Lecture 11

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, 17 May 2012

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

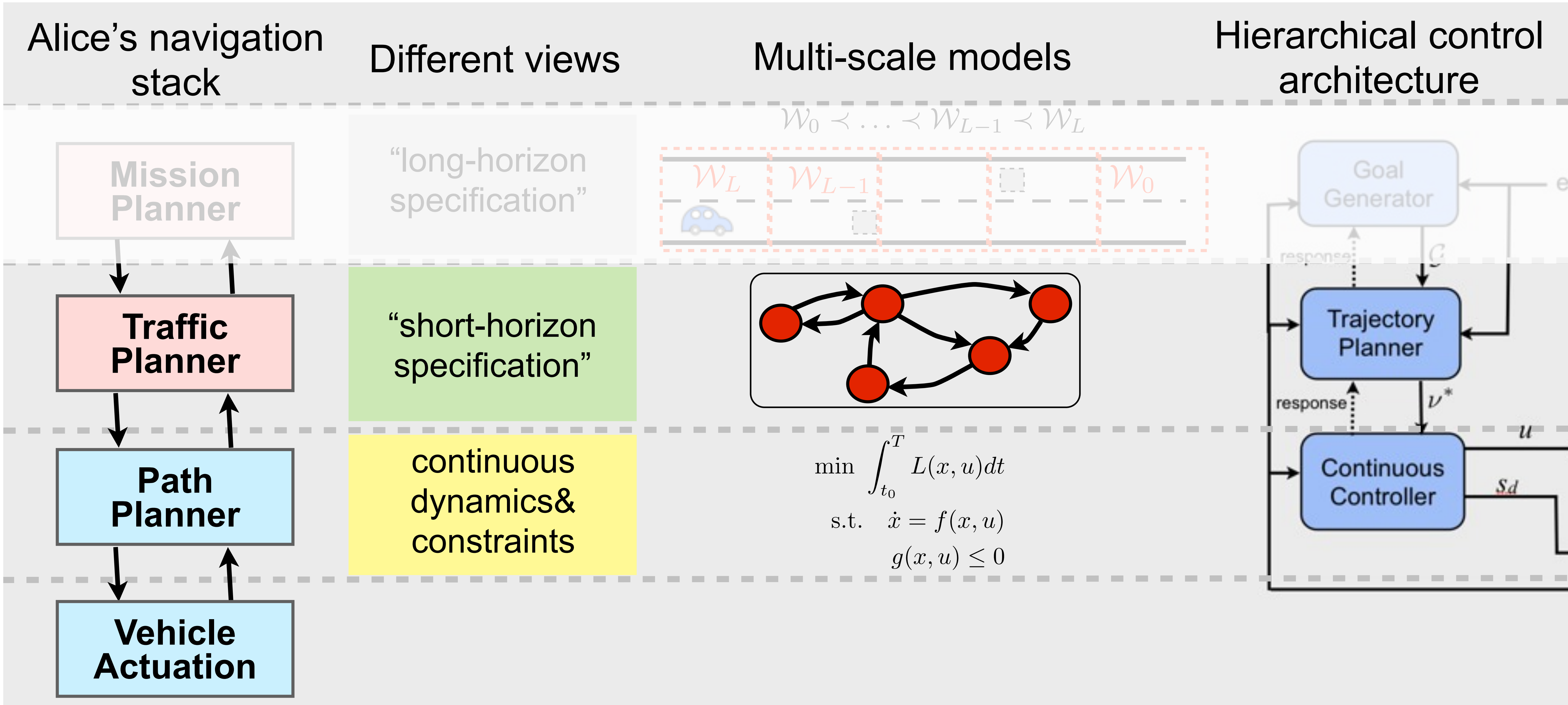
$$\begin{aligned} s(t+1) &= As(t) + Bu(t) + Ed(t)) \\ u(t) &\in U \\ d(t) &\in D \end{aligned}$$

  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

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

TuLiP for Hierarchical Control



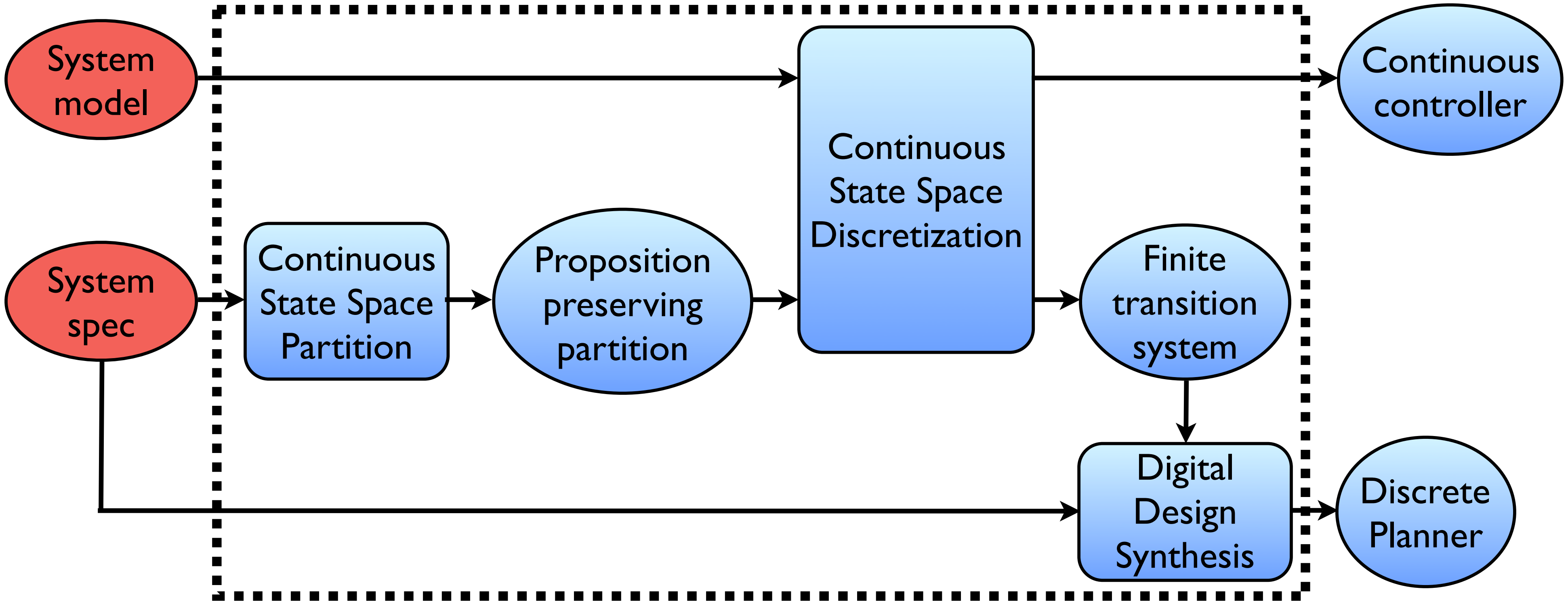
Input:

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

Output:

- "strategy" to be implemented in each layer

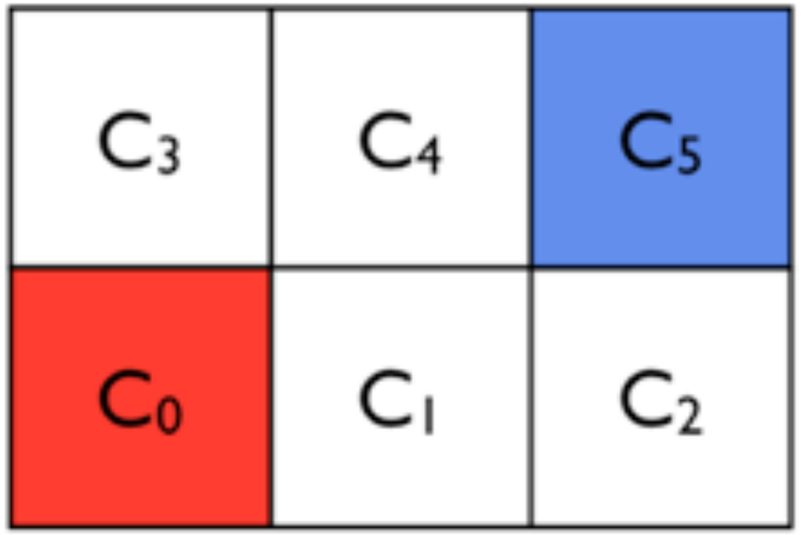
Main Steps



- 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 = **discretize**(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 $\dot{x} = u_x, \dot{y} = u_y$ 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

$$\varphi = \square\lozenge(\neg park) \implies (\square\lozenge(s \in C_5) \wedge \square(park \implies \lozenge(s \in C_0)))$$

This spec is not a GR[1] formula

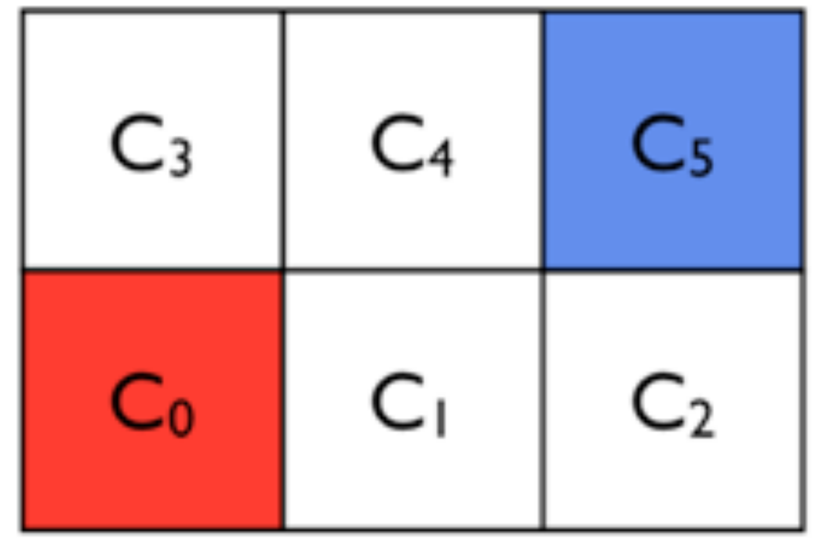
- Introduce an auxiliary variable *X0reach* that starts with True
- $\square(\bigcirc X0reach = (s \in C_0 \vee (X0reach \wedge \neg park)))$
- $\square\lozenge X0reach$

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

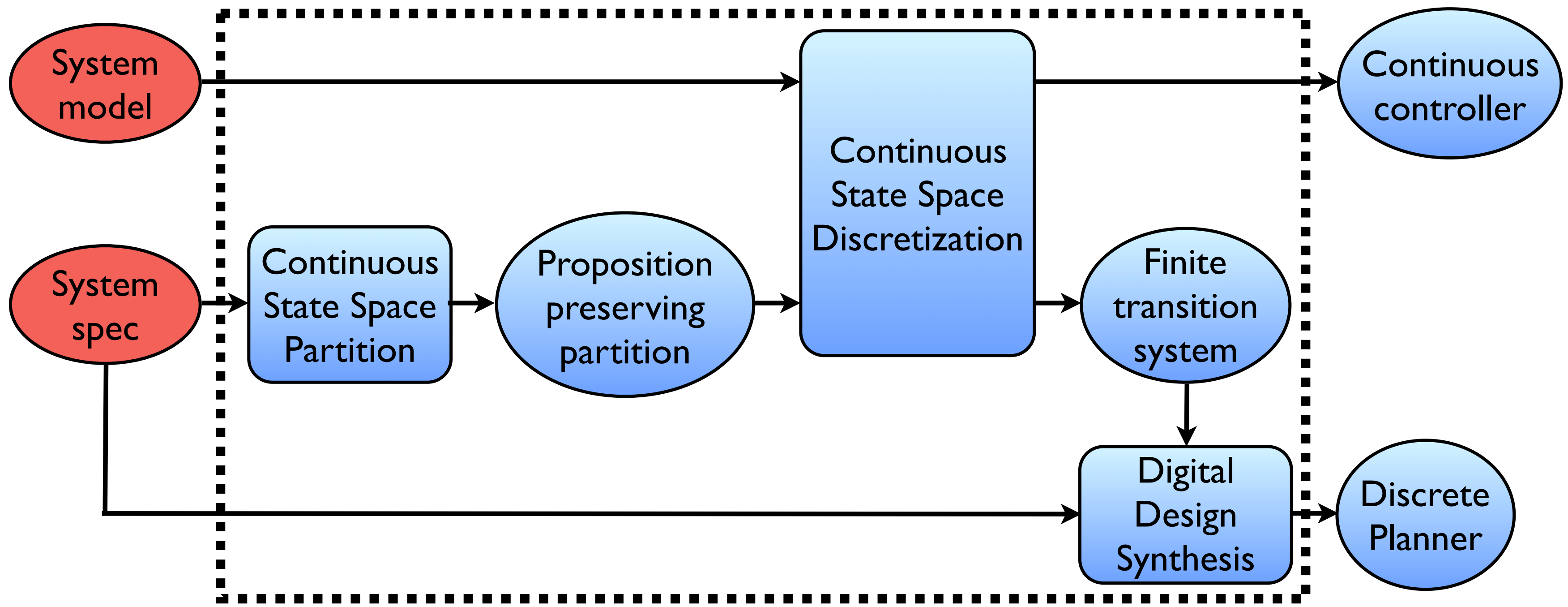
$$\varphi = \Box\Diamond(\neg park) \implies (\Box\Diamond(s \in C_5) \wedge \Box(park \implies \Diamond(s \in C_0)))$$

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

- Self-contained structure for defining an embedded control software synthesis problem
- Fields of SynthesisProb
  - *env_vars*
  - *sys_vars*
  - *spec*
  - *disc_cont_var*
  - *disc_dynamics*



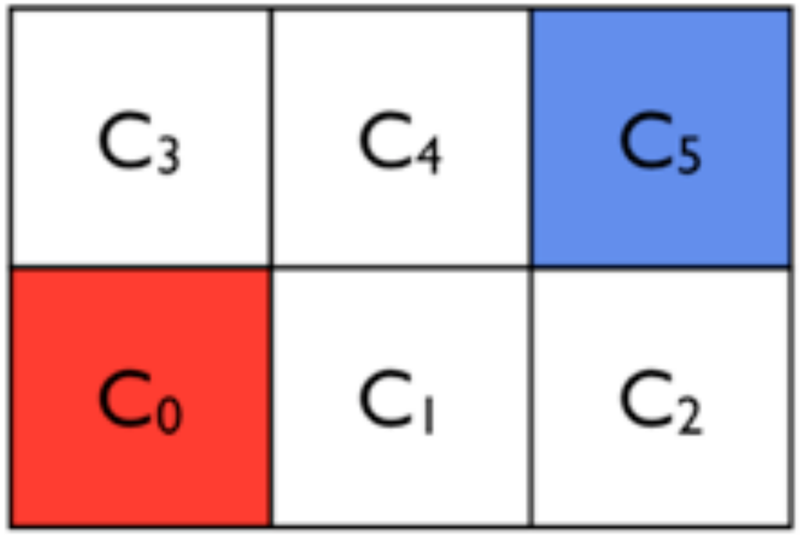
- Useful methods
  - ***checkRealizability***(*heap_size='-Xmx128m', pick_sys_init=True, verbose=0*): check whether this problem is realizable
  - ***getCounterExamples***(*recompute=False, heap_size='-Xmx128m', pick_sys_init=True, verbose=0*):

    return the set of initial states starting from which the system cannot satisfy the spec
  - ***synthesizePlannerAut***(*heap_size='-Xmx128m', priority_kind=3, init_option=1, verbose=0*):

    synthesize 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

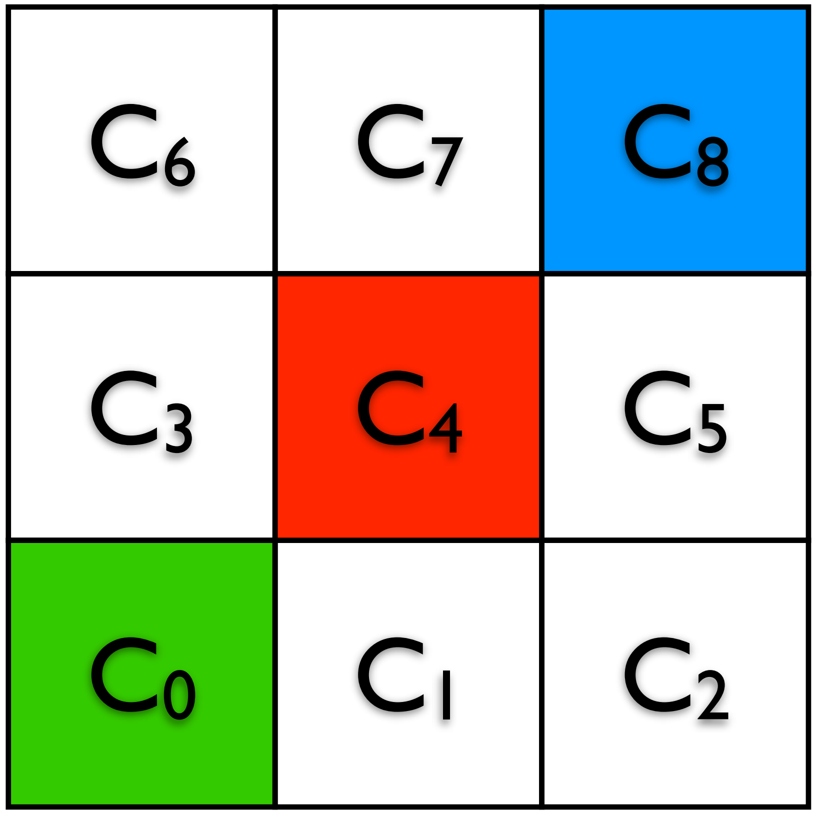
- Infinitely often, PARK signal is not received

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

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 Lab



Synthesize a reactive planner for the robot with the following specification

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

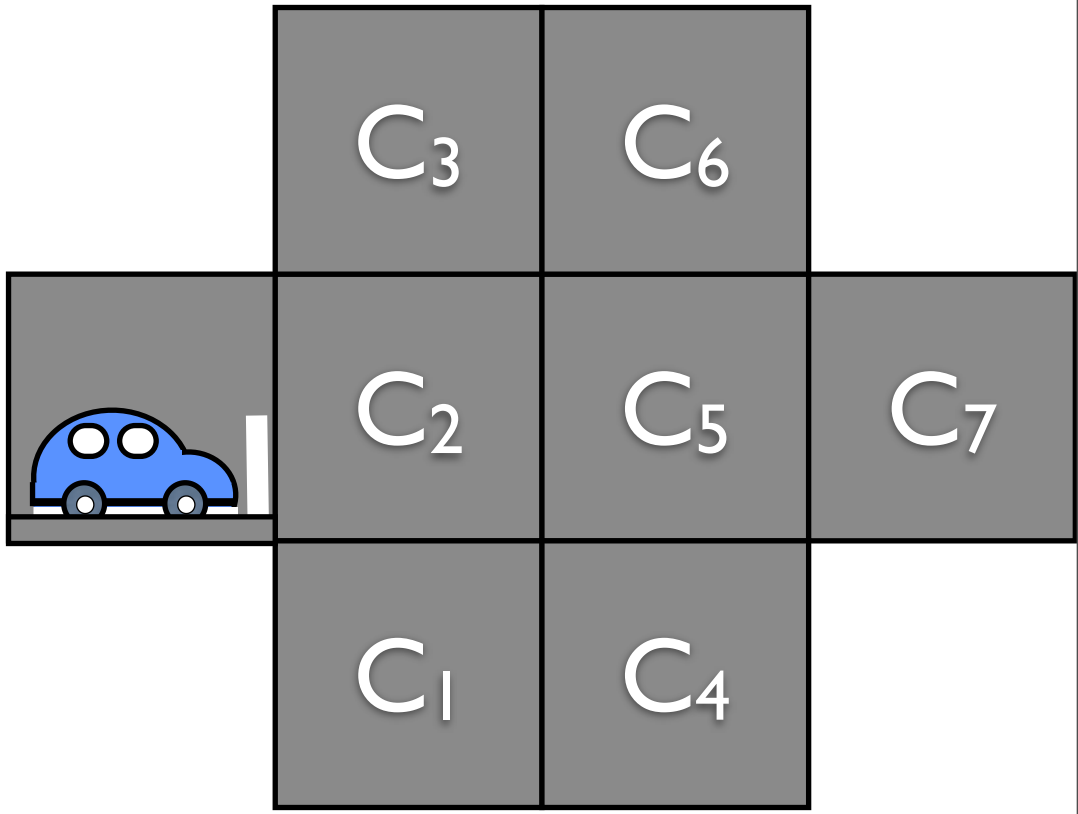
Assumption

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

Constraint

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

Computer Lab



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

- ??

Constraint

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