Temporal Logic Control of Switched Affine Systems with an Application in Fuel Balancing

Petter Nilsson[†], Necmiye Ozay[*], Ufuk Topcu[*] and Richard M. Murray[*]

[†]: Royal Institute of Technology, email pettni@kth.se
[*]: California Institute of Technology, email {necmiye, utopcu, murray}@cds.caltech.edu

29 June 2012

Outline

Introduction

- Often hard to determine if the implementation of a controller will result in desired behavior.
- Would like to construct a controller that *guarantees* correctness with respect to some criteria.
- Need framework to connect 'high-level' specification language with 'low-level' dynamics.
- Extension to handle switched systems.

Problem statement

We consider a switched control system where each of the k different switching modes is an affine time-discrete control system.

$$\begin{aligned} x(t+1) &= A_{\sigma(t)}x(t) + B_{\sigma(t)}u(t) + E_{\sigma(t)}d(t) + K_{\sigma(t)}, \\ u(t) &\in U_{\sigma(t)}(s(t)), \quad d(t) \in D_{\sigma(t)}, \quad \sigma(t) \in \{1, \ldots k\}. \end{aligned}$$

We assume that the sets U_k and D_k are convex with piecewise flat sides (i.e. polytopes).

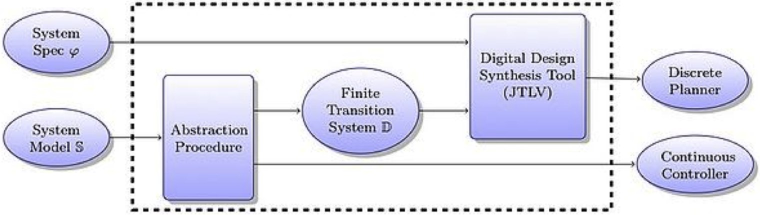
Problem

Given a control system on the form above together with a list of specifications on desired behavior, synthesize a controller that guarantees the fulfillment of the specifications for all switching functions $\sigma : \mathbb{N} \to \{1, \ldots k\}$.

Synthesis strategy: Overview

For non-switched systems, as presented by Wongpiromsarn et. al. [3].

- Use Linear Temporal Logic (LTL) for specifications.
- Problem: The state space is continuos, infinite number of states.
- Approach: Lift the problem to the discrete level to enable planning.
  - Partition the continuous state space into finitely many *discrete states*.
  - Treat problem as a two-player game against the environment and find a discrete plan using software such as JTLV [1].
  - Implement discrete plan using a continuous controller.
- Software toolbox to implement synthesis method: TuLiP [2].

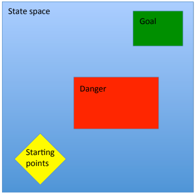


Linear temporal logic

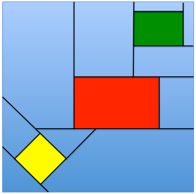
- Extension of basic logic operators ($\wedge$, $\vee$, $\neg$, $\Rightarrow$) to include the temporal operators *next* ($\bigcirc$), *until* ($\mathcal{U}$), *always* ($\Box$) and *eventually* ($\Diamond$).
- These can be combined to write powerful specifications about the behavior of a system.
- Examples:
  - $\Box$ stay away from danger (safety).
  - $\Diamond$ reach target (goal).
  - $\Diamond\Box$ stay close to target (convergence).

State space partitioning

- Given: Continuous state space and specifications. The specifications relate to given parts of the state space.
- Want: A partition of the state space into discrete states.
  - To guarantee correctness, the partition has to be *proposition preserving*.
  - Also want to establish as many *reachability relations* between the discrete states as possible to make a planner synthesis possible.
- Step 1: Create a (convex) proposition preserving partition.



Initial situation.



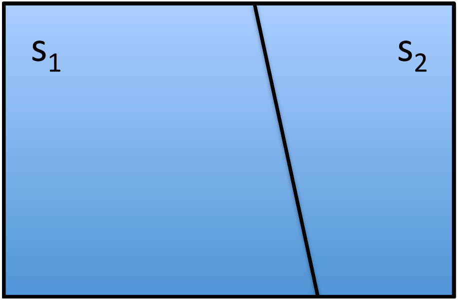
Proposition preservation partition.

Partitioning algorithm

- Step 2: Further discretization based on reachability.
- Procedure:

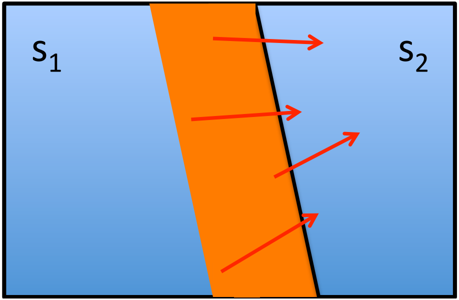
Partitioning algorithm

- Step 2: Further discretization based on reachability.
- Procedure:
  - Pick two discrete states s_1 and s_2.



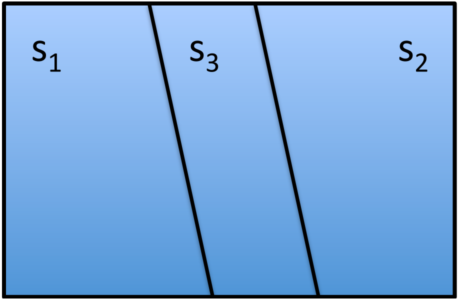
Partitioning algorithm

- Step 2: Further discretization based on reachability.
- Procedure:
  - Pick two discrete states s_1 and s_2.
  - Determine the part $s_3 \subset s_1$ from where we can get to s_2.



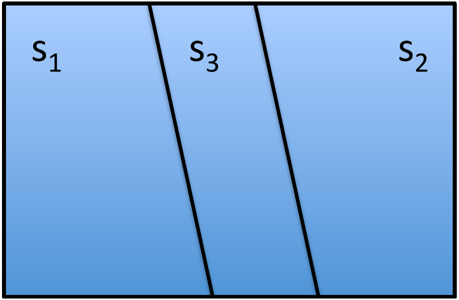
Partitioning algorithm

- Step 2: Further discretization based on reachability.
- Procedure:
  - Pick two discrete states s_1 and s_2.
  - Determine the part $s_3 \subset s_1$ from where we can get to s_2.
  - Divide s_1 into s_3 and $s_1 \setminus s_3 \to s_1$.



Partitioning algorithm

- Step 2: Further discretization based on reachability.
- Procedure:
  - Pick two discrete states s_1 and s_2.
  - Determine the part $s_3 \subset s_1$ from where we can get to s_2.
  - Divide s_1 into s_3 and $s_1 \setminus s_3 \to s_1$.
  - Iterate unit some termination criteria (minimal cell volume, "enough transitions") is met.

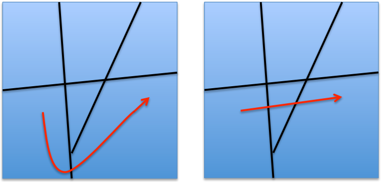


Algorithm enhancement

Definition (Reachability)

We say a discrete state s_2 is *reachable* (under the switched state κ) from the discrete state s_1 in N steps if there for every continuous state $\varsigma_1 \in s_1$ exists a control sequence $u(0), \ldots u(N-1)$ that takes the plant to a state $\varsigma_2 \in s_2$ when $\sigma(t) \equiv k$. We require that $u(t) \in U_\kappa(x(t))$ and $x(t) \in P$ for all steps, where P is the *parent proposition cell*.

- Looser, but yet 'safe', definition of reachability.
- More possible transitions, larger optimization set.

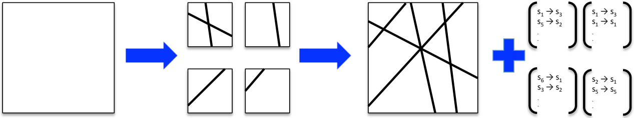


Extension to switched systems

To enable planning for a switched system, the following procedure is used.

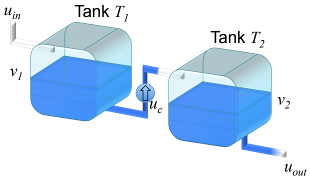
1. Run the reachability partitioning algorithm *for each* dynamical mode. This results in k different partitions $\mathcal{P}_i$ of the state space.
2. *Merge* all the k partitions into one single partition $\mathcal{P}$.
3. Search for reachability relations between cells in $\mathcal{P}$ *for each* dynamical mode.

Results in one single partition together with k lists (one for each dynamical mode) of possible transitions between the discrete states in the partition. This information is then plugged into a synthesize algorithm.



Fuel tanks: Set-up

A model of two airplane fuel tanks T_1 and T_2 that can operate in two different modes, *normal mode* and *aerial refueling mode*. The state variables are the tank volumes v_1 and v_2.



Dynamics:

$$\begin{bmatrix} v_1(t+1) \\ v_2(t+1) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} u_c(t) + \begin{bmatrix} u_{in}(t) \\ -1 \end{bmatrix}$$

- 1 fuel unit is taken from tank T_2 in each time step.
- During aerial refueling mode 3 fuel units are added to tank T_1 in each time step, i.e. $u_{in}(t) \in \{0, 3\}$.
- We control a pump $u_c(t)$ that moves fuel from tank T_1 to tank T_2.

Fuel tanks: Specifications

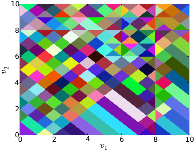
- The capacity of both tanks is 10 fuel units. This gives the continuous state space $\{(v_1, v_2) \mid 0 \leq v_1, v_2 \leq 10\}$.
- Assumptions:
  - When $v_1 + v_2 \leq 2$, refueling will start (switch from $u_{in} = 0$ to $u_{in} = 3$).
  - When $v_2 \geq 8$, refueling will stop (switch from $ui_{in} = 3$ to $u_{in} = 0$).
- Requirements:
  - Always satisfy $|v_1 - v_2| \leq 2$.
  - Always eventually require that $|v_1 - v_2| \leq 1$.
- The assumptions and requirements can be written in LTL.

Fuel tanks: Specifications

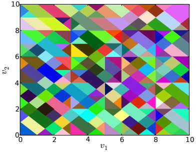
- The capacity of both tanks is 10 fuel units. This gives the continuous state space $\{(v_1, v_2) \mid 0 \leq v_1, v_2 \leq 10\}$.
- Assumptions:
  - When $v_1 + v_2 \leq 2$, refueling will start (switch from $u_{in} = 0$ to $u_{in} = 3$).
  - When $v_2 \geq 8$, refueling will stop (switch from $ui_{in} = 3$ to $u_{in} = 0$).
- Requirements:
  - Always satisfy $|v_1 - v_2| \leq 2$.
  - Always eventually require that $|v_1 - v_2| \leq 1$.
- The assumptions and requirements can be written in LTL.
- Next step: discretize the state space.

Fuel tanks: Discretization

Step 1: Run the discretization algorithm for both dynamical modes.

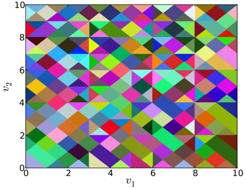


Partition for normal mode.



Partition for refuel mode.

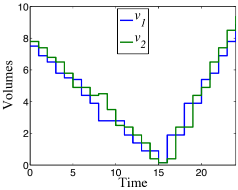
Step 2: Merge the partitions and find transitions.



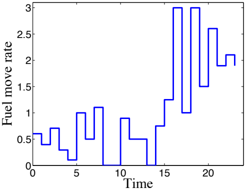
Final partition.

Fuel tanks: Simulation

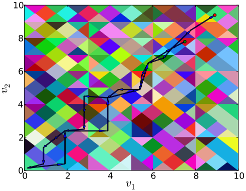
We implement the discrete plan using a continuous controller. Simulation of 25 time steps. Switches are controlled by the environment.



Volumes in the tanks.



Fuel move rate.



Plant trajectory.

Conclusions

- Method only feasible for low-dimensional systems due to computational complexity.

Future work:

- Revise discretization algorithm.
    - Algorithm termination criteria.
    - Taking specifications and/or information about switches into account.

References

- Amir Pnueli, Yaniv Sa'ar, and Lenore D. Zuck.
  JTLV: A framework for developing verification algorithms.
  In *22nd International Conference on Computer Aided Verification*, 2010.

- T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. Murray.
  TuLiP: a software toolbox for receding horizon temporal logic planning.
  In *International Conference on Hybrid Systems: Computation and Control*, 2011.

- Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray.
  Receding horizon temporal logic planning.
  IEEE Transactions on Automatic Control (submitted), 2010.