FONDAZIONE
BRUNO KESSLER

Deliberation with Nondeterministic Models

Malik Ghallab, Dana Nau, Paolo Traverso
Automated Planning and Acting
Cambridge University Press

IJCAI 2016 Tutorial
New York, July 11th,2016

FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

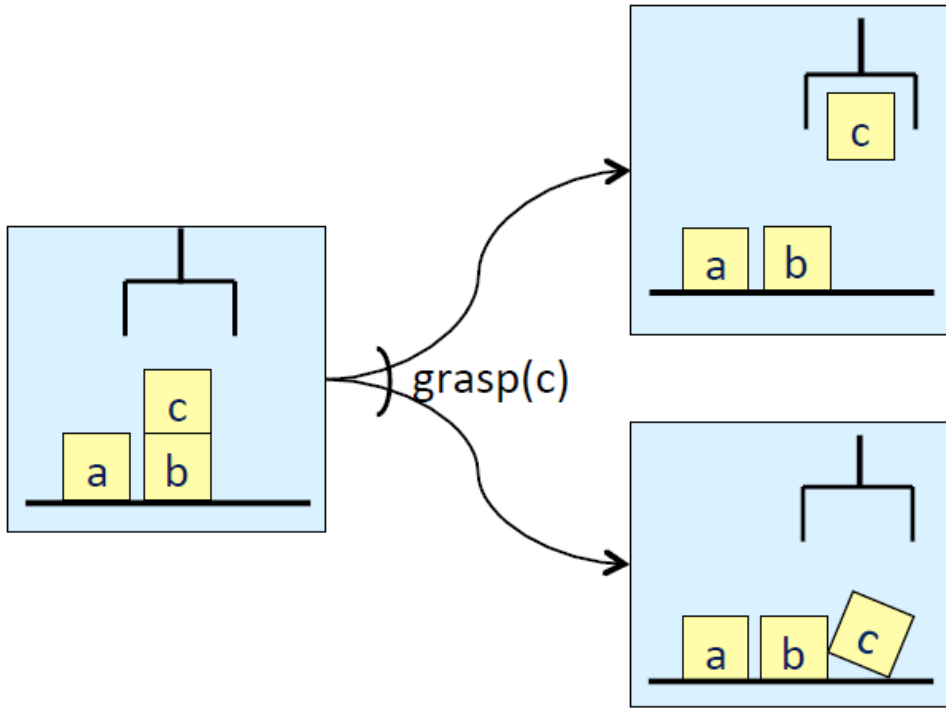
On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

Introduction & Motivation

- Actions are modeled with more than one possible outcomes
- In some cases this is a design choice



Introduction & Motivation

- Actions are modeled with more than one possible outcomes
- In some cases this is a design choice

Introduction & Motivation

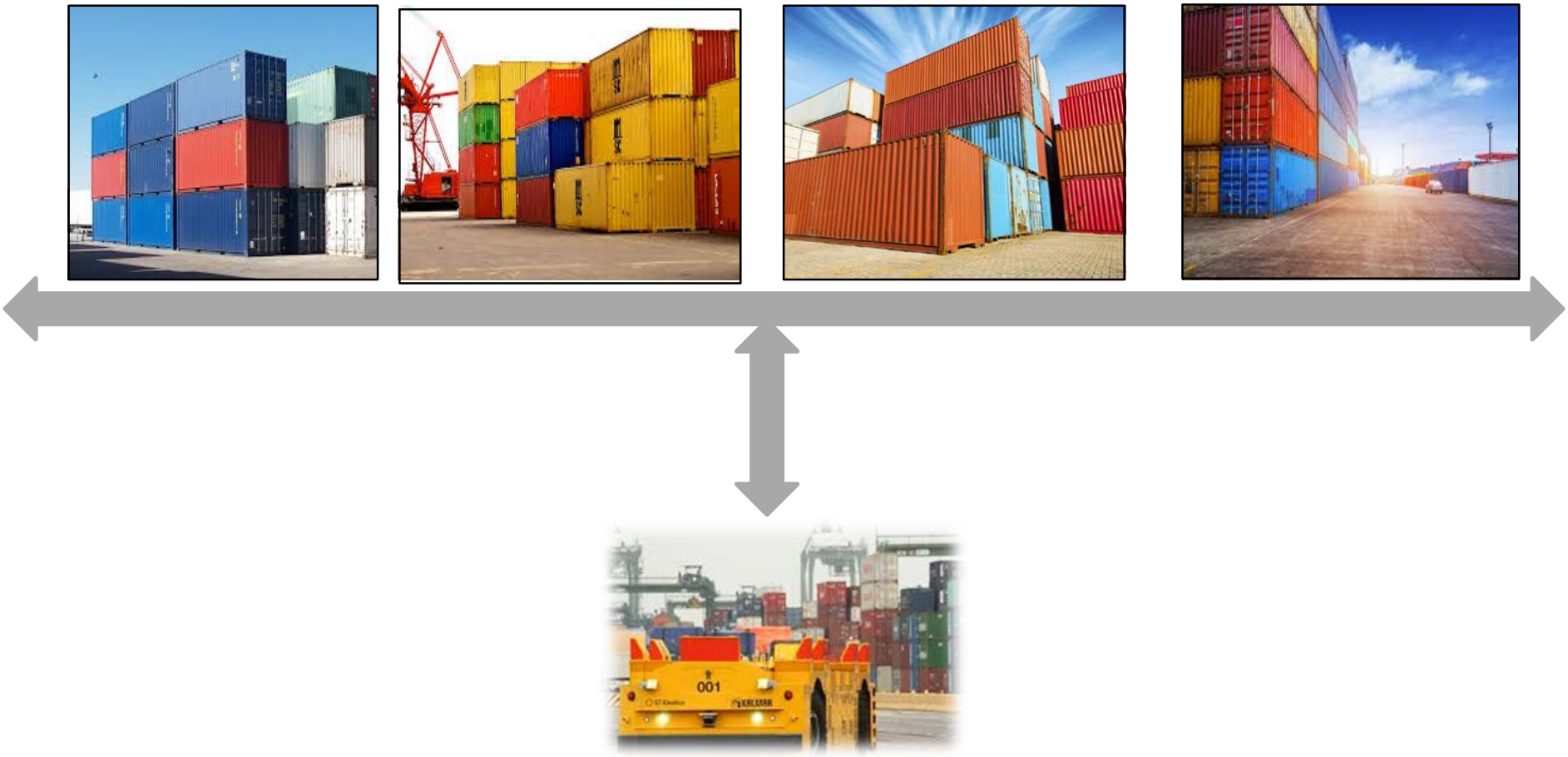
- Actions are modeled with more than one possible outcomes
- In some cases this is a design choice
- In other cases this is a must!

In some cases nondeterminism is a must!

- The PAMp example, “Automated Planning and Acting”, Chap 6, page 326

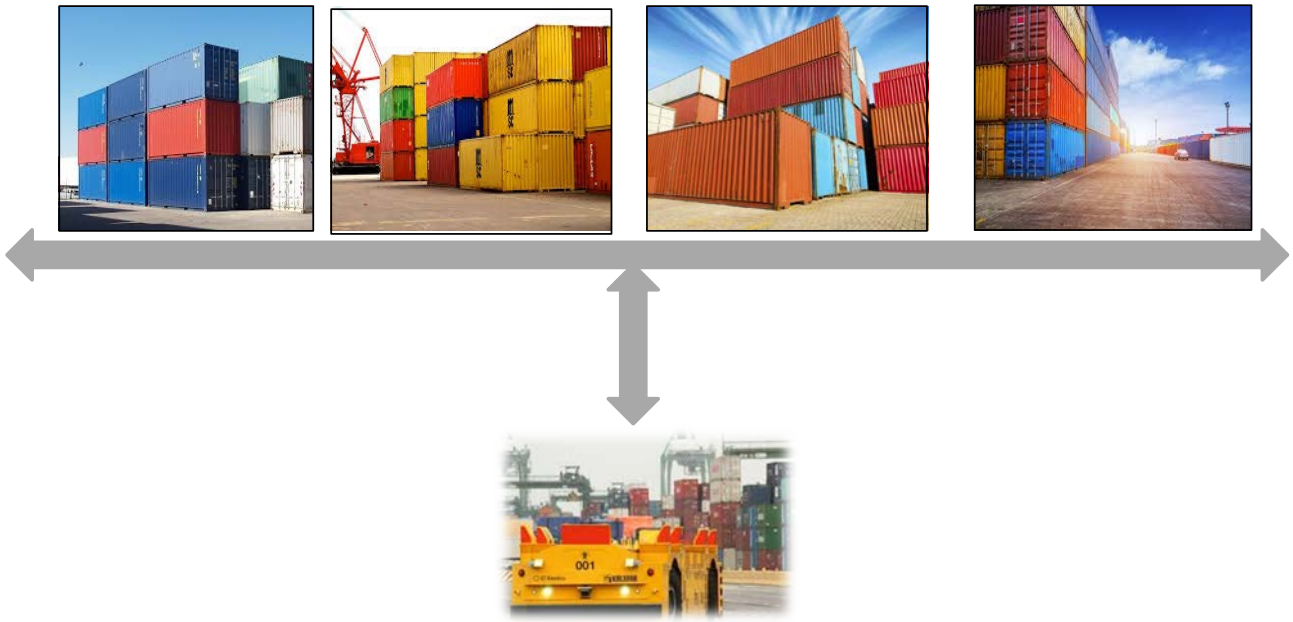
In some cases nondeterminism is a must!

- The PAMp example, “Automated Planning and Acting”, Chap 6, page 326



In some cases nondeterminism is a must!

- The PAMp example, “Automated Planning and Acting”, Chap 6, page 326
- Location can be busy, an exogenous event that is modeled with *switch(loc)*
- State space = 1.6 * 10^12 (in the case of 10 containers per location)



In some cases nondeterminism is a must!

- Different types of containers
- Sensing action *perceive(container)* lets the robot know the type
- There is no clear “nominal case”

In some cases nondeterminism is a must!

eni

saipem

FONDAZIONE
BRUNO KESSLER

In some cases nondeterminism is a must!

Nondeterminism in acting is a must!

FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

Nondeterministic models

RAE method for opening a door: pull, push, or slide?

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)

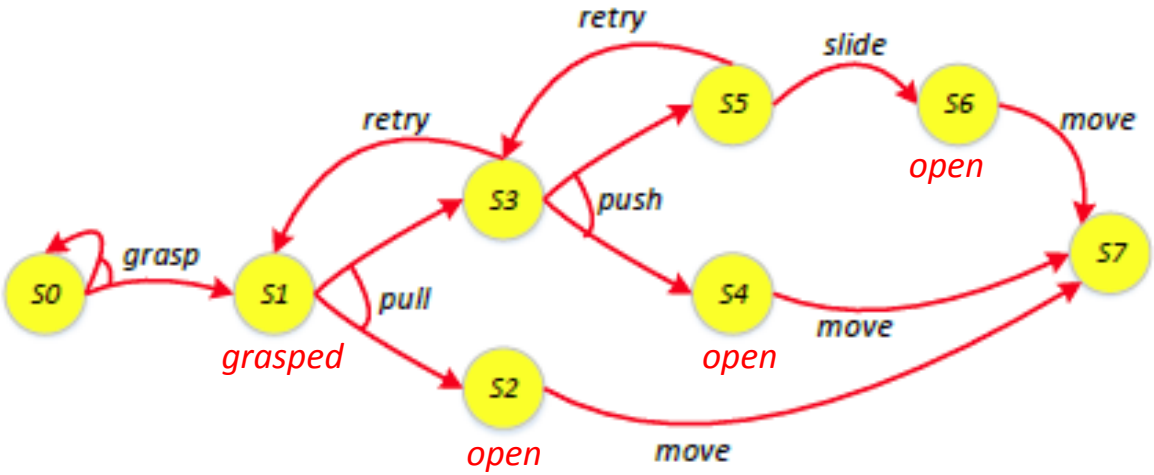
m-retry-pull(r, d, l, o)
  task: pull-push(r, d)
  body: pull(r, d);
  if door-status(d)=open then move(r, d)
  else pull-push(r, d)

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)

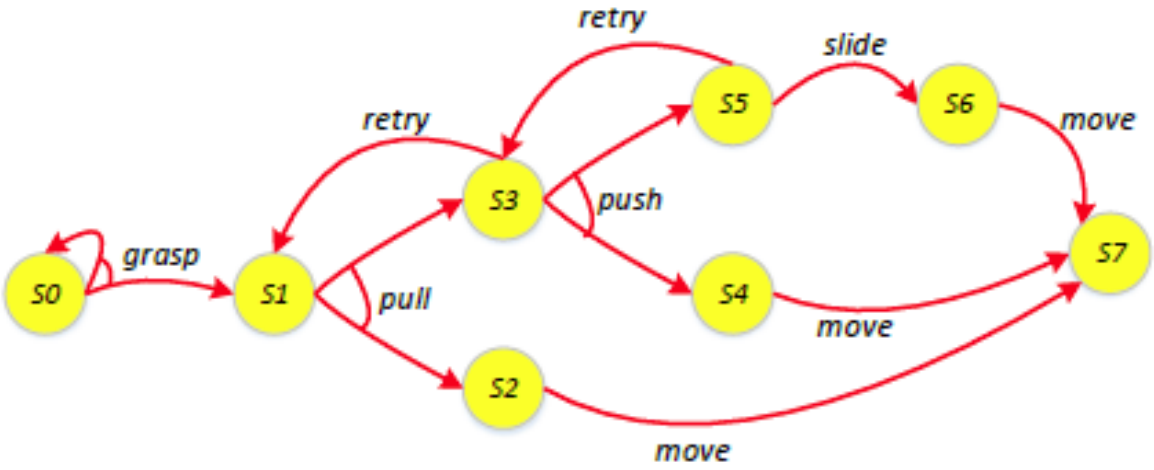
m-retry-push(r, d, l, o)
  task: push-slide(r, d)
  body: push(r, d);
  if door-status(d)=open then move(r, d)
  else push-slide(r, d)

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

Nondeterministic Models



Nondeterministic Models: Planning Domain

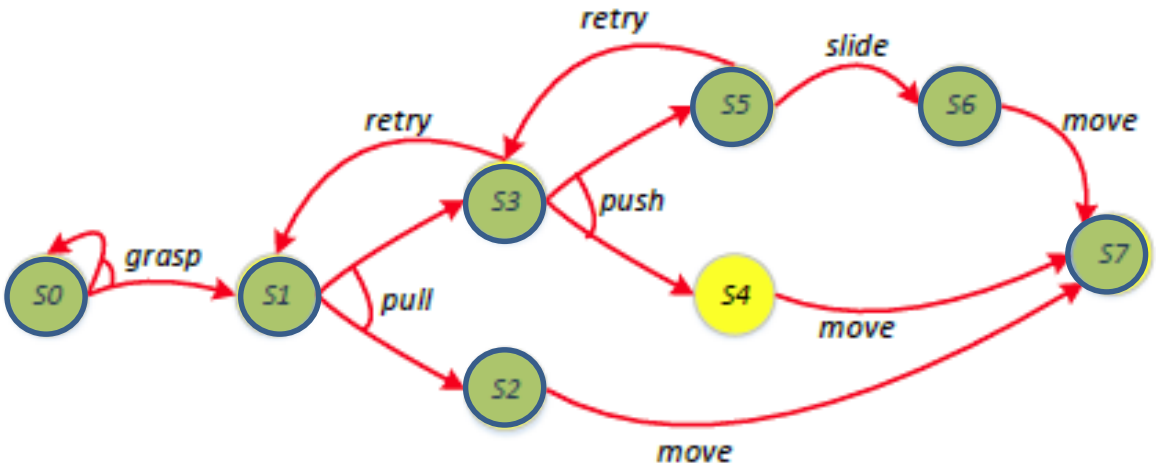


Definition 5.1. (Planning Domain) A *nondeterministic planning domain* Σ is the tuple (S, A, γ), where S is the finite set of states, A is the finite set of actions, and $\gamma : S \times A \to 2^S$ is the state transition function. □

An action $a \in A$ is applicable in state $s \in S$ if and only if $\gamma(s, a) \neq \varnothing$. Applicable($s$) is the set of actions applicable to state s:

$$\text{Applicable}(s) = \{a \in A \mid \gamma(s, a) \neq \varnothing\}$$

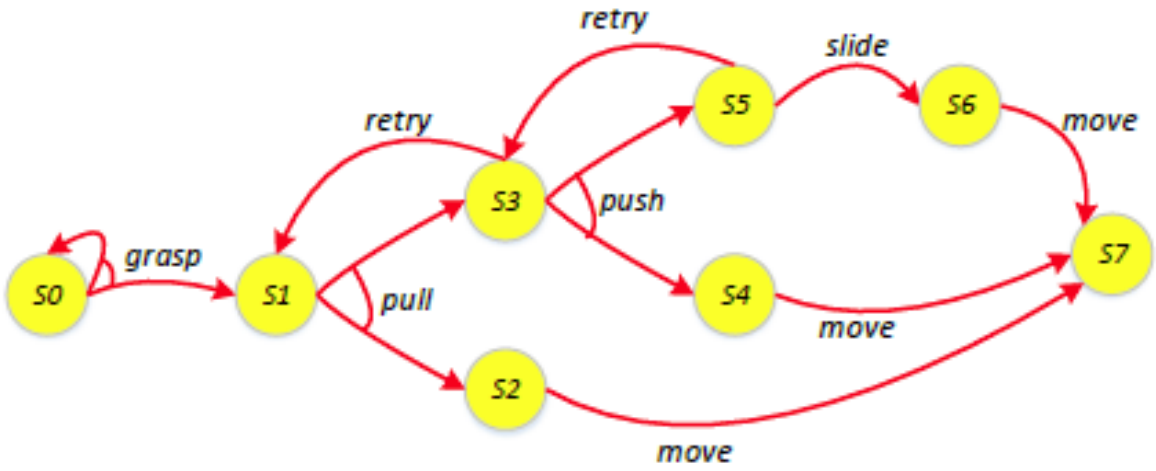
Plans: Sequential Plans?



⟨grasp; pull; move⟩

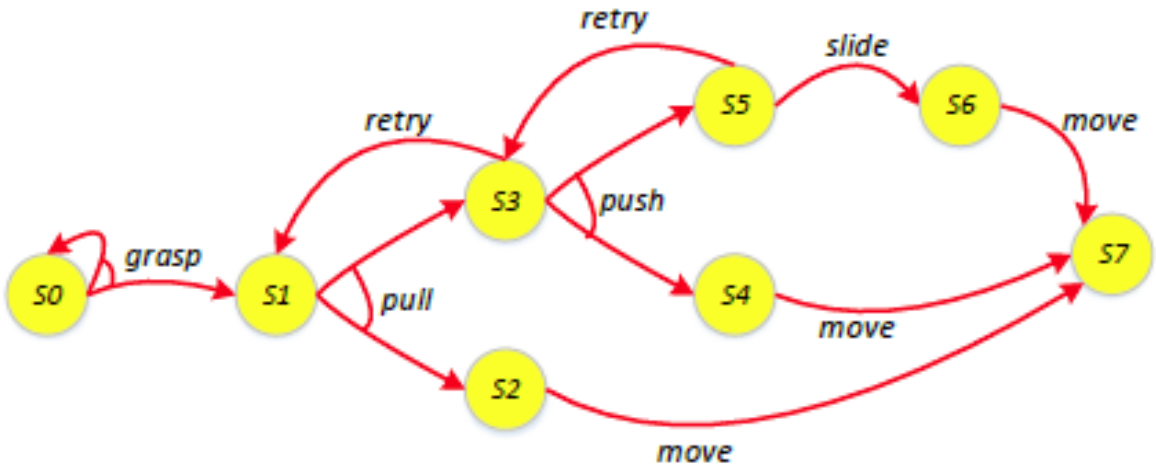
⟨grasp; pull; push; slide; move⟩

Plans = Policies



Definition 5.3. (Policy) Let $\Sigma = (S, A, \gamma)$ be a planning domain. Let $S' \subseteq S$. A *policy* π for a planning domain Σ is a function $\pi : S' \rightarrow A$ such that, for every $s \in S'$, $\pi(s) \in Applicable(s)$. It follows that $\mathrm{Dom}(\pi) = S'$. □

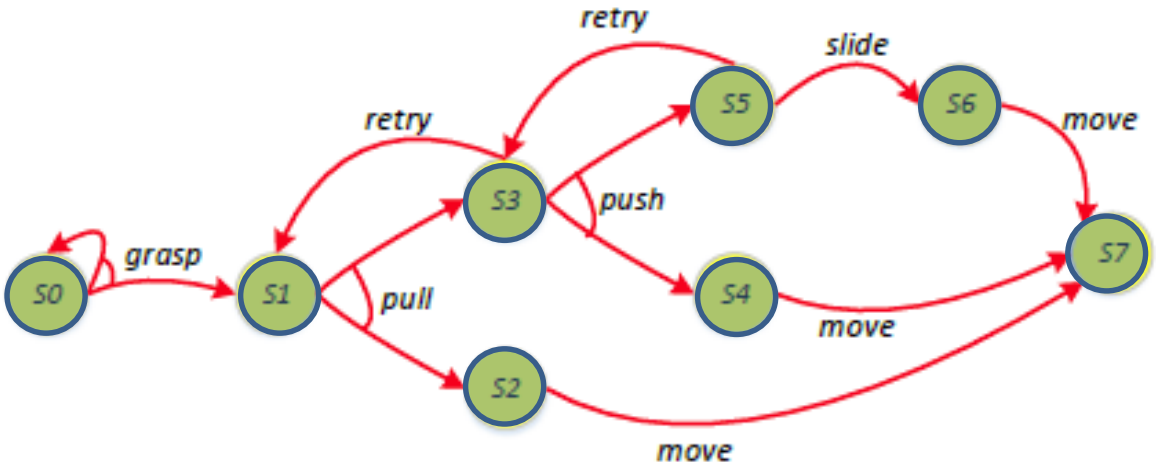
Plans = Policies



Algorithm 5.1 Procedure for performing the actions of a policy.

PerformPolicy(π)
  $s \leftarrow$ observe the current state
  while $s \in \text{Dom}(\pi)$ do
    perform action $\pi(s)$
    $s \leftarrow$ observe the current state

Plans = Policies



π_1 :

$\pi_1(s_0) = \text{grasp}$

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

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

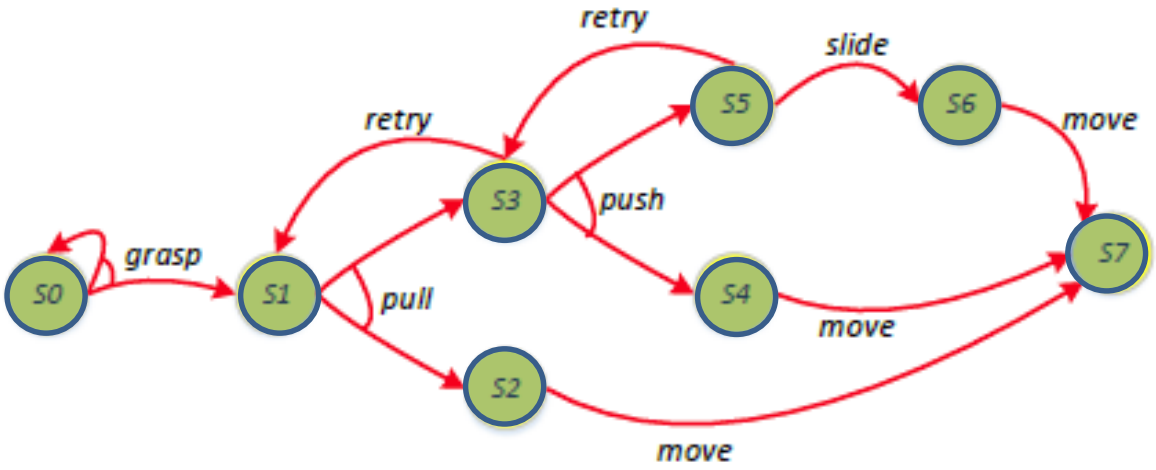
$\pi_1(s_3) = \text{push}$

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

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

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

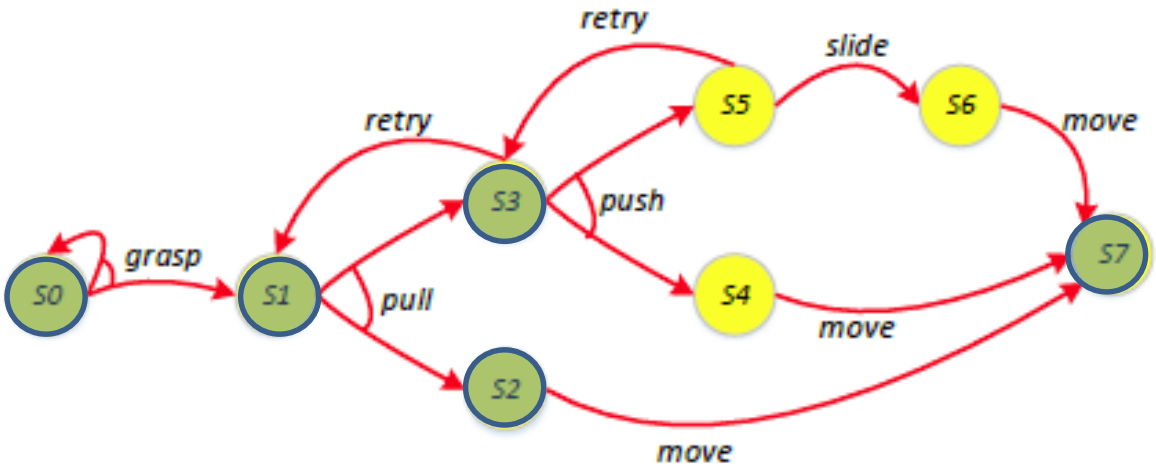
Plans = Policies



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

$$\pi_2: \quad \pi_2(s_0) = \text{grasp}$$
$$\pi_2(s_1) = \text{pull}$$
$$\pi_2(s_2) = \text{move}$$
$$\pi_2(s_3) = \text{retry}$$

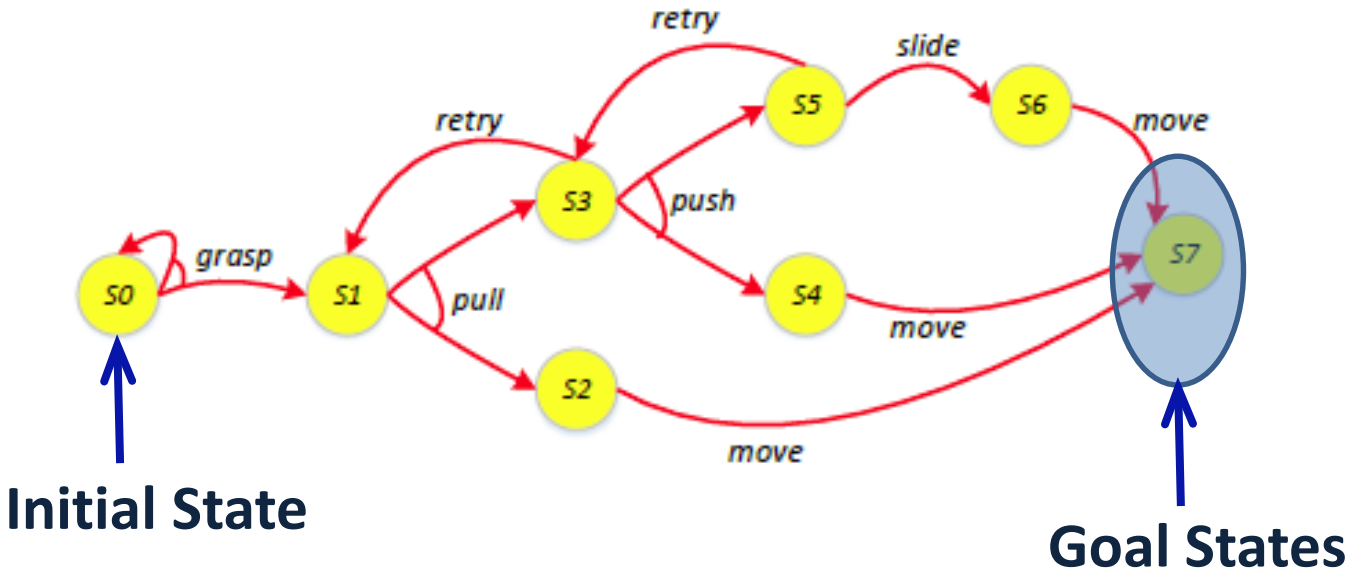
Plans = Policies



π_1 :
$\pi_1(s_0) = \text{grasp}$
$\pi_1(s_1) = \text{pull}$
$\pi_1(s_2) = \text{move}$
$\pi_1(s_3) = \text{push}$
$\pi_1(s_4) = \text{move}$
$\pi_1(s_5) = \text{slide}$
$\pi_1(s_6) = \text{move}$

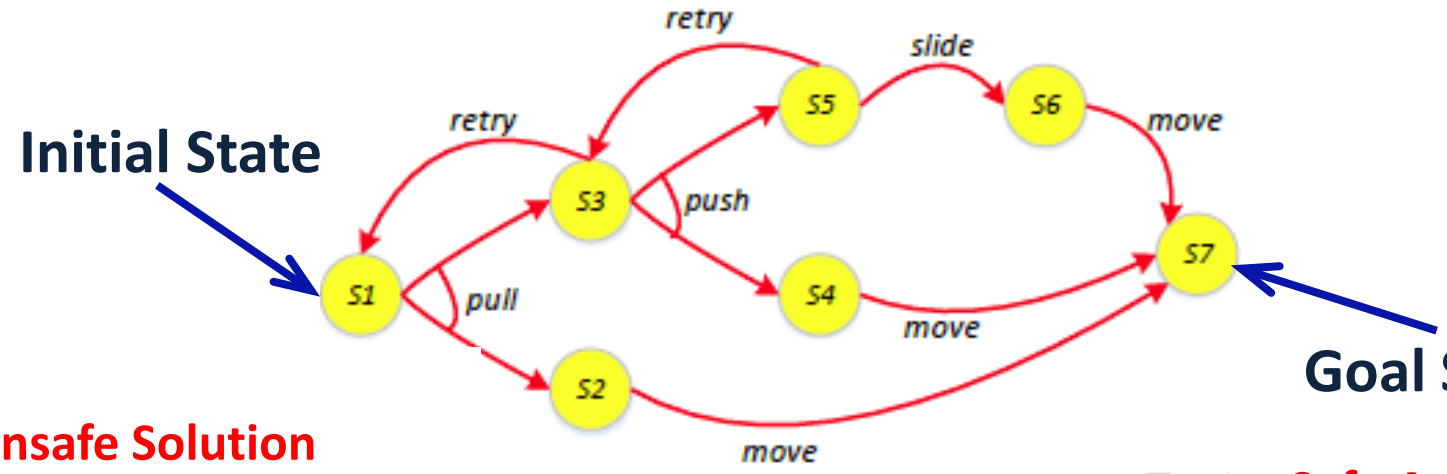
π_2 :
$\pi_2(s_0) = \text{grasp}$
$\pi_2(s_1) = \text{pull}$
$\pi_2(s_2) = \text{move}$
$\pi_2(s_3) = \text{retry}$

Planning Problems



Definition 5.6. (Planning Problem) Let $\Sigma = (S, A, \gamma)$ be a planning domain. A *planning problem* P for Σ is a tuple $P = (\Sigma, s_0, S_g)$, where $s_0 \in S$ is the initial state and $S_g \subseteq S$ is the set of goal states. □

Planning Problems and Solutions



Unsafe Solution

π_3 : $\pi_3(s_1) = \text{pull}$
$\pi_3(s_2) = \text{move}$

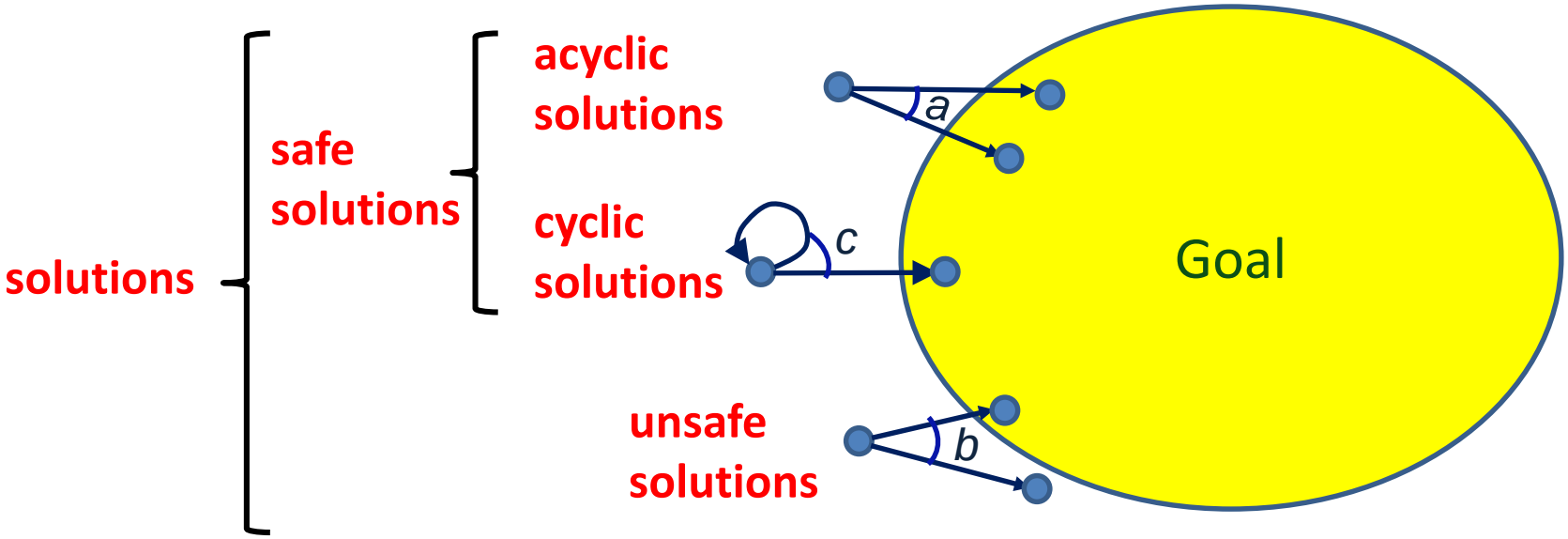
π_2 : **Safe Cyclic Solution**

$\pi_2(s_1) = \text{pull}$
$\pi_2(s_2) = \text{move}$
$\pi_2(s_3) = \text{retry}$

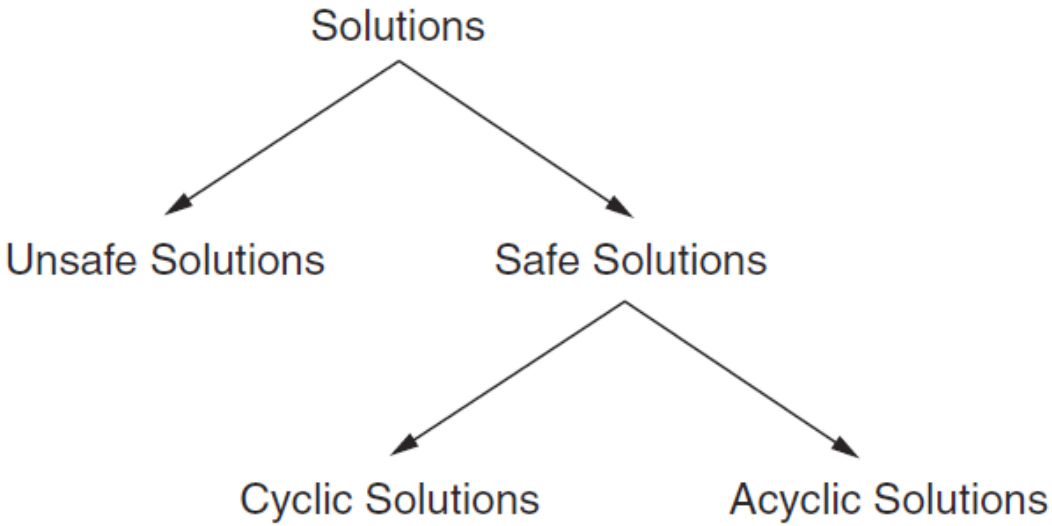
π_1 : **Safe Acyclic Solution**

$\pi_1(s_1) = \text{pull}$
$\pi_1(s_2) = \text{move}$
$\pi_1(s_3) = \text{push}$
$\pi_1(s_4) = \text{move}$
$\pi_1(s_5) = \text{slide}$
$\pi_1(s_6) = \text{move}$

The Planning Problem: 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

FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

FONDAZIONE BRUNO KESSLER

Some Planning Techniques

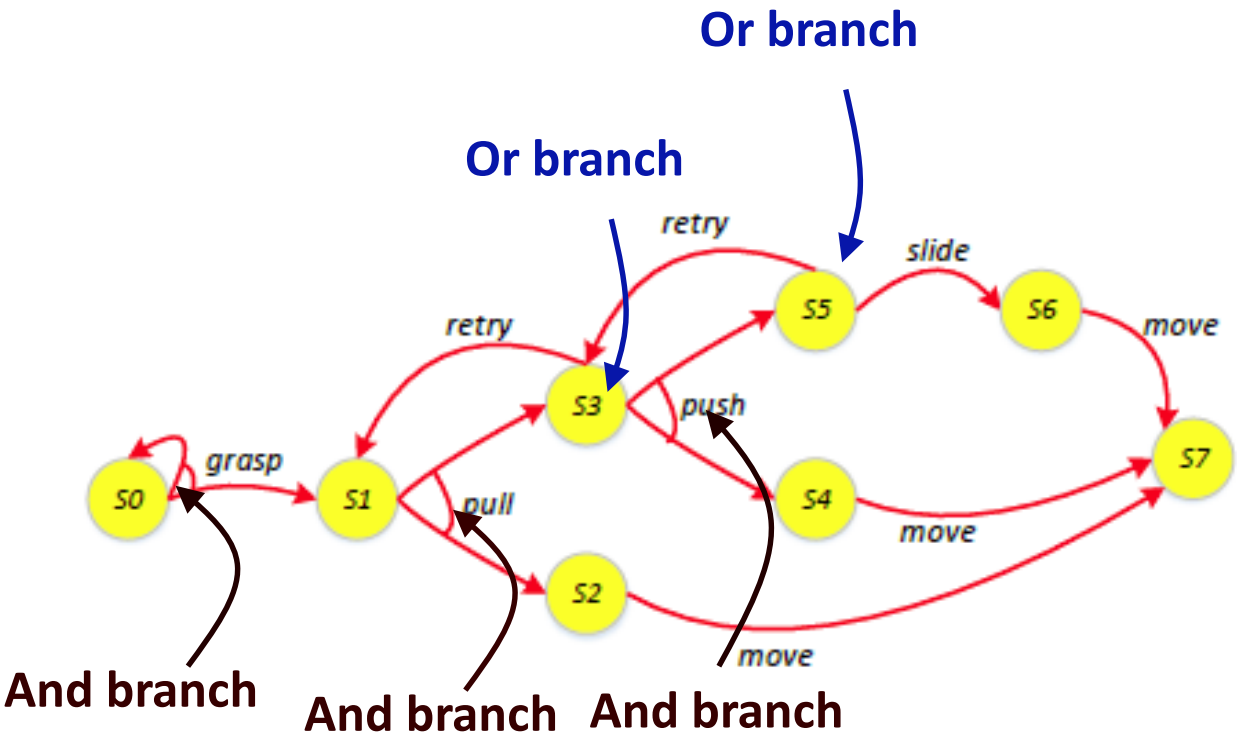
- **And/Or Graph Search**
- **Symbolic Model Checking**
- **Determinization**

FONDAZIONE BRUNO KESSLER

Some Planning Techniques

- **And/Or Graph Search**
- **Symbolic Model Checking**
- **Determinization**

And/Or Graphs



And/Or Graphs: Finding (Unsafe) Solutions

Find-Solution (Σ, s_0, S_g)

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

loop

if $s \in S_g$ then return π

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

if $A' = \varnothing$ then return failure

nondeterministically choose $a \in A'$

nondeterministically choose $s' \in \gamma(s, a)$

if $s' \in Visited$ then return failure

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

Decide which state to plan for

Algorithm 5.2: Planning for solutions by forward search.

And/Or Graphs: Finding Safe Solutions

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

nondeterministically choose $a \in$ 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

has-unsafe-loops$(\pi, a, Frontier)$ iff

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

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

And/Or Graphs: Finding Safe Acyclic Solutions

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)
  - $\pi \leftarrow \pi \cup (s, a)$
  - $Frontier \leftarrow Frontier \cup (\gamma(s, a) \setminus \text{Dom}(\pi))$
  - if has-loops$(\pi, a, Frontier)$ then return failure
- return π

has-loops$(\pi, a, Frontier)$ iff
$\exists s \in (\gamma(s, a) \cap \text{Dom}(\pi))$ such that $s \in \hat{\gamma}(s, \pi)$

Check whether π contains any cycles

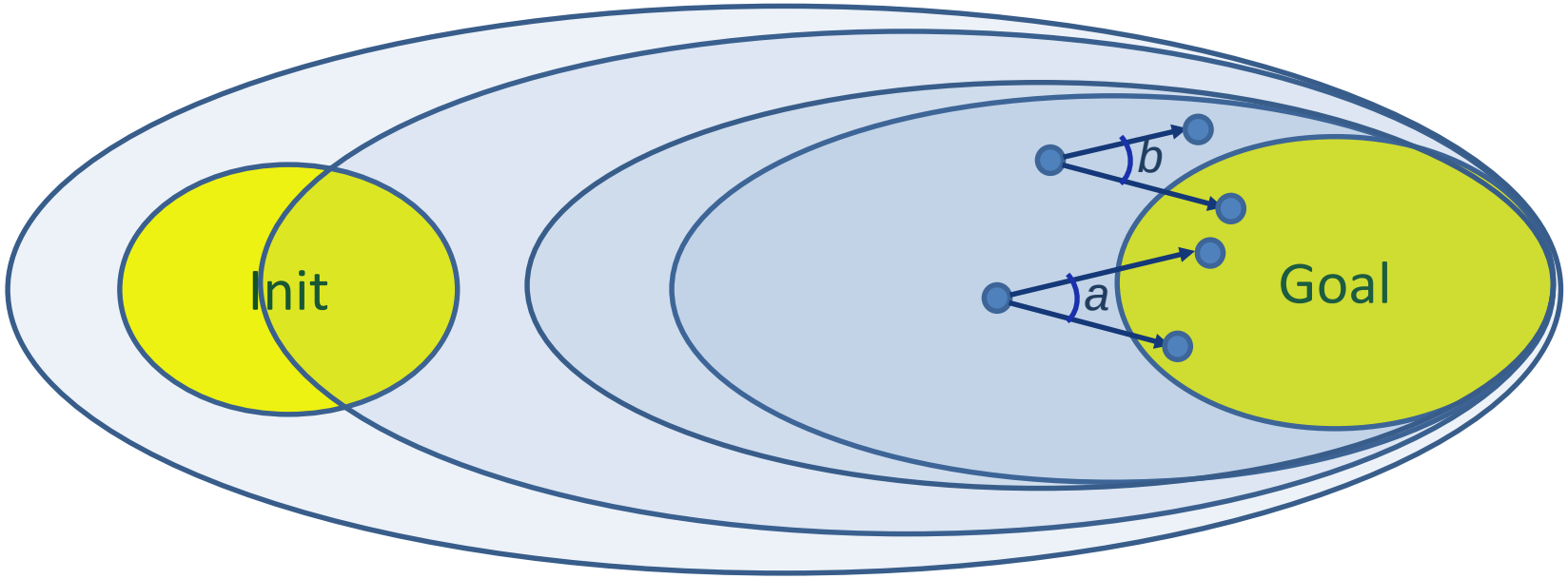
FONDAZIONE BRUNO KESSLER

Some Planning Techniques

- And/Or Graph Search
- Symbolic Model Checking
- Determinization

Planning via Symbolic Model Checking

Safe Acyclic Solutions



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

FONDAZIONE BRUNO KESSLER

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

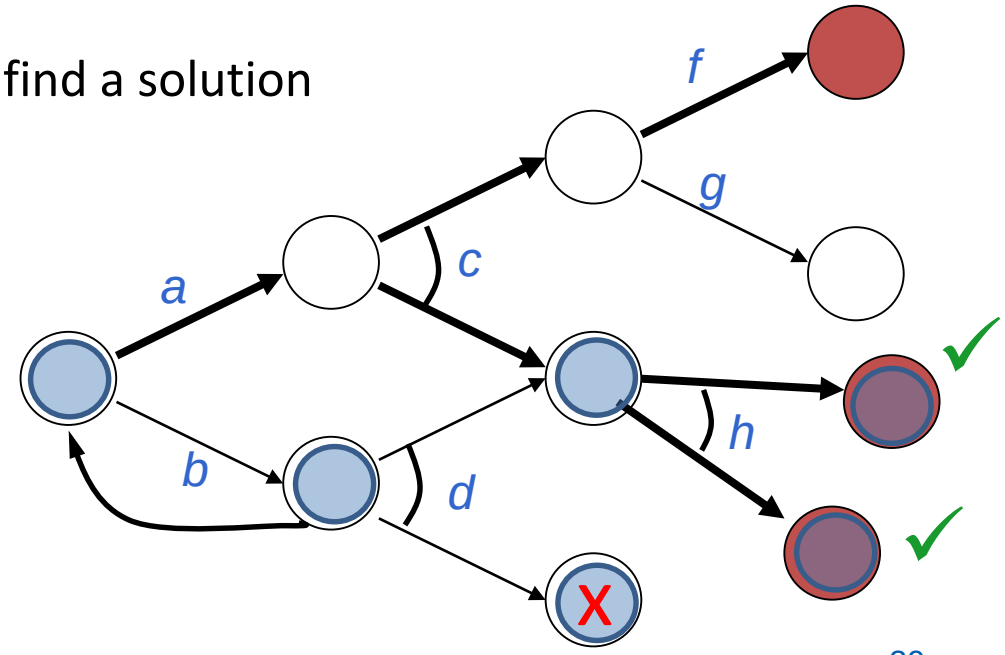
FONDAZIONE BRUNO KESSLER

Some Planning Techniques

- And/Or Graph Search
- Symbolic Model Checking
- Determinization

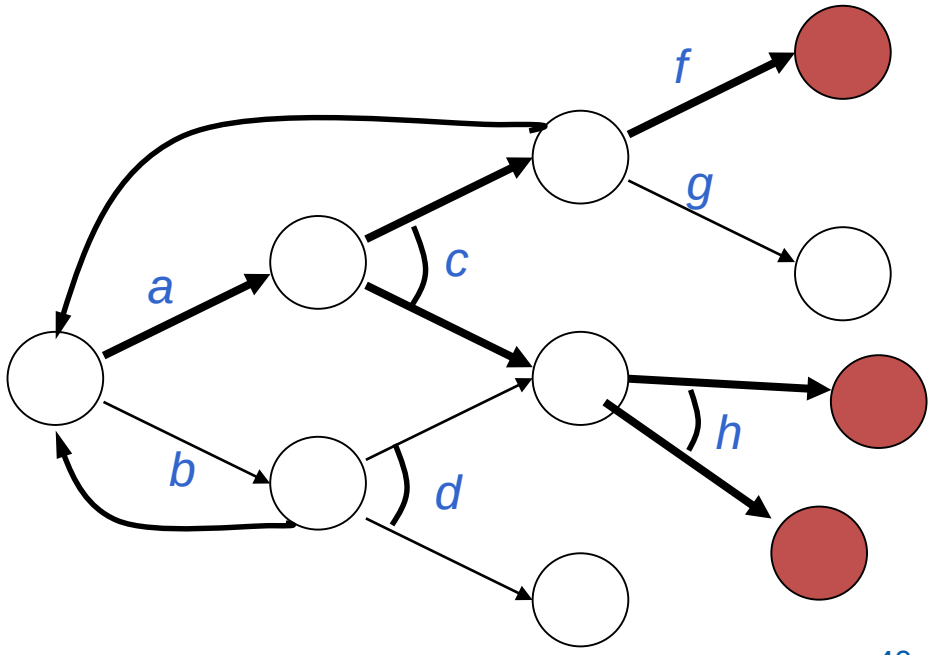
Determinization

- Motivation:
  - Much easier to find solutions if they don't have to be safe
  - Find-safe-solutions needs plans for all possible outcomes
  - Find-solution only needs a plan for one of them
- Idea:
  - Loop
    - Find a solution π
    - Look at each leaf node of π
      - If the leaf node isn't a goal, find a solution and incorporate it into π



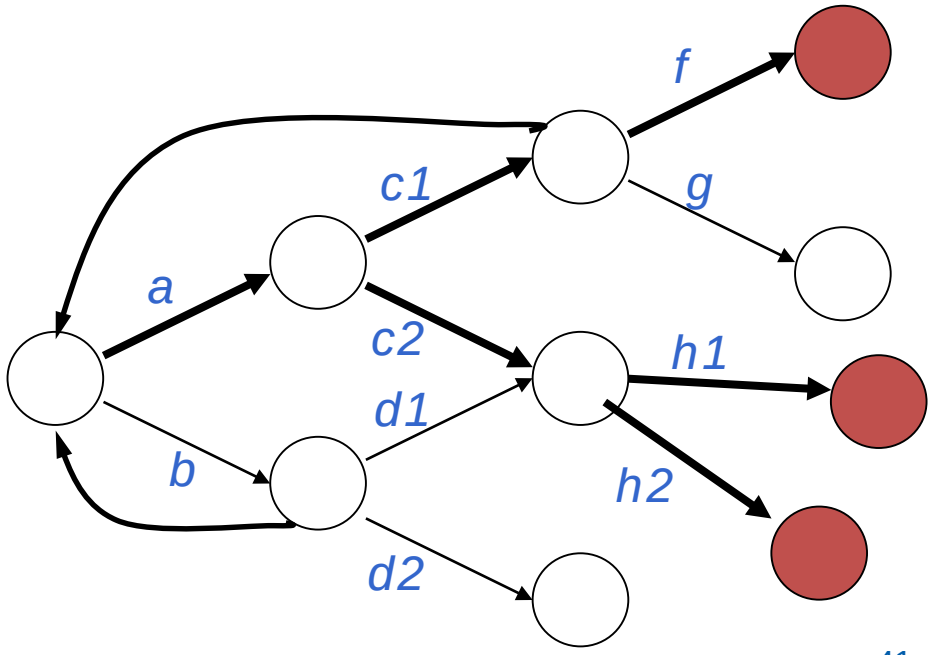
Determinization

- A possible (efficient) implementation:
  - Transform the nondeterministic domain into a deterministic one
  - If a has n outcomes, replace a with a_1, …, a_n deterministic actions
  - Find-solution replaced with an efficient classical planner …
  - The planner returns a sequential plan …



Determinization

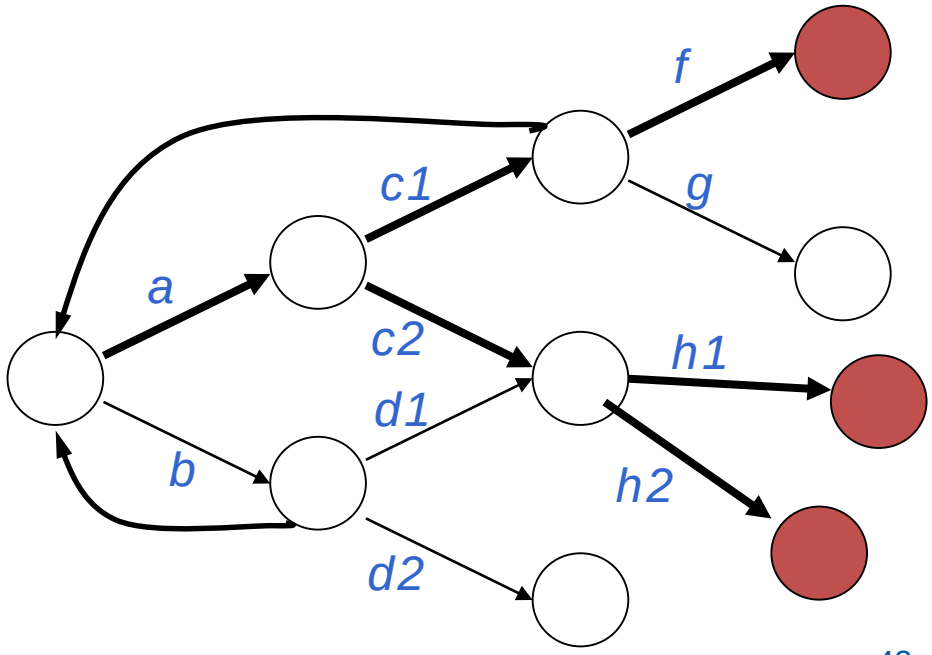
- A possible (efficient) implementation:
  - Transform the nondeterministic domain into a deterministic one
  - If a has n outcomes, replace a with a_1, …, a_n deterministic actions
  - Find-solution replaced with an efficient classical planner …
  - The planner returns a sequential plan …



Determinization

- A possible (efficient) implementation:
  - Transform the nondeterministic domain into a deterministic one
  - If a has n possible outcomes, replace a with a1, …, an deterministic actions
  - Find-solution replaced with an efficient classical planner …
  - The planner returns a sequential plan …

- NDP by determinization:
  - Loop
    - Find a sequential plan p
    - Look at each state induced by p
      - find a classical plan for each outcome of the action



FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

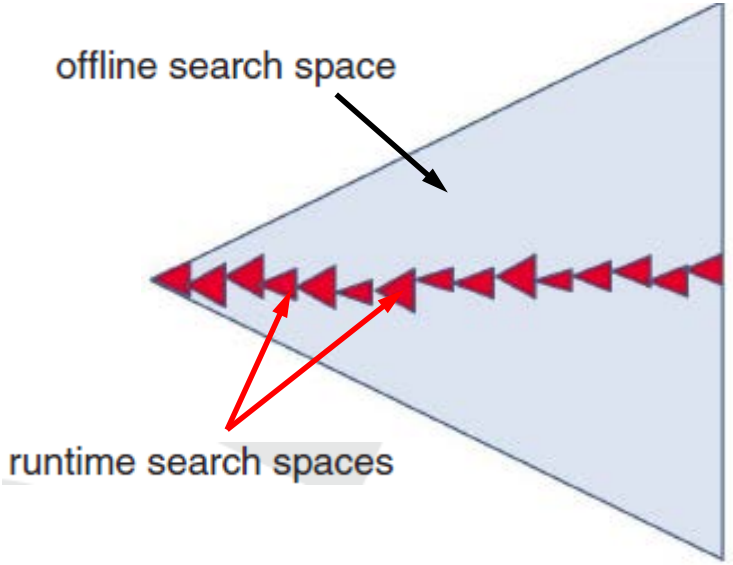
On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

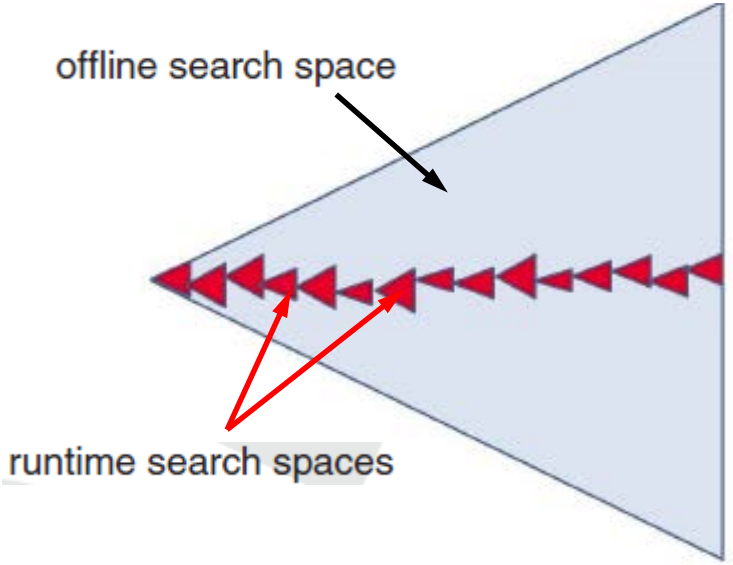
On-line Approaches

- Motivation:
  - Planning models are just approximations
  - Sensing is required to adapt to a changing environment
  - Need for dealing with large state spaces
- Idea:
  - Interleave planning and acting
  - find a partial policy the next few "good" actions, perform all or some of them, and repeat these two steps from the state that has been actually reached.



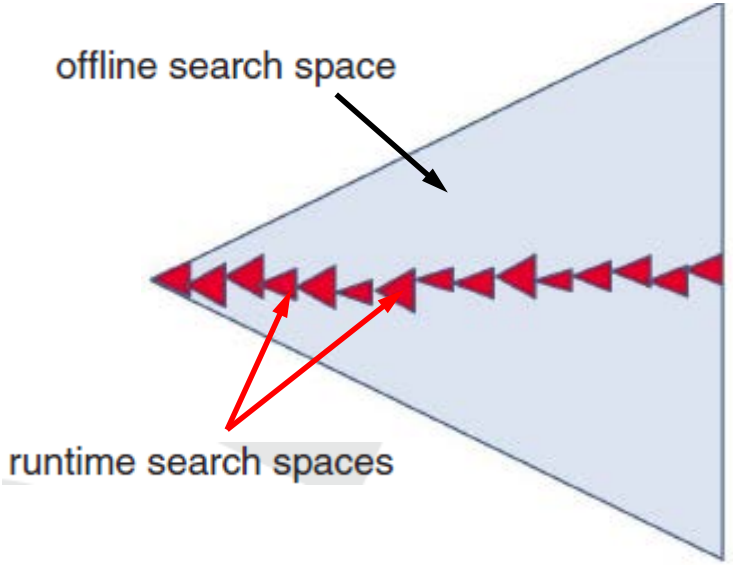
On-line Approaches

- Selection of “good” actions: **Lookahead**
  - by estimations of distances from the goal (e.g., heuristic search)
  - by learning step by step after each application better estimates



On-line Approaches

- **Lookahead:** two dimensions
  - **bound:** number of steps to lookahead (or time to lookahead)
    - extreme case: reactive planner where bound = 1
  - **limited nondeterminism: select just some of the outcomes**
    - extreme case: determinization (just 1 outcome)

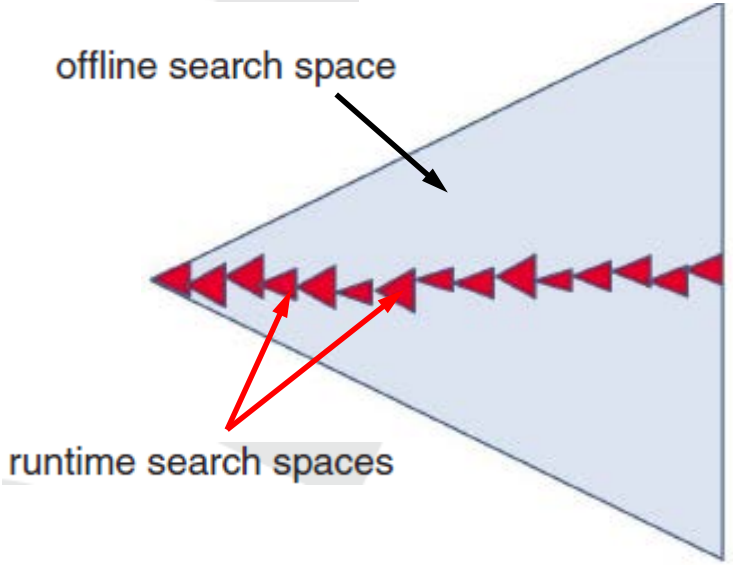


On-line Approaches

Algorithm 5.15 Online determinization planning and acting algorithm.

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
      $\pi_d \leftarrow$ Forward-search (Σ_d, s, S_g)
      if $\pi_d =$ failure then return failure
    perform action $\pi_d(s)$
    $s \leftarrow$ observe resulting state

Σ_d is the determinization of Σ



On-line Approaches

Algorithm 5.16 MinMax Learning Real Time A*.

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

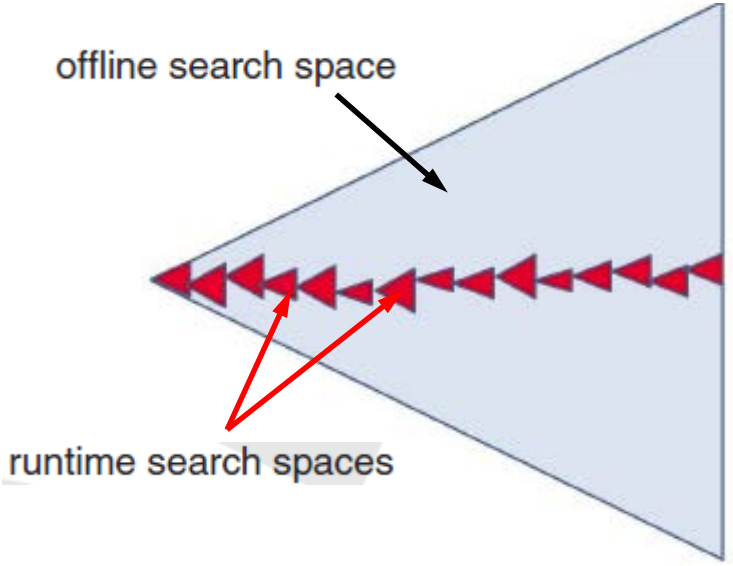
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

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

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

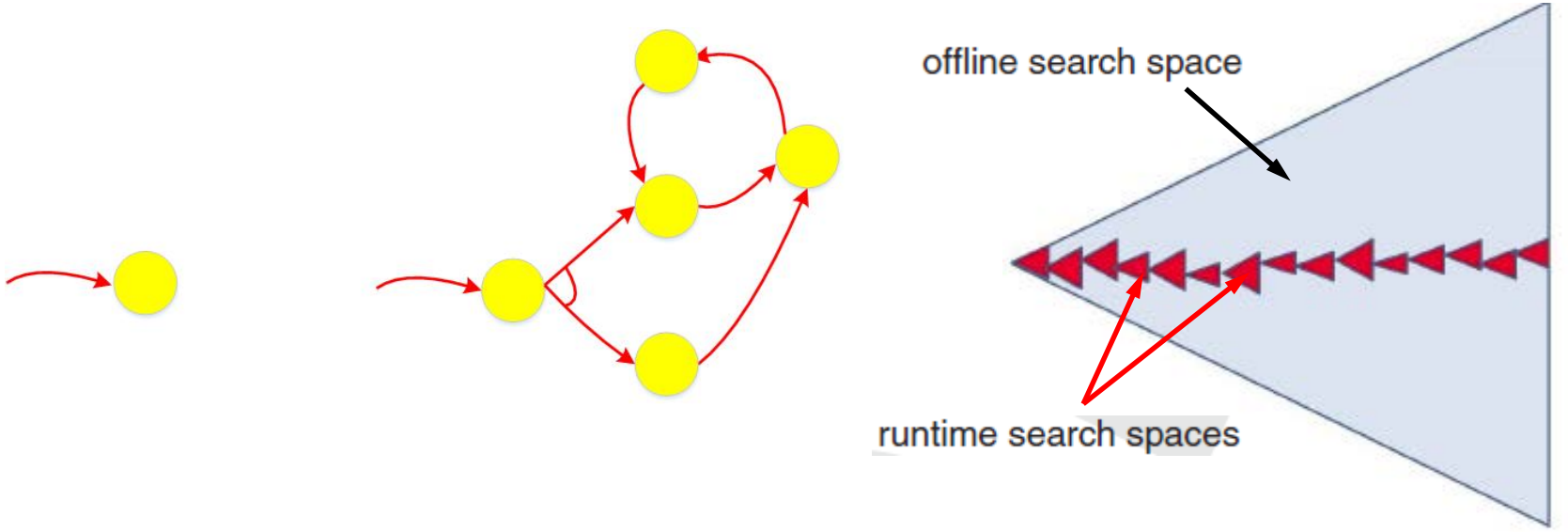
perform action a

$s \leftarrow$ the current state



On-line Approaches

- Critical issue: **Dead ends**
  - Possibility of getting stuck during acting
  - Completeness only in “safely explorable” domains



On-line Approaches

Algorithm 5.16 MinMax Learning Real Time A*.

Min-Max LRTA* (Σ, s_0, S_g)

$s \leftarrow s_0$

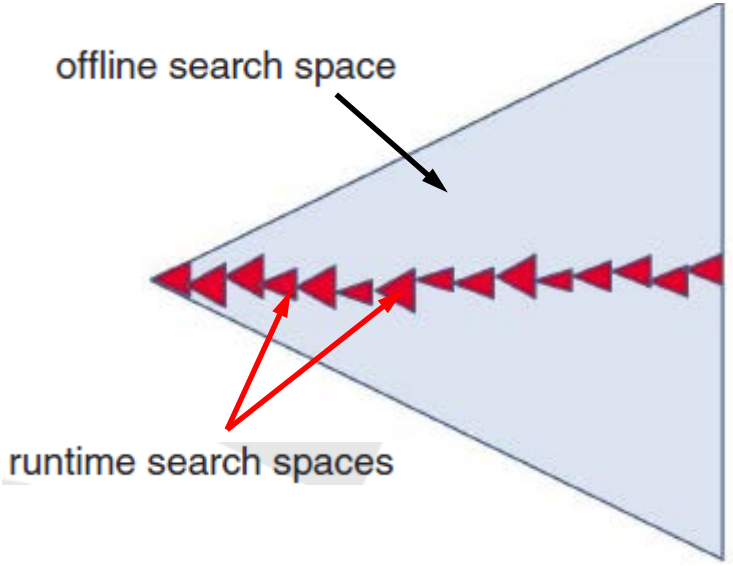
while $s \notin S_g$ and Applicable$(s) \neq \varnothing$ do

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

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

perform action a

$s \leftarrow$ the current state



FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata

ECAI 1992: Invited Talk by Oliviero Stock

IRST

New scenarios (cont'd)

c) Communicating with objects

- we shall have interfaces between humans and objects without even knowing there are computers in between
- also static objects will possibly react to human presence or will possibly provide information on events

  At Xerox PARC.... (no language so far)
- **How to say words with things**

  e.g. Metaphors in virtual reality to convey meaning

Acting by interactions!

How can I open you? ←

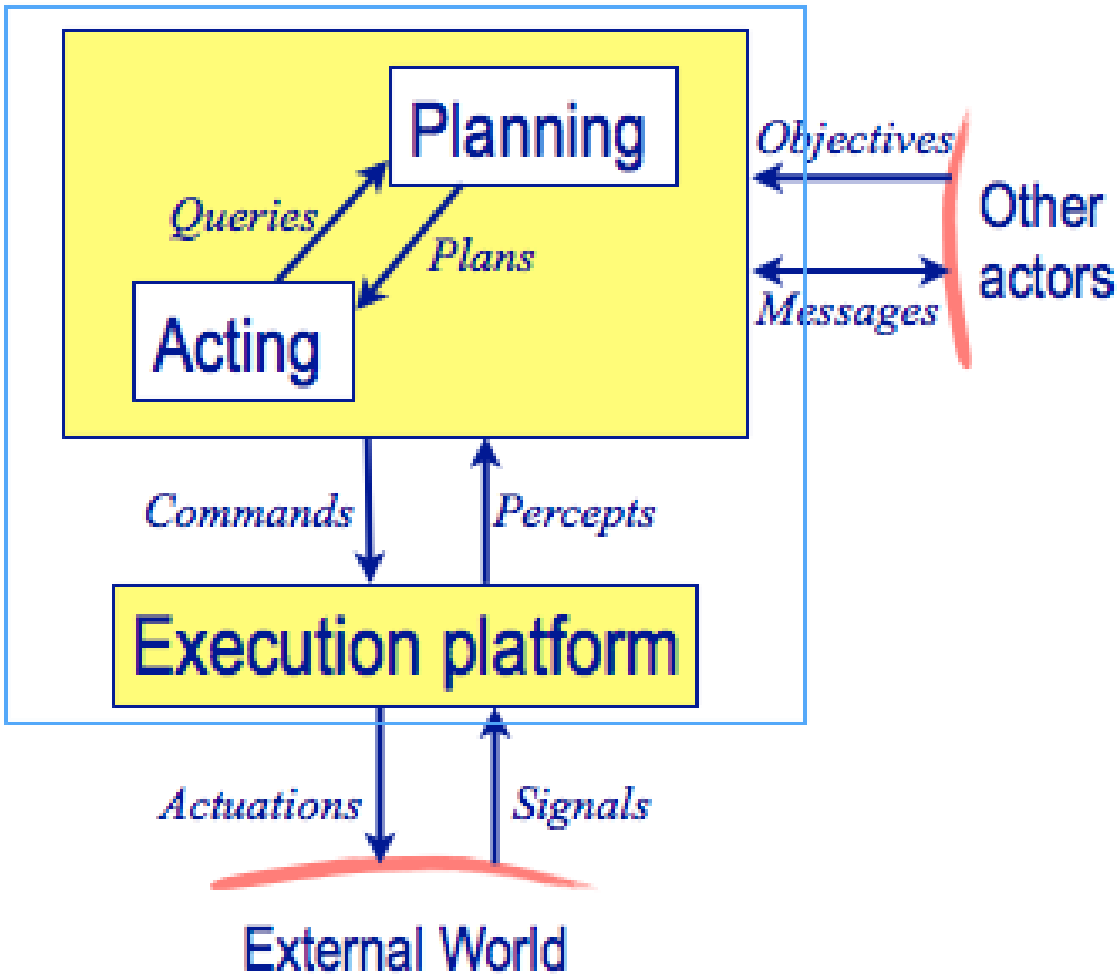
I am a sliding door →

Please provide me with your opening instructions ←

Door: opening module →

⋮

Acting is interaction!



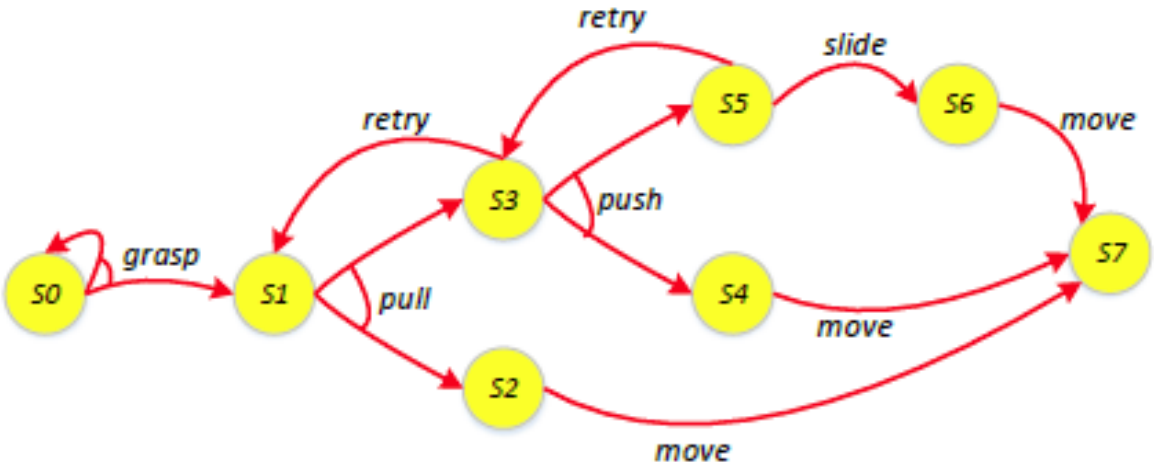
Acting with I/O Automata: Motivations

- Acting is a closed loop with a stream of inputs and outputs
- Interactions can be modeled with I/O automata
- Need to control I/O automata (by means of Control Automata)
- Hand specified Control Automata ≈ RAE with interaction
- Control Automata can be synthesized by planning with nondeterministic models

Acting with I/O Automata: Motivations

- Acting is a closed loop with a stream of inputs and outputs
- Interactions can be modeled with I/O automata
- Need to control I/O automata (by means of Control Automata)
- Hand specified Control Automata ≈ RAE with interaction
- Control Automata can be synthesized by planning with nondeterministic models

I/O Automata



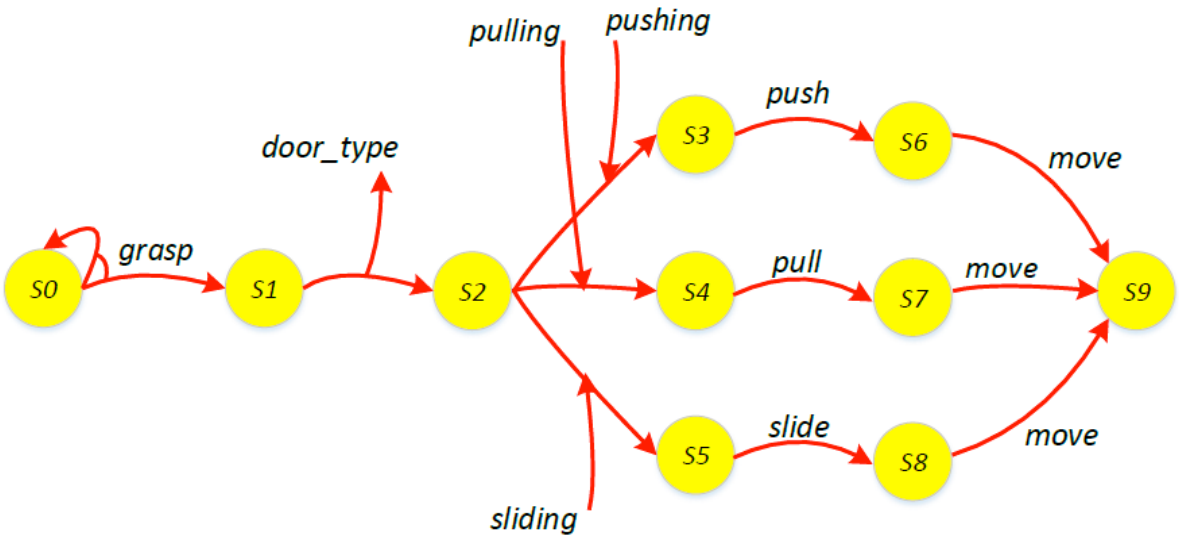
How can I open you?

I am a sliding door

Please provide me with your opening instructions

Door: opening module

I/O Automata



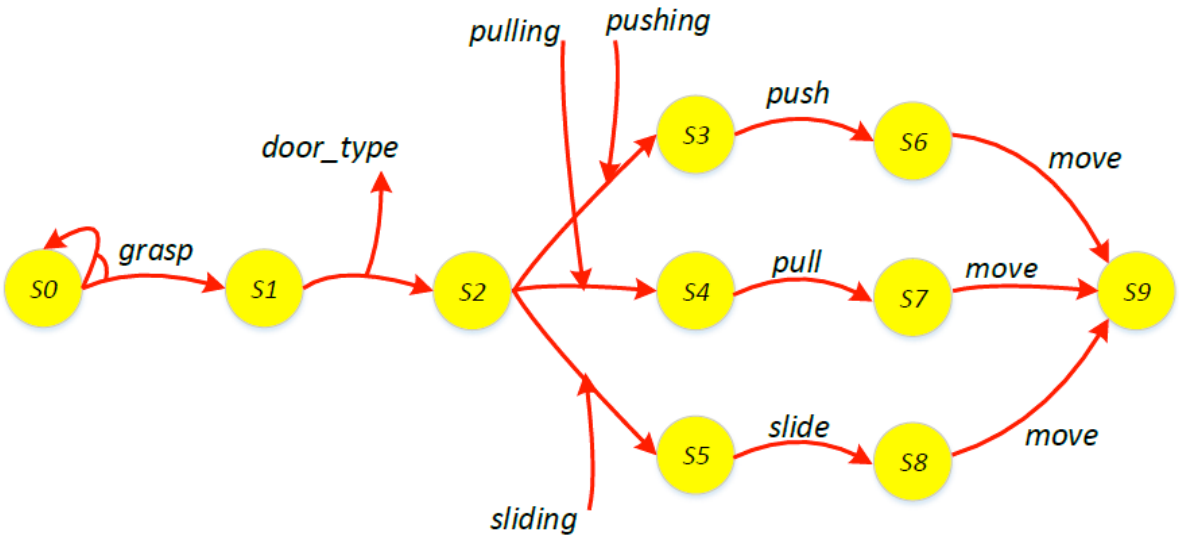
How can I open you?

I am a sliding door

Please provide me with your opening instructions

Door: opening module

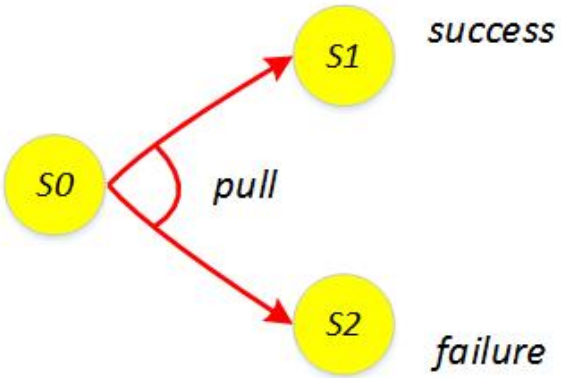
I/O Automata

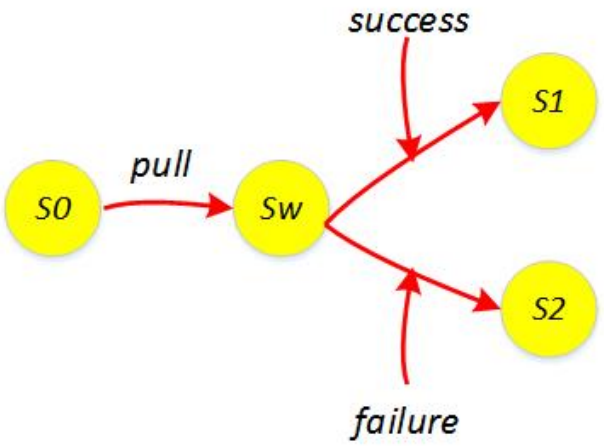


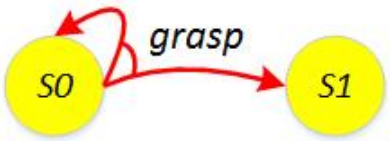
Input/output automaton $A = (S, S^0, I, O, C, \gamma)$:

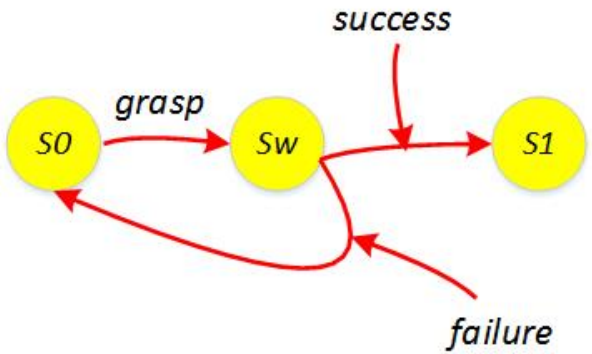
- S is a finite set of states
- $S^0 \subseteq S$ is the set of possible initial states the automaton can start in
- I is the set of inputs, O is the set of outputs, and C is a set of commands, with I, O, and C disjoint sets;
- $\gamma : S \times (I \cup O \cup C) \rightarrow 2^S$ is the nondeterministic state transition function

Inputs can model nondeterminism





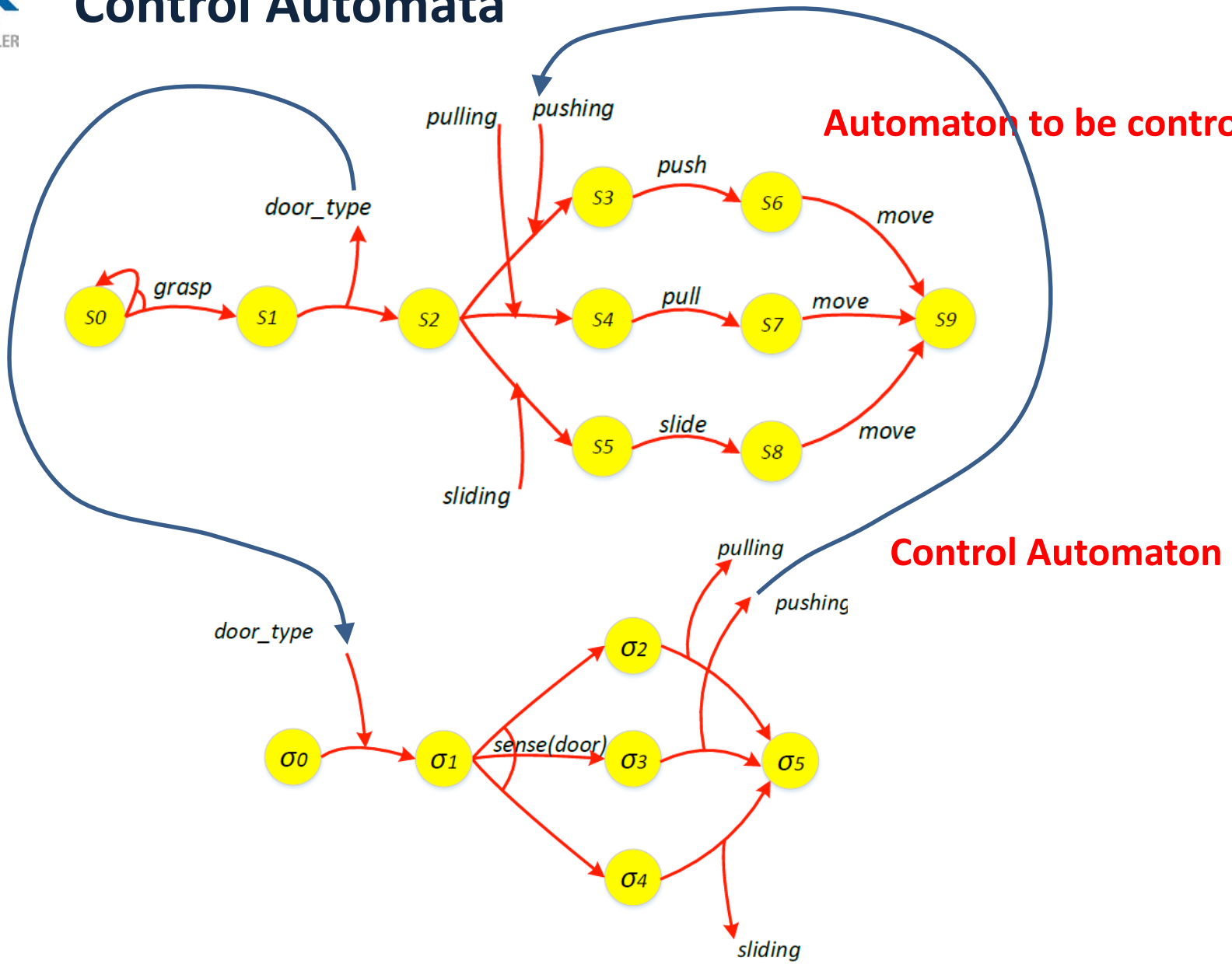




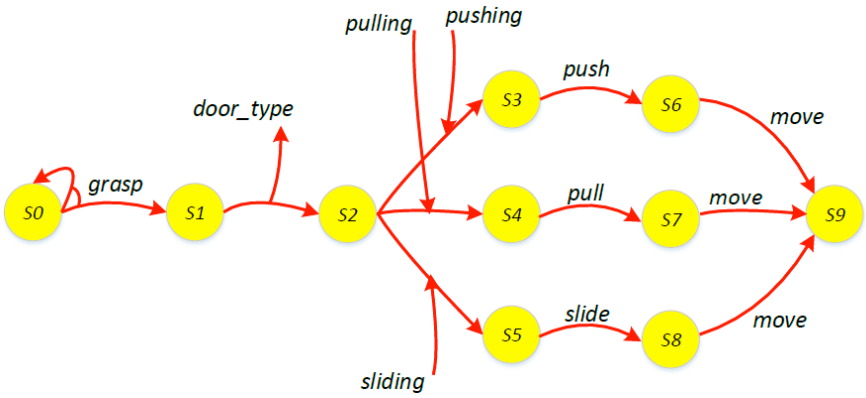
Acting with I/O Automata: Motivations

- Acting is a closed loop with a stream of inputs and outputs
- Interactions can be modeled with I/O automata
- Need to control I/O automata (by means of **Control Automata**)
- Hand specified Control Automata ≈ RAE with interaction
- Control Automata can be synthesized by planning with nondeterministic models

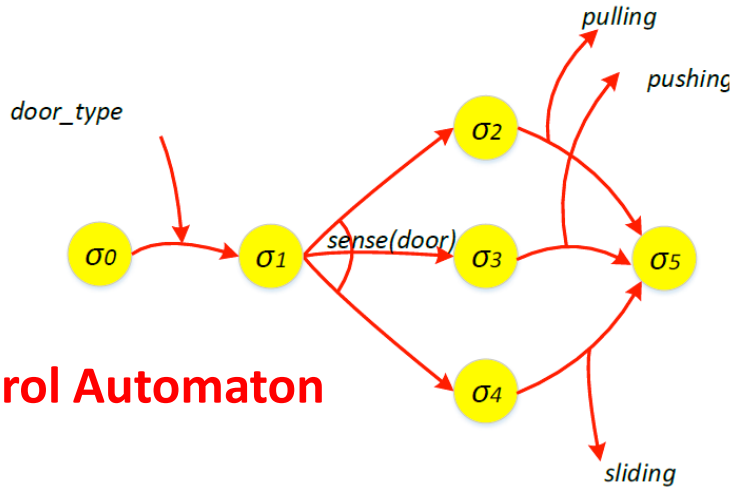
Control Automata



Control Automata



Automaton to be controlled



Control Automaton

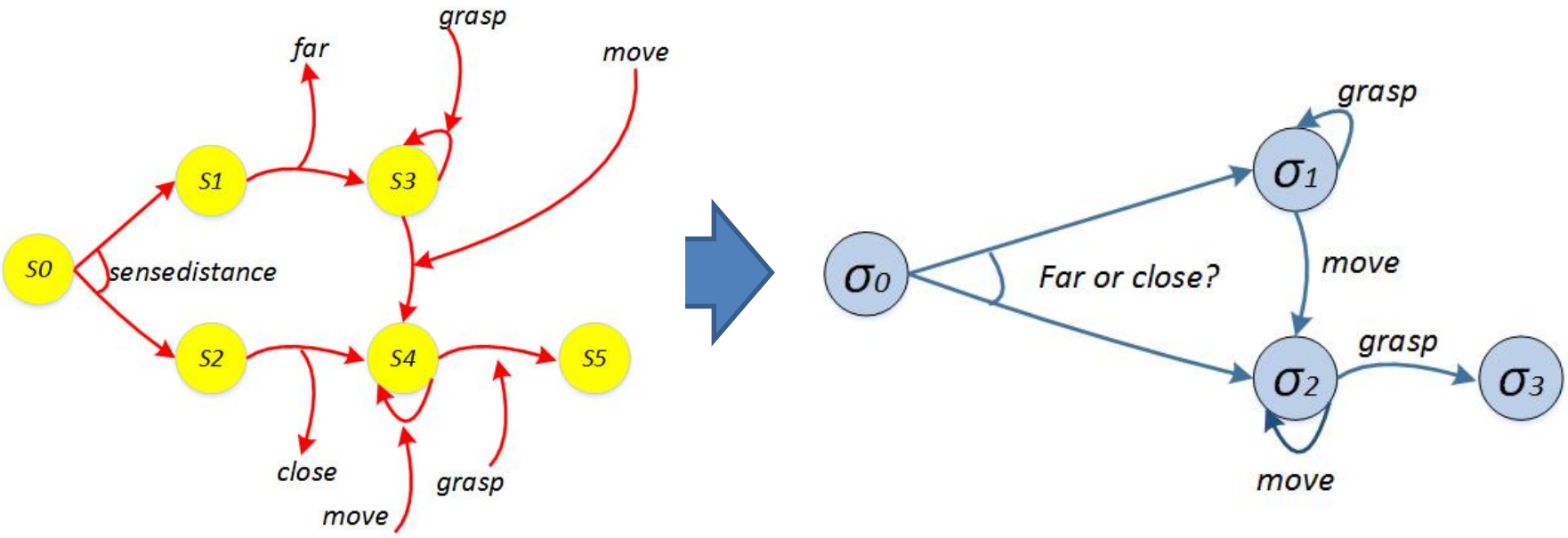
Definition 5.27. (Control Automaton) Let $A = (S, S^0, I, O, C, \gamma)$ be an input/output automaton. A Control Automaton for A is an input/output automaton $A_c = (S_c, S_c^0, O, I, C_c, \gamma_c)$. □

Acting with I/O Automata: Motivations

- Acting is a closed loop with a stream of inputs and outputs
- Interactions can be modeled with I/O automata
- Need to control I/O automata (by means of Control Automata)
- Hand specified Control Automata ≈ RAE with interaction
- Control Automata can be synthesized by planning with nondeterministic models

We can synthesize control automata by planning

- Transform I/O automaton into a nondeterministic planning domain



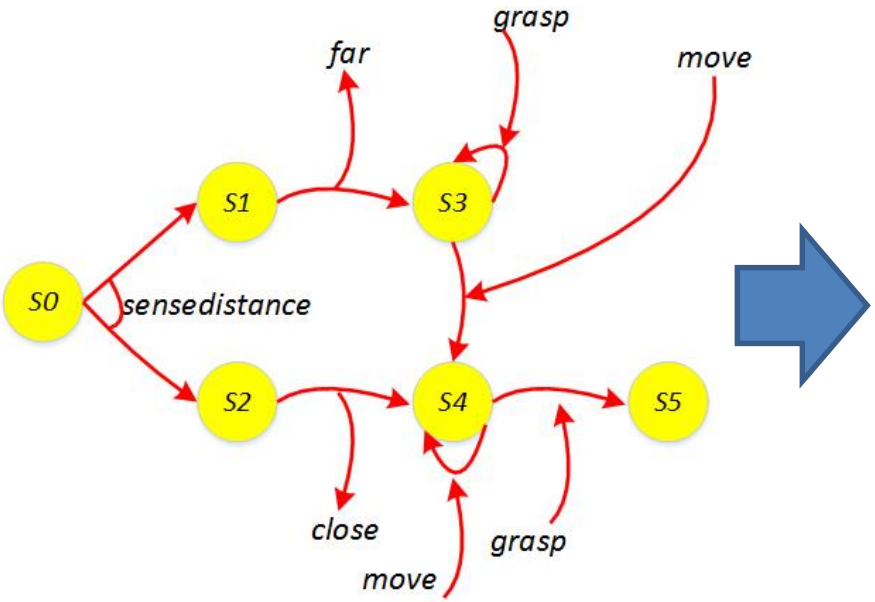
Automaton to be controlled

Nondeterministic planning domain

- Planning with nondeterministic models can synthesize control automata
- We can use and/or graph search, symbolic model checking, determinization

We can synthesize control automata by planning

- Transform I/O automaton into a nondeterministic planning domain



Automaton to be controlled

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

policy

- Planning with nondeterministic models can synthesize control automata
- We can use and/or graph search, symbolic model checking, determinization

FONDAZIONE BRUNO KESSLER

Agenda

Introduction & Motivation

Nondeterministic Models

Some Planning Techniques

On-line Approaches

Acting with I/O Automata

Hierarchical I/O Automata (Research Challenge)

RAE method for opening a door: pull, push, or slide?

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)

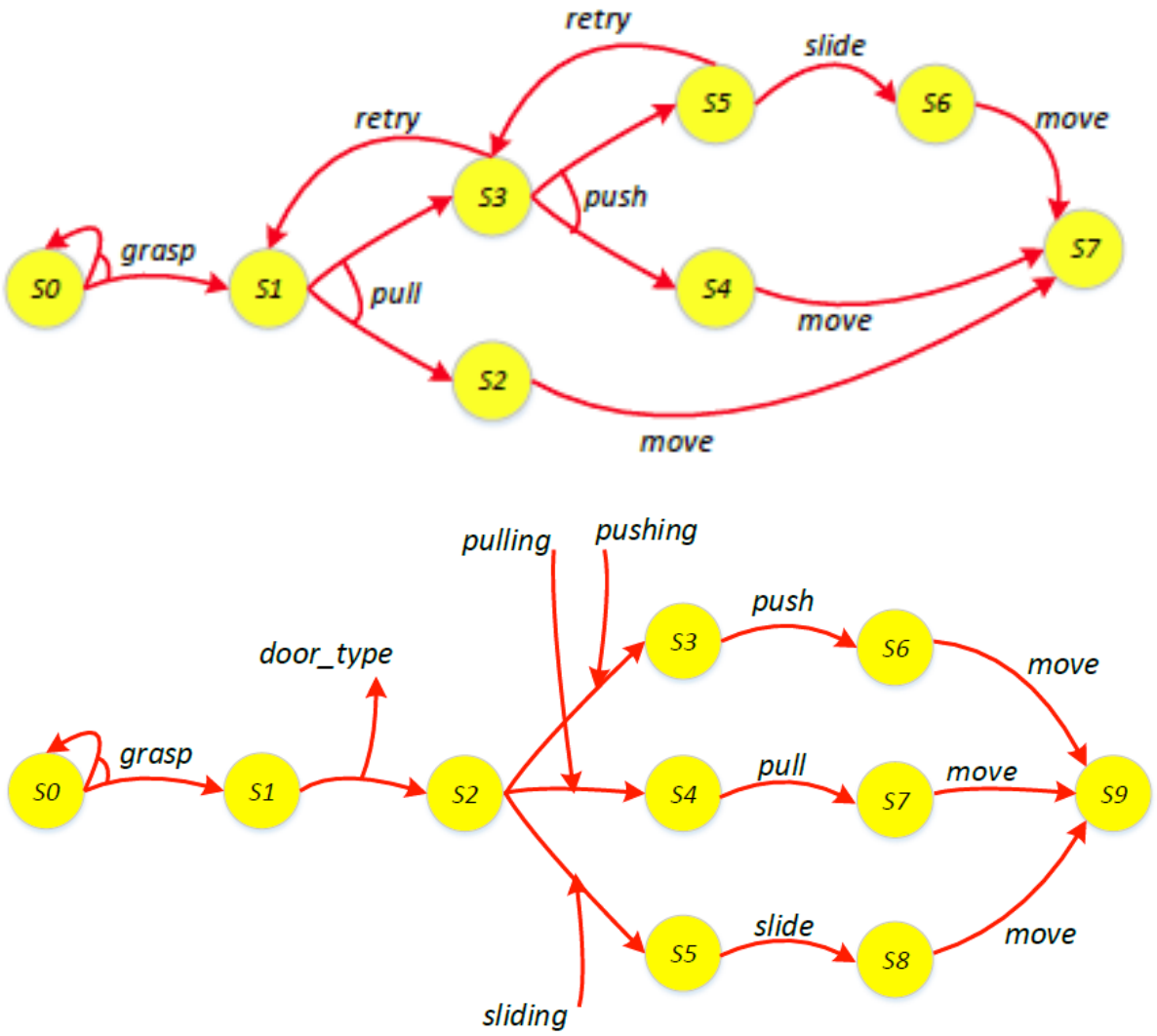
m-retry-pull(r, d, l, o)
task: pull-push(r, d)
body: pull(r, d); pull-push(r, d)

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);push-slide(r, d)

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

Flat (!) Nondeterministic Models



Refinement methods: Hierarchical I/O Automata

HIOA $= \langle S, S_0, I, O, C, T, \gamma \rangle$

- S is a finite set of states,
- $S_0 \subseteq S$ is the set of possible initial states the automaton can start in
- I is a set of inputs, O is a set of outputs, C is a set of commands, and T is a set of tasks, with I, O, C, and T disjoint sets
- $\gamma : S \times (I \cup O \cup C \cup T) \rightarrow 2^S$ is the nondeterministic state transition function.

Refinement methods: example

m-opendoor(r, d, l, o)

task: opendoor(r, d)

pre: loc(r) $=$ l $\wedge$ adjacent(l, d) $\wedge$ handle(d, o)

body: while $\neg$reachable(r, o) do

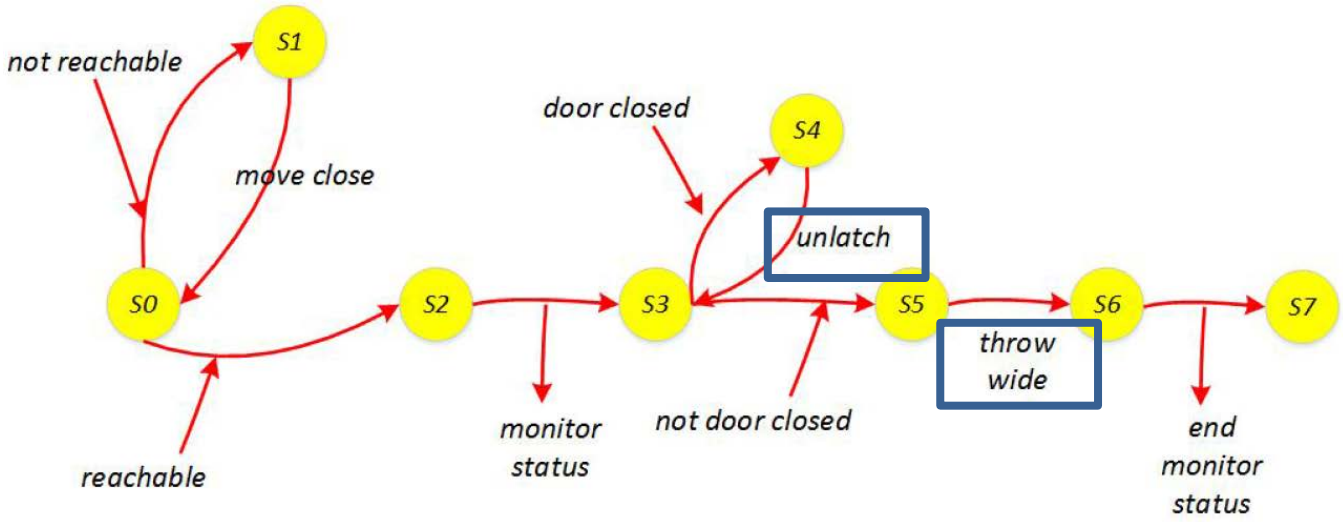
move-close(r, o)

monitor-status(r, d)

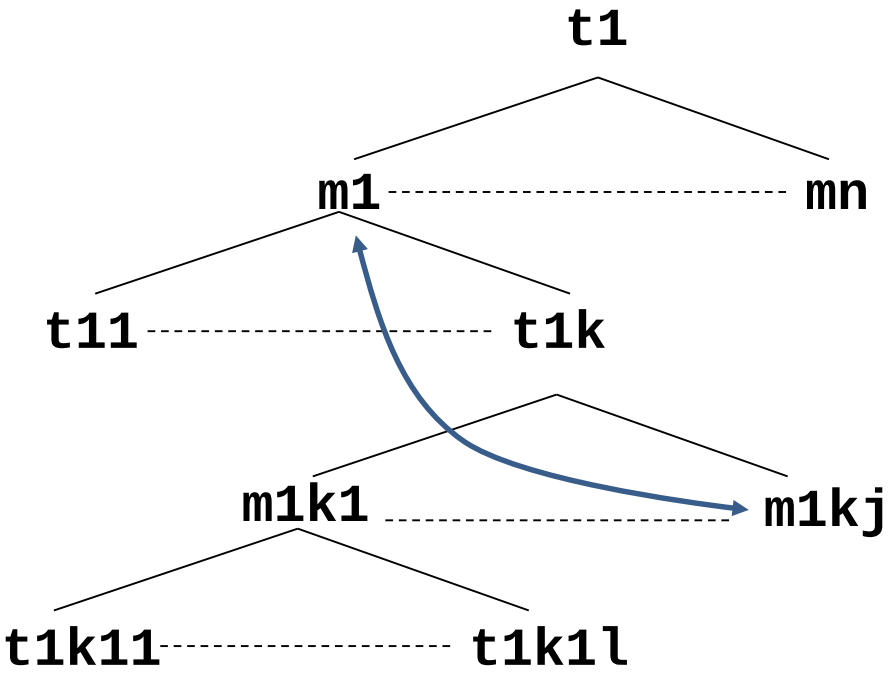
if door-status(d)=closed then unlatch(r, d)

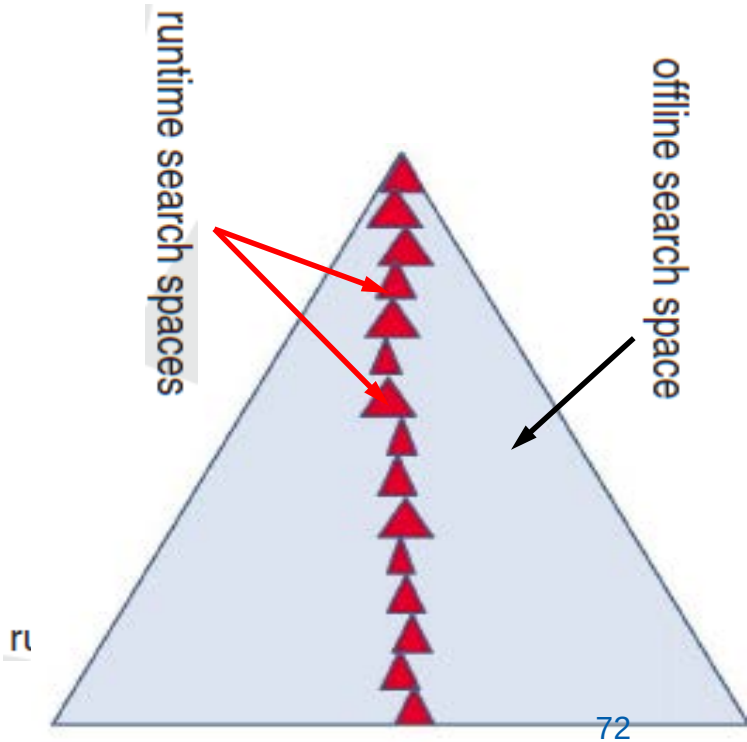
throw-wide(r, d)

end-monitor-status(r, d)



Refinement methods





Hierarchical Lookahead by refinement

FONDAZIONE BRUNO KESSLER

Conclusions

- Nondeterminism can be a design choice
- In some cases nondeterminism is a must!
- Nondeterminism in Acting is a must!
- Deliberation by planning with nondeterminism
- Interleaving planning and acting sometimes is a must!
- Acting is a closed loop with a stream of inputs and outputs
- Deliberative Acting with I/O automata

FONDAZIONE
BRUNO KESSLER

Deliberation with Nondeterministic Models

Malik Ghallab, Dana Nau, Paolo Traverso
Automated Planning and Acting
Cambridge University Press

IJCAI 2016 Tutorial
New York, July 11th,2016