Deliberation with Refinement Methods

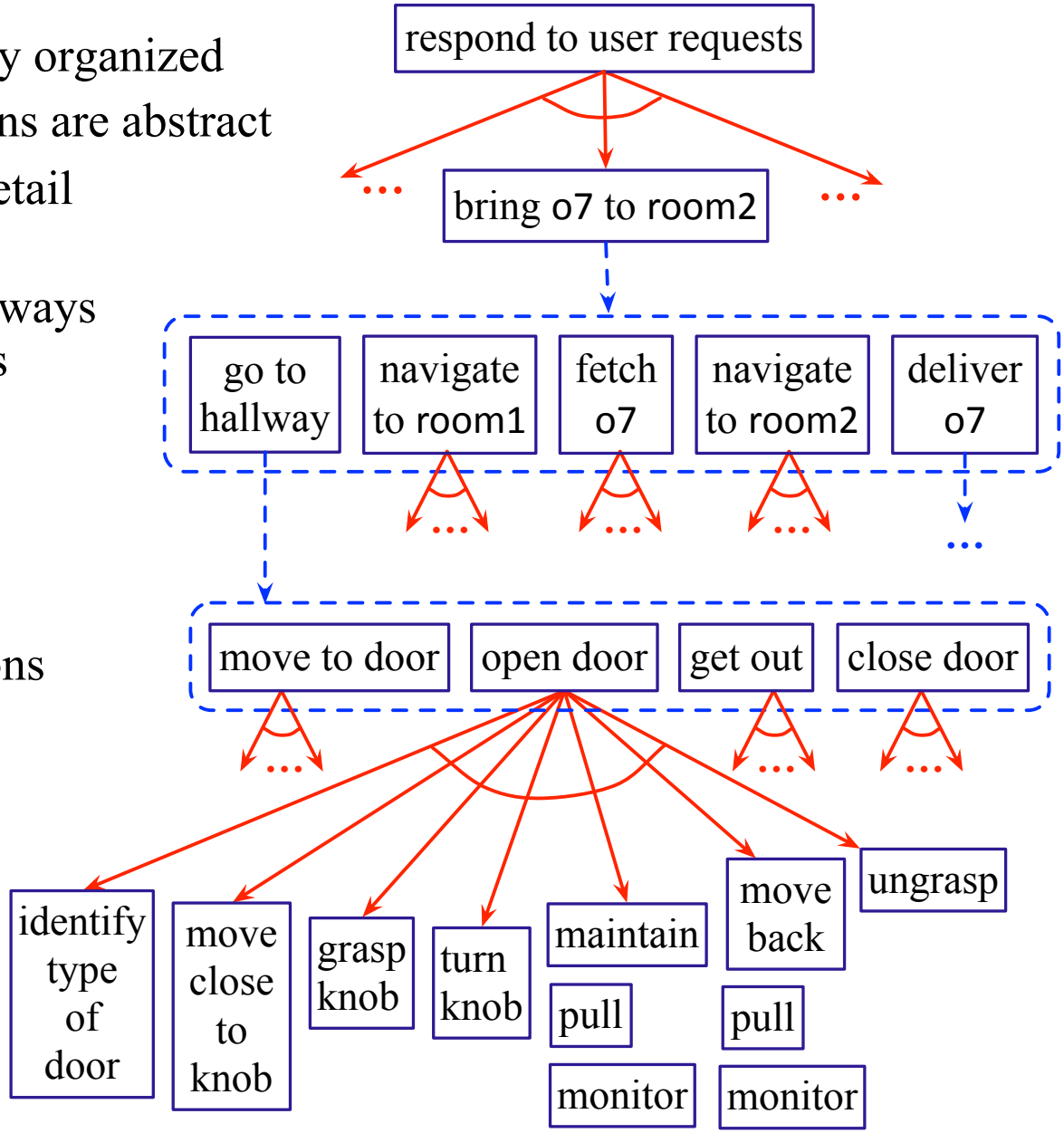
Malik Ghallab, Dana Nau, Paolo Traverso
Automated Planning and Acting
Cambridge University Press

IJCAI 2016 Tutorial
New York, July 11th,2016

Last updated 7/11/16

Motivation

- Deliberation is hierarchically organized
  - At high levels, the actions are abstract
  - At lower levels, more detail
- Refine abstract actions into ways of carrying out those actions
  - How?
- In some cases, can use predictive models
  - Precondition-effect actions
  - State-space planning
- In others, need operational models
  - *Refinement methods*

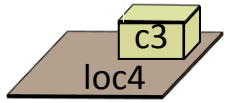


Outline

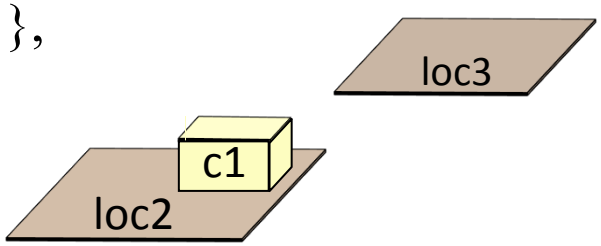
1. **Representation**
   a. State variables, commands, refinement methods
   b. Example
2. **Acting**
   a. `Rae` (Refinement Acting Engine)
   b. Example
   c. Extensions
3. **Planning**
   a. Motivation and basic ideas
   b. Deterministic action models
   c. `SeRPE` (Sequential Refinement Planning Engine)
4. **Using Planning in Acting**
   a. Techniques
   b. Caveats

1a. State-Variable Representation …

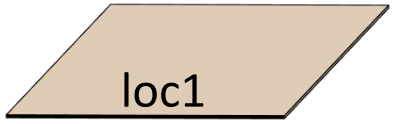
- State-variable representation

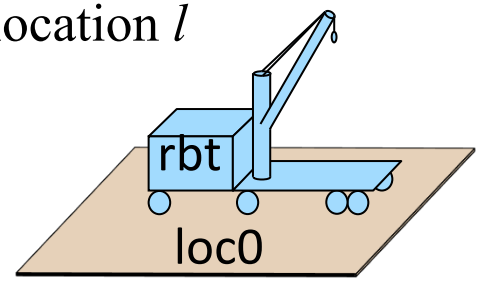


- Objects: *Robots* = {rbt}, *Containers* = {c1, c2, c3, …},
  Locations = {loc0, loc1, loc2, …}



- *State variables*: syntactic terms to which we can assign values
  - loc(r) ∈ *Locations*
  - load(r) ∈ *Containers* ∪ {nil}
  - pos(c) ∈ *Locations* ∪ *Robots* ∪ {unknown}
  - view(r,l) ∈ {T, F} – whether robot r has looked at location l
    - r can only see what's at its current location



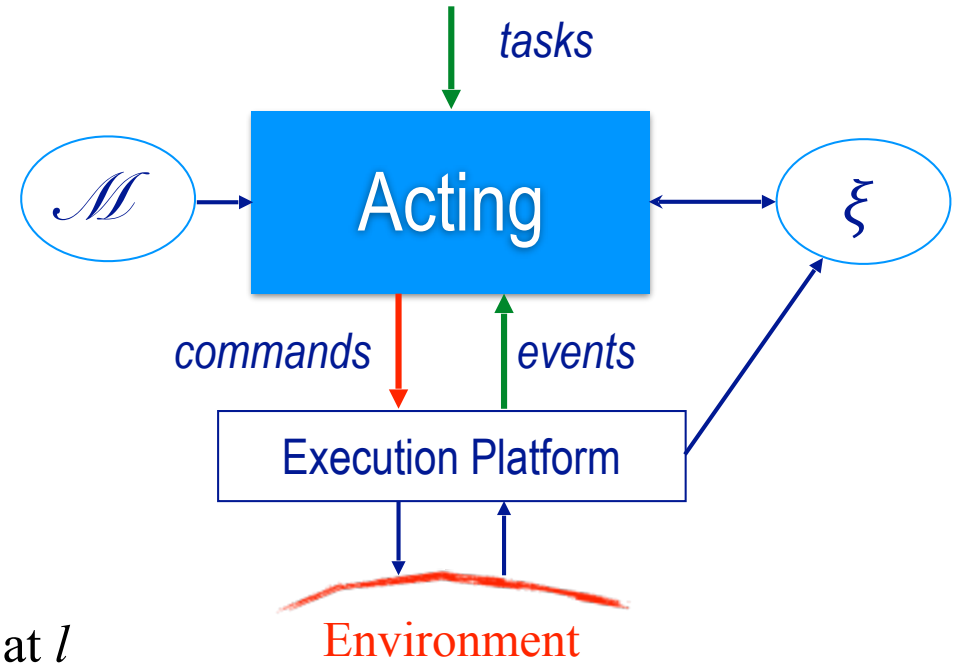


- *State*: assign a value to each state variable
  - {loc(rbt) = loc0, pos(c1) = loc2, pos(c3) = loc4, pos(c2) = unknown, …}

Details: *Automated Planning and Acting,* Sections 2.1 and 3.1.1

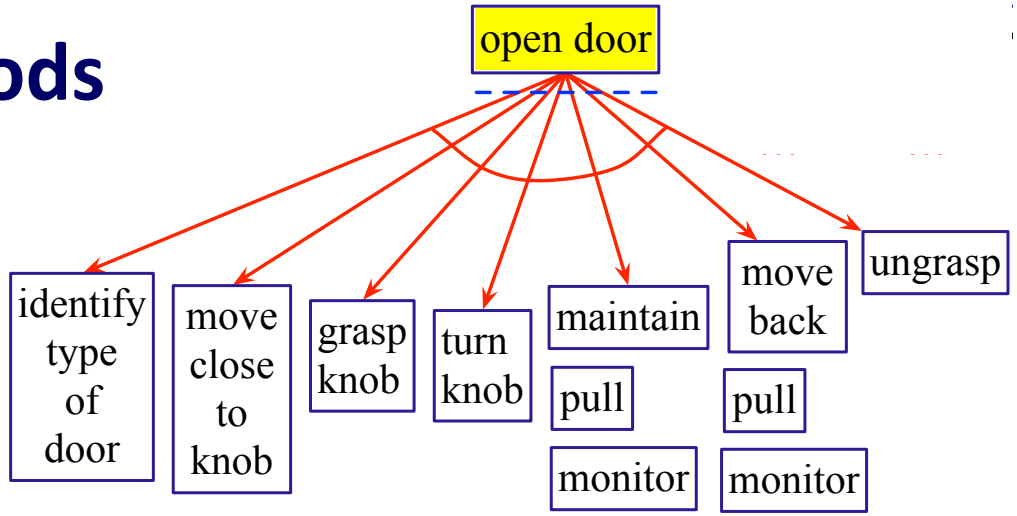
Commands

- *Command*: primitive function that the execution platform can perform
  - take(r,o,l):
    robot r takes object o at location l
  - put(r,o,l):
    r puts o at location l
  - perceive(r,l):
    robot r perceives what objects are at l
    - r can only perceive what's at its current location
  - …



Tasks and Methods

- *Task*: an activity for the actor to perform
- For each task, a set of *refinement methods*
- *Operational* models:
  - tell *how to perform* the task
  - don't predict *what the effects* will be



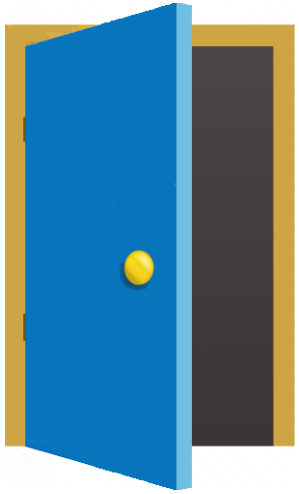
method-name(arg_1, …, arg_k)
  task:* *task-identifier*
  pre: *test*
  body: *a program*

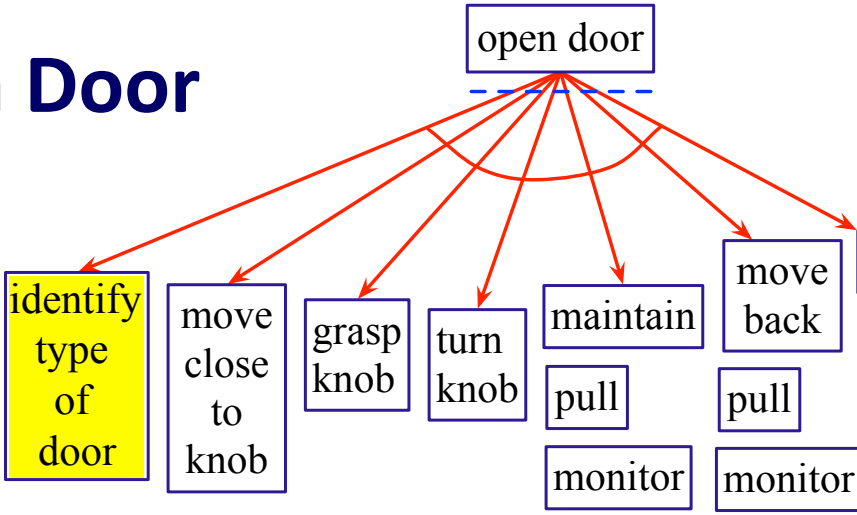
- May contain
  - assignment statements
  - control constructs
    - if-then-else, while, loop, etc.
  - tasks to perform
  - commands to the execution platform

*Can also have methods for events, goals

1b. Example: Opening a Door

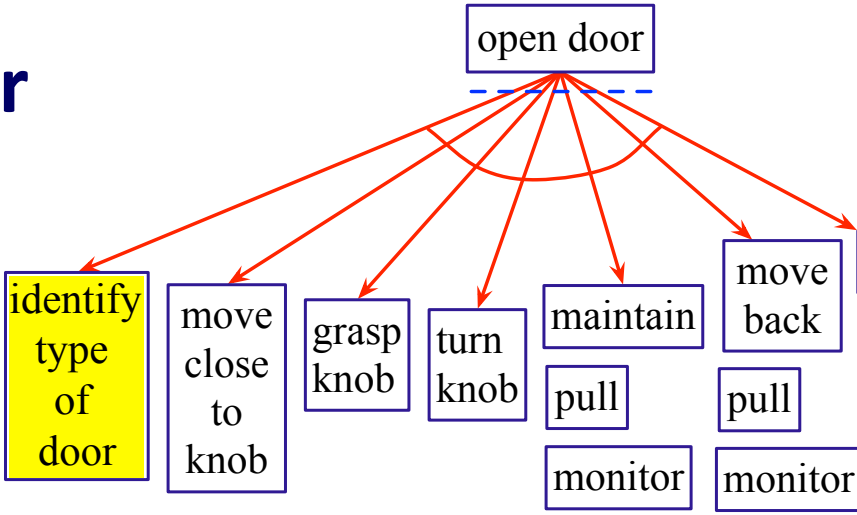
- Many different methods, depending on what kind of door
  - Sliding or hinged?

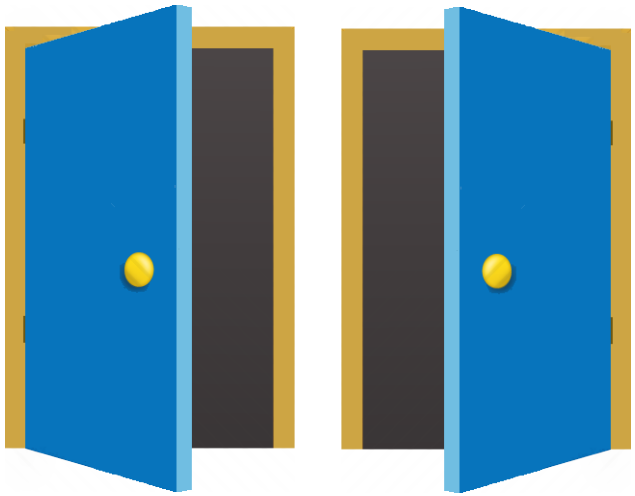




Opening a Door

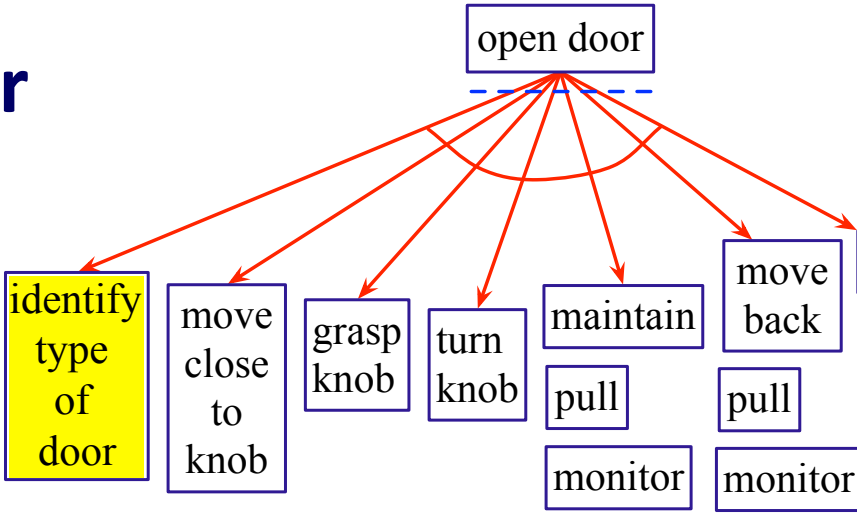
- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?

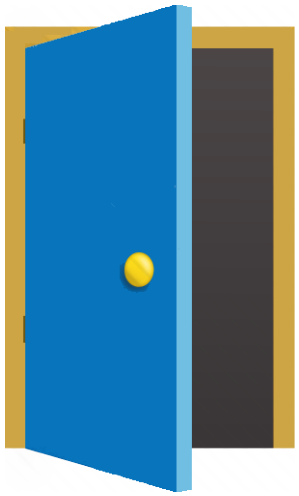


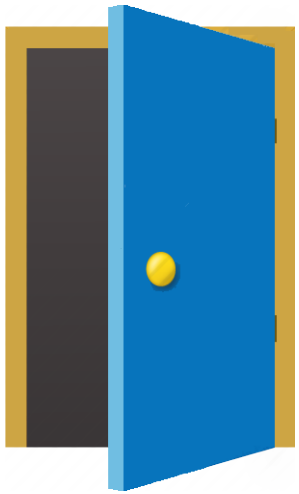


Opening a Door

- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?

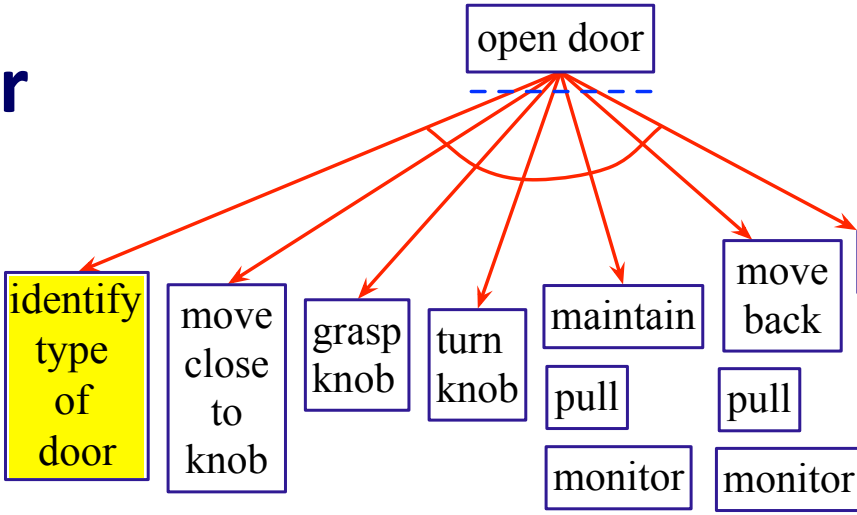


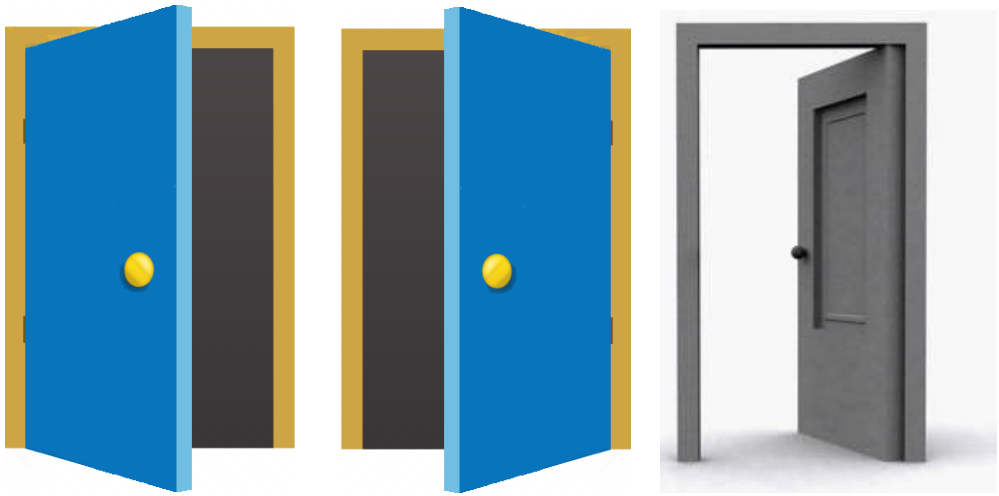




Opening a Door

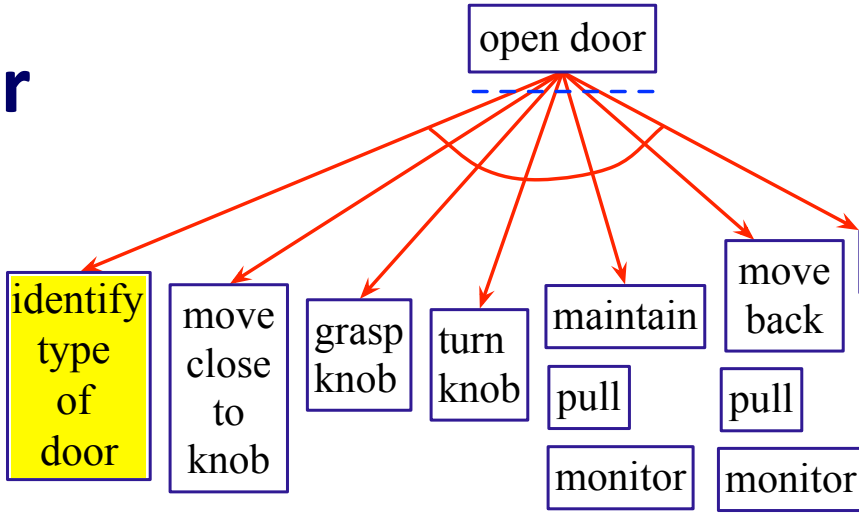
- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, …

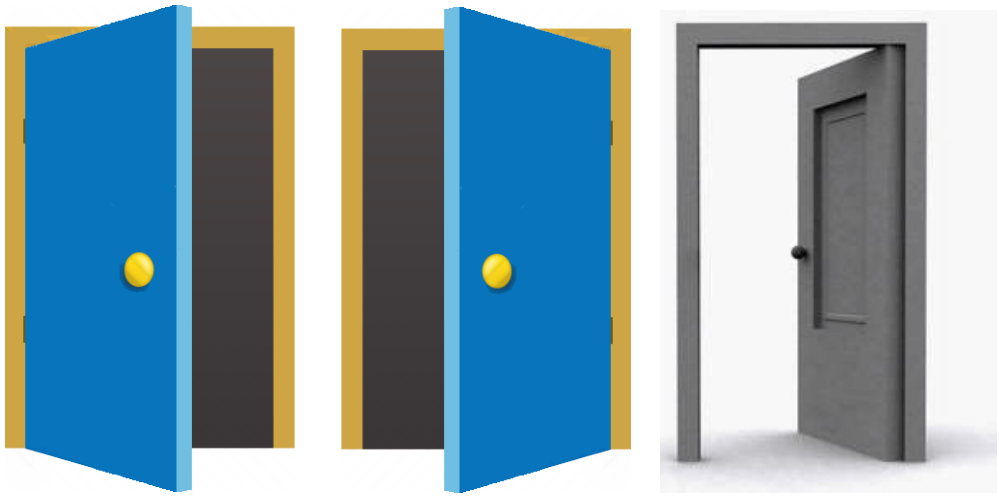




Opening a Door

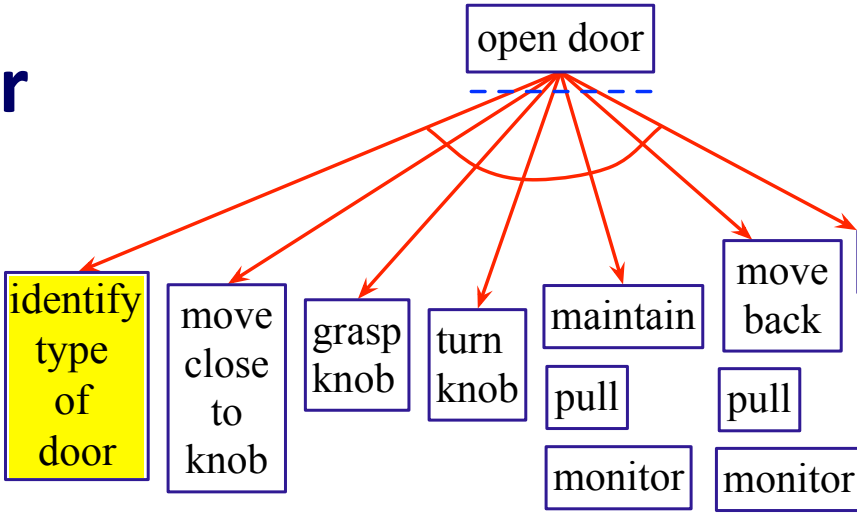
- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, …

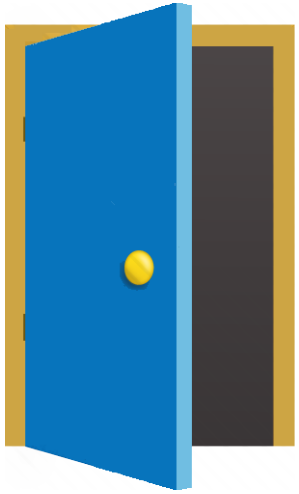


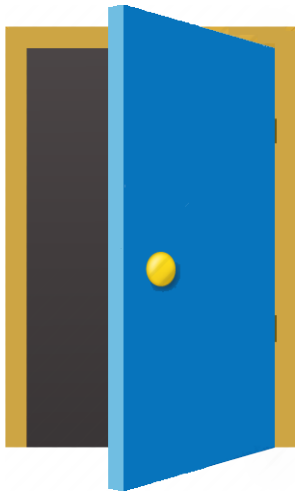


Opening a Door

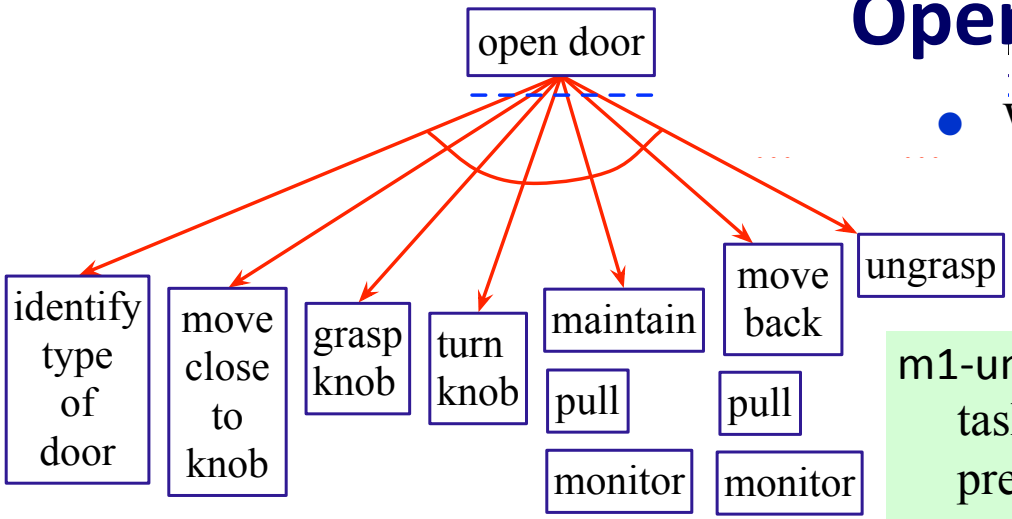
- Many different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
  - Open toward or away?
  - Knob, lever, push bar, pull handle, push plate, something else?



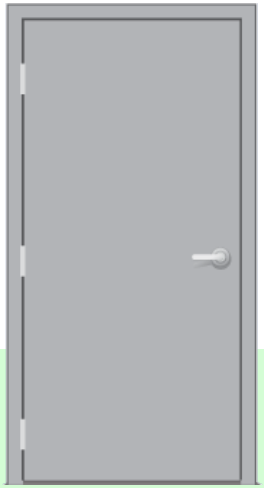




Opening a Door



- What kind:
  - Hinged on left, opens toward us, lever handle



m-opendoor(r,d,l,h)
  task: opendoor(r,d)
  pre: loc(r) = l ∧ adjacent(l,d)
    ∧ handle(d,h)
  body:
    while ¬reachable(r,h) do
      move-close(r,h)
    monitor-status(r,d)
    if door-status(d)=closed then
      unlatch(r,d)
    throw-wide(r,d)
    end-monitor-status(r,d)

m1-unlatch(r,d,l,o)
  task: unlatch(r,d)
  pre: loc(r,l) ∧ toward-side(l,d)
    ∧ side(d,left) ∧ type(d,rotate) ∧ handle(d,o)
  body: grasp(r,o)
    turn(r,o,alpha1)
    pull(r,val1)
    if door-status(d)=cracked then ungrasp(r,o)
    else fail

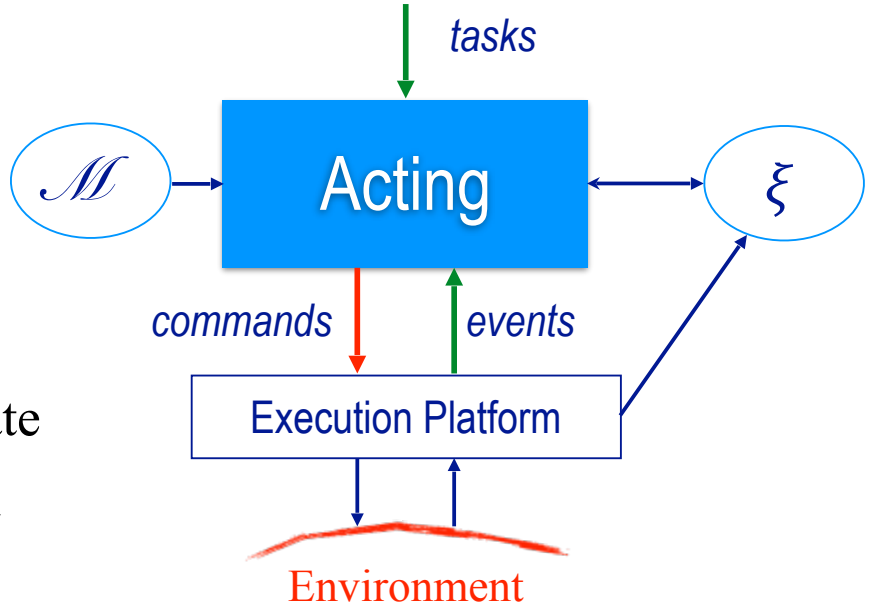
m1-throw-wide(r,d,l,o)
  task: throw-wide(r,d)
  pre: loc(r,l) ∧ toward-side(l,d)
    ∧ side(d,left) ∧ type(d,rotate)
    ∧ handle(d,o) ∧ door-status(d)=cracked
  body: grasp(r,o)
    pull(r,val1)
    move-by(r,val2)

Outline

1. **Representation**
   a. State variables, commands, refinement methods
   b. Example
2. **Acting**
   a. `Rae` (Refinement Acting Engine)
   b. Example
   c. Extensions
3. **Planning**
   a. Motivation and basic ideas
   b. Deterministic action models
   c. `SeRPE` (Sequential Refinement Planning Engine)
4. **Using Planning in Acting**
   a. Techniques
   b. Caveats

2a. Rae (Refinement Acting Engine)

- Based on OpenPRS
  - Programming language, open-source robotics software
  - Deployed in many applications



- Input: external tasks, events, current state
- Output: commands to execution platform

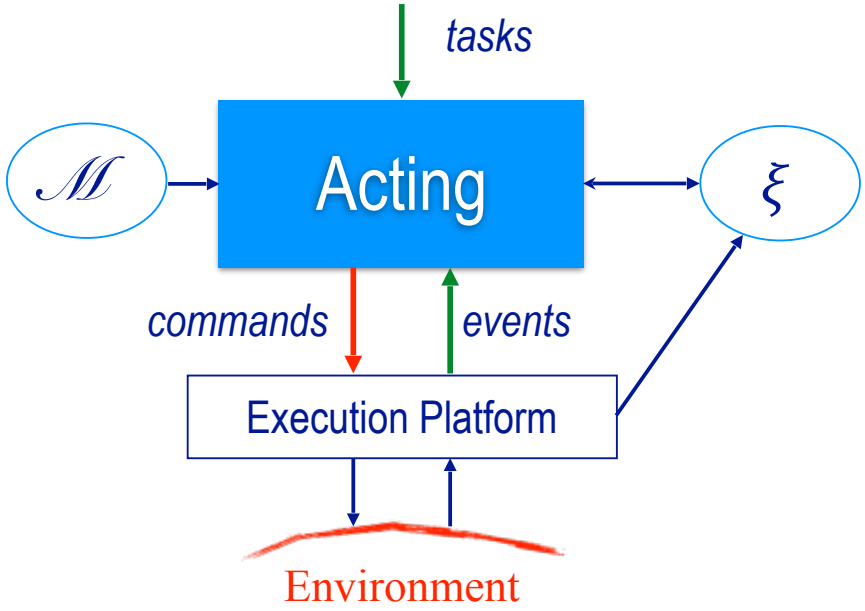
- Perform tasks/events in parallel
  - Purely reactive, no lookahead

- For each task/event, a *refinement stack*
  - current path in Rae's search tree for the task/event
- *Agenda* = {all current refinement stacks}

Details: *Automated Planning and Acting*, Section 3.2

Rae (Refinement Acting Engine)

- loop:
  - if new external tasks/events then add them to *Agenda*
  - Progress each stack in *Agenda*



Rae($\mathcal{M}$)
  $Agenda \leftarrow \varnothing$
  loop
    until the input stream of external tasks and events is empty do
      read τ in the input stream
      $Candidates \leftarrow$ Instances($\mathcal{M}, \tau, \xi$)
      if $Candidates = \varnothing$ then output("failed to address" τ)
      else do
        arbitrarily choose $m \in Candidates$
        $Agenda \leftarrow Agenda \cup \{\langle(\tau, m, \mathsf{nil}, \varnothing)\rangle\}$
    for each $stack \in Agenda$ do
      Progress($stack$)
      if $stack = \varnothing$ then $Agenda \leftarrow Agenda \setminus \{stack\}$

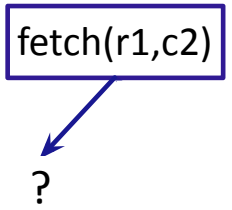
2b. Example

```
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
```

Refinement stack:

- fetch(r1,c2)



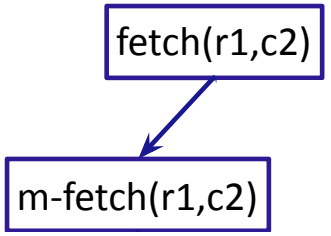
Example

```
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
```

Refinement stack:

- fetch(r1,c2): m-fetch(r1,c2)



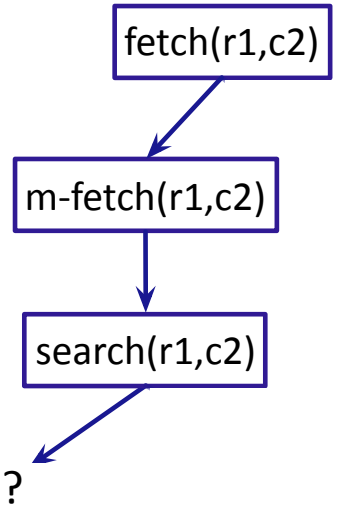
Example

```
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
```

Refinement stack:

- search(r1,c2)
- fetch(r1,c2): m-fetch(r1,c2)



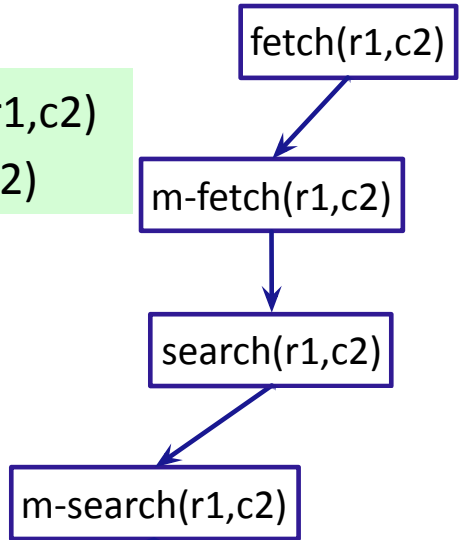
Example

```
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
```

Refinement stack:

- search(r1,c2): m-search(r1,c2)
- fetch(r1,c2): m-fetch(r1,c2)

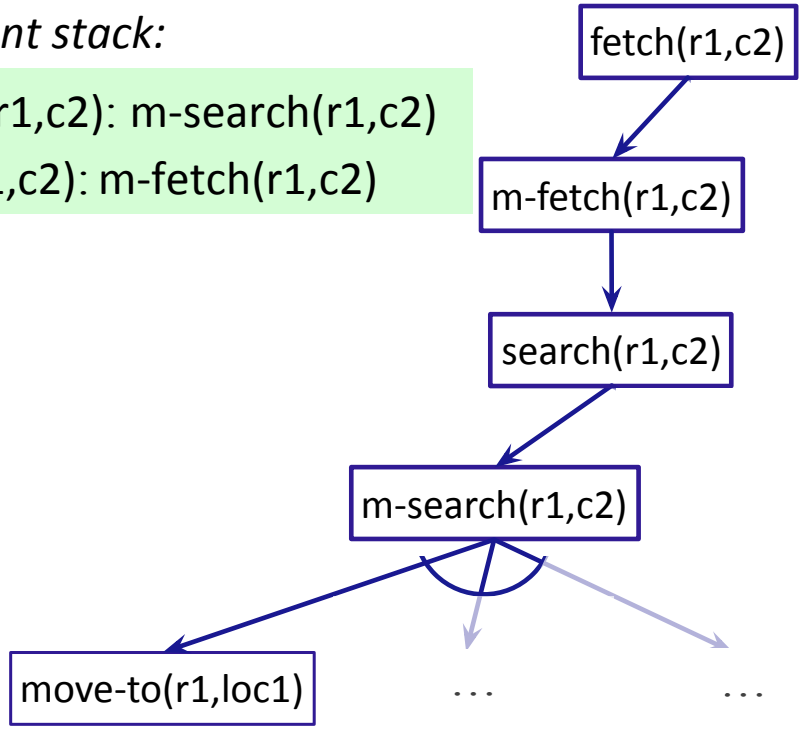


Example

```
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
```

Refinement stack:

- search(r1,c2): m-search(r1,c2)
- fetch(r1,c2): m-fetch(r1,c2)

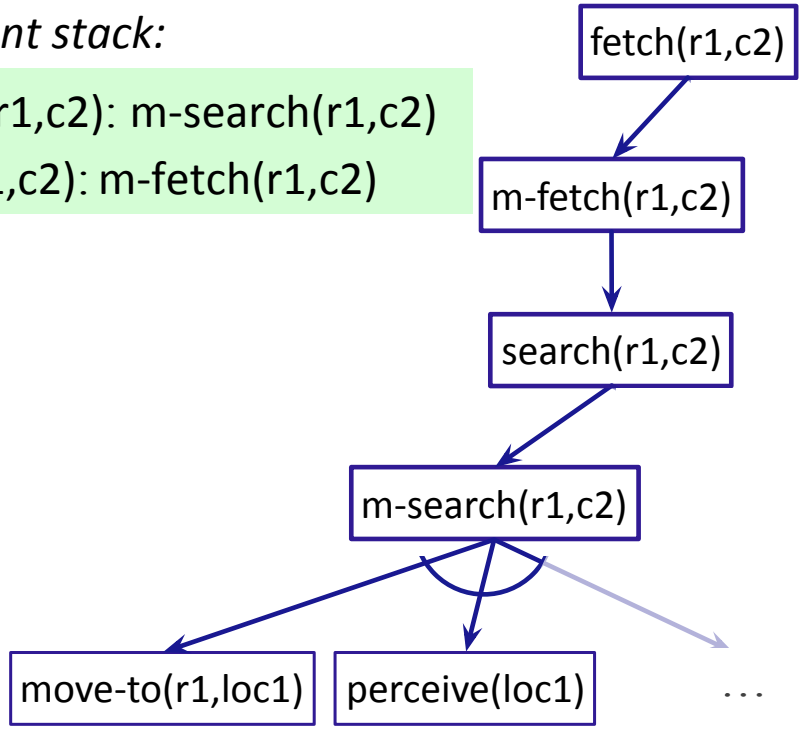


Example

```
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
```

Refinement stack:

- search(r1,c2): m-search(r1,c2)
- fetch(r1,c2): m-fetch(r1,c2)



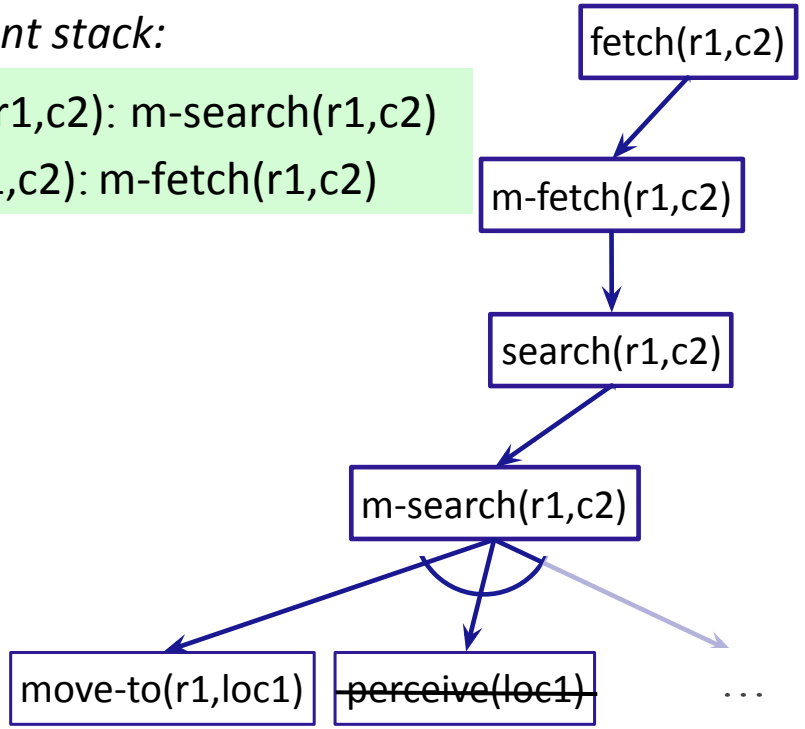
Example

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

Refinement stack:

- search(r1,c2): m-search(r1,c2)
- fetch(r1,c2): m-fetch(r1,c2)



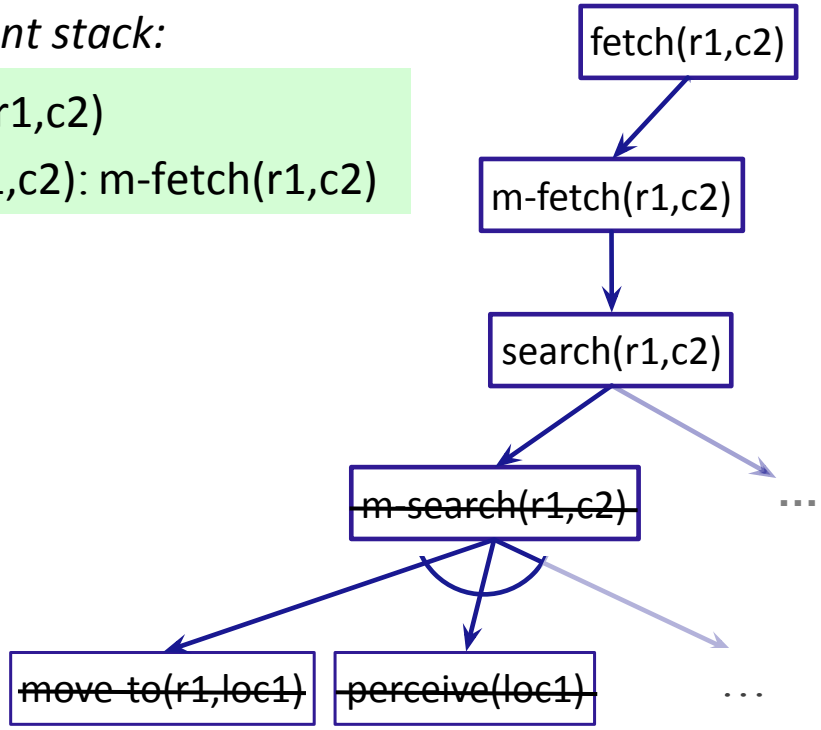
sensor failure

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
```

Refinement stack:

- search(r1,c2)
- fetch(r1,c2): m-fetch(r1,c2)



sensor failure

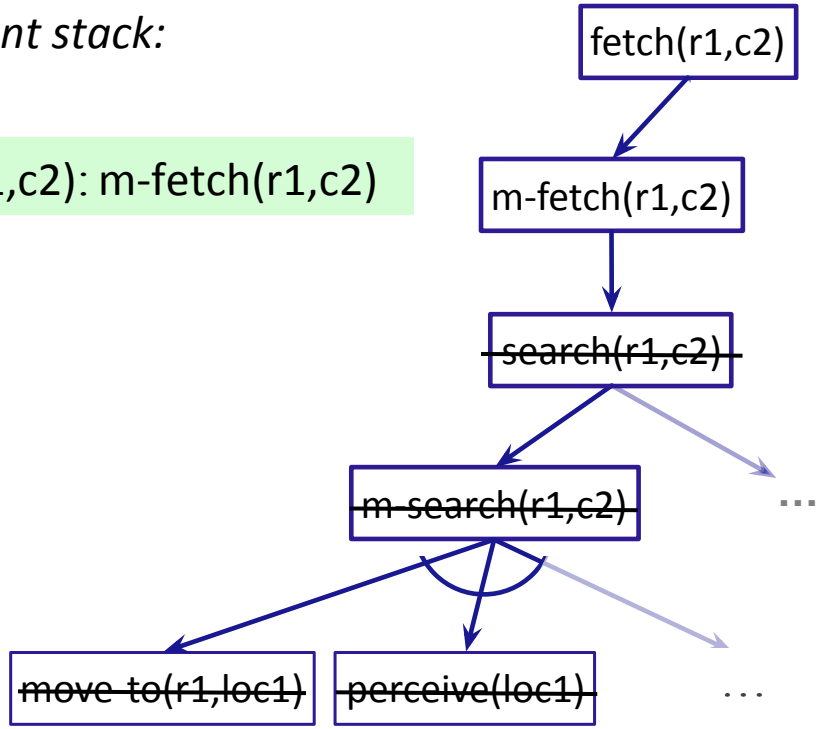
- If other candidates for search(r1,c2), try them
- Not same as backtracking
  - Different current state

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
```

Refinement stack:

- fetch(r1,c2): m-fetch(r1,c2)



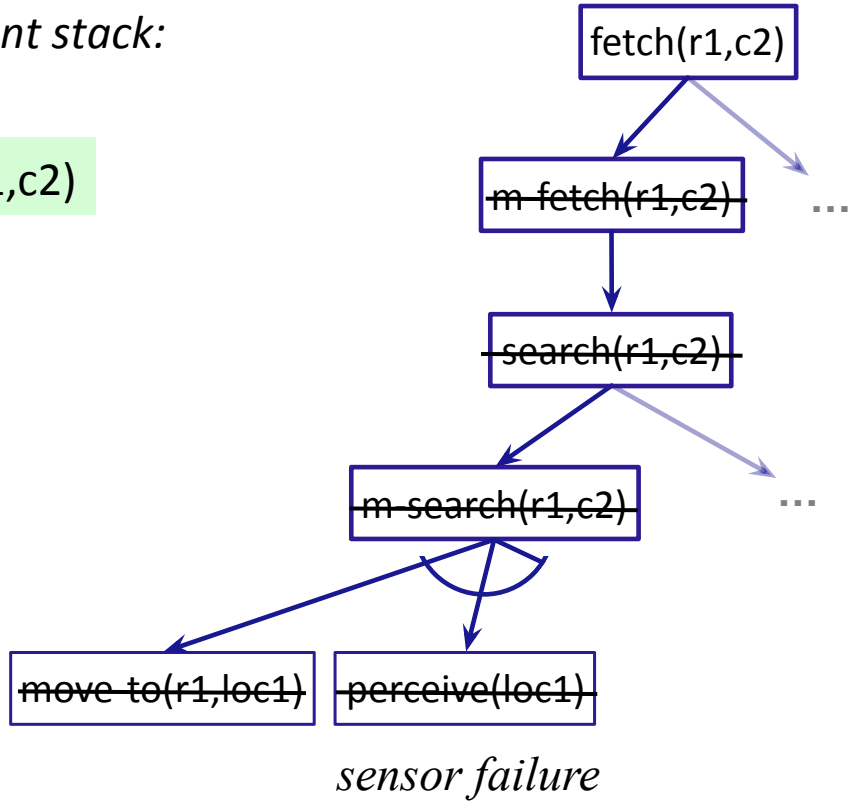
sensor failure

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
```

Refinement stack:

- fetch(r1,c2)



sensor failure

- If other candidates for fetch(r1,c2), try them
- Not same as backtracking
  - Different current state

2c. Extensions

Events

method-name(arg_1, …, arg_k)
  event: *event-identifier*
  pre: *test*
  body: *program*

- Example: an emergency
  - If you aren't already handing another emergency, then
    - stop what you're doing, go handle the emergency

m-emergency(r,l,i) l = location, i = event ID
  event: emergency(l,i)
  pre: emergency-handling(r) = F
  body: emergency-handling(r) ← T
    if load(r) ≠ nil then put(r,load(r))
    move-to(l)
    address-emergency(l,i)

Goals

method-name(arg_1, …, arg_k)
  task: achieve(*condition*)
  pre: *test*
  body: *program*

- Write goal as a special kind of task
  - achieve(*condition*)
- Like other tasks, but includes monitoring
  - if *condition* becomes true before finishing body(m), stop early
  - if *condition* isn't true after finishing body(m), fail and try another method

Extensions

- Concurrent subtasks
  - refinement stack for each one

body of a method:
…
{concurrent: $\tau_1, \tau_2, \ldots, \tau_n$}
…

- Controlling the progress of tasks
  - e.g., suspend a task for a while

$Agenda = \{stack_1, stack_2, \ldots, stack_n\}$

- If there are multiple stacks, which ones get higher priority?
  - Application-specific heuristics

- For a task τ, which candidate to try first?
  - *Refinement planning*

$Candidates = \text{Instances}(\tau, \mathcal{M}, \xi)$

Outline

1. **Representation**
   a. State variables, commands, refinement methods
   b. Example
2. **Acting**
   a. `Rae` (Refinement Acting Engine)
   b. Example
   c. Extensions
3. **Planning**
   a. Motivation and basic ideas
   b. Deterministic action models
   c. `SeRPE` (Sequential Refinement Planning Engine)
4. **Using Planning in Acting**
   a. Techniques
   b. Caveats

3a. Motivation

- When dealing with an event or task, Rae may need to make either/or choices
  - *Agenda*: tasks τ_1, τ_2, …, τ_n
    - Several tasks/events, how to prioritize?
  - Candidates for τ_1: m_1, m_2, …,
    - Several candidate methods or commands, which one to try first?
- Rae immediately executes commands
  - Bad choices may be costly
    - or irreversible

Refinement Planning

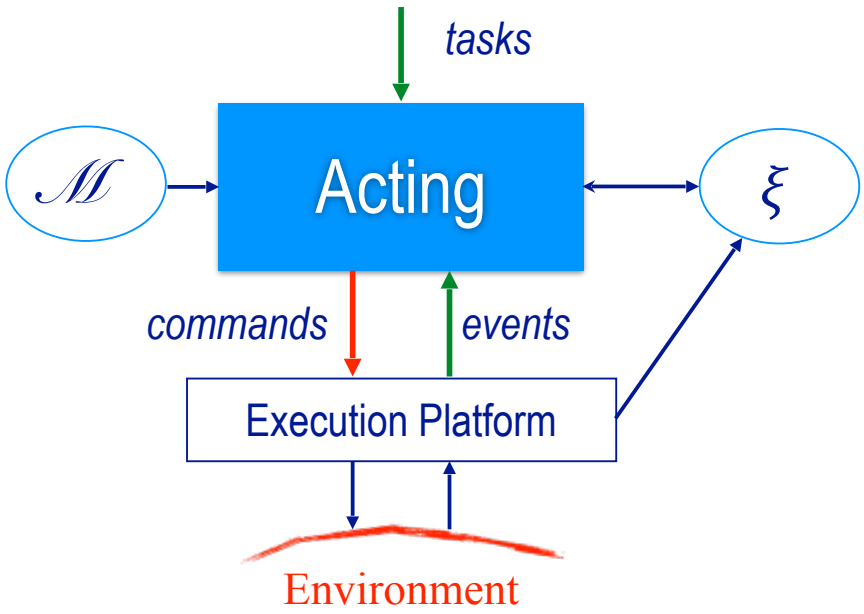
- Basic idea:
  - Go step by step through `Rae`, but don't send commands to execution platform
  - For each command, use a descriptive action model to predict the next state
    - Tells *what*, not *how*
  - Whenever we need to choose a method
    - Try various possible choices, explore consequences, choose best

- Generalization of HTN planning
  - HTN planning: body of a method is a list of tasks
  - Here: body of method is the same program `Rae` uses
  - Use it to *generate* a list of tasks

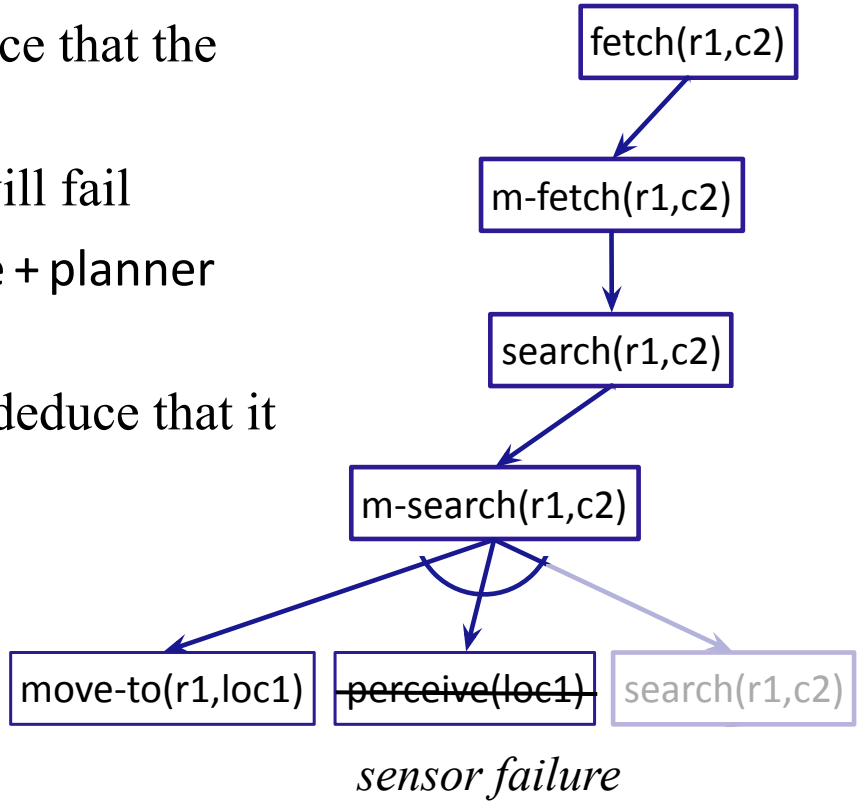
Refinement Planning

Example

- Suppose Rae + planner learns in advance that the sensor isn't available
  - Lookahead tells it that m-search will fail
  - If another method is available, Rae + planner will use it
    - Otherwise, Rae + planner will deduce that it cannot do fetch



Search tree:



sensor failure

3b. Descriptive Action Models

- Predict the outcome of performing a command
  - Preconditions-and-effects representation

- *Command*:
  - take(r,o,l):
    robot r takes object o at location l

- *Action model*

take(r,o,l)
pre: cargo(r) = nil, loc(r) = l, loc(o) = l
eff: cargo(r) ← o, loc(o) ← r

Descriptive Action Models

- Predict the outcome of performing a command
  - Preconditions-and-effects representation

- *Command*:
  - take(r,o,l):
    robot r takes object o at location l
  - put(r,o,l):
    r puts o at location l

- *Action model*

take(r,o,l)
pre: cargo(r) = nil, loc(r) = l, loc(o) = l
eff: cargo(r) ← o, loc(o) ← r

put(r,o,l)
pre: loc(r) = l, loc(o) = r
eff: cargo(r) ← nil, loc(o) ← l

Descriptive Action Models

- Predict the outcome of performing a command
  - Preconditions-and-effects representation

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

- *Action model*

take(r,o,l)
pre: cargo(r) = nil, loc(r) = l, loc(o) = l
eff: cargo(r) ← o, loc(o) ← r

put(r,o,l)
pre: loc(r) = l, loc(o) = r
eff: cargo(r) ← nil, loc(o) ← l

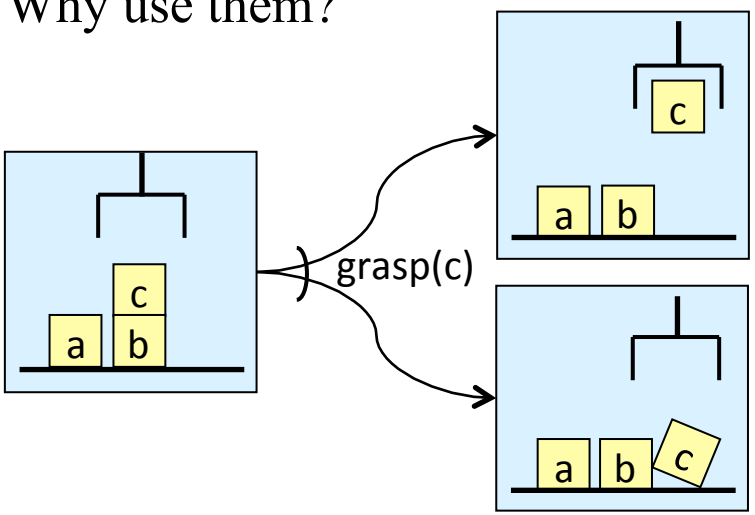
perceive(r,l):
?

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

Can't do the *fetch* example

Limitation

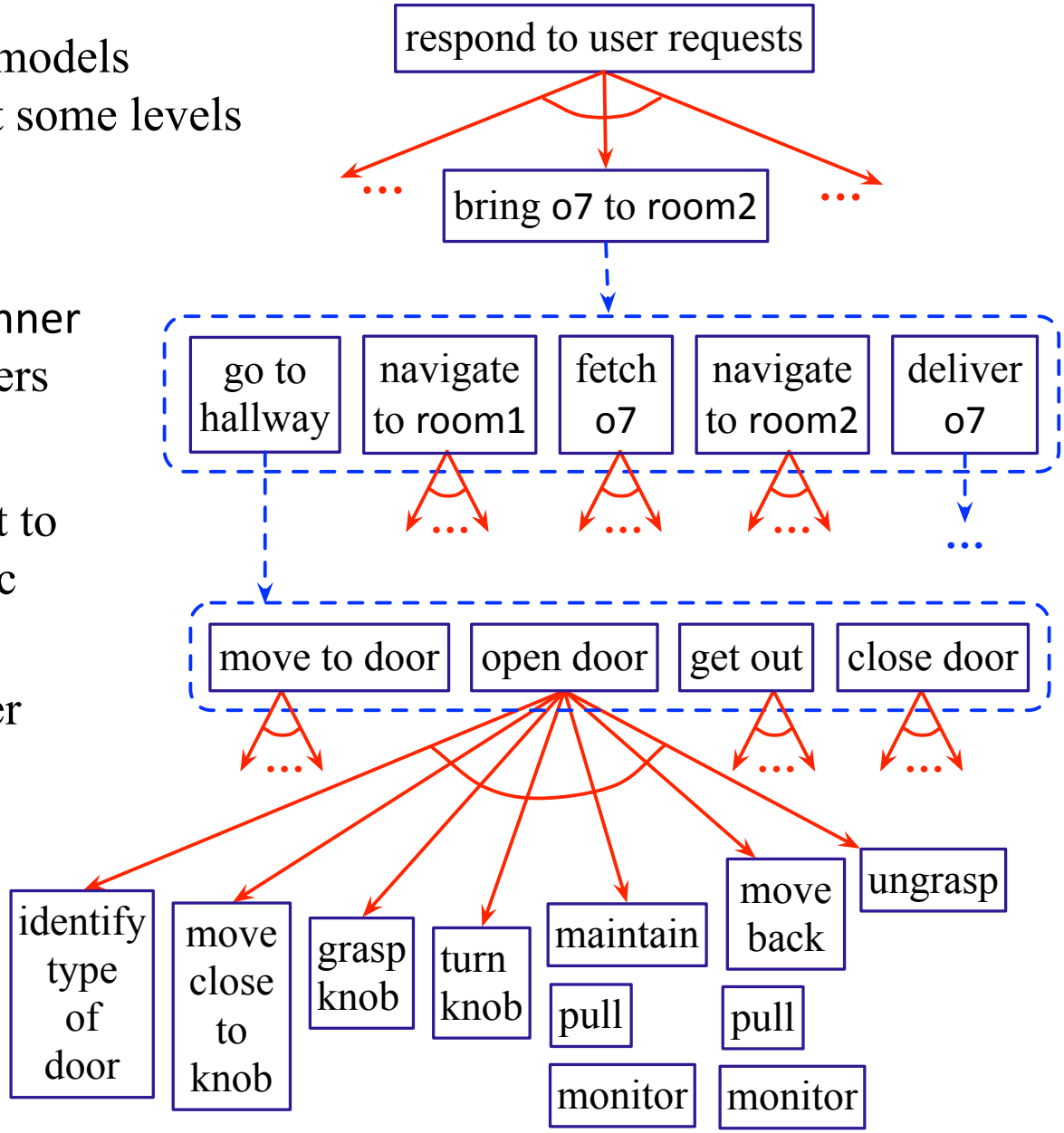
- Most environments are inherently nondeterministic
  - Deterministic action models won't always make the right prediction
- Why use them?



- Deterministic models => much simpler planning algorithms
  - Use when errors are infrequent and don't have severe consequences
  - Actor can fix the errors online

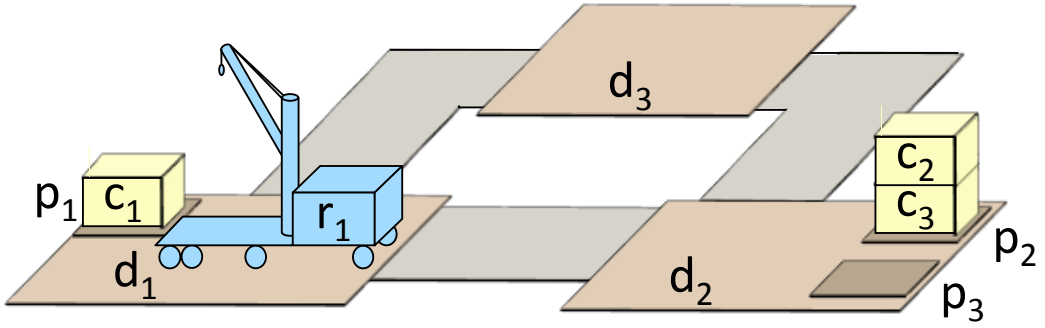
Planning/Acting at Different Levels

- Sometimes deterministic models will work more reliably at some levels than at others
- May want to use Rae+planner at some levels, Rae at others
- In other cases, might want to plan with nondeterministic outcomes
  - Paolo will discuss later



Simple Deterministic Domain

- Robot can move containers
  - 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)←c, pile(c)←nil, pos(c)←r, top(p)←c'

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

move(r,d,d')
pre: adjacent(d,d'), loc(r)=d, occupied(d')=F
eff: loc(r)=d', occupied(d)=F, occupied(d')=T

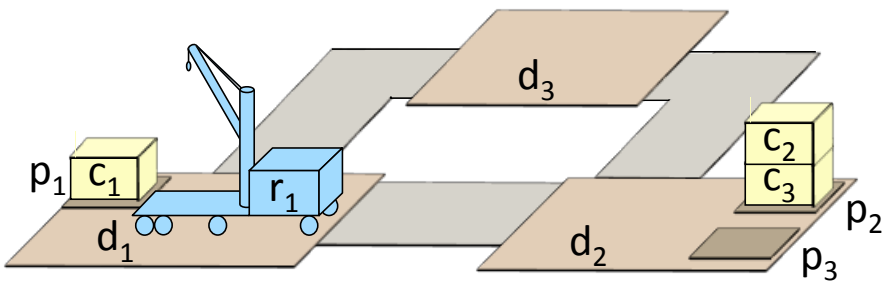
Tasks and Methods

- Task: put-in-pile(c,p') – put c into pile p' if it isn't there already

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

If c is already in p', do nothing

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) ≠ d then navigate(r,d)
uncover(c)
load(r, c, pos(c), p, d)
if loc(r) ≠ d' then navigate(r,d')
unload(r, c, top(p'), p', d)



If c isn't in p'
- find a route to c, follow it to c
- uncover c, load c onto r
- move to p', unload c

Tasks and Methods

- Task: uncover(c) – remove everything that's on c

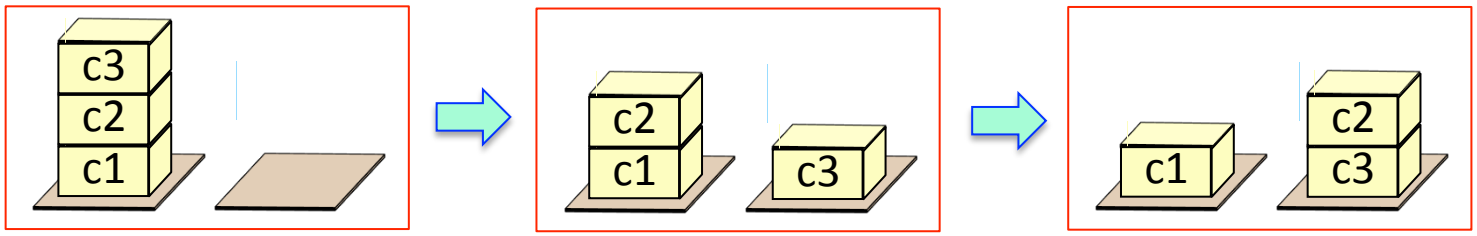
m1-uncover(c)
  task: uncover(c)
  pre: top(pile(c))=c
  body: // *empty*

If nothing is on c, do nothing

m2-uncover(r,c,c,p',d)
  task: uncover(c)
  pre: pile(c)=p ∧ top(p)≠c
    ∧ at(p,d) ∧ at(p',d) ∧ p'≠p
    ∧ loc(r)=d ∧ cargo(r)=nil
  body: while top(p) ≠ c do
    c' ← top(p)
    load(r,c',pos(c'),p,d)
    unload(r,c',top(p'),p',d)

while something is on c

- remove whatever is at the top of the stack



3c. SeRPE (Sequential Refinement Planning Engine)

SeRPE($\mathcal{M}, \mathcal{A}, s, \tau$)
  $Candidates \leftarrow$ Instances($\mathcal{M}, \tau, s$)
  if $Candidates = \varnothing$ then return failure
  nondeterministically choose $m \in Candidates$
  return Progress-to-finish($\mathcal{M}, \mathcal{A}, s, \tau, m$)

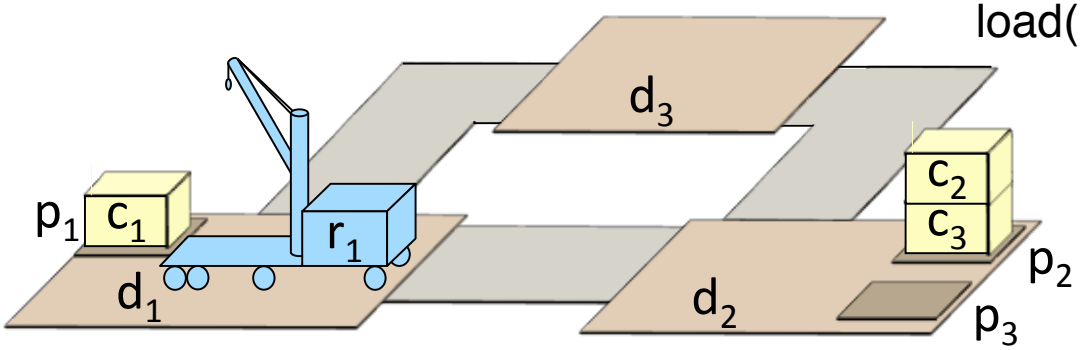
- Which candidate method for τ?
- Rae: *arbitrary choice*
  - no search, purely reactive
- SeRPE: *nondeterministic choice*
  - search among alternatives
  - many possible search strategies

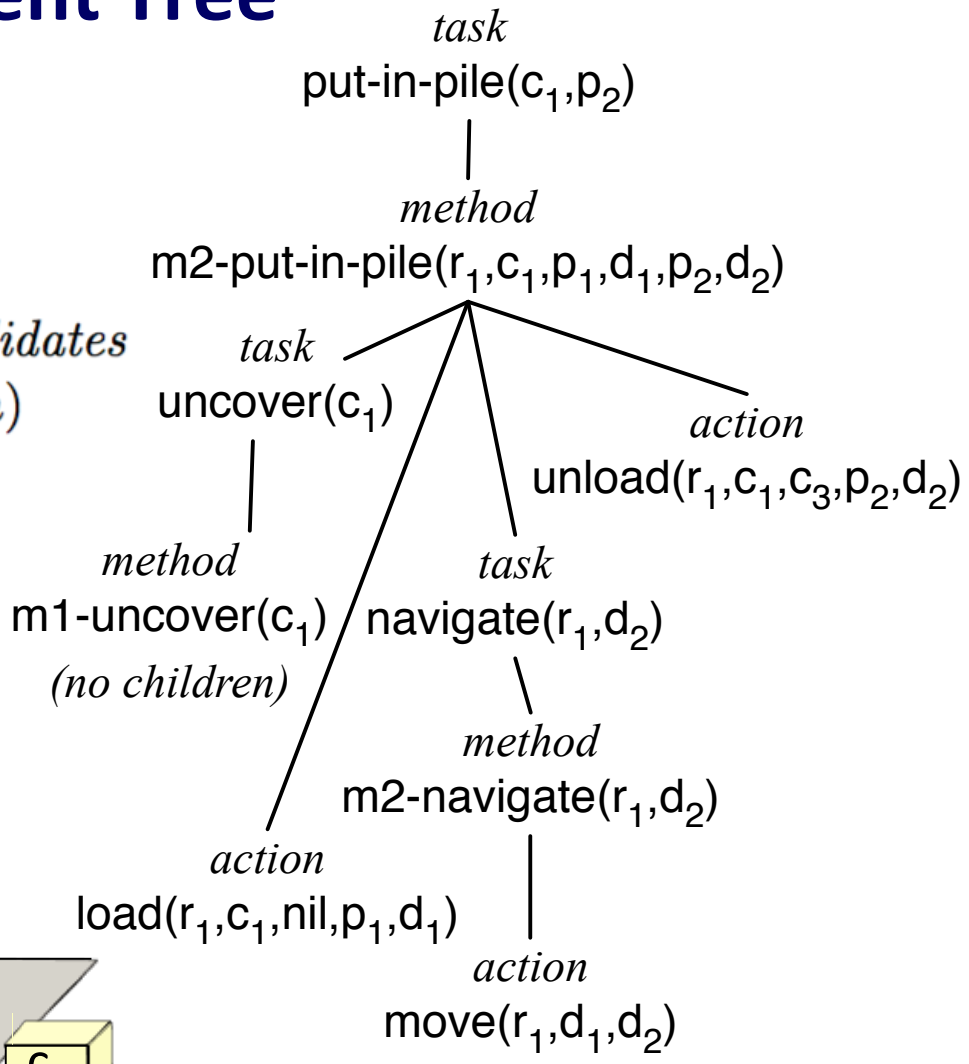
$\mathcal{M}$ = {methods}
$\mathcal{A}$ = {action models}
s = initial state
τ = task or goal

Rae($\mathcal{M}$)
  $Agenda \leftarrow \varnothing$
  loop
    until the input stream of external tasks and events is empty do
      read τ in the input stream
      $Candidates \leftarrow$ Instances($\mathcal{M}, \tau, \xi$)
      if $Candidates = \varnothing$ then output("failed to address" τ)
      else do
        arbitrarily choose $m \in Candidates$
        $Agenda \leftarrow Agenda \cup \{\langle(\tau, m, \text{nil}, \varnothing)\rangle\}$
    for each $stack \in Agenda$ do
      Progress($stack$)
      if $stack = \varnothing$ then $Agenda \leftarrow Agenda \setminus \{stack\}$

Refinement Tree

SeRPE($\mathcal{M}, \mathcal{A}, s, \tau$)
  $Candidates \leftarrow$ Instances($\mathcal{M}, \tau, s$)
  if $Candidates = \varnothing$ then return failure
  nondeterministically choose $m \in Candidates$
  return Progress-to-finish($\mathcal{M}, \mathcal{A}, s, \tau, m$)





Heuristics For SeRPE

SeRPE($\mathcal{M}, \mathcal{A}, s, \tau$)
  $Candidates \leftarrow$ Instances($\mathcal{M}, \tau, s$)
  if $Candidates = \varnothing$ then return failure
  nondeterministically choose $m \in Candidates$
  return Progress-to-finish($\mathcal{M}, \mathcal{A}, s, \tau, m$)

- *Ad hoc* approaches:
  - domain-specific estimates
  - keep statistical data on how well each method works
  - try methods (or actions) in the order that they appear in $\mathcal{M}$ (or $\mathcal{A}$)
- Ideally, would want to implement using heuristic search (e.g., GBFS)
  - What heuristic function?
  - Open problem
- SeRPE is a generalization of HTN planning
  - In some cases classical-planning heuristics can be used, in other cases they become intractable [Shivashankar *et al.*, ECAI-2016]

Outline

1. **Representation**
   a. State variables, commands, refinement methods
   b. Example
2. **Acting**
   a. `Rae` (Refinement Acting Engine)
   b. Example
   c. Extensions
3. **Planning**
   a. Motivation and basic ideas
   b. Deterministic action models
   c. `SeRPE` (Sequential Refinement Planning Engine)
4. **Using Planning in Acting**
   a. Techniques
   b. Caveats

4a. Using Planning in Acting

- Two approaches:
  - REAP (Refinement Engine for Acting and Planning)
    - RAE-like actor, uses SeRPE-like planning at all levels
    - Pseudocode is complicated
    - We'll skip it
      - (see Section 3.4 of *Automated Planning and Acting*)
  - Non-hierarchical actor with refinement planning
    - Much simpler
    - Illustrates the basic issues

Using Planning in Acting

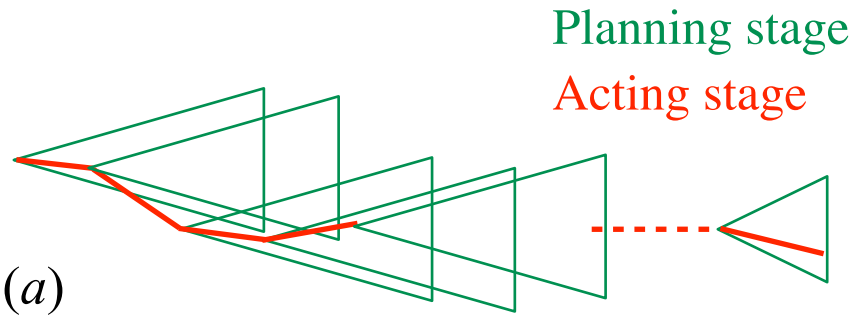
Run-Lookahead

while ($s \leftarrow$ observed state) $\not\models g$ do

$\pi \leftarrow$ Lookahead($\mathcal{M}$,$\mathcal{A}$,s,τ)

if π = failure then return failure

$a \leftarrow$ pop-first-action(π); perform(a)



- Lookahead: modified version of SeRPE (discuss later)
  - Searches part of the search space, returns a partial plan

- Useful when unpredictable things are likely to happen
  - Always replans immediately
- Potential problem:
  - May pause repeatedly while waiting for Lookahead to return
  - What if s changes during the wait?

Using Planning in Acting

Run-Lazy-Lookahead

$s \leftarrow$ observed state

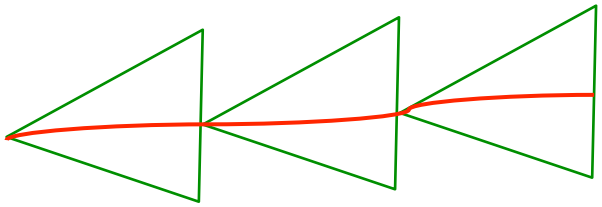
while $s \not\models g$ do

$\pi \leftarrow$ Lookahead($\mathcal{M}$,$\mathcal{A}$,s,τ)

if π = failure then return failure

while $\pi \neq \langle\,\rangle$ and $s \not\models g$ and Simulate(s,g,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 does a simulation of the plan
  - Can be more detailed than SeRPE's action models
    - e.g., 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

$\pi \leftarrow \langle\,\rangle$; $s \leftarrow$ observed state

thread 1:

loop

$\pi \leftarrow$ Lookahead($\mathcal{M}$,$\mathcal{A}$,s,τ)

thread 2:

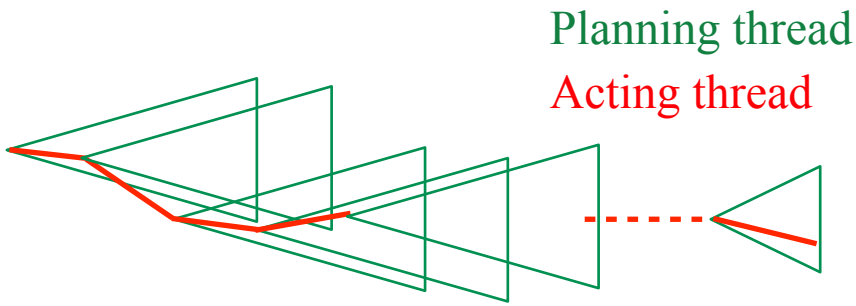
loop

if $s \vDash g$ then return success

else if π = failure then return failure

else if $\pi \neq \langle\,\rangle$ and Simulate(s,g,π) $\neq$ failure do

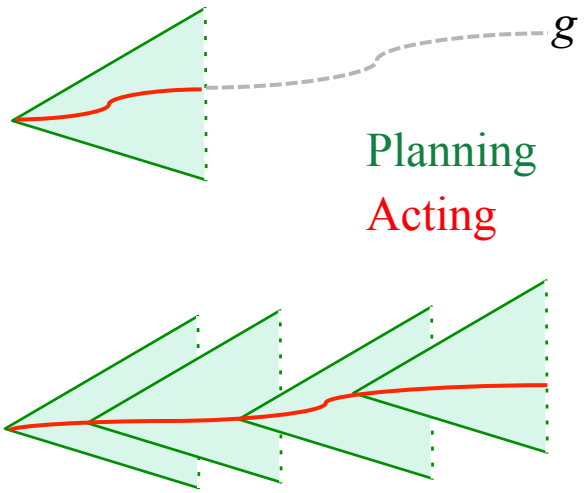
$a \leftarrow$ pop-first-action(π); perform(a); $s \leftarrow$ observed state



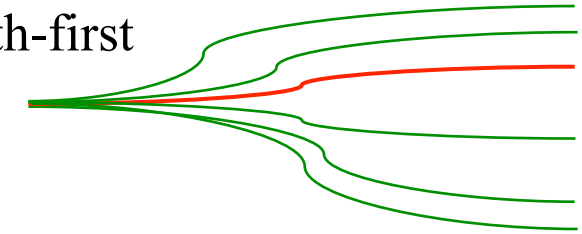
- Objective:
  - Balance tradeoffs between Run-Lookahead and Run-Lazy-Lookahead
  - More up-to-date plans than Run-Lazy-Lookahead, but without waiting for Lookahead to return

How to do Lookahead

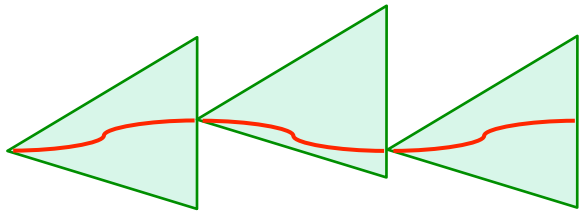
- *Receding horizon*
  - Cut off search before reaching g
    - e.g., if plan's length exceeds $l_{\max}$
    - or if plan's cost exceeds $c_{\max}$
    - or when we're running out of time
  - Horizon "recedes" on the actor's successive calls to the planner



- *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 it's finished with g_i, actor calls planner on next subgoal g_{i+1}



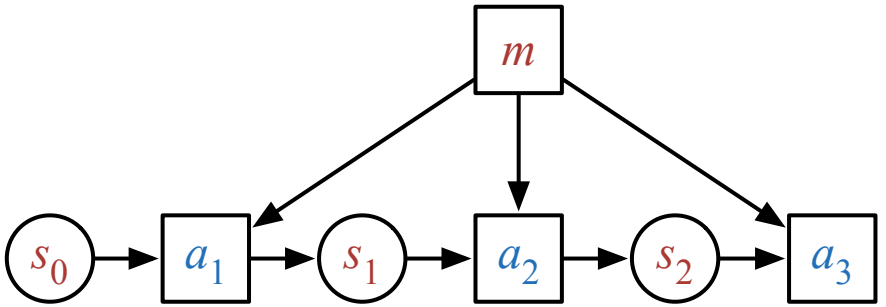
- Can use combinations of these

Example

- **Killzone 2**
  - video game
- SeRPE-like planner
  - Domain-specific
  - 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

4b. Caveats

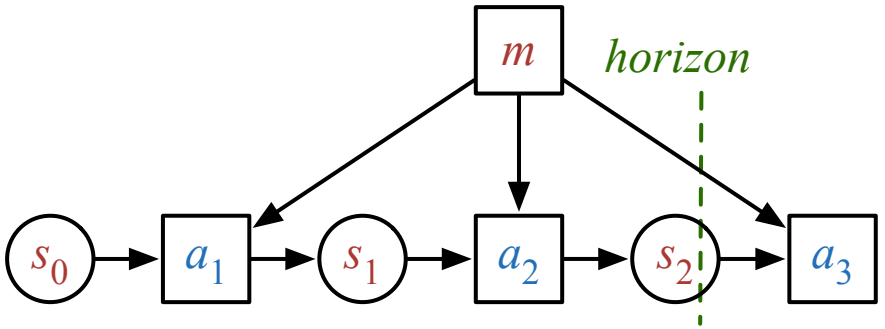
- Start in state s_0, want to accomplish task τ
  - Refinement method m:
    - task: τ
    - pre: s_0
    - body: a_1, a_2, a_3



- Actor uses Run-Lookahead
  - Lookahead = SeRPE, returns $\langle a_1, a_2, a_3 \rangle$
  - Actor performs a_1, calls Lookahead again
  - No applicable method for τ in s_1, SeRPE returns failure
- Fixes
  - When writing refinement methods, make them general enough to work in different states
  - In some cases Lookahead might be able to fall back on classical planning until it finds something that matches a method
  - Keep snapshot of SeRPE's search tree at s_1, resume next time it's called

Caveats

- Start in state s_0, want to accomplish task τ
  - Refinement method m:
    - task: τ
    - pre: s_0
    - body: a_1, a_2, a_3

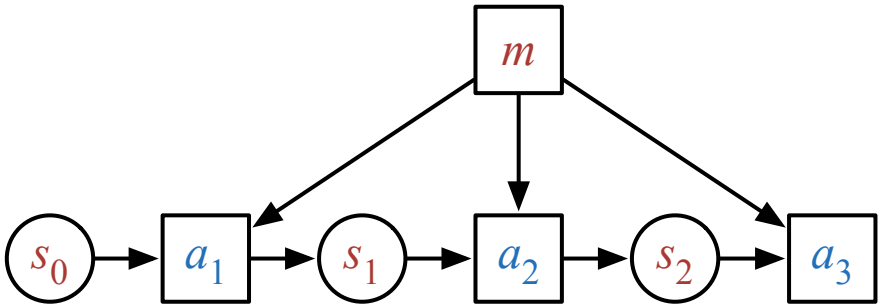


- Actor uses Run-Lazy-Lookahead
  - Lookahead = SeRPE with receding horizon, returns $\langle a_1, a_2 \rangle$
  - Actor performs them, calls Lookahead again
  - No applicable method for τ in s_2, SeRPE returns failure

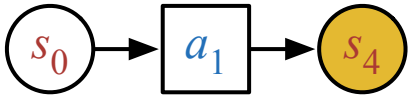
- Can use the same fixes on previous slide, with one modification
  - Keep snapshot of SeRPE's search tree at horizon

Caveats

- Start in state s_0, want to accomplish task τ
  - Refinement method m:
    - task: τ
    - pre: s_0
    - body: a_1, a_2, a_3



- Actor uses Run-Lazy-Lookahead
  - Lookahead = SeRPE, returns $\langle a_1, a_2, a_3 \rangle$
  - While acting, unexpected event
  - Actor calls Lookahead again
  - No applicable method for τ in s_4, SeRPE returns failure



- Can use most of the fixes on last two slides, with this modification:
  - Keep snapshot of SeRPE's search tree after each action
    - Restart it immediately after a_1, using s_4 as current state
- Also: make *recovery methods* for unexpected states
    - e.g., fix flat tire, get back on the road

Summary

- Representation:
  - state variables, commands/actions, refinement methods
- Refinement Acting Engine (RAE)
  - Purely reactive
    - For each task, event, or goal, select a method and apply it
- Refinement planning (SeRPE)
  - Simulate RAE's operation on a single task/event/goal
  - Deterministic actions
    - OK if we're confident of outcome, can recover if things go wrong
- Acting and planning
  - Lookahead: search part of the search space, return a partial solution
    - Several techniques for doing that
  - Caveats
    - Current state may not be what we expect
    - Possible ways to handle that

Deliberation with Refinement Methods

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

IJCAI 2016 Tutorial
New York, July 11th,2016