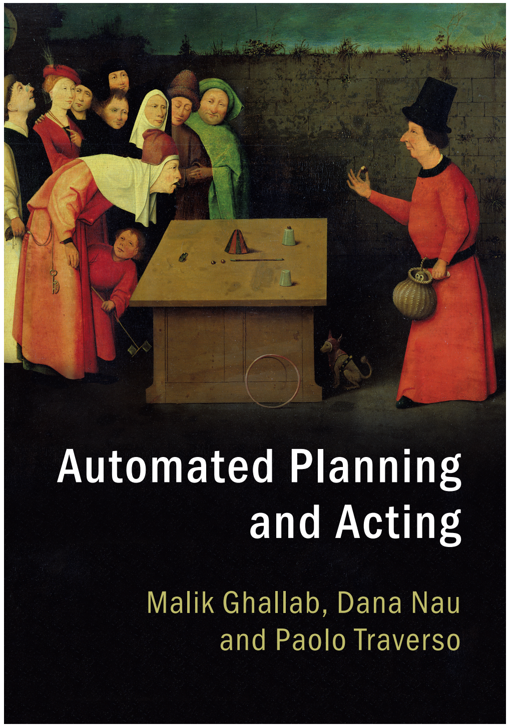
Last update: June 22, 2017



http://www.laas.fr/planning

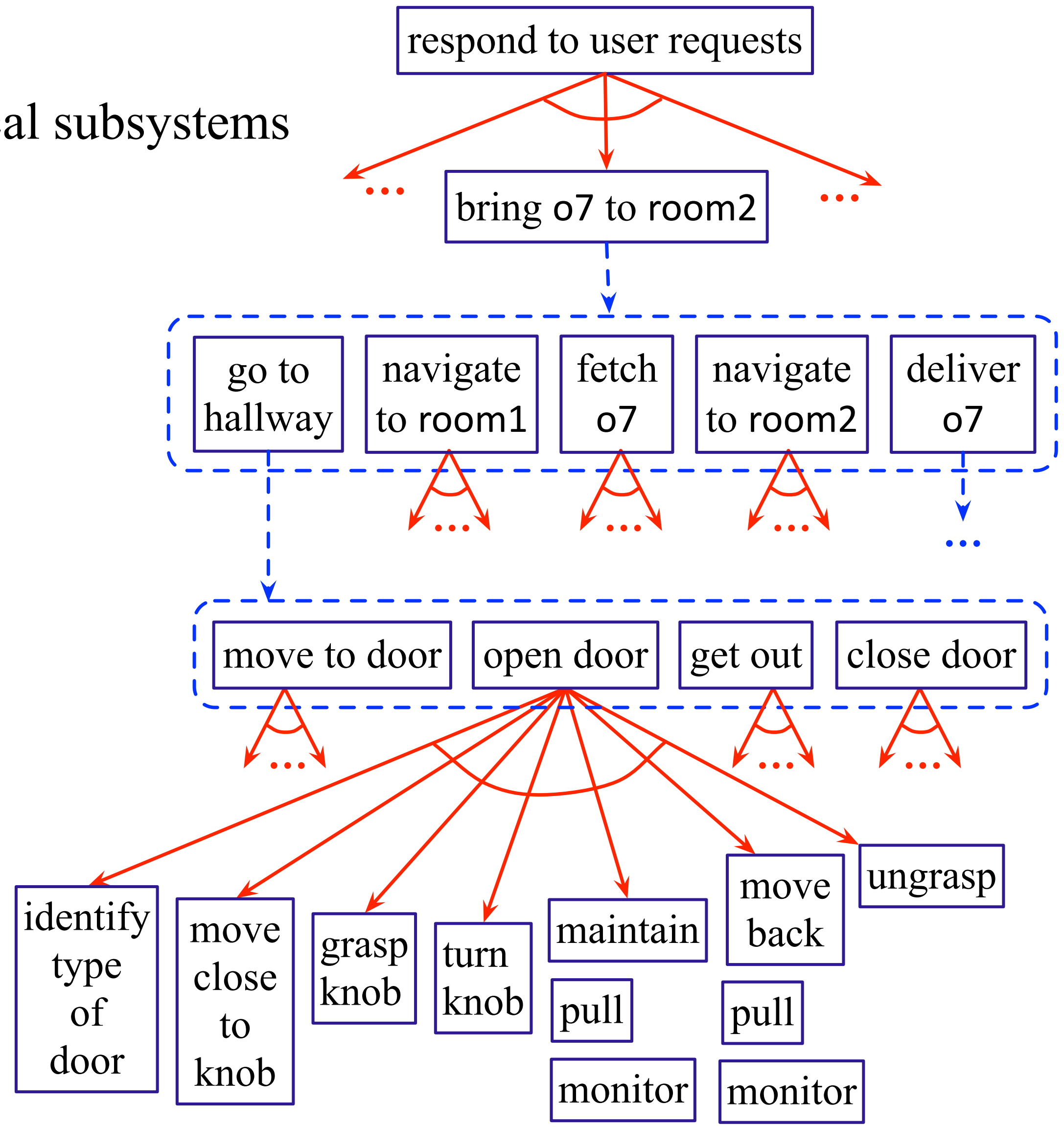
Deliberation in Planning and Acting

Part 2: Refinement Models

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

Motivation

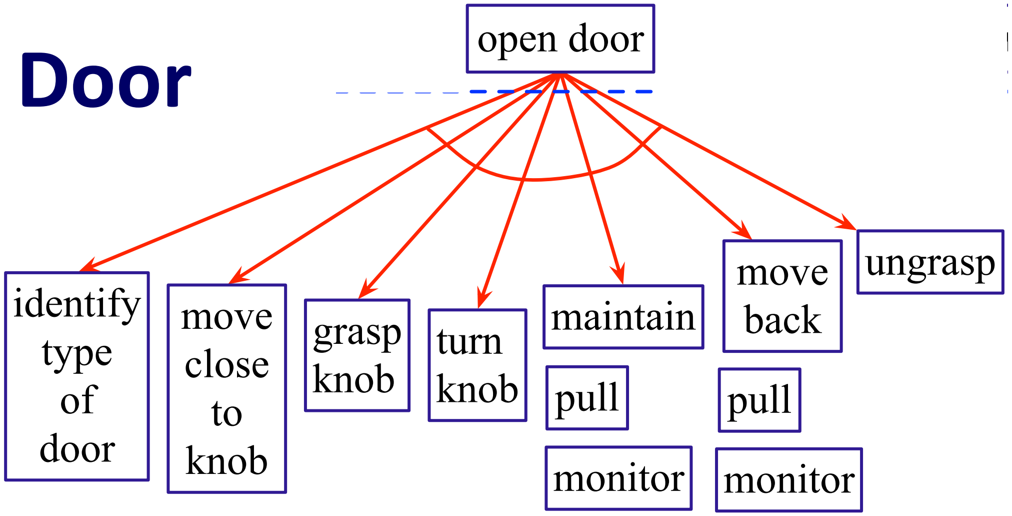
- *Multiple levels of abstraction*
  - Actors organized into physical subsystems
  - Deliberation reflects this
- *Heterogeneous reasoning*
  - Different techniques
    - at different levels
    - in different subsystems at same level
- *Continual online planning*
  - World models partial, can't plan everything in advance
  - Plans are abstract and partial until more detail is perceived at acting time



Opening a Door

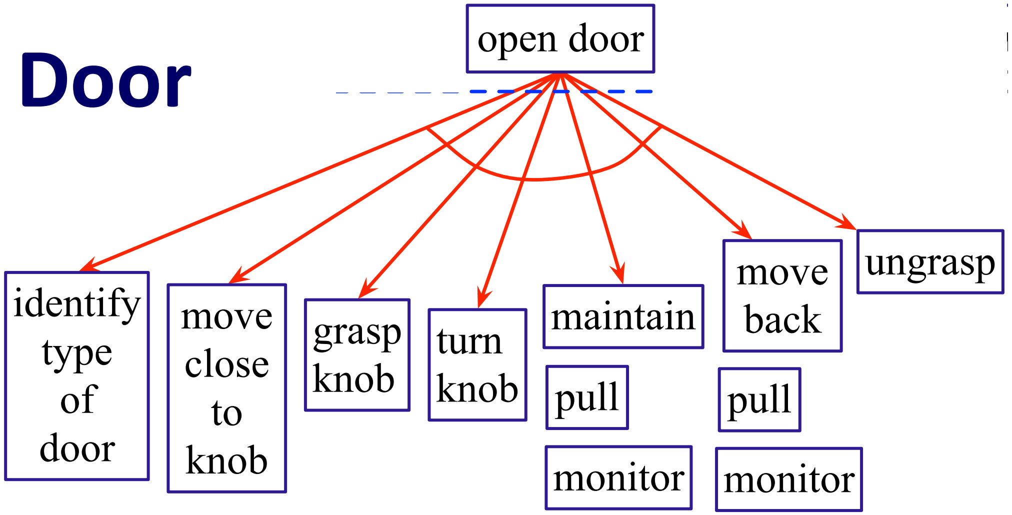
- Continual interaction with the environment
- *How* depends on what kind of door

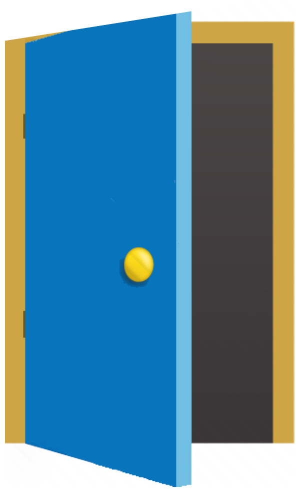
not known until acting time



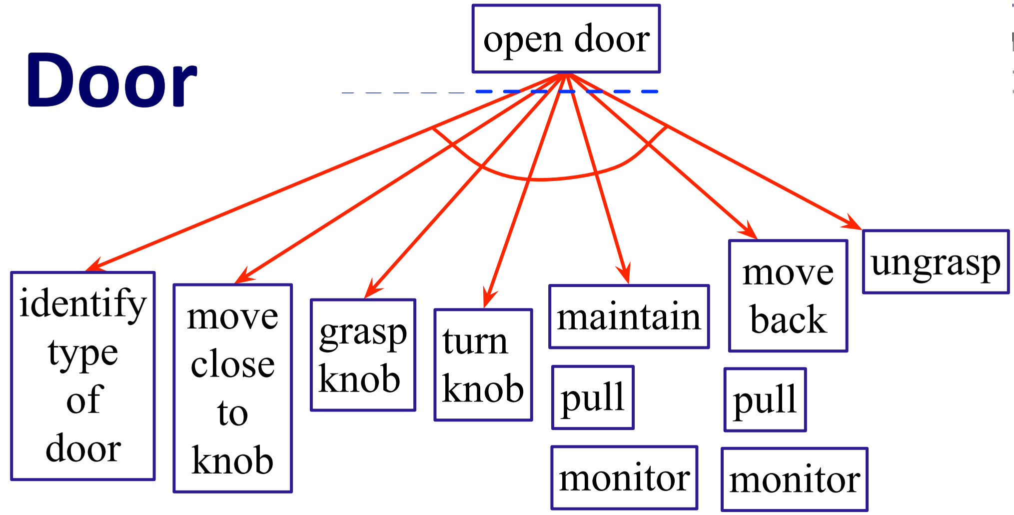
Opening a Door

- Continual interaction with the environment
- *How* depends on what kind of door
  - Sliding or hinged?

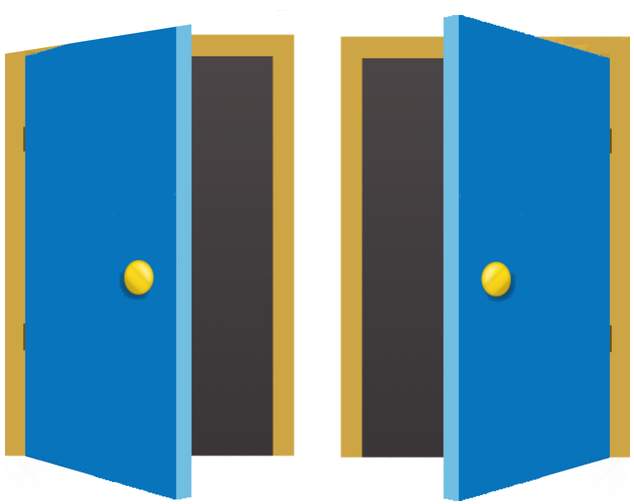




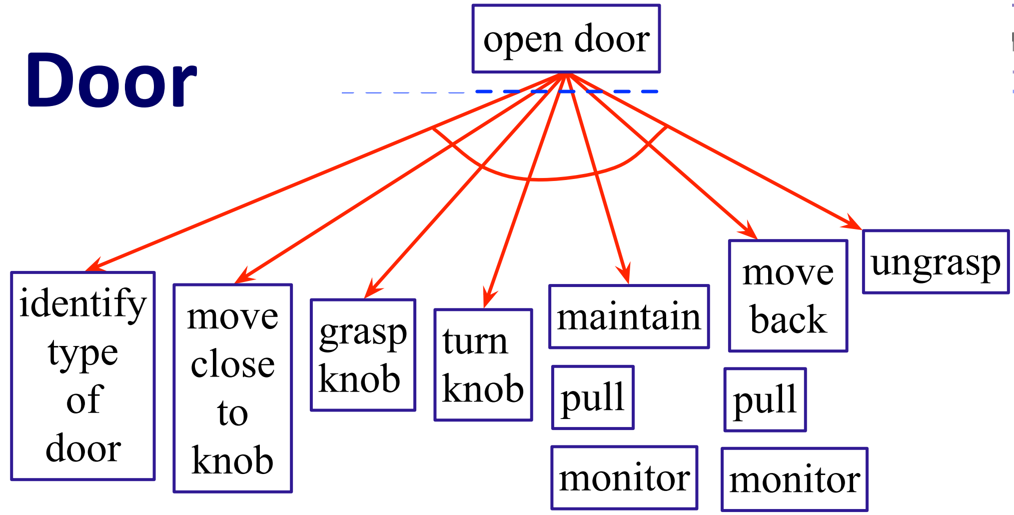
Opening a Door



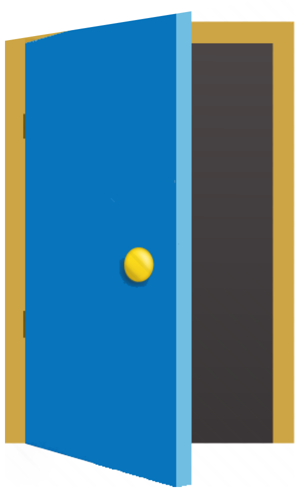
- Continual interaction with the environment
- *How* depends on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?

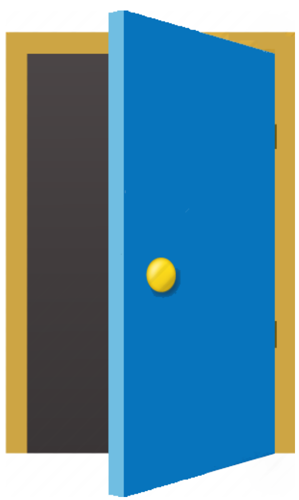


Opening a Door

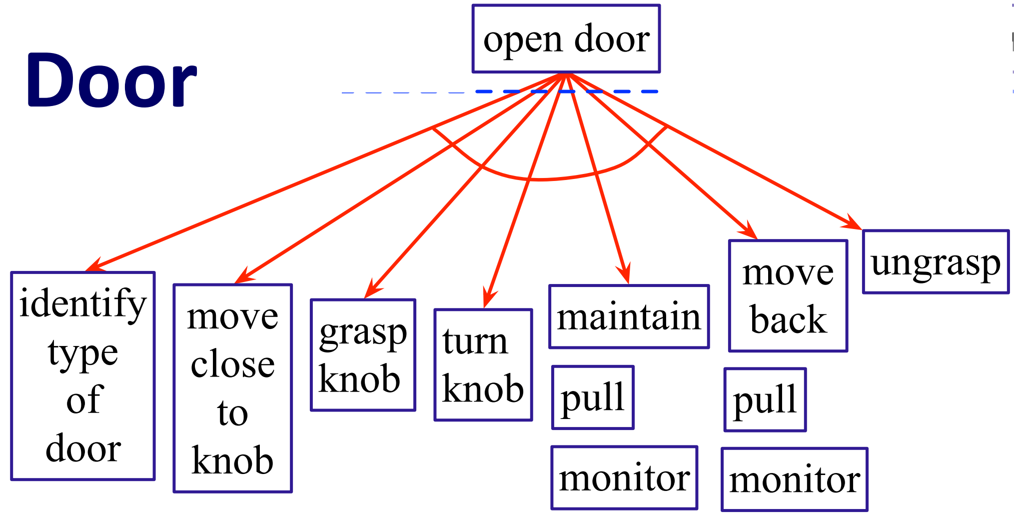


- Continual interaction with the environment
- *How* depends on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?

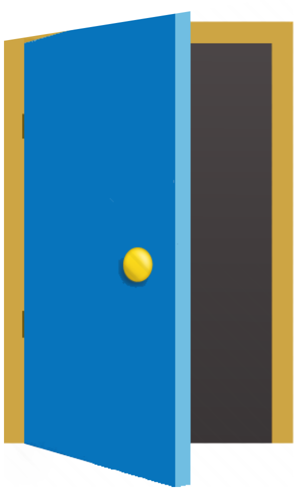


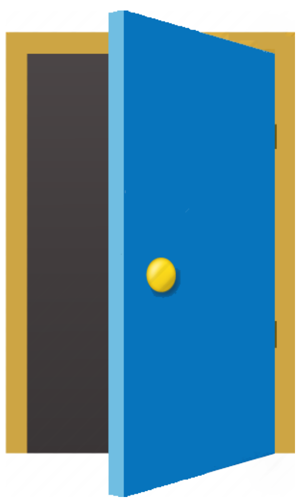


Opening a Door

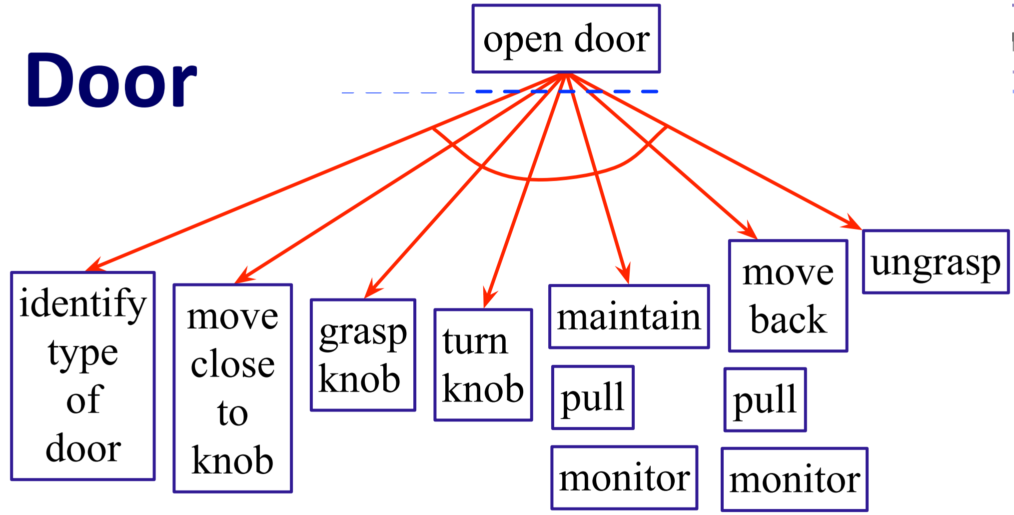


- Continual interaction with the environment
- *How* depends on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, …

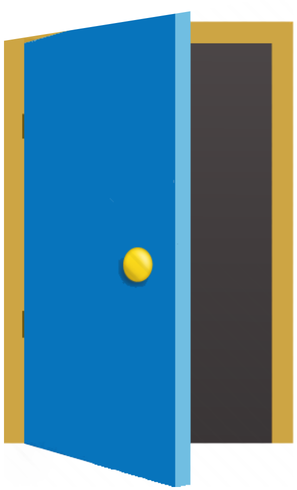


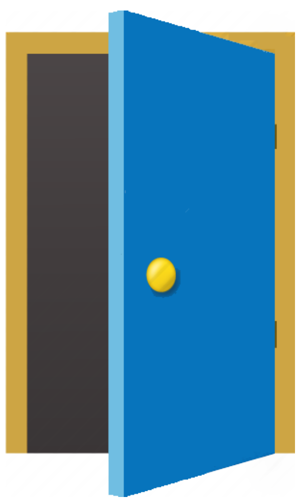


Opening a Door

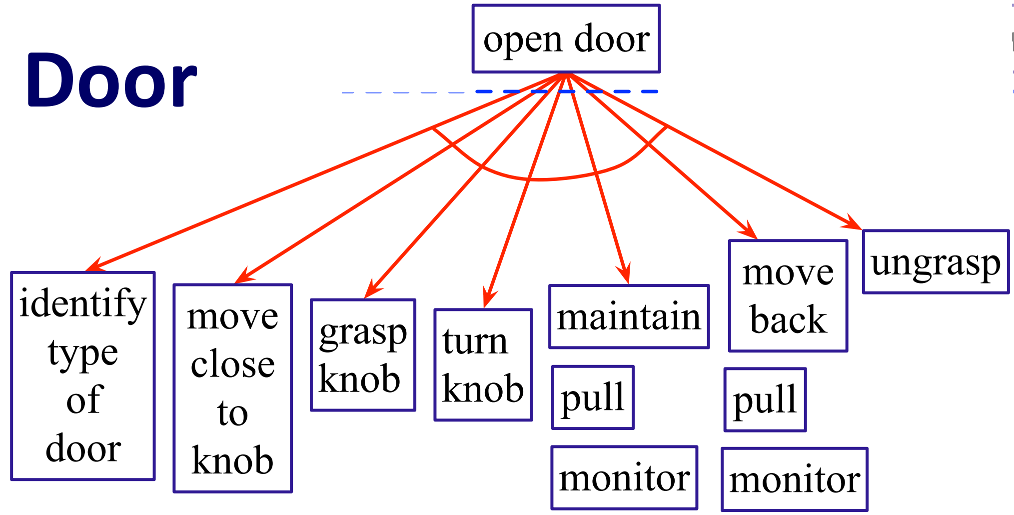


- Continual interaction with the environment
- *How* depends on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, …

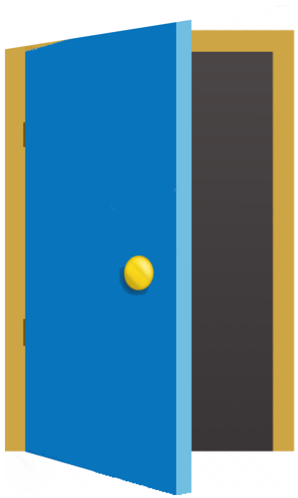




Opening a Door



- Continual interaction with the environment
- *How* depends on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, something else?



Outline

1. **Representation**
   State variables, commands, refinement methods
2. **Acting**
   Rae (Refinement Acting Engine)
3. **Planning**
   SeRPE (Sequential Refinement Planning Engine)
4. **Using Planning in Acting**
   Techniques

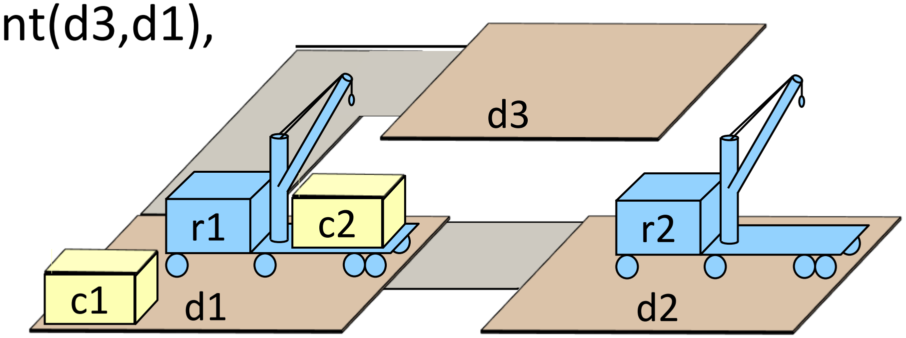
State Variables

- *Classical* representation represents states as sets of logical atoms

s_1 = { adjacent(d1,d2), adjacent(d2,d1),
adjacent(d1,d3), adjacent(d3,d1),
loc(r1,d1), cargo(r1,c2),
loc(r2,d2), empty(r2),
pos(c1,d1), pos(c2,r2) }

rigid

varying



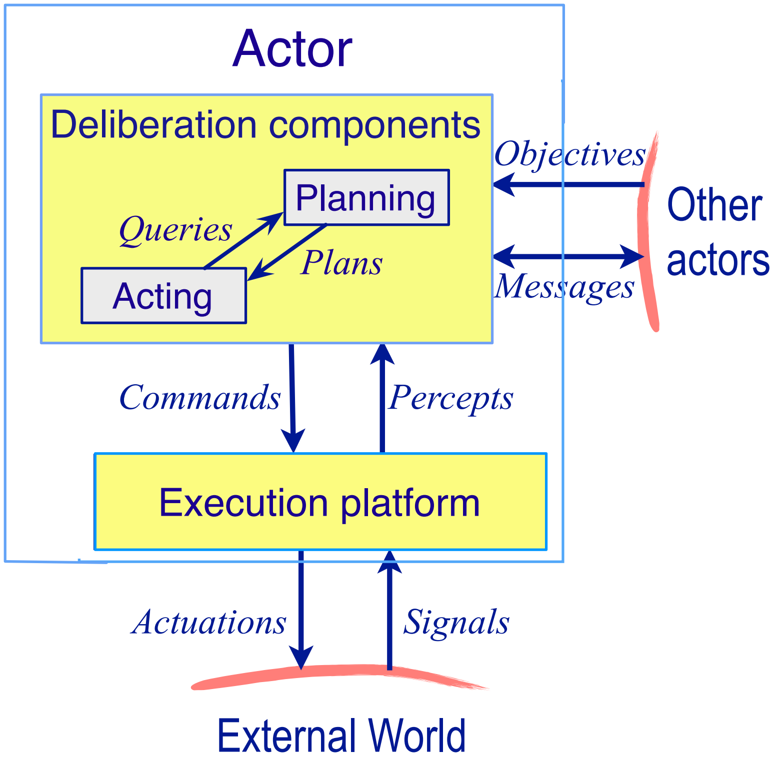
- *State Variable* representation:
  - Logical atoms for rigid properties

    adjacent(d1,d2), adjacent(d2,d1), adjacent(d1,d3), adjacent(d3,d1)

  - *State variables* for varying properties
    - syntactic terms that have different values in different states

      s_1 = { loc(r1) = d1, cargo(r1) = c2,
      loc(r2) = d2, cargo(r2) = nil,
      pos(c1) = d1, pos(c2) = r2 }

Tasks and Methods



- *Task*: activity for the actor to perform
- For each task, one or more *refinement methods*
  - Operational models
  - Different ways to perform the task

method-name(*arguments*)
  task: *task-name*(*arguments*)
  pre: *test*
  body: *a program*

- assignment statements
- control constructs
  - if-then-else, while, for, case, etc.
- tasks to perform
- goals to achieve
- *commands* to the execution platform

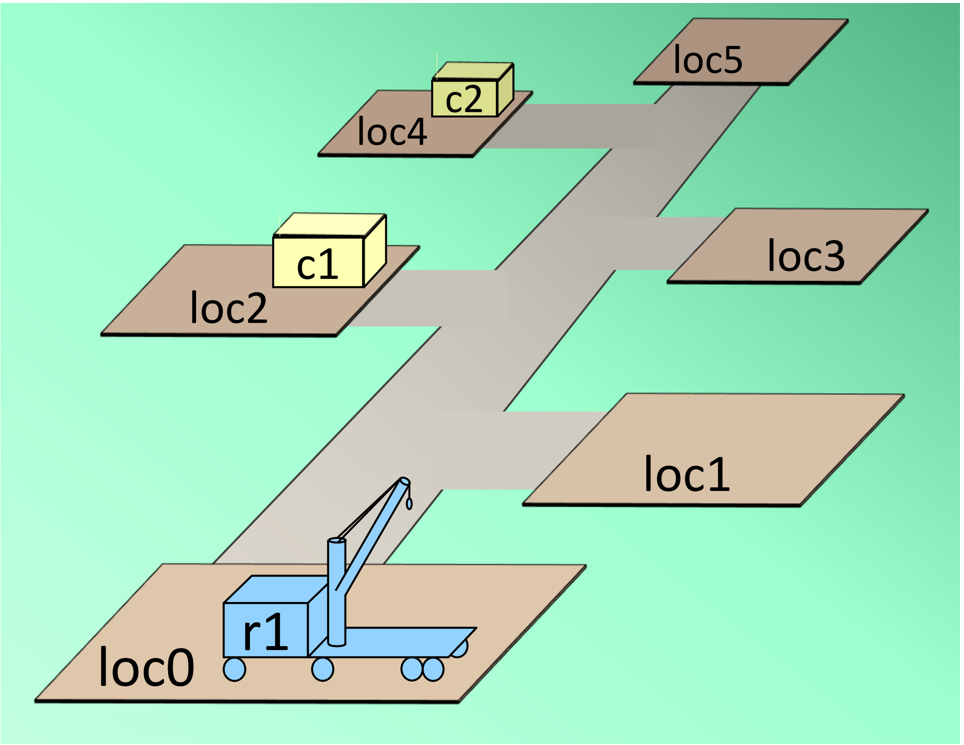
- Can also have
  - methods for achieving goals
  - methods for responding to events

Example

```
m-fetch(r,c)
    task:  fetch(r,c)
    pre:
    body:
        if pos(c) = unknown then
            search(r,c)
        else if loc(r) = pos(c) then
            take(r,c,pos(c))
        else do
            move-to(r,pos(c))
            take(r,c,pos(c))

m-search(r,c)
    task:   search(r,c)
    pre:    pos(c) = unknown
    body:
        if ∃l (view(r,l) = F) then
            move-to(r,l)
            perceive(l)
            if pos(c) = l then
                take(r,c,l)
            else search(r,c)
        else fail
```

- Tell robot r1 to fetch c2
- r1 doesn't know c2's location, needs to search
- Commands the execution platform can handle:
  - take, put, perceive, move-to



Goals

- Write goal as a special kind of task
  - achieve(*condition*)

method-name(*args*)
    task: achieve(*condition*)
    pre: *test*
    body: *program*

- Like other tasks, but includes monitoring
  - if *condition* becomes true before finishing body(*m*), return without finishing
  - if *condition* isn't true after finishing body(*m*), then fail

Events

- *Event*: something that may happen in a dynamic environment

method-name(*args*)
    event: *event-name*(*args*)
    pre: *test*
    body: *program*

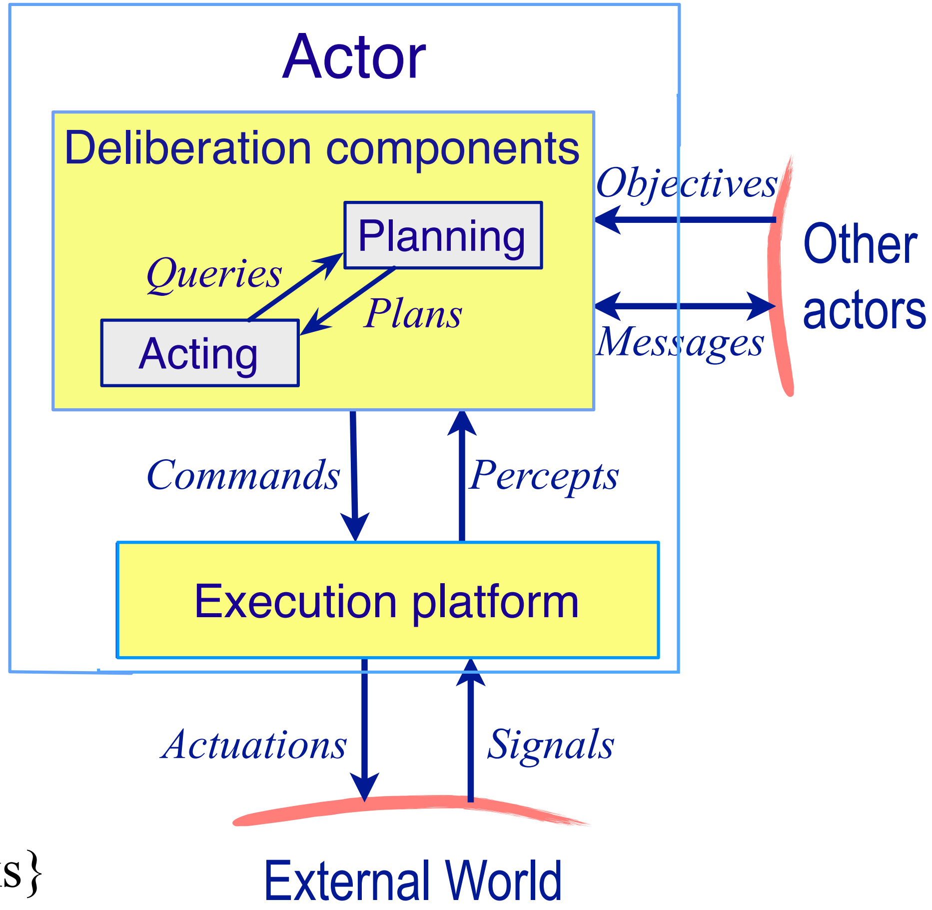
- Example: an emergency
  - If *r* isn't already handling another emergency, then
    - stop what it's doing
    - go handle the emergency

Outline

1. **Representation**
   State variables, commands, refinement methods
2. **Acting**
   Rae (Refinement Acting Engine)
3. **Planning**
   SeRPE (Sequential Refinement Planning Engine)
4. **Using Planning in Acting**
   Techniques

Rae (Refinement Acting Engine)

- Generalization and formalization of OpenPRS
- Input:
  - external tasks, goals, events
  - current state
- Output:
  - commands to execution platform
- Concurrently handle multiple tasks, goals, events
  - For each one, a *refinement stack*
- *Agenda* = {all current refinement stacks}



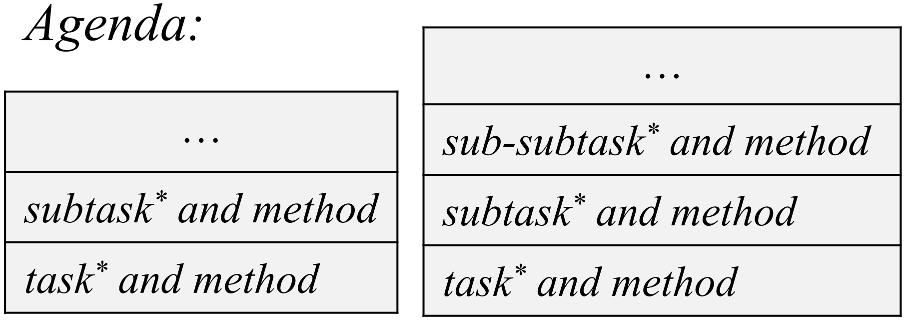
Rae (Basic Idea)

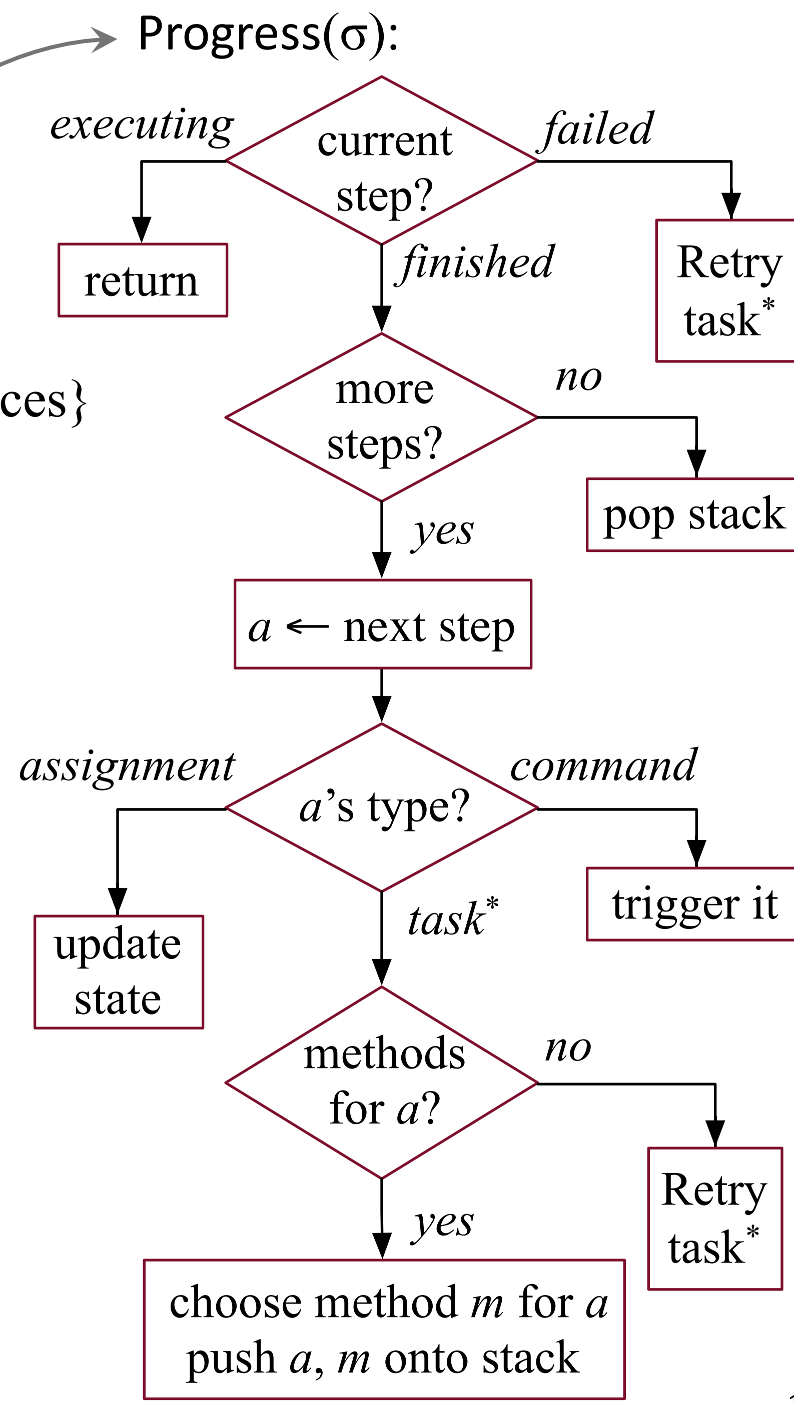
initialize *Agenda*

loop:

- for every new external task* t
  - *Candidates* = {applicable method instances}
  - arbitrarily choose m in *Candidates*
  - create refinement stack with t and m
  - add it to *Agenda*
- for each stack σ in *Agenda*
  - Progress(σ)
  - if σ is finished, remove it from *Agenda*

Agenda:



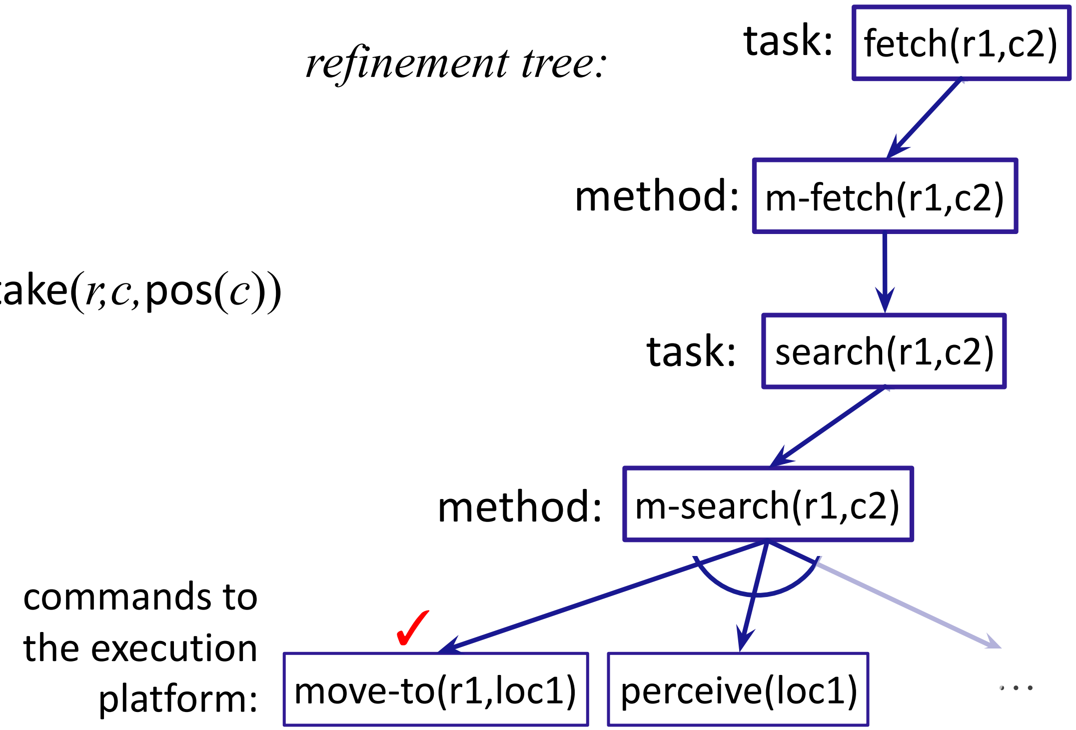


task* = task or goal or event

Example

```
m-fetch(r,c)
    task: fetch(r,c)
    pre:
    body:
        if pos(c) = unknown then
            search(r,c)
        else if loc(r) = pos(c) then take(r,c,pos(c))
        else do
            move-to(r,pos(c))
            take(r,c,pos(c))
```

refinement tree:



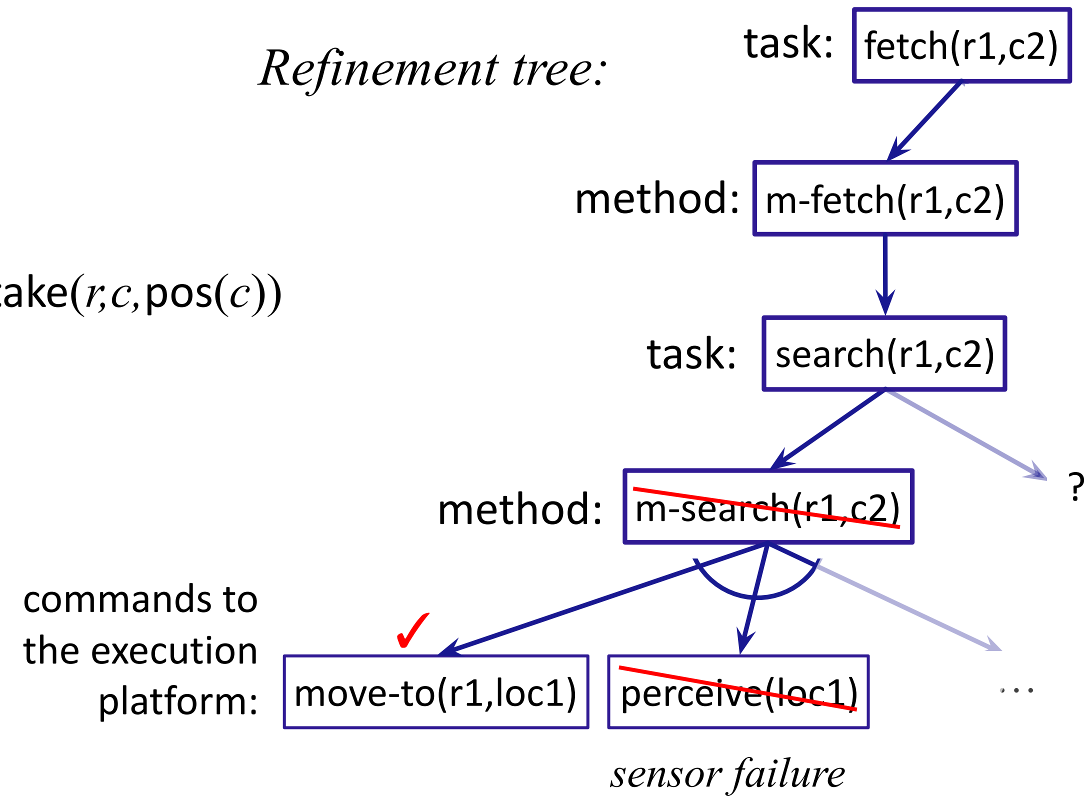

```
m-search(r,c)
    task: search(r,c)
    pre:  pos(c) = unknown
    body:
        if ∃ l (view(r,l) = F) then
            move-to(r,l) ✓
            perceive(l)
            if pos(c) = l then
                take(r,c,l)
            else search(r,c)
        else fail
```

refinement stack = current path through the refinement tree

perceive(loc1)
m-search(r1,c2)
search(r1,c2)
m-fetch(r1,c2)
fetch(r1,c2)

Recovering from Failure

m-fetch(*r*,*c*)
    task: fetch(*r*,*c*)
    pre:
    body:
        if pos(*c*) = unknown then
            search(*r*,*c*)
        else if loc(*r*) = pos(*c*) then take(*r*,*c*,pos(*c*))
        else do
            move-to(*r*,pos(*c*))
            take(*r*,*c*,pos(*c*))



~~m-search(*r*,*c*)~~
    task: search(*r*,*c*)
    pre: pos(*c*) = unknown
    body:
        if ∃ *l* (view(*r*,*l*) = F) then
            move-to(*r*,*l*) ✓
            ~~perceive(*l*)~~
            if pos(*c*) = *l* then
                take(*r*,*c*,*l*)
            else search(*r*,*c*)
        else fail

- perceive fails, so m-search fails
  - If other method instances for search, try them
  - Else m-fetch fails, look for other fetch methods
- Analogous to backtracking
  - But can't backtrack to previous state

Outline

1. **Representation**

    State variables, commands, refinement methods

2. **Acting**

    Rae (Refinement Acting Engine)

3. **Planning**

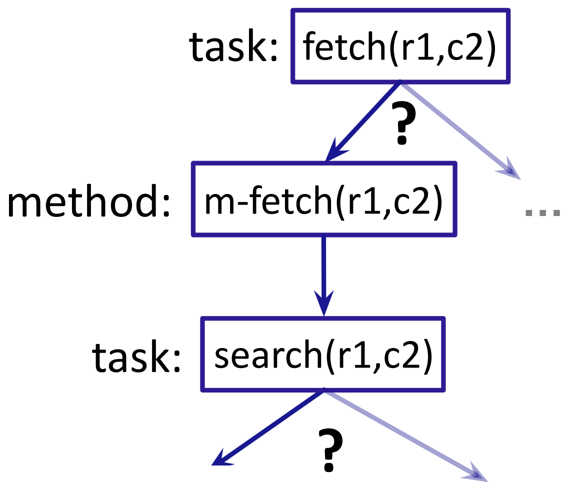
    SeRPE (Sequential Refinement Planning Engine)

4. **Using Planning in Acting**

    Techniques

Motivation

- When dealing with an event or task,
  Rae may need to make either/or choices
  - Several possible methods for a task
    - Which one to use?
  - *Agenda*: refinement stacks for several tasks
    - How to prioritize?



- Rae chooses reactively
  - Bad choices may be costly or irreversible

- Use a planner to look ahead
  - Explore the possible choices
  - Predict what will work well, what won’t

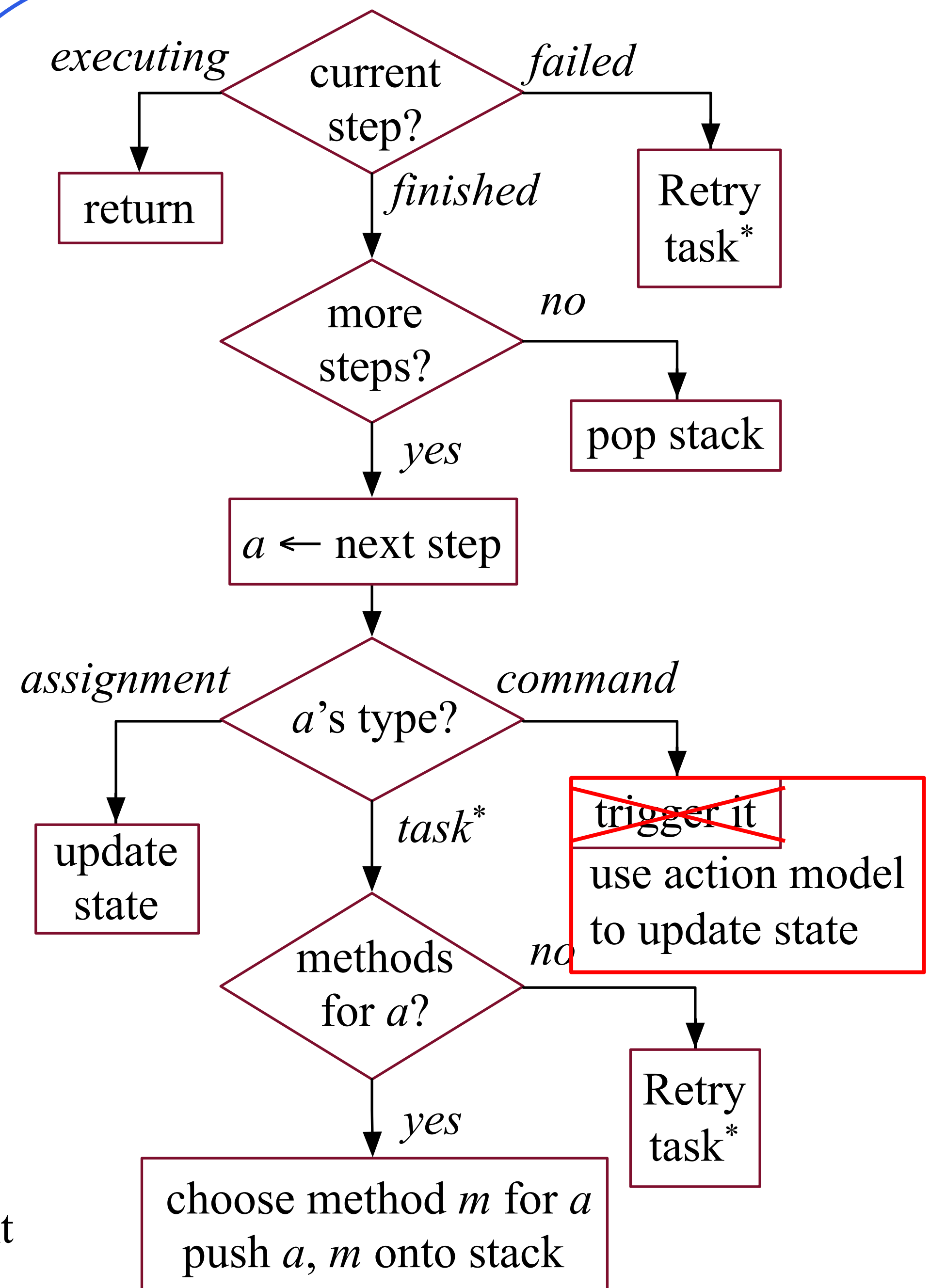
Refinement Planning

- SeRPE algorithm (pseudocode in the book)
  - Basic idea: simulate Rae
    - For each command, a *descriptive action model*
      - predict *what* the command will do, not *how*
  - Heuristic search through Rae's possible alternatives
    - Different possible method instances ⇒ different refinement trees
    - Simulate, explore consequences
- Generalization of HTN planning (the SHOP algorithm)
  - SHOP
    - Body of a method is a list of tasks $\langle \tau_1, \tau_2, \ldots, \tau_n \rangle$
    - Backtracking search through methods for each τ_i
  - SeRPE uses the same methods that Rae uses
    - Body of a method is a program to *generate* tasks and goals
    - Need to backtrack over the statements in the program

SeRPE (Basic Idea)

- SeRPE(*t*)
  - nondeterministically choose a method *m* for *t*
  - progress *m* repeatedly until it's finished

like calling Progress(σ) repeatedly



- *Nondeterministic choice*
  - Multiple possible choices, algorithm doesn't specify how to choose
    - Theoretical model: nondeterministic Turing machine considers all of them
  - Can implement as backtracking, A* search, GBFS, etc.

task* = task or goal or event

Descriptive Action Models

- Preconditions-and-effects representation
  - Like classical operators, but with state variables instead of logical atoms

- *Command*:
  - take(r,o,l):
    robot r takes object o at location l
  - put(r,o,l):
    r puts o down at location l
  - perceive(r,l):
    r perceives what objects are at l
    - can only perceive what's at current location

- *Action model*

take(r,o,l)
  pre: cargo(r) = nil, loc(r) = l, loc(o) = l
  eff: cargo(r) $\leftarrow o$, loc(o) $\leftarrow r$

put(r,o,l)
  pre: loc(r) = l, loc(o) = r
  eff: cargo(r) $\leftarrow$ nil, loc(o) $\leftarrow l$

perceive(r,l)
  ?

If we knew this in advance, perception wouldn't be necessary

Can't do the *fetch* example

Limitation

- Models of the environment are inherently incomplete
  - Even nondeterministic models don't always predict *all* possible contingencies

- Techniques can be extended to nondeterministic models
  - Part 4 of this talk
- Deterministic action models => much simpler planning algorithm
  - Use when errors are infrequent and don't have severe consequences
  - Actor can recover online

Simple Deterministic Example

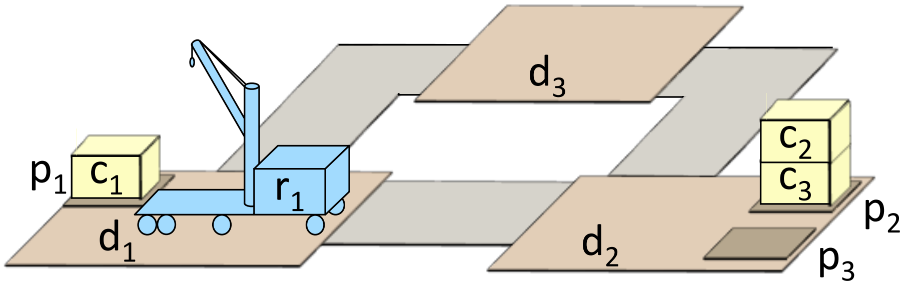
- Robot can move containers, put them into piles
- Deterministic action models

load(r,c,c',p,d)
  pre: at(p,d), cargo(r)=nil, loc(r)=d, pos(c)=c', top(p)=c
  eff: cargo(r)$\leftarrow c$, pile(c)$\leftarrow$nil, pos(c)$\leftarrow r$, top(p)$\leftarrow c'$

unload(r,c,c',p,d)
  pre: at(p,d), pos(c)=r, loc(r)=d, top(p)=c'
  eff: cargo(r)$\leftarrow$nil, pile(c)$\leftarrow p$, pos(c)$\leftarrow c'$, top(p)$\leftarrow c$

move(r,d,d')
  pre: adjacent(d,d'), loc(r)=d
  eff: loc(r)=d'

s_0 = {loc(r_1)=d_1, cargo(r_1)=nil,
pos(c_1)=nil, pile(c_1)=p_1, top(p_1)=c_1,
pos(c_2)=c_3, pile(c_2)=p_2, top(p_2)=c_2,
pos(c_3)=nil, pile(c_3)=p_2, top(p_3)=nil}



Example

task

put-in-pile(c_1,p_2)

|

method

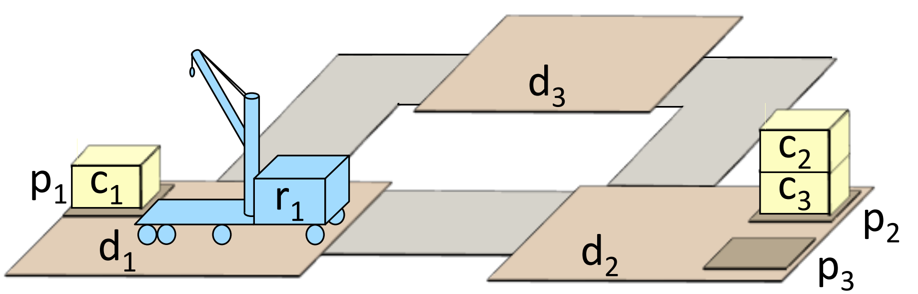
m2-put-in-pile(r_1,c_1,p_1,d_1,p_2,d_2)

refinement tree

m1-put-in-pile(c, p')
task: put-in-pile(c, p')
pre: pile(c)=p'
body: // *empty*

r_1,c_1,p_1,d_1,p_2,d_2
m2-put-in-pile(r, c, p, d, p', d')
task: put-in-pile(c,p')
pre: pile(c)=p, at(p,d), at(p',d),
$p \neq p'$, cargo(r)=nil
body: if loc(r) $\neq d$ then navigate(r,d)
uncover(c)
load(r, c, pos(c), p, d)
if loc(r) $\neq d'$ then
navigate(r,d')
unload(r, c, top(p'), p', d)

s = {loc(r_1)=d_1, cargo(r_1)=nil,
pos(c_1)=nil, pile(c_1)=p_1, top(p_1)=c_1,
pos(c_2)=c_3, pile(c_2)=p_2, top(p_2)=c_2,
pos(c_3)=nil, pile(c_3)=p_2, top(p_3)=nil}



Example

task
put-in-pile(c_1,p_2)
|
method
m2-put-in-pile(r_1,c_1,p_1,d_1,p_2,d_2)
task
uncover(c_1)
method
m1-uncover(c_1)
(no children)

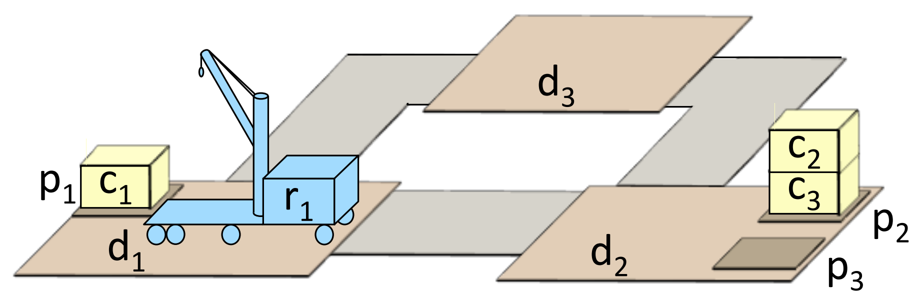
m1-uncover(c)
  task: uncover(c)
  pre: top(pile(c))=c
  body: // *empty*

m2-uncover(r,c,c,p',d)
  task: uncover(c)
  pre: top(pile(c)) $\neq c$
  body: …

m1-put-in-pile(c, p')
  task: put-in-pile(c, p')
  pre: pile(c)=p'
  body: // *empty*

r_1,c_1,p_1,d_1,p_2,d_2
m2-put-in-pile(r, c, p, d, p', d')
  task: put-in-pile(c,p')
  pre: pile(c)=p, at(p,d), at(p',d),
    $p \neq p'$, cargo(r)=nil
  body: if loc(r) $\neq d$ then navigate(r,d)
    uncover(c)
    load(r, c, pos(c), p, d)
    if loc(r) $\neq d'$ then
      navigate(r,d')
    unload(r, c, top(p'), p', d)

s = {loc(r_1)=d_1, cargo(r_1)=nil,
  pos(c_1)=nil, pile(c_1)=p_1, top(p_1)=c_1,
  pos(c_2)=c_3, pile(c_2)=p_2, top(p_2)=c_2,
  pos(c_3)=nil, pile(c_3)=p_2, top(p_3)=nil}



Example

task
put-in-pile(c_1,p_2)

method
m2-put-in-pile(r_1,c_1,p_1,d_1,p_2,d_2)

task
uncover(c_1)

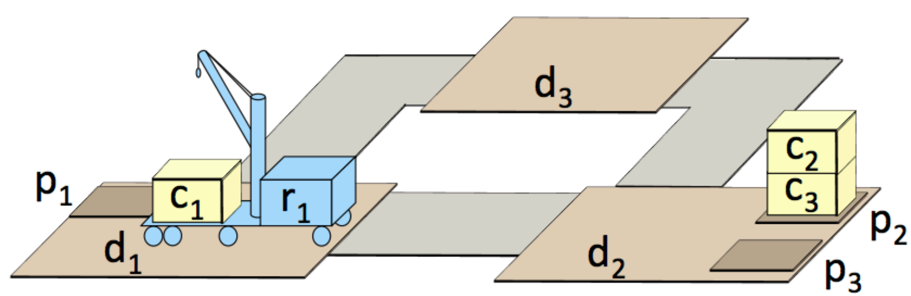
method
m1-uncover(c_1)
(no children)

action
load(r_1,c_1,nil,p_1,d_1)

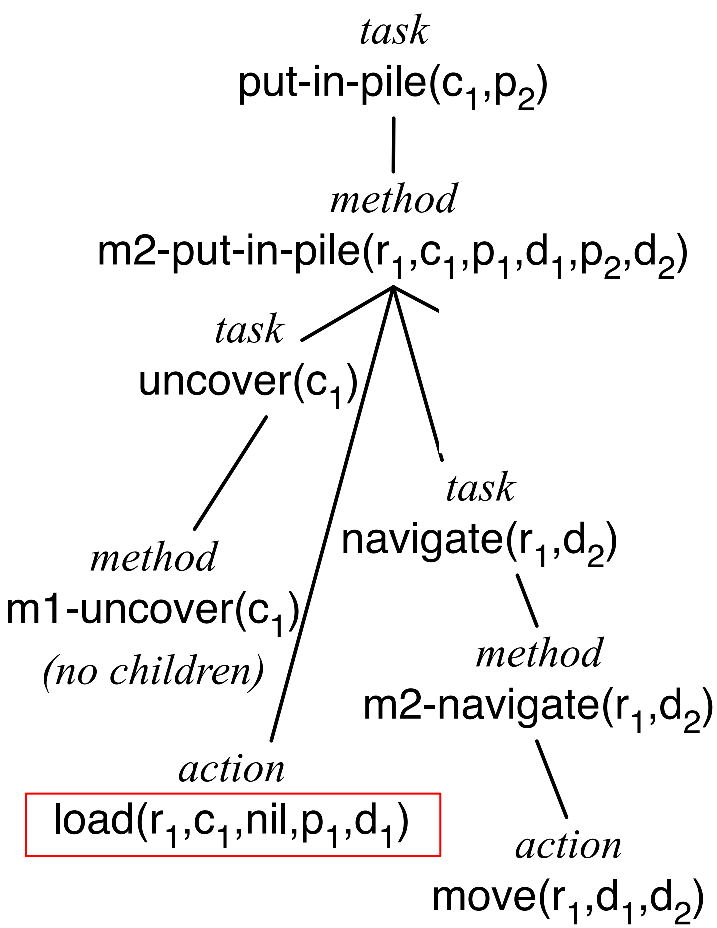
m1-put-in-pile(c, p')
  task: put-in-pile(c, p')
  pre: pile(c)=p'
  body: // *empty*

r_1,c_1,p_1,d_1,p_2,d_2
m2-put-in-pile(r, c, p, d, p', d')
  task: put-in-pile(c,p')
  pre: pile(c)=p, at(p,d), at(p',d),
    $p \neq p'$, cargo(r)=nil
  body: if loc(r) $\neq d$ then navigate(r,d)
    uncover(c)
    load(r, c, pos(c), p, d)
    if loc(r) $\neq d'$ then
      navigate(r,d')
    unload(r, c, top(p'), p', d)

s = {loc(r_1)=d_1, cargo(r_1)=c_1,
pos(c_1)=r_1, pile(c_1)=nil, top(p_1)=nil,
pos(c_2)=c_3, pile(c_2)=p_2, top(p_2)=c_2,
pos(c_3)=nil, pile(c_3)=p_2, top(p_3)=nil}



Example

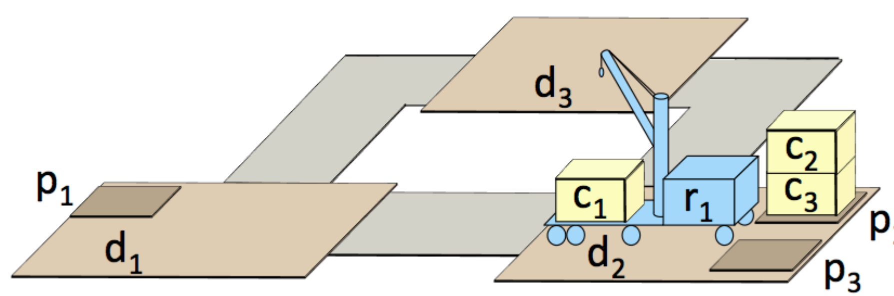


m1-put-in-pile(c, p')
- task: put-in-pile(c, p')
- pre: pile(c)=p'
- body: // *empty*

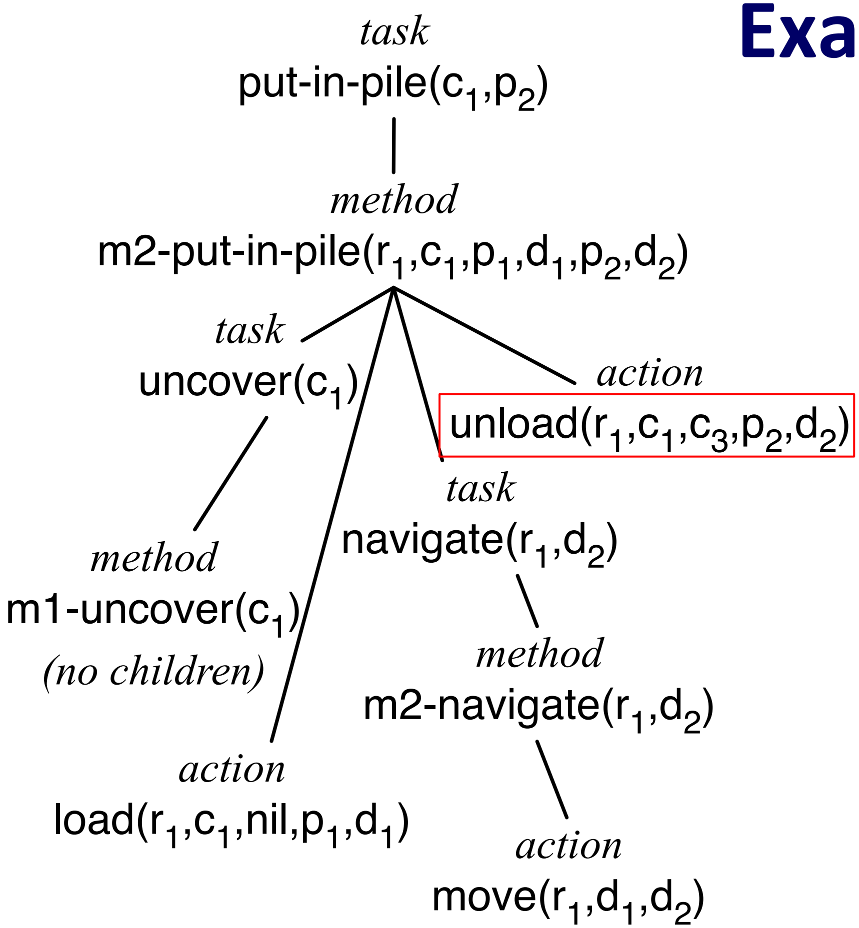
r_1,c_1,p_1,d_1,p_2,d_2

m2-put-in-pile(r, c, p, d, p', d')
- task: put-in-pile(c,p')
- pre: pile(c)=p, at(p,d), at(p',d), $p \neq p'$, cargo(r)=nil
- body: if loc(r) $\neq d$ then navigate(r,d)
  - uncover(c)
  - load(r, c, pos(c), p, d)
  - if loc(r) $\neq d'$ then
    - navigate(r,d')
  - unload(r, c, top(p'), p', d)

s = {loc(r_1)=d_2, cargo(r_1)=c_1,
pos(c_1)=r_1, pile(c_1)=nil, top(p_1)=nil,
pos(c_2)=c_3, pile(c_2)=p_2, top(p_2)=c_2,
pos(c_3)=nil, pile(c_3)=p_2, top(p_3)=nil}



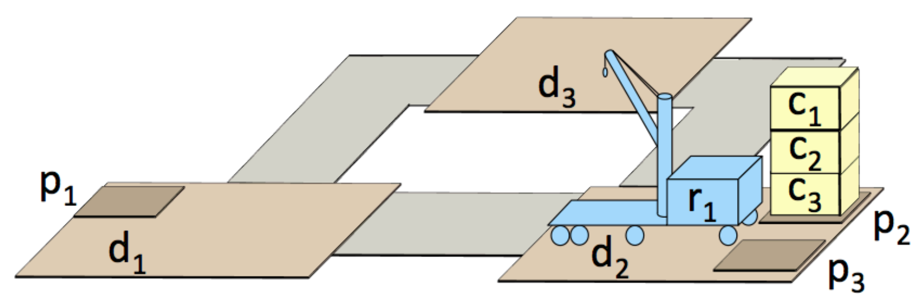
Example



m1-put-in-pile(c, p')
  task: put-in-pile(c, p')
  pre: pile(c)=p'
  body: // *empty*

$r_1, c_1, p_1, d_1, p_2, d_2$
m2-put-in-pile(r, c, p, d, p', d')
  task: put-in-pile(c, p')
  pre: pile(c)=p, at(p, d), at(p', d),
    $p \neq p'$, cargo(r)=nil
  body: if loc(r) $\neq d$ then navigate(r, d)
    uncover(c)
    load(r, c, pos(c), p, d)
    if loc(r) $\neq d'$ then
      navigate(r, d')
    unload(r, c, top(p'), p', d)

s = {loc(r_1)=d_2, cargo(r_1)=nil,
  pos(c_1)=c_2, pile(c_1)= p_2, top(p_1)=nil,
  pos(c_2)=c_3, pile(c_2)=p_2, top(p_2)=c_1,
  pos(c_3)=nil, pile(c_3)=p_2, top(p_3)=nil}



Heuristics for SeRPE

- SeRPE(*t*)
  - nondeterministically choose a method *m* for *t*
  - progress *m* repeatedly until it's finished
- *Nondeterministic choice*
  - Multiple possible choices, algorithm doesn't specify how to choose
    - Theoretical model: nondeterministic Turing machine considers all of them
  - Can implement as backtracking, A* search, GBFS, etc.

- What heuristic function?
  - Open problem
  - In some cases classical-planning heuristics can be used, in other cases they become intractable [Shivashankar *et al.*, ECAI-2016]
- *Ad hoc* approaches:
  - try methods in the order that they're given
  - domain-specific estimates
  - statistical data on how well each method works

task* = task or goal or event

Outline

1. **Representation**

   State variables, commands, refinement methods

2. **Acting**

   Rae (Refinement Acting Engine)

3. **Planning**

   SeRPE (Sequential Refinement Planning Engine)

4. **Using Planning in Acting**

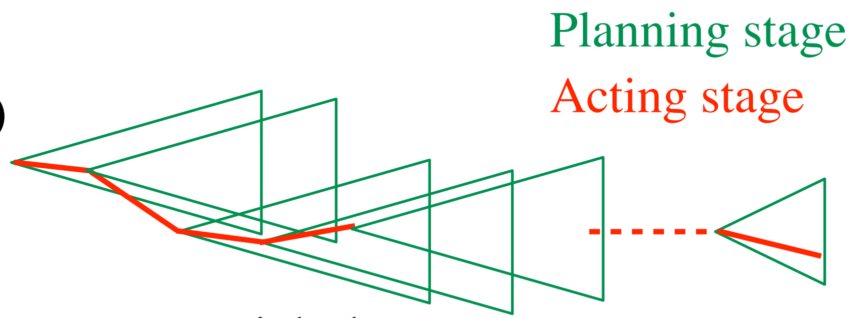
   Techniques

Using Planning in Acting

- Book describes two approaches:
  - REAP (Refinement Engine for Acting and Planning)
    - RAE-like actor
      - uses SeRPE-like planning whenever it needs to make a choice
    - Complicated; I'll skip it
  - Non-hierarchical actor with refinement planning
    - Much simpler
    - Illustrates the basic issues

Using Planning in Acting

Run-Lookahead
    while ($s \leftarrow$ observed state) doesn't satisfy g do
        $\pi \leftarrow$ Lookahead(Σ,s,g)
        if π = failure then return failure
        $a \leftarrow$ pop-first-action(π); perform(a)



- Lookahead: modified version of SeRPE
  - Searches part of the search space, returns a partial plan π
- Run-Lookahead executes the first action of π, then calls Lookahead again
  - Somewhat like minimax game tree search in chess
- Useful when unpredictable things are likely to happen
  - Replans immediately
- Potential problem:
  - May pause repeatedly while waiting for Lookahead to return
  - What if ξ changes during the wait?

Using Planning in Acting

Run-Lazy-Lookahead
  while ($s \leftarrow$ observed state) $\not\models g$ do
    $\pi \leftarrow$ Lookahead(Σ, s, g)
    if π = failure then return failure
    while $\pi \neq \varnothing$ and $s \not\models g$ and Simulate(π) $\neq$ failure do
      $a \leftarrow$ pop-first-action(π); perform(a); $s \leftarrow$ observed state

- Call Lookahead, execute the plan as far as possible, don't call Lookahead again unless necessary
- Simulate tests whether the rest of the plan will execute correctly
  - Could just compute $\gamma(s,\pi)$, or could do something more detailed
    - lower-level refinement, physics-based simulation
- Potential problems
  - May wait too long to replan, not notice problems until it's too late
  - Might miss opportunities to replace π with a better plan

Using Planning in Acting

```
Run-Lazy-Lookahead(Σ, g)
    s ← abstraction of observed state ξ
    while s ⊭ g do
        π ← Lookahead(Σ, s, g)
        if π = failure then return failure
        while π ≠ ⟨⟩ and s ⊭ g and Simulate(Σ, s, g, π) ≠ failure do
            a ← pop-first-action(π);  perform(a)
            s ← abstraction of observed state ξ
```

- Call Lookahead, execute the plan as far as possible, don't call Lookahead again unless necessary
- Simulate tests whether the plan will execute correctly
  - Could just compute $\gamma(s,\pi)$, or could do something more detailed
    - lower-level refinement, physics-based simulation
- Potential problem: may wait too long to replan
  - Might not notice problems until it's too late
  - Might miss opportunities to replace π with a better plan

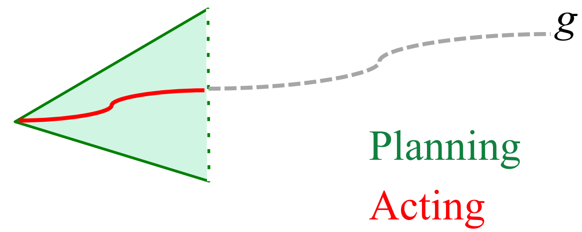
Using Planning in Acting

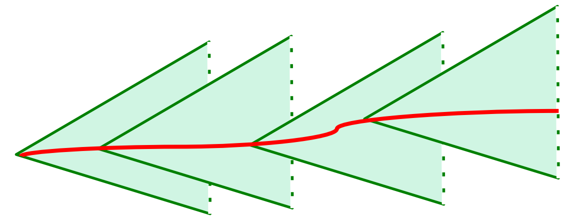
Run-Concurrent-Lookahead(Σ, g)
    $\pi \leftarrow \langle\rangle$; $s \leftarrow$ abstraction of observed state ξ
    thread 1:   // threads 1 and 2 run concurrently
        loop
            $\pi \leftarrow$ Lookahead(Σ, s, g)
    thread 2:
        loop
            if $s \models g$ then return success
            else if $\pi =$ failure then return failure
            else if $\pi \neq \langle\rangle$ and Simulate$(\Sigma, s, g, \pi) \neq$ failure then
                $a \leftarrow$ pop-first-action(π); perform(a)
                $s \leftarrow$ abstraction of observed state ξ

- Avoids Run-Lookahead's problem with waiting for Lookahead to return
- May detect problems & opportunities earlier than Run-Lazy-Lookahead
- May miss some that Run-Lookahead could find
  (not a problem if Lookahead is fast)

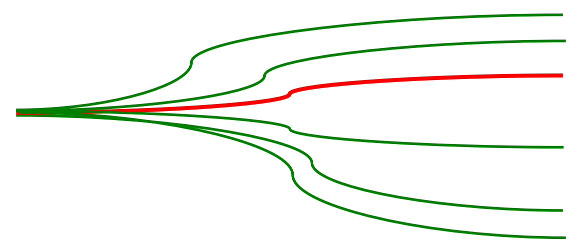
Lookahead

- *Receding horizon*
  - Cut off search before reaching *g*
    - e.g., bound on search depth or time
  - Horizon "recedes" on the actor's successive calls to the planner

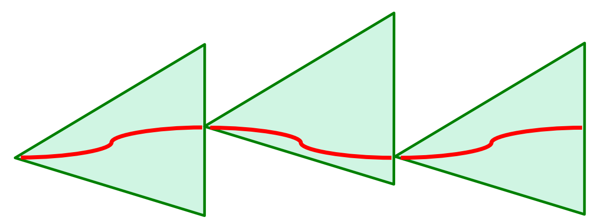




- *Sampling*
  - Try a few (e.g., randomly chosen) depth-first rollouts, take the one that looks best



- *Subgoaling*
  - Instead of planning for ultimate goal *g*, plan for a subgoal g_i
  - When finished with g_i, actor calls planner on next subgoal g_{i+1}



- Can use combinations of these

Example

- **Killzone 2**
  - video game, Oct 2009
  - I didn't learn about it until ≈ 2012
- Planner based on SHOP (which SeRPE generalizes)
  - Plans enemy actions at the squad level
- Don't want to get the best possible plan
  - Need actions that appear believable and consistent to human users
  - Need them very quickly
- Use subgoaling
  - e.g., "get to shelter"
  - solution plan is maybe 4–6 actions long
- Replan several times per second as the world changes

Implementation of Rae and SeRPE

- Rae and SeRPE are new algorithms
  - Developed while writing the book
- Some of my students are implementing them in Python
  - Nearly finished
  - We'll make the implementations available
- Demo: Rae playing Pac-Man
  - https://youtu.be/NtLwI7Pc8U8
  - Author: Zheng Yan

Summary

- Refinement Acting Engine (RAE)
  - Body of a refinement method is a simple program that includes commands to the execution platform
- Refinement planning (SeRPE)
  - Simulate RAE's operation on a single task/event/goal
  - Limitation: deterministic action models
- Acting and planning
  - Lookahead: search part of the search space, return a partial solution

Relation to the Book

- 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

Any questions?