ROCS: A Robustly Complete Control Synthesis Tool for Nonlinear Dynamical Systems[*]

Yinan Li
Department of Applied Mathematics
University of Waterloo
yinan.li@uwaterloo.ca

Jun Liu
Department of Applied Mathematics
University of Waterloo
j.liu@uwaterloo.ca

ABSTRACT

This paper presents ROCS, an algorithmic control synthesis tool for nonlinear dynamical systems. Different from other formal control synthesis tools, it guarantees to generate a control strategy with respect to a robustly realizable specification for nonlinear system. At the core of ROCS is the interval branch-and-bound scheme with a precision control parameter that reflects the robustness of the realizability of the specification. It also supports multiple variable precision control parameters to achieve higher efficiency.

KEYWORDS

Control Synthesis, Nonlinear Systems, Temporal Logic, Interval Methods

1 INTRODUCTION

Control strategies can be algorithmically synthesized for dynamical systems so that they are correct by construction with respect to specifications given in formal languages. This idea is motivated by the principles of formal verification and model checking in computer science and engineering [1, 3].

Linear temporal logic (LTL) is suitable for specifying control objectives, because most of these objectives are desired properties for dynamical systems over time [1]. Invariance and reachability, which are fundamental for dynamical systems, can be described as LTL formulae with temporal operators. Control objectives in some applications, such as robot motion planning and setpoint regulation, are consistent with the Büchi and coBüchi (or reach-and-stay as in [9]) objectives in model checking. A Büchi objective requires that a property of the system can be satisfied infinitely often and a Büchi objective requires that the property can be maintained after it is attained. Control synthesis with GR(1) formulae, which represents a variety of LTL specifications [2, 21], can be solved based on the algorithm for Büchi objectives.

Theories have been developed in recent years to address the problems encountered in formal synthesis when dealing with an infinite-state space and nonlinear dynamics. Most of them follow the framework based on symbolic models or finite abstractions, which are finite-state approximations of the original system [7, 14, 15, 17–19, 22]. A control strategy is synthesized over the finite abstraction and refined to control the original system. Various tools built on abstraction-based methods are available for the purpose of control synthesis. Tools such as Pessoa [9], CoSyMA [16], SCOTS [20], and abstr-refinement [17] are capable of nonlinear system control while TuLiP [21] focuses on more complicated specifications (e.g. GR(1)) but is limited to linear systems. For nonlinear systems without stability assumptions [19, 22], Pessoa, SCOTS and abstr-refinement can be used. LTLMoP [6] is another tool tailored for robot motion planning.

The main purpose of developing ROCS is to perform formal control synthesis for general dynamical systems. Discrete-time models are used in ROCS for computation, and it also works for continuous-time systems by using a fixed sampling time. In this sense, ROCS is similar to SCOTS, Pessoa and abstr-refinement. However, the distinct features of ROCS include:

- Synthesis algorithms are *sound and robustly complete* in the sense that control strategies can be found whenever the given specification is robustly realizable [11, 12]. Such completeness guarantee works for potentially unstable systems while CoSyMA only has such guarantee for incrementally stable systems. Also, different from dReach [10], ROCS is robustly complete in synthesizing control strategies instead of bounded-time reachability verification.
- Specifications and dynamics are considered at the same time in synthesis so that the *discretization precision can be adaptively refined* to generate a non-uniform partition of the

[*] The research was supported in part by the Natural Sciences and Engineering Council (NSERC) of Canada and the Canada Research Chairs (CRC) Program.

HSCC '18, April 11–13, 2018, Porto, Portugal

ACM ISBN 978-1-4503-5642-8/18/04…$15.00
https://doi.org/10.1145/3178126.3178153

system state space. This feature benefits control synthesis in 1) providing an automatic and efficient discretization precision refinement scheme, and 2) avoiding using unnecessarily high precision uniformly over the entire state space in order to yield a feasible control strategy.

- It supports relative and variable precisions with flexible parameter setting. The user only needs to specify the highest relative precision for solving a control synthesis problem. The actual discretization will be adjusted to achieve the balance between accuracy and efficiency.

2 TOOL FUNCTION

The control system considered is in the form of a tuple

$$\Sigma :< \mathcal{T}, \mathcal{X}, \mathcal{U}, \mathcal{D}, f, AP, L >$$

- $\mathcal{T} = \mathbb{Z}_{\geq 0}$ is a set of time instances.
- $\mathcal{X} \subseteq \mathbb{R}^n$ is a non-empty set of states.
- $\mathcal{U} \subseteq \mathbb{R}^m$ is a non-empty set of control inputs.
- $\mathcal{D} \subseteq \mathbb{R}^n$ is a set of bounded perturbations given by
$$\mathcal{D} := \{d \in \mathbb{R}^n \mid |d|_\infty \leq \delta,\ \delta \geq 0\}\,.$$
- $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a continuous function. Evolution of Σ is determined by
$$x_{t+1} = f_{u_t}(x_t) + d_t, \quad t \in \mathcal{T}, \tag{1}$$
where $x_t \in \mathcal{X}$, $u_t \in \mathcal{U}$ and $d_t \in \mathcal{D}$.
- AP is a set of atomic propositions, which are true or false statements regarding some properties.
- $L : \mathcal{X} \to 2^{AP}$ is a *labeling function*, which associates properties to every state in the state space $\mathcal{X}$.

A control system Σ is said to be *deterministic* if $\delta = 0$ and *nondeterministic* otherwise. Specifically, we refer to a deterministic system and a nondeterministic system by Σ^* and Σ^δ, respectively.

A sequence of control inputs $\mathbf{u} = \{u_i\}_{i=0}^\infty$, where $u_i \in \mathcal{U}$, is called a *control signal*. Similarly, we denote by $\mathbf{d} = \{d_i\}_{i=0}^\infty$ a sequence of disturbances. A *solution* of control system Σ is denoted by an infinite sequence of states $\mathbf{x} = \{x_i\}_{i=0}^\infty$, which is generated by an initial condition $x_0 \in \mathcal{X}$, a control signal $\mathbf{u}$ and a disturbance $\mathbf{d}$ according to (1). The *trace* of a solution $\mathbf{x}$ is defined by $\mathrm{Trace}(\mathbf{x}) = \{L(x_i)\}_{i=0}^\infty$ and is used to interpret an LTL formula over a control system.

A*memoryless control strategy* of system Σ is a function that maps a system state to a subset of control inputs:

$$\kappa : \mathcal{X} \to 2^{\mathcal{U}}. \tag{2}$$

A control signal $\mathbf{u} = \{u_k\}_{k=0}^\infty$ is said to *conform to* a control strategy c, if $u_t \in \kappa(x_t)$, $\forall t \geq 0$, where $\{x_t\}_{t=0}^\infty$ is the resulting solution of Σ.

Definition 2.1. An LTL formula φ is said to be *realizable* for system Σ if there exists an initial condition $x_0 \in \mathcal{X}$ and a control strategy κ such that, for any control signal that conforms to κ, the resulting trace of system Σ is guaranteed to satisfy φ. Specifically, if φ is realizable for Σ^δ, i.e., the system with perturbations of bound $\delta > 0$, we say that φ is δ*-robustly realizable* for Σ^*, where δ is the *robustness margin.*

The set of all initial conditions from which φ is can be realized for Σ is called the *winning set* of φ, denoted by $\mathrm{Win}_\Sigma(\varphi)$.

ROCS addresses the following control synthesis problem: given an LTL specification φ for system Σ^*,

(i) determine whether φ is robustly realizable for Σ^*;
(ii) synthesize a feedback control strategy such that the closed-loop system satisfies φ if possible.

In a nutshell, ROCS takes a control system Σ and an LTL specification as inputs and returns a control strategy exportable to Matlab for simulation and display. It currently supports control synthesis for system Σ with invariance, reachability, Büchi, and coBüchi objectives, which can be shown to be realizable using memoryless control strategies.

3 DESIGN AND TECHNICAL DETAILS

Figure 1 shows the current architecture of ROCS, which is composed of three modules. The module "control problem" mounts system dynamics and specifications provided by the user. To solve the control problem, the module "solver" performs formal synthesis algorithms and will generate a winning set together with a control strategy that realizes the given specification. The information of the winning set, control strategy, as well as intermediate results from iterations, are managed by the module "interval paver", which is implemented using a binary tree structure.

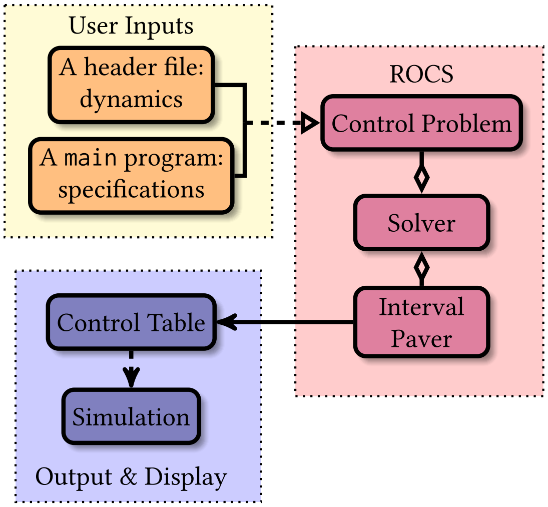

Figure 1: The architecture of ROCS.

In the following, we present technical details on the implementation of these three modules.

3.1 Control strategy defined on intervals

Definition 3.1. Given $\Omega \subseteq \mathbb{R}^n$, a finite collection of sets $\mathcal{P} = \{P_1, P_2, \cdots, P_N\}$ is said to be a *partition* of Ω if (i) $P_i \subseteq \Omega$; (ii) $\text{int}(P_i) \cap \text{int}(P_j) = \varnothing$, where $\text{int}(P_i)$ denotes the interior of P_i; (iii) $\Omega \subseteq \bigcup_{i=1}^{N} P_i$, for all $i \in \{1, \cdots, N\}$. Each element P_i of the partition $\mathcal{P}$ is called a *cell.*

The control strategy generated by ROCS is partition-based. Let $\mathcal{Y} = \{Y_1, Y_2, \cdots, Y_N\}$ be the resulting partition of $\mathcal{X}$ and $C = \{C_1, C_2, \cdots, C_N\}$ be the set of control inputs such that the given LTL specification can be realized. For each $i \in \{1, 2, \cdots, N\}$, C_i corresponds to the cell Y_i and will be determined by the solver. Then the control strategy returned by ROCS is given by $\kappa(x) = \bigcup_{i \in N} \psi_{Y_i}(x)$, where $x \in \mathcal{X}$, and for $i = 1, 2, \ldots, N$, $\psi_{Y_i}(x) = C_i$ if $x \in Y_i$, and $\psi_{Y_i}(x) = \varnothing$ if $x \notin Y_i$.

Theoretically, cells may be of any shape and size (e.g. triangles and intervals). In ROCS, each cell Y_i is represented by an interval vector (interval for short) in $\mathbb{R}^n$, which is denoted by $[x] := [x_1] \times \cdots \times [x_n] \subseteq \mathbb{R}^n$, where $[x_i] = [\underline{x}_i, \overline{x}_i] \subseteq \mathbb{R}$ for $i = 1, \cdots, n$, $\underline{x}_i$ represents the infimum of $[x_i]$, and $\overline{x}_i$ the supremum. We also write $[x] = [\underline{x}, \overline{x}]$, where $\underline{x} = [\underline{x}_1, \cdots, \underline{x}_n]^T$ and $\overline{x} = [\overline{x}_1, \cdots, \overline{x}_n]^T$. The width of the interval $[x]$ is defined as $w([x]) := \max_{1 \le i \le n}\{\overline{x_i} - \underline{x_i}\}$. The set of all intervals in $\mathbb{R}^n$ is denoted by $\mathbb{IR}^n$. The advantage of using intervals is that they are simple and easy to manipulate automatically.

Uniform grids implemented in [9, 16, 20] for constructing abstractions can be considered as intervals with uniform widths. Instead of using uniform intervals, in ROCS, the system state space is adaptively partitioned according to the given specification and system dynamics. Further details on such a partitioning scheme will be illustrated in section 3.3.

3.2 Control synthesis algorithms

Similar to model checking problems, the realizability of φ for control system Σ is determined by examining $\text{Win}_\Sigma(\varphi)$. If $\text{Win}_\Sigma(\varphi) \neq \emptyset$, then φ can be realized for system Σ.

Therefore, the task of the module "solver" is to determine the winning set of a given specification. For different LTL specifications, different algorithms have to be applied. Nevertheless, all of these algorithms involve repetitive evaluation of the *predecessor* of a set $X \subseteq \mathcal{X}$ for system Σ with respect to system dynamics (1), which is a set of states defined by

$$\begin{aligned}\text{Pre}(X) := \{x \in \mathcal{X} \mid \exists u \in \mathcal{U}, \forall d_t \in \mathcal{D} \text{ s.t.} \\ f_u(x) + d_t \in X\}.\end{aligned} \tag{3}$$

In the context where deterministic system Σ^* and nondeterministic systems Σ^δ have to be distinguished, we denote by $\text{Pre}^*(X)$ the predecessor of X for Σ^* and $\text{Pre}^\delta(X)$ that for Σ^δ.

For the invariance specification $\varphi_i = \Box p(A)$, where $A \subseteq \mathcal{X}$ has to be controlled invariant and $p(A)$ is true if and only if the system state is in A, the winning set of φ_i for system Σ is the maximal controlled invariant set inside A [11]. Written in the form of μ-calculus [5], which is a formal language embedding fixed points,

$$\text{Win}_\Sigma(\varphi_i) = \nu X.(A \cap \text{Pre}(X)), \tag{4}$$

where $\nu X.T(X)$ denotes the greatest fixed point of the mapping $T(X) := A \cap \text{Pre}(X), X \subseteq \mathcal{X}$.

For the reachability specification $\varphi_r = \Diamond p(A)$, where $A \subseteq \mathcal{X}$ denotes the region to be reached in some future time, the μ-calculus formula expressing the winning set is

$$\text{Win}_\Sigma(\varphi_r) = \mu X.(A \cup \text{Pre}(X)), \tag{5}$$

where $\mu X.T(X)$ is the least fixed point of $T(X) := A \cup \text{Pre}(X)$.

The winning sets of the Büchi specification $\varphi_b = \Box\Diamond p(A)$ and the coBüchi specification $\varphi_c = \Diamond\Box p(A)$ can be determined by the alternating fixed-point operations [4]:

$$\text{Win}_\Sigma(\varphi_b) = \nu Y \mu X.[\text{Pre}(X) \cup (A \cap \text{Pre}(Y))], \tag{6}$$

$$\text{Win}_\Sigma(\varphi_c) = \mu Y \nu X.[\text{Pre}(Y) \cup (A \cap \text{Pre}(X))]. \tag{7}$$

Fixed-point algorithms for computing (4) to (7) can be implemented easily on finite-state systems (e.g., game structures). For infinite-state system Σ, computing (3) is difficult especially for nonlinear dynamics. Using the interval approximation of (3), which is proposed in [11], ROCS currently solves the aforementioned four types of LTL specifications based on the fixed-point characterizations (4) to (7).

3.3 Control strategy generation via interval branch-and-bound scheme

The kernel of ROCS is the approximation of predecessors using interval branch-and-bound scheme, which is shown in Algorithm 1 and conducted by the module "interval paver". It naturally yields a non-uniform partition of the state space according to the dynamics and the given specification.

Given $X, Y \subseteq \mathcal{X}$, Algorithm 1 approximates the predecessor of Y that resides in set X, i.e., $X \cap \text{Pre}(Y)$. The set approximation precision is controlled by a parameter $\varepsilon > 0$. When an interval $[x]$ with $w([x]) > \epsilon$ cannot be determined to be part of $X \cap \text{Pre}(Y)$ or not, it is bisected to $L([x])$ and $R([x])$, which are given by $L[x] = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_j, (\underline{x}_j + \overline{x}_j)/2] \times \cdots \times [\underline{x}_n, \overline{x}_n]$ and $R[x] = [\underline{x}_1, \overline{x}_1] \times \cdots \times [(\underline{x}_j + \overline{x}_j)/2, \overline{x}_j] \times \cdots \times [\underline{x}_n, \overline{x}_n]$, respectively, where j is the bisected dimension.

The interval function $[f] : \mathbb{IR}^n \to \mathbb{IR}^m$ is called a *convergent inclusion function* of f if (i) $f([x]) \subseteq [f]([x])$ for all $[x] \in \mathbb{IR}^n$ and (ii) $\lim_{w([x]) \to 0} w([f]([x])) = 0$ [8].

Such a convergent inclusion function is not unique for a given function f defined on $\mathbb{R}^n$. The natural inclusion function and mean-value inclusion function are usually used.

In addition to adaptive partitioning with respect to the dynamics f and the input set X, Y, Algorithm 1 records *valid control values* for each cell, under which the cell are mapped

Algorithm 1 Predecessor of Y bounded by X

```
1:  procedure CPred($[f_u]_{u \in \mathcal{U}}, X, Y, \varepsilon$)
2:      $K \leftarrow \emptyset, \underline{X} \leftarrow \emptyset, \Delta X \leftarrow \emptyset, X_c \leftarrow \emptyset$
3:      $List \leftarrow X$
4:      while $List \neq \emptyset$ do
5:          $[x] \leftarrow List.first$
6:          if $[f_u]([x]) \cap Y = \emptyset$ for all $u \in \mathcal{U}$ then
7:              $X_c \leftarrow X_c \cup [x]$
8:          else if $[f_u]([x]) \subseteq Y$ for some $u \in \mathcal{U}$ then
9:              $\underline{X} \leftarrow \underline{X} \cup [x]$
10:             $K \leftarrow K \cup ([x], u)$
11:         else
12:             if $w([x]) < \varepsilon$ then
13:                 $\Delta X \leftarrow \Delta X \cup [x]$
14:             else
15:                 $\{L[x], R[x]\} = Bisect([x])$
16:                 $List.add(\{L[x], R[x]\})$
17:             end if
18:         end if
19:     end while
        return $K, \underline{X}, \Delta X, X_c$
20: end procedure
```

into Y completely in one step of time. The returned set K containing pairs of intervals and their corresponding valid control values. Fixed-point algorithms (4) to (7) rely on CPred for the computation of each iteration and will terminate in a finite number of steps, returning both the partition $\mathcal{Y}$ and corresponding set of valid control inputs C. The outputs of CPred after every iteration are kept in stacks and used as the inputs of the next call of CPred.

ROCS supports interval representation of the input sets X and Y, and also a very general representation of Y, e.g. $Y := \{y \in \mathbb{R}^n \mid g(y) \leq 0\}$, where $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$. In this case, the condition $[f_u]([x]) \cap Y = \emptyset$ and $[f_u]([x]) \subseteq Y$ in line 6 and 8 are replaced by $[g \circ f_u]([x]) \subseteq [0, \infty]^m$ and $[g \circ f_u]([x]) \subseteq [-\infty, 0]^m$, respectively.

An important fact about ROCS is that the user controlled predecessor approximation precision ε also reflects the robustness of the realizability of the specification. Theoretical results can be found in [11, 12].

4 TOOL USAGE

The tool is implemented in C++ and currently released as a bundle of APIs accessing the core synthesis algorithms. The source code and examples can be downloaded from https://git.uwaterloo.ca/hybrid-systems-lab/rocs.

To solve a control synthesis problem using ROCS, the user needs to provide:

- an interval inclusion function of the flow map of the system to be controlled, and
- a main program that defines the control problem and executes control synthesis.

To be compatible with different interval inclusion functions provided by the user, we use an abstract class VFunctor in ROCS, which is declared in the file vectorfield.h and associated with the CntlProb class. Thus, the user specified inclusion function should be defined as a derived class of VFunctor. We call it an *inclusion functor*. For a continuous-time system, the inclusion function can be provided as an over-approximation of the discrete-time flow map. To write such an inclusion functor, the user may install external packages Armadillo and Boost[1] and refer to vectorfield.h or the examples shipped with the source code.

To manage a control synthesis process, the user has to write a main function for each control problem. Figure 2 is a sample main function for invariance control synthesis of a DC-DC converter, which shows the synthesis workflow.

```
  int main()
{   /* state and input space */
    double xlb[]={-2,0.70};
    double xub[]={2,1.50};
    input_type U{{1},{2}};  //two modes
    /* specification */
    double glb[]={1.15,1.09};  //target area
    double gub[]={1.55,1.17};
    /* functor of dynamics */
    DCDC *ptrDC=new DCDC(U,TS); //TS:sampling time
    /* define a control problem */
    CntlProb dcdcInv("dcdc",XD,UD,xlb,xub,ptrDC);
    /* create a solver and solve the problem */
    CSolver *solver=new CSolver(&dcdcInv);
    solver->init(GOAL,glb,gub);
    solver->invariance_control(0.001,RELMAXG);
    solver->print_controller_info();
    /* save the control strategy to file */
    dcdcInv.write2mat_settings("dcdc_spec.mat");
    solver->write2mat_controller("dcdc_cbox.mat");
    delete solver;   delete ptrDC;
    return 1;
}
```

Figure 2: A sample main function for the invariance control synthesis of a DC-DC converter. A partition precision of 0.001 and the relative bisection type RELMAXG are used when calling invariance_control, which is a member function of CSolver.

First, the state and input space are specified by their lower and upper bounds. A particular input_type can be used when the system has discrete modes. Next, after loading the

[1] http://arma.sourceforge.net, http://www.boost.org

customized inclusion functor, a control problem (a `CntlProb` object) will be instantiated as specified by the user. Then a solver (a `CSolver` object) is created to attach to the problem and gradually refines the the partition (an `interval_paver` object) of the system state space under the corresponding synthesis algorithm. Finally when the iteration terminates, in order to test and visualize the control performance, the user can write the entire case information, including system and specification setups, and control strategy to `.mat` files. Utility functions for Matlab display are provided under the `matlab` folder of the ROCS package.

To actually perform control synthesis, the user would have to choose an algorithm provided by the `CSolver` class depending on the control objective. For example, in Figure 2, the `invariance_control` algorithm is used. Three other available algorithms include `reachability_control`, `buchi`, and `cobuchi`.

These algorithms take in the following three types of arguments:

- the precision control parameter,
- the bisection type, and
- a boolean indicating variable or fixed precision.

The precision control parameter determines the precision of the resulting partition and is related to the robustness margin of the specification (see [11, 12]). The bisection type indicates whether to subdivide an interval along the dimension of the greatest absolute or relative width to the state space/target area. For specifications related to reachability, it is usually more efficient to use a variable precision (by setting the boolean argument to be `true`). For detailed descriptions and usage of the parameters of each algorithm, the user may refer to the documentation of the `CSolver` class.

In the package, we provide complete sets of examples, including interval inclusion functions, main program files, and files for Matlab simulation, to show how to use ROCS for control synthesis.

5 DEMONSTRATION

5.1 DC-DC converter

In the fist example, we use ROCS to solve an invariance control problem for DC-DC boost converter, which has been used for testing in [16, 20]. This is a 2-dimensional 2-mode switched system whose model and the corresponding parameters can be found in [7]. The discrete-time model is obtained with sampling time $t_s = 0.5$. The Lipschitz constant is $L = 1.0737$.

The invariance specification is given by $\varphi_i = \Box p(\Omega)$, where $\Omega = [1.15, 1.55] \times [1.09, 1.17]$. We use the natural inclusion function, an absolute precision control parameter 0.001, and the bisection type `RELMAXG`. Figure 3 shows a satisfactory controlled system solution from the initial condition $(1.2, 1.12)$. We compare the run time of ROCS with those of Pessoa, CoSyMa, and SCOTS as reported in [20] in Table 1, and it shows that ROCS performs well in terms of efficiency. We denote by "t_{abst}" and "t_{syn}" the time spent on computing abstractions and control synthesis, respectively.

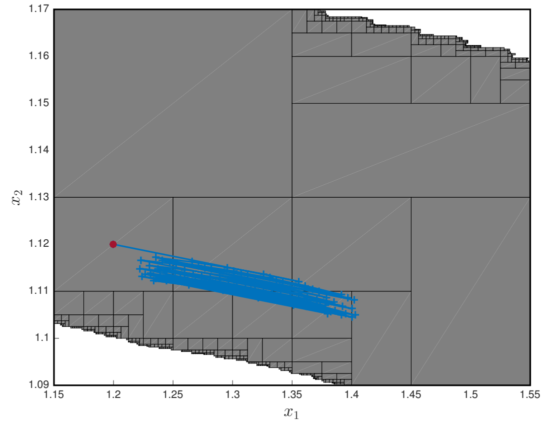

Figure 3: Controlled trajectory from the initial condition $(1.2, 1.12)$**.**

Table 1: Comparison of run times

	CPU [GHz]	t_{abst} [s]	t_{syn} [s]
Pessoa	i7 3.5	478.7	65.2
CoSyMA	N/A	N/A	8.32
SCOTS	i7 3.5	18.1	74.5
ROCS	i5 2.4	0	0.36

To see how robustly φ_i can be realized for the converter, we vary the values of the precision control parameters. When the value is greater 0.0063, φ_i becomes unrealizable. Then φ_i is not realizable for any perturbation with amplitude greater than 0.0068.

5.2 Inverted pendulum

In the second example, we aim to control a pendulum on a cart [11] to the upright position with an angle stabilization precision ±0.05 rad. The state variables are the angle of the pendulum θ and the angle change rate $\dot{\theta}$. The control input is a force applied to the cart. Written as an LTL formula, the specification is $\varphi_c = \Diamond\Box p(G)$, where $G := [-0.05, 0.05] \times [-0.01, 0.01]$. This system is neither globally asymptotically stable nor incrementally asymptotically stable around the upright position.

We consider the state space $\mathcal{X} := [-2, 2] \times [-3.2, 3.2]$ and discrete-time flow map with the sampling time $\tau_s = 0.01$s. The control input u is chosen from the finite set $\mathcal{U} = 0.05\,\{-10, -9, \cdots, 9, 10\}$. Limited by the size of G, we use a precision $\varepsilon = 0.001$ for the CPred of the inner greatest fixed-point iteration in (7). Since the state space X is nearly 40 times the size of G, we use relative precision in the outer least fixed-point iteration. The inner loop precision reflects the bound of the perturbation that can be tolerated by the resulting switching strategy. A local growth bound [19] is used as the inclusion function. Figure 4 shows a satisfying control result from the initial condition $(1, 1)$.

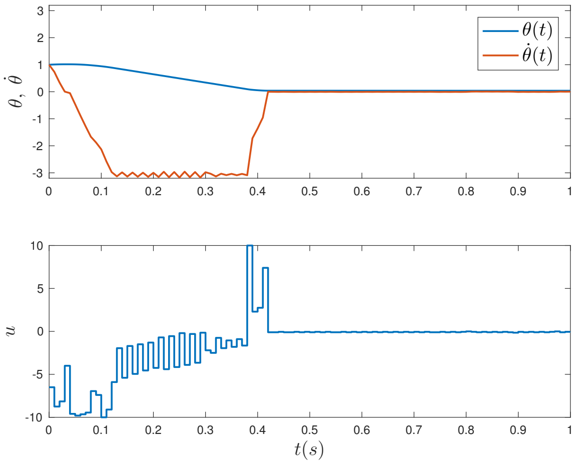

Figure 4: Closed-loop simulation with the initial condition $(\theta_0, \dot{\theta}_0) = (1, 1)$ for inverted pendulum.

To draw a comparison with abstraction-based synthesis tools, we tried to solve the same example using SCOTS. To use SCOTS, we need to apply a rather small grid size, e.g. 0.001, to X because of the small size of G. As a result, X will be discretized to 2.56×10^7 cells. Computation of the abstraction lasts for more than 12 hours without returning any result. In contrast, ROCS generates a winning set covering most of the state space in around 400 seconds with 26340 partitions.

6 DISCUSSIONS AND FUTURE WORK

We introduced ROCS in the current paper, which is a tool for control synthesis for general dynamical systems. The main feature that differentiates ROCS from other available tools for nonlinear system control synthesis is that it guarantees to generate feasible control strategies if the given specification is robustly realizable. While a sound and robustly complete abstraction usually requires a very small grid size [13], the use of interval branch-and-bound scheme with variable precision control yields a non-uniform partition of a much smaller size, adaptively tailored for each specification.

Extensions of ROCS to support more general LTL specifications will be added in the future. Future development of the tool will also improve its scalability by exploring separable structures in higher dimensional dynamical systems.

REFERENCES

[1] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT press.

[2] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'Ar. 2012. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.* 78, 3 (2012), 911–938.

[3] Edmund Clarke, Orna Grumberg, and Doron A. Peled. 1999. *Model Checking*. MIT Press.

[4] L. de Alfaro, T.A. Henzinger, and R. Majumdar. 2001. From verification to control: dynamic programs for omega-regular objectives. In *Proc. of Annu. IEEE Symp. Log. Comput. Sci.* 279–290.

[5] E. Allen Emerson and Chin-Laung Lei. 1986. Efficient model checking in fragments of the propositional mu-calculus. In *First Annu. IEEE Symp. Log. Comput. Sci.* 267–278.

[6] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. 2010. LTLMoP: Experimenting with language, temporal logic and robot control. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst.* IEEE, 1988–1993.

[7] Antoine Girard, Giordano Pola, and Paulo Tabuada. 2010. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Trans. Automat. Contr.* 55, 1 (2010), 116–126.

[8] Luc Jaulin. 2001. *Applied Interval Analysis*. Springer Science & Business Media.

[9] Manuel Mazo Jr., Anna Davitian, and Paulo Tabuada. 2010. PESSOA: a tool for embedded controller synthesis. In *Proc. of Computer-Aided Verification (CAV)*. 566–569.

[10] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. 2015. dReach: δ-Reachability analysis for hybrid systems. In *Proc. of TACAS*. 200–205.

[11] Yinan Li and Jun Liu. 2017. Invariance Control Synthesis for Switched Nonlinear Systems: An Interval Analysis Approach. *IEEE Trans. Automat. Contr.* (2017). https://doi.org/10.1109/TAC.2017.2760106

[12] Yinan Li and Jun Liu. to appear. Robustly Complete Reach-and-Stay Control Synthesis for Switched Systems via Interval Analysis. In *Proc. of the 2018 American Control Conf. (ACC)*.

[13] Jun Liu. 2017. Robust abstractions for control synthesis: completeness via robustness for linear-time properties. In *Proc. of HSCC*.

[14] Jun Liu and Necmiye Ozay. 2016. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Analysis: Hybrid Systems* 22 (2016), 1–15.

[15] Jun Liu, Necmiye Ozay, Ufuk Topcu, and Richard M. Murray. 2013. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. Automat. Contr.* 58, 7 (2013), 1771–1785.

[16] Sebti Mouelhi, Antoine Girard, and Gregor Gössler. 2013. CoSyMA: a tool for controller synthesis using multi-scale abstractions. In *Proc. of HSCC*. 83–88.

[17] Petter Nilsson, Necmiye Ozay, and Jun Liu. 2017. Augmented finite transition systems as abstractions for control synthesis. *Discret. Event Dyn. Syst.* 27, 2 (2017), 301–340.

[18] Giordano Pola, Antoine Girard, and Paulo Tabuada. 2008. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica* 44, 10 (2008), 2508–2516.

[19] Gunther Reissig, Alexander Weber, and Matthias Rungger. 2017. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Trans. Automat. Contr.* 62, 4 (2017), 1781–1796.

[20] Matthias Rungger and Majid Zamani. 2016. SCOTS: a tool for the synthesis of symbolic controllers. In *Proc. of HSCC*. 99–104.

[21] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. 2011. TuLiP: a software toolbox for receding horizon temporal logic planning. In *Proc. of HSCC*. 313.

[22] Majid Zamani, Giordano Pola, Manuel Mazo Jr., and Paulo Tabuada. 2012. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Trans. Automat. Contr.* 57, 7 (2012), 1804–1809.