Integrated Planning and Acting
Using Operational Models

Dana S. Nau and Sunandita Patra
University of Maryland
Motivation
Harbor Management

- Multiple levels of abstraction
  - Physical/managerial organization of harbor

- Higher levels:
  - Plan abstract tasks

- Lower levels:
  - Multiple agents, partial observability, dynamic change

- Continual online planning
  - Plans are abstract and partial until more detail needed
Hypothetical Worker Robot

- Multiple levels of abstraction
- At higher levels:
  - Plan abstract tasks
- At lower levels:
  - Nondeterminism, partial observability, dynamic change
- Continual online planning
  - Plans are abstract and partial until more detail needed

Tasks:

Planning
- ... 
- respond to user requests
- ... 
- bring o7 to room2
- ... 
- go to hallway
- navigate to room1
- ... 
- move to door
- open door
- get out
- close door

Acting
- ... 
- identify type of door
- move close to knob
- grasp knob
- turn knob
- move back
- ungrasp
- maintain
- move
- pull
- pull
- monitor
- monitor
- ... 
- navigate to room2
- ... 
- navigate to room1
- ... 
- ... 
- ... 
- ... 
- ... 
- ...
Planning and Acting

Planning

- **Prediction + search**
  - Search over predicted states, possible organizations of tasks and actions
- Uses *descriptive* models (e.g., PDDL)
  - predict *what* the actions will do
  - don’t include instructions for performing it

Acting

- **Performing actions**
  - Dynamic, unpredictable, partially observable environment
  - Adapt to context, react to events
- Uses *operational* models
  - instructions telling *how* to perform the actions
Opening a Door

Tasks:

- respond to user requests
  - bring o7 to room2
    - go to hallway
    - navigate to room1
    - fetch o7
    - navigate to room2
    - deliver o7
    - move to door
    - open door
    - get out
    - close door
    - identify type of door
    - move close to knob
    - grasp knob
    - turn knob
    - maintain
    - move back
    - ungrasp
    - move
    - pull
    - monitor
    - monitor

Planning

Acting
Opening a Door

- Different methods, depending on what kind of door
  - Sliding or hinged?
Opening a Door

- Different methods, depending on what kind of door
  - Sliding or hinged?
  - Hinge on left or right?
Opening a Door

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

![Diagram of opening a door]

- Identify type of door
- Move close to knob
- Grasp knob
- Turn knob
- Maintain
- Pull
- Monitor
- Move back
- Ungrasp

- Open door
- Get out
- Close door
- Move to door
- Fetch
- Navigate to room2
- Navigate to room1
- Identify type of door
- Respond to user requests
- Bring o7 to room2
- Go to hallway
- Deliver o7
Opening a Door

- 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

- 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

- 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?
RAE and UPOM

- Python implementation:
  - [https://github.com/sunandita/ICAPS_Summer_School_RAE_2020](https://github.com/sunandita/ICAPS_Summer_School_RAE_2020)
  - Full code: [https://bitbucket.org/sunandita/rae/](https://bitbucket.org/sunandita/rae/)

- Related publications
Outline

1. Motivation

2. **Representation** – state variables, commands, tasks, refinement methods

3. **Acting** – Rae (Refinement Acting Engine)

4. **Planning** – UPOM (UCT-like Planner for Operational Models)

5. **Acting with Planning** – Rae + UPOM

6. **Using the implementation** – Rae code, UPOM code, examples
Objects
- **Robots** = \{r1, r2\}
- **Containers** = \{c1, c2\}
- **Locations** = \{loc1, loc2, loc3, loc4\}

Rigid relations (properties that won’t change)
- adjacent(loc0,loc1), adjacent(loc1,loc0), adjacent(loc1,loc2), adjacent(loc2,loc1), adjacent(loc2,loc3), adjacent(loc3,loc2), adjacent(loc3,loc4), adjacent(loc4,loc3)

State variables (fluents)
- where \( r \in \text{Robots}, \ c \in \text{Containers}, \ l \in \text{Locations} \)
- \( \text{loc}(r) \in \text{Locations} \)
- \( \text{cargo}(r) \in \text{Containers} \cup \{\text{empty}\} \)
- \( \text{pos}(c) \in \text{Locations} \cup \text{Robots} \cup \{\text{unknown}\} \)
- \( \text{view}(l) \in \{T, F\} \)
  - Whether a robot has looked at location \( l \)
  - If \( \text{view}(l) = T \) then \( \text{pos}(c) = l \) for every container \( c \) at \( l \)

Commands to the execution platform:
- \( \text{take}(r,o,l) \): \( r \) takes object \( o \) at location \( l \)
- \( \text{put}(r,o,l) \): \( r \) puts \( o \) at location \( l \)
- \( \text{perceive}(r,l) \): robot \( r \) perceives what objects are at \( l \)
- \( \text{move-to}(r,l) \): robot \( r \) moves to location \( l \)
Tasks and Methods

- **Task**: an activity for the actor to perform
  - taskname($arg_1, \ldots, arg_k$)

- For each task, one or more refinement methods
  - Operational models telling how to perform the task

```plaintext
method-name($arg_1, \ldots, arg_k$)  
  task: task-identifier
  pre: test
  body: a program
```

```plaintext
m-fetch1(r,c)  
  task: fetch(r,c)
  pre: pos(c) = unknown
  body:
  if $\exists l$ (view($l$) = F) then
    move-to(r,l)
    perceive(r,l)
    if pos(c) = l then
      take(r,c,l)
    else
      fetch(r,c)
  else
    fail
```

```plaintext
m-fetch2(r,c)  
  task: fetch(r,c)
  pre: pos(c) \neq unknown
  body:
  if loc(r) = pos(c) then
    take(r,c,pos(c))
  else do
    move-to(r,pos(c))
    take(r,c,pos(c))
```

command

Nau and Patra – ICAPS 2020 Summer School
Outline

1. Motivation

2. **Representation** – state variables, commands, tasks, refinement methods

3. **Acting** – Rae (Refinement Acting Engine)

4. **Planning** – UPOM (UCT-like Planner for Operational Models)

5. **Acting with Planning** – Rae + UPOM

6. **Using the implementation** – Rae code, UPOM code, examples
Rae (Refinement Acting Engine)

- Performs multiple tasks in parallel
  - Purely reactive, no lookahead
- For each task or event $\tau$, a *refinement stack*
  - execution stack
  - corresponds to current path in Rae’s search tree for $\tau$
- Agenda = {all current refinement stacks}

```
procedure Rae:
    loop:
        for every new external task or event $\tau$ do
            choose a method instance $m$ for $\tau$
            create a refinement stack for $\tau, m$
            add the stack to Agenda
        for each stack $\sigma$ in Agenda
            Progress($\sigma$)
            if $\sigma$ is finished then remove it
```
## Representation

### Objects

- **Robots** = \{r_1, r_2\}
- **Containers** = \{c_1, c_2\}
- **Locations** = \{loc_1, loc_2, loc_3, loc_4\}

### Rigid relations (properties that won’t change)

- adjacent(loc_0, loc_1), adjacent(loc_1, loc_0), adjacent(loc_1, loc_2), adjacent(loc_2, loc_1), adjacent(loc_2, loc_3), adjacent(loc_3, loc_2), adjacent(loc_3, loc_4), adjacent(loc_4, loc_3)

### State variables (fluents)

- where \( r \in \text{Robots} \), \( c \in \text{Containers} \), \( l \in \text{Locations} \)
  - \( \text{loc}(r) \in \text{Locations} \)
  - \( \text{cargo}(r) \in \text{Containers} \cup \{\text{nil}\} \)
  - \( \text{pos}(c) \in \text{Locations} \cup \text{Robots} \cup \{\text{unknown}\} \)
  - \( \text{view}(l) \in \{T, F\} \)
    - Whether a robot has looked at location \( l \)
    - If \( \text{view}(l) = T \) then \( \text{pos}(c) = l \) for every container \( c \) at \( l \)

### Commands to the execution platform:

- **take**(\( r, o, l \)): \( r \) 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 \( l \)
- **move-to**(\( r, l \)): robot \( r \) moves to location \( l \)
procedure Rae:

loop:

for every new external task or event $\tau$ do

choose a method instance $m$ for $\tau$

create a refinement stack for $\tau$, $m$

add the stack to Agenda

for each stack $\sigma$ in Agenda

Progress($\sigma$)

if $\sigma$ is finished then remove it

m-fetch1($r,c$)

- task: fetch($r,c$)
- pre: pos($c$) = unknown
- body:
  - if $\exists l$ (view($l$) = F) then
    - move-to($r,l$)
    - perceive($r,l$)
    - if pos($c$) = $l$ then
      - take($r,c,l$)
    - else fetch($r,c$)
  - else fail

m-fetch2($r,c$)

- task: fetch($r,c$)
- pre: pos($c$) $\neq$ unknown
- body:
  - if loc($r$) = pos($c$) then
    - take($r,c,pos(c)$)
  - else do
    - move-to($r,pos(c)$)
    - take($r,c,pos(c)$)

- Container locations unknown
- Partially observable
  - Robot only sees current location
Example

\[ \text{m-fetch1}(r,c) \quad r = r_0, c = c_2 \]

**task:** fetch\((r,c)\)

**pre:** pos\((c) = \text{unknown} \)

**body:**
- if \( \exists l \) (view\((l) = F) \) then
  - move-to\((r,l)\)
  - perceive\((r,l)\)
- if pos\((c) = l\) then
  - take\((r,c,l)\)
- else fetch\((r,c)\)
- else fail

\[ \text{m-fetch2}(r,c) \]

**task:** fetch\((r,c)\)

**pre:** pos\((c) \neq \text{unknown} \)

**body:**
- if loc\((r) = \text{pos}(c)\) then
  - take\((r,c,\text{pos}(c))\)
- else do
  - move-to\((r,\text{pos}(c))\)
  - take\((r,c,\text{pos}(c))\)

**Candidates**
\[ \{ \text{m-fetch}(r_1,c_2), \text{m-fetch}(r_2,c_2) \} \]

\[ \text{Search tree} \]

- Container locations unknown
- Partially observable
  - Robot only sees current location

\[ \tau \]

\[ \text{procedure Rae:} \]

\[ \text{loop:} \]

- for every new external task or event \( \tau \) do
  - choose a method instance \( m \) for \( \tau \)
  - create a refinement stack for \( \tau, m \)
  - add the stack to \( \text{Agenda} \)
- for each stack \( \sigma \) in \( \text{Agenda} \)
  - Progress\((\sigma)\)
  - if \( \sigma \) is finished then remove it
**Example**

**Search tree**

- **Candidates**
  - \{m-fetch(r1,c2), m-fetch(r2,c2)\}

**procedure Rae:**

```
loop:
  for every new external task or event \(\tau\) do
    choose a method instance \(m\) for \(\tau\)
    create a refinement stack for \(\tau, m\)
    add the stack to Agenda
for each stack \(\sigma\) in Agenda
  Progress(\(\sigma\))
  if \(\sigma\) is finished then remove it
```

**m-fetch1\((r,c)\)**

- **task:** \(fetch(r,c)\)
- **pre:** \(pos(c) = \text{unknown}\)
- **body:**
  - if \(\exists l\) (\(\text{view}(l) = F\)) then
    - move-to\((r,l)\)
    - perceive\((r,l)\)
    - if \(pos(c) = l\) then
      - take\((r,c,l)\)
    - else fetch\((r,c)\)
  - else fail

**m-fetch2\((r,c)\)**

- **task:** \(fetch(r,c)\)
- **pre:** \(pos(c) \neq \text{unknown}\)
- **body:**
  - if \(\text{loc}(r) = pos(c)\) then
    - take\((r,c,pos(c))\)
  - else do
    - move-to\((r,pos(c))\)
    - take\((r,c,pos(c))\)

**Example**

\(r = r1, c = c2\)

- **Container locations unknown**
- **Partially observable**
  - Robot only sees current location

- **Search tree**

  - Container locations unknown
  - Partially observable
  - Robot only sees current location
m-fetch1(r,c)  \( r = r_1, c = c_2 \)

**task:** fetch\((r,c)\)

**pre:** pos\((c)\) = unknown

**body:**
- if \( \exists l \) (view\((l)\) = F) then
  - move-to\((r,l)\)
  - perceive\((r,l)\)
  - if pos\((c)\) = \( l \) then
    - take\((r,c,l)\)
  - else search\((r,c)\)
- else fail

m-fetch2(r,c)

**task:** fetch\((r,c)\)

**pre:** pos\((c)\) \( \neq \) unknown

**body:**
- if loc\((r)\) = pos\((c)\) then
  - take\((r,c,pos(c))\)
- else do
  - move-to\((r,pos(c))\)
  - take\((r,c,pos(c))\)

---

**Candidates**
\( = \{m\text{-}fetch(r_1,c_2), m\text{-}fetch(r_2,c_2)\} \)

---

**Example**

- Container locations unknown
- Partially observable
  - Robot only sees current location

---

**procedure Rae:**

**loop:**
for every new external task or event \( \tau \) do
choose a method instance \( m \) for \( \tau \)
create a refinement stack for \( \tau, m \)
add the stack to **Agenda**

for each stack \( \sigma \) in **Agenda**
Progress\((\sigma)\)
if \( \sigma \) is finished then remove it
m-fetch1(r,c)  \( r = r_1, \ c = c_2 \)

- **task:** fetch \((r,c)\)
- **pre:** pos\((c)\) = unknown
- **body:**
  - if \(\exists l \) (view\((l) = F\)) then
    - move-to\((r,l)\)
  - perceive\((r,l)\)
  - if pos\((c)\) = \(l\) then
    - take\((r,c,l)\)
  - else fetch\((r,c)\)
  - else fail

m-fetch2(r,c)

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

**Example**

- Container locations unknown
- Partially observable
  - Robot only sees current location
m-fetch\(_1(r,c)\) \( r = r_1, c = c_2 \)

- **task**: fetch\((r,c)\)
- **pre**: pos\((c)\) = unknown
- **body**: 
  - if \( l = \text{loc1} \) then
    - move-to\((r,l)\)
    - perceive\((r,l)\)
    - if pos\((c)\) = \( l \) then
      - take\((r,c,l)\)
    - else fetch\((r,c)\)
  - else fail

m-fetch\(_2(r,c)\)

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

**Example**

- Container locations unknown
- Partially observable
  - Robot only sees current location

**Progress(\(\sigma\))**:

- **started \(m\)?**
  - yes 
    - m's current step a command?
      - yes 
        - command status?
          - yes 
            - succeed
            - no 
              - more steps in \(m\)?
                - yes 
                  - pop(\(\sigma\))
                - no 
                  - retry \(\tau\) using an untried candidate
          - no 
            - return
      - failed
    - no 
      - retry \(\tau\) using an untried candidate

- no 
  - no more steps in \(m\)?
    - yes 
      - pop(\(\sigma\))
    - no 
      - retry \(\tau\) using an untried candidate
Example

m-fetch1(r, c) \ r = r1, c = c2

- task: fetch(r, c)
- pre: pos(c) = unknown
- body:
  \ l = loc1
  - if \ \exists l \ (view(l) = F) then
    - move-to(r, l)
    - perceive(r, l)
    - if pos(c) = l then
      - take(r, c, l)
    - else fetch(r, c)
  - else fail

m-fetch2(r, c)

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

- Container locations unknown
- Partially observable
  - Robot only sees current location

Progress(\sigma):

- started m?
  - yes
  - is m’s current step a command?
    - yes
    - command status?
      - running
      - return success
      - succeeded
      - more steps in m?
        - yes
        - \tau’ ← next step of m
        - pop(\sigma)
      - no
    - failed
      - retry \tau using an untried candidate
  - no
- no more steps in m?
  - yes
  - command
    - type(\tau’)
      - assignment
      - candidates for \tau’?
        - yes
        - send \tau’ to the execution platform
        - candidates
          - choose a candidate \ m’
          - push (\tau’, m’, …) onto \sigma
        - no
      - retry \tau using an untried candidate
    - no
- no candidates for \tau’?
  - yes
Example

\[ m\text{-fetch1}(r,c) \quad r = r_1, \quad c = c_2 \]

- **task:** fetch\((r,c)\)
- **pre:** pos\((c) = \text{unknown} \)
- **body:**
  - if \(\exists l \) (view\((l) = F\)) then
    - move-to\((r,l)\)
    - perceive\((r,l)\)
    - if pos\((c) = l\) then
      - take\((r,c,l)\)
    - else fetch\((r,c)\)
  - else fail

\[ m\text{-fetch2}(r,c) \]

- **task:** fetch\((r,c)\)
- **pre:** pos\((c) \neq \text{unknown} \)
- **body:**
  - if loc\((r) = \text{pos}\((c)\) then
    - take\((r,c,\text{pos}\((c)\))\)
  - else do
    - move-to\((r,\text{pos}\((c)\))\)
    - take\((r,c,\text{pos}\((c)\))\)

**procedure Rae:**

**loop:**

for every new external task or event \(\tau\) do

- choose a method instance \(m\) for \(\tau\)
- create a refinement stack for \(\tau, m\)
- add the stack to Agenda

for each stack \(\sigma\) in Agenda

- **Progress**\((\sigma)\)
  - if \(\sigma\) is finished then remove it

**Progress**\((\sigma)\):

- \(\tau\) \(\rightarrow\) next step of \(m\)
- for \(\tau\) \(\rightarrow\) command

- if \(\tau\) \(\rightarrow\) success
  - send \(\tau\) to the execution platform
  - update state

- if \(\tau\) \(\rightarrow\) failed
  - retry \(\tau\) using an untried candidate

- if \(\tau\) \(\rightarrow\) no candidates
  - retry \(\tau\) using an untried candidate

- if \(\tau\) \(\rightarrow\) more steps in \(m\)?
  - yes
    - pop\((\sigma)\)
  - no

- if \(\tau\) \(\rightarrow\) no steps in \(m\)?
  - yes
    - pop\((\sigma)\)
  - no
Example

m-fetch1(r, c)  \( r = r_1, c = c_2 \)
任务: fetch(r, c)
前置: pos(c) = unknown
主体:
1. \( l = \text{loc}_1 \)
2. if \( \exists l \) (view(l) = F) then
   a. move-to(r, l)
   b. perceive(r, l)
3. if pos(c) = l then
   a. take(r, c, l)
4. else fetch(r, c)
5. else fail

m-fetch2(r, c)
任务: fetch(r, c)
前置: pos(c) \( \neq \) unknown
主体:
1. if loc(r) = pos(c) then
   a. take(r, c, pos(c))
2. else do
   a. move-to(r, pos(c))
   b. take(r, c, pos(c))

Progress(\( \sigma \)):
1. started \( m \)？
   a. is \( m \’s \) current step a command？
   i. yes
   1. command status？
      a. succeeded
      1. \( \tau \) ← next step of \( m \)
      2. type(\( \tau \))
      3. command
      4. send \( \tau \) to the execution platform
   ii. failed
      a. retry \( \tau \) using an untried candidate
   b. no
      a. no more steps in \( m \)？
         i. yes
            a. return success
         ii. no
            a. pop(\( \sigma \))

2. no
   a. no candidates for \( \tau \)？
      a. yes
         a. retry \( \tau \) using an untried candidate
      b. no
         a. retry \( \tau \) using an untried candidate

3. yes
   a. assignment
   b. type(\( \tau \))
   c. command
   d. candidates for \( \tau \)？
      a. yes
         a. choose a candidate \( m \)‘
         1. push \( \tau, m, \ldots \) onto \( \sigma \)
      b. no
         a. retry \( \tau \) using an untried candidate
Example

\[ \text{m-fetch1}(r,c) \quad r = r1, \: c = c2 \]

\begin{itemize}
  \item task: \quad \text{fetch}(r,c)
  \item pre: \quad \text{pos}(c) = \text{unknown}
  \item body:
    \begin{itemize}
      \item if \( \exists l (\text{view}(l) = \text{F}) \) then
        \begin{itemize}
          \item move-to(r,l)
          \item perceive(r,l)
          \item if \text{pos}(c) = l \text{ then take}(r,c,l)
        \end{itemize}
      \item else \text{fetch}(r,c)
      \item else \text{fail}
    \end{itemize}
\end{itemize}

\[ \text{m-fetch2}(r,c) \]

\begin{itemize}
  \item task: \quad \text{fetch}(r,c)
  \item pre: \quad \text{pos}(c) \neq \text{unknown}
  \item body:
    \begin{itemize}
      \item if \text{loc}(r) = \text{pos}(c) \text{ then take}(r,c,\text{pos}(c))
      \item else do
        \begin{itemize}
          \item move-to(r,\text{pos}(c))
          \item take(r,c,\text{pos}(c))
        \end{itemize}
    \end{itemize}
\end{itemize}

\[ \text{Progress}(\sigma): \]

- \text{started m?}
  \begin{itemize}
    \item yes: \text{is m’s current step a command?}
      \begin{itemize}
        \item yes: \text{return success}
        \item failed: retry \( \tau \) using an untried candidate
      \end{itemize}
  \end{itemize}

- \text{more steps in m?}
  \begin{itemize}
    \item yes: \text{next step of m}
    \item no: \text{pop(\sigma)}
  \end{itemize}

- \text{type(\tau’)}
  \begin{itemize}
    \item task: \text{send} \tau’ \text{ to the execution platform}
  \end{itemize}

- \text{candidates for \tau’?}
  \begin{itemize}
    \item yes: choose a candidate \( m' \) push \( (\tau', m', \ldots) \) onto \( \sigma \)
    \item no: retry \( \tau \) using an untried candidate
  \end{itemize}
Example

\text{m-fetch1}(r,c) \quad r = r1, c = c2

\text{task: fetch}(r,c) \\
\text{pre: pos}(c) = \text{unknown} \\
\text{body:} \\
\quad l = \text{loc1} \\
\quad \text{if } \exists l \, (\text{view}(l) = F) \, \text{then} \\
\quad \quad \text{move-to}(r,l) \\
\quad \quad \text{perceive}(r,l) \\
\quad \quad \text{if pos}(c) = l \, \text{then} \\
\quad \quad \quad \text{take}(r,c,l) \\
\quad \quad \text{else} \, \text{fetch}(r,c) \\
\quad \text{else} \, \text{fail}

\text{m-fetch2}(r,c) \\
\text{task: fetch}(r,c) \\
\text{pre: pos}(c) \neq \text{unknown} \\
\text{body:} \\
\quad \text{if loc}(r) = \text{pos}(c) \, \text{then} \\
\quad \quad \text{take}(r,c,\text{pos}(c)) \\
\quad \text{else} \, \text{do} \\
\quad \quad \text{move-to}(r,\text{pos}(c)) \\
\quad \quad \text{take}(r,c,\text{pos}(c))

\text{Progress(σ):}

\text{continue} \quad \text{m’s current step a command?} \\
\quad \text{running} \quad \text{command status?} \\
\quad \quad \text{success} \\
\quad \quad \text{more steps in } m? \\
\quad \quad \text{yes} \\
\quad \quad \text{type}(τ’) \\
\quad \quad \text{task} \\
\quad \quad \text{send } τ’ \text{ to the execution platform} \\
\quad \quad \text{candidates for } τ’? \\
\quad \quad \text{yes} \\
\quad \quad \text{choose a candidate } m’ \\
\quad \quad \text{push } (τ’, m’, \ldots) \text{ onto } σ \\
\quad \quad \text{no} \\
\quad \quad \text{retry } τ \text{ using an untried candidate} \\
\quad \text{failed} \\
\quad \text{return} \\
\quad \text{success} \\
\quad \text{no more steps in } m? \\
\quad \text{yes} \\
\quad \text{pop}(σ) \\
\quad \text{no candidates for } τ’? \\
\quad \text{yes} \\
\quad \text{no} \\
\quad \text{no} \\
\quad \text{retry } τ \text{ using an untried candidate} \\
\quad \text{no} \\
\quad \text{started } m? \\
\quad \text{no}
m-fetch1\((r,c)\)
\[
\text{task: } \text{fetch}(r,c) \\
\text{pre: } \text{pos}(c) = \text{unknown} \\
\text{body: } \begin{cases} 
    l = \text{loc1} & \text{if } \exists l \ (\text{view}(l) = F) \\
    \text{move-to}(r,l) \\
    \text{perceive}(r,l) \\
    \text{if } \text{pos}(c) = l & \text{then } \text{take}(r,c,l) \\
    \text{else } \text{fetch}(r,c) \\
\end{cases} \\
\text{else fail}
\]

m-fetch2\((r,c)\)
\[
\text{task: } \text{fetch}(r,c) \\
\text{pre: } \text{pos}(c) \neq \text{unknown} \\
\text{body: } \begin{cases} 
    \text{if } \text{loc}(r) = \text{pos}(c) & \text{then } \text{take}(r,c,\text{pos}(c)) \\
    \text{else } \text{do } \\
    \text{move-to}(r,\text{pos}(c)) \\
    \text{take}(r,c,\text{pos}(c)) \\
\end{cases}
\]

Example

Search tree

\[
\begin{array}{c}
\text{fetch}(r_0,c_2) \\
\text{m-fetch1}(r_1,c_2) \\
\text{move-to}(r_1,\text{loc1}) \\
\text{perceive}(r_1,\text{loc1}) \\
\end{array}
\]

Progress(\(\sigma\)):

- started \(m\)?
  - no
  - yes
    - \(m\)'s current step a command?
      - no
        - \(\tau\) ← next step of \(m\)
      - yes
        - retry \(\tau\) using an untried candidate
  - yes
    - return
    - succeed
    - pop(\(\sigma\))

- command status?
  - running
  - yes
    - more steps in \(m\)?
      - yes
        - retry \(\tau\) using an untried candidate
      - no
        - τ' ← next step of \(m\)
        - type(τ')
        - command
          - update state
          - send \(\tau\) to the execution platform
          - candidates for \(\tau'\)?
            - yes
              - choose a candidate \(m'\)
              - push (\(\tau', m', \ldots\)) onto \(\sigma\)
            - no
              - retry \(\tau\) using an untried candidate
Example

Candidates = \{m\text{-}fetch(r_1,c_2), m\text{-}fetch(r_2,c_2)\}

Progress(\sigma):
- started \(m\)?
- yes
- no
  - \(m\)'s current step a command?
    - yes
      - command status?
        - success
          - more steps in \(m\)?
            - yes
              - \(\tau\) ← next step of \(m\)
            - no
              - pop(\sigma)
        - failed
          - retry \(\tau\) using an untried candidate
      - retry \(\tau\) using an untried candidate
    - no
      - \(\tau\) ← next step of \(m\)
- no
- no
- no

\[\text{Candidates} = \{m\text{-}fetch(r_1,c_2), m\text{-}fetch(r_2,c_2)\}\]

\[r = r_2, c = c_2\]

\[\text{m\text{-}fetch1}(r,c)\]

task: \text{fetch}(r,c)

pre: pos(c) = unknown

body:
- if \(\exists l\) (view(l) = F) then
  - move-to\((r,l)\)
- perceive\((r,l)\)
- if pos(c) = l then
  - take\((r,c,l)\)
- else
  - fetch\((r,c)\)

else

- fail

\[\text{m\text{-}fetch2}(r,c)\]

task: \text{fetch}(r,c)

pre: pos(c) ≠ unknown

body:
- if loc\((r)\) = pos\((c)\) then
  - take\((r,c,pos(c))\)
- else
  - move-to\((r,pos(c))\)
  - take\((r,c,pos(c))\)

m\text{-}fetch_1(r\_0,c\_2)\]

\[r_0 = r_2\]

\[\text{move-to}\((r_1,\text{loc1})\)\]

\[\text{perceive}\((r_1,\text{loc1})\)\]

\[\text{sensor failure}\]

\[\text{code execution}\]

\[\text{search tree}\]

\[\text{Candidates} = \{m\text{-}fetch(r_1,c_2), m\text{-}fetch(r_2,c_2)\}\]
m-fetch1(r,c) \[ r = r_2, c = c_2 \]

task: \( \text{fetch}(r,c) \)

pre: \( \text{pos}(c) = \text{unknown} \)

body:
- if \( \exists l \) (view(l) = F) then
  - move-to(r,l)
  - perceive(r,l)
- if pos(c) = l then
  - take(r,c,l)
- else fetch(r,c)
- else fail

m-fetch2(r,c)

task: \( \text{fetch}(r,c) \)

pre: \( \text{pos}(c) \neq \text{unknown} \)

body:
- if loc(r) = pos(c) then
  - take(r,c,pos(c))
- else do
  - move-to(r,pos(c))
  - take(r,c,pos(c))

Example

Candidates = \{m-fetch(r1,c2), m-fetch(r2,c2)\}

Is this the same as a backtracking search?

Progress(\(\sigma\)):
Extensions to Rae

- Methods for events
  - e.g., an emergency
- Methods for goals
  - special kind of task: achieve(goal)
  - sets up a monitor to see if the goal has been achieved
- Concurrent subtasks
Outline

- **Motivation**
- **Representation** – state variables, commands, tasks, refinement methods
- **Acting** – Rae (Refinement Acting Engine)
- **Planning** – UPOM (UCT-like Planner for Operational Models)
- **Acting with Planning** – Rae + UPOM
- **Using the implementation** – Rae code, UPOM code, examples
procedure Rae:
   loop:
      for every new external task or event \( \tau \) do
        choose a method instance \( m \) for \( \tau \)
        create a refinement stack for \( \tau, m \)
        add the stack to Agenda
      for each stack \( \sigma \) in Agenda
         Progress(\( \sigma \))
         if \( \sigma \) is finished then remove it
   ● Bad choice may lead to
     ▶ more costly solution
     ▶ failure, need to recover
     ▶ unrecoverable failure
   ● Idea: do simulations to predict outcomes

Why Plan?

Progress(\( \sigma \)):
- \( \text{started } m? \)
  - yes
    - \( \text{is } m\text{'s current step a command?} \)
      - yes
        - retry \( \tau \) using an untried candidate
      - failed
        - \( \text{return success} \)
        - \( \text{succeeded} \)
          - more steps in \( m? \)
            - yes
              - retry \( \tau \) using an untried candidate
            - no
              - pop(\( \sigma \))
          - no
            - \( \tau' \leftarrow \text{next step of } m \)

- \( \text{assignment} \)
  - \( \text{update state} \)
  - \( \text{candidates for } \tau'? \)
    - yes
      - \( \text{send } \tau' \text{ to the execution platform} \)
    - no
      - retry \( \tau \) using an untried candidate

- \( \text{command} \)
  - \( \text{type}(\tau') \)
  - \( \text{task} \)
  - \( \text{choose a candidate } m' \)
    - push \( (\tau', m', \ldots) \) onto \( \sigma \)
Planner

- Basic ideas
  - Repeated Monte Carlo rollouts on a single task \( t \)
  - Choose method instances using a UCT-like formula
  - Simulated execution of commands

UPOM(\( \tau \)):

- Choose a method instance \( m \) for \( \tau \)
- Create refinement stack \( \sigma \) for \( \tau \) and \( m \)
- Loop while Simulate-Progress(\( \sigma \)) \( \neq \) failure
  - If \( \sigma \) is completed then return \((m, \text{utility of outcome})\)
  - Return failure

UPOM-Lookahead (task \( \tau \)):

- Call UPOM(\( \tau \)) multiple times
- Return the \( m \in \text{Candidates} \) that has the highest average utility
Simulating a command

- Simplest case:
  - probabilistic action template
    \[ a(x_1, \ldots, x_k) \]
    \hspace{1cm} \begin{align*}
    & \text{pre: } \ldots \hspace{1cm} \text{eff}_i: \ e_{1i}, e_{12}, \ldots \\
    & \hspace{1cm} \ldots \hspace{1cm} \ldots \\
    & \hspace{1cm} \ldots \hspace{1cm} \text{eff}_m: \ e_{mi}, e_{m2}, \ldots
    \end{align*}
  - Choose randomly, each eff\(_i\) has probability \(p_i\)
  - Use eff\(_i\) to update the current state

- More general:
  - Arbitrary computation, e.g., physics-based simulation
  - Run the code to get prediction of effects

---

Simulate-Progress(\(\sigma\)):}

- \(\text{started } m?\)
  - yes
    - \(\text{is } m\text{'s current step a command?}\)
      - yes
        - return \(\text{success}\)
      - no
        - \(\text{running}\)
          - \(\text{simulation status?}\)
            - \(\text{failed}\)
              - retry \(\tau\) using an untried candidate
            - \(\text{succeeded}\)
              - more steps in \(m?\)
                - yes
                  - \(\tau' \leftarrow \text{next step of } m\)
                - no
                  - \(\text{pop}(\sigma)\)
          - \(\text{no}\)
            - \(\text{assignment}\)
              - \(\text{command}\)
                - \(\text{start simulation of } \tau'\)
                  - \(\text{candidates for } \tau'?\)
                    - yes
                      - choose a candidate \(m'\)
                        push \((\tau', m', \ldots)\) onto \(\sigma\)
                    - no
                      - retry \(\tau\) using an untried candidate
                  - \(\text{no}\)
                    - \(\text{update state}\)
                      - \(\text{task}\)
                        - \(\text{type}(\tau')\)
                          - \(\text{assignment}\)
                            - \(\text{command}\)
                              - \(\text{start simulation of } \tau'\)
                                - \(\text{candidates for } \tau'?\)
                                  - yes
                                    - choose a candidate \(m'\)
                                      push \((\tau', m', \ldots)\) onto \(\sigma\)
                                  - no
                                    - retry \(\tau\) using an untried candidate
Monte Carlo Rollouts

- Rollouts on MDPs
  - At each state, choose action at random, get random outcome
- UCT algorithm
  - Choice of action balances exploration vs exploitation
  - Converges to optimal choice at root of tree

- UPOM search tree more complicated
  - tasks, methods, commands, code execution
- If no exogenous events, can map it into UCT on a complicated MDP
  - proof of convergence to optimal
Outline

- Motivation

- **Representation** – state variables, commands, tasks, refinement methods

- **Acting** – Rae (Refinement Acting Engine)

- **Planning** – UPOM (UCT-like Planner for Operational Models)

- **Acting with Planning** – Rae + UPOM

- **Using the implementation** – Rae code, UPOM code, examples
procedure Rae:
  loop:
    for every new external task or event $\tau$ do
      choose a method instance $m$ for $\tau$
      create a refinement stack for $\tau$, $m$
      add the stack to Agenda
    for each stack $\sigma$ in Agenda
      Progress($\sigma$)
      if $\sigma$ is finished then remove it
  
  Whenever RAE needs to choose a method instance, use UPOM-Lookahead to make the choice
Summary of Experimental Results

- Five different domains, different combinations of characteristics
- Evaluation criteria:
  - Efficiency, successes vs failures, how many retries
- Result: planning helps
  - Rae operates better with UPOM than without
  - Rae operates better with more planning than with less planning
Other Details

- **Receding horizon**
  - Cut off search before accomplishing $\tau$
    - e.g., depth $d_{max}$ or when we run out of time
  - At leaf nodes, use heuristic function

- **Learning a heuristic function**
  - Supervised learning
Outline

1. Motivation

2. Representation – state variables, commands, tasks, refinement methods

3. Acting – Rae (Refinement Acting Engine)

4. Planning – UPOM (UCT-like Planner for Operational Models)

5. Acting with Planning – Rae + UPOM

6. Using the implementation – Rae code, UPOM code, examples
Code Demo

- Github repository: [https://github.com/sunandita/ICAPS_Summer_School_RAE_2020](https://github.com/sunandita/ICAPS_Summer_School_RAE_2020)

- System requirements:
  - Unix based operating system preferred
  - Have Docker or the Python Conda environment preinstalled

- Things to play with:
  - Domain file: ICAPS_Summer_School_RAE_2020/domains/domain_\(x\).py
  - Problem file: ICAPS_Summer_School_RAE_2020/problems/\(x\)/problemId_\(x\).py
  - \(x\) ∈ \{chargeableRobot, explorableEnv, searchAndRescue, springDoor, orderFulfillment\}

- How to run?
  - cd ICAPS_Summer_School_RAE_2020/RAE_and_UPOM
  - python3 testRAEandUPOM.py –h