibly nonlinear transition function;

—`linsys_map::F2`, the sampled linearized transition function,

—`error_map::F3`, the difference between `linsys_map` and `sys_map`, and

—`sys_inv_map::F4`, the inverse of `sys_map`.

The controllers implement the abstract type `Controller` and have a field `c_eval` that corresponds to the set-valued function $\mathcal{C}(x)$. For example, the structure `ConstantController{T,VT}` implements controllers of the form $\mathcal{C}(x) = \{c\}$, where c is a constant. It contains the fields `c::VT` that is the constant c, where `VT<:AbstractVector{T}` and `T<:Real`, and `c_eval`.

Finally, the module `System` contains descriptions of trajectories. For example, the structure `ContinuousTrajectory{T, XVT<:AbstractVector{T}, UVT<:AbstractVector{T}}` contains the fields `x::Vector{XVT}` and `u::Vector{UVT}`, and implements trajectories of the form $(\boldsymbol{x}, \boldsymbol{u})$ such as described in Section 2.

4.2 The `Problem` module

The `Problem` module contains the two structures that respectively define the reach-avoid and invariance specification problems in `Dionysos.jl`. Both implement the abstract type `ProblemType`.

`OptimalControlProblem{S,XI,XT,XC,TC,T<:Real}` is the first structure, and implements a reach-avoid problem. Its fields are

—`system::S`, the system to be controlled,

—`initial_set::XI`, the initial set $\mathcal{X}_I$,

—`target_set::XT`, the target set $\mathcal{X}_T$,

—`state_cost::XC`, the state cost function,

—`transition_cost::TC`, the transition cost function, and

—`time::T`, the number of allowed steps.

Note that, in `Dionysos.jl`, the obstacles are encoded as part of the domain of the system. The second structure is `SafetyProblem{S,XI,XS,T<:Real}`, and implements an invariance problem. Its fields are

—`system::S`, the system to be controlled,

—`initial_set::XI`, the set $\mathcal{X}_I$,

—`safe_set::XS`, the safe set $\mathcal{X}_S$,

—`time::T`, the number of allowed steps.

For both structures, the type `S` is typically a system type from the packages `MathematicalSystems.jl` and `HybridSystems.jl` or from the `System` module.

4.3 The `Optim` module

The `Optim` module contains both abstraction-based and classical control strategies. Table 1 gathers the implemented strategies. All strategies are viewed as *solvers* inheriting from `JuMP.jl` [32], a powerful optimization framework embedded in Julia. More precisely, they are all subtypes of the abstract type `MOI.AbstractOptimizer`, and implement the function `MOI.optimize!` from the package `MathOptInterface.jl` [28].

A simple example of an implementation of a classical abstraction-based method is given in the following. In Code 2, the steps given in Figure 1 are followed. In Code 1, the structure for the optimizer is defined. To be initialized, it needs a concrete problem to solve, as well as a discretization for the state space and the input space.

Module	Type	Reference
`BemporadMorari`	Classical	[8]
`BranchAndBound`	Classical	[30]
`AB.UniformGridAbstraction`	Classical abstraction	[38]
`AB.EllipsoidsAbstraction`	Smart abstraction	[20]
`AB.HierarchicalAbstraction`	Smart abstraction	[16]
`AB.LazyAbstraction`	Smart abstraction	[16]
`AB.LazyEllipsoidsAbstraction`	Smart abstraction	[14].

Table 1. : Modules, types and corresponding references of all the control strategies implemented in `Dionysos.jl`, where `AB = Abstraction` in the codebase.

Code 1: Definition of an abstraction-based strategy structure implementing the abstract type `MOI.AbstractOptimizer`.

```
mutable struct Optimizer{T}<:MOI.AbstractOptimizer
    concrete_problem::Union{
        Nothing,
        PR.OptimalControlProblem,
        PR.SafetyProblem
    }
    abstract_problem::Union{
        Nothing,
        PR.OptimalControlProblem,
        PR.SafetyProblem
    }
    abstract_controller::Any
    concrete_controller::Any
    state_grid::Any
    input_grid::Any
    function Optimizer{T}(cp,sg,ig) where {T}
        return new{T}(
            cp,
            nothing,
            nothing,
            nothing,
            nothing,
            sg,
            ig
        )
end
end
```

Code 2: Definition of the `MOI.optimize!` function for the strategy `Optimizer` defined in Code 1.

```
function MOI.optimize!(opt::Optimizer)
    # Build the abstraction
    abstract_system = build_abs_system(
        opt.concrete_problem.system,
        opt.state_grid,
        opt.input_grid,
    )
    # Build the abstract problem
    abstract_problem = build_abs_problem(
        opt.concrete_problem,
        abstract_system
    )
    opt.abstract_problem = abstract_problem
    # Solve the abstract problem
    abstract_controller = solve_abs_problem(
        abstract_problem
    )
    opt.abstract_controller = abstract_controller
    # Solve the concrete problem
    opt.concrete_controller = solve_conc_problem(
        abstract_system,
        abstract_controller
    )
end
```

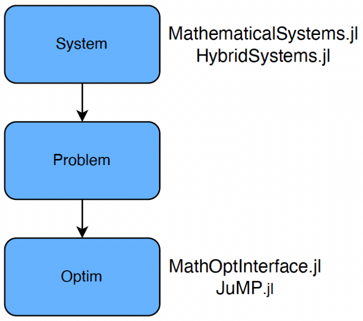

Fig. 6: Summary of Section 4. The `System` module is an extension of `MathematicalSystems.jl` and `HybridSystems.jl`, and implements mathematical definitions of dynamical systems. The `Problem` module contains mathematical definitions of control problems. All problem structures have a system as a field. The `Optim` module contains the control strategies to solve the problems. It is built on top of the optimization packages `MathOptInterface.jl` and `JuMP.jl`.

5. Numerical example

In this section, we provide an example of how `Dionysos.jl` is used to control a nonlinear system with a smart abstraction method. We first define the problem, then solve it with `Dionysos.jl`, and finally we provide visualization results, as recipes are implemented for all visualizable structures[3]. For the sake of conciseness, the code presented in this section is slightly simplified[4].

EXAMPLE 1. *Let $E(c, P) \subseteq \mathbb{R}^n$ be an ellipsoid with center $c \in \mathbb{R}^n$ and shaped defined by $P \in \mathbb{R}^{n\times n}$, that is*

$$E(c, P) := \{x \in \mathbb{R}^n : (x - c)^\top P(x - c) \leq 1\}. \tag{8}$$

We consider a reach-avoid problem. The studied dynamical system is noted $(\mathcal{X} \setminus \mathcal{X}_O, \mathcal{U}, F)$, where $\mathcal{X} = [-20, 20]^2$, $\mathcal{X}_O = E(0, 0.02I_2)$, $\mathcal{U} = [-10, 10]^2$, and $F(x, u) = \{f(x, u)\}$. The function f is defined as

$$f(x, u) = \begin{pmatrix} 1.1x_1 - 0.2x_2 - \mu x_2^3 + u_1 \\ 1.1x_2 + 0.2x_1 + \mu x_1^3 + u_2 \end{pmatrix}, \tag{9}$$

where $\mu = 5\times10^{-5}$. We want to solve a reach-avoid problem where $\mathcal{X}_I = E((-10, -10), 10I_2)$ and $\mathcal{X}_T = E((10, 10), I_2)$. There is no state cost, and the transition cost function is defined as

$$t(x, u) = x^\top x + u^\top u + 1. \tag{10}$$

The system in Example 1, as well as other examples, are already defined in `Dionysos.jl/problems`. The `OptimalControlProblem` defining the problem in Example 1 can be found in the `NonLinear` module in `problems/non_linear.jl`. In Code 3, we import this problem.

Code 3: The problem stated above is imported from the `problems` directory.

```julia
concrete_problem = NonLinear.problem()
concrete_system = concrete_problem.system
```

[3]See `https://docs.juliaplots.org/latest/recipes/` for an introduction on recipes.
[4]See `https://dionysos-dev.github.io/Dionysos.jl/stable/generated/Lazy-Ellipsoids-Abstraction/` for the full example.

We choose to use `AB.LazyEllispoidsAbstraction` (see Table 1) to solve this problem, a smart abstraction method that constructs a memoryless concretization relation (MCR) that partially covers the state space with ellipsoids. In Code 4, we instantiate the corresponding `Optimizer`, then set all the needed fields, including `concrete_problem`, the reach-avoid problem stated above. For the sake of conciseness, we do not state the purpose of the other fields, and we gather them in the variable `other_parameters` in the code.

Code 4: The `Optimizer` is instantiated, and the solver parameters are set.

```julia
optimizer = MOI.instantiate(
    AB.LazyEllipsoidsAbstraction.Optimizer
)
AB.LazyEllipsoidsAbstraction.set_optimizer!(
    optimizer,
    concrete_problem,
    other_parameters...
)
```

In Code 5, we solve the problem and retrieve the corresponding abstract system, abstract problem and concrete controller (see Figure 1). Note that every operation follows the `MathOptInterface.jl` syntax.

Code 5: The problem is solved, and the result is extracted from the solver.

```julia
MOI.optimize!(optimizer)
abstract_system = MOI.get(
    optimizer,
    MOI.RawOptimizerAttribute("abstract_system")
)
abstract_problem = MOI.get(
    optimizer,
    MOI.RawOptimizerAttribute("abstract_problem")
    )
abstract_controller = MOI.get(
    optimizer,
    MOI.RawOptimizerAttribute("abstract_controller")
)
concrete_controller = MOI.get(
    optimizer,
    MOI.RawOptimizerAttribute("concrete_controller")
)
```

In `Dionysos.jl`, we can also generate visualizations thanks to the implemented recipes. In Code 6, we visualize the constructed abstract system, and it gives the plot in Figure 7.

Code 6: The recipe implemented for the abstraction is used to visualize the abstract system.

```julia
fig = plot(aspect_ratio=:equal)
title!(fig, "Abstraction");
plot!(
    fig,
    abstract_system;
    arrowsB = true,
    cost = false
)
```

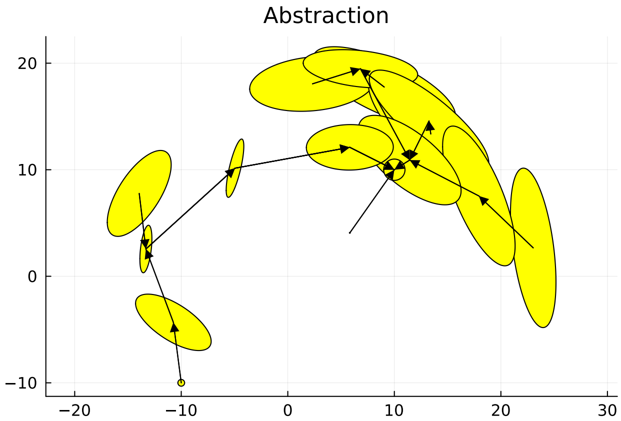

Fig. 7: Result of Code 6.

6. Benchmarking

In order to evaluate the performance of `Dionysos.jl`, we compare the performance of our package against other similar packages, namely SCOTS [40] and CoSyMA [35]. We exclude PESSOA [34, 39] from the comparison as it is outperformed by SCOTS [40]. The code for comparing these packages is published on CodeOcean [13], and is entirely reproducible. For more information, we invite the reader to read the `README.md` file in the CodeOcean capsule.

We reproduced the two numerical experiments of [40]. First, the DC-DC converter example presented in [40, Section 4.2] is reproduced with `Dionysos.jl`, SCOTS and CoSyMA. Thanks to the modularity of `Dionysos.jl`, we can specify to the package that the system is incrementally stable, resulting in sped-up abstraction and synthesis procedures [19]. For the sake of completeness, we also provide the performance of `Dionysos.jl` in the setting where no prior knowledge on the stability is given. The results can be found in Table 2.

	Abstraction [s]	Synthesis [s]	Total [s]
`Dionysos.jl` (no prior)	**1.24**	**3.53**	**4.77**
`Dionysos.jl` (prior)	**0.63**	**2.76**	**3.39**
SCOTS	19.05	74.01	93.06
CoSyMA	—	—	5.31

Table 2. : Comparaison between SCOTS, CoSyMA and `Dionysos.jl` (`AB.UniformGridAbstraction` solver from Table 1) for the DC-DC converter example. `Dionysos.jl` outperforms SCOTS and CoSyMA with and without prior knowledge of the system's incrementally stable property.

Second, the path planning problem presented in [40, Section 4.1] is executed with `Dionysos.jl` and SCOTS. We exclude CoSyMA because this system is not incrementally stable. The results can be found in Table 3.

	Abstraction [s]	Synthesis [s]	Total [s]
`Dionysos.jl`	**8.58**	**6.45**	**15.03**
SCOTS	117.52	480.44	597.96

Table 3. : Comparaison between SCOTS, CoSyMA and `Dionysos.jl` (`AB.UniformGridAbstraction` solver from Table 1) for the path planning example. `Dionysos.jl` outperforms SCOTS.

We see that `Dionysos.jl` outperforms the other packages for these two examples. We also reproduced [40, Figures 3 and 4] in Figure 8 and Figure 9, which shows that they compute the same controller. The visualizations of the DC-DC converter controller with and without prior knwoledge are identical, as it can be verified in [13].

The reason for such a difference between SCOTS, written in C++, and `Dionysos.jl` does not lie in the programming language used to write the package but in the synthesis algorithm itself. For example, unlike SCOTS, our package does not make use of *Binary Decision Diagrams* (or *BDDs*) [12], which as recognized in [40] results in substantially longer execution times compared to tools that use alternative data structures.

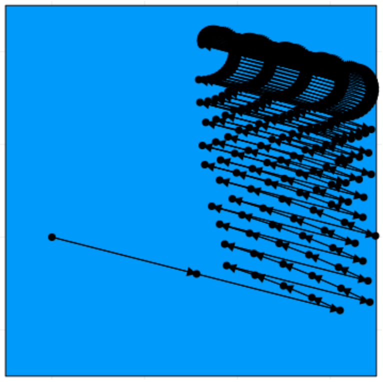
Fig. 8: Reproduction of [40, Figure 4] with `Dionysos.jl`.

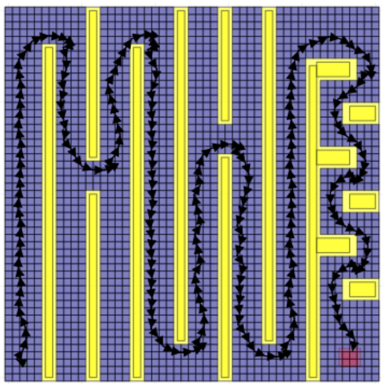
Fig. 9: Reproduction of [40, Figure 3] with `Dionysos.jl`.

7. Conclusions and further work

In this paper, we introduce `Dionysos.jl`, a new software package that provides both a new abstract symbolic representation of the system and controllers with safety guarantees. It generalizes existing toolboxes limited to classical abstractions by allowing the construction of abstractions within the memoryless concretization relation framework, which provides a simple concretization step and the design of low-level controllers. We provide a description of the structure of the package, and describe further the main modules of it. We then show how `Dionysos.jl` can be used in practice by providing a reach-avoid control problem example. Finally, we demonstrated the performance of our package compared to existing similar toolboxes.

As outlined in Section 4 with the array of implemented solvers, the goal of `Dionysos.jl` is to provide a modular environment to facilitate the implementation of new smart abstraction algorithms based for instance on a partial cover of the state-space and the use of piecewise state-dependent controllers.

Because of the hardness of the control problem we aim to solve, the choice of an appropriate solver and its meta-parameters can be self-tuned by end-users. In future work, we plan to design a meta-solver within `Dionysos.jl` which would combine these modules in an ad-hoc and opportunistic approach, thanks to Machine Learning and Artificial Intelligence techniques, in order to exploit the particular problem structures and alleviate the curse of dimensionality.

8. Acknowledgment

JC and AB are FRIA Research Fellows. The research of BL is supported by the European Commission (ERC Adv. Grant) under Grant 885682. RJ is a FNRS honorary Research Associate. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 864017 - L2C.

9. References

[1] Matthias Althoff. *Reachability analysis and its application to the safety assessment of autonomous cars*. PhD thesis, Technische Universität München, 2010.

[2] Rajeev Alur. *Principles of cyber-physical systems*. MIT press, 2015.

[3] Rajeev Alur, Thomas A Henzinger, Orna Kupferman, and Moshe Y Vardi. Alternating refinement relations. In *CONCUR'98 Concurrency Theory: 9th International Conference Nice, France, September 8–11, 1998 Proceedings 9*, pages 163–178. Springer, 1998.

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

[5] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[6] Adrien Banse, Licio Romao, Alessandro Abate, and Raphael Jungers. Data-driven memory-dependent abstractions of dynamical systems. In *Learning for Dynamics and Control Conference*, pages 891–902. PMLR, 2023.

[7] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017.

[8] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, March 1999. doi:10.1016/s0005-1098(98)00178-2.

[9] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[10] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.

[11] Alessandro Borri, Giordano Pola, and Maria Domenica Di Benedetto. Design of symbolic controllers for networked control systems. *IEEE Transactions on Automatic Control*, 64(3):1034–1046, 2018.

[12] Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992.

[13] Julien Calbert, Adrien Banse, Benoît Legat, and Raphaël M. Jungers. Dionysos.jl: a modular platform for smart symbolic control, 2024. doi:10.24433/CO.6327570.V2.

[14] Julien Calbert, Lucas N Egidio, and Raphaël M Jungers. Smart abstraction based on iterative cover and non-uniform cells. *arXiv preprint arXiv:2403.02190*, 2024.

[15] Julien Calbert and Raphaël M Jungers. Data-driven heuristic symbolic models and application to limit-cycle detection. In *2023 American Control Conference (ACC)*, pages 4351–4356. IEEE, 2023.

[16] 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)*, pages 593–598. IEEE, 2021.

[17] Julien Calbert, Sébastien Mattenet, Antoine Girard, and Raphaël M. Jungers. Memoryless concretization relation, 2024. doi:10.48550/ARXIV.2403.09556.

[18] Javier Camara, Antoine Girard, and Gregor Gössler. Safety controller synthesis for switched systems using multi-scale symbolic models. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 520–525. IEEE, 2011.

[19] Javier Cámara, Antoine Girard, and Gregor Gössler. Synthesis of switching controllers using approximately bisimilar multiscale abstractions. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, HSCC '11. ACM, April 2011. doi:10.1145/1967701.1967730.

[20] Lucas N Egidio, Thiago Alves Lima, and Raphaël M Jungers. State-feedback abstractions for optimal control of piecewise-affine systems. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 7455–7460. IEEE, 2022.

[21] Antoine Girard, Gregor Gössler, and Sebti Mouelhi. Safety controller synthesis for incrementally stable switched systems using multiscale symbolic models. *IEEE Transactions on Automatic Control*, 61(6):1537–1549, 2015.

[22] Antoine Girard, Giordano Pola, and Paulo Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2009.

[23] Ebru Aydin Gol, Mircea Lazar, and Calin Belta. Language-guided controller synthesis for linear systems. *IEEE Transactions on Automatic Control*, 59(5):1163–1176, May 2014. doi:10.1109/tac.2013.2295664.

[24] 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)*, pages 120–129, 2018.

[25] Kyoung-Dae Kim and Panganamala R Kumar. Cyber–physical systems: A perspective at the centennial. *Proceedings of the IEEE*, 100(Special Centennial Issue):1287–1308, 2012.

[26] Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal methods in system design*, 19:291–314, 2001.

[27] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT press, 2016.

[28] Benoît Legat, Oscar Dowson, Joaquim Dias Garcia, and Miles Lubin. Mathoptinterface: a data structure for mathematical optimization problems. *INFORMS Journal on Computing*, 34(2):672–689, 2022.

[29] Benoît Legat, Marcelo Forets, Christian Schilling, kpotomkin, and Julia TagBot. blegat/HybridSystems.jl: v0.4.3, 2024. doi:10.5281/ZENODO.10460005.

[30] Benoît Legat, Raphaël M Jungers, and Jean Bouchat. Abstraction-based branch and bound approach to Q-learning for hybrid optimal control. In *Learning for Dynamics and Control*, pages 263–274. PMLR, 2021.

[31] Winfried Lohmiller and Jean-Jacques E Slotine. On contraction analysis for non-linear systems. *Automatica*, 34(6):683–696, 1998.

[32] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15:581–589, 2023. doi:10.1007/s12532-023-00239-3.

[33] Frederik Baymler Mathiesen, Morteza Lahijanian, and Luca Laurenti. IntervalMDP.jl: Accelerated Value Iteration for Interval Markov Decision Processes, 2024. doi:10.48550/ARXIV.2401.04068.

[34] Manuel Mazo Jr, Anna Davitian, and Paulo Tabuada. Pessoa: A tool for embedded controller synthesis. In *International conference on computer aided verification*, pages 566–569. Springer, 2010.

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

[36] Stephen Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006.

[37] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *International Workshop on Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

[38] Gunther Reissig, Alexander Weber, and Matthias Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4):1781–1796, 2016.

[39] Pritam Roy, Paulo Tabuada, and Rupak Majumdar. Pessoa 2.0: a controller synthesis tool for cyber-physical systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, HSCC '11. ACM, April 2011. doi:10.1145/1967701.1967748.

[40] Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th international conference on hybrid systems: Computation and control*, pages 99–104, 2016.

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

[42] Yuichi Tazaki and Jun-ichi Imura. Discrete-state abstractions of nonlinear systems using multi-resolution quantizer. In *International Workshop on Hybrid Systems: Computation and Control*, pages 351–365. Springer, 2009.

[43] Ben Wooding and Abolfazl Lavaei. IMPaCT: Interval MDP Parallel Construction for Controller Synthesis of Large-Scale Stochastic Systems, 2024. doi:10.48550/ARXIV.2401.03555.