TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning & Computer Lab 2

Nok Wongpiromsarn

Richard M. Murray **Ufuk Topcu**

EECI, 21 March 2013

Outline

- Key Features of TuLiP
  - Embedded control software synthesis
  - Receding horizon temporal logic planning
- Computer Lab

Problem Description

Problem: Given a plant model and an LTL specification φ, design a controller to ensure that any execution of the system satisfies φ

- The evolution of the system is described by differential/difference equations

$$\begin{aligned} s(t+1) &= As(t) + Bu(t) + Ed(t)) \\ u(t) &\in U \\ d(t) &\in D \end{aligned}$$

  where $s \in \mathbb{R}^n, U \subseteq \mathbb{R}^m, D \subseteq \mathbb{R}^p$

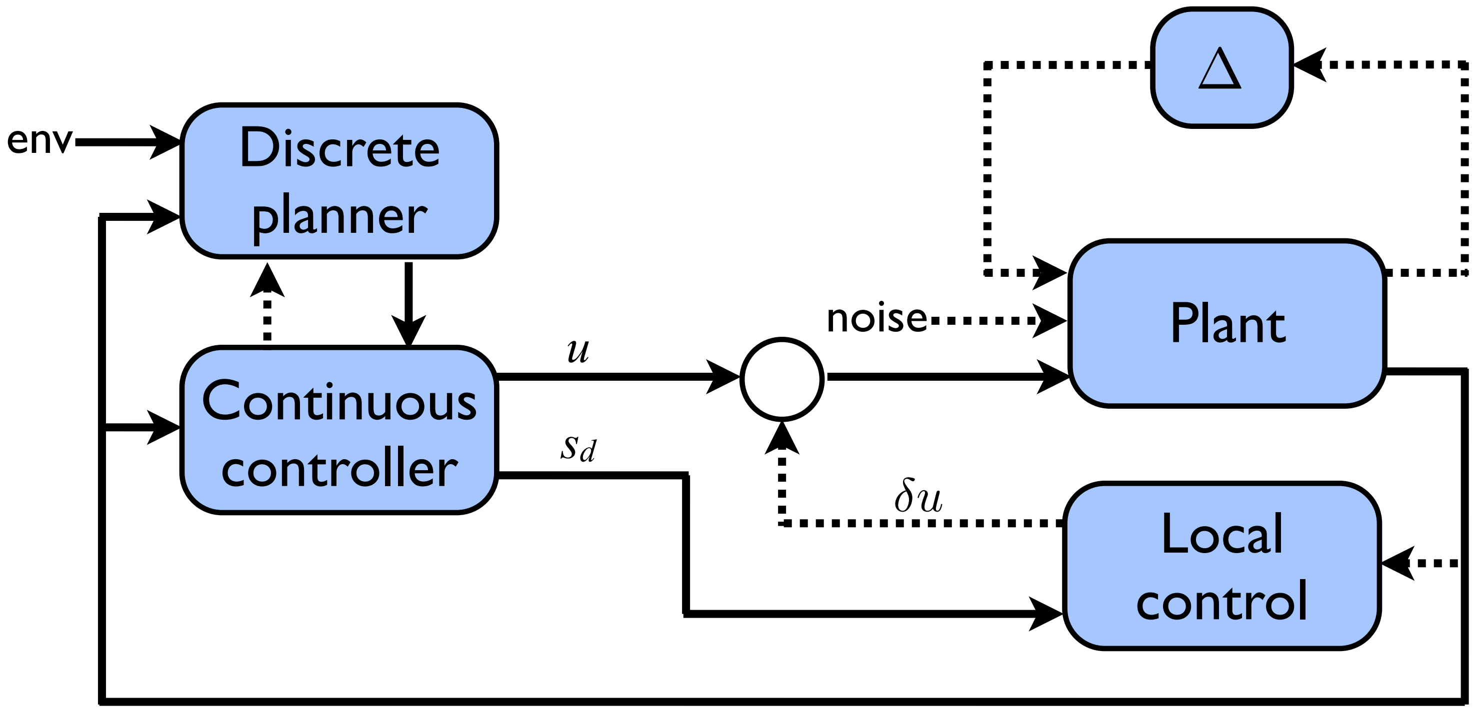
- φ must be satisfied regardless of the environment in which the system operates
- Assume that φ is of the form

$$\varphi = \Big(\underbrace{\psi^e_{init}}_{\text{assumptions on initial condition}} \wedge \underbrace{\Box \psi^e_s \wedge \bigwedge_{i \in I_f} \Box \Diamond \psi^e_{f,i}}_{\text{assumptions on environment}} \Big) \implies \underbrace{\Big(\psi^s_{init} \wedge \Box \psi^s_s \wedge \bigwedge_{i \in I_g} \Box \Diamond \psi^s_{g,i} \Big)}_{\text{desired behavior}}$$

Embedded Control Software Synthesis

Key elements to specify the problem

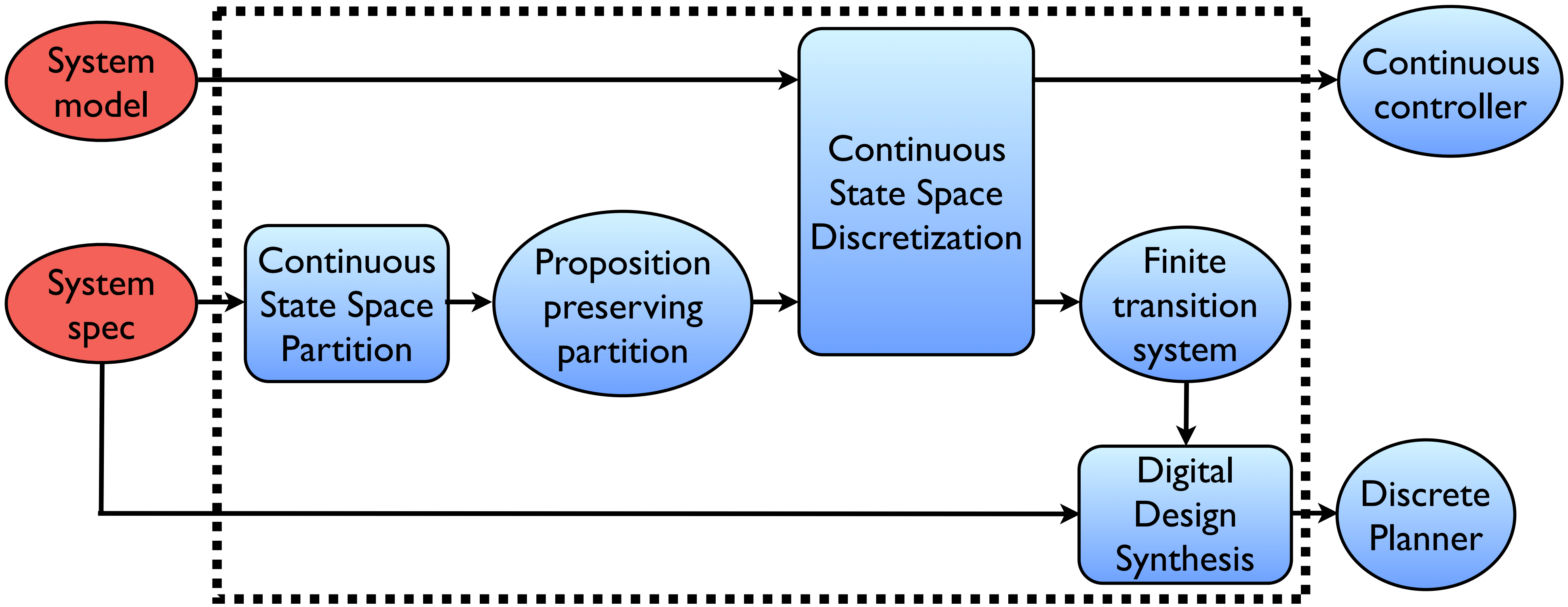
- discrete system state
- continuous system state
- (discrete) environment state
- specification



Hierarchical Approach

- Discrete planner computes the next cell to go to in order to satisfy φ
  - The synthesis algorithm considers all the possible behaviors of the environment
  - **Issue**: state explosion
- Continuous controller *simulates* the plan
  - Constrained optimal control problem
  - Continuous execution preserves the correctness of the plan

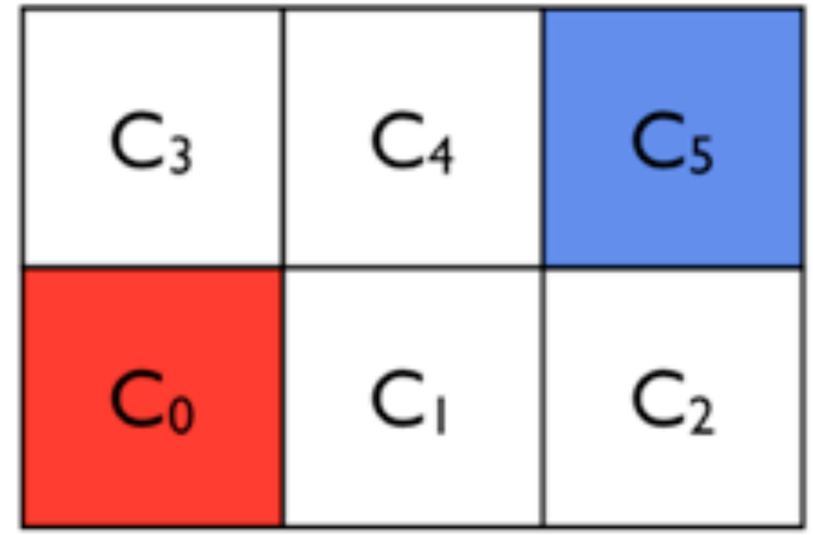
Main Steps



- 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
  - 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)

Example: robot_simple.py

Dynamics $\dot{x} = u_x, \dot{y} = u_y \text{ where } u_x, u_y \in [-1, 1]$



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

$$\varphi = \Box\Diamond(\neg park) \implies (\Box\Diamond(s \in C_5) \wedge \Box(park \implies \Diamond(s \in C_0)))$$

This spec is not a GR[1] formula

- Introduce an auxiliary variable *X0reach* that starts with True
- $\Box(\bigcirc X0reach = ((s \in C_0 \vee X0reach) \wedge \neg park))$
- $\Box\Diamond X0reach$

Manually Constructing disc_dynamics: robot_discrete_simple.py

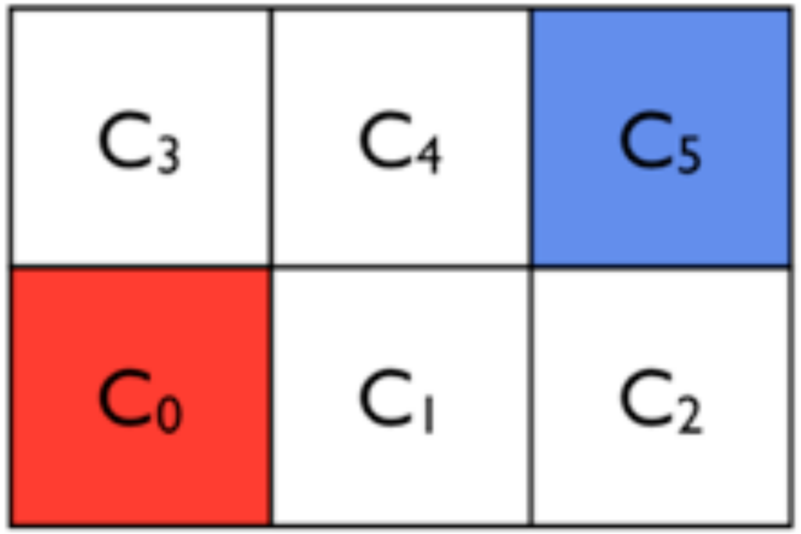
System Model: Robot can move to the cells that share a face with the current cell

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



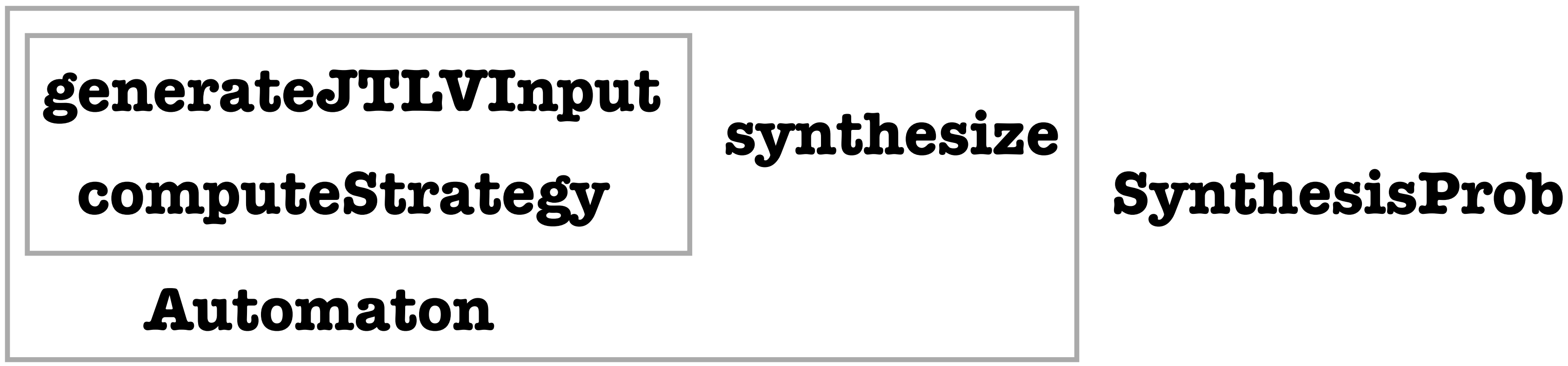
$$\varphi = \Box\Diamond(\neg park) \implies (\Box\Diamond(s \in C_5) \wedge \Box(park \implies \Diamond(s \in C_0)))$$

This spec is not a GR[1] formula

- Introduce an auxiliary variable $X0reach$ that starts with True
- $\Box(\bigcirc X0reach = (s \in C_0 \vee (X0reach \wedge \neg park)))$
- $\Box\Diamond X0reach$

Defining a synthesis problem: SynthesisProb class

Combine the three steps of digital design synthesis:



Fields of SynthesisProb:

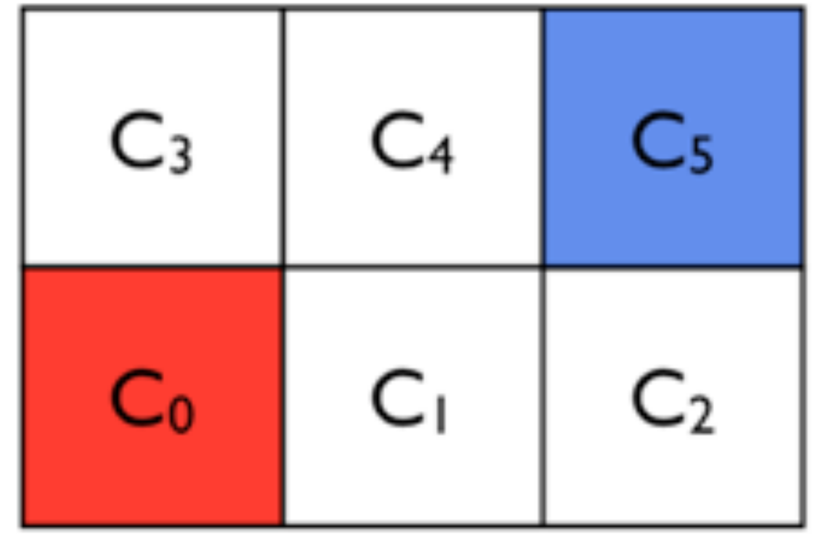
- *env_vars*
- *sys_vars*
- *spec*
- *disc_cont_var*
- *disc_dynamics*

Useful methods:

- ***checkRealizability***(*heap_size='-Xmx128m', pick_sys_init=True, verbose=0*):
  checks whether the problem is realizable
- ***getCounterExamples***(*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***(*heap_size='-Xmx128m', priority_kind=3, init_option=1, verbose=0*)
  synthesizes the planner that ensures system correctness

Example: robot_simple2.py

Dynamics $\dot{x} = u_x, \dot{y} = u_y$ 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

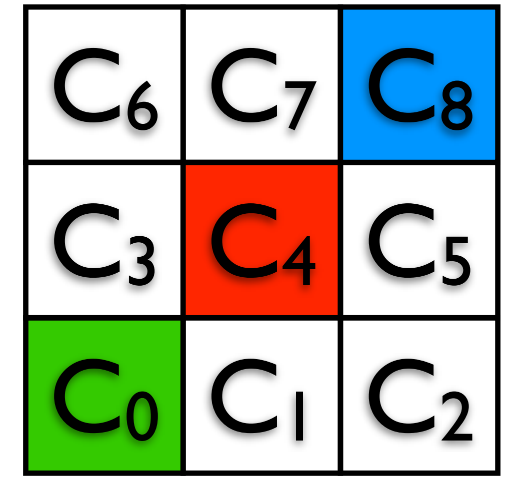
- Infinitely often, PARK signal is not received

$$\varphi = \Box\Diamond(\neg park) \implies (\Box\Diamond(s \in C_5) \wedge \Box(park \implies \Diamond(s \in C_0)))$$

This spec is not a GR[1] formula

- Introduce an auxiliary variable *X0reach* that starts with True
- $\Box(\bigcirc X0reach = ((s \in C_0 \vee X0reach) \wedge \neg park))$
- $\Box\Diamond X0reach$

Computer exercise 1



Synthesize a reactive planner with the following specifications

System variables: X0,...,X8 -- Xi = 1 if robot in Ci, Xi = 0 otherwise.
Environment variables: obs $\in$ {1,4,7}, park $\in$ {0,1}

Desired Properties

- Visit the blue cell (C_8) infinitely often
- Eventually go to the green cell (C_0) after 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

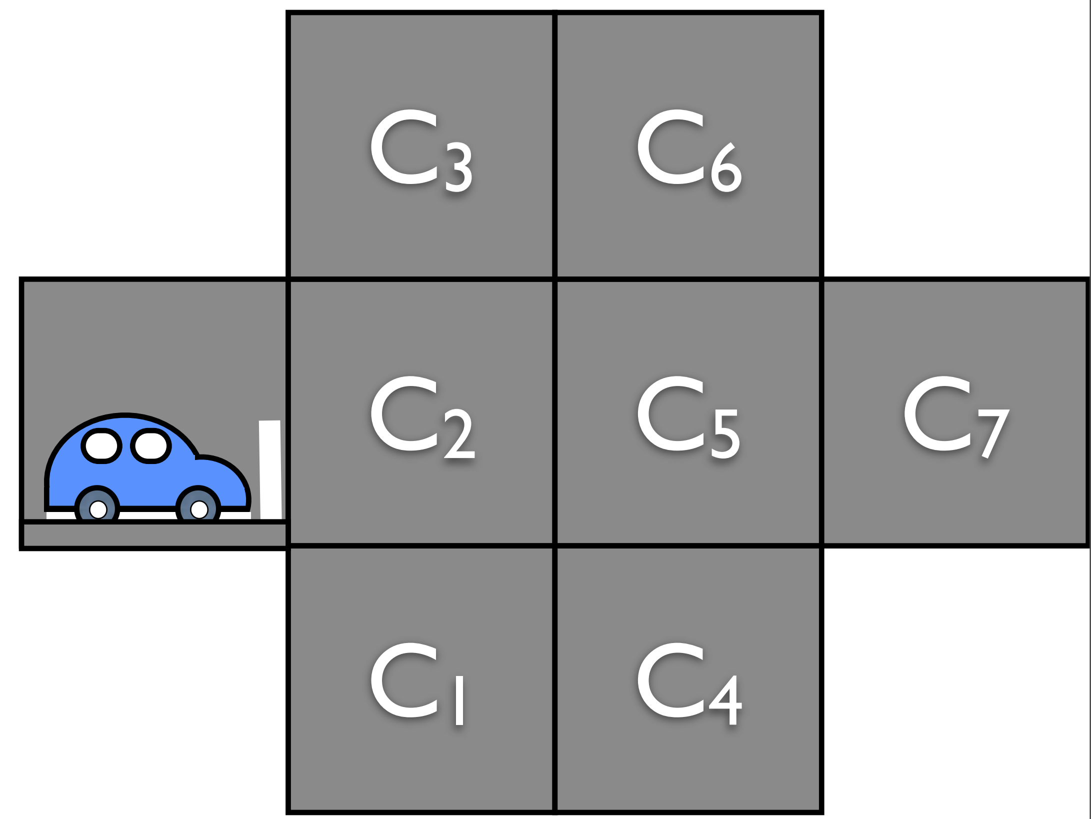
- Infinitely often, PARK signal is not received
- The obstacle always moves to an adjacent cell

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 C_6
- If there is a car at one of the C_3, C_4, C_7 cells at initial state, need to wait until it disappears before going through the intersection
- Go through the intersection only when C_2 and C_5 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

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

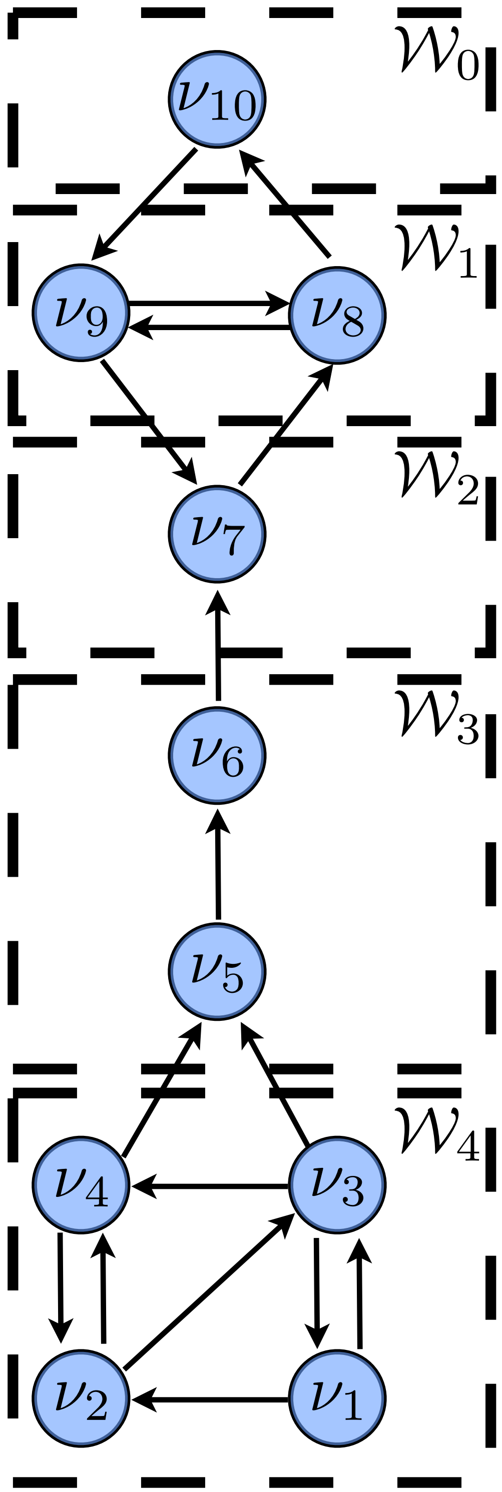
$$\varphi = \left(\psi_{init} \wedge \Box \psi_e^e \wedge \bigwedge_{i \in I_f} \Box \diamond \psi_{f,i}^e \right) \implies \left(\bigwedge_{i \in I_s} \Box \psi_{s,i} \wedge \Box \diamond \psi_g \right)$$

always — Infinitely often — always — Infinitely often

- Organize cells into a partially ordered set $\mathcal{P} = (\{\mathcal{W}_j\}, \preceq_{\psi_g})$ where $\mathcal{W}_0$ is the set of "goal states," i.e., all cells in $\mathcal{W}_0$ satisfy ψ_g
- Assume that for each j, there exist a proposition Φ and a mapping $\mathcal{F}$ such that the following short-horizon specification is realizable

$$\Psi_j = \left((\varsigma \in \mathcal{W}_j) \wedge \Phi \wedge \Box \psi_e^e \wedge \bigwedge_{k \in I_f} \Box \diamond \psi_{f,k}^e \right) \implies \left(\bigwedge_{k \in I_s} \Box \psi_{s,k} \wedge \Box \diamond (\varsigma \in \mathcal{F}(\mathcal{W}_j)) \wedge \Box \Phi \right)$$

  - Φ describes receding horizon invariants
  - $\mathcal{F}(\mathcal{W}_j) \prec \mathcal{W}_j, \forall j \neq 0$ defines intermediate goal for starting in $\mathcal{W}_j$
  - Partial order condition guarantees that we move closer to goal



$\mathcal{W}_0 \prec \mathcal{W}_1 \prec \ldots \prec \mathcal{W}_4$

$\mathcal{F}(\mathcal{W}_4) = \mathcal{W}_2, \mathcal{F}(\mathcal{W}_3) = \mathcal{W}_1, \mathcal{F}(\mathcal{W}_2) = \mathcal{W}_0, \mathcal{F}(\mathcal{W}_1) = \mathcal{W}_0, \mathcal{F}(\mathcal{W}_0) = \mathcal{W}_0$

Key Elements

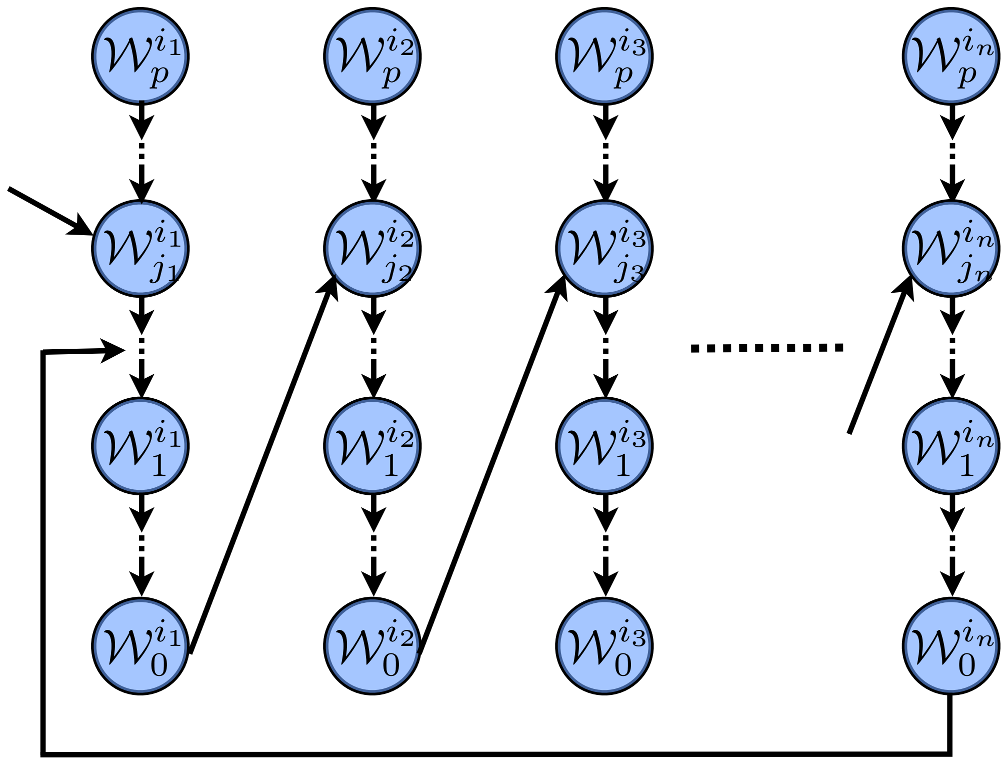
Original specification:

$$\varphi = \left(\psi_{init} \wedge \Box\psi_e^e \wedge \bigwedge_{i \in I_f} \Box\Diamond\psi_{f,i}^e\right) \implies \left(\bigwedge_{i \in I_s} \Box\psi_{s,i} \wedge \bigwedge_{i \in I_g} \Box\Diamond\psi_{g,i}\right)$$

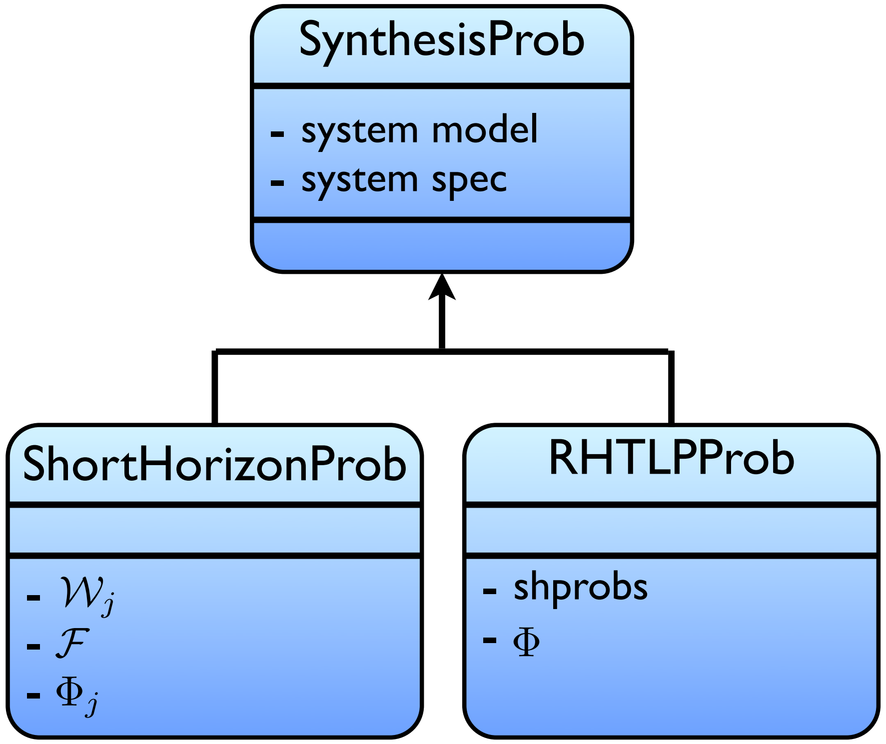
Short horizon specification:

$$\Psi_j^i = \left(\left(\varsigma \in \mathcal{W}_j^i\right) \wedge \Phi \wedge \Box\psi_e^e \wedge \bigwedge_{k \in I_f} \Box\Diamond\psi_{f,k}^e\right) \implies \left(\bigwedge_{k \in I_s} \Box\psi_{s,k} \wedge \Box\Diamond\left(\varsigma \in \mathcal{F}^i(\mathcal{W}_j^i)\right) \wedge \Box\Phi\right)$$

- Partially ordered set $\mathcal{P}^i = (\{\mathcal{W}_0^i, \ldots, \mathcal{W}_p^i\}, \preceq_{\psi_{g,i}})$
  - $\mathcal{W}_0^i \cup \mathcal{W}_1^i \cup \ldots \cup \mathcal{W}_p^i = \mathcal{V}$
  - $\mathcal{W}_0^i$ 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 Φ
  - $\psi_{init} \implies \Phi$ 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 Φ that makes this short horizon problem realizable.

RHTLPProb

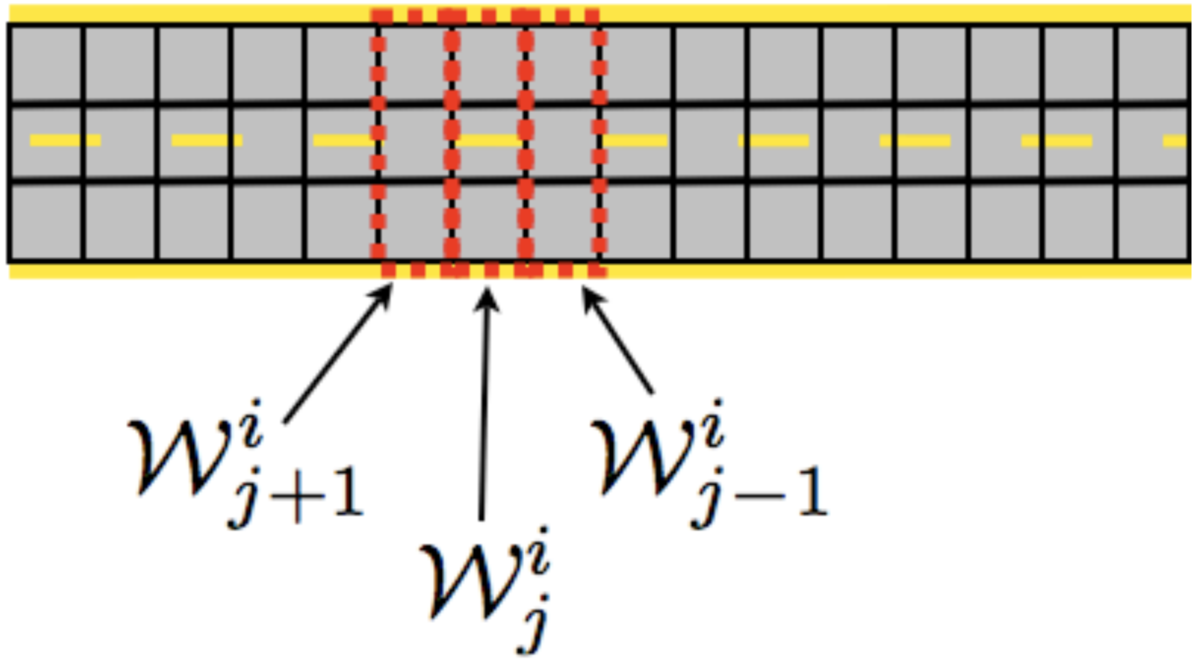
- A class for defining a receding horizon temporal logic planning problem
- Contains a collection of short-horizon problems
- Useful methods
  - *computePhi*(): automatically compute Φ for this receding horizon temporal logic planning problem if one exists.
  - *validate*(): check whether all the sufficient conditions for doing receding horizon temporal logic planning are satisfied

Example: autonomous_car_road.py

Autonomous vehicle navigating an urban environment

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