
# SCOTS: A Tool for the Synthesis of Symbolic Controllers

Matthias Rungger
Hybrid Control Systems Group
Technical University of Munich
matthias.rungger@tum.de

Majid Zamani
Hybrid Control Systems Group
Technical University of Munich
zamani@tum.de

## ABSTRACT

We introduce SCOTS a software tool for the automatic controller synthesis for nonlinear control systems based on symbolic models, also known as discrete abstractions. The tool accepts a differential equation as the description of a nonlinear control system. It uses a Lipschitz type estimate on the right-hand-side of the differential equation together with a number of discretization parameters to compute a symbolic model that is related with the original control system via a feedback refinement relation. The tool supports the computation of minimal and maximal fixed points and thus natively provides algorithms to synthesize controllers with respect to invariance and reachability specifications. The atomic propositions, which are used to formulate the specifications, are allowed to be defined in terms of finite unions and intersections of polytopes as well as ellipsoids. While the main computations are done in C++, the tool contains a Matlab interface to simulate the closed loop system and to visualize the abstract state space together with the atomic propositions. We illustrate the performance of the tool with two examples from the literature. The tool and all conducted experiments are available at www.hcs.ei.tum.de

## Keywords

Symbolic Models; Discrete Abstractions; Feedback Refinement Relations; C++/Matlab Toolbox

## 1. INTRODUCTION

In recent years, controller synthesis techniques based on so-called *symbolic models* or *discrete abstractions* have received considerable attention within the control systems community, see e.g. [1–13]. Following those methods, it is possible to synthesize correct-by-construction controllers for general nonlinear systems to enforce complex specifications formulated for example in linear temporal logic (LTL).

There exist numerous theories [1–13] that account for a great variety of different control system dynamics as well as specifications. The dynamics range from simple double integrators [5], over linear [1, 3, 8, 11] and piecewise affine systems [10] to nonlinear [4, 6, 7, 9, 12] and stochastic control systems [2, 13]. The specifications range from reach-avoid specifications [2, 4, 6, 7, 9, 13] and safety [4, 7, 11] specification, over reactivity fragments of LTL [5, 8, 12] to full LTL [1, 3, 10]. Additionally, there exist various software tools such as Pessoa [14], CoSyMA [15], LTLMoP [16], and TuLiP [17] were some of the previously mentioned theories are implemented.

In this paper, we introduce SCOTS, yet another software tool for the synthesis of symbolic controllers, i.e. controllers based on symbolic models. Similar to Pessoa and CoSyMA the tool supports the computation of abstractions of nonlinear control systems. LTLMoP and TuLiP are more restrictive in that respect and accept merely simple integrator dynamics and piecewise affine control systems, respectively. Moreover, the tool provides algorithms for the computation of minimal and maximal fixed points and thus, again similar to Pessoa and CoSyMA, natively supports the controller design to enforce invariance and reachability (time-bounded reachability for CoSyMA) specifications. More complex specifications like GR(1) [18], which are supported by LTLMoP and TuLiP, are currently not natively available in SCOTS.

The differences between Pessoa, CoSyMA, and SCOTS become apparent in terms of the *type* of symbolic model which is used to solve the synthesis problem. CoSyMA requires the original system to be incrementally stable [19] and computes symbolic models that are *approximately bisimilar* to the original system. Pessoa, additionally to approximately bisimilar symbolic models, supports the computation of *approximately alternatingly similar* symbolic models [4]. Approximate alternating simulation relations, compared to approximate bisimulation relations, are a one-sided notion of system relations and the symbolic model can be constructed without stability assumptions [9]. The symbolic models created by SCOTS are based on *feedback refinement relations*, a novel notion of system relation that was recently introduced in [20]. Although the computation of symbolic models based on approximate alternating simulation relations and feedback refinement relations are similar cf. [20, Chapter VIII] and [9], the controller refinement procedure differs. Specifically, the controller based on feedback refinement relations requires quantized state information only, as opposed to exact state information. Moreover, the controller does not require to include the abstraction as a building block [20].

This work was supported in part by the German Research Foundation (DFG) grant ZA 873/1-1.



*HSCC'16, April 12 - 14, 2016, Vienna, Austria*



ISBN 978-1-4503-3955-1/16/04...$15.00

DOI: http://dx.doi.org/10.1145/2883817.2883834

Both properties are essential for a practical implementation of the controller.

From an implementation point of view SCOTS is close to Pessoa and CoSyMA. All three tools use boolean functions to represent the atomic propositions as well as the transition relations of the symbolic models and employ binary decision diagrams [21] as underlying data structures to store and manipulate the boolean functions. Specifically, all three tools use the CUDD binary decision diagram library [22]. Even though binary decision diagrams provide a memory efficient way to represent symbolic models, they result in a considerable computation-time overhead when it comes to the construction of the boolean functions. A similar observation is reported in [15] and CoSyMA offers hash tables as an alternative data structure.

# 2. SYMBOLIC CONTROLLER SYNTHESIS

We provide a short introduction to the solution of control problems based on symbolic models as developed in [20]. In general the notation should be self-explanatory. However, in case of any ambiguity, the reader is referred to [20, Sec. II].

In our framework, a system is defined as *simple system*, which is a triple $S = (X, U, F)$, where the *state alphabet* $X$ and *input alphabet* $U$ are non-empty sets and the *transition function* $F : X \times U \rightrightarrows X$ is a set-valued map [23]. The behavior $\mathcal{B}(S)$ consists of the set of the signals $(x, u)$, with *state signal* $x : [0; T[ \to X$ and *input signal* $u : [0; T[ \to U$, that satisfy the transition function $F$ for all $t \in [0; T-1[$. If $T$ is finite then $F(x(T-1), u(T-1)) = \varnothing$ must hold. Any subset $\Sigma \subseteq (X \times U)^{[0;T[}$, $T \in \mathbb{N} \cup \{\infty\}$ constitutes a *specification* for $S$. Subsequently, we focus mostly on specifications on state signals, in which case we omit the input signals and consider $\mathcal{B}(S)$ and $\Sigma$ as subsets of $X^{[0;T[}$, $T \in \mathbb{N} \cup \{\infty\}$. A system $S$ and a specification $\Sigma$ (for $S$) constitute a *control problem* $(S, \Sigma)$. A *controller* $C$, which is itself a system that is *feedback composable* with $S$, *solves* the control problem $(S, \Sigma)$ if the closed loop behavior satisfies $\mathcal{B}(C \times S) \subseteq \Sigma$. We use $U_S(x) = \{u \in U \mid F(x, u) \neq \varnothing\}$ to denote the set of *admissible inputs*. The interested reader is referred to [20] for the precise definitions of the various terms and objects.

A symbolic controller synthesis scheme as it is implemented in SCOTS proceeds, roughly speaking, in three steps. First, given a control problem $(S_1, \Sigma_1)$, a finite simple system $S_2$ as a substitute of $S_1$, together with an *abstract specification* $\Sigma_2$ is computed. In this context $S_1$ and $S_2$ are often referred to as *plant* and *symbolic model*, respectively. In the second step, a *controller* $C_2$, i.e., system that is feedback composable with $S_2$, which solves the control problem $(S_2, \Sigma_2)$ is computed. Provided that the synthesis process of $C_2$ is successful, the controller $C_2$ is *refined* to a controller $C_1$ that solves the original problem $(S_1, \Sigma_1)$ in the third step.

The correctness of this approach is guaranteed by relating the plant $S_1 = (X_1, U_1, F_1)$ with its symbolic model $S_2 = (X_2, U_2, F_2)$ via a *feedback refinement relation* $Q \subseteq X_1 \times X_2$. Here it is assumed that $U_2 \subseteq U_1$ and every pair $(x_1, x_2) \in Q$ satisfies two conditions (see [20, Def. V.2] for details):

1. $U_{S_2}(x_2) \subseteq U_{S_1}(x_1)$ and
2. $u \in U_{S_2}(x_2) \implies Q(F_1(x_1, u)) \subseteq F_2(x_2, u)$ holds.

A distinct feature of the utilization of feedback refinement relations in a symbolic controller synthesis scheme is the particular simple refinement step, in which the refined controller $C_1$ for the plant $S_1$ is naturally obtained from the abstract controller $C_2$ by using the relation $Q$ as *quantizer* to map the plant states $x_1$ to abstract states $x_2 \in Q(x_1)$. The refinement scheme is illustrated in Fig. 1.

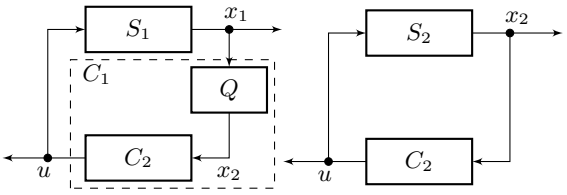


**Figure 1: Refined closed loop (left) and abstract closed loop (right).**

## 2.1 Computation of symbolic models

SCOTS supports the computation of symbolic models of the sampled behavior of perturbed control systems of the form

$$\dot{\xi}(t) \in f(\xi(t), u) + [\![-w, w]\!] \tag{1}$$

where $f$ is given by $f : \mathbb{R}^n \times U \to \mathbb{R}^n$ and $U \subseteq \mathbb{R}^m$. We assume that the set $U$ is non-empty and $f(\cdot, u)$ is continuously differentiable for every $u \in U$. The vector $w = [w_1, \ldots, w_n] \in \mathbb{R}^n_+$ is a perturbation bound and $[\![-w, w]\!]$ denotes the hyper-interval $[-w_1, w_1] \times \ldots \times [-w_n, w_n]$. Additionally to the perturbations on the right-hand-side of (1), we consider measurement errors, with bound $z \in \mathbb{R}^n_+$, which we model by a set-valued map $P \colon \mathbb{R}^n \rightrightarrows \mathbb{R}^n$,

$$P(x) = x + [\![-z, z]\!]. \tag{2}$$

Let $\tau > 0$ be the sampling time. Formally, we represent the $\tau$-sampled behavior of the control system (1) as simple system $S_1 = (X_1, U_1, F_1)$, with $X_1 = \mathbb{R}^n$, $U_1 = U$ and $F_1$ is implicitly defined by $x' \in F_1(x, u)$ iff there exists a solution $\xi$ of (1) under input $u \in U$ so that $\xi(0) = x$ and $x' = \xi(\tau)$.

The construction of a symbolic model of $S_1$ is based on the over-approximation of attainable sets. To this end, we use the notion of a growth bound introduced in [20]. A *growth bound* of (1) is a function $\beta \colon \mathbb{R}^n_+ \times U' \to \mathbb{R}^n_+$, which is defined with respect to a sampling time $\tau > 0$, a set $K \subseteq \mathbb{R}^n$ and a set $U' \subseteq U$. Basically, it provides an upper bound on the deviation of solutions $\xi$ of (1) from *nominal solutions*[1] $\varphi$ of (1), i.e., for every solution $\xi$ of (1) on $[0, \tau]$ with input $u \in U'$ and $\xi(0), p \in K$, we have

$$|\xi(\tau) - \varphi(\tau, p, u)| \leq \beta(|\xi(0) - p|, u). \tag{3}$$

Here, $|x|$ for $x \in \mathbb{R}^n$, denotes the component-wise absolute value. A growth bound can be obtained essentially by bounding the Jacobian of $f$. Let $L \colon U' \to \mathbb{R}^{n \times n}$ satisfy

$$L_{i,j}(u) \geq \begin{cases} D_j f_i(x, u) & \text{if } i = j, \\ |D_j f_i(x, u)| & \text{otherwise} \end{cases} \tag{4}$$

for all $x \in K' \subseteq \mathbb{R}^n$ and $u \in U' \subseteq U$. Then

$$\beta(r, u) = e^{L(u)\tau} r + \int_0^\tau e^{L(u)s} w \, \mathrm{d}s,$$

is a growth bound on $[0, \tau]$, $K$, $U'$ associated with (1). The domain $K'$ on which (4) needs to hold, is assumed to be convex and contain any solution $\xi$ on $[0, \tau]$ of (1) with $u \in U'$

[1] A nominal solution $\varphi(\cdot, p, u)$ of (1) is defined as solution of the initial value problem $\dot{x} = f(x, u)$, $x(0) = p$.

and $\xi(0) \in K$, see [20, Thm. VIII.5]. Note that $\beta$ is obtained by evaluating the solution of the initial value problem

$$\dot{\zeta}(t) = L(u)\zeta(t) + w, \quad \zeta(0) = r. \tag{5}$$

In the computation of a symbolic model $S_2$ of $S_1$ we restrict our attention to the case in which $X_2$ forms a cover of the state alphabet $X_1$ where the elements of the cover $X_2$ are non-empty, closed hyper-intervals, subsequently, referred to as *cells*. Specifically, we work with a subset $\bar{X}_2 \subseteq X_2$ of congruent cells that are aligned on a uniform grid

$$\eta\mathbb{Z}^n = \{c \in \mathbb{R}^n \mid \exists_{k\in\mathbb{Z}^n} \forall_{i\in[1;n]}\ c_i = k_i\eta_i\} \tag{6}$$

with *grid parameter* $\eta \in (\mathbb{R}_+ \setminus \{0\})^n$, i.e.,

$$x_2 \in \bar{X}_2 \implies \exists_{c\in\eta\mathbb{Z}^n}\ x_2 = c + [\![-\eta/2, \eta/2]\!]. \tag{7}$$

The remaining cells $X_2 \setminus \bar{X}_2$ are considered as "overflow" symbols, see [6, Sect III.A]. The symbolic model of $S_1$ is given by $S_2 = (X_2, U_2, F_2)$ where $U_2$ is a finite subset of $U_1$ and $F_2(x_2, u) = \varnothing$ for all $x_2 \in X_2 \setminus \bar{X}_2$. For the remaining cells $x_2 \in \bar{X}_2$, the transition function is computed according to Alg. 1. For a correct implementation, the growth bound $\beta$ needs to be defined w.r.t. $\tau > 0$, $\cup_{x_2\in\bar{X}_2} P(x_2)$, and $U_2$.

**Algorithm 1** Computation of $F_2 : \bar{X}_2 \times U_2 \rightrightarrows \bar{X}_2$

**Require:** $\bar{X}_2$, $U_2$, $\beta$, $\varphi$, $z$, $r = \eta/2$, $\tau$
1: **for all** $c + [\![-r, r]\!] \in \bar{X}_2$ and $u \in U_2$ **do**
2: $r' := \beta(r + z, u)$
3: $c' := \varphi(\tau, c, u)$
4: $A := \{x_2' \in X_2 \mid (c' + [\![-r' - z, r' + z]\!]) \cap x_2' \neq \varnothing\}$
5: **if** $A \subseteq \bar{X}_2$ **then**
6: $F_2(x_2, u) := A$
7: **else**
8: $F_2(x_2, u) := \varnothing$

Using a similar line of reasoning as in [20, Thm. VIII.4], we can show that $Q' = Q \circ P$ with $Q \subseteq X_1 \times X_2$ defined by $(x_1, x_2) \in Q$ iff $x_1 \in x_2$, is a feedback refinement relation from $S_1$ to $S_2$. It follows that a refined controller from an abstract controller is robust w.r.t. the measurement errors $P$, see [20, Thm. VI.4].

## 2.2 (Abstract) Specifications

Currently, SCOTS supports the synthesis of controllers to enforce reachability and invariance (often referred to safety) specifications [4]. Given a simple system $S_1 = (X_1, U_1, F_1)$ let $X_1^\infty = \cup_{T\in\mathbb{N}\cup\{\infty\}} X_1^{[0;T[}$. A *reachability* specification associated with $I_1, Z_1 \subseteq X_1$ is defined by

$$\Sigma_1 = \{x_1 \in X_1^\infty \mid x_1(0) \in I_1 \implies \exists_{t\in[0;T[} : x_1(t) \in Z_1\}.$$

An *invariance* specification associated with $I_1, Z_1$ follows by

$$\Sigma_1 = \{x_1 \in X_1^{[0;\infty[} \mid x_1(0) \in I_1 \implies \forall_{t\in[0;\infty[} : x_1(t) \in Z_1\}.$$

SCOTS supports two classes of sets to define $I_1$ and $Z_1$:

- polytopes $R = \{x \in \mathbb{R}^n \mid Hx \leq h\}$ parameterized by $H \in \mathbb{R}^{q\times n}$, $h \in \mathbb{R}^q$, and
- ellipsoids $E = \{x \in \mathbb{R}^n \mid |L(x - y)|_2 \leq 1\}$ parameterized by $L \in \mathbb{R}^{n\times n}$ and $y \in \mathbb{R}^n$, where $|\cdot|_2$ denotes the Euclidean norm.

Consider the plant $S_1$, the symbolic model $S_2$ and the relations $Q, P$ as defined in the previous subsections. Let $\Sigma_1$ be a reachability (invariant) specification associated with $I_1$ and $Z_1$. An abstract specification $\Sigma_2$ for $S_2$ is simply obtained as reachability (invariant) specification associated with $I_2$ and $Z_2$, where $I_2$ is an outer approximation of $I_1$, i.e., $x_1 \in I_1$ implies $Q \circ P(x_1) \subseteq I_2$ and $Z_2$ is an inner approximation, i.e., $x_2 \in Z_2$ implies $P^{-1} \circ Q^{-1}(x_2) \subseteq Z_1$.

## 2.3 Synthesis via fixed point computations

For the synthesis of controllers $C$ to enforce reachability, respectively, invariance specifications, SCOTS provides two fixed point algorithms. Consider $S_2 = (X_2, U_2, F_2)$ with $X_2$ finite and $I_2, Z_2 \in X_2$. For $Y_2 \subseteq X_2$, we define the map

$$\mathrm{pre}(Y_2) = \{x_2 \in X_2 \mid \exists_{u\in U_{S_2}(x_2)} F_2(x_2, u_2) \subseteq Y_2\}.$$

Consider the functions

$$\check{G}(Y) = \mathrm{pre}(Y) \cup Z_2 \quad \text{and} \quad \hat{G}(Y) = \mathrm{pre}(Y) \cap Z_2.$$

SCOTS supports the minimal fixed point computation of $\check{G}$ and the maximal fixed point computation of $\hat{G}$. Let us shortly recall how we can extract a controller from a fixed point computation.

Suppose we are given a reachability problem $(S_2, \Sigma_2)$, i.e., $\Sigma_2$ is a reachability specification for $S_2$ associated with $I_2$ and $Z_2$. Let $Y_\infty$ denote the minimal fixed point of $\check{G}$ and consider the sets $Y_0 = \varnothing$, $Y_{i+1} = \check{G}(Y_i)$ obtained in the fixed point iteration. Let $j(x) = \inf\{i \in \mathbb{N} \mid x \in Y_i\}$. Then we derive a controller as a system according to [20, Def. III.1] (ver. 2) by $C = (\{q\}, \{q\}, X_2, X_2, U_2, F_c, H_c)$ with

$$H_c(q, x_2) = \begin{cases} H_c'(x_2) \times \{x_2\} & \text{if } x_2 \in Y_\infty \\ U_2 \times \{x_2\} & \text{otherwise} \end{cases}$$

$$F_c(q, x_2) = \begin{cases} \{q\} & \text{if } x_2 \in Y_\infty \\ \varnothing & \text{otherwise} \end{cases}$$

where $H_c'(x_2) = \{u \in U_{S_2}(x_2) \mid F_2(x_2, u) \subseteq Y_{j(x_2)-1}\}$.

To solve an invariance problem $(S_2, \Sigma_2)$, i.e., $\Sigma_2$ is in an invariance specification associated with $I_2$ and $Y_2$, we compute the maximal fixed point $Y_\infty$ of $\hat{G}(\cdot)$. The controller is identical to the reachability controller with the difference that $H_c'$ is set to $H_c'(x_2) = \{u \in U_{S_2}(x_2) \mid F_2(x_2, u) \subseteq Y_\infty\}$ if $x_2 \in Y_\infty$ and otherwise to $H_c'(x_2) = U_2$.

For both cases, it is well known that $C_2$ solves the control problem $(S_2, \Sigma_2)$ iff $I_2 \subseteq Y_\infty$, see e.g. [4]. Also for both types of specifications the controller is *memoryless* or *static*.

# 3. TOOL DETAILS

In this section we describe the architecture of SCOTS. The C++ part of the software tool provides the algorithms to compute the symbolic model and to compute the minimal and maximal fixed points. The algorithms are basically distributed across three classes:

- `SymbolicSet`
- `SymbolicModel > SymbolicModelGrowthBound`
- `FixedPoint`

## 3.1 SymbolicSet

The class `SymbolicSet` is used to define the symbolic state space $\bar{X}_2$ (excluding the overflow symbols), the symbolic input space $U_2$, and the atomic propositions such as the target set or safe set $Z_2 \subseteq \bar{X}_2$. It accepts the state space dimension $n$, the grid parameter $\eta \in \mathbb{R}_+^n$ and a compact hyper-rectangle $[\![a, b]\!]$, $a, b \in \mathbb{R}^n$, $a \leq b$ as input. Optionally, the measurement error bound $z \in \mathbb{R}_+^n$ as given in (2) can

be provided. The default value is set to $z = 0$. We use the `SymbolicSet` to represent a subset of the grid points in $[\![a, b]\!] \cap \eta\mathbb{Z}^n$. Initially, the set is empty and one should use the various methods of the `SymbolicSet` like `addGridPoints`, `addPolytope`, `remPolytope`, `addEllipsoid` or `remEllipsoid` to add and remove grid points to the symbolic set. Each command (except the `addGridPoints`) accepts the option `INNER` and `OUTER` whose usage is as follows.

Suppose we want to add grid points associated with the set $R = \{x \in \mathbb{R}^n \mid Hx \leq h\}$. In order to add the set $\{p \in [\![a, b]\!] \cap \eta\mathbb{Z}^n \mid p + [\![-\eta/2 - z, \eta/2 + z]\!] \cap R \neq \varnothing\}$ we pick `OUTER` as option in `addPolytope`. If we pick `INNER`, the set $\{p \in [\![a, b]\!] \cap \eta\mathbb{Z}^n \mid p + [\![-\eta/2 - z, \eta/2 + z]\!] \subseteq R\}$ is added. Similarly, for an ellipsoid $\{x \in \mathbb{R}^n \mid |L(x - y)|_2 \leq 1\}$ with $L \in \mathbb{R}^{n \times n}$ and $y \in \mathbb{R}^n$, the outer and inner approximations are determined by $\{p \in [\![a, b]\!] \cap \eta\mathbb{Z}^n \mid |L(p - y)|_2 \leq 1 + r\}$ and $\{p \in [\![a, b]\!] \cap \eta\mathbb{Z}^n \mid |L(p - y)|_2 \leq 1 - r\}$, respectively, where $r = \max_{x \in [\![-\eta/2 - z, \eta/2 + z]\!]} |Lx|_2$.

Technically, we interpret the map $H'_c : Z_\infty \rightrightarrows U_2$ as a subset of $\bar{X}_2 \times U_2$ and represent the controller function $H'_c$ also as `SymbolicSet`. To this end, we can instantiate a `SymbolicSet` with two instances of a `SymbolicSet` with parameters $n, \eta, [\![a, b]\!]$ and $m, \mu, [\![c, d]\!]$, where $n, \eta, [\![a, b]\!]$ and $m, \mu, [\![c, d]\!]$ are the parameters associated with $\bar{X}_2$ and $U_2$. respectively.

Any instance of a `SymbolicSet` set can be written with all the relevant information to a file. Such a file can be loaded within the C++ library as well as within Matlab. In that way one can use Matlab's build-in functions not only to visualize various entities like atomic propositions and the domain of the controller, but also to simulate the closed loop.

We use binary decision diagrams (BDDs) [21] as underlying data structure. Specifically we use the object oriented wrapper to the `CUDD` library [22].

## 3.2 SymbolicModel

The class `SymbolicModel` is the base class of `SymbolicModelGrowthBound`. The base class manages BDD related information, such as number and indices of BDD variables. Alg. 1 to compute $S_2 = (X_2, U_2, F_2)$ using growth bounds is implemented in the class `SymbolicModelGrowthBound` which requires the ordinary differential equations (1) and (5) as inputs. In our case studies, we use a Runge-Kutta scheme.

## 3.3 FixedPoint

The `FixedPoint` class is instantiated with an instance of a `SymbolicModel` and provides two methods `minimalFixedPoint` and `maximalFixedPoint` for the fixed point computation of $\check{G}$ and $\hat{G}$. The input to those methods is a BDD that represents the set $Z_2$. The outputs are two BDDs, one represents the fixed point and one the map $H'_c$. The results can be stored to file by first, adding the BDDs to the appropriate instances of a `SymbolicSet` and second, using the `writeToFile` method provided by `SymbolicSet`.

The general work flow with the different user inputs and the possible tool output is illustrated in Fig. 2.

# 4. NUMERICAL EXPERIMENTS

The tool and all details of the conducted experiments can be found at www.hcs.ei.tum.de

## 4.1 A path planning problem

As a first example we consider a path planning problem, also considered in [20] and [9], for the bicycle dynamics of a

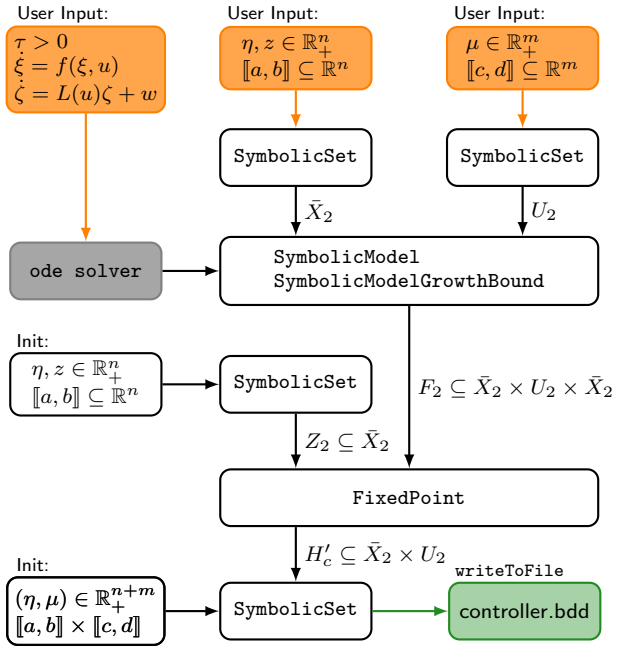


**Figure 2: The work flow in `SCOTS` to compute a symbolic model $S_2 = (X_2, U_2, F_2)$ of the sampled system $S_1$ associated with (1) and to synthesize a controller to enforce an invariance (reachability) specification where $Z_2 \subseteq \bar{X}_2$ is the safe (target) set.**

vehicle. The control system consists of a 3-dimensional state space and a 2-dimensional input space. The states $(x_1, x_2)$ and $x_3$ correspond to the position, respectively, orientation of the vehicle in the 2-dimensional plane. The inputs are given by the velocity and steering angle. There are no measurement errors or perturbations present. The objective is to steer the robot from a given initial position (green dot) to a target set (red rectangle) while avoiding the obstacles (blue rectangles), see Fig. 3. The precise control problem can be found in [20].

In order to use `SCOTS` we create a C++ file `vehicle.cc` in which we include, among other, the cudd library `cuddObj.hh` and the header-only classes `SymbolicSet.hh`, `SymbolicModelGrowthBound.hh` and `FixedPoint.hh`. We begin with the definition of the dynamics and the growth bound. For example, to provide the nominal solution $\varphi$ for Alg. 1 we write

```
typedef std::array<double,3> state_t; /* state type */
typedef std::array<double,2> input_t; /* input type */
auto vehicle_post = [](state_t &x, input_t &u) {
  /* the ode describing the vehicle dynamics */
  auto rhs =[](state_t& xx, const state_t &x, const input_t &u) {
    double alpha=std::atan(std::tan(u[1])/2.0);
    xx[0] = u[0]*std::cos(alpha+x[2])/std::cos(alpha);
    xx[1] = u[0]*std::sin(alpha+x[2])/std::cos(alpha);
    xx[2] = u[0]*std::tan(u[1]);
  };
  size_t nint=5; /* number of intermediate steps */
  double h=0.06; /* h*nint = sampling time */
  ode_solver(rhs,x,u,nint,h); /* runge kutte solver */
};
```

Subsequently, we define the uniform grids for the sets $\bar{X}_2$ and $U_2$ as `SymbolicSet`. To define $\bar{X}_2$ on the hyper-rectangle $[\![lb, ub]\!]$ with grid parameter $\eta$ we write

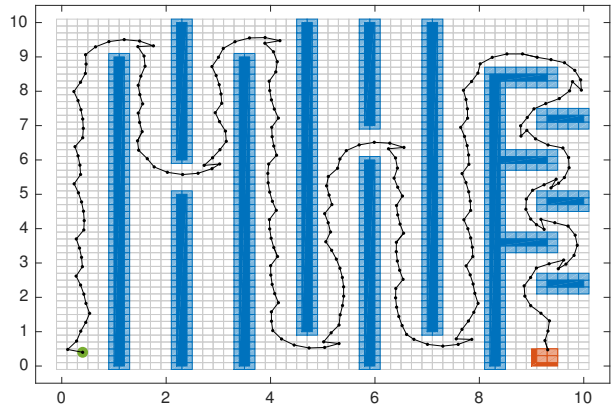

**Figure 3: The green dot, the dark blue, and dark red rectangles correspond to the original specifications. The light blue cells are added to the overflow symbols in $X_2$ and the light red cells are used to define the abstract reachability specification.**

```
double lb[3]={0,0,-M_PI-0.4}; /* lower bounds */
double ub[3]={10,10,M_PI+0.4}; /* upper bounds */
double eta[3]={.2,.2,.2}; /* grid parameter */
scots::SymbolicSet ss(mgr,3,lb,ub,eta); /* the uniform grid */
ss.addGridPoints(); /* fill the SymbolicSet */
```

At first the `SymbolicSet` is empty. In the last line we add all grid points contained in $[\![lb, ub]\!]$ to `ss`. It is possible to account for the obstacles by adding them to the overflow symbols, i.e., we remove an outer approximation from `ss`. To remove an obstacle we write, for appropriately defined $H \in \mathbb{R}^{4\times 3}$ and $h \in \mathbb{R}^4$

```
/* remove outer approximation of P={ x | H x <= h } from ss */
ss.remPolytope(4,H,h,scots::OUTER);
```

Similarly, we remove all the other obstacles and define a `SymbolicSet ts` to represent the target $Z_2$. The computation of the transition function $F_2$ is implemented by

```
scots::SymbolicModelGrowthBound<state_t,input_t> abs(&ss, &is);
abs.computeTransitionRelation(vehicle_post, radius_post);
```

and the minimal fixed point is computed by

```
scots::FixedPoint reach(&abs);
/* the fixed point algorithm operates directely on BDD's */
BDD target = ts.getSymbolicSet(); /* extract the BDD from ts */
BDD fp; /* the fixed point is stored in fp */
BDD con; /* the controller is stored in con */
reach.minimalFixedPoint(target, fp, con);
```

We store the controller to `vehicle_controller.bdd` by

```
scots::SymbolicSet controller(ss,is);
controller.setSymbolicSet(con);
controller.writeToFile("vehicle_controller.bdd");
```

In order to load the controller in Matlab and to get the inputs associated with a state $x$ we use

```
>> con=SymbolicSet('vehicle_controller.bdd');
>> u=controller.getInputs(x);
```

After we compiled and executed the C++ program, we simulated the controlled vehicle in Matlab. A closed loop trajectory is illustrated in Fig. 3.

A comparison of `SCOTS` with `Pessoa` and the tool developed at the University of Federal Armed Force (UniBW) in Munich (*which is not publicly available*), is listed in Tab. 1. The table contains (apart from the CPU type) the number $\#F_2$ of elements in the transition relation $F_2$, the time $t_{\text{abs}}$ spent to compute the symbolic model and the time $t_{\text{syn}}$ spent to solve the abstract synthesis problem. We conducted two experiments. In `SCOTS` #1, we accounted for the obstacles in the synthesis part. To this end, we implemented a modified minimal fixed point computation so that we can solve reach-avoid problems. In this case $\bar{X}_2$ also contains the obstacles. While in `SCOTS` #2, we added an outer approximation of the obstacles (as described above) to the overflow symbols in $X_2$, which results in a smaller $\bar{X}_2$ and, hence, a smaller number of transitions.

| | CPU [GHz] | $\#F_2$ | $t_{\text{abs}}$[sec] | $t_{\text{syn}}$[sec] |
|---|---|---|---|---|
| `Pessoa` | Core2Duo 2.4 | 34020088 | 13509 | 535 |
| `SCOTS` #1 | i7 3.5 | 37316812 | 100 | 413 |
| UniBW | i7 2.9 | 28398299 | 2.33 | 0.22 |
| `SCOTS` #2 | i7 3.5 | 18991758 | 53 | 210 |

**Table 1: Comparison of `SCOTS` with numbers reported in [9] and [20].**

Unsurprisingly, `SCOTS` outperforms `Pessoa` in terms of the computation times for the symbolic model. This stems from the fact that in `Pessoa` (at least for the nonlinear case) the computation is partly implemented in C/C++ and partly implemented in Matlab, which causes a large overhead. While the difference in the synthesis times is explainable by the different CPUs. The UniBW tool outperforms both `Pessoa` and `SCOTS` with respect to both computation times. We believe that the difference in $t_{\text{abs}}$ is mainly due to the large overhead that the management (accessing, iterating over, adding, and removing elements) of the BDD data structure requires, see also [15] for a similar observation. Additionally to the efficient data structure and contrary to the iterative implementation of the minimal fixed point in `SCOTS` and `Pessoa`, the UniBW tool uses a Dijkstra like algorithm [24] and is therefore able to drastically reduce $t_{\text{syn}}$.

## 4.2 DC-DC boost converter

In this case study, we synthesize a controller for a DC-DC boost converter to enforce a invariance specification. The DC-DC boost converter is modeled by a switched system with two modes, with a 2-dimensional linear dynamics in each mode. The system is incrementally stable and therefore, amenable to the construction of approximately bisimilar symbolic models, see [19] for details. We solve the synthesis problem using `SCOTS` as well as `Pessoa`. The domain of the controller, synthesized using `SCOTS`, together with a closed trajectory is illustrated in Fig. 4. We run `Pessoa` with two option #1 and #2. For option #1 we computed an approximately bisimilar symbolic model, which results in a deterministic transition function. For option #2, we computed an approximately alternatingly similar symbolic model, whose computation is based on attainable sets and therefore closer to the symbolic model obtained with `SCOTS`. The run times of various computations are listed in Tab. 2. Surprisingly, `SCOTS` outperforms `Pessoa` in terms of the construction of the symbolic model, even though, contrary to the nonlinear case, all the computations are implemented in C. In the last row of Tab. 2 we list the numbers reported in [15] ($t_{\text{abs}}$ is not listed in [15]). Again we can observe the substantial overhead induced by the BDD usage compared

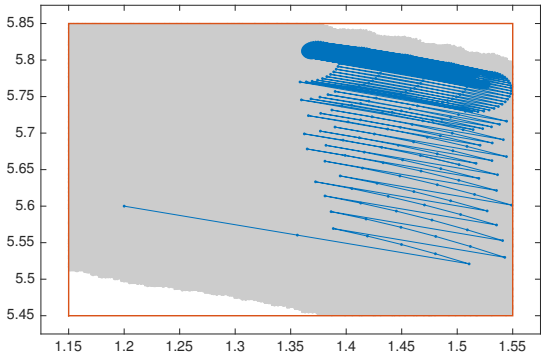


**Figure 4: The domain of the invariance controller.**

| | CPU [GHz] | $t_{\text{abs}}$[sec] | $t_{\text{syn}}$[sec] |
|---|---|---|---|
| `Pessoa` #1 | i7 3.5 | 478.7 | 65.2 |
| `Pessoa` #2 | i7 3.5 | 934.4 | 91.1 |
| `SCOTS` | i7 3.5 | 18.1 | 75.4 |
| `CoSyMA` | – | – | 8.32 |

**Table 2: Comparison of `SCOTS`, `Pessoa` and [15].**

to the hash tables in `CoSyMA`.

## 5. CONCLUSION

In this paper we introduced `SCOTS`, a software tool to synthesize controllers for nonlinear control systems based on symbolic models. Contrary to other available tools, it supports the construction of symbolic models in the framework of feedback refinement relations and therefore facilitates a rather straightforward controller refinement procedure [20]. We illustrated by two examples from the literature the effectiveness of the tool by synthesizing controllers for invariance and reachability problems. For those experiments, we could observe a reduction of the time complexity compared to `Pessoa`, whose implementation similar to `SCOTS` is based on BDDs. On the other hand, we observed substantial longer run times compared to tools that uses alternative data structures, e.g. hash tables in `CoSyMA`. Hence, for the future we plan to incorporate such alternatives in `SCOTS` and investigate the various approaches not only in terms of time complexity but also with respect to memory usage. Additionally, we plan to exploit symbolic and/or automatic differentiation and nonlinear optimization methods to support the user in finding tight Lipschitz matrices that are needed in the computation of growth bounds.

## 6. REFERENCES

[1] P. Tabuada and G. J. Pappas. “Linear Time Logic Control of Discrete-Time Linear Systems”. In: *IEEE TAC* 51 (2006), pp. 1862–1877.

[2] A. Abate et al. “Computational approaches to reachability analysis of stochastic hybrid systems”. In: *HSCC*. Springer, 2007, pp. 4–17.

[3] M. Kloetzer and C. Belta. “A fully automated framework for control of linear systems from temporal logic specifications”. In: *IEEE TAC* 53 (2008), pp. 287–297.

[4] P. Tabuada. *Verification and Control of Hybrid Systems – A Symbolic Approach*. Springer, 2009.

[5] G. E. Fainekos et al. “Temporal logic motion planning for dynamic robots”. In: *Automatica* 45 (2009), pp. 343–352.

[6] G. Reißig. “Computing abstractions of nonlinear systems”. In: *IEEE TAC* 56 (2011), pp. 2583–2598.

[7] A. Girard. “Controller synthesis for safety and reachability via approximate bisimulation”. In: *Automatica* 48 (2012), pp. 947–953.

[8] T. Wongpiromsarn, U. Topcu, and R. M. Murray. “Receding Horizon Temporal Logic Planning”. In: *IEEE TAC* 57 (2012), pp. 2817–2830.

[9] M. Zamani et al. “Symbolic Models for Nonlinear Control Systems Without Stability Assumptions”. In: *IEEE TAC* 57 (2012), pp. 1804–1809.

[10] B. Yordanov et al. “Temporal logic control of discrete-time piecewise affine systems”. In: *IEEE TAC* 57 (2012), pp. 1491–1504.

[11] M. Rungger, M. Jr. Mazo, and P. Tabuada. “Specification-Guided Controller Synthesis for Linear Systems and Safe Linear-Time Temporal Logic”. In: *HSCC*. ACM, 2013, pp. 333–342.

[12] J. Liu et al. “Synthesis of Reactive Switching Protocols From Temporal Logic Specifications”. In: *IEEE TAC* 58 (2013), pp. 1771–1785.

[13] M. Zamani et al. “Symbolic control of stochastic systems via approximately bisimilar finite abstractions”. In: *IEEE TAC* 59.12 (2014).

[14] M. Jr. Mazo, A. Davitian, and P. Tabuada. “Pessoa: A tool for embedded controller synthesis”. In: *Computer Aided Verification*. Springer. 2010, pp. 566–569.

[15] S. Mouelhi, A. Girard, and G. Gössler. “CoSyMA: a tool for controller synthesis using multi-scale abstractions”. In: *HSCC*. ACM. 2013, pp. 83–88.

[16] C. Finucane, G. Jing, and H. Kress-Gazit. “LTLMoP: Experimenting with language, temporal logic and robot control”. In: *IROS*. IEEE. 2010, pp. 1988–1993.

[17] T. Wongpiromsarn et al. “TuLiP: a software toolbox for receding horizon temporal logic planning”. In: *HSCC*. ACM. 2011, pp. 313–314.

[18] N. Piterman, A. Pnueli, and Y. Saár. “Synthesis of reactive (1) designs”. In: *Verification, Model Checking, and Abstract Interpretation*. 2006.

[19] A. Girard, G. Pola, and P. Tabuada. “Approximately bisimilar symbolic models for incrementally stable switched systems”. In: *IEEE TAC* 55.1 (2010), pp. 116–126.

[20] G. Reißig, A. Weber, and M. Rungger. *Feedback Refinement Relations for the Synthesis of Symbolic Controllers*. 2015. arXiv: 1503.03715v1.

[21] R. E. Bryant. “Symbolic Boolean manipulation with ordered binary-decision diagrams”. In: *ACM Computing Surveys* 24.3 (1992), pp. 293–318.

[22] F. Somenzi. “CUDD: CU decision diagram package”. In: *University of Colorado at Boulder* (1998).

[23] R. T. Rockafellar and R. J.-B. Wets. *Variational analysis*. Vol. 317. 3rd corr printing 2009. Springer, 1998.

[24] G. Gallo et al. “Directed hypergraphs and applications”. In: *Discrete Applied Mathematics* 42 (1993), pp. 177–201.