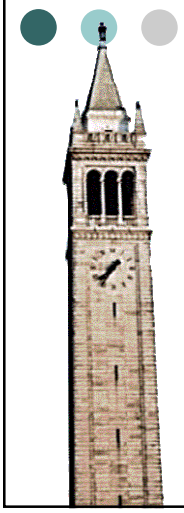
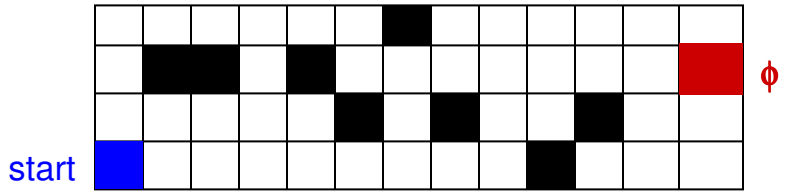


Introduction to Embedded Systems

Edward A. Lee & Sanjit A. Seshia
UC Berkeley
EECS 149
Spring 2009

Lecture 19: Controller Synthesis

A Robot delivery service, with moving obstacles



ϕ = destination for robot

At any time step:

Robot can move Left, Right, Up, Down, Stay Put

Environment can move one obstacle Up or Down or Stay Put

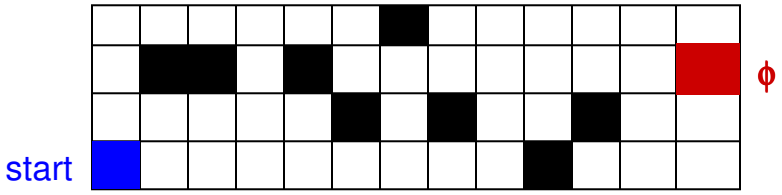
→ But only total of 2 times over all time steps

Can model Robot and Env as FSMs

→ Robot state = its position,

→ Env state = positions of obstacles and count

Recap of topics covered last time



1. Stated the goal in temporal logic: **F** ϕ
   - The problem is a "reachability problem"
2. Gave an algorithm to solve a version of the reachability problem
3. Considered some alternative goals:

   F $\phi_1 \wedge$ **F** $\phi_2 \wedge \ldots \wedge$ **F** ϕ_n

   F($\phi_1 \wedge$ **F (** $\phi_2 \wedge \ldots \wedge$ **F** ϕ_n **))**

Reachability Analysis, Revisited

The reachability problem:

Given an FSM M = (Q, δ, Q_0), and a state ϕ,
is *s* reachable from some $q_0 \in Q_0$ by following δ ?

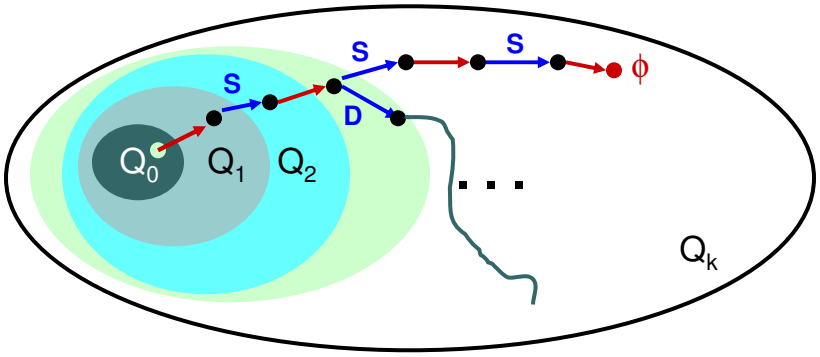
System evolution according to δ is a sequence:
robot step, env step, robot step, env step, …

ϕ is reachable if there is some sequence of robot and env steps from q_0 to ϕ

→ This seq need not have the worst-case env steps!

→ It's optimistic – assumes helpful environment (not adversarial)

Visualizing 'Optimistic' Controller Synthesis



S – obstacles "stay put"

D – obstacle "moves down"

Synthesis with an Adversarial Environment

Suppose at every step, an adversary picks the worst possible action to stop the robot's progress

This is a game between the system and its adversarial environment

We want to modify the search performed by the reachability algorithm to handle this worst-case behavior.

Any ideas?

Controllable States

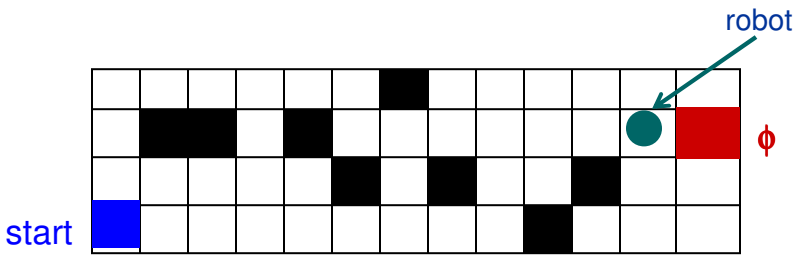
Idea:

Compute the set of states from which, no matter what the environment does, the robot can reach the red square ϕ.

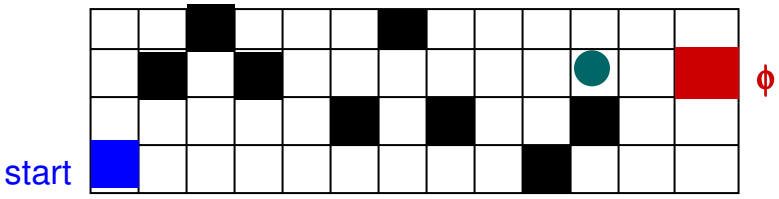
Such states are called controllable states.

What are some examples of controllable states for our robot example?

Examples of Controllable States

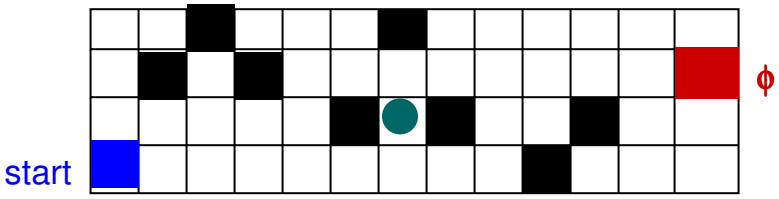


Examples of Controllable States



Env_Moves = MAX_MOVES

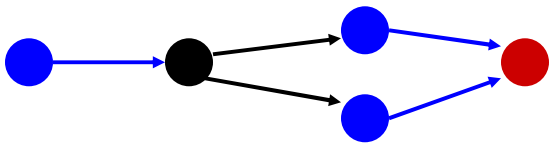
Examples of Controllable States



Env_Moves = MAX_MOVES

Synthesis Algorithm

1. Start with the set of trivially controllable states S_0
2. Add all states from which the robot can reach S_0 in one *combined step* (robot step, env step), no matter what the environment does



3. Repeat until no new states added
4. Check if this set contains a start state.
   If yes, then we found a strategy.
   If no, then no strategy exists against the worst-case environment (adversary).

Synthesis Algorithm: Formal Description

Input: Description of M: (Q_0, δ), Goal state ϕ

env / robot

Each state $q = (q_1, q_2)$, δ_1 updates q_1, δ_2 updates q_2

Output: Does Q_0 contain a controllable state?

```
Init: S := S_new := φ;    /* Note φ != ∅ */
while (S_new != ∅) {
  if (S_new ∩ Q_0 != ∅) return YES;
  S' := { q  |  ∀ p_1 ∈ δ_1(q) ∃ p_2 ∈ δ_2(p_1) s.t. p_2 ∈ S }
                ∪ S
  S_new := S' \ S;   S  := S';
}
return NO;
```

S is the set of controllable states

Controller Synthesis for **G** *p*

Suppose we want the system to always satisfy property p
→ alternatively, we want the system to never satisfy ¬ p

E.g. robot should never hit an obstacle; aircraft should never collide; etc.

How can we use the previous algorithm to synthesize a control strategy for **G** p?

[Idea: switch the roles of the environment and the system (robot) – now the environment is trying to reach a goal state. Avoid the states controllable by the environment]

Recap of Concepts

Synthesis is a Game
between the Robot ("System") and its Environment

Goal for robot: **F** ϕ
- Robot wins if it reaches ϕ
- Environment wins otherwise
- → Zero-sum game

Goal for env: **G** ¬ ϕ
- Environment wins if ϕ is always false
- Robot wins otherwise

Rest of today's lecture

- Discuss synthesis for G F p
- How to synthesize a continuous trajectory

Handling other kinds of Temporal Logic Goals
G F p

Example:
The iRobot must visit the charging station infinitely often

- Consider the FSM formed by
  composing the iRobot FSM with its Environment FSM
- Visualize this FSM as a directed graph
- Suppose that "visiting the charging station" is a state p in this graph

  What graph property corresponds to visiting the state p infinitely often?

Winning Strategy for **G F** p

The system must visit state p infinitely often

For benign environment (optimistic synthesis):

→ The state graph must contain a cycle with state p

→ How can we detect this if we have to build the graph on the fly?

Finding Winning Strategy for **G F** p

The system must visit state p infinitely often

For benign environment (optimistic synthesis):

→ The state graph must contain a cycle with state p

→ How can we detect this if we have to build the graph on the fly?

Two steps:

1. Check if p is reachable from the initial state
2. Check if p is reachable from itself

Winning Strategy for **G F** p: Adversarial Setting

The system must visit state p infinitely often

How do we check this for an adversarial environment?

Perform "Adversarial Reachability"!

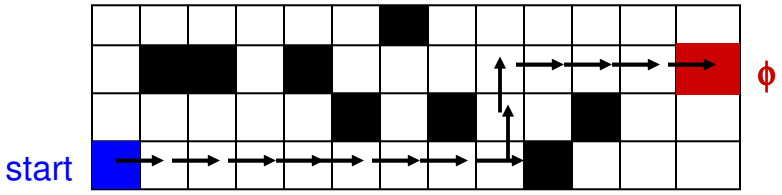
Checking that p is reached infinitely often for an adversarial environment:

Two steps:

1. Check if p is reachable from the initial state,
   no matter what the adversary does
2. Check if p is reachable from itself,
   no matter what the adversary does

For each of these steps, use the algorithm we used on slide 12 of this lecture

Synthesizing a Continuous Trajectory



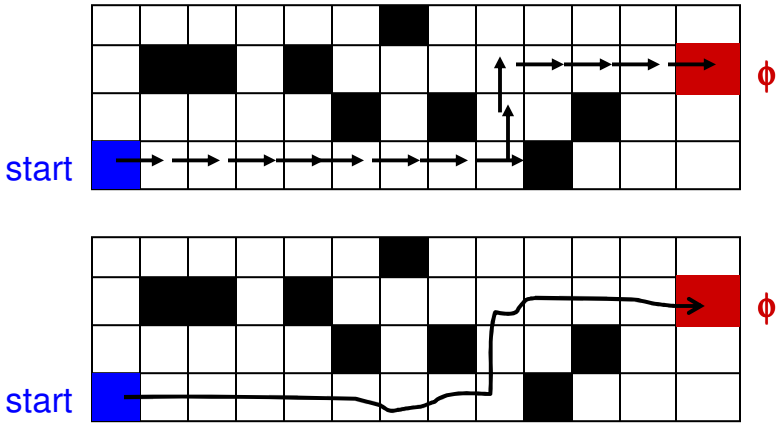
Suppose we have a discrete trajectory (path) to ϕ

How do we transform that into the desired continuous trajectory?

(assume static obstacles)

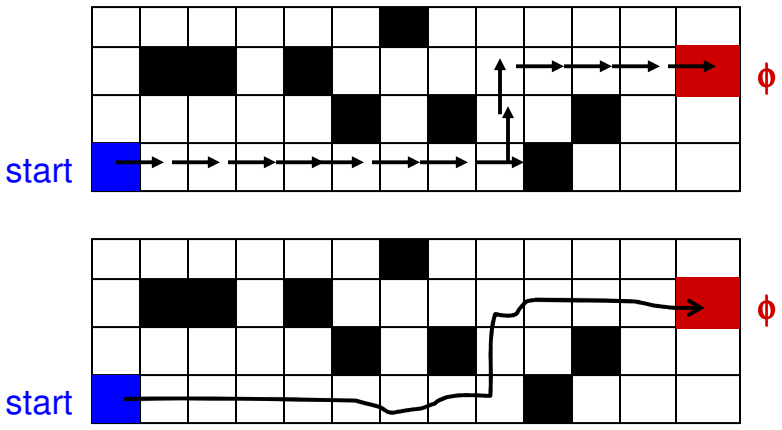
Necessary Condition

If there exists a discrete trajectory, then there must also exist a continuous trajectory



The Other Condition

If there isn't a discrete trajectory, it is possible/OK for a continuous trajectory to exist?



Bisimulation (revisited)

The property we need is bisimulation.

Given:
System = Robot + Environment
The original system H, which is a hybrid system
The discretized version D of H, which is an FSM

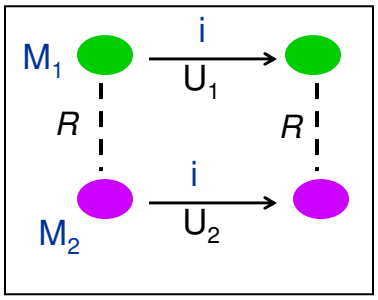
Claim:
If there is a bisimulation between D and H, that suffices to map a trajectory of D to one of H, and vice versa

Bisimulation for FSMs

Let $M_1 = (S_1, I_1, O_1, U_1, s_{10})$ and $M_2 = (S_2, I_2, O_2, U_2, s_{20})$
where $I = I_1 = I_2$ and $O = O_1 = O_2$

We say M_1 bisimulates M_2 iff
there exists a set $R \subseteq S_1 \times S_2$ such that

1. $R(s_{10}, s_{20})$
2. For all $(s_1, s_2) \in R$, the following conditions hold:
   For all $i \in I$, and $(t_2, o_2) = U_2(s_2, i)$,
   there exists a $(t_1, o_1) = U_1(s_1, i)$ s.t.
   $(t_1, o_1) \in R$ and $o_2 = o_1$
   For all $i \in I$, and $(t_1, o_1) = U_1(s_1, i)$,
   there exists a $(t_2, o_2) = U_2(s_2, i)$ s.t.
   $(t_2, o_2) \in R$ and $o_2 = o_1$



Bisimulation between FSM and Hybrid System (HS)

Given:

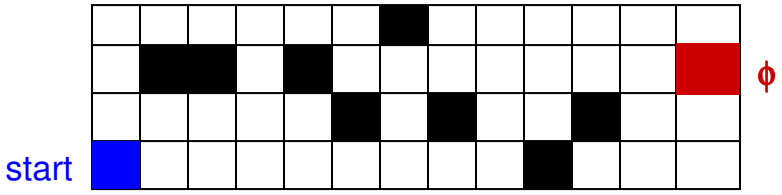
Suppose the FSM M is obtained by partitioning up the continuous state space of the HS H into regions (e.g. rectangles)

- Let the partition be $P : \mathbb{R}^2 \rightarrow Q$

Then M bisimulates H if:

1. If P(x) = P(y), then points x and y are observationally equivalent
2. If P(x) = P(y), then
   for every x' reachable from x, there is a y' reachable from y s.t. P(x') = P(y')
   and vice-versa

Our Example

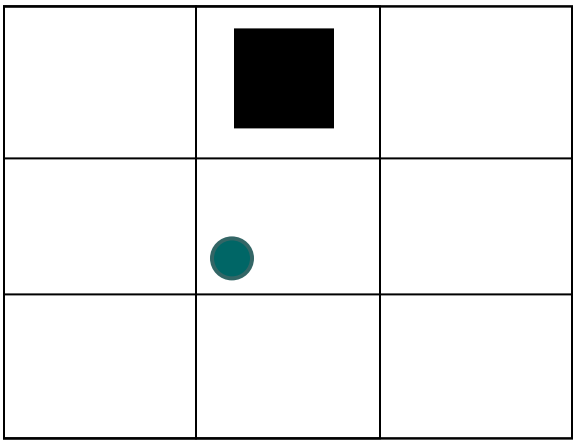


The grid above is a partition P of the 2-D space in the room

When is P a bisimulation?

Zooming In: Bisimulation Condition 1

Sensors should work the same anywhere in a square



Zooming In: Bisimulation Condition 2

Synthesize local control laws that mimic a discrete step from square to adjacent square