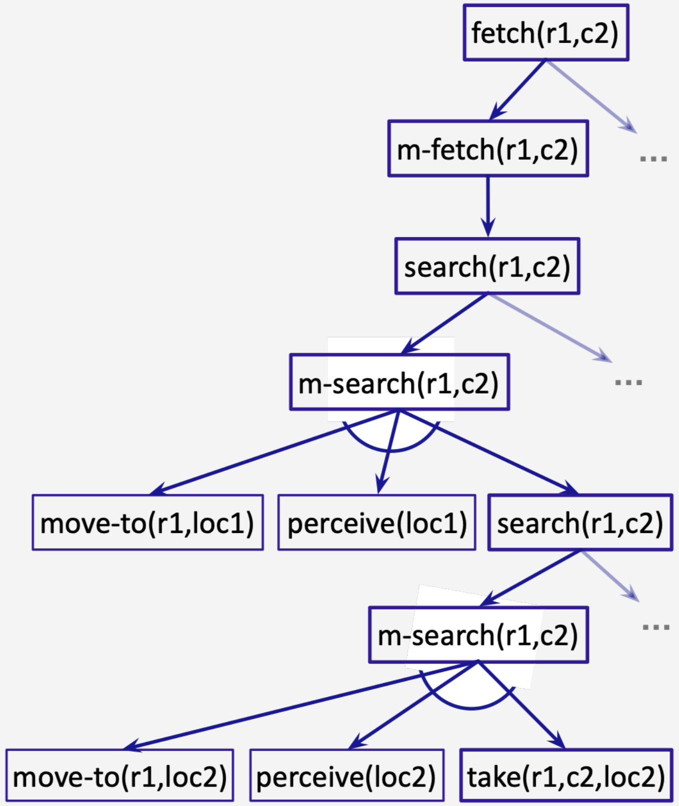
Universität Münster

Automated Planning and Acting

Refinement Methods



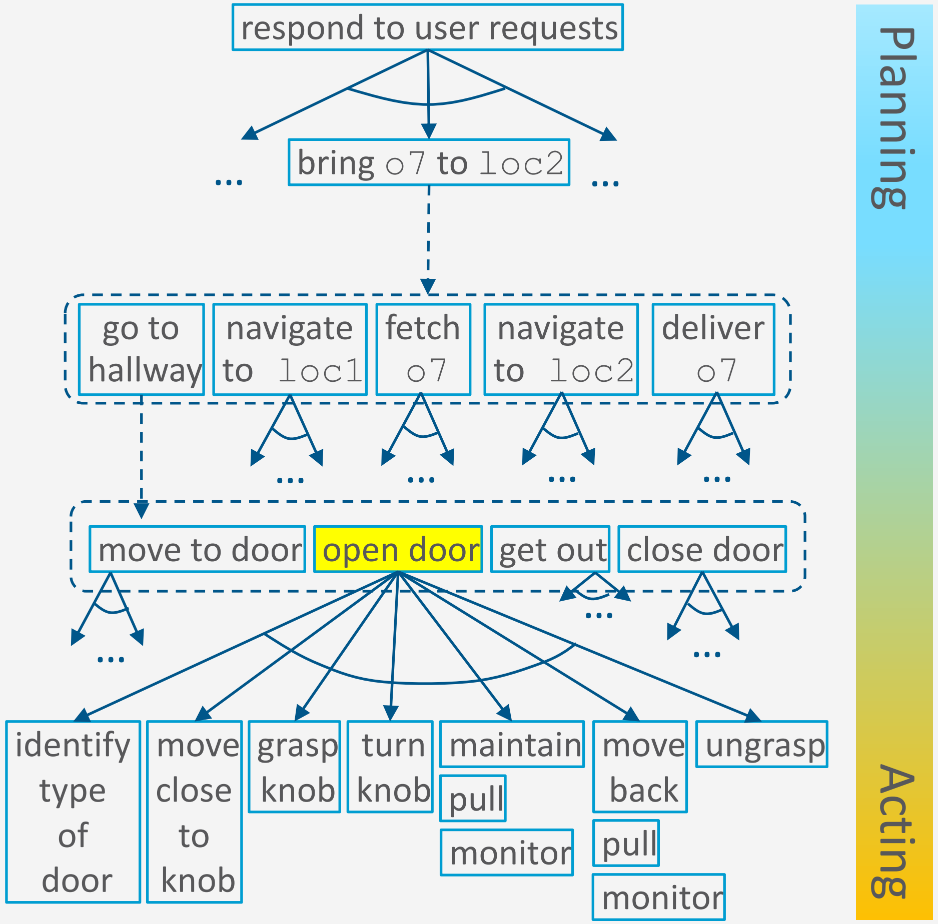
living.knowledge

Tanya Braun
Research Group Data Science, Computer Science Department

Content: Planning and Acting

Motivation & Assumptions

- Hierarchically organised deliberation
  - At high levels, abstract actions
  - At lower levels, more detail
- Refine abstract actions into ways of carrying them out
  - How?
- Remove / weaken assumptions from classical planning
  - Characteristics
    - Dynamic environment
    - Imperfect information
    - Overlapping actions
    - Nondeterminism
    - Hierarchy
    - Discrete and continuous variables



Outline per the Book

3.1 *Representation*

- State variables, commands, refinement methods
- Example

3.2 *Acting*

- RAE (Refinement Acting Engine)
- Example
- Extensions

3.3 *Planning*

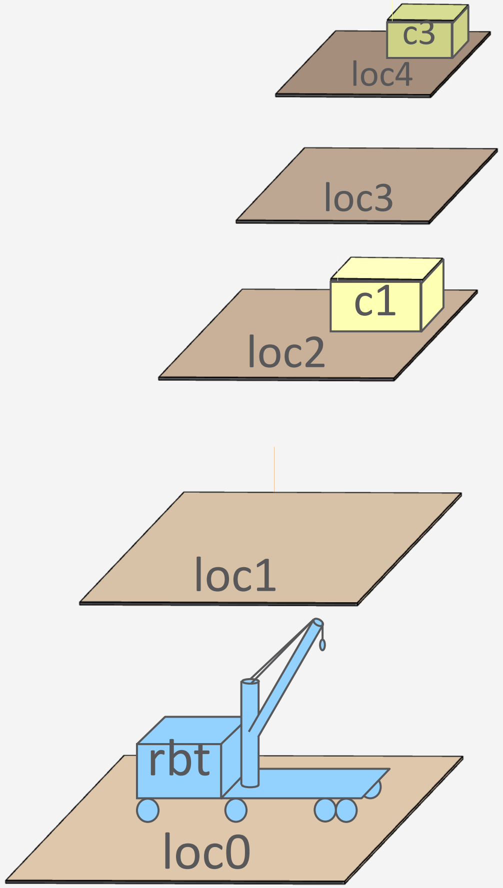
- Motivation and basic ideas
- Deterministic action models
- SeRPE (Sequential Refinement Planning Engine)

3.4 *Using Planning in Acting*

- Techniques
- Caveats

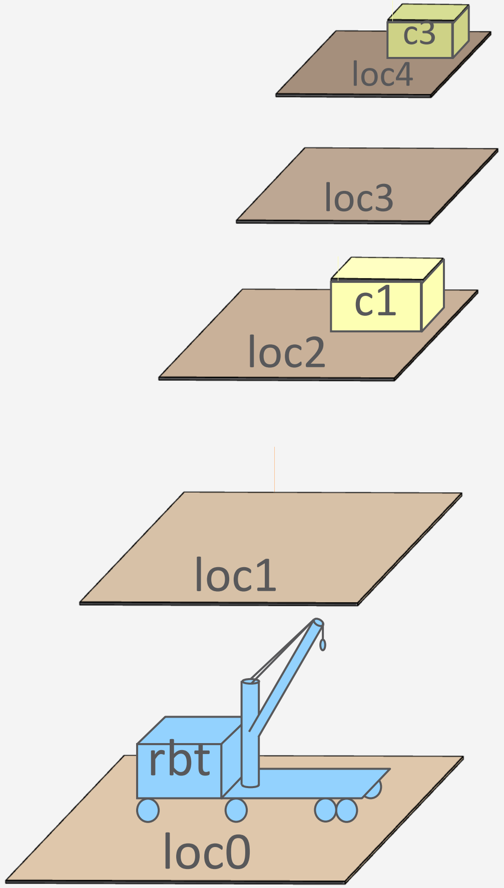
State-variable Representation (Recap)

- Objects:
  - $Robots = \{rbt\}$
  - $Containers = \{c1, c2, c3, \ldots\}$
  - $Locations = \{loc0, loc1, loc2, \ldots\}$
- State variables: syntactic terms to which we can assign values
  - $loc(r) \in Locations$
  - $load(r) \in Containers \cup \{nil\}$
  - $pos(c) \in Locations \cup Robots \cup \{unknown\}$
  - $view(r, l) \in \{T, F\}$
    - whether robot r has looked at location l
    - r can only see what is at its current location
- State: assign a value to each state variable
  - $\{loc(rbt) = loc0, pos(c1) = loc2,\ pos(c3) = loc4, pos(c2) =\ unknown, \ldots\}$



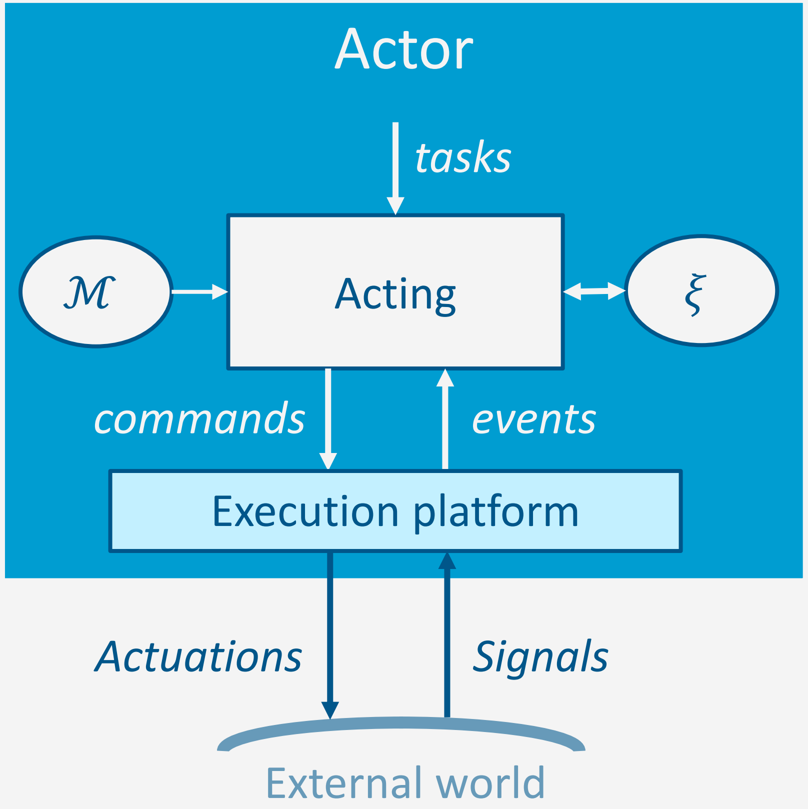
State-variable Representation: Extensions

- Range $\mathrm{ran}(x)$
  - Can be finite, infinite, continuous, discontinuous, vectors, matrices, other data structures
- Assignment statement $x \leftarrow expr$
  - Expression $expr$ returns a ground value in $\mathrm{ran}(x)$ and has no side-effects on the current state
- Tests (e.g., preconditions)
  - Simple: $x = v, x \neq v, x > v, x < v$
  - Compound: conjunction, disjunction, or negation of simple tests



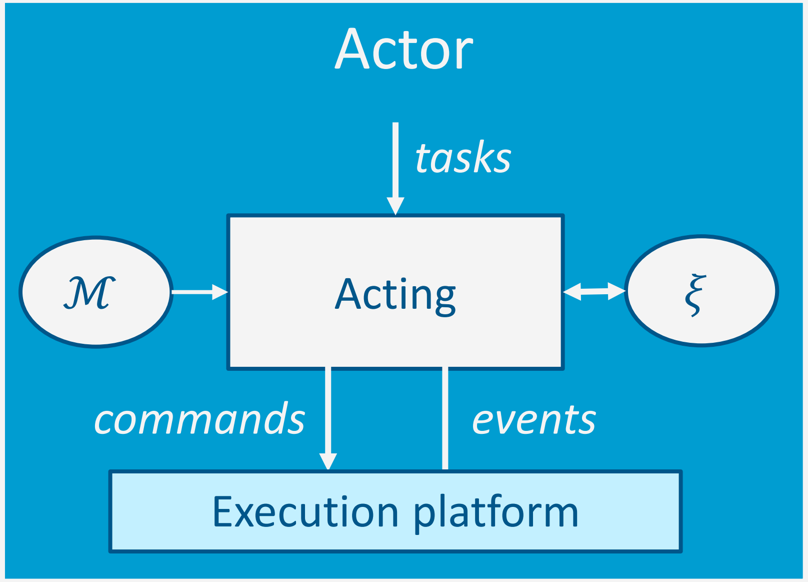
Commands

- 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 is at its current location
- Event: occurrence detected by execution platform
  - $event\text{—}name(args)$
  - Exogenous changes in the environment to which the actor may have to react
  - E.g., emergency signal, arrival of transportation vehicle
- For later: $\mathcal{M}$: library of methods, ξ: current state (abstraction)



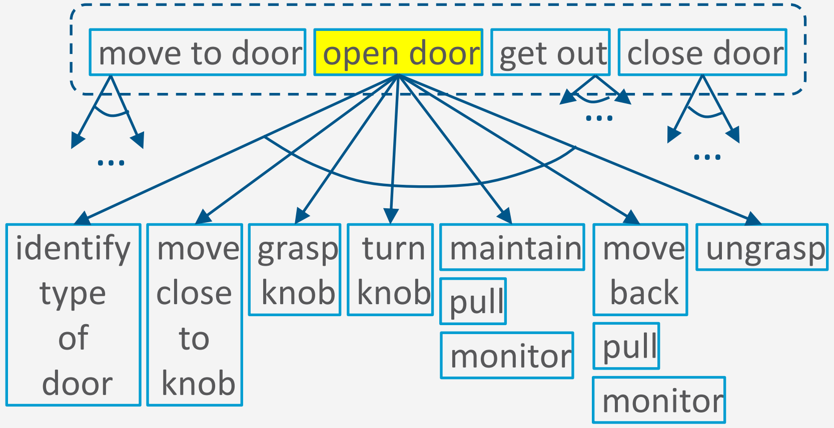
Tasks and Methods

- Task: an activity for the actor to perform
  - Could be an abstract action of a plan
- For each task, a set of refinement *methods*
  - Operational models:
    - Tell *how* to perform the task
    - Do not predict *what* it will do



$method\text{—}name(arg_1, \ldots, arg_k)$
task: *task-identifier*
pre: *test*
body: *a program*

- assignment statements
- control constructs: if-then-else, while, …
- tasks (can extend to include events, goals)
- commands to the execution platform



Example: “open door” task

- What kind:
  - Hinged on left
  - Opens toward us
  - Lever handle

→ Refinement method

```
m-opendoor(r,d,l,h)
    task:   opendoor(r,d)
    pre:    loc(r) = l ∧ adj(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)
```

Intermediate Summary

- 3.1 Operational models
  - Tasks, events
  - Commands to the execution platform
  - Extensions to state-variable representation
  - Refinement method: name, task/event, preconditions, body
  - Example: opening a door

Outline per the Book

3.1 *Representation*
- State variables, commands, refinement methods
- Example

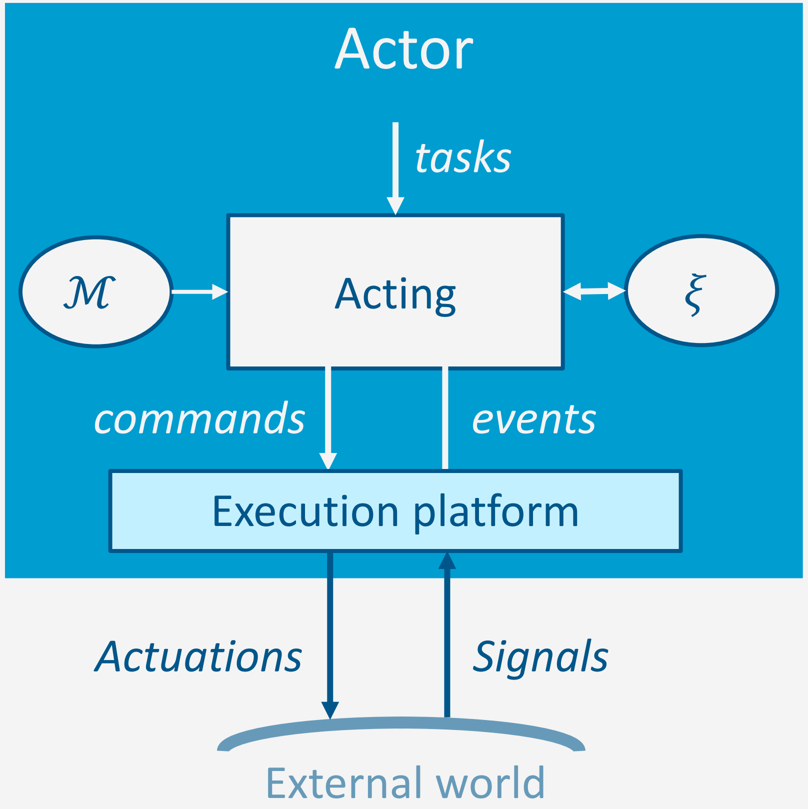
3.2 *Acting*
- RAE (Refinement Acting Engine)
- Example
- Extensions

3.3 *Planning*
- Motivation and basic ideas
- Deterministic action models
- SeRPE (Sequential Refinement Planning Engine)

3.4 *Using Planning in Acting*
- Techniques
- Caveats

RAE (Refinement Acting Engine)

- Based on OpenPRS programming language
  - Open-source robotics software, deployed in many applications
- Input
  - External tasks, events, current state ξ, library of methods $\mathcal{M}$
- Output
  - Commands to execution platform
- Perform multiple 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}



RAE (Refinement Acting Engine)

- Input
  - Library of methods $\mathcal{M}$
  - External tasks and events from an input stream, current state ξ
- Basic idea

  loop:

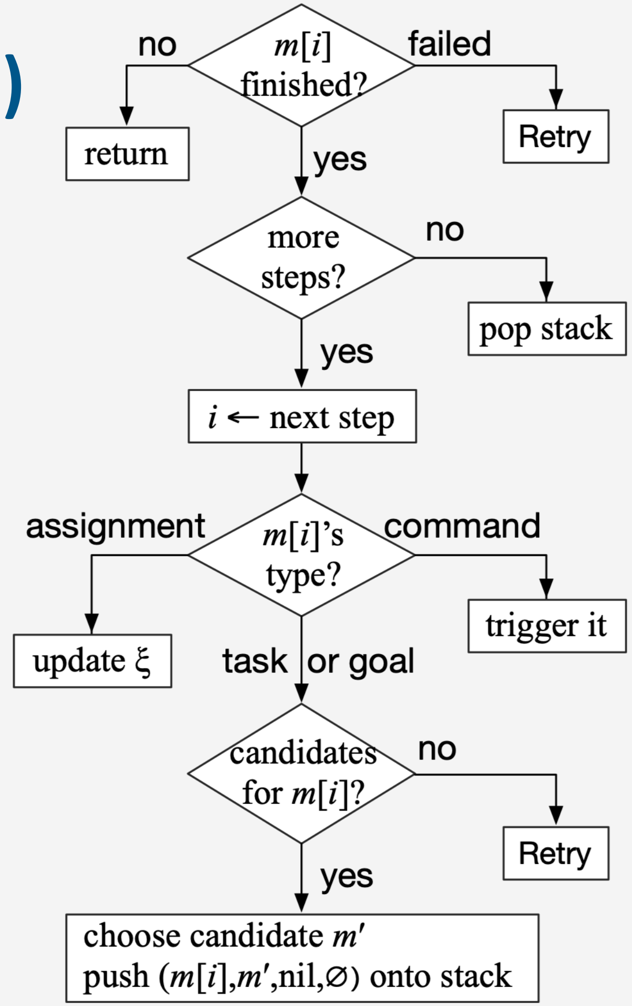
  - **if** new external tasks/events **then**
    - Add them to Agenda
  - **for each** stack in Agenda
    - Progress it
    - Remove it if it is finished

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

Stack element(τ,m,i,$tried$)

τ	task
m	instance of a method in $\mathcal{M}$
i	instruction pointer to step in body of m
$tried$	method instances already tried

Progress (subroutine)



Just a decision tree

```
Progress(stack)
    $(\tau, m, i, tried) \leftarrow$ top(stack)
    if $i \neq$ nil and $m[i]$ is a command then
        case status($m[i]$)
            running: return
            failure: Retry(stack); return
            done: continue
    if $i$ is the last step of $m$ then
        pop(stack)
    else do
        $i \leftarrow$ nextstep($m, i$)
        case type($m[i]$)
            assignment: update $\xi$ according
                              to $m[i]$; return
            command: trigger $m[i]$; return
            task or goal: continue
        $\tau' \leftarrow m[i]$
        Candidates $\leftarrow$ Instances($\mathcal{M}, \tau', \xi$)
        if Candidates $= \emptyset$ then
            Retry(stack)
        else do
            arbitrarily choose $m' \in$ Candidates
            push(($\tau', m'$, nil, $\emptyset$), stack)
```

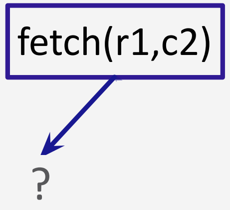
Example

```
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
```

τ: fetch(r1,c2)
m: ?
i: (see method)
tried:∅

Refinement stack



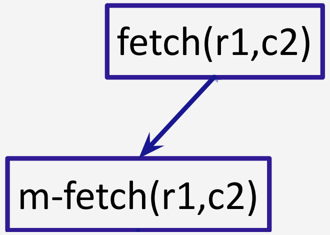
Example

```
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
```

τ: fetch(r1,c2)
m: **m-fetch(r1,c2)**
i: (see method)
tried: ∅

Refinement stack



Example

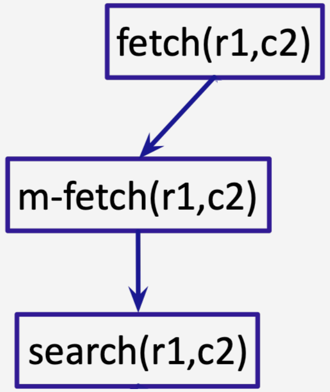
```
m-fetch(r,c)
    task:  fetch(r,c)
    pre:
    body:
        if pos(c) = unknown then
            search(r,c)
        else if loc(r) = pos(c) then
            take(r,c,pos(c))
        else do
            move-to(r,pos(c))
            take(r,c,pos(c))
```

```
m-search(r,c)
    task:   search(r,c)
    pre:    pos(c) = unknown
    body:
        if $\exists l$ (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
```

τ: **search(r1,c2)**
m: ?
i: (see method)
tried:$\emptyset$

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:$\emptyset$

Refinement stack



Example

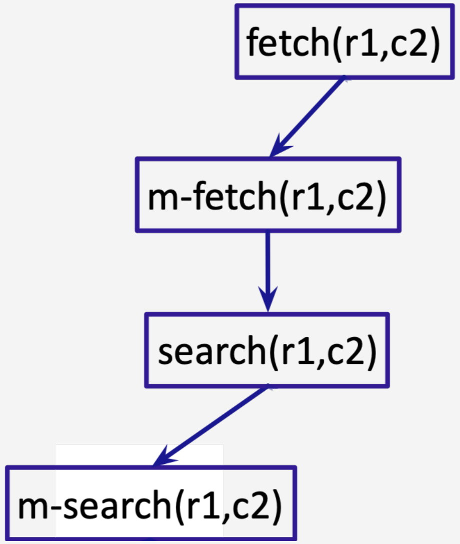
```
m-fetch(r,c)
    task:  fetch(r,c)
    pre:
    body:
        if pos(c) = unknown then
            search(r,c)
        else if loc(r) = pos(c) then
            take(r,c,pos(c))
        else do
            move-to(r,pos(c))
            take(r,c,pos(c))
```

```
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
```

τ: search(r1,c2)
m: **m-search(r1,c2)**
i: (see method)
tried:∅

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:∅

Refinement stack


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

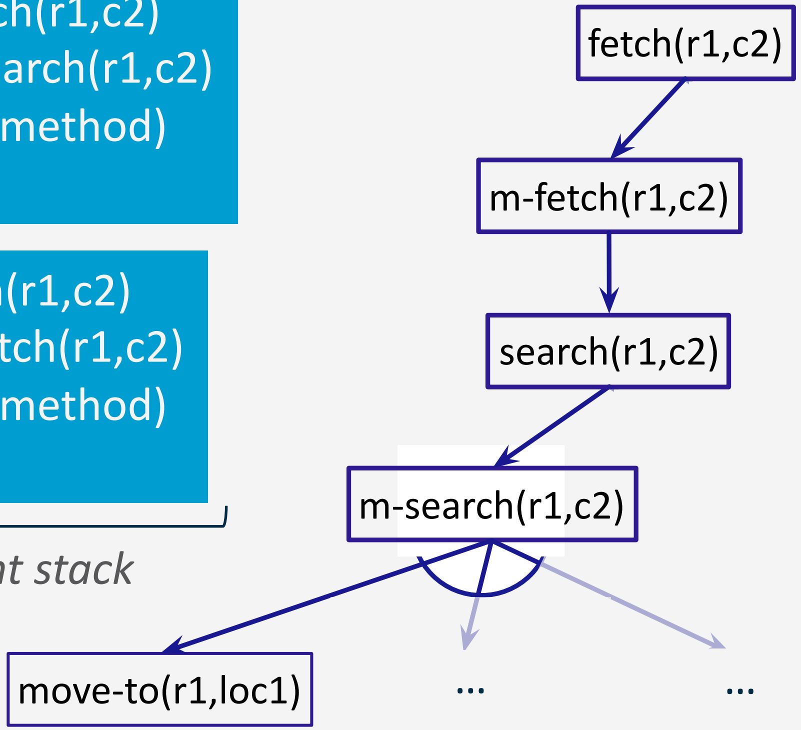
```
m-search(r,c)
    task:   search(r,c)
    pre:    pos(c) = unknown
    body:
        if ∃l (view(r,l) = F) then
            move-to(r,l)
            perceive(l)
            if pos(c) = l then
                take(r,c,l)
            else search(r,c)
        else fail
```

...

τ: search(r1,c2)
m: m-search(r1,c2)
i: (see method)
tried:∅

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:∅

Refinement stack



fetch(r1,c2)
m-fetch(r1,c2)
search(r1,c2)
m-search(r1,c2)
move-to(r1,loc1)
...
...

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))
```

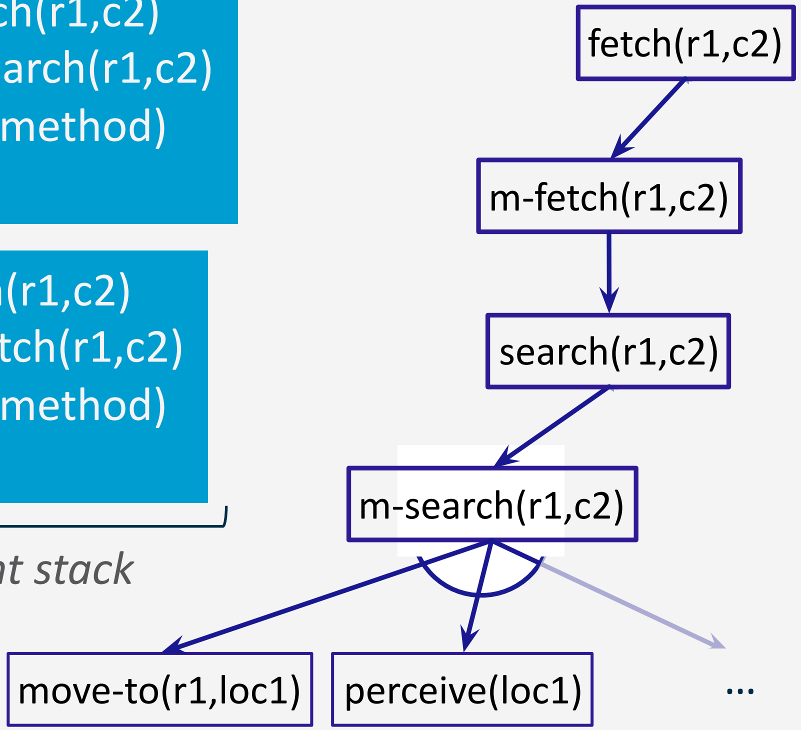
```
m-search(r,c)
    task:   search(r,c)
    pre:    pos(c) = unknown
    body:
        if ∃l (view(r,l) = F) then
            move-to(r,l)
            perceive(l)
            if pos(c) = l then
                take(r,c,l)
            else search(r,c)
        else fail
```

…

τ: search(r1,c2)
m: m-search(r1,c2)
i: (see method)
tried:∅

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:∅

Refinement stack



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
```

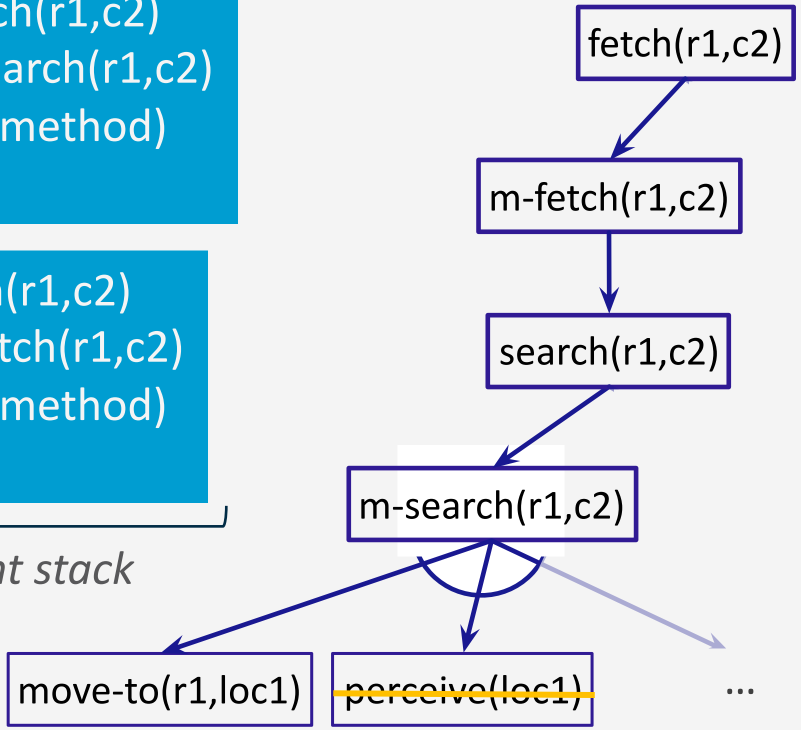
sensor failure

…

τ: search(r1,c2)
m: m-search(r1,c2)
i: (see method)
tried:∅

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:∅

Refinement stack



Example

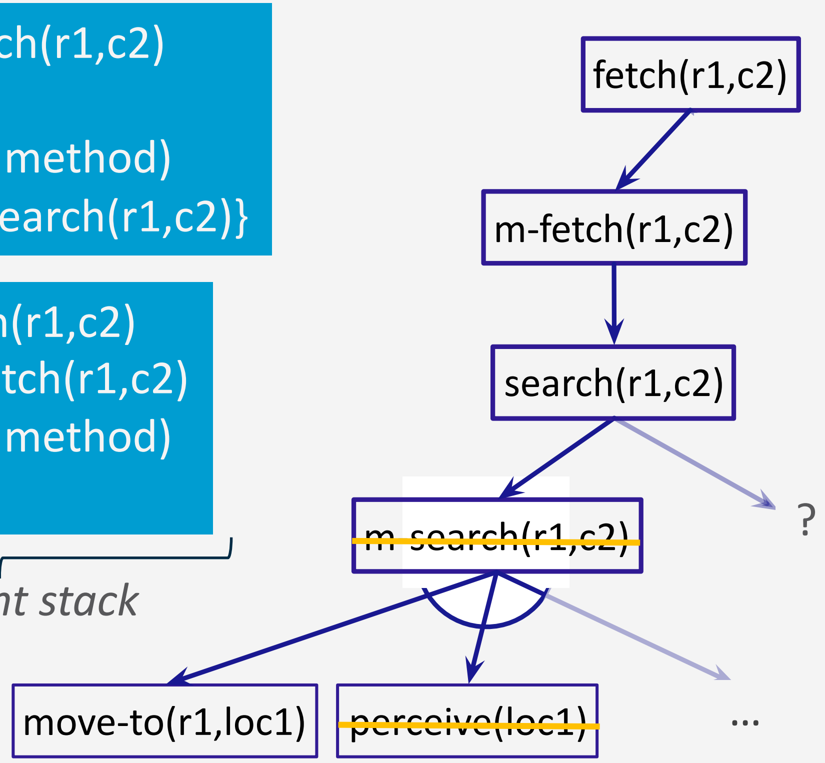
```
m-fetch(r,c)
    task:  fetch(r,c)
    pre:
    body:
        if pos(c) = unknown then
            search(r,c)
        else if loc(r) = pos(c) then
            take(r,c,pos(c))
        else do
            move-to(r,pos(c))
            take(r,c,pos(c))
```

```
~~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
```

τ: search(r1,c2)
m: ?
i: (see method)
tried:{m-search(r1,c2)}

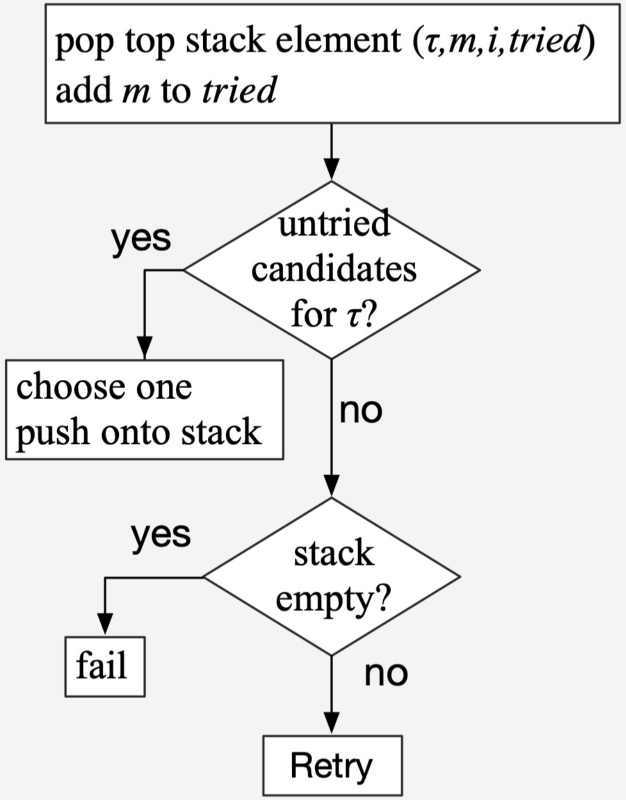
τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:∅

Refinement stack



If other candidates for *search(r1,c2)*, try them.

Retry (subroutine)



pop top stack element (τ,*m*,*i*,*tried*)
add *m* to *tried*

untried candidates for τ?

yes

choose one
push onto stack

no

stack empty?

yes

fail

no

Retry

Another decision tree

```
Retry(stack)
    $(\tau, m, i, tried) \leftarrow$ pop(stack)
    $tried \leftarrow tried \cup \{m\}$
    $Candidates \leftarrow$ Instances$(\mathcal{M}, \tau, \xi) \setminus tried$
    if $Candidates \neq \emptyset$ then
        arbitrarily choose $m' \in Candidates$
        push($(\tau, m', $nil$, \emptyset)$, stack)
    else do
        if $stack \neq \emptyset$ then
            Retry(stack)
        else do
            output("failed to accomplish" $\tau$)
            $Agenda \leftarrow Agenda \setminus stack$
```

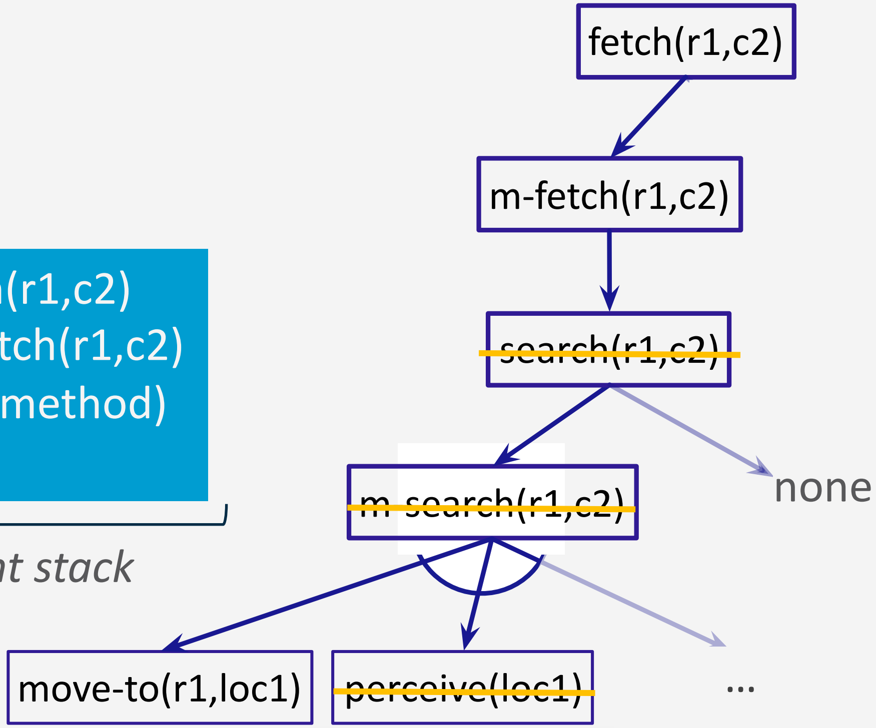
Example

```
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
```

τ: fetch(r1,c2)
m: m-fetch(r1,c2)
i: (see method)
tried:∅

Refinement stack



sensor failure

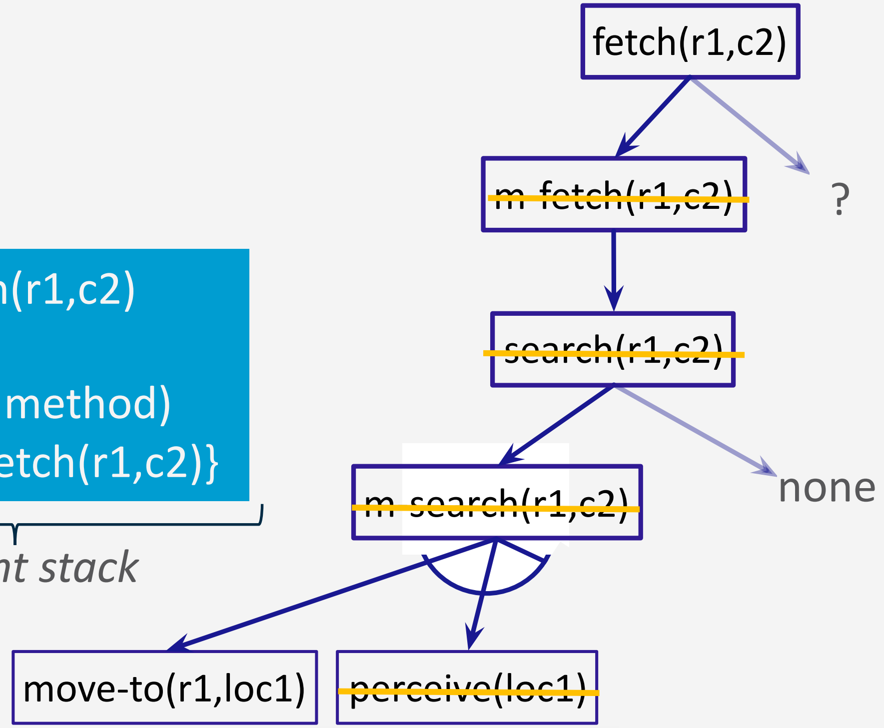
Example

```
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
```

τ: fetch(r1,c2)
m: ?
i: (see method)
tried:{m-fetch(r1,c2)}

Refinement stack



fetch(r1,c2)

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

?

~~search(r1,c2)~~

none

~~m-search(r1,c2)~~

move-to(r1,loc1)

~~perceive(loc1)~~

sensor failure

If other candidates for *fetch(r1,c2)*, try them.

Extensions to RAE: Events

- Example: an emergency
  - If r is not already handling another emergency, then
    - Stop what it is doing
    - Go handle the emergency

```
m-emergency(r,l,i)
    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)
```

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

$method\text{—}name(arg_1, \ldots, arg_k)$
task: *event-identifier*
pre: *test*
body: *a program*

Extensions to RAE: Goals

- Write as a special task
  - Like others, but includes monitoring: modify Progress
    - if *condition* becomes true before finishing body(*m*), stop early
    - if *condition* isn't true after finishing body(*m*), fail and try another method

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

m-fetch(*r*,*c*)
    task: fetch(*r*,*c*)
    pre:
    body: ~~if pos(*c*) = unknown then~~
        ~~search(*r*,*c*)~~
        achieve(pos(*c*)≠unknown)
        move-to(*r*,pos(*c*))
        take(*r*,*c*,pos(*c*))

m-find-where(*r*,*c*)
    task: achieve(pos(*c*)≠unknown)
    pre:
    body: while exists loc. *l* s.t. view(*l*) = F do
        move-to(*l*)
        perceive(*l*)

$method\text{—}name(arg_1, \ldots, arg_k)$
    task: achieve(*condition*)
    pre: *test*
    body: *a program*

Other Extensions to RAE

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

$\mathit{Agenda} = \{\mathit{stack}_1, \mathit{stack}_2, \ldots, \mathit{stack}_n\}$

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

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

Intermediate Summary

- 3.2 Refinement Acting Engine (RAE)
  - Purely reactive: select a method and apply it
  - RAE: input stream, *Candidates*, Instances, *Agenda*, refinement stacks
  - Progress: command status, nextstep, type of step
  - Retry: *Candidates* \ *tried, Agenda* \ *stack*
  - Refinement trees
  - Concurrent tasks: for each, a refinement stack
  - Goal: achieve(condition), uses monitoring
  - Controlling progress, heuristics

Outline per the Book

3.1 *Representation*

- State variables, commands, refinement methods
- Example

3.2 *Acting*

- RAE (Refinement Acting Engine)
- Example
- Extensions

3.3 *Planning*

- Motivation and basic ideas
- Deterministic action models
- SeRPE (Sequential Refinement Planning Engine)

3.4 *Using Planning in Acting*

- Techniques
- Caveats

Motivation

- When dealing with an event or task, RAE may need to make either/or choices
  - *Agenda*: tasks $\tau_1, \tau_2, \dots, \tau_n$
    - Several tasks/events, how to prioritise?
  - Candidates for τ_1: $m_1, m_2, \dots$
    - 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 do not 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
- Generalisation of Hierarchical Task Network (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

SeRPE (Sequential Refinement Planning Engine)

- SeRPE inputs

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

- Which candidate method for τ?
  - SeRPE: Nondeterministic choice
    - Backtracking point
    - How to implement?
      - Hierarchical adaptation of backtracking, A*, GBFS, ...
  - RAE: Arbitrary choice
    - No search, purely reactive

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

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

SeRPE (Sequential Refinement Planning Engine)

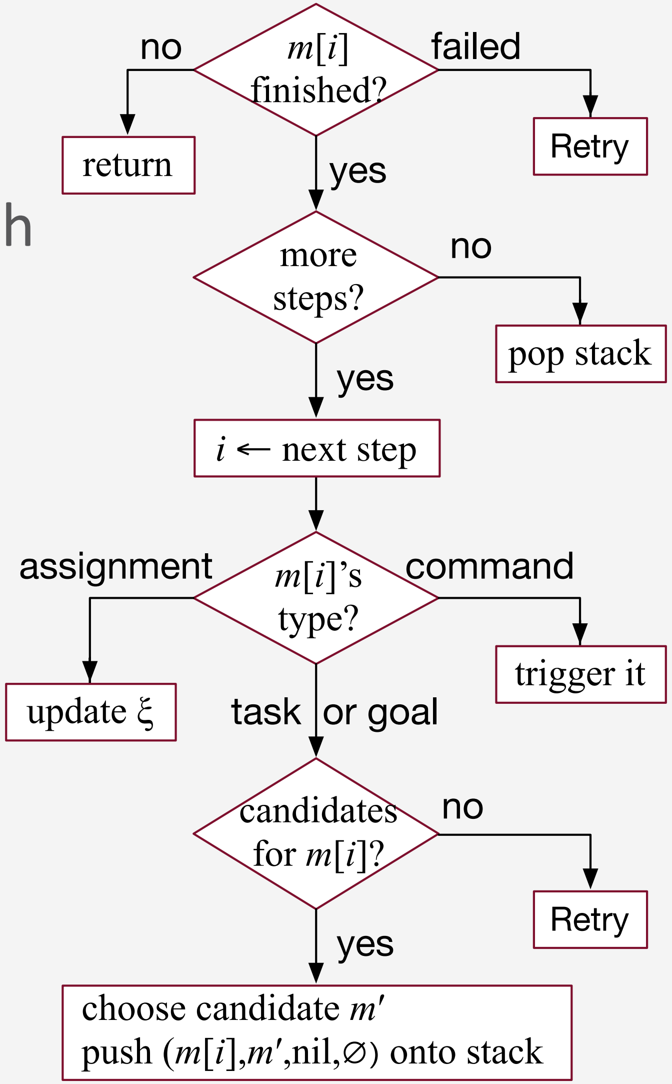
- SeRPE
  - One external task
  - Simulate progressing it all the way to the end
- RAE
  - Several external tasks
  - Each time through loop,
    progress each one by one step

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

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

Progress-to-finish

- Like RAE progress with a loop around it
  - Simulates the commands



no
$m[i]$ finished?
failed
return
Retry
yes
more steps?
no
pop stack
yes
$i \leftarrow$ next step
assignment
$m[i]$'s type?
command
update ξ
trigger it
task
or goal
candidates for $m[i]$?
no
Retry
yes
choose candidate m'
push $(m[i], m', \text{nil}, \varnothing)$ onto stack

```
Progress-to-finish($\mathcal{M}$,$\mathcal{A}$,$s$,$\tau$,$m$)
    $i \leftarrow$ nil; $\pi \leftarrow \langle\rangle$
    loop
        if $\tau$ is a goal and $s \models \tau$ then
            return $\pi$
        if $i$ is the last step of $m$ then
            if $\tau$ is a goal and $s \not\models \tau$ then
                return failure
            return $\pi$
        $i \leftarrow$ nextstep($m$,$i$)
        case type($m[i]$)
            assignment:
                update $s$ according to $m[i]$
            command:
                $a \leftarrow$ descriptive model of $m[i]$ in $\mathcal{A}$
                if $s \models$ pre($a$) then
                    $s \leftarrow \gamma(s,a)$; $\pi \leftarrow \pi.a$
                else
                    return failure
            task or goal:
                $\pi' \leftarrow$ SeRPE($\mathcal{M}$,$\mathcal{A}$,$s$,$m[i]$)
                if $\pi'$ = failure then
                    return failure
                $s \leftarrow \gamma(s,\pi')$; $\pi \leftarrow \pi.\ \pi'$
```

Progress-to-finish

- Inputs: $\mathcal{M}$ = {methods} , $\mathcal{A}$ = {action models}, s = initial state, τ = task or goal, m = chosen method
- Simulate RAE's goal monitoring
- If $m[i]$ is a command: Use action model to predict outcome
- If current step is a task: Call SeRPE recursively
  - Recursion stack ≈ RAE's refinement stack
- For failures, no Retry (RAE)
  - A failure means SeRPE could not find a solution
  - Implementation: hierarchical adaptations of backtracking, A*, GBFS, ...

```
Progress-to-finish($\mathcal{M}$,$\mathcal{A}$,$s$,$\tau$,$m$)
    $i$ ← nil; $\pi$ ← $\langle\rangle$
    loop
        if $\tau$ is a goal and $s \models \tau$ then
            return $\pi$
        if $i$ is the last step of $m$ then
            if $\tau$ is a goal and $s \not\models \tau$ then
                return failure
            return $\pi$
        $i$ ← nextstep($m$,$i$)
        case type($m[i]$)
            assignment:
                update $s$ according to $m[i]$
            command:
                $a$ ← descriptive model of $m[i]$ in $\mathcal{A}$
                if $s \models$ pre($a$) then
                    $s \leftarrow \gamma(s,a)$; $\pi \leftarrow \pi.a$
                else
                    return failure
            task or goal:
                $\pi'$ ← SeRPE($\mathcal{M}$,$\mathcal{A}$,$s$,$m[i]$)
                if $\pi'$ = failure then
                    return failure
                $s \leftarrow \gamma(s,\pi')$; $\pi \leftarrow \pi.\ \pi'$
```

Example

task
put-in-pile(c_1,p_2)

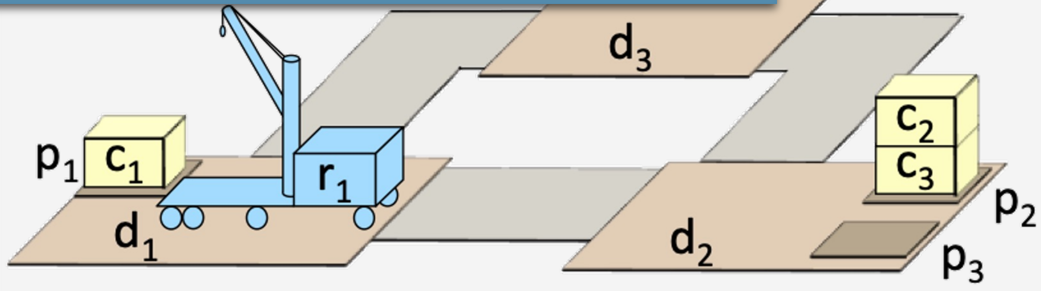
Candidates = {~~m1-put-in-pile(c_1,p_2)~~,
m2-put-in-pile(r,c_1,p_1,d,p',d')}

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

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

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* ≠ *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*)

s_0 = {loc(r_1)=d_1, cargo(r_1)=nil, occupied(d_1)=T,
occupied(d_2)=F, occupied(d_3)=F,
pos(c_1)=nil, pos(c_2)=c_3, pos(c_3)=nil,
pile(c_1)=p_1, pile(c_2)=p_2, pile(c_3)=p_2,
top(p_1)=c_1, top(p_2)=c_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

- The SeRPE pseudocode does not return this, but can easily be modified to do so

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

r_1,c_1,p_1,d_1,p_2,d_2

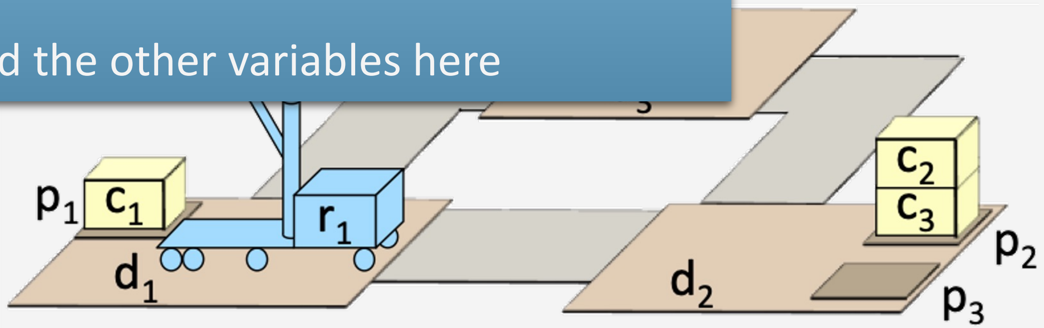
m2-put-in-pile(r, c, p, d, p', d')

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) ≠ 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)

- m2-put-in-pile starts with c=c_1, p'=p_2, and r, d, p', d' unbound
- Bind the other variables here



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)

```
Progress-to-finish($\mathcal{M}$,$\mathcal{A}$,$s$,$\tau$,$m$)
    $i \leftarrow$ nil; $\pi \leftarrow \langle\rangle$
    loop
        if $\tau$ is a goal and $s \models \tau$ then
            return $\pi$
        if $i$ is the last step of $m$ then
            if $\tau$ is a goal and $s \not\models \tau$ then
                return failure
            return $\pi$
        $i \leftarrow$ nextstep($m$,$i$)
        case type($m[i]$)
            assignment:
                update $s$ according to $m[i]$
            command:
                $a \leftarrow$ descriptive model of $m[i]$ in $\mathcal{A}$
                if $s \models$ pre($a$) then
                    $s \leftarrow \gamma(s,a)$; $\pi \leftarrow \pi.a$
                else
                    return failure
            task or goal:
                $\pi' \leftarrow$ SeRPE($\mathcal{M}$,$\mathcal{A}$,$s$,$m[i]$)
                if $\pi'$ = failure then
                    return failure
                $s \leftarrow \gamma(s,\pi')$; $\pi \leftarrow \pi.\ \pi'$
```

loc(r_1) = d_1 = *d*

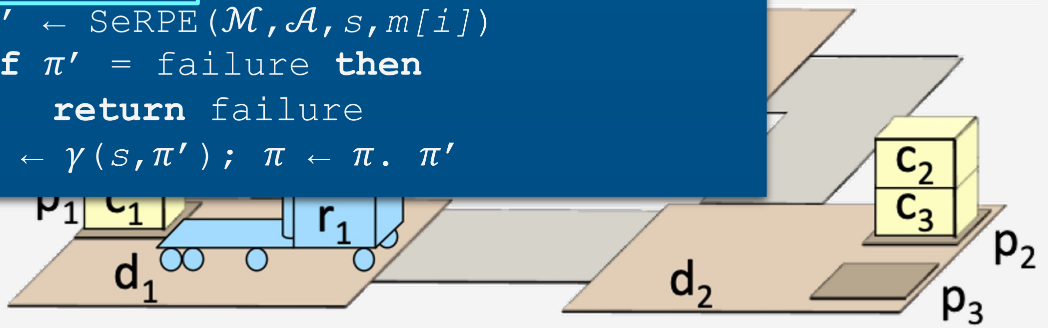
r_1,c_1,p_1,d_1,p_2,d_2

m2-put-in-pile(*r, c, p, d, p′, d′*)

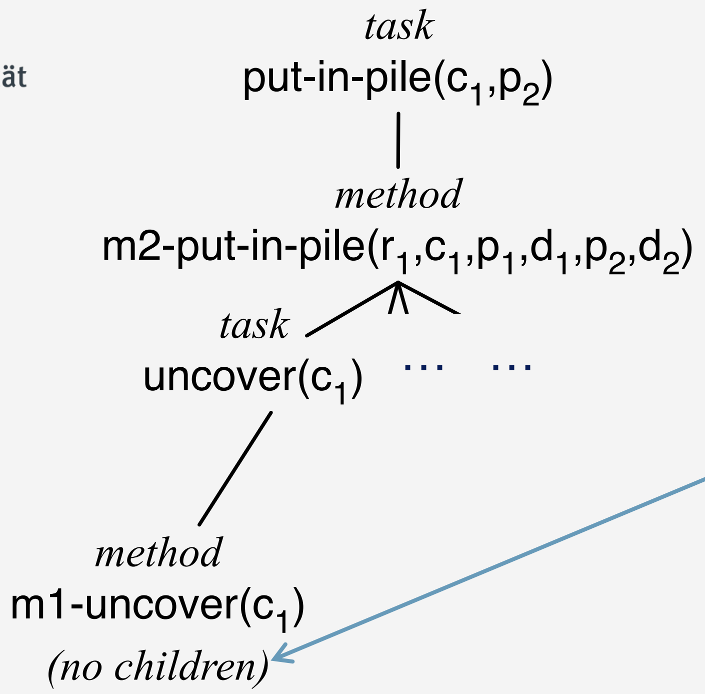
task: put-in-pile(*c, p′*)

pre: pile(*c*)=*p* ∧ at(*p, d*) ∧ at(*p′, d′*) ∧ *p* ≠ *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*)



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)

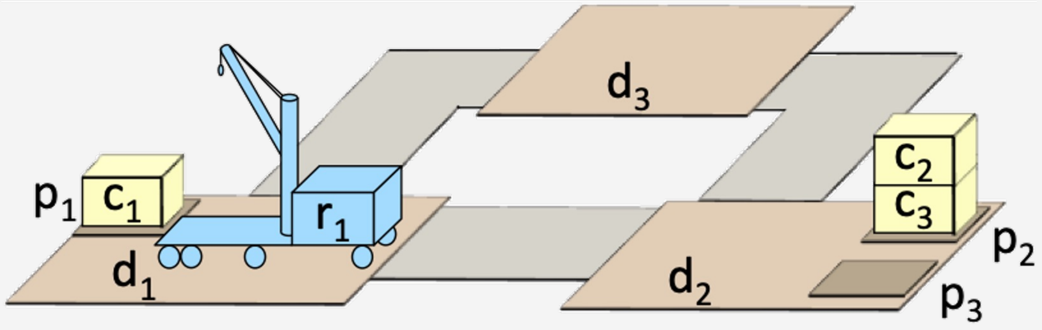
(no children)

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

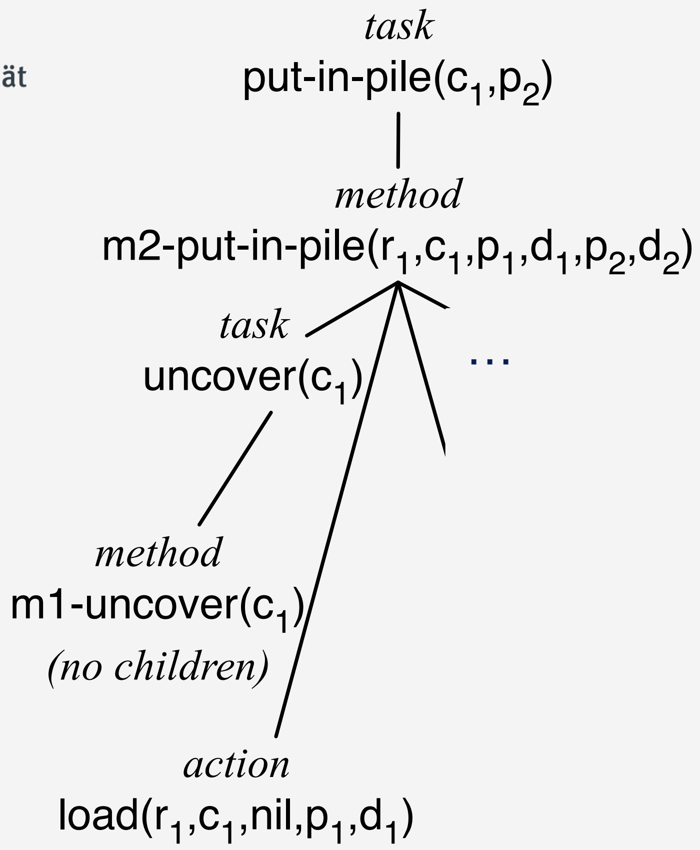
```
m2-uncover(r,c,c,p′,d)
    task:  uncover(c)
    pre:   pile(c)=p ∧ top(p)≠c
                ∧ at(p,d) ∧ at(p′,d) ∧ p′≠p
                ∧ loc(r)=d ∧ cargo(r)=nil
    body:    while top(p) ≠ c do
                        c′ ← top(p)
                        load(r,c′,pos(c′),p,d)
                 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

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≠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)
```



Example



task
$\text{put-in-pile}(c_1,p_2)$

method
$\text{m2-put-in-pile}(r_1,c_1,p_1,d_1,p_2,d_2)$

task
$\text{uncover}(c_1)$

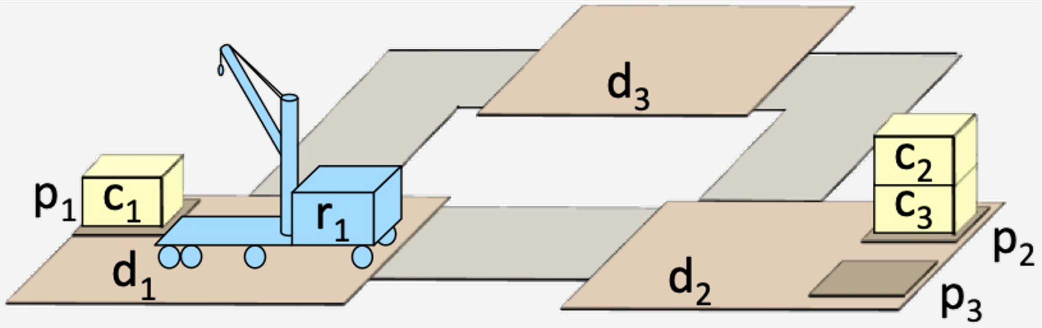
…

method
$\text{m1-uncover}(c_1)$
(no children)

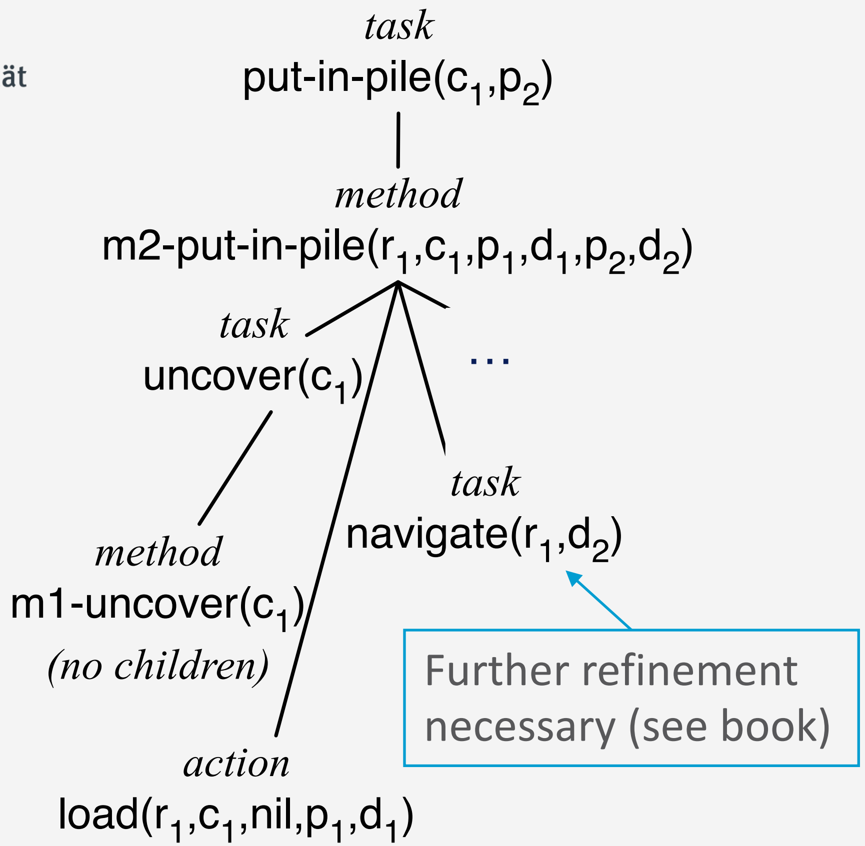
action
$\text{load}(r_1,c_1,\text{nil},p_1,d_1)$

r_1,c_1,p_1,d_1,p_2,d_2

```
m2-put-in-pile($r, c, p, d, p', d'$)
    …
    body:  if loc($r$) $\neq d$ then navigate($r,d$)
           uncover($c$)
           load($r, c$, pos($c$), $p, d$)  action
           if loc($r$) $\neq d'$ then navigate($r, d'$)
           unload($r, c$, top($p'$), $p', d$)
```



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)

…

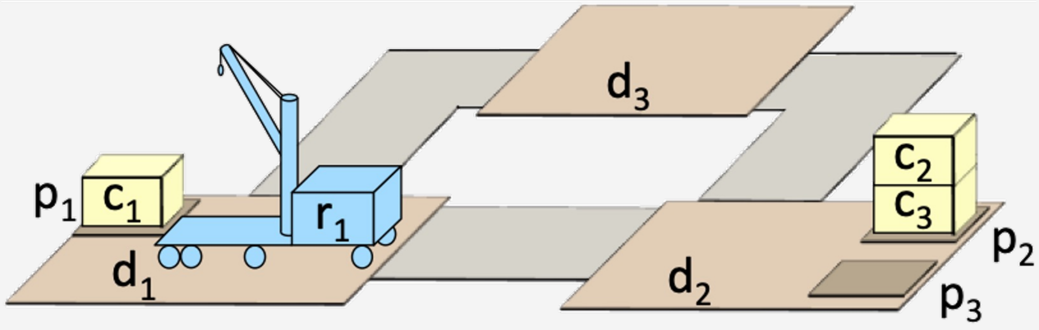
method
m1-uncover(c_1)
(no children)

task
navigate(r_1,d_2)

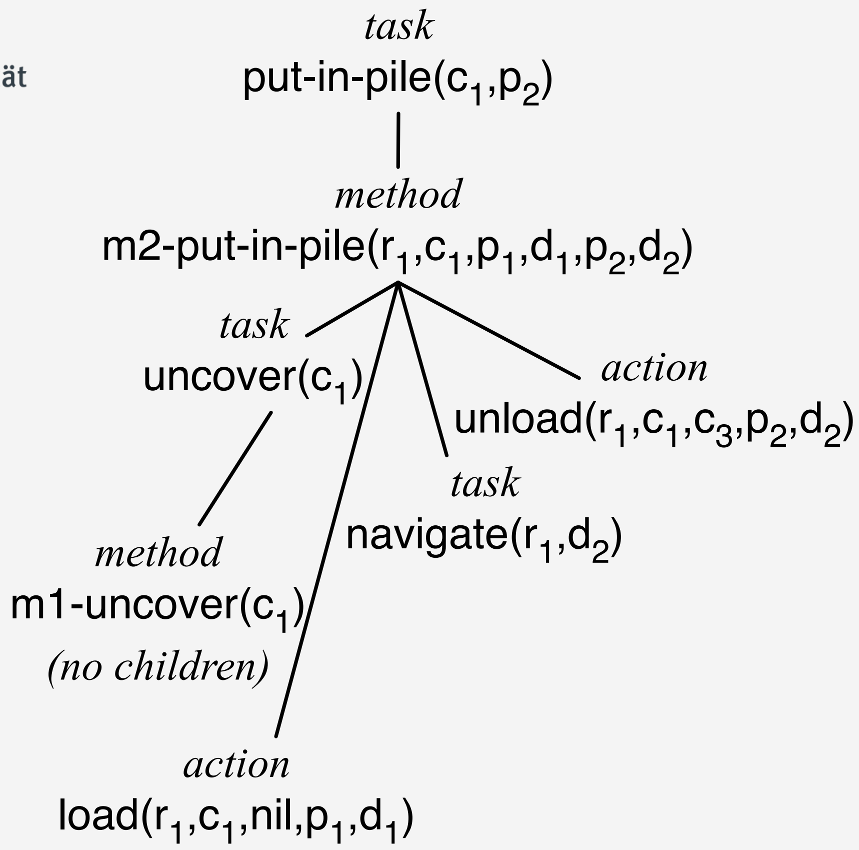
Further refinement necessary (see book)

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

```
                 r1,c1,p1,d1,p2,d2
m2-put-in-pile(r, c, p, d, p′, d′)
    …
    body:  if loc(r) ≠ d then navigate(r, d)
           uncover(c)
           load(r, c, pos(c), p, d)     task
           if loc(r) ≠ d′ then navigate(r, d′)
           unload(r, c, top(p′), p′, d)
```



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)

task
navigate(r_1,d_2)

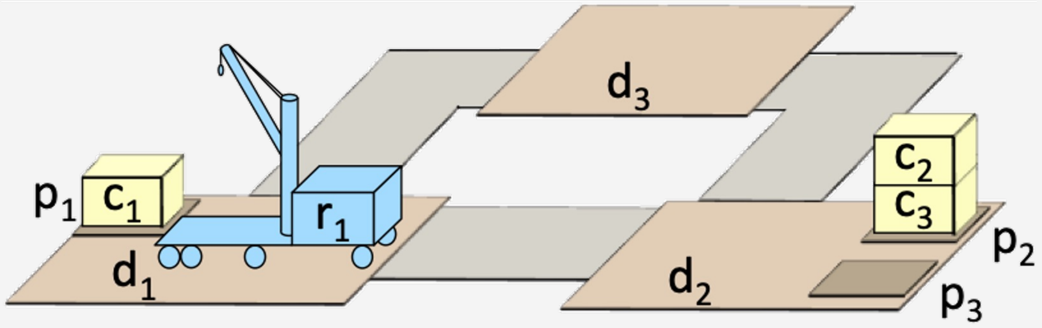
action
unload(r_1,c_1,c_3,p_2,d_2)

r_1,c_1,p_1,d_1,p_2,d_2

m2-put-in-pile(*r, c, p, d, p′, d′*)

…

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*) action



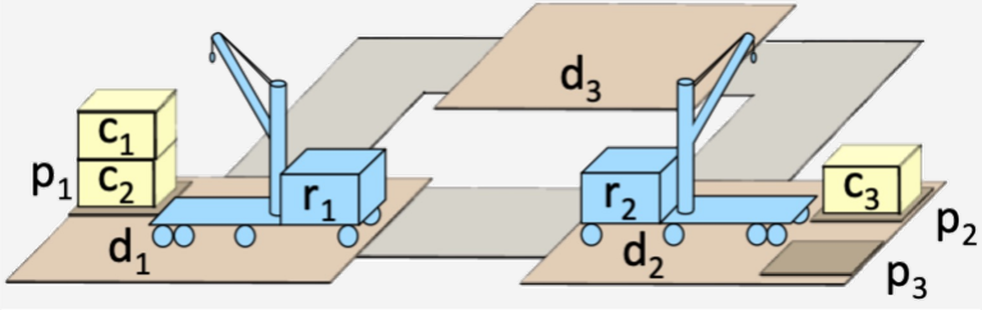
Heuristics For SeRPE

```
SeRPE($\mathcal{M}$,$\mathcal{A}$,$s$,$\tau$)
    Candidates ← Instances($\mathcal{M}$,$\tau$,$s$)
    if Candidates = ∅ then
        return failure
    nondeterministically choose $m$ ∈ Candidates
    return Progress-to-finish($\mathcal{M}$,$\mathcal{A}$,$s$,$\tau$,$m$)
```

- *Ad hoc* approaches:
    - Domain-specific estimates
    - 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 generalisation of HTN planning
    - In some cases, classical-planning heuristics can be used, in other cases they become intractable [Shivashankar *et al.*, ECAI-2016]

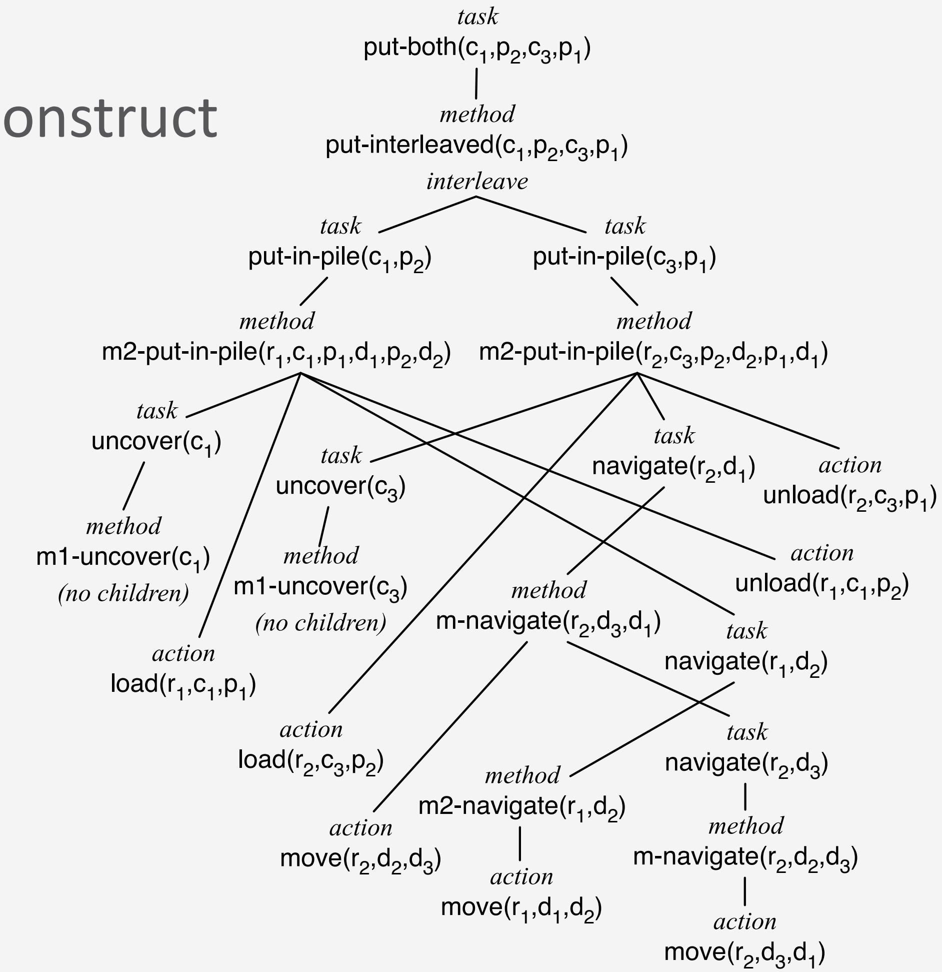
Interleaving

- Want to move c_1 to p_2, using this plan
  - $\langle load(r_1, c_1, c_2, p_1, d_1),$
    $move(r_1, d_1, d_2),$
    $unload(r_1, c_1, p_3, nil, d_2)\rangle$
- … and move c_3 to p_1 using this plan:
  - $\langle load(r_2, c_3, nil, p_2, d_2),$
    $move(r_2, d_2, d_3),$
    $move(r_2, d_3, d_1),$
    $unload(r_2, c_3, c_2, p_1, d_1)\rangle$
- For it to work, must interleave the plans
  - $\langle\ load(r_2, c_3, nil, p_2, d_2),$
    $move(r_2, d_2, d_3),$
    $load(r_1, c_1, c_2, p_1, d_1),$
    $move(r_1, d_1, d_2),$
    $unload(r_1, c_1, p_3, nil, d_2),$
    $move(r_2, d_3, d_1),$
    $unload(r_2, c_3, c_2, p_1, d_1)\rangle$



Interleaved Refinement Tree (IRT) Procedure

- SeRPE does not allow the “concurrent” programming construct
- Partial fix:
  extend SeRPE to interleave plans for different tasks
- Details:
  Section 3.3.2



task
put-both(c_1,p_2,c_3,p_1)

method
put-interleaved(c_1,p_2,c_3,p_1)
interleave

task
put-in-pile(c_1,p_2)

task
put-in-pile(c_3,p_1)

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

method
m2-put-in-pile$(r_2,c_3,p_2,d_2,p_1,d_1)$

task
uncover(c_1)

method
m1-uncover(c_1)
(no children)

task
uncover(c_3)

method
m1-uncover(c_3)
(no children)

action
load(r_1,c_1,p_1)

task
navigate(r_2,d_1)

action
unload(r_2,c_3,p_1)

action
unload(r_1,c_1,p_2)

method
m-navigate(r_2,d_3,d_1)

task
navigate(r_1,d_2)

action
load(r_2,c_3,p_2)

task
navigate(r_2,d_3)

method
m2-navigate(r_1,d_2)

action
move(r_2,d_2,d_3)

action
move(r_1,d_1,d_2)

method
m-navigate(r_2,d_2,d_3)

action
move(r_2,d_3,d_1)

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 perceives what objects are at location l
    - Can only perceive what is at its current location
    - If we knew this in advance, perception would not be necessary

- Action model

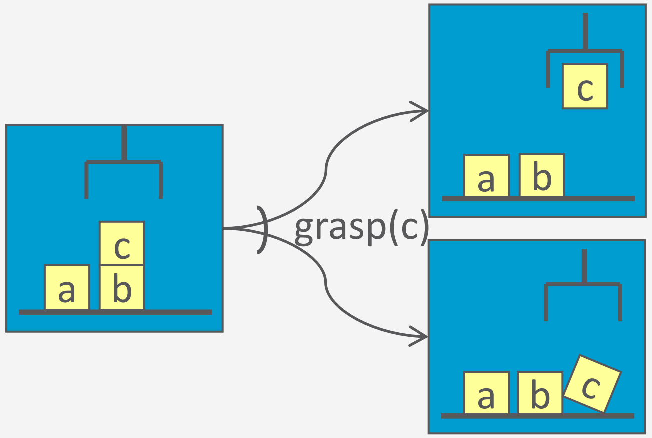
take(r,o,l)
pre: cargo(r) = nil, loc(r) = l, loc(o) = l
eff: cargo(r) $\leftarrow$ o, loc(o) $\leftarrow$ r

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

perceive(r,l)
?

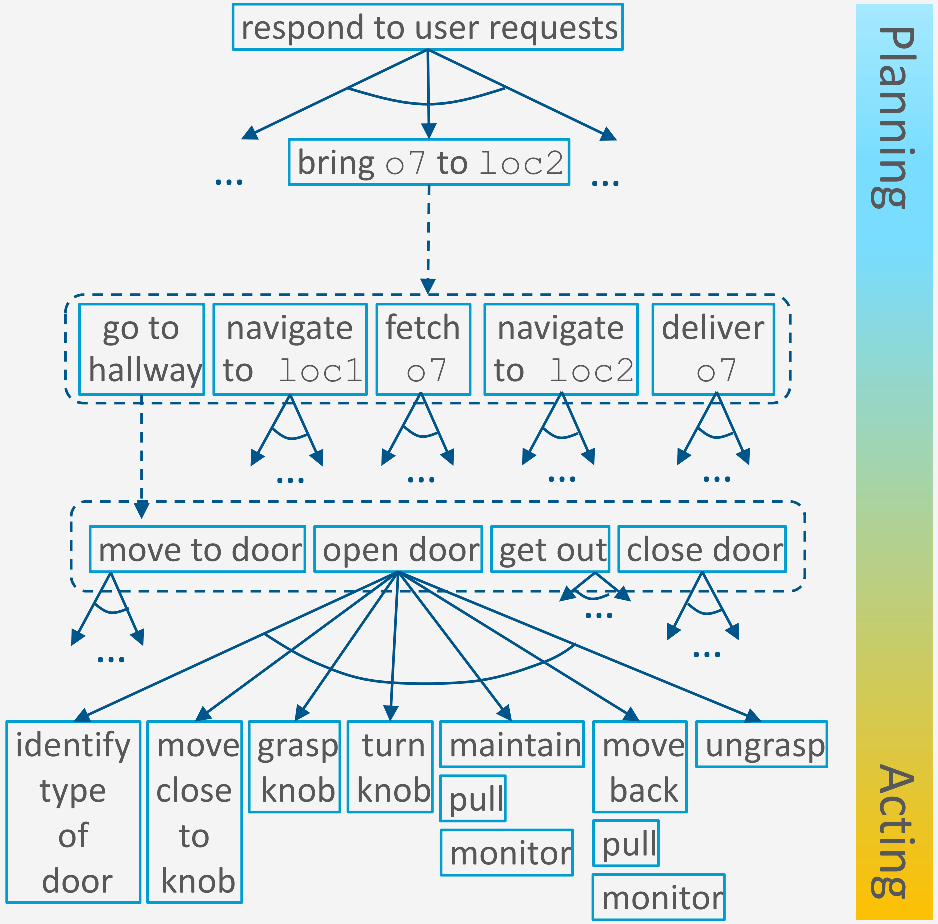
Limitation

- Most environments are inherently nondeterministic
  - Deterministic action models will not always make the right prediction
- Why use them?
  - Deterministic models ⇒ much simpler planning algorithms
    - Use when errors are infrequent and do not have severe consequences
    - Actor can fix the errors online



Planning/Acting at Different Levels

- Deterministic models may work better at some levels than others
- May want
  - RAE at some levels
  - RAE+planner at some levels
  - Planner at some levels
- In some cases, might want the planner to reason about nondeterministic outcomes
  - Later in lecture (Book: Ch. 5 + 6)
- Ongoing research on extending refinement planning to handle nondeterminism [Patra *et al.*, AAAI-2019]



respond to user requests

… bring `o7` to `loc2` …

go to hallway | navigate to `loc1` | fetch `o7` | navigate to `loc2` | deliver `o7`

… … … …

move to door | open door | get out | close door

… … …

identify type of door | move close to knob | grasp knob | turn knob | maintain | pull | monitor | move back | pull | monitor | ungrasp

Summary

- Refinement planning (SeRPE)
  - Plan by simulating RAE on a single external task/event/goal
  - Deterministic actions
    - OK if we are confident of outcome, can recover if things go wrong
  - Interleaved plans (brief example)

Outline per the Book

3.1 *Representation*
- State variables, commands, refinement methods
- Example

3.2 *Acting*
- RAE (Refinement Acting Engine)
- Example
- Extensions

3.3 *Planning*
- Motivation and basic ideas
- Deterministic action models
- SeRPE (Sequential Refinement Planning Engine)

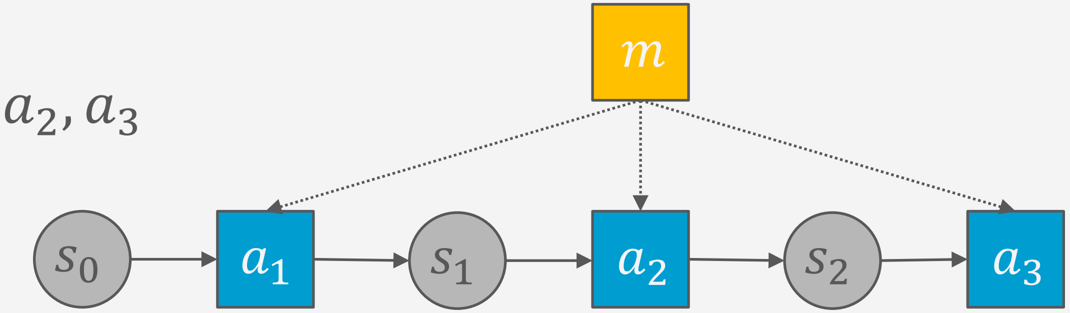
3.4 *Using Planning in Acting*
- Techniques
- Caveats

Acting and Refinement Planning

- Hierarchical acting with refinement planning
  - REAP: a RAE-like actor uses SeRPE-like planning at all levels
- Non-hierarchical actor with refinement planning
  - Refine-Lookahead, Refine-Lazy-Lookahead, Refine-Concurrent-Lookahead
    - Essentially the same as
      - Run-Lookahead, Run-Lazy-Lookahead, Run-Concurrent-Lookahead
  - But they call SeRPE instead of a classical planner for acting out a given task
  - Lookahead same as before
    - Receding horizon, sampling, subgoaling

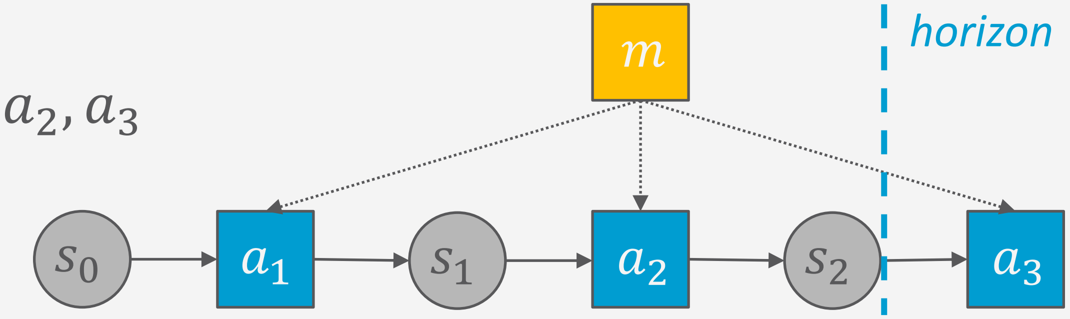
Caveats

- Start in state s_0, want to accomplish task τ
  - Refinement method m with task τ, pre s_0, body a_1, a_2, a_3
- Actor uses Refine-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 there next time



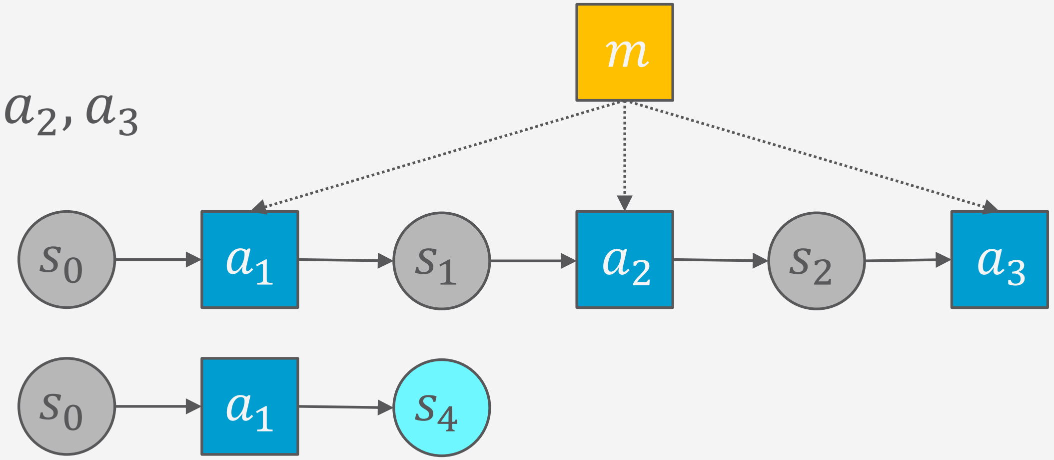
Caveats

- Start in state s_0, want to accomplish task τ
  - Refinement method m with task τ, pre s_0, body a_1, a_2, a_3
- Actor uses Refine-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
- Fixes
  - Can use the same fixes on previous slide, with one modification
    - Keep snapshot of SeRPE's search tree at the horizon, resume next time it is called



Caveats

- Start in state s_0, want to accomplish task τ
  - Refinement method m with task τ, pre: s_0, body a_1, a_2, a_3
- Actor uses Refine-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
- Fixes
  - Can use most of the fixes on last two slides, with this modification
    - Keep snapshot of SeRPE's search tree after each action
      - In example: restart it immediately after a_1, using s_4 as current state
  - Also: make recovery methods for unexpected states



Summary

- Acting and planning
  - Lookahead: search part of the search space, return a partial solution
  - Refine-Lookahead, Refine-Lazy-Lookahead, Refine-Concurrent-Lookahead
    - Like Run-Lookahead, Run-Lazy-Lookahead, Run-Concurrent-Lookahead, but call SeRPE
  - Caveats
    - Current state may not be what we expect
    - Possible ways to handle that

Outline per the Book

3.1 *Representation*
- State variables, commands, refinement methods
- Example

3.2 *Acting*
- RAE (Refinement Acting Engine)
- Example
- Extensions

3.3 *Planning*
- Motivation and basic ideas
- Deterministic action models
- SeRPE (Sequential Refinement Planning Engine)

3.4 *Using Planning in Acting*
- Techniques
- Caveats

⟹ Next: Planning and Acting with Temporal Models