TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning

Nok Wongpiromsarn
Singapore-MIT Alliance for Research and Technology

Richard M. Murray and Ufuk Topcu
California Institute of Technology

EECI, 26 April 2012

Outline

• Key Features of TuLiP
  • Embedded control software synthesis
  • Receding horizon temporal logic planning
• Computer Lab
TuLiP for receding horizon temporal logic planning

Alice’s navigation stack

Mission Planner

Traffic Planner

Path Planner

Vehicle Actuation

Different views

“long-horizon specification”

“short-horizon specification”

continuous dynamics & constraints

Multi-scale models

\[ \mathcal{W}_0 \prec \ldots \prec \mathcal{W}_{L-1} \prec \mathcal{W}_L \]

\[
\begin{align*}
\min & \int_{t_0}^{T} L(x, u) dt \\
\text{s.t.} & \quad \dot{x} = f(x, u) \\
& \quad g(x, u) \leq 0
\end{align*}
\]

Hierarchical control architecture

Inputs:

• Models for evolution of discrete/continuous system states
• Assumptions on (discrete) environment state

Outputs:

• “Strategy” to be implemented in each layer
Digital design synthesis (middle layer)

• TuLiP interfaces to a game solver in JTLV to solve discrete, two-player, GR[1] games

\[
\varphi = (\psi_{\text{init}}^e \land \Box \psi_s^e \land \bigwedge_{i \in I_f} \Diamond \psi_{f,i}^e) \implies (\psi_{\text{init}}^s \land \Box \psi_s^s \land \bigwedge_{i \in I_g} \Diamond \psi_{g,i}^s)
\]

JTLV:
Java-based framework for developing formal verification algorithms

• prob = `generateJTLVInput`(env_vars, sys_disc_vars, spec, disc_props, disc_dynamics, smv_file, spc_file)
• realizability = `checkRealizability`(smv_file, spc_file, aut_file, heap_size)
• realizability = `computeStrategy`(smv_file, spc_file, aut_file, heap_size)
• aut = `Automaton`(aut_file)
**System model:** Robot can move to the cells which share a face with the current cell.

**Desired Properties**
- Visit the blue cell infinitely often
- Eventually go to the red cell when a PARK signal is received

**Assumption**
- Infinitely often, PARK signal is not received

\[ \varphi = \square \Diamond (\neg park) \implies (\square \Diamond (s \in C_5) \land \square (park \implies \Diamond (s \in C_0))) \]

The specification is not a GR[1] formula
- Introduce an auxiliary variable \( X_0reach \) that starts with True
- \( \square (\Diamond X_0reach = (s \in C_0 \lor X_0reach) \land \neg park) \)
- \( \square \Diamond X_0reach \)

**Link to the tutorial:** [http://goo.gl/uOH8Q](http://goo.gl/uOH8Q)
\( \varphi = \Box \Diamond (\neg \text{park}) \quad \implies \quad (\Box \Diamond (s \in C_5) \land \\
\Box (\text{park} \implies \Diamond (s \in C_0))) \)

\( \Box (\bigcirc X0reach = ((s \in C_0 \lor X0reach) \land \neg \text{park})) \)

\( \Box \Diamond X0reach \)
Aut file (partial)

Specification is realizable...

===== Building an implementation =====

State 0 with rank 0 -> <park:1, cellID:0, X0reach:0>
  With successors: 1, 2
State 1 with rank 0 -> <park:0, cellID:1, X0reach:0>
  With successors: 3, 4
State 2 with rank 0 -> <park:1, cellID:1, X0reach:0>
  With successors: 3, 4
State 3 with rank 0 -> <park:0, cellID:4, X0reach:0>
  With successors: 5, 4
State 4 with rank 0 -> <park:1, cellID:4, X0reach:0>
  With successors: 5, 4
State 5 with rank 0 -> <park:0, cellID:5, X0reach:0>
  With successors: 6, 7
State 6 with rank 1 -> <park:0, cellID:4, X0reach:0>
  With successors: 6, 7
State 7 with rank 1 -> <park:1, cellID:4, X0reach:0>
  With successors: 8, 9
State 8 with rank 1 -> <park:0, cellID:1, X0reach:0>
  With successors: 10, 11
State 9 with rank 1 -> <park:1, cellID:1, X0reach:0>
  With successors: 10, 11
State 10 with rank 1 -> <park:0, cellID:0, X0reach:0>
  With successors: 12, 13
State 11 with rank 1 -> <park:1, cellID:0, X0reach:0>
  With successors: 10, 11
State 12 with rank 1 -> <park:0, cellID:0, X0reach:1>
  With successors: 22, 23
State 13 with rank 1 -> <park:1, cellID:0, X0reach:1>

Sample simulation from grsim with inputs:
  • park signal
  • initial value of X0reach

\[ \varphi = \square\diamond (\neg park) \implies (\square\diamond (s \in C_5) \land \square (park \implies \diamond (s \in C_0))) \land (\diamond X0reach = ((s \in C_0 \lor X0reach) \land \neg park)) \land \Box \diamond X0reach \]
Interface the discrete planner and continuous controller

Use the finite-state abstraction procedure to write the problem in terms of the inputs to digital design synthesis.

Main steps:
- Partition the continuous state space
- Propositions shall preserve their meanings
- Establish transitions between the cells in the partition
- Call the routines for digital design synthesis

Continuous dynamics:
\[
\begin{align*}
    s(t+1) &= As(t) + Bu(t) + Ed(t) \\
    u(t) &\in U \\
    d(t) &\in D \\
    s &\in \mathbb{R}^n, U \subseteq \mathbb{R}^m, D \subseteq \mathbb{R}^p
\end{align*}
\]
Workflow: discrete planner + continuous controller

- Generate a proposition preserving partition of the continuous state space
  - cont_partition = prop2part2(state_space, cont_props)
- Discretize the continuous state space based on the evolution of the continuous state
  - disc_dynamics = discretizeM(cont_partition, ssys, N=10)
- Digital design synthesis: generateJTLVInput, checkRealizability, computeStrategy, Automaton,...
Example: incorporating continuous dynamics

System model: \( \dot{x} = u_x, \dot{y} = u_y \) where \( u_x, u_y \in [-1, 1] \)

Desired Properties
- Visit the blue cell infinitely often
- Eventually go to the red cell when a PARK signal is received

Assumption
- Infinitely often, PARK signal is not received

\[
\varphi = \square \Diamond (\neg \text{park}) \implies (\square \Diamond (s \in C_5) \land \\
\square (\text{park} \implies \Diamond (s \in C_0)))
\]

The specification is not a GR[1] formula
- Introduce an auxiliary variable \( X0reach \) that starts with True
- \( \square (\bigcirc X0reach = ((s \in C_0 \lor X0reach) \land \neg \text{park})) \)
- \( \square \Diamond X0reach \)

Link to the tutorial: http://goo.gl/6FsM0
Defining a synthesis problem: SynthesisProb class

Combine the three steps of digital design synthesis:

\[
\begin{array}{ccc}
\text{generateJTLVInput} & \text{computeStrategy} & \text{synthesize} \\
\text{Automaton} & & \text{SynthesisProb}
\end{array}
\]

Useful methods:

- **checkRealizability**\(\text{(heap_size='\!-Xmx128m', pick_sys_init=True, verbose=0)}\): checks whether the problem is realizable
- **getCounterExamples**\(\text{(recompute=False, heap_size='\!-Xmx128m', pick_sys_init=True, verbose=0)}\) returns the set of initial states from which the system cannot satisfy the spec
- **synthesizePlannerAut**\(\text{(heap_size='\!-Xmx128m', priority_kind=3, init_option=1, verbose=0)}\) synthesizes the planner that ensures system correctness

Fields of SynthesisProb:
- env_vars
- sys_vars
- spec
- disc_cont_var
- disc_dynamics

Link to an example: [http://goo.gl/fNZxS](http://goo.gl/fNZxS)
Receding Horizon Framework for LTL Specifications

Idea: Reduce the synthesis problem to a set of smaller problems of short horizon

- Consider a specification of the form
  \[ \varphi = (\psi_{\text{init}} \land \Box \psi_e \land \bigwedge_{i \in I_f} \Box \psi_{f,i}) \implies \bigwedge_{i \in I_s} \Box \psi_{s,i} \land \Box \Diamond \psi_g \]

- Organize cells into a partially ordered set \( P = (\{W_j\}, \preceq_{\psi_g}) \) where \( W_0 \) is the set of “goal states,” i.e., all cells in \( W_0 \) satisfy \( \psi_g \)

- Assume that for each \( j \), there exist a proposition \( \Phi \) and a mapping \( F \) such that the following short-horizon specification is realizable
  \[ \Psi_j = \left( (\varsigma \in W_j) \land \Phi \land \Box \psi_e \land \bigwedge_{k \in I_f} \Box \Diamond \psi_{f,k} \right) \implies \left( \bigwedge_{k \in I_s} \Box \psi_{s,k} \land \Box \Diamond (\varsigma \in F(W_j)) \land \Box \Phi \right) \]

  - \( \Phi \) describes receding horizon invariants
  - \( F(W_j) \prec W_j, \forall j \neq 0 \) defines intermediate goal for starting in \( W_j \)
  - Partial order condition guarantees that we move closer to goal

\[ F(W_4) = W_2, F(W_3) = W_1, F(W_2) = W_0, F(W_1) = W_0, F(W_0) = W_0 \]
Key Elements

Original specification:

\[ \varphi = \left( \psi_{\text{init}} \land \Box \psi_e \land \bigwedge_{i \in I_f} \Box \Diamond \psi_{f,i} \right) \implies \left( \bigwedge_{i \in I_s} \Box \psi_{s,i} \land \bigwedge_{i \in I_g} \Box \Diamond \psi_{g,i} \right) \]

Short horizon specification:

\[ \Psi_j^i = \left( \left( \zeta \in \mathcal{W}_j^i \right) \land \Phi \land \Box \psi_e \land \bigwedge_{k \in I_f} \Box \Diamond \psi_{f,k} \right) \implies \left( \bigwedge_{k \in I_s} \Box \psi_{s,k} \land \Box \Diamond \left( \zeta \in \mathcal{F}^i(\mathcal{W}_j^i) \right) \land \Box \Phi \right) \]

- Partially ordered set \( P^i = (\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}, \preceq_{\psi_g,i}) \)
  - \( \mathcal{W}_0^i \cup \mathcal{W}_1^i \cup \ldots \cup \mathcal{W}_p^i = \mathcal{V} \)
  - \( \mathcal{W}_0^i \) is the set of “goal states,” i.e., all cells in \( \mathcal{W}_0^i \) satisfy \( \psi_{g,i} \)
  - \( \mathcal{W}_0^i \prec_{\psi_g,i} \mathcal{W}_j^i, \forall j \neq 0 \)
- Receding horizon invariant \( \Phi \)
  - \( \psi_{\text{init}} \implies \Phi \) is a tautology
- Mapping \( \mathcal{F}^i : \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\} \to \{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\} \)
  - \( \mathcal{F}^i(\mathcal{W}_j^i) \prec_{\psi_g,i} \mathcal{W}_j^i, \forall j \neq 0 \)
Receding Horizon Temporal Logic Planning Problem

**ShortHorizonProb**
- A class for defining a short horizon problem
- Useful methods
  - `computeLocalPhi()`: automatically compute \( \Phi \) that makes this short horizon problem realizable.

**RHTLPProb**
- A class for defining a receding horizon temporal logic planning problem
- Contains a collection of short-horizon problems
- Useful methods
  - `computePhi()`: automatically compute \( \Phi \) for this receding horizon temporal logic planning problem if one exists.
  - `validate()`: check whether all the sufficient conditions for doing receding horizon temporal logic planning are satisfied
Traffic rules
• No collision
• Stay in the travel lane unless there is an obstacle blocking the lane

Progress requirement
• Reach the end of the road

Assumptions
• Obstacle may not block a road
• Obstacle is detected before the vehicle gets too close to it
• Limited sensing range
• Obstacle does not disappear when the vehicle is in its vicinity

Link to an example: [http://goo.gl/oyHvg](http://goo.gl/oyHvg)
Computer exercise 1

Synthesize a reactive planner with the following specifications

Desired Properties
- Visit the blue cell \((C_8)\) infinitely often
- Eventually go to the green cell \((C_0)\) when a PARK signal is received
- Avoid an obstacle (red cell) which can be one of the \(C_1, C_4, C_7\) cells and can move arbitrarily

Assumption
- Infinitely often, PARK signal is not received
- The obstacle always moves to an adjacent cell

Constraints (or discrete dynamics)
- The robot can only move to an adjacent cell, i.e., a cell that shares an edge with the current cell
Computer exercise 2

Synthesize intersection logic for the car with the following specification

**Desired Properties**
- Eventually go to C₆
- If there is a car at one of the C₃, C₄, C₇ cells at initial state, need to wait until it disappears before going through the intersection
- Go through the intersection only when C₂ and C₅ are clear
- No collision with other cars

**Assumption**
- ?? (find a set of “non-trivial” assumptions that render the problem realizable)

**Constraint**
- The robot can only move forward to an adjacent cell, i.e., a cell that shares an edge with the current cell