SCOTS (0.2) – USER MANUAL

MATTHIAS RUNGGER

Contents

1. ToDo

a lot...

2. About SCOTS v0.2

SCOTS is an open source software tool (available at http://www.hcs.ei.tum.de) published under the 3-Clause BSD License. It provides a basic implementation of the construction of symbolic models, also known as discrete abstractions, of possibly perturbed, nonlinear control systems according to [1] together with the implementation of two algorithms for the synthesis of symbolic controllers. It is mainly implemented in C++ , but also provides a small MATLAB interface to access atomic propositions and the synthesized controllers from the MATLAB workspace.

SCOTS natively supports invariance and reachability specifications. It can also be used in combination with the synthesis tool `slugs` [2] to account for general reactivity one (GR(1)) specifications. Moreover, expert users have the possibility to write customized synthesis algorithms.

SCOTS is mainly intended to be used (and possibly extended) by researchers and lecturers in the area of formal methods for cyber-physical systems. The implementation does not use validated numerics or similar methods for rigorous implementations and such is prone to ODE solver inaccuracies and rounding errors.

This manual gives an overview of SCOTS and contains installation notes, usage details and a brief theoretical background. For implementation details please see the `doxygen` documentation in `./doc`.

Although, there are no compelling reasons, why SCOTS should not work under Windows, we developed and tested the code only under Linux and macOS environments. As a result, the installation notes apply to Linux/macOS systems only.

Bug reports and feature requests should be mailed to matthias.rungger@tum.de.

3. Quickstart

The best way to try SCOTS is to clone the source code from https://gitlab.lrz.de/matthias/SCOTSv0.2 and run one of the examples. Each of the example directories contains a `readme` file that provides some background information on the example itself and explains the compilation process.

Optionally, some examples contain an m-file for the simulation of the closed loop in MATLAB. See http://www.mathworks.com for installation instructions.

3.1. Invariance and Reachability.
The three most easy-to-compile examples are found in

```
./examples/dcdc     /* invariance problem for a DCDC boost converter */
./examples/vehicle  /* reach-avoid problem for a vehicle */
./examples/aircraft /* landing maneuver of an aircraft (requires 32GB memory) */
```

The examples can be run without any additional software and require only a C++ compiler with C++ 11 support.

3.2. A Priori Enclosure and Growth Bound.
The directory

```
./examples/aircraft/helper
```

contains various programs that use the interval-arithmetic based ODE solver `vnodelp` to obtain: **a)** an a priori enclosure and **b)** a growth bound.

4. Installation Notes

SCOTS is implemented in "header-only" style and only a working C++ developer environment with C++ 11 support is needed. In the basic variant it is possible to

(1) compute abstractions
(2) synthesize controllers with respect to invariance and reachability specifications
(3) simulate the closed loop in C++

For various reasons, one might consider to use additional software in combination with SCOTS.

4.1. **Additional software.**

(1) MATLAB: for closed loop simulation and visualization options.
See `http://www.mathworks.com` for installation instructions.
To access the controllers produced by SCOTS from the MATLAB workspace the mex file `mexStaticController.mex` needs to be compiled:
  (a) setup the mex compiler with the MATLAB command

```
>> mex -setup C++
```

  (b) In a terminal, navigate to `./mfiles/mexfiles`:
    - edit the `makefile` and adjust the `MATLABPATH`
    - run `make sparse`

(2) `boost`: to use advance ode solvers.
For Linux `boost` is usually distributed via the package management system. On macOS `MacPorts` `https://www.macports.org/` or `Homebrew` `http://brew.sh/` provide an easy way to install `boost`.

(3) `vnodelp`: to compute a priori enclosures and growth bounds.
`vnodelp` is an open source software tool to compute validated solutions of initial value problems based on interval arithmetic. It is available at `http://www.cas.mcmaster.ca/~nedialk/vnodelp/`.

Part 1. THEORY BASICS

5. The Symbolic Approach

A detailed description of *the symbolic approach to controller synthesis* that is implemented in SCOTS is presented in [1]. The article [1] contains also a detailed explanation of the notation that we use in the following text.

5.1. Control Problems.

SCOTS supports the computation of controllers for nonlinear 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 $U \subseteq \mathbb{R}^m$. The vector $w = [w_1, \ldots, w_n] \in \mathbb{R}^n_+$ is a perturbation bound and $[\![-w, w]\!]$ denotes the hyper-interval $[-w_1, w_1] \times \ldots \times [-w_n, w_n]$. Given a time horizon $\tau > 0$, we define a *solution of* (1) *on* $[0, \tau]$ *under (constant) input* $u \in U$ as an absolutely continuous function $\xi : [0, \tau] \to \mathbb{R}^n$ that satisfies (1) for almost every (a.e.) $t \in [0, \tau]$.

The desired behavior of the closed loop is defined with respect to the τ-sampled behavior of the continuous-time systems (1). To this end, the sampled behavior of (1) is casted as *simple system* (with initial states)

$$S_1 := (X_1, X_{1,0}, U_1, F_1) \tag{2}$$

with the *state alphabet* $X_1 := \mathbb{R}^n$, the *set of initial states*, the *input alphabet* $U_1 := U$ and the *transition function* $F_1 : X_1 \times U_1 \rightrightarrows X_1$ defined by

$$F_1(x, u) := \{x' \mid \exists_{\xi \text{ is a solution of (1) on } [0,\tau] \text{ under } u} : \xi(0) = x \wedge \xi(\tau) = x'\}. \tag{3}$$

A *specification* Σ_1 for a simple system (2) is simply a set

$$\Sigma_1 \subseteq (U_1 \times X_1)^\infty := \bigcup_{T \in \mathbb{Z}_{\geq 0} \cup \{\infty\}} (U_1 \times X_1)^{[0;T[} \tag{4}$$

of possibly finite and infinite input-state sequences. A simple system S_1 together with a specification Σ_1 constitute an *control problem* (S_1, Σ_1).

The *solution* of a control problem (S_1, Σ_1) is a *system* $C = (X_c, X_{c,0}, U_c, V_c, Y_c, F_c, H_c)$ which is *feedback composable* with S_1, see [1, Def. III.3], and satisfies

$$\mathcal{B}(C \times S_1) \subseteq \Sigma_1.$$

In this context C and $C \times S_1$ are usually referred to as *controller*, respectively, *closed loop*. The symbol $\mathcal{B}(C \times S_1)$ denotes the *behavior* of the closed loop $C \times S_1$, see [1, Def. V.1]. We say that a control problem (S_1, Σ_1) is *solvable* iff there exists a system C that solves (S_1, Σ_1).

A block diagram of the feedback composition of a controller synthesized with SCOTS and the system S_1 is illustrated in Fig. 1.

5.2. Supported Specifications.

SCOTS natively supports

- invariance (often referred to as safety) specifications;
- reachability specifications;
- reach-avoid specifications.

An *invariance* specification for (2) associated with $Z_1 \subseteq X_1$ is defined by

$$\Sigma_1 := \{(u, x) \in (U_1 \times X_1)^{[0;\infty[} \mid \forall_{t \in [0;\infty[} : x(t) \in Z_1\}.$$

A *reachability* specification for (2) associated with $Z_1 \subseteq X_1$ is defined by

$$\Sigma_1 := \{(u, x) \in (U_1 \times X_1)^\infty \mid \exists_{t \in [0;\infty[} : x(t) \in Z_1\}.$$

A *reach-avoid* specification for (2) associated with $A_1 \subseteq X_1$ and $Z_1 \subseteq X_1$ is defined by

$$\Sigma_1 := \left\{(u, x) \in (U_1 \times X_1)^\infty \mid \exists_{t \in [0;\infty[} : x(t) \in Z_1 \wedge \forall_{t' \in [0;t[} : x(t') \notin A_1\right\}.$$

In the context of Linear Temporal Logic, the sets A_1 and Z_1 are often identified with *atomic propositions*. In this sense, SCOTS allows to define arbitrary sets as atomic propositions.

5.3. **Auxiliary Control Problems.** Given a simple system (2) representing the τ-sampled behavior of (1) and a specification Σ_1 for (2), the control problem (S_1, Σ_1) is not solved directly, but an auxiliary, finite control problem (S_2, Σ_2) is used in the synthesis process. Here,

$$S_2 = (X_2, X_{2,0}, U_2, F_2) \tag{5}$$

is referred to as *symbolic model* or (discrete) *abstraction* of S_1 and Σ_2 is an *abstract specification*.

The state alphabet of X_2 is a cover of X_1 and the input alphabet U_2 is a subset of U_1. The set X_2 contains a subset $\bar{X}_2$, representing the "real" quantizer symbols, while the remaining symbols $X_2 \setminus \bar{X}_2$ are interpreted as "overflow" symbols. The set of real quantizer symbols $\bar{X}_2$ are given by congruent hyper-rectangles 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* $\eta \in (\mathbb{R}_+ \setminus \{0\})^n$. The elements of $\eta\mathbb{Z}^n$ are called *grid points*. The real quantizer symbols are further parameterized by two vectors $a, b \in \mathbb{R}^n$ representing the lower-left and upper-right corners of the hyper-interval $[\![a, b]\!]$ confining the set $\bar{X}_2$:

$$\bar{X}_2 := \{x_2 \mid \exists_{c\in(\eta\mathbb{Z}^n\cap[\![a,b]\!])}\, x_2 = c + [\![-\eta/2, \eta/2]\!]\}. \tag{7}$$

The elements of the real quantizer symbols are also referred to as *cells*. Each cell $x_2 = c + [\![-\eta/2, \eta/2]\!]$ is associated with a *center* $c \in \mathbb{R}^n$ (which is also a *grid point* $c \in \eta\mathbb{Z}^n$) and a *radius* $r \in \mathbb{R}^n_{\geq 0}$.

SCOTS computes symbolic models that are related via feedback refinement relations with the plant. A *feedback refinement relation* from S_1 to S_2 is a strict relation $Q \subseteq X_1 \times X_2$ that satisfies for all $(x_1, x_2) \in Q$ and $u \in U_2$ the conditions

(1) $x_1 \in X_{1,0}$ implies $x_2 \in X_{2,0}$
(2) $F_2(x_2, u) \neq \varnothing$ implies $F_1(x_1, u) \neq \varnothing$ and $Q(F_1(x_1, u)) \subseteq F_2(x_2, u)$.

In SCOTS, the feedback refinement relation Q is given by the set-membership relation

$$Q := \{(x_1, x_2) \mid x_1 \in x_2\}. \tag{8}$$

Given an invariance (reachability) specification Σ_1 for (2) associated with Z_1, then an *abstract specification* is given by the invariance (reachability) specification for $S_2 = (X_2, X_{2,0}, U_2, F_2)$ associated with

$$Z_2 = \{x_2 \in X_2 \mid x_2 \subseteq Z_1\}. \tag{9}$$

An abstract reach-avoid specification from A_1, Z_1 for S_2 follows by

$$A_2 = \{x_2 \in X_2 \mid x_2 \cap A_1 \neq \varnothing\} \tag{10}$$

and Z_2 as defined in (9). The algorithms to solve the control problems (S_2, Σ_2) implemented in SCOTS are outlined in Section 7.

5.4. **Closed Loop.** The main statement facilitating the use of an auxiliary control problem reads as follows [1, Thm. VI.3]:

Consider two control problems (S_i, Σ_i), $i \in \{1, 2\}$. Suppose that Q is a feedback refinement relation from S_1 to S_2 and Σ_2 is an abstract specification of Σ_1. If C solves the control problem (S_2, Σ_2), then $C \circ Q$ solves the control problem (S_1, Σ_1).

The controller $C \circ Q$ for S_1 is given by the serial composition of the quantizer $Q : X_1 \rightrightarrows X_2$ with the controller C. The closed loop resulting from a simple system Σ_1 which represents the τ-sampled behavior of (1) and a controller $C \circ Q$ is illustrated in Fig. 1. At each $k \in \mathbb{Z}_{\geq 0}$ sampling time $\tau > 0$, the plant state $x_1 = \xi(k\tau)$ is measured and fed to the quantizer Q, which is used to determine a cell $x_2 \in X_2$ that contains $x_1 \in x_2$. Then x_2 is fed to the controller C to pick the input $u \in U_2 \subseteq U_1$ which is applied to (1).

Additionally to the perturbations on the right-hand-side of (1), it is possible to account for measurement errors modeled by a set-valued map $P \colon \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ given by

$$P(x) := x + [\![-z, z]\!] \quad \text{with} \quad z \in \mathbb{R}^n_+. \tag{11}$$

Please see [1, Sec. VI.B] and [3] for some background theory. The closed loop with measurement errors is illustrated in Fig. 2.

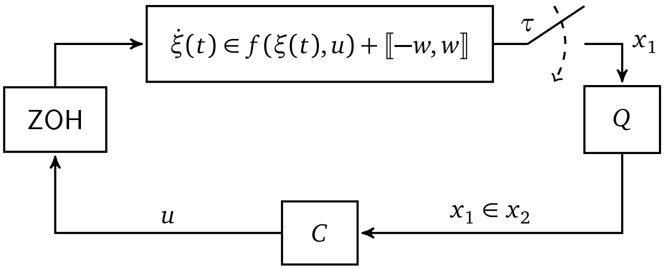

FIGURE 1. Sample-and-hold implementation of a controller synthesized with SCOTS.

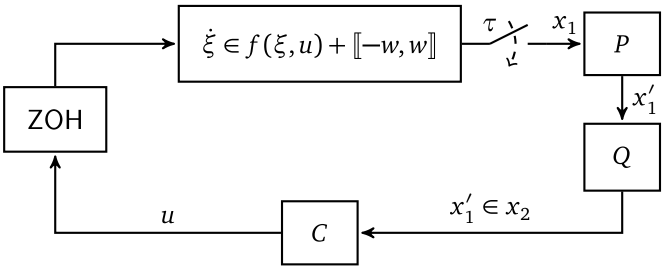

FIGURE 2. Closed loop with measurement errors modeled by the set-valued map $x_1' \in P(x_1)$.

6. Construction of Symbolic Models

6.1. **Growth Bound and A Priori Enclosure.** The construction of a symbolic model S_2 of S_1 is based on the over-approximation of attainable sets. In SCOTS, the over-approximation of the attainable sets requires a so-called growth bound [1]. A *growth bound* of (1) is a function $\beta : \mathbb{R}_+^n \times U' \to \mathbb{R}_+^n$, 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 ξ of (1) from *nominal solutions*[1] φ of (1), i.e., for every solution ξ of (1) on $[0, \tau]$ with input $u \in U'$ and $\xi(0)$, $p \in K$ we have

$$|\xi(\tau) - \varphi(\tau, p, u)| \leq \beta(|\xi(0) - p|, u). \tag{12}$$

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

$$L_{i,j}(u) \geq \begin{cases} D_j f_i(x, u) & \text{if } i = j, \\ |D_j f_i(x, u)| & \text{otherwise} \end{cases} \tag{13}$$

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

$$\beta(r, u) = e^{L(u)\tau} r + \int_0^\tau e^{L(u)s} w \, \mathrm{d}s, \tag{14}$$

is a growth bound on $[0, \tau]$, K, U' associated with (1). The set K' on which (13) needs to hold, is a so-called *a priori enclosure*, i.e., K' is assumed to be convex and contain any solution ξ on $[0, \tau]$ of (1) with $u \in U'$ and $\xi(0) \in K$, see [1, Thm. VIII.5].

In order to use SCOTS, the user needs to provide a growth bound, which for nonlinear control systems can be provided in terms of the parameterized matrix $L(u)$ whose entries satisfy (13). A priori enclosures as well as growth bounds can be computed automatically using interval arithmetic based ODE solvers. See Section 8 for more details on how to automatically obtain a priori enclosures and growth bounds.

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

6.2. **The Transition Function.** Recall that the state alphabet X_2 of the symbolic model (5) is composed of the real quantizer symbols $\bar{X}_2$, which are cells aligned on a uniform grid, and the overflow symbols $X_2 \smallsetminus \bar{X}_2$. For $x_2 \in X_2 \smallsetminus \bar{X}_2$ the transition function is defined for all $u \in U_2$ by

$$F_2(x_2,u) := \varnothing. \tag{15}$$

In order to determine the successors $x_2' \in F_2(x_2,u)$ for $x_2 = c + [\![-\eta/2,\eta/2]\!] \in \bar{X}_2$ and $u \in U_2$, we first compute the hyper-interval

$$R := \varphi(\tau,c,u) + [\![-\beta(\eta/2,u),\beta(\eta/2,u)]\!] \tag{16}$$

which is an over-approximation of the attainable set of (1) with respect to the set $c + [\![-\eta/2,\eta/2]\!]$ and input u. If P is not covered by the real quantizer symbols, i.e., $R \not\subseteq \cup_{x_2 \in \bar{X}_2} x_2$, then we define

$$F_2(x_2,u) := \varnothing. \tag{17}$$

Otherwise, we define the successor cells function by

$$x_2' \in F_2(x_2,u) :\iff x_2' \cap R \neq \varnothing. \tag{18}$$

Using similar arguments as in [1, Thm. VIII.4], it is straightforward to show that Q is a feedback refinement relation from S_1 to S_2. Note that F_2 satisfies

$$F_2(x_2,u) \subseteq \bar{X}_2. \tag{19}$$

If we need to be robust against measurement errors $P(x) = x + [\![-z,z]\!]$, we slightly modify the computation of R to

$$R := \varphi(\tau,c,u) + [\![-\beta(\eta/2+z,u),\beta(\eta/2+z,u)]\!] \tag{20}$$

and define the transition function to (instead of (18))

$$x_2' \in F_2(x_2,u) :\iff (x_2' + [\![-z,z]\!]) \cap R \neq \varnothing. \tag{21}$$

As a result we obtain that $Q \circ P$ is a feedback refinement relation from S_1 to S_2, see [4, Thm. III.5], which enables the correct controller refinement under measurement errors [1, Sec. VI.B]. The use of the perturbation parameter $z \in \mathbb{R}^n_{>0}$ in SCOTS is explained in detail in Section 8.

In the implementation of the computation of F_2 in `AbstractionGB.hh`, we use a numerical ODE solver to compute an approximation of $\varphi(\tau,c,u)$ as well as $\beta(\eta/2,u)$.

7. Controller Synthesis

In this section, we discuss the algorithms that are implemented in SCOTS to solve synthesis problems for the finite symbolic model (5). Usually, synthesis algorithms are developed in the context of two-player games on graphs. The player associated with the controller tries to enforce the specification while the player associated with disturbances tries to violate the specification, e.g. [5].

Given a control problem (S_2,Σ_2) with S_2 given in (5), the construction of a controller to enforce a specification Σ_2 proceeds in two steps. First, a subset Y_∞ of the state alphabet X_2 is computed, which characterizes the largest set of initial states so that the control problem is solvable, i.e., (S_2,Σ_2) is solvable if and only if $X_{2,0} \subseteq Y_\infty$. The set Y_∞ is referred to as *winning domain* (or set of *winning states*) associated with (S_2,Σ_2). In a second step, the controller C is derived form the set Y_∞ and some other information available in synthesis algorithm.

We use the following notation. The set of *admissible inputs* at $x_2 \in X_2$ is denoted by

$$U_{S_2}(x_2) := \{u \in U_2 \mid F_2(x_2,u) \neq \varnothing\}. \tag{22}$$

We use the Weierstrass symbol $\wp$ to denote the power set and define $\mathrm{pre} : \wp(X_2) \to \wp(X_2)$ by

$$\mathrm{pre}(Y) := \{x_2 \in X_2 \mid \exists_{u \in U_{S_2}(x_2)} : F_2(x_2,u) \subseteq Y\}. \tag{23}$$

Algorithm 1 Controller synthesis for invariance specs associated Z_2

Input: Z_2, $S_2 = (X_2, U_2, F_2)$
Require: $F_2(X_2, U_2) \subseteq \bar{X}_2$ and $Z_2 \subseteq \bar{X}_2$

$Q := \varnothing$ ▷ FIFO queue of bad states
$E := \varnothing$ ▷ bookkeeping of the bad states
$D := \varnothing$ ▷ valid state-input pairs
for all $x_2 \in \bar{X}_2$ **do**
  if $x_2 \notin Z_2$ or $U_2(x_2) = \varnothing$ **then** ▷ mark all states outside Z_2 or blocking states as bad
    $Q := Q \cup \{x_2\}$
  else
    $D := D \cup (\{x_2\} \times U_2(x_2))$
$E := Q$
while $Q \neq \varnothing$ **do**
  $x'_2 :\in Q$ ▷ remove oldest element
  $Q := Q \setminus \{x'_2\}$
  for all $(x_2, u) \in F_2^{-1}(x'_2)$ **do**
    $D := D \setminus \{(x_2, u)\}$ ▷ remove state-input pairs that lead to bad states
    if $x_2 \notin D^{-1}(U_2)$ and $x_2 \notin E$ **then** ▷ no valid input left and not already marked bad
      $Q := Q \cup \{x_2\}$ ▷ add to queue of bad states
      $E := E \cup \{x_2\}$

Output: D

7.1. **Invariance.** Let Σ_2 be an invariance specification associated with Z_2. We use Alg. 1, which is implemented in the function `solve_invariance_game` in the file `GameSolver.hh`, to synthesize a controller that solves the control problem (S_2, Σ_2). Alg. 1 runs in $O(m)$ time, where m is the number of transitions, i.e., the number of triples (x_2, u, x'_2) with $x'_2 \in F_2(x_2, u)$, since each state is added to the queue of bad states Q at most once.

Let $D \subseteq X_2 \times U_2$ be the set of state-input pairs computed in Alg. 1. One can show that $D^{-1}(U_2)$ is the *maximal fixed point* of the map $G : \wp(X_2) \to \wp(X_2)$ defined by

$$G(Y) := Z_2 \cap \mathrm{pre}(Y). \tag{24}$$

The maximal fixed point of (24) corresponds to the winning domain of (S_2, Σ_2) and it follows that the synthesis problem (S_2, Σ_2) is solvable if and only if $X_{2,0} \subseteq D^{-1}(U_2)$.

Suppose that $X_{2,0} \subseteq D^{-1}(U_2)$ holds, then we obtain a controller $C = (\{q\}, \{q\}, X_2, X_2, U_2, F_c, H_c)$ that solves (S_2, Σ_2) by

$$\begin{aligned} H_c(q, x_2) &= \begin{cases} D(x_2) \times \{x_2\} & \text{if } x_2 \in D^{-1}(U_2) \\ U_2 \times \{x_2\} & \text{otherwise} \end{cases} \\ F_c(q, x_2) &= \begin{cases} \{q\} & \text{if } x_2 \in D^{-1}(U_2) \\ \varnothing & \text{otherwise.} \end{cases} \end{aligned} \tag{25}$$

The refined controller $C \circ Q$ that is feedback composed with S_1 is implemented in `SCOTS` in the `StaticController` class. The details are explained in Section 10.

7.2. **Reachability.** Let Σ_2 be a reachability specification associated with $Z_2 \subseteq \bar{X}_2$. We use Alg. 2, which is implemented in the function `solve_reachability_game` in the file `GameSolver.hh`, to synthesize a controller that solves the control problem (S_2, Σ_2). The algorithm is a variant of Dijkstra's shortest path algorithm for hyper-graphs taken from [6]. It runs in $O(m)$ time, where m is the number of transitions, i.e., the number of elements in $X_2 \times U_2 \times X_2$ that satisfy $x'_2 \in F_2(x_2, u)$.

Let D be the output of Alg. 2. The set $D^{-1}(U_2)$ is the *minimal fixed point* of the map $G : \wp(X_2) \to \wp(X_2)$ defined by

$$G(Y) := Z_2 \cup \mathrm{pre}(Y). \tag{26}$$

Algorithm 2 Controller synthesis for reachability specs associated with Z_2

Input: Z_2, $S_2 = (X_2, U_2, F_2)$, $u_0 \in U_2$
Require: $Z_2 \subseteq \bar{X}_2$
1: $Q := Z_2$ ▷ FIFO queue
2: $V := \infty$ ▷ value function
3: $M := 0$ ▷ intermediate values
4: $E := \varnothing$ ▷ bookkeeping of processed states
5: **for all** $x_2 \in \bar{X}_2$ **do**
6: $\quad D(x_2) := \varnothing$ ▷ keep track of optimal input
7: $\quad$ **if** $x_2 \in Z_2$ **then**
8: $\quad\quad V(x_2) := 0$ ▷ cost at target are zero
9: $\quad\quad D(x_2) := \{u_0\}$
10: **while** $Q \neq \varnothing$ **do**
11: $\quad x_2' :\in Q$ ▷ remove oldest element
12: $\quad Q := Q \setminus \{x_2'\}$
13: $\quad E := E \cup \{x_2'\}$
14: $\quad$ **for all** $(x_2, u) \in F_2^{-1}(x_2')$ **do**
15: $\quad\quad M(x_2, u) := \max\{M(x_2, u), V(x_2')\}$
16: $\quad\quad$ **if** $F_2(x_2, u) \subseteq E$ and $V(x_2) > 1 + M(x_2, u)$ **then** ▷ if u leads to better cost update input
17: $\quad\quad\quad V(x_2) := 1 + M(x_2, u)$
18: $\quad\quad\quad Q := Q \cup \{x_2\}$
19: $\quad\quad\quad D(x_2) := \{u\}$
Output: D, V

The minimal fixed point of (26) corresponds to the winning domain of (S_2, Σ_2) and it follows that the synthesis problem (S_2, Σ_2) is solvable if and only if $X_{2,0} \subseteq D^{-1}(U_2)$.

Given that $X_{2,0} \subseteq D^{-1}(U_2)$ holds, a controller $C = (\{q\}, \{q\}, X_2, X_2, U_2, F_c, H_c)$ that solves (S_2, Σ_2) is identical to the controller that solves the invariance problem, i.e., F_c and H_c are given by (25).

Again, the refined controller $C \circ Q$ that is feedback composed with S_1 is implemented in SCOTS in the `StaticController` class. The details are explained in Section 10.

7.3. **Reach-Avoid.** Let Σ_2 be a reach-avoid specification associated with A_2, $Z_2 \subseteq \bar{X}_2$. SCOTS provides two alternatives to synthesize controllers to enforce reach-avoid specifications. In the first method, the avoid set A_2 is accounted for in the transition function, i.e.,

$$\forall_{x_2 \in A_2} : \quad F_2(x_2, u) := \varnothing. \tag{27}$$

In this way we can reduce a reach-avoid problem to a reachability problem. Given that the transition function satisfies (27), any controller that solves the control problem (S_2, Σ_2'), where Σ_2' is the reachability specification associated with Z_2, also solves the control problem (S_2, Σ_2). In order to enforce the equation (27) in the computation of the transition function F_2 the user can optionally supply the avoid set to the function `AbstractionGB::compute`, see Section 8.

For the second method, the computation of the transition function is left unchanged, but Alg. 2 is modified to account for the avoid set. To this end the condition in line 16 in Alg. 2 is modified to

$$F_2(x_2, u) \subseteq E \quad \text{and} \quad V(x_2) > 1 + M(x_2, u) \quad \text{and} \quad x_2 \notin A_2. \tag{28}$$

Again the user can optionally supply the avoid set to the function `solve_reachability_game`, see Section 9.

Part 2. USAGE

Work in progress...

8. Computation of Symbolic Models

Let S_1 be a simple system that represents the τ-sampled behavior of continuous-time system (1) as defined in (2). In order to compute the transition function F_2 of a symbolic model $S_2 = (X_2, X_{2,0}, U_2, F_2)$ of S_1 the following ingredients are needed.

(1) The solution $\varphi(\tau, x, u)$ of the IVP $\dot{\xi} = f(\xi, u)$, $\xi(0) = x$ at time τ, which is provided in terms of a lambda function with signature

```
[](state_type& x, const input_type& u) -> void {}
```

The state solution of the IVP at time τ, i.e., $\varphi(\tau, x, u)$ is expected to be stored in the variable `x`. The `state_type` and `input_type` are aliases for `std::array<double,dim>` in the appropriate dimension.

(2) a growth bound $\beta(r, u)$, which is provided by a lambda function with signature

```
[](state_type& r, const state_type& c, const input_type& u) -> void {}
```

Again, the value of the growth bound $r' = \beta(r, u)$ is assumed to be stored in the variable `r`. In the implementation, the growth bound is allowed to depend the center of the cell `c`. See [4] for details.

(3) the set of real quantizer symbols $\bar{X}_2$ as defined in (7), which are provided in terms of instantiations of the class with constructor

```
template<class grid_point_t>
UniformGrid(const int dim,
            const grid_point_t& lb,
            const grid_point_t& ub,
            const grid_point_t& eta)
```

where the variables `dim`, `lb`, `ub` and `eta` represent the state space dimension, the lower and upper bound of the hyper-interval confining the set $\bar{X}_2$, and the grid parameter, respectively.

(4) the input alphabet U_2, which is again provided by an instance of a `UniformGrid`;

(5) optionally, it is possible to provide a measurement error bound using the member function of

```
scots::AbstractionGB::set_measurement_error_bound(const state_type& z)
```

Subsequently, we demonstrate the usage of SCOTS to compute a symbolic model of the τ-sampled aircraft dynamics used in [1, Sec. IX.B]. The system consists of three states x_1, x_2, x_3, which respectively correspond to the velocity, the flight path angle and the altitude of the aircraft. The input alphabet is given by $U = \left[0, 160 \cdot 10^3\right] \times [0°, 10°]$ and represents the thrust of the engines (in Newton) and the angle of attack. Additionally, we use the disturbance vectors $z = (0.0125, 0.0025°, 0.05)^\top$ in (11) and $w = (0.108, 0.002, 0)^\top$ in (1) to model possible measurement errors, respectively, input disturbances.

First, we define the aliases for the `state_type` and `input_type` by

```
/* state space dim */
const int state_dim=3;
/* input space dim */
const int input_dim=2;
using state_type = std::array<double,state_dim>;
using input_type = std::array<double,input_dim>;
```

Afterwards, we define the lambda for the solution of the IVP for a sampling time of $\tau = 0.25$ by

```
auto aircraft_post = [] (state_type &x, const input_type &u) {
  /* the ode describing the aircraft */
  auto rhs =[] (state_type& xx, const state_type &x, const input_type &u) {
    double mg = 60000.0*9.81;
```

```
    double mi = 1.0/60000;
    double c=(1.25+4.2*u[1]);
    xx[0] = mi*(u[0]*std::cos(u[1])-(2.7+3.08*c*c)*x[0]*x[0]-mg*std::sin(x[1]));
    xx[1] = (mi/x[0])*(u[0]*std::sin(u[1])+68.6*c*x[0]*x[0]-mg*std::cos(x[1]));
    xx[2] = x[0]*std::sin(x[1]);
  };
  /* use 10 intermediate steps */
  scots::runge_kutta_fixed4(rhs,x,u,state_dim,tau,10);
};
```

where we use the fixed step-size ODE solver `scots::runge_kutta_fixed4` that ships with SCOTS. An implementation, where we use more advanced ODE solvers like the adaptive step-size solvers with error control implemented in `boost` can be found in `./examples/aircraft_boost`.

We continue with the definition of the growth bound, for which we determine the matrix $L(u)$ in (13). First, we compute an a priori enclosure K' of (1) with respect to $\tau = 0.25$, $U' = U$ and the hyper-interval

$$K = [58, 83] \times [-3^\circ, 0] \times [0, 56]$$

that confines the set of real quantizer symbols. We use interval arithmetic based ODE solver `vnodelp` and obtain

$$K' = [57.55, 83.23] \times [-4.29^\circ, 1.22^\circ] \times [-1.38, 56.28].$$

The implementation of the computation of the a priori enclosure can be found in the example directory `./examples/aircraft/helper/a_priori_enclosure`. Given the a priori enclosure, we maximize the partial derivatives of $f(\cdot, u)$ over the set K' to obtain the matrix $L(u)$. The solution of the linear ODE

$$\dot{r} = L(u)r + w$$

provides the growth bound (14), which we define by

```
auto radius_post = [] (state_type &r, const state_type &, const input_type &u) {
  /* the ode for the growth bound */
  auto rhs =[&] (state_type& rr, const state_type &r, const input_type &) {
    /* lipschitz matrix */
    double L[3][2];
    L[0][0]=-0.001919*(2.7+3.08*(1.25+4.2*u[1])*(1.25+4.2*u[1]));
    L[0][1]=9.81;
    L[1][0]=0.002933+0.004802*u[1];
    L[1][1]=0.00361225;
    L[2][0]=0.07483;
    L[2][1]=83.22;
    /* to account for input disturbances */
    state_type w={{0.108,0.002,0}};
    rr[0] = L[0][0]*r[0]+L[0][1]*r[1]+w[0];
    rr[1] = L[1][0]*r[0]+L[1][1]*r[1]+w[1];
    rr[2] = L[2][0]*r[0]+L[2][1]*r[1]+w[2];
  };
  /* use 10 intermediate steps */
  scots::runge_kutta_fixed4(rhs,r,u,state_dim,tau,10);
};
```

The real quantizer symbols $\bar{X}_2$ and the input alphabet U_2 are introduced as instances of the class `UniformGrid` by

```
  state_type s_lb={{58,-3*M_PI/180,0}};
  state_type s_ub={{83,0,56}};
  state_type s_eta={{25.0/362,3*M_PI/180/66,56.0/334}};
  scots::UniformGrid ss(state_dim,s_lb,s_ub,s_eta);
```

```
input_type i_lb={{0,0}};
input_type i_ub={{32000,8*M_PI/180}};
input_type i_eta={{32000,8.0/9.0*M_PI/180}};
scots::UniformGrid is(input_dim,i_lb,i_ub,i_eta);
```

The grid parameter s_eta for the real quantizer symbols $\bar{X}_2$ has been determined according to the optimization procedure in [4].

For the computation of the symbolic transition function F_2 we instantiate the class AbstractionGB and set the measurement error bound z

```
scots::AbstractionGB<state_type,input_type> abs(ss,is);
state_type z={{0.0125,0.0025/180*M_PI,0.05}};
abs.set_measurement_error_bound(z);
```

The computation of F_2 itself is invoked by

```
abs.compute(tf,aircraft_post,radius_post);
```

Note that, due to the numerical errors in the computation of the solution of the IVP $\varphi(\tau, x, u)$ it is possible that the computed transition function F_2 does not satisfies the requirements in Sectin 6.2. Similarly, the computation of the growth bound $\beta(r, u)$ might be erroneous. Therefore, it is crucial to pick adequate ODE solver parameters to obtain high confidence in the correctness of the computation. We list some runtimes of the computation of F_2 for two different solvers for various parameters in Table 8. The implementations can be found in `./exampels/aircraft` and `./examples/aircraft_boost`.

adaptive step size `runge_kutta_dopri5` `abs_tol/rel_tol`			fixed step size `runge_kutta_4` # intermediate steps	
$10^{-10}/10^{-10}$	$10^{-12}/10^{-12}$	$10^{-16}/10^{-16}$	5	10
252 sec	370 sec	1036 sec	233 sec	335 sec

TABLE 1. Runtimes to compute a symbolic transition function of the aircraft dynamics with varying ODE solvers and solver parameters. All computations resulted in an identical transition function with $5.86 \cdot 10^9$ transitions.

9. Controller Synthesis

10. Closed Loop Simulation

The function `StaticController::control` takes as input a state $x_1 \in X_1$ and rounds it to the nearest grid point in $\eta\mathbb{Z}$ in order to determine a symbolic state $x_2 \in Q(x_1)$. If there does not exist a valid input $u \in U_2$ associated with x_2, i.e., $x_2 \notin \pi_{X_2}(D)$, then the $F_c(q, x_2) = \varnothing$ and function returns the error message

```
scots::StaticController::control: no progress possible at state <values>
```

Otherwise, `StaticController::control` returns the set of valid control inputs.

11. Writing To and Reading From Hard Disk

11.1. Directly.

(1) Atomic Propositions
(2) StaticController
(3) UniformGrid
(4) TransitionFunction

11.2. Using the CUDD Library.

12. MATLAB Interface

13. Customized Synthesis Algorithms

References

[1] G. Reißig, A. Weber, and M. Rungger. “Feedback Refinement Relations for the Synthesis of Symbolic Controllers”. In: *IEEE Transactions on Automatic Control* (2015). DOI: 10.1109/TAC.2016.2593947.

[2] R. Ehlers and V. Raman. “Slugs: extensible GR(1) synthesis”. In: *International Conference on Computer Aided Verification*. Springer. 2016, pp. 333–339.

[3] M. Rungger and M. Zamani. “SCOTS: A Tool for the Synthesis of Symbolic Controllers”. In: *HSCC*. ACM, 2016.

[4] A. Weber, M. Rungger, and G. Reissig. “Optimized State Space Grids for Abstractions”. In: *IEEE Transactions on Automatic Control* (2017). DOI: 10.1109/TAC.2016.2642794.

[5] R. Bloem et al. “Synthesis of reactive (1) designs”. In: *Journal of Computer and System Sciences* 78.3 (2012), pp. 911–938.

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