Dionysos.jl: a Modular Platform for Smart Symbolic Control

Julien Calbert, Adrien Banse, Raphaël M. Jungers
ICTEAM/INMA

October 5, 2023
Julia and Optimization Days 2023
Control theory

The goal is to provide a generic procedure to design efficient controllers with formal guarantees.
A Paradigm Shift in Control Theory

Classical applications made the golden age of Control Theory

However modern applications are increasingly complex...

State space representation unleashed analytic approaches

\[
\begin{bmatrix}
\dot{x}_1 \\
\dot{x}_2 \\
\dot{x}_3
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
-5 & -26 & -5
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
1
\end{bmatrix} u
\]

\[
y = \begin{bmatrix}
1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
\]

Cyber-Physical Systems paradigm shift

... and so are their models

Cyber-Physical Systems paradigm shift
1. Origin of the project (L2C)
2. Abstraction-based control
3. Toolbox
4. Benchmarks
5. Conclusion
Learning to control (L2C)

We need a new control paradigm

Smart and Data-Driven Formal Methods for Cyber-Physical Systems control

- Generic but modular
- Opportunistic
- State-space driven
- Safety-critical
- Scalable
- Logic-enhanced
- Data-friendly
- ...

European Research Council
Established by the European Commission
Abstraction-based control

1) Abstraction

2) Abstract controller design

3) Concretization
Classical abstraction-based control

Additional non-determinism resulting from the discretization
The number of cells grows exponentially with the dimension of the state space

Curse of dimensionality
Co-design the abstraction and the controller

⇒ Partially discretize the state space with non-uniform cells with respect to the specific control task.
The objectives of Dionysos are as follows

- Implement our state-of-the-art smart abstraction algorithms developed in L2C.
- Implement existing algorithms in our modular framework and demonstrate the effectiveness of the Julia language.

Main contributors: Benoît Legat, Julien Calbert, Adrien Banse, Lucas N. Egidio
Dionysos in Julia

Hybrid optimal control solver combining:
- Smart abstraction
- Q-learning
- Branch and bound
- Path-Complete
- Sum-of-Squares
- ...

[Diagrams and logos related to JuliaReach, JUMP, Julia, JuliaRobotics]
Package structure

Control

Domain

Utils

Problem

Dionysos

Mapping

Optim

System

Symbolic

MathOptInterface.jl
JuMP.jl

MathematicalSystems.jl
HybridSystems.jl
Package structure: System

- Structures for mathematical definitions of
  - Control dynamical systems $x^{+} = f(x, u)$
  - Controllers $u(x) = K(x)$

- Methods
  - For example: Runge Kutta scheme

- Built on top of
  - JuliaReach/MathematicalSystems.jl
  - blegat/HybridSystems.jl
• Structures to define control problems
• For now, two types of problem
  - OptimalControlProblem: initial, target and cost
  - SafetyProblem: safe/unsafe sets
• Each problem is composed of a system, and problem specifications
Package structure: Optim

- Methods to solve the problems, the optimizers
- src/optim
  - abstraction
    - SCOTS_abstraction.jl
    - ellipsoids_abstraction.jl
    - hierarchical_abstraction.jl
    - lazy_abstraction.jl
    - lazy_ellipsoids_abstraction.jl
    - bemporad_morari.jl
    - branch_and_bound.jl
    - q_learning.jl
- Built on top of
  - jump-dev/MathOptInterface.jl
  - jump-dev/JuMP.jl

Every optimizer is a subtype of MOI.AbstractOptimizer

- Each optimizer is composed of a problem, and method specifications
Examples

Now, let’s focus on two examples

1. Implementation of a smart abstraction method on a simple problem
2. Implementation of a abstraction method from state-of-the-art, and comparison to existing implementations
First example: Simple problem

Consider the very simple discrete-time system

\[ x_{t+1} = x_t + hu, \]

where \( h \in \mathbb{R} \) is a time step, \( x_t \in \mathbb{R}^2 \) is the state and \( u \in \mathbb{R}^2 \) is an input.

Control objective = Drive the state \( x \) from an initial position to a target position while avoiding obstacles

For that, we will use a smart abstraction method presented in [Calbert et al., 2021], called hierarchical abstractions.

First example: Simple problem (sketch of the code)

- First, we define the `system`

```python
function system(
    rectX,
    obstacles,
    rectU,
    Uobstacles,
    tstep,
    measnoise,
    periodic,
    periods,
    T0,
)
    ...
    return SimpleSystem(...)
sys = system(...)
```
First example: Simple problem (sketch of the code)

- We then define the problem

```python
problem = OptimalControlProblem(
    sys,
    initial_set,
    target_set,
    state_cost,
    transition_cost,
    N
)
```
And finally we can define the **smart abstraction method** (an optimizer)

```plaintext
const AB = Dionysos.Optim.Abstraction
optimizer = MOI.instantiate(AB.HierarchicalAbstraction.Optimizer)
AB.HierarchicalAbstraction.set_optimizer!(optimizer,
    concrete_problem,
    hx_global,
    Ugrid,
    compute_reachable_set,
    minimum_transition_cost,
    local_optimizer,
    max_iter,
    max_time;
    option = true,
)
```
First example: Simple problem (sketch of the code)

- We solve the whole problem

```
MOI.optimize!(optimizer)
```

- And we can extract, for example, its abstract system

```
abstract_system = MOI.get(optimizer, MOI.RawOptimizerAttribute("abstract_system"))
```

- Go to Documentation > Examples > Hierarchical-abstraction for the full example!
First example: Simple problem (sketch of the code)

```julia
fig = plot(; aspect_ratio = :equal);

plot!(
    optimizer.hierarchical_problem;
    path = optimizer.optimizer_BB.best_sol,
    heuristic = false,
    fine = true,
)
plot!(UT.DrawTrajectory(x_traj); ms = 0.5)
```
Second example: Path planning

Consider the model of a vehicle in the 2-dimensional plane given by

\[ \dot{x} = f(x, u) = \begin{pmatrix} u_1 \cos(\alpha + x_3) \cos(\alpha^{-1}) \\ u_1 \sin(\alpha + x_3) \cos(\alpha^{-1}) \\ u_1 \tan(u_2) \end{pmatrix}, \]

with \( U = [-1, 1] \times [-1, 1] \), and \( \alpha = \arctan(\tan(u_2)/2) \).

- \((x_1, x_2)\) is the position,
- \(x_3\) is the orientation of the vehicle,
- \(u_1\) is the rear wheel velocity,
- \(u_2\) is the steering angle.

We study the sampled problem with a sampling time \( \tau \).
Second example: Path planning

- Control objective = Drive the vehicle from an initial position to a target position while avoiding obstacles.
- To solve this problem, we use our implementation of an abstraction method described in [Reissig et al., 2017]
- Let’s have an overview of how it looks like using Dionysos.jl...

Second example: Path planning (sketch of the code)

- We first choose the right optimizer

```julia
optimizer = MOI.instantiate(AB.SCOTSAbstraction.Optimizer)
```

- We then set the concrete problem to the optimizer

```julia
MOI.set(
    optimizer,
    MOI.RawOptimizerAttribute("concrete_problem"),
    concrete_problem
)
```

Where `concrete_problem` is defined in `problems/path_planning.jl`. 
Second example: Path planning (sketch of the code)

- Now, we define the state/input grids

```python
MOI.set(optimizer, MOI.RawOptimizerAttribute("state_grid"), state_grid)
MOI.set(optimizer, MOI.RawOptimizerAttribute("input_grid"), input_grid)
```

- We solve the problem

```python
MOI.optimize!(optimizer)
```
Second example: Path planning (sketch of the code)

- Our solver then creates an abstract problem, finds an abstract controller, and refines it to a concrete controller

```python
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")
)
```
Second example: Path planning (sketch of the code)

- Let’s now extract the closed-loop trajectory and plot the result

```python
x_traj, u_traj = ...
# ... Plotting the domain thanks
# ... to implemented Recipes
plot!(UT.DrawTrajectory(x_traj); ms = 0.5)
```

- Go to Documentation > Examples > Path Planning for the full example!
Preliminary benchmarks

Planar switched affine system with univariate control and 2 modes

<table>
<thead>
<tr>
<th></th>
<th>Abstraction [s]</th>
<th>Synthesis [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pessoa</td>
<td>478.7</td>
<td>65.2</td>
</tr>
<tr>
<td>SCOTS</td>
<td>18.1</td>
<td>75.4</td>
</tr>
<tr>
<td>Dionysos</td>
<td>1.02</td>
<td>0.22</td>
</tr>
</tbody>
</table>
Preliminary benchmarks

Nonlinear system with 3 states, 2 inputs, obstacles and target

<table>
<thead>
<tr>
<th></th>
<th>Abstraction [s]</th>
<th>Synthesis [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pessoa</td>
<td>13509</td>
<td>535</td>
</tr>
<tr>
<td>SCOTS</td>
<td>53</td>
<td>210</td>
</tr>
<tr>
<td>Dionysos</td>
<td>6.59</td>
<td>0.57</td>
</tr>
</tbody>
</table>
Conclusions

In summary

- Dionysos implements state-of-the-art and smart abstraction methods to solve control problems for complex dynamical systems.
- It offers a common framework thanks to its implementation based on JuMP and MathOptInterface.
- It is highly modular and benefits from the power/convenience of many other Julia packages.

Future work

- Solving the 27 issues on the github...
- Implementation of an orchestrator.
- Benchmarking Dionysos on our walking robot.
Thank you for listening!

https://github.com/dionysos-dev/Dionysos.jl
About Dionysos


About other toolboxes

