femington

*Int. J. Artif. Intell. Agent Syst. (2026) 1(1):1–13*
Article ID: IJAIAS-2026-780

# SmartSnake: An Adaptive Framework for Intelligent Control Synthesis

Prashanth Chevva
prashanthch6034@gmail.com
Independent Researcher

## Abstract

In this paper, we present SmartSnake, a flexible software environment based on Dionysos.jl for the intelligent synthesis of abstract-based control of complex dynamical systems. The platform offers tools that allow researchers and practitioners to design custom abstraction algorithms, which support partial state-space coverage and state-dependent controllers. The modular architecture of SmartSnake allows for an easy integration of machine learning and artificial intelligence to help develop future meta-solvers that will adapt intelligently to the structure of problems and overcome the curse of dimensionality as well. The solvers that can be utilized in SmartSnake are evaluated with respect to how practical they are and their computational efficiency for control synthesis scenarios. We will also present ongoing work on improving the platform with data-driven methods and heuristic symbolic models. SmartSnake provides a flexible framework for both reinforcement learning approaches and classical optimization of controllers.

## Keywords

• Symbolic Control • Abstraction-Based Control Synthesis • Smart Abstractions • Memoryless Concretization Relations • Cyber-Physical Systems • Intelligent Control Frameworks

## 1. Introduction

### 1.1 The Challenge of Modern Control Systems

Today's technological landscape features increasingly sophisticated control systems across smart grids, autonomous vehicles, robotic systems, and the Internet of Things [1, 2]. These cyber-physical systems, as they have come to be known in both academic and industrial circles, require advanced coordination between computational algorithms and physical processes [3]. While classical control methodologies remain widely employed in industry, they increasingly fall short of meeting the stringent requirements imposed by modern applications. The safety-critical nature of many of these systems means that suboptimal performance can have serious consequences, with numerous important technologies operating well below their theoretical potential.

### 1.2 Formal Methods and Abstraction-Based Control

Formal verification frameworks offer rigorous approaches to ensuring system correctness [4]. Barrier certificate methods mathematically prove that system trajectories cannot enter unsafe regions [5, 6], while reachability analysis identifies all possible states a system might occupy [7]. Tools like `JuliaReach` enable computation of over-approximated reachable sets [8]. However, these verification-focused approaches face limitations when applied to certain dynamical system classes.

 

Abstraction-based control, also known as symbolic control, presents an alternative methodology for correct-by-design synthesis [9]. This approach constructs finite-state abstractions—symbolic models—that approximate the behavior of original continuous or hybrid systems. Mathematical relationships, such as alternating simulation relations or feedback refinement relations, formally connect these abstractions to their concrete counterparts [10, 11]. Several open-source toolboxes have implemented abstraction-based controllers, including SCOTS [12], CoSyMA [13], and PESSOA [14, 15], yet the curse of dimensionality severely restricts their applicability, with few tools capable of handling problems beyond three dimensions. This scalability barrier has prevented widespread adoption, particularly in robotics applications.

### 1.3 Dionysos.jl: A New Direction

`Dionysos.jl` emerges as a Julia package addressing optimal control problems through integrated state-of-the-art techniques spanning control theory, optimization, and machine learning [16]. Built upon established packages like `JuMP.jl` and `MathOptInterface.jl` [17, 18], it offers comprehensive problem definitions and multiple abstraction-based solution methods. The package originates from the European Research Council's Learning to Control (L2C) project, which aims to develop control techniques offering safety guarantees while accommodating non-standard constraints and information sources—whether derived from physical first principles, regulatory requirements, or human recommendations [19–24].

### 1.4 Innovations in Abstraction Techniques

Recent research has proposed various strategies to reduce the computational burden of abstraction-based approaches, including the concept of lazy abstractions and memoryless concretization relations [24, 25]. `Dionysos.jl` provides a general-purpose environment for implementing these innovative approaches, featuring:

- Abstractions as covers rather than strict partitions of the state space,
- Flexible discretization templates including hyperrectangles and ellipsoids,
- Controller templates ranging from piecewise constant to piecewise affine,
- Novel abstraction relations such as alternating simulation, feedback refinement, and memoryless concretization relations.

Our methods and software have been validated on academic benchmarks, as documented in the package's examples. The remainder of this paper is organized as follows. Section 2 provides background on abstraction-based control. Section 3 describes the key features of `Dionysos.jl`. Section 4 presents the package architecture and main modules. In Sections 5 and 6, we illustrate usage with numerical examples and compare performance against existing toolboxes.

**Notation:** Let $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{Z}_+$ denote the sets of real numbers, integers, and non-negative integers, respectively. For sets $A$ and $B$, a single-valued map is $f : A \to B$, while a set-valued map is $f : A \to 2^B$. We identify a binary relation $R \subseteq A \times B$ with its associated set-valued map $R(a) = \{b \mid (a, b) \in R\}$. A relation is total if $R(a) \neq \emptyset$ for all $a \in A$, and single-valued if $R(a)$ is a singleton.

## 2. Abstraction-Based Control Fundamentals

This section outlines the core concepts of abstraction-based control as realized in `Dionysos.jl`. Further details can be found in the referenced literature [9, 26, 27].

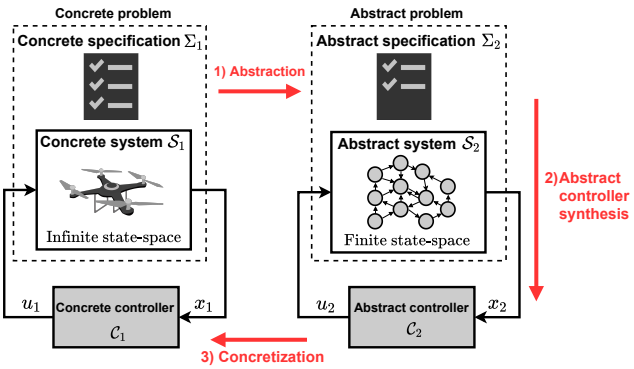


Figure 1: Overview of abstraction-based control synthesis steps.

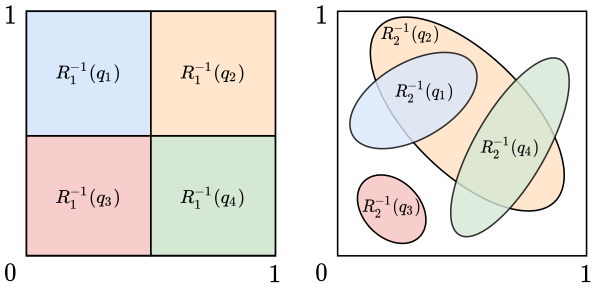


Figure 2: Comparison of discretization types. Left: a total single-valued relation inducing a full partition. Right: a non-total set-valued relation forming a partial cover.

## 2.1 System Model

We consider discrete-time transition systems defined as follows:

**Definition 1.** *A* transition control system *is a tuple* $\mathcal{S} = (X, U, F)$ *where* $X$ *and* $U$ *are state and input sets, and* $F : X \times U \to 2^X$ *is a set-valued transition map. The evolution satisfies*

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

The set-valued nature of $F$ captures non-determinism or bounded disturbances. For a disturbance set $W$, we may write

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

The set of admissible inputs at state $x$ is $\mathcal{U}_F(x) = \{u \in U \mid F(x, u) \neq \emptyset\}$. The system is deterministic if $F(x, u)$ is either a singleton or empty.

A trajectory of length $T + 1$ is a pair $(\mathbf{x}, \mathbf{u}) \in X^{T+1} \times U^T$. A static controller is a set-valued map $C : X \to 2^U$ such that $\emptyset \neq C(x) \subseteq \mathcal{U}_F(x)$ for all $x$. The controlled system $C \times \mathcal{S}$ has transitions $x' \in F(x, u)$ with $u \in C(x)$.

A control problem is a pair $(\mathcal{S}, \Sigma)$ where $\Sigma$ is a specification over infinite trajectories. A controller solves the problem if all controlled trajectories satisfy $\Sigma$.

## 2.2 Classical Abstraction Framework

The abstraction-based approach replaces the original system $\mathcal{S}_1$ with a finite-state abstraction $\mathcal{S}_2$ and a relation $R \subseteq X_1 \times X_2$ that preserves relevant properties. Figure 1 illustrates the three-step process: (1)

construct an abstraction and translate the specification, (2) synthesize a controller for the abstraction, and (3) refine the abstract controller to the concrete system.

To enable controller refinement, the relation $R$ must satisfy certain simulation conditions. The feedback refinement relation (FRR) is defined as follows [11]:

**Definition 2.** *A relation $R \subseteq X_1 \times X_2$ is a* feedback refinement relation *if for every $(x_1, x_2) \in R$:*

1. $\mathcal{U}_2(x_2) \subseteq \mathcal{U}_1(x_1)$;
2. *for all $u \in \mathcal{U}_2(x_2)$ and all $x_1' \in F_1(x_1, u)$, there exists $x_2' \in R(x_1')$ such that $x_2' \in F_2(x_2, u)$.*

Under an FRR, a concrete controller can be obtained simply as $C_1(x_1) = C_2(R(x_1))$.

### 2.3 Smart Abstraction and Memoryless Concretization

Classical approaches rely on predefined partitions, which limit controller performance. `Dionysos.jl` overcomes this by allowing overlapping cells, state-dependent controllers, and lazy construction of abstractions. Instead of FRR, the package computes a memoryless concretization relation (MCR) [25]:

**Definition 3.** *A relation $R \subseteq X_1 \times X_2$ is an* MCR *if for all $(x_1, x_2) \in R$:*

$$\forall u_2 \in \mathcal{U}_2(x_2),\ \exists u_1 \in \mathcal{U}_1(x_1),\ \forall x_1' \in F_1(x_1, u_1) :\ R(x_1') \subseteq F_2(x_2, u_2). \tag{2}$$

An interface map $I_R$ then allows reconstruction of the concrete controller as $C_1(x_1) = (C_2 \circ_{I_R} R)(x_1)$.

By combining local feedback controllers and flexible covers, SmartSnake reduces non-determinism and avoids discretizing the input space. This enables more efficient, objective-oriented abstractions, including ellipsoidal covers and convex optimization formulations (see Figure 5) [24].

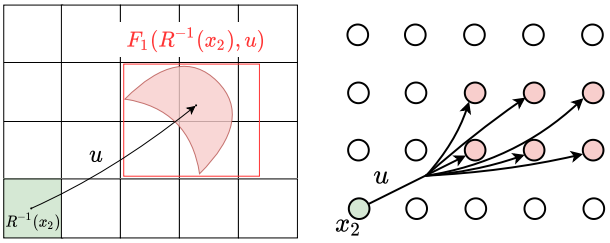


Figure 3: Piecewise constant controller: each cell is assigned a single input.

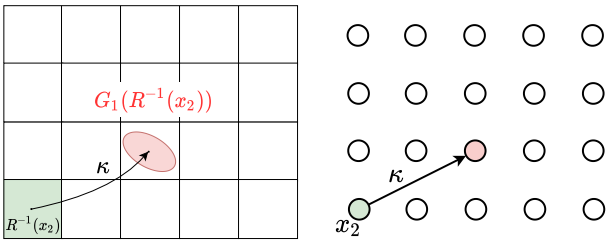


Figure 4: Piecewise affine controller: within each cell, the input is an affine function of the state.

## 3. Features of `Dionysos.jl`

This section summarizes the main features supported by the package.

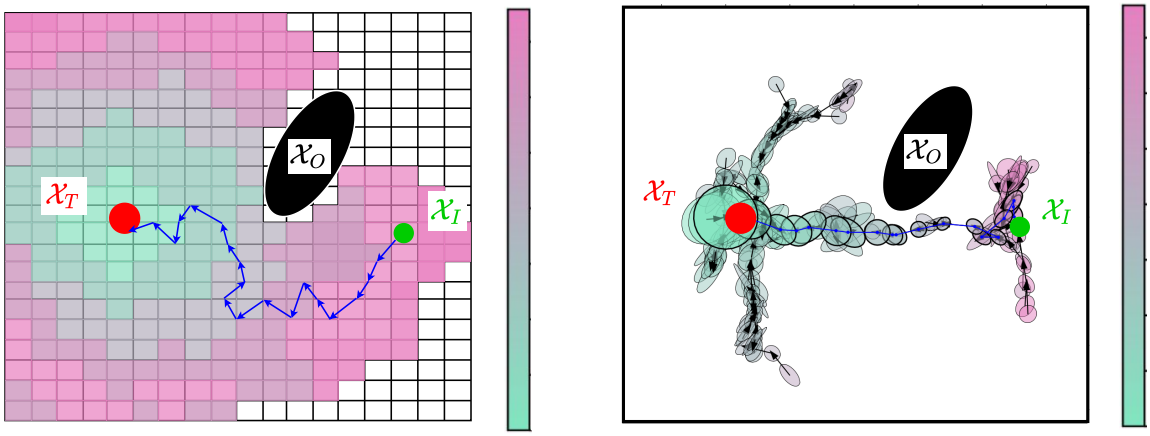


Figure 5: Comparison of classical grid-based abstraction (left) and smart ellipsoidal abstraction with local feedback (right) for a planar reach-avoid problem. Non-colored regions indicate where no controller could be designed.

**System Types** The package supports systems with bounded disturbances as in (1) and returns static controllers for both abstract and concrete levels.

**Specifications** Two classes of specifications are supported: reach-avoid and invariance (safety). A reach-avoid specification is defined by initial set $X_I$, target set $X_T$, and obstacle set $X_O$: all trajectories starting in $X_I$ must reach $X_T$ in finite time while avoiding $X_O$. An invariance specification requires all trajectories starting in $X_I$ to remain forever within a safe set $X_S$. Additionally, state and transition cost functions can be provided to optimize cumulative cost while satisfying the specification.

**Discretization Templates** The quantizer $R$ can be a partial or full partition/cover of the state space. Two geometric primitives are supported:

- Hyperrectangles: $\mathcal{H}(c, h) = \{x \in \mathbb{R}^n \mid |x_i - c_i| \leq h_i\}$,
- Ellipsoids: $\mathcal{E}(c, P) = \{x \in \mathbb{R}^n \mid (x - c)^\top P(x - c) \leq 1\}$, with $P \succ 0$.

## 4. Package Architecture

We describe the main modules of `Dionysos.jl`. The architecture comprises seven core modules; we focus on three—`System`, `Problem`, and `Optim`—while the remaining are documented in the package manual. Figure 6 provides a high-level overview.

### 4.1 The `System` Module

This module provides mathematical representations of control systems, controllers, and trajectories. It extends `MathematicalSystems.jl` and `HybridSystems.jl` [28].

Control systems are subtypes of the abstract type `ControlSystem{N,T}`, where $N$ is the state dimension and $T$ the numeric type. For example, `ControlSystemLinearized` implements a linearized system:

$$\dot{x}(t) \in \tilde{F}(x, u), \quad \tilde{F}(x, u) = \{\tilde{f}(x, u) + w \mid w \in [-W, W]^{n_x}\},$$

where $\tilde{f}$ is a linearization of a possibly nonlinear $f$. Sampling is performed using a numerical integration scheme (e.g., Runge–Kutta 4). The structure includes fields for time step, measurement noise bound, and various maps (system, linearized, error, and inverse).

A simplified definition is shown in Listing 1.

Listing 1: Simplified struct for a linearized control system.

```
struct ControlSystemLinearized{N,T,F1,F2,F3,F4} <: ControlSystem{N,T}
    tstep::Float64
    measnoise::SVector{N,T}
    sys_map::F1
    linsys_map::F2
    error_map::F3
    sys_inv_map::F4
end
```

Controllers implement the abstract type `Controller` with a field `c_eval` representing $C(x)$. For instance, `ConstantController` implements $C(x) = \{c\}$.

Trajectories are represented by structures like `ContinuousTrajectory`, which store sequences of states and inputs.

### 4.2 The `Problem` Module

This module defines reach-avoid and safety problems as subtypes of `ProblemType`.

The `OptimalControlProblem` structure for reach-avoid contains:

- `system`: the system model,
- `initial_set`: $X_I$,
- `target_set`: $X_T$,
- `state_cost`, `transition_cost`: cost functions,
- `time`: horizon length.

The `SafetyProblem` structure for invariance includes initial set, safe set, and horizon.

### 4.3 The `Optim` Module

This module hosts both classical and abstraction-based optimization strategies. Table 1 lists the implemented solvers. All solvers inherit from `MOI.AbstractOptimizer` and implement `MOI.optimize!`.

Table 1: Control strategies implemented in `Dionysos.jl`; `AB` denotes `Abstraction`.

| Module | Type | Reference |
|---|---|---|
| `BemporadMorari` | Classical | [29] |
| `BranchAndBound` | Classical | [20] |
| `AB.UniformGridAbstraction` | Classical abstraction | – |
| `AB.EllipsoidsAbstraction` | Smart abstraction | [24] |
| `AB.HierarchicalAbstraction` | Smart abstraction | [19] |
| `AB.LazyAbstraction` | Smart abstraction | [24] |
| `AB.LazyEllipsoidsAbstraction` | Smart abstraction | [24] |

Listing 2: Optimizer structure.

```
mutable struct Optimizer{T} <: MOI.AbstractOptimizer
    cp::Union{Nothing, PR.OptimalControlProblem, PR.SafetyProblem}
    ap::Union{Nothing, PR.OptimalControlProblem, PR.SafetyProblem}
    abs_ctrl::Any
    con_ctrl::Any
    sg::Any
    ig::Any
end
```

Listing 3: Implementation of `MOI.optimize!` for the abstraction strategy.

```
function MOI.optimize!(opt::Optimizer)
    abstract_system = build_abs_system(opt.concrete_problem.system,
                                       opt.state_grid,
                                       opt.input_grid)
    abstract_problem = build_abs_problem(opt.concrete_problem, abstract_system)
    opt.abstract_problem = abstract_problem
    abstract_controller = solve_abs_problem(abstract_problem)
    opt.abstract_controller = abstract_controller
    opt.concrete_controller = solve_conc_problem(abstract_system, abstract_controller)
end
```

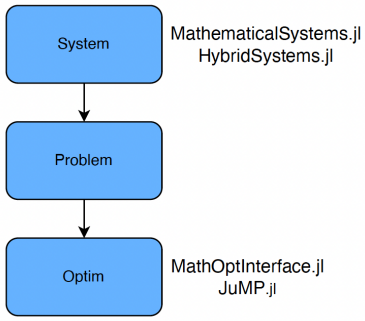


Figure 6: Overview of the package architecture. The `System` module extends existing packages; the `Problem` module defines control problems; the `Optim` module provides solvers built on `MathOptInterface.jl` and `JuMP.jl`.

## 5. Numerical Example

We demonstrate the usage of `Dionysos.jl` on a nonlinear system using a smart abstraction method. We define the problem, solve it, and visualize the results with built-in plotting recipes.

The example problem is drawn from the package's examples (see the online documentation for the full code). We first import the problem:

Listing 4: Importing the pre-defined reach-avoid problem.

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

We then instantiate the `AB.LazyEllipsoidsAbstraction` solver and set its parameters:

Listing 5: Setting up the optimizer for lazy ellipsoidal abstraction.

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

Solving the problem and extracting results follows the `MathOptInterface` convention:

Listing 6: Solving and retrieving the abstract system, problem, and controllers.

```
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"))
```

Finally, we visualize the abstraction using the package's plotting recipes:

Listing 7: Visualizing the abstract system with arrows.

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

The resulting plot is shown in Figure 7.

## 6. Benchmark Evaluation

To assess the performance of `Dionysos.jl`, we compare it against other state-of-the-art toolboxes, namely SCOTS [12] and CoSyMA [13] (PESSOA is excluded as it is outperformed by SCOTS). The benchmark code is publicly available and fully reproducible.

We replicated two numerical experiments from the literature [13, 27]. The first is the DC-DC converter example [27]. We ran the uniform grid abstraction solver in `Dionysos.jl` with and without exploiting incremental stability [30]. The results are shown 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: Comparison on the DC-DC converter example. `Dionysos.jl` outperforms SCOTS and CoSyMA.

The second example is a vehicle path planning problem (not incrementally stable). CoSyMA is excluded. Results are given in Table 3.

These improvements are not attributable to the programming language (C++ vs. Julia) but rather to the synthesis algorithm; for instance, SCOTS uses Binary Decision Diagrams [31], which can lead to longer runtimes.

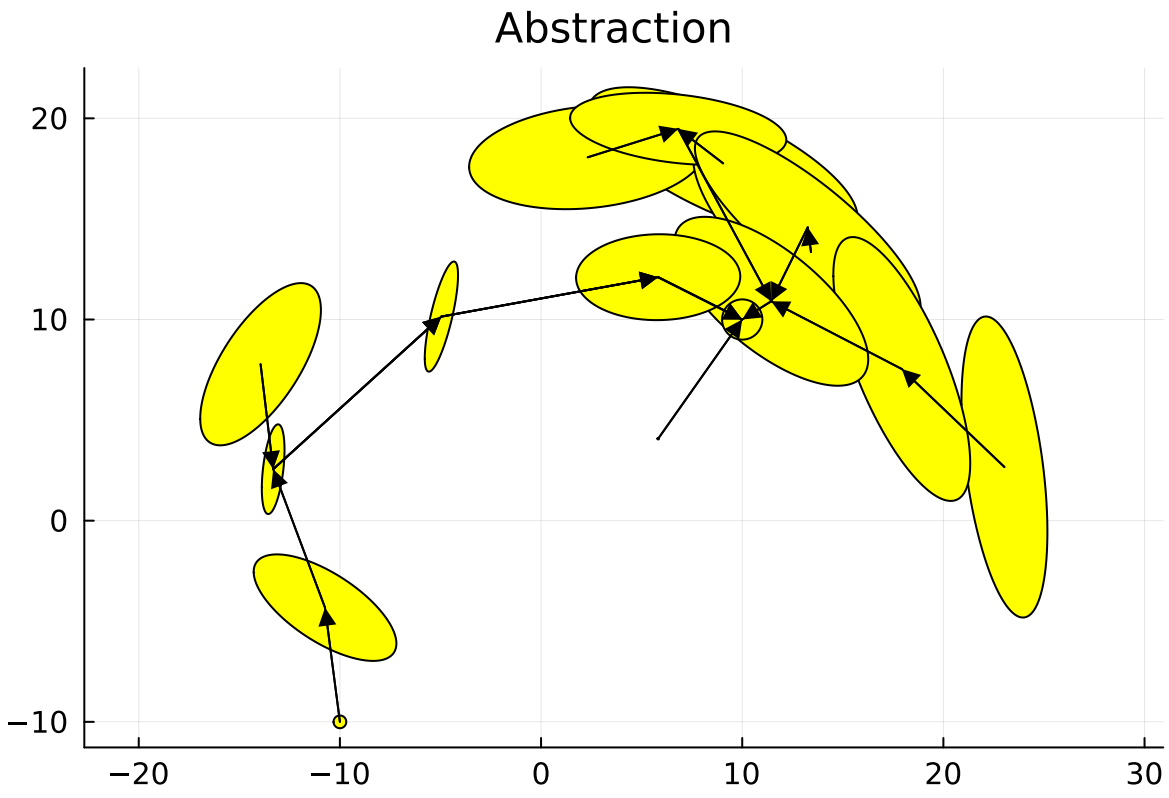


Figure 7: Visualization of the lazy ellipsoidal abstraction for the reach-avoid problem.

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

Table 3: Comparison on the path planning example.

We reproduce the controller visualizations from the original papers in Figures 8 and 9, confirming that the computed controllers match those in the literature.

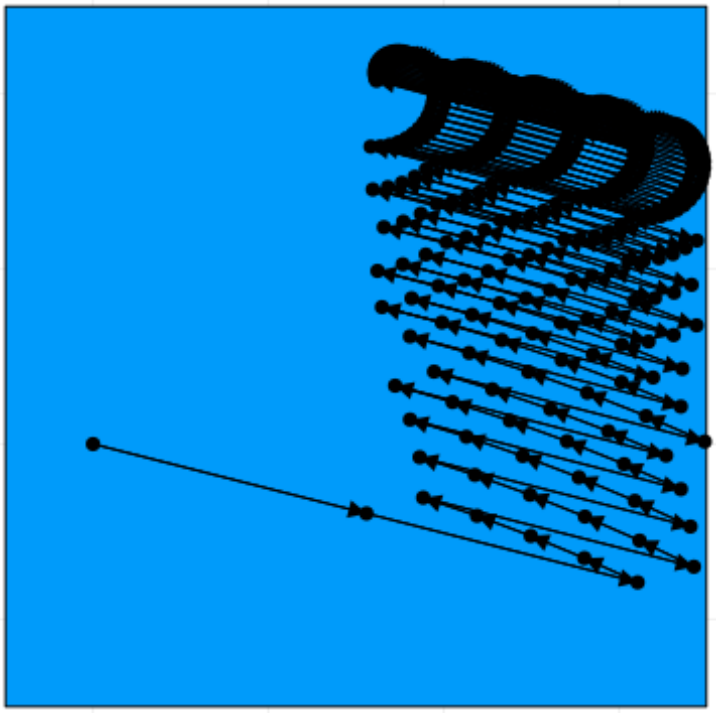

Figure 8: Reproduced DC-DC converter controller from the literature [27].

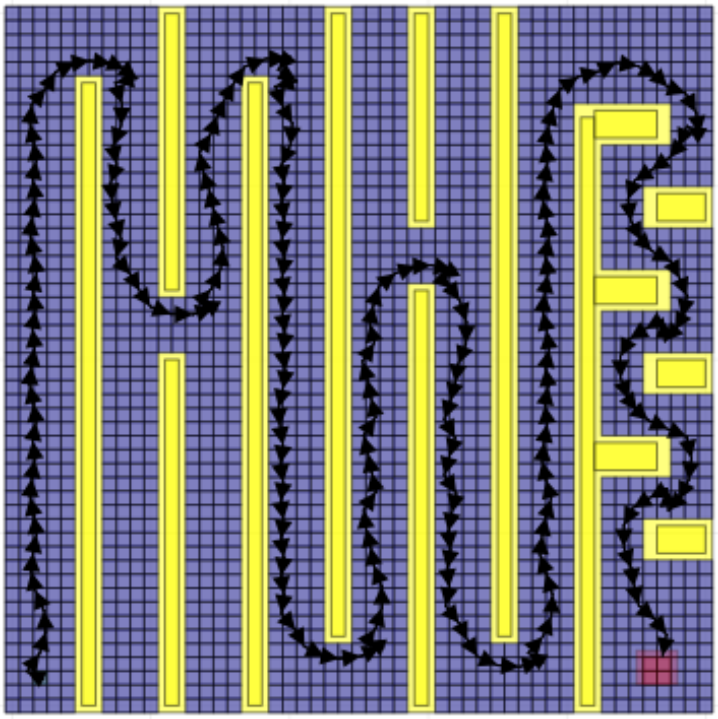

Figure 9: Reproduced vehicle path planning controller from [27].

## 7. Conclusions and Future Directions

We have presented `Dionysos.jl`, a new software package that introduces flexible abstraction-based control synthesis with safety guarantees. By leveraging memoryless concretization relations [25], it generalizes existing toolboxes that are restricted to classical abstraction paradigms. The modular design

enables straightforward implementation of novel smart abstraction algorithms, including those based on partial covers and state-dependent feedback [24].

Ongoing work focuses on the development of a meta-solver within `Dionysos.jl` that will harness machine learning and artificial intelligence to adaptively select and configure solvers according to problem structure, thereby alleviating the curse of dimensionality and expanding the applicability of abstraction-based methods to larger-scale systems.

## References

[1] 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. doi: 10.1109/JPROC.2012.2189792.

[2] Rajeev Alur. *Principles of cyber-physical systems*. MIT press, 2015. doi: 10.1017/9781107588981.

[3] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to embedded systems: A cyber-physical systems approach*. MIT press, 2016. doi: 10.1145/1719010.1719011.

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

[5] Stephen Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006. doi: 10.1016/j.automatica.2005.08.007.

[6] 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. doi: 10.1007/978-3-540-24743-2_32.

[7] Matthias Althoff. *Reachability analysis and its application to the safety assessment of autonomous cars*. PhD thesis, Technische Universität München, 2010. URL `https://mediatum.ub.tum.de/963752`.

[8] 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. doi: 10.1145/3302504.3311804.

[9] Paulo Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009. doi: 10.1007/978-1-4419-0224-5.

[10] 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. doi: 10.1007/BFb0055622.

[11] 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. doi: 10.1109/CDC.2014.7039364.

[12] 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. doi: 10.1145/2883817.2883834.

[13] 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. doi: 10.1145/2461328.2461343.

[14] 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. doi: 10.1007/978-3-642-14295-6_49.

[15] 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.

[16] Julien Calbert, Adrien Banse, Benoît Legat, and Raphaël M. Jungers. Dionysos.jl: a modular platform for smart symbolic control, 2024.

[17] 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.

[18] 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. doi: 10.1287/ijoc.2021.1067.

[19] 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. doi: 10.1109/CDC45484.2021.9683448.

[20] 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. doi: 10.48550/arXiv.2011.11029.

[21] 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. doi: 10.1109/CDC51059.2022.9992495.

[22] 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. doi: 10.48550/arXiv.2212.01926.

[23] 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. doi: 10.23919/ACC55779.2023.10156175.

[24] 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. doi: 10.48550/arXiv.2403.02190.

[25] Julien Calbert, Sébastien Mattenet, Antoine Girard, and Raphaël M. Jungers. Memoryless concretization relation, 2024.

[26] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal methods for discrete-time dynamical systems*, volume 89. Springer, 2017. doi: 10.1007/978-3-319-50763-7.

[27] 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. doi: 10.1109/TAC.2009.2034922.

[28] Benoît Legat, Marcelo Forets, Christian Schilling, kpotomkin, and Julia TagBot. blegat/HybridSystems.jl: v0.4.3, 2024.

[29] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, March 1999. ISSN 0005-1098. doi: 10.1016/s0005-1098(98)00178-2.

[30] David Angeli. A Lyapunov approach to incremental stability properties. *IEEE Transactions on Automatic Control*, 47(3):410–421, 2002. doi: 10.1109/9.989067.

[31] Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3):293–318, 1992. doi: 10.1145/136035.136043.