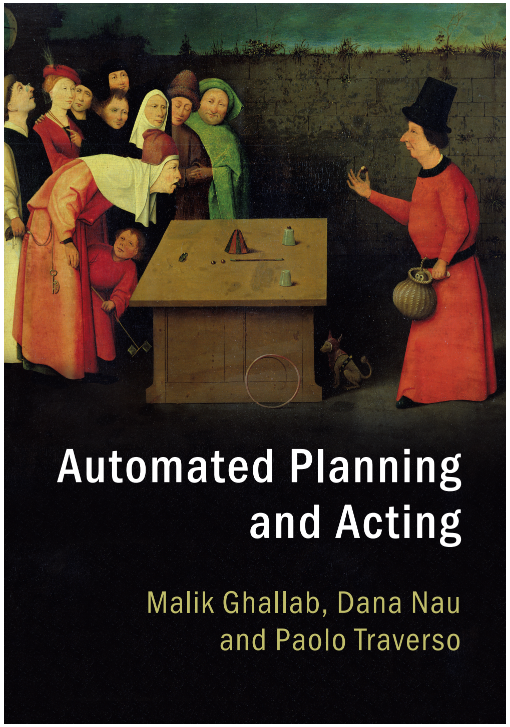
Last update: June 22, 2017

Deliberation in Planning and Acting



http://www.laas.fr/planning

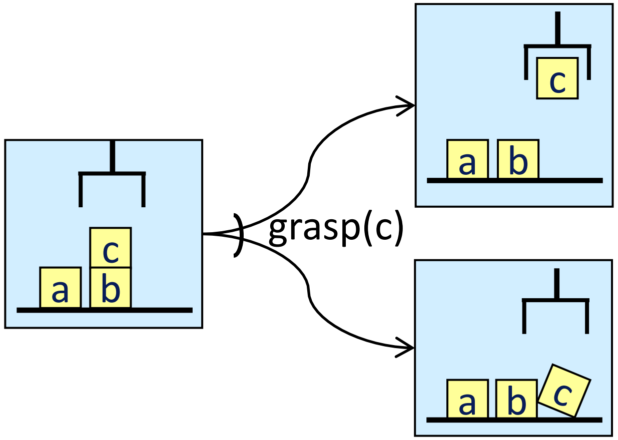
Part 4: Nondeterministic Models

Malik Ghallab LAAS/CNRS, University of Toulouse
Dana Nau University of Maryland
Paolo Traverso FBK ICT IRST, Trento, Italy

Motivation

- Nondeterministic action: more than one possible outcome

- In some cases, whether to model nondeterminism is a design choice
  - In Part 2 we discussed conditions under which it's OK to have a deterministic model of a nondeterministic environment
    - Model the "nominal case"
      - The outcome we usually expect
    - Recover at acting time if things turn out differently



In Some Cases, Nondeterminism is a Must

- No clear "nominal case"

In Some Cases, Nondeterminism is a Must

- Huge state space

  At least $n!$, where n = number of containers

- Busy location

  Many exogenous events

  Many possible outcomes

  No clear “nominal case”

- Individual actor knows very little about the current state

  e.g., sensing to identify containers

Outline

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

Ambiguous Door

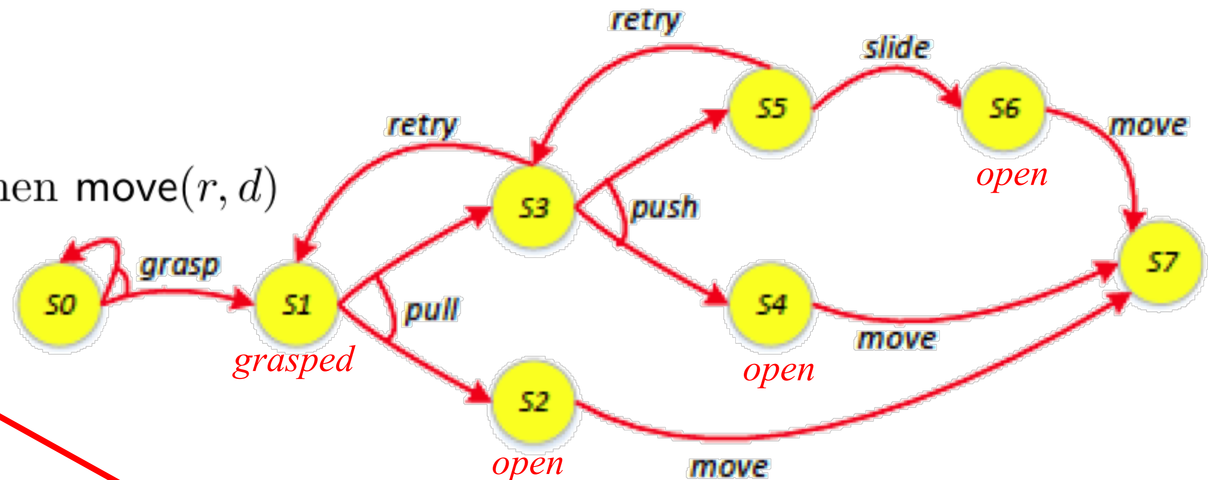
- Door to one of the ICAPS workshop rooms
  - Pull, or push?
  - I had to try both
- Next: Rae methods for that …

Outside the room, going in

Inside the room, going out

Rae Methods

m-opendoor(r, d, l, o)
task: opendoor(r, d)
pre: loc$(r) = l \wedge$ adjacent$(l, d) \wedge$ handle(d, o)
body: while $\neg$grasped(d) do
grasp(r, d)
pull(r, d)
if door-status(d)=open then move(r, d)
else pull-push(r, d)



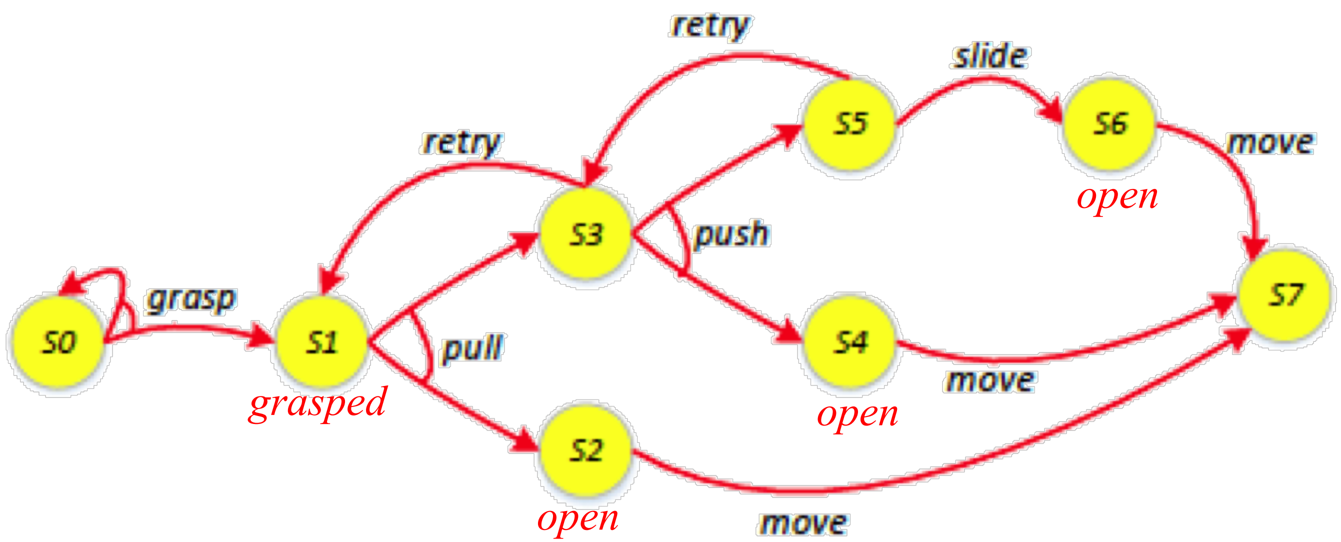
m-retry-pull(r, d, l, o)
task: pull-push(r, d)
body: pull(r, d);
if door-status(d)=open then move(r, d)
else pull-push(r, d)

m-push(r, d, l, o)
task: pull-push(r, d)
body: push(r, d)
if door-status(d)=open then move(r, d)
else push-slide(r, d)

m-retry-push(r, d, l, o)
task: push-slide(r, d)
body: push(r, d);
if door-status(d)=open then move(r, d)
else push-slide(r, d)

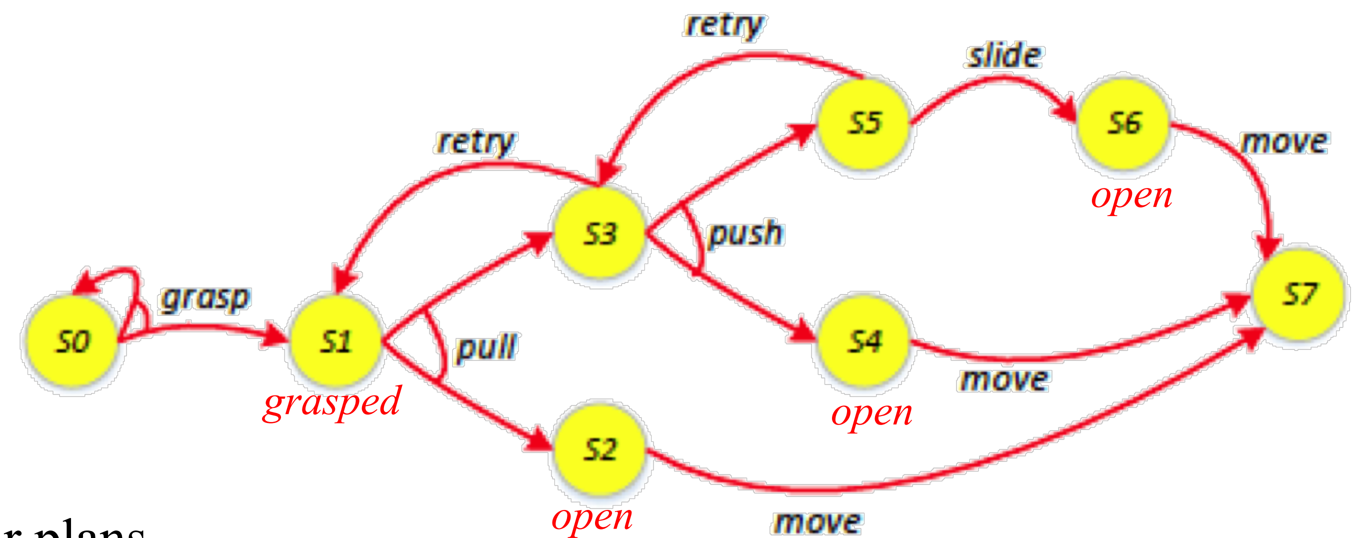
m-slide(r, d, l, o)
task: push-slide(r, d)
body: slide(r, d)

Nondeterministic Planning Domain



- S – finite set of states
- A – finite set of actions
- Applicable(s) = {all actions applicable in state s}
- γ: $S \times A \rightarrow 2^S$ state-transition function
  - $\gamma(s,a)$ = all possible outcomes of applying action a in state s

~~Plans~~ Policies



- Can't use linear plans
  - ⟨grasp, pull, move⟩
  - ➢ Can't pull if grasp doesn't succeed
  - ➢ Can't move if push doesn't open the door
- Instead, use a *policy*
  - ➢ function that maps states to actions
  - ➢ $\pi(s)$ = action to perform in state s

PerformPolicy(π)

  $s \leftarrow$ observe current state
  while $s \in \mathrm{Dom}(\pi)$ do
    perform action $\pi(s)$
    $s \leftarrow$ observe current state

Policies

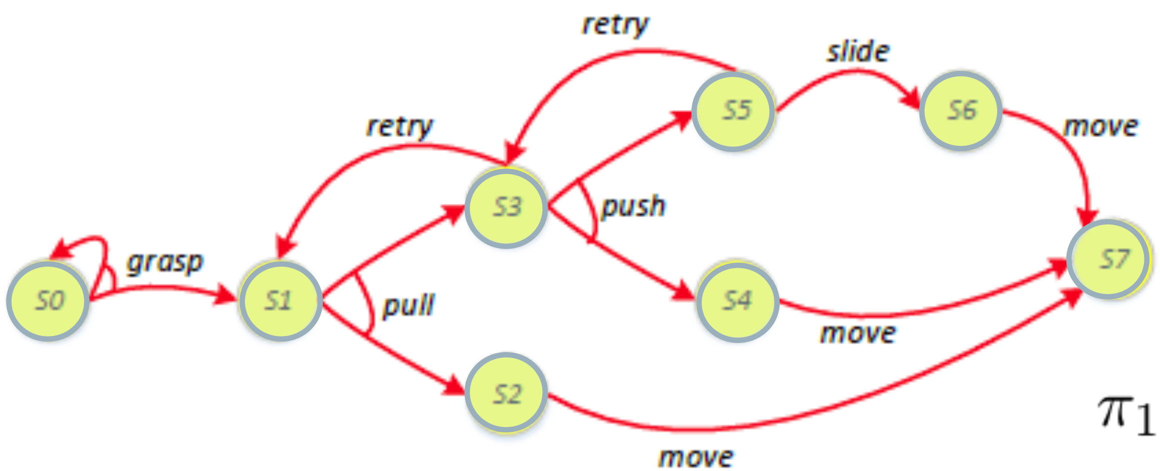
PerformPolicy(π)

$s \leftarrow$ observe current state

while $s \in \mathrm{Dom}(\pi)$ do

perform action $\pi(s)$

$s \leftarrow$ observe current state



π_1 : $\boxed{\pi_1(s_0) = \text{grasp}}$

$\boxed{\pi_1(s_1) = \text{pull}}$

$\boxed{\pi_1(s_2) = \text{move}}$

$\boxed{\pi_1(s_3) = \text{push}}$

$\boxed{\pi_1(s_4) = \text{move}}$

$\boxed{\pi_1(s_5) = \text{slide}}$

$\boxed{\pi_1(s_6) = \text{move}}$

Policies

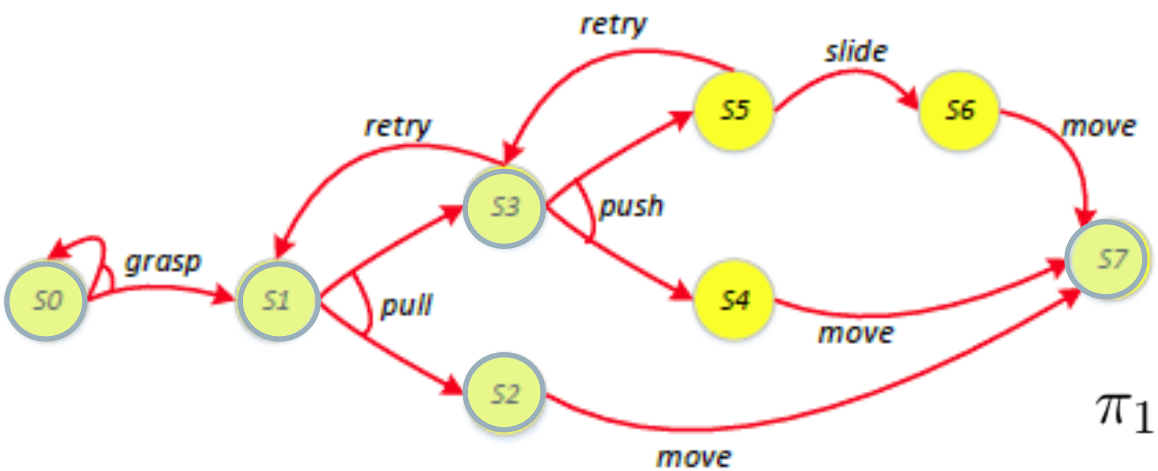
PerformPolicy(π)

$s \leftarrow$ observe current state

while $s \in \mathrm{Dom}(\pi)$ do

perform action $\pi(s)$

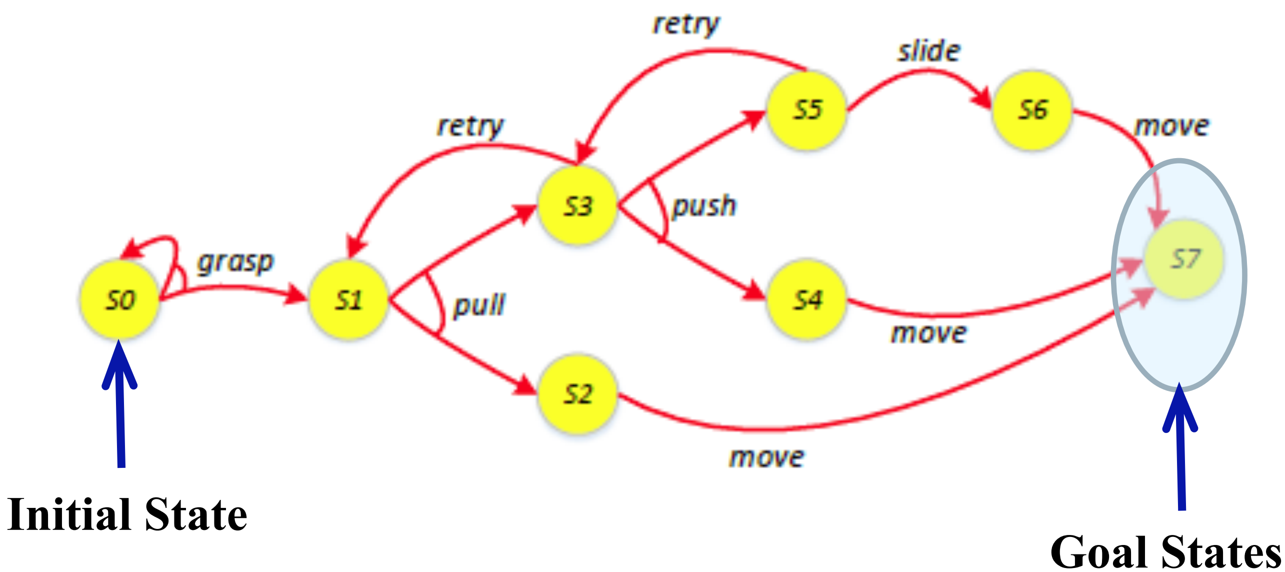
$s \leftarrow$ observe current state



$$\pi_1: \quad \begin{aligned} \pi_1(s_0) &= \text{grasp} \\ \pi_1(s_1) &= \text{pull} \\ \pi_1(s_2) &= \text{move} \\ \pi_1(s_3) &= \text{push} \\ \pi_1(s_4) &= \text{move} \\ \pi_1(s_5) &= \text{slide} \\ \pi_1(s_6) &= \text{move} \end{aligned}$$

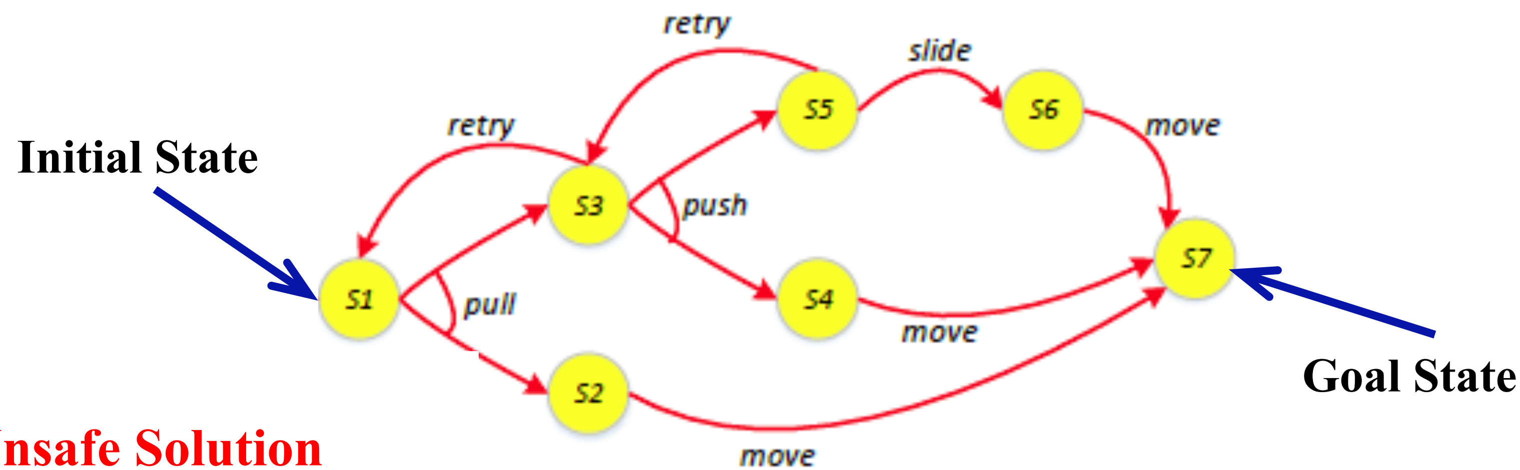
$$\pi_2: \quad \begin{aligned} &\boxed{\pi_2(s_0) = \text{grasp}} \\ &\boxed{\pi_2(s_1) = \text{pull}} \\ &\boxed{\pi_2(s_2) = \text{move}} \\ &\boxed{\pi_2(s_3) = \text{retry}} \end{aligned}$$

Planning Problems



- Nondeterministic planning problem
  - Nondeterministic planning domain
  - Initial state s_0
  - Set of goal states S_g

Solutions to Planning Problems



Unsafe Solution

π_3 : $\pi_3(s_1) = \text{pull}$

$\pi_3(s_2) = \text{move}$

π_2 : **Safe Cyclic Solution**

$\pi_2(s_1) = \text{pull}$

$\pi_2(s_2) = \text{move}$

$\pi_2(s_3) = \text{retry}$

π_1 : **Safe Acyclic Solution**

$\pi_1(s_1) = \text{pull}$

$\pi_1(s_2) = \text{move}$

$\pi_1(s_3) = \text{push}$

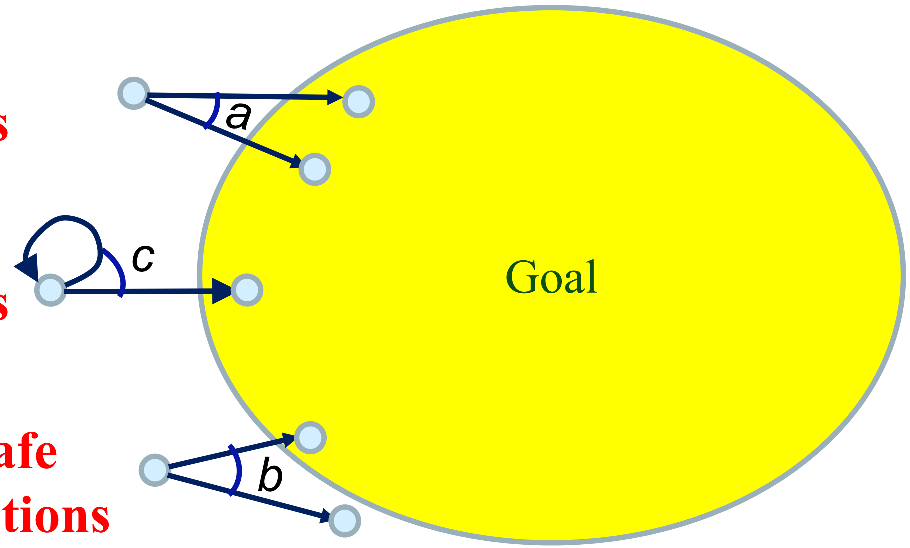
$\pi_1(s_4) = \text{move}$

$\pi_1(s_5) = \text{slide}$

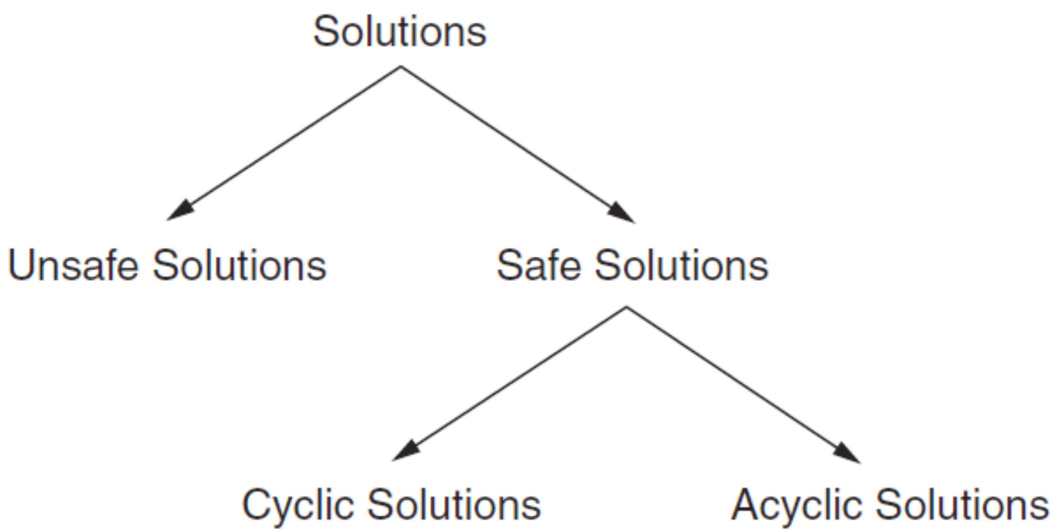
$\pi_1(s_6) = \text{move}$

Solutions to Planning Problems

- solutions
  - safe solutions
    - acyclic solutions
    - cyclic solutions
  - unsafe solutions



The Planning Problem: Solutions



our terminology	**nondeterminism**	**probabilistic**
solutions	*weak solutions*	-
unsafe solutions	-	*improper solutions*
safe solutions	*strong cyclic solutions*	*proper solutions*
cyclic safe solutions	-	-
acyclic safe solutions	*strong solutions*	-

Table 5.1: Solutions: Different terminologies in the literature

Outline

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

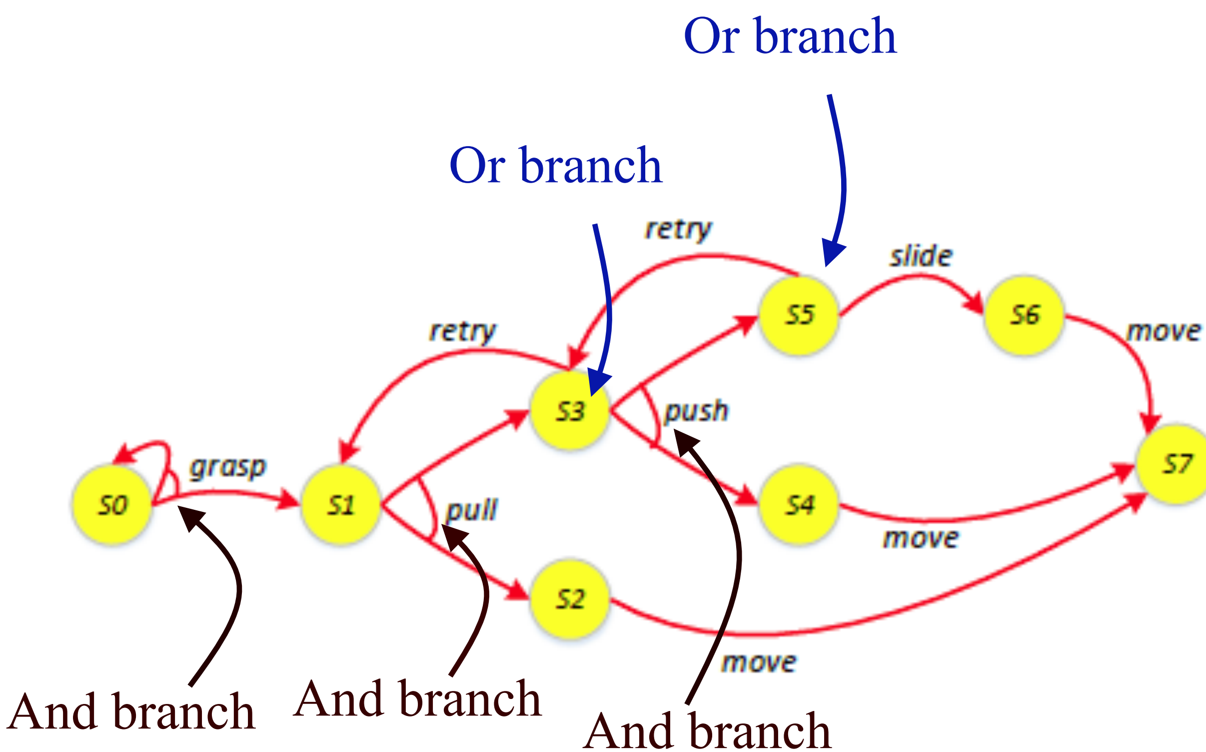
Some Planning Techniques

And/Or Graph Search

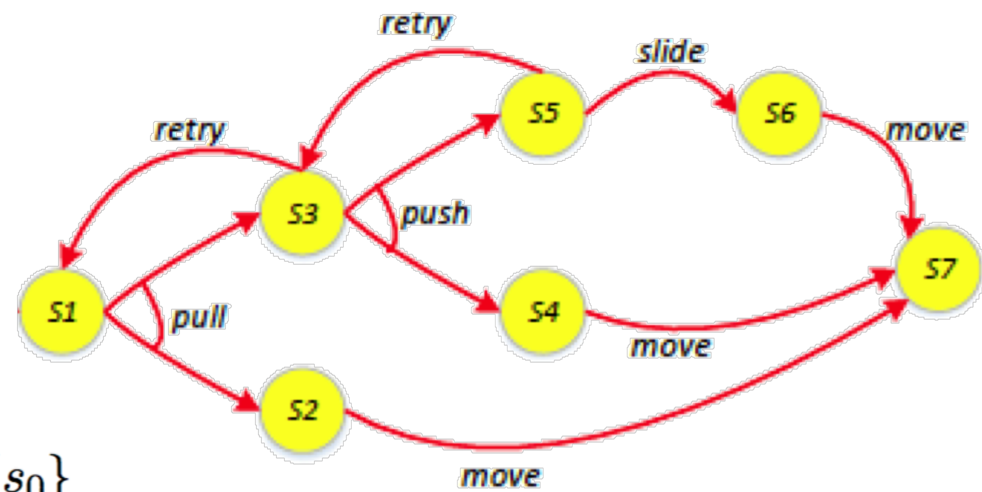
Symbolic Model Checking

Determinization

And/Or Graphs



Finding (Unsafe) Solutions



Find-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing;\ \ s \leftarrow s_0;\ \ \mathit{Visited} \leftarrow \{s_0\}$

loop

if $s \in S_g$ then return π

$A' \leftarrow \text{Applicable}(s)$

if $A' = \varnothing$ then return failure

nondeterministically choose $a \in A'$

$\boxed{\text{nondeterministically choose } s' \in \gamma(s, a)}$

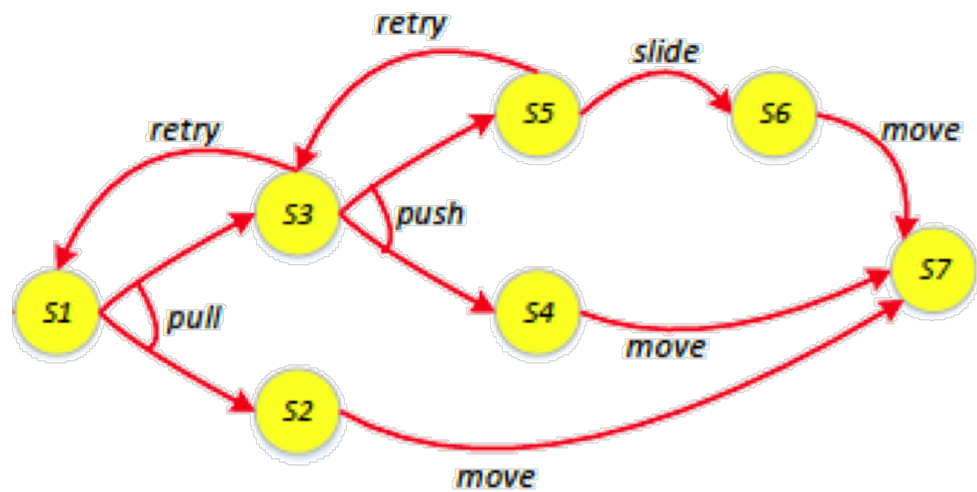
if $s' \in \mathit{Visited}$ then return failure

$\pi(s) \leftarrow a;\ \ \mathit{Visited} \leftarrow \mathit{Visited} \cup \{s'\};\ s \leftarrow s'$

Decide which state to plan for

Finding Acyclic Safe Solutions

*Keep track of unexpanded states, like A**



Find-Acyclic-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$
  $\mathit{Frontier} \leftarrow \{s_0\}$
  for every $s \in \mathit{Frontier} \setminus S_g$ do
    $\mathit{Frontier} \leftarrow \mathit{Frontier} \setminus \{s\}$
    if $\mathrm{Applicable}(s) = \varnothing$ then return failure
    nondeterministically choose $a \in \mathrm{Applicable}(s)$
    $\pi \leftarrow \pi \cup (s, a)$
    $\mathit{Frontier} \leftarrow \mathit{Frontier} \cup (\gamma(s,a) \setminus \mathrm{Dom}(\pi))$
    if has-loops$(\pi, a, \mathit{Frontier})$ then return failure
  return π

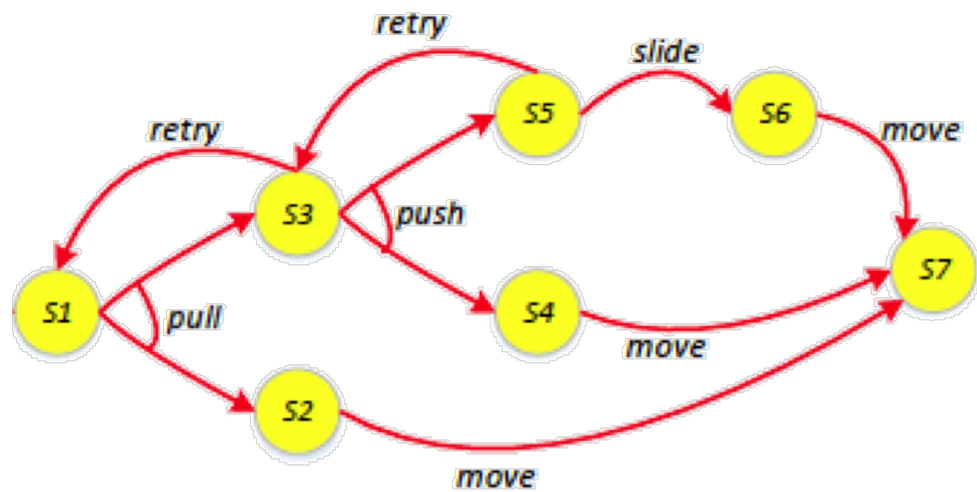
Add all outcomes that π doesn't already handle

Check whether π contains any cycles:

$$\exists s' \in (\gamma(s,a) \cap \mathrm{Dom}(\pi)) \text{ such that } s \in \hat{\gamma}(s',\pi)$$

Finding (Cyclic) Safe Solutions

*Keep track of unexpanded states, like A**



Find-Safe-Solution (Σ, s_0, S_g)
  $\pi \leftarrow \varnothing$
  $Frontier \leftarrow \{s_0\}$
  for every $s \in Frontier \setminus S_g$ do
    $Frontier \leftarrow Frontier \setminus \{s\}$
    if $\text{Applicable}(s) = \varnothing$ then return failure
    nondeterministically choose $a \in \text{Applicable}(s)$
    $\pi \leftarrow \pi \cup (s, a)$
    $Frontier \leftarrow Frontier \cup (\gamma(s,a) \setminus \text{Dom}(\pi))$
    if has-unsafe-loops$(\pi, a, Frontier)$ then return failure
  return π

Add all outcomes that π doesn't already handle

Check whether π contains any cycles that can't be escaped:

$$\exists\, s' \in (\gamma(s,a) \cap \text{Dom}(\pi)) \text{ such that } \hat{\gamma}(s',\pi) \cap Frontier = \varnothing$$

Some Planning Techniques

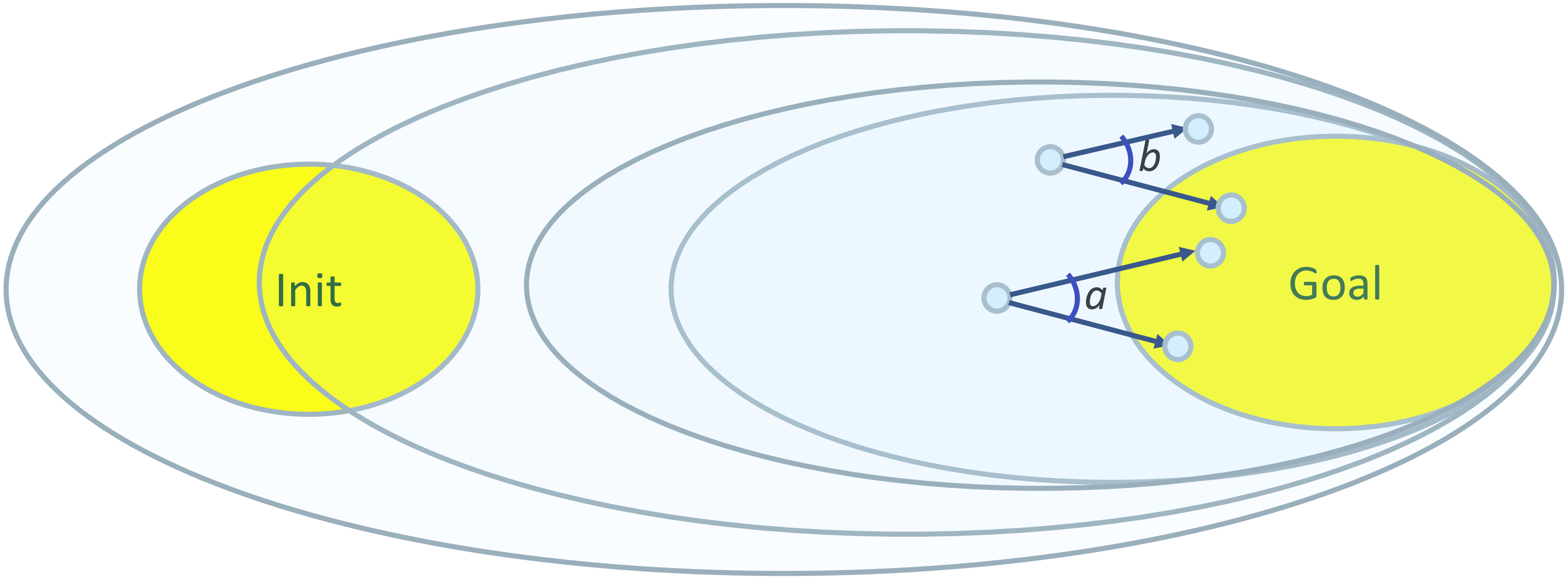
And/Or Graph Search

Symbolic Model Checking

Determinization

Planning via Symbolic Model Checking

Safe Acyclic Solutions



$$\text{StrongPreImg}(S) \; = \; \{(s,a) \mid \gamma(s,a) \neq \emptyset \text{ and } \gamma(s,a) \subseteq S\}$$

Planning via Symbolic Model Checking

- Simple propositional formulas can represent very large sets of states

- Quantified Boolean Formulas can represent transitions

- BDD representation and manipulation of propositional formulas

Some Planning Techniques

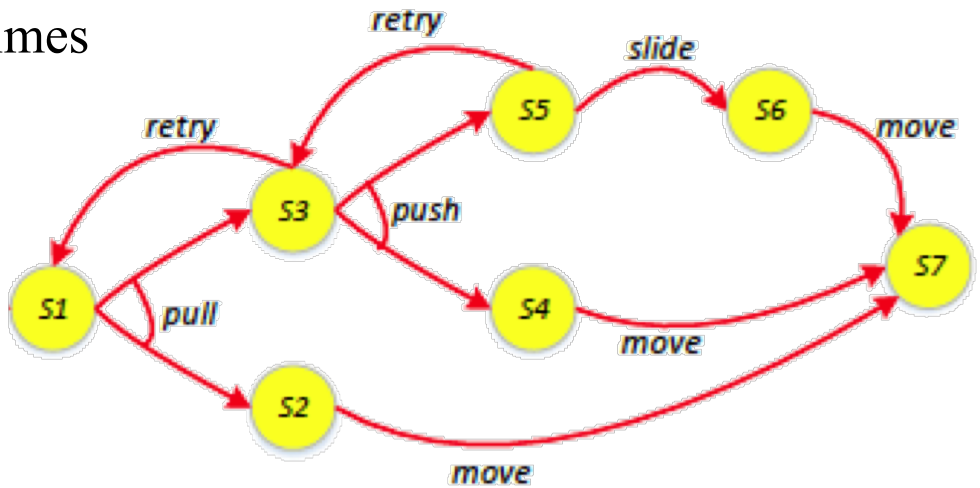
And/Or Graph Search

Symbolic Model Checking

Determinization

Motivation

- Much easier to find solutions if they don't have to be safe
  - Find-safe-solution needs plans for all possible outcomes
  - Find-solution only needs a plan for one of them
- Idea: call Find-solution multiple times



Guided-Find-Safe-Solution

find a solution π

for each leaf node of π

if the leaf node isn't a goal then

find a solution and incorporate it into π

$\pi(s_1)$ = pull
$\pi(s_2)$ = move
next, plan for s_3

- (some additional details needed to handle dead ends)

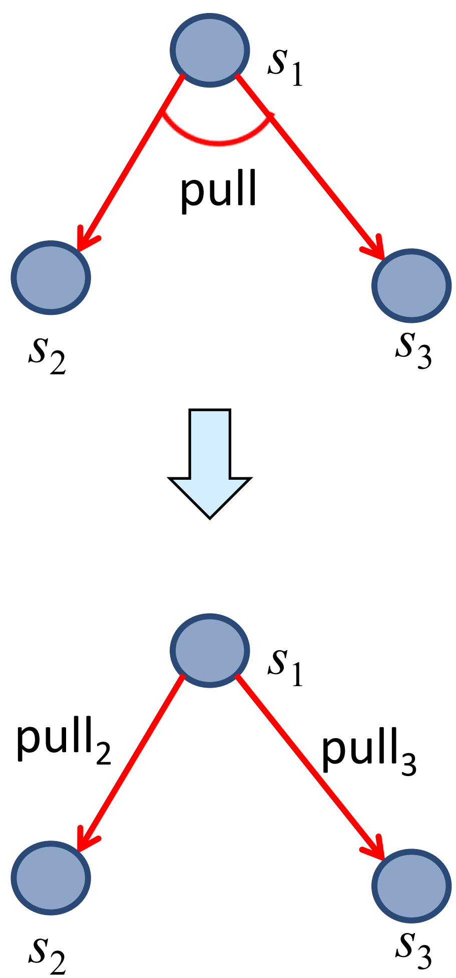
Determinization

- How to implement something like Guided-Find-Safe-Solution?
  - Need an implementation of Find-Solution
  - Need it to be very efficient
    - We'll call it many times

- Idea: instead of using Find-Solution, use a classical planner
  - Can use all the work on doing fast classical planning
    - efficient algorithms, search heuristics

Determinization

- Need to convert the nondeterministic actions into something the classical planner can use
- *Determinize* them
  - Suppose a_i has n possible outcomes
  - n deterministic actions, one for each outcome



- Classical planner returns a plan $p = \langle a_1, a_2, \ldots, a_n \rangle$
- If p is acyclic, can convert it to an (unsafe) solution
  - $\{(s_0,\boldsymbol{a}_1), (s_1,\boldsymbol{a}_2), \ldots, (s_{n-1},\boldsymbol{a}_n)\rangle$

    where
    - each s_i is the state produced by $\langle a_1,\ldots, a_i \rangle$
    - each $\boldsymbol{a}_i$ is the nondeterministic action whose determinization includes a_i

Determinization

Find-Safe-Solution-by-Determinization

find classical solution p

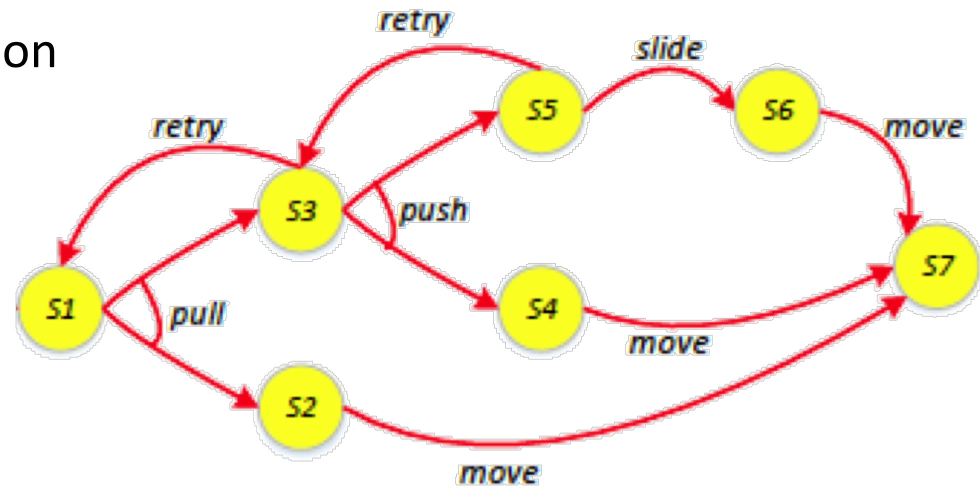
translate to policy π

for each leaf node of π

find classical solution

translate it to a policy

incorporate the policy into π



$p = \langle \text{pull}_2, \text{move} \rangle$ →

$\pi(s_1) = \text{pull}$

$\pi(s_2) = \text{move}$

next, plan for s_3

- (some additional details needed to handle dead ends)

Outline

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

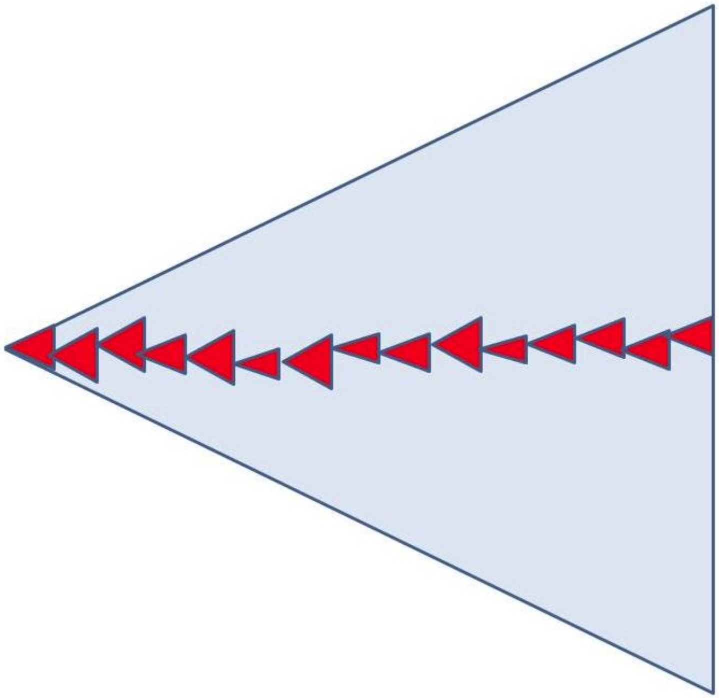
On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

Online Approaches

- Motivation
  - Planning models are approximate – execution seldom works out as planned
  - Large problems may require too much planning time
- 2nd motivation even more stronger in nondeterministic domains
  - Nondeterminism makes planning exponentially harder
    - Exponentially more time, exponentially larger policies



Offline vs Runtime Search Spaces

Online Approaches

- Need to identify *good* actions without exploring entire search space
  - Can be done using heuristic estimates
- Some domains are *safely explorable*
  - Safe to create partial plans, because goal states are reachable from all situations
- Other domains contain dead-ends, partial planning won't guarantee success
  - Can get trapped in dead ends that we would have detected if we had planned fully
    - No applicable actions
      - robot goes down a steep incline and can't come back up
    - Applicable actions, but caught in a loop
      - robot goes into a collection of rooms from which there's no exit
  - However, partial planning can still make success more likely

Lookahead-Partial-Plan

- Like Run-Lazy-Lookahead (Part 2)

- Lookahead is any planning algorithm that returns a policy π
  - π may be partial solution, or unsafe solution
  - Lookahead-Partial-Plan executes π as far as it will go, then calls Lookahead again

```
Lookahead-Partial-Plan(Σ, s₀, S_g)
    s ← s₀
    while s ∉ S_g and Applicable(s) ≠ ∅ do
        π ← Lookahead(s, θ)
        if π = ∅ then return failure
        else do
            perform partial plan π
            s ← observe current state
```

FS-Replan

- Like Run-Lookahead (Part 2)
- Calls classical planner on determinized model, converts plan to policy
  - Unsafe solution

FS-Replan (Σ, s, S_g)
  $\pi_d \leftarrow \varnothing$
  while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do
    if π_d undefined for s then do
      $\pi_d \leftarrow$ Plan2policy(Forward-search$(\Sigma_d, s, S_g), s)$
      if $\pi_d =$ failure then return failure
    perform action $\pi_d(s)$
    $s \leftarrow$ observe resulting state

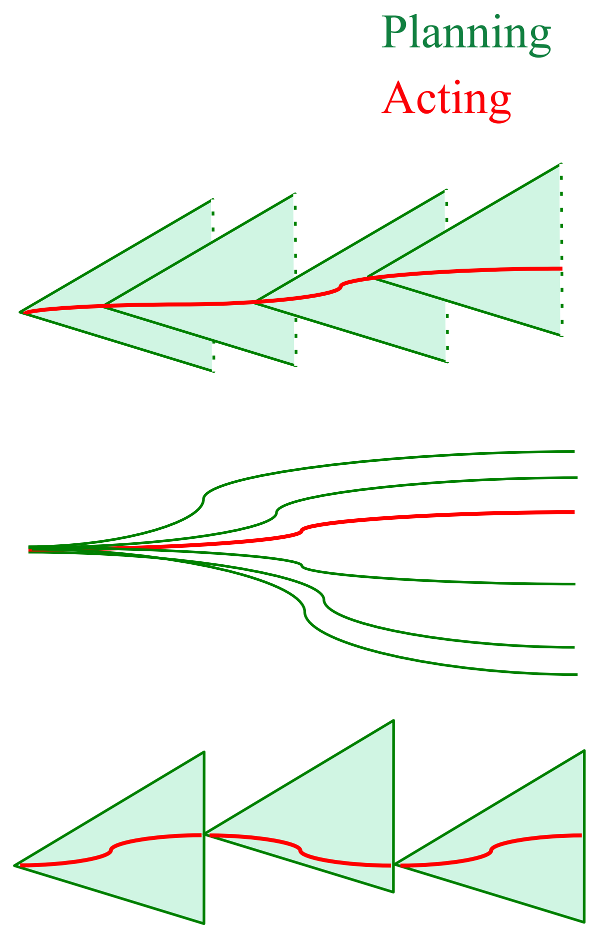
- Generalization:
  - Lookahead can be any planning algorithm that returns a policy π

FS-Replan (Σ, s, S_g) *(generalized)*
  $\pi_d \leftarrow \varnothing$
  while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do
    if π_d undefined for s then do
      $\pi_d \leftarrow$ Lookahead(s,θ)
      if $\pi_d =$ failure then return failure
    perform action $\pi_d(s)$
    $s \leftarrow$ observe resulting state

Possibilities for Lookahead

- Lookahead could be one of the algorithms we discussed earlier

  Find-Safe-Solution
  Find-Acyclic-Solution
  Guided-Find-Safe-Solution
  Find-Safe-Solution-by-Determinization



- What if it doesn't have time to run to completion?
  - Use the same techniques we discussed earlier
    - Receding horizon
    - Sampling
    - Subgoaling
    - Iterative deepening
  - Another technique

Possibilities for Lookahead

- *Full horizon, limited breadth:*
  - look for solution that works for *some* of the outcomes
  - E.g., modify Find-Acyclic-Solution to examine *i* outcomes of every action
- *Iterative broadening*:

  for i = 1 by 1 until time runs out

  look for a solution that handles i outcomes per action

Find-Acyclic-Solution (Σ, s_0, S_g)

$\pi \leftarrow \varnothing$

$Frontier \leftarrow \{s_0\}$

for every $s \in Frontier \setminus S_g$ do

  $Frontier \leftarrow Frontier \setminus \{s\}$

  if Applicable$(s) = \varnothing$ then return failure

  nondeterministically choose $a \in$ Applicable(s)

  $\pi \leftarrow \pi \cup (s, a)$

  $Frontier \leftarrow Frontier \cup$ {i elements of $\gamma(s,a) \setminus \text{Dom}(\pi)$}

  if has-loops$(\pi, a, Frontier)$ then return failure

return π

The UCT algorithm for Monte-Carlo rollouts is a kind of iterative broadening where i differs at each node

Online Approaches

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

$a \leftarrow \text{argmin}_{a \in \text{Applicable}(s)} \max_{s' \in \gamma(s,a)} h(s')$

$h(s) \leftarrow \max\{h(s), 1 + \max_{s' \in \gamma(s,a)} h(s')\}$

perform action a

$s \leftarrow$ the current state

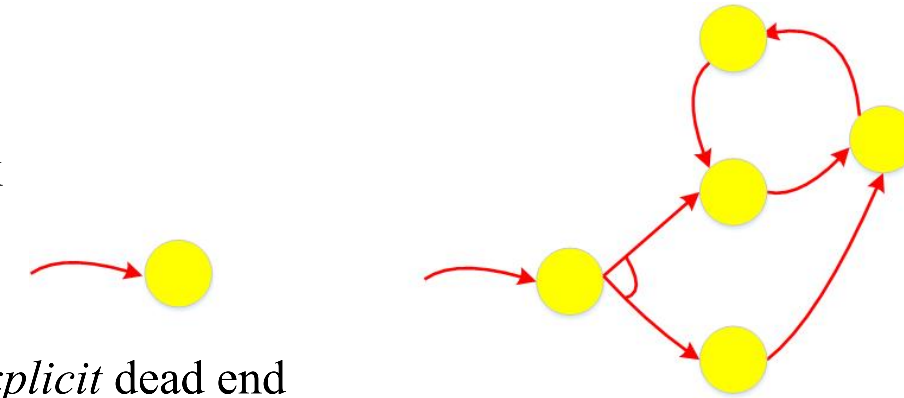
Assumes each action has cost 1
Can easily be modified to use cost $\neq 1$

- loop
  - choose an action a that (according to h) has least worst-case cost
    - Update $h(s)$ to use a's worst-case cost
    - Perform a

- In safely explorable domains with no "unfair" executions, guaranteed to reach a goal

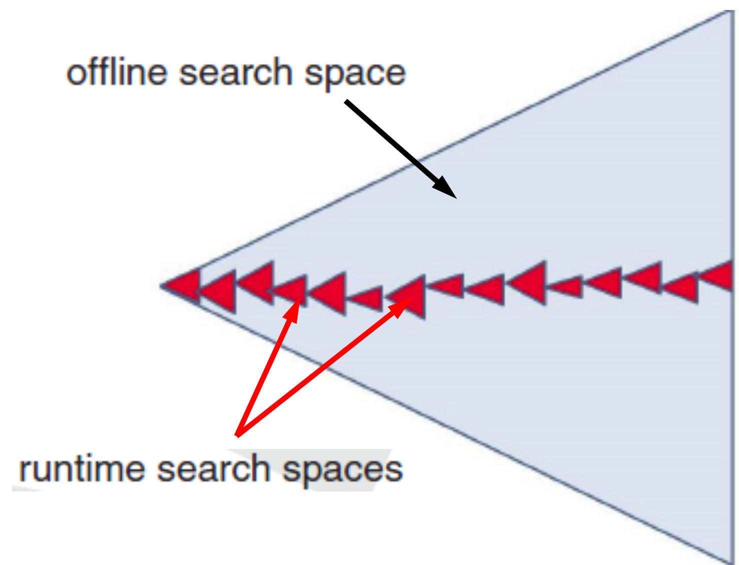
Online Approaches

- Critical issue: *dead ends*
  - Possibility of getting stuck during acting



explicit dead end

implicit dead end



- Completeness only in domains that are *safely explorable*
  - At every state, ∃ a path to the goal

Outline

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

Acting by interactions

How can I open you?

I am a sliding door

Please give me your opening instructions

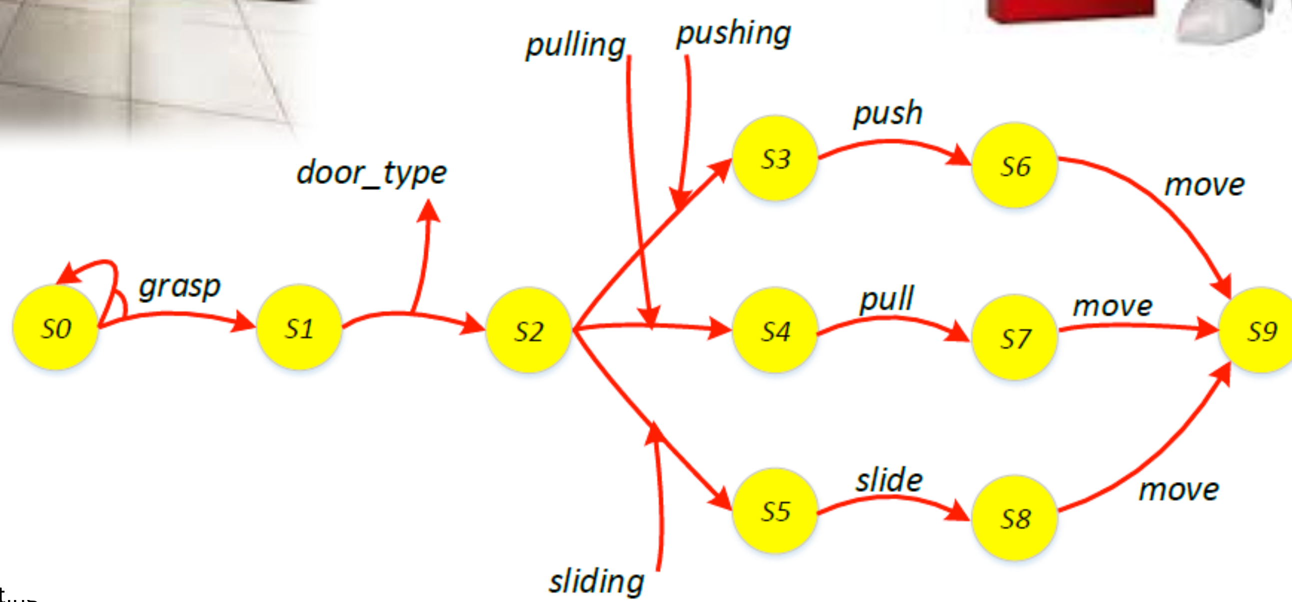
Door: opening module

⋮

I/O Automata

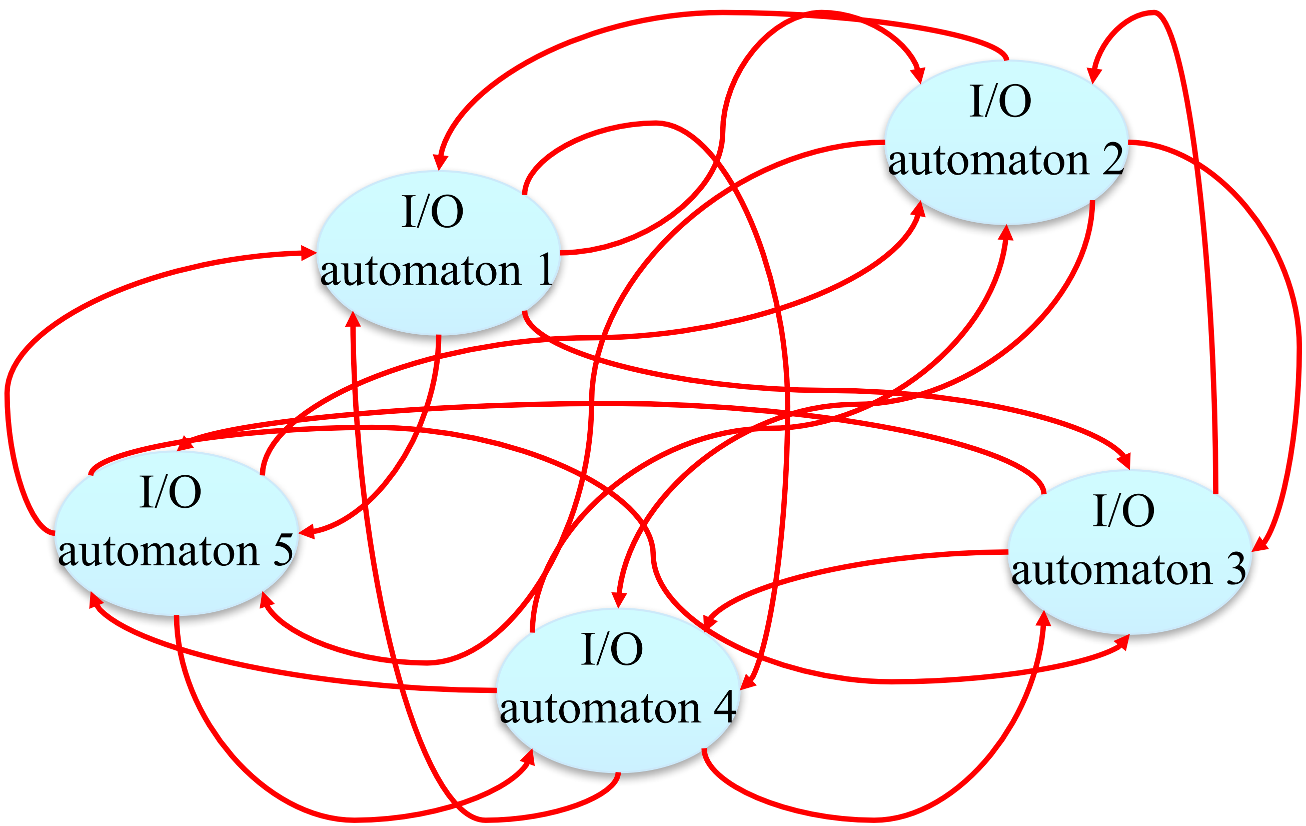
What type of door are you?

I am a sliding door



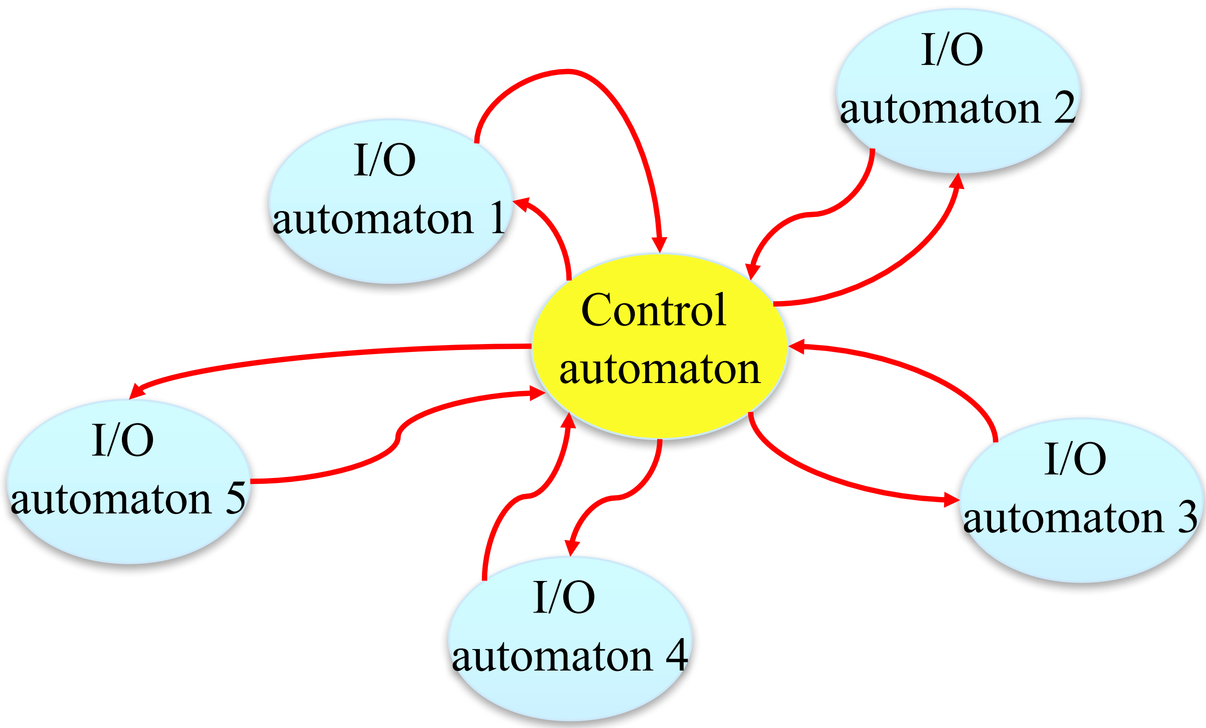
Need to Coordinate the Interactions

- Collection of I/O automata
  - Can't just start them running and expect it to work
  - Need to control *which* automata interact, *how*, in *what* circumstances
- Need a *control automaton*

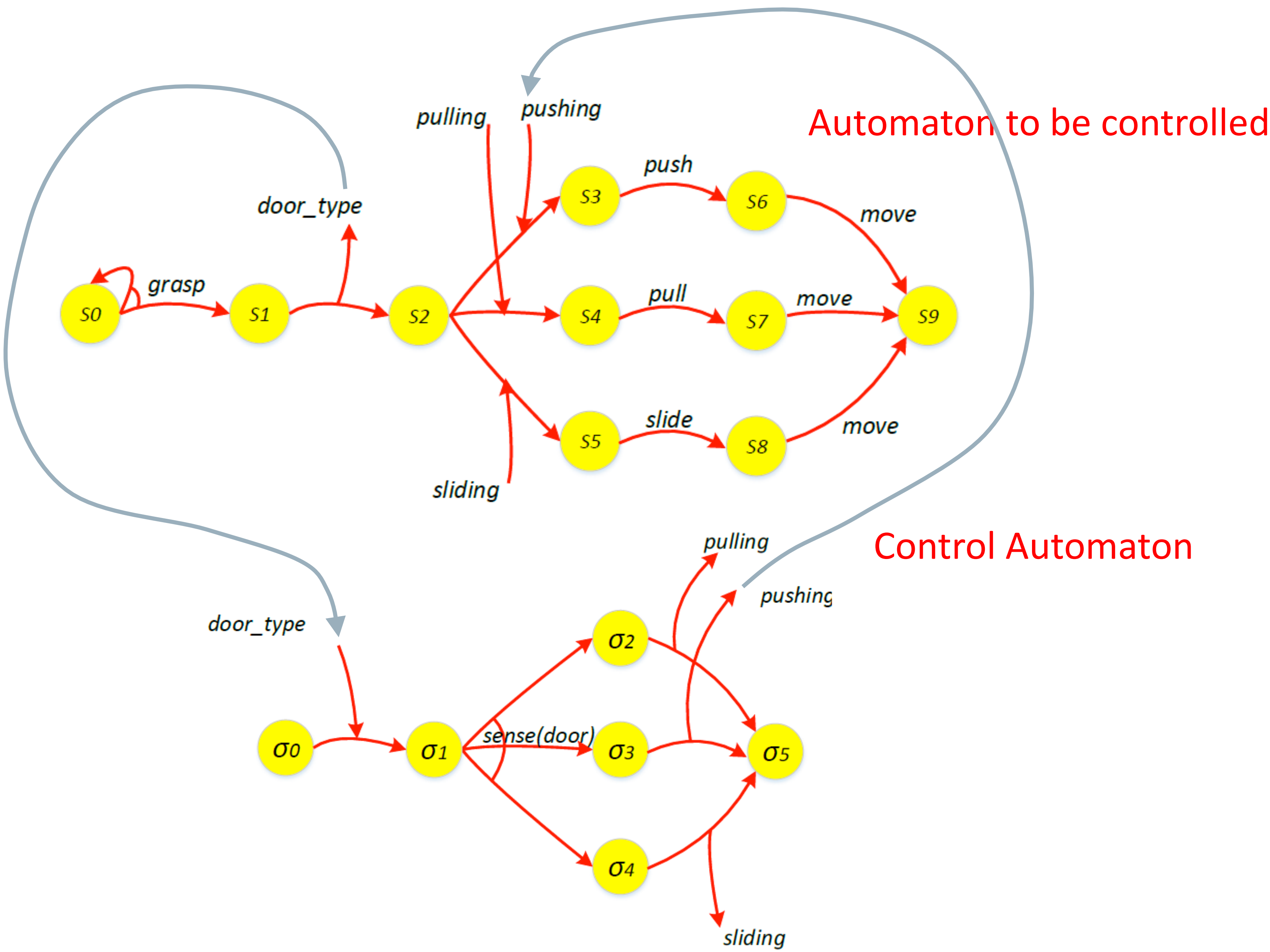


Need to Coordinate the Interactions

- Collection of I/O automata
  - Can't just start them running and expect it to work
  - Need to control *which* automata interact, *how*, in *what* circumstances
- Need a *control automaton*

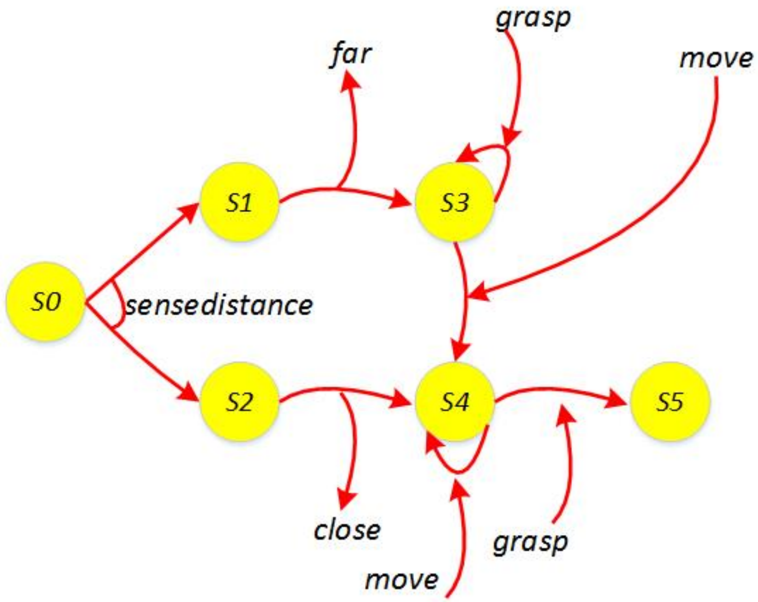


Trivial Example

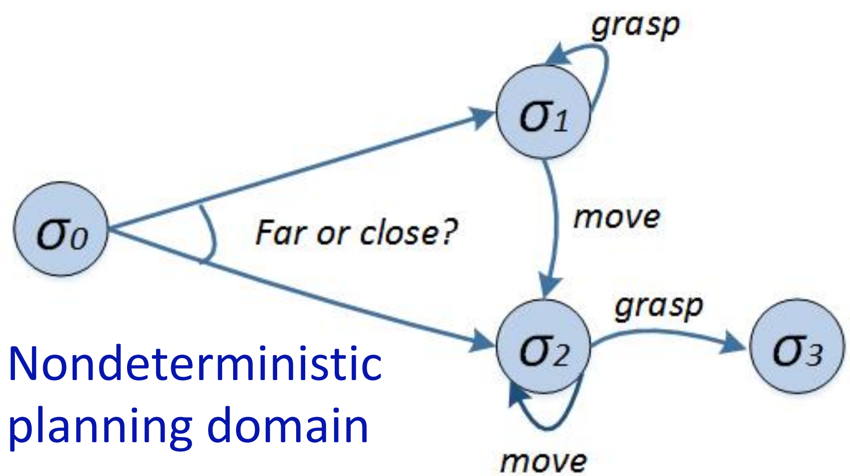


Synthesizing Control Automata

- Can synthesize control automata using nondeterministic models
  - Transform I/O automaton into a nondeterministic planning domain
  - Use and/or graph search, symbolic model checking, determinization



Automaton to be controlled



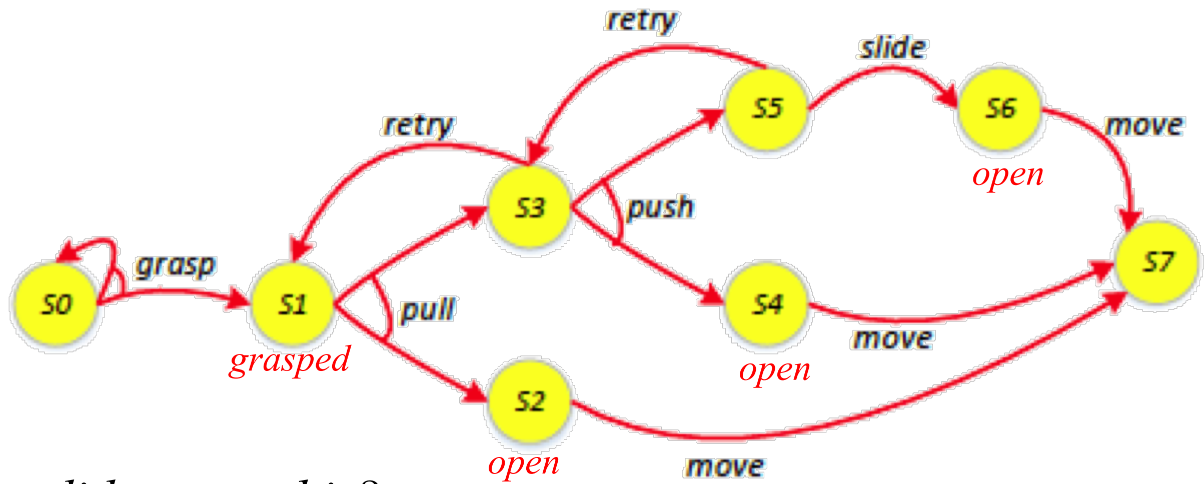
Nondeterministic planning domain

Control policy

$$\pi(\sigma_0) = \text{farorclose}$$
$$\pi(\sigma_1) = \text{move}$$
$$\pi(\sigma_2) = \text{grasp}$$

Research Challenge

m-opendoor(r, d, l, o)
task: opendoor(r, d)
pre: $\text{loc}(r) = l \wedge \text{adjacent}(l, d) \wedge \text{handle}(d, o)$
body: while $\neg$grasped(d) d
grasp(r, d)
pull(r, d)
if door-status(d)=ope
else pull-push(r, d)



How did we get this?

How do we do it in general?

m-retry-pull(r, d, l, o)
task: pull-push(r, d)
body: pull(r, d);
if door-status(d)=open then move(r, d)
else pull-push(r, d)

task: pull-push(r, d)
body: push(r, d)
if door-status(d)=open then move(r, d)
else push-slide(r, d)

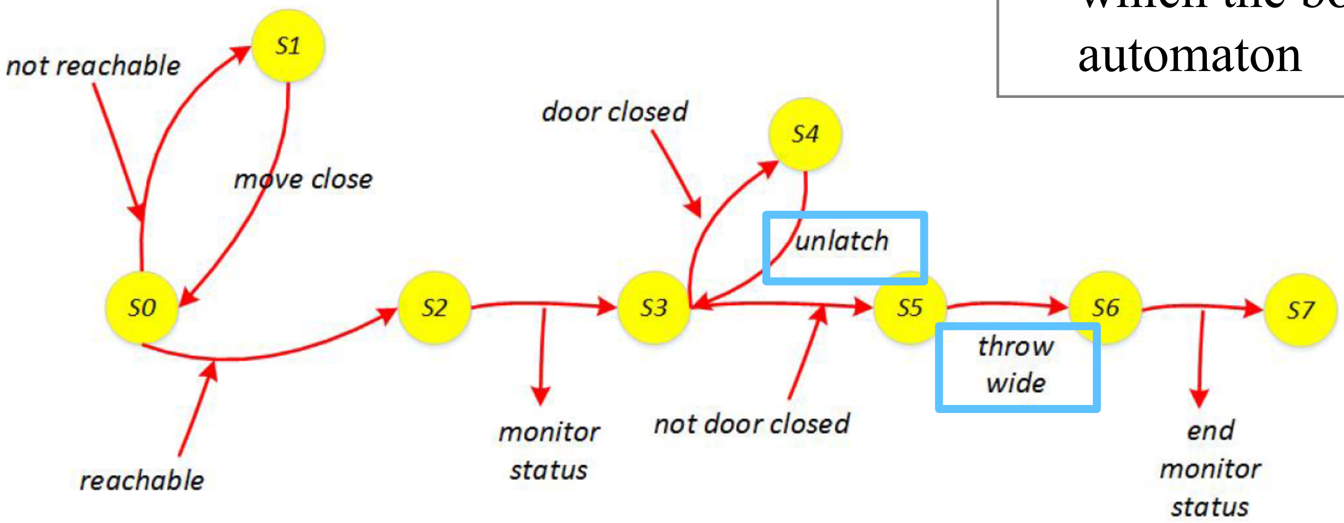
m-retry-push(r, d, l, o)
task: push-slide(r, d)
body: push(r, d);
if door-status(d)=open then move(r, d)
else push-slide(r, d)

m-slide(r, d, l, o)
task: push-slide(r, d)
body: slide(r, d)

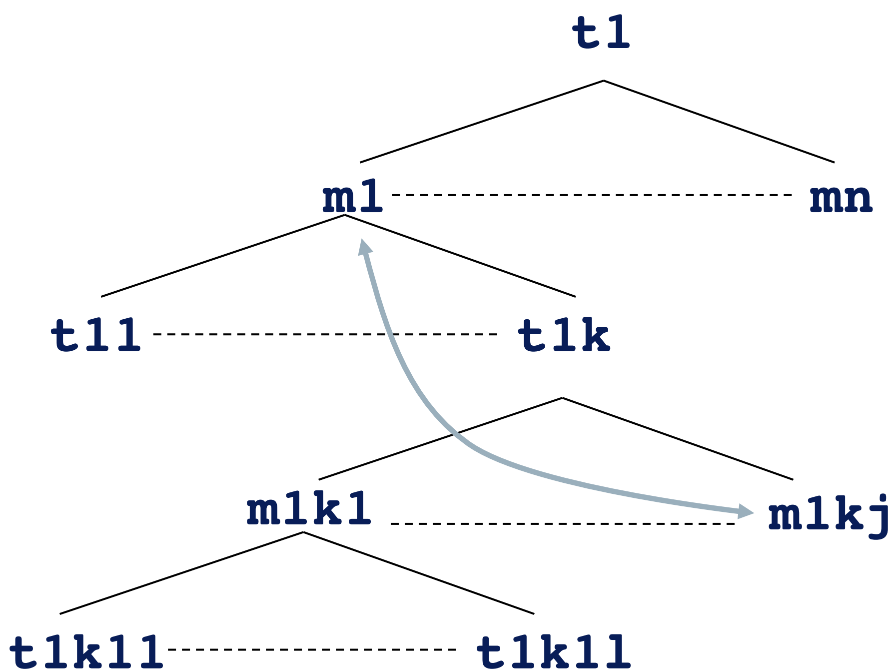
Refining and Controlling I/O Automata

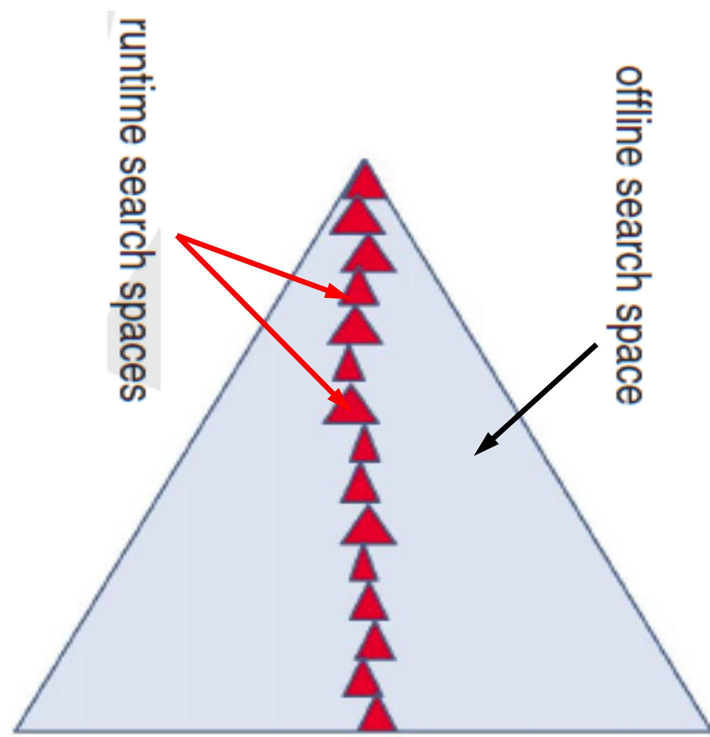
$\boxed{\texttt{m-opendoor}(r,d,l,o)}$
task: $\texttt{opendoor}(r,d)$
pre: $\texttt{loc(r)} = \texttt{l} \wedge \texttt{adjacent(l,d)} \wedge \texttt{handle(d,o)}$
body: while $\neg\texttt{reachable}(r,o)$ do
  $\texttt{move-close}(r,o)$
  $\texttt{monitor-status}(r,d)$
  if $\texttt{door-status}(d)\texttt{=closed}$ then $\boxed{\texttt{unlatch}(r,d)}$
  $\boxed{\texttt{throw-wide}(r,d)}$
  $\texttt{end-monitor-status}(r,d)$

- Refinement methods in which the body is an I/O automaton



Refining and Controlling I/O Automata





- Work in progress
  - Sunandita Patra *et al.*, GenPlan workshop

Summary

- Use nondeterministic models?
  - Sometimes a design choice
  - Sometimes a must
- Nondeterministic planning problems
  - types of solutions
    - unsafe, acyclic safe, cyclic safe
  - planning algorithms for each
  - determinization techniques
- Online approaches
  - ways to do the lookahead
- Controlling the interactions among multiple actors
  - I/O automata

Relation to the Book

Any questions?

- Ghallab, Nau, and Traverso (2016). *Automated Planning and Acting.* Cambridge University Press
- Free downloads:
  - Lecture slides, final manuscript
  - http://www.laas.fr/planning
- Table of Contents
  1. Introduction
  2. Deterministic Models
  3. Refinement Methods
  4. Temporal Models
  5. Nondeterministic Models
  6. Probabilistic Models
  7. Other Deliberation Functions