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

Tasks:
- manage incoming shipment
  - unload
  - unpack
  - store
  - await order
  - prepare
  - deliver

Planning

Acting
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
    - Fetch o7
    - Navigate to room2
    - Deliver o7
  - ... (other tasks)

- Acting
  - Move to door
  - Open door
  - Get out
  - Close door
  - ... (other actions)

- Identify type of door
- Move close to knob
- Grasp knob
- Turn knob
- Maintain knob
- Move back
- Ungrasp
- Monitor
- Pull
- ... (other actions)
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
- pull
- move back
- ungrasp
- 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?
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, …

Identify type of door

- Move close to knob
- Grasp knob
- Turn knob
- Maintain
- Pull
- Monitor
- Move back
- Ungrasp

Open door

- Get out
- Close door

Respond to user requests

- Bring o7 to room2
- Go to hallway
- Deliver o7
- Move to door
- Fetch o7
- Navigate to room2
- Navigate to room1
- Identify type of door
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?

![Door Types](image.png)

![Door Accessories](image2.png)
**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
- \( Robots = \{r_1, r_2\} \)
- \( Containers = \{c_1, c_2\} \)
- \( Locations = \{loc_1, loc_2, loc_3, loc_4\} \)

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 \( r \in Robots, c \in Containers, l \in Locations \)
  - \( loc(r) \in Locations \)
  - \( cargo(r) \in Containers \cup \{\text{empty}\} \)
  - \( pos(c) \in Locations \cup Robots \cup \{\text{unknown}\} \)
  - \( view(l) \in \{T, F\} \)

  - Whether a robot has looked at location \( l \)
  - If \( view(l) = T \) then \( pos(c) = l \) for every container \( c \) at \( l \)

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

- **Task**: an activity for the actor to perform
  - `taskname(arg_1, …, arg_k)`

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

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

```
command
  m-fetch1(r,c)
  task: fetch(r,c)
  pre: pos(c) = 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
```

```
command
  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))
```

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
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 ∈ Robots, c ∈ Containers, l ∈ Locations
- loc(r) ∈ Locations
- cargo(r) ∈ Containers ∪ \{nil\}
- pos(c) ∈ Locations ∪ Robots ∪ \{unknown\}
- view(l) ∈ \{T, F\}
  - Whether a robot has looked at location l
  - If view(l) = T then 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

Search tree
$
\text{fetch}(r_0,c2)$
$
\tau$

Example

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

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($r,c,pos(c)$)
m-fetch1(r, c) \( r = r_0, c = c_2 \)

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

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

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

m-fetch2(r, c)

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

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

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

---

**Example**

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

---

**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 \(\text{Agenda}\)
- for each stack \(\sigma\) in \(\text{Agenda}\)
  - \(\text{Progress}(\sigma)\)
  - if \(\sigma\) is finished then remove it

---

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

---

**Search tree**

\(\text{fetch}(r_0, c_2)\)

---

\(\tau\)
m-fetch1(r,c)  \( r = r_1, c = c_2 \)
\begin{itemize}
  \item task: fetch(r,c)
  \item pre: pos(c) = unknown
  \item body:
    \begin{itemize}
      \item if \( \exists l \) (view(l) = F) then
        \begin{itemize}
          \item move-to(r,l)
        \end{itemize}
      \end{itemize}
      \begin{itemize}
        \item perceive(r,l)
        \end{itemize}
      \begin{itemize}
        \item if pos(c) = l then
          \begin{itemize}
            \item take(r,c,l)
          \end{itemize}
        \end{itemize}
      \begin{itemize}
        \item else fetch(r,c)
      \end{itemize}
    \end{itemize}
  \item else fail
\end{itemize}

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

Example

\begin{itemize}
  \item Container locations unknown
  \item Partially observable
    \begin{itemize}
      \item Robot only sees current location
    \end{itemize}
\end{itemize}

procedure Rae:
\begin{itemize}
  \item loop:
    \begin{itemize}
      \item for every new external task or event \( \tau \) do
        \begin{itemize}
          \item choose a method instance \( m \) for \( \tau \)
          \item create a refinement stack for \( \tau, m \)
          \item add the stack to \( \text{Agenda} \)
        \end{itemize}
    \end{itemize}
  \item for each stack \( \sigma \) in \( \text{Agenda} \)
    \begin{itemize}
      \item Progress(\( \sigma \))
      \item if \( \sigma \) is finished then remove it
    \end{itemize}
\end{itemize}
m-fetch1\((r,c)\)  \(r = r1, c = c2\)

- **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
      - search\((r,c)\)
  - else
    - fail

---

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
    - move-to\((r,\text{pos}(c))\)
    - take\((r,c,\text{pos}(c))\)

---

**Candidates**

\[\{m\text{-}fetch(r1,c2), m\text{-}fetch(r2,c2)\}\]

---

**Example**

**Search tree**

- **m-fetch1\((r1,c2)\)**
- **Candidates**:
  - \(\{m\text{-}fetch(r1,c2), m\text{-}fetch(r2,c2)\}\)

---

**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

---

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

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

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

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

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

\[
\text{m-fetch2}(r,c) \quad r = r_1, c = c_2
\]

- task: \text{fetch}(r,c)
- pre: \text{pos}(c) \neq \text{unknown}
- body:
  - if \text{loc}(r) = \text{pos}(c) \text{ then}
    - take\((r,c,\text{pos}(c))\)
  - else do
    - move-to\((r,\text{pos}(c))\)
    - take\((r,c,\text{pos}(c))\)
m-fetch1(r,c) \quad r = r1, c = c2

- 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

Search tree

**Progress(σ):**

- started m?
  - yes
  - is m’s current step a command?
    - yes
    - command status?
      - success
      - more steps in m?
        - yes
        - continue
        - no
        - failed
        - retry τ using an untried candidate
    - no
    - return
  - no

- pop(σ)
  - assignment
  - type(τ’)
  - command
    - candidates for τ’?
      - yes
      - send τ’ to the execution platform
      - no
      - no
    - choose a candidate m’
    - push (τ’, m’,...) onto σ
    - retry τ using an untried candidate

- search tree
  - τ = m-fetch1(r0,c2)
  - r0 = r1
  - τ’ = next step of m

- code execution

- move-to(r1,loc1)
  - \tau’

Nau and Patra – ICAPS 2020 Summer School
Example

Search tree

progress(σ):

no

yes

started m?

is m’s current step a command?

running

command status?

succeeded

more steps in m?

yes

no

return

code execution

failure?

try τ using an untried candidate

pop(σ)

no candidates for τ’?

yes

no

assignmnet

type(τ’)

command

update

state

task

send τ’ to execution platform

choose a candidate m’

push (τ’, m’, …) onto σ

retry τ using an untried candidate

...
Example

\[ m\text{-}fetch1(r_1, c_2) \]

\textbf{task: } fetch(r_1, c_2)

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

\textbf{body: }

\begin{align*}
&l = \text{loc}_1 \\
&\text{if } \exists l \text{ (view}(l) = F) \text{ then} \\
&\text{move-to}(r_1, l) \\
&\text{perceive}(r_1, l) \\
&\text{if } \text{pos}(c) = l \text{ then} \\
&\text{take}(r_1, c_1, l) \\
&\text{else } \text{fetch}(r_1, c_2) \\
&\text{else } \text{fail}
\end{align*}

\[ m\text{-}fetch2(r_1, c_2) \]

\textbf{task: } fetch(r_1, c_2)

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

\textbf{body: }

\begin{align*}
&\text{if } \text{loc}(r_1) = \text{pos}(c) \text{ then} \\
&\text{take}(r_1, c_1, \text{pos}(c)) \\
&\text{else } \text{do} \\
&\text{move-to}(r_1, \text{pos}(c)) \\
&\text{take}(r_1, c_1, \text{pos}(c))
\end{align*}

\textbf{procedure Rae:}

\textbf{loop: }

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

\(\text{m-fetch1}(r, c)\) \hspace{1cm} r = r1, c = c2

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

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

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

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

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

- \(\text{type}(\tau’)\) command
  - send \(\tau’\) to the execution platform
  - candidates for \(\tau’?\)
    - yes:
      - choose a candidate \(m’\)
      - push \((\tau’ , m’, \ldots)\) onto \(\sigma\)
    - no:
      - update state
- assignment
- task
- send \(\tau’\) to the execution platform
- candidates for \(\tau’?\)
Example

Progress($\sigma$):

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

$\tau'$ ← next step of $m$

Assignment:

- type($\tau'$)
- command

Send $\tau'$ to the execution platform

Candidates for $\tau'$?

- yes
  - update state
  - task
    - type($\tau'$)
      - command
  - send $\tau'$ to the execution platform
- no
  - choose a candidate $m'$
    - push ($\tau'$, $m'$,...) onto $\sigma$
Example

\[
\begin{align*}
\text{m-fetch1}(r,c) & : r = r1, c = c2 \\
\text{task: } & \text{fetch}(r,c) \\
\text{pre: } & \text{pos}(c) = \text{unknown} \\
\text{body: } & \begin{align*}
& l = \text{loc1} \\
& \text{if } \exists l \ (\text{view}(l) = F) \text{ then} \\
& \text{move-to}(r,l) \\
& \text{perceive}(r,l) \\
& \text{if } \text{pos}(c) = l \text{ then} \\
& \text{take}(r,c,l) \\
& \text{else fetch}(r,c) \\
& \text{else fail}
\end{align*}
\end{align*}
\]

m-fetch2(r,c)

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

Progress(σ):

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

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

- **Task:** fetch\((r,c)\)
- **Pre:** pos\((c) = \text{unknown} \)
- **Body:**
  \[
  l = \text{loc1}
  \]
  \[
  \text{if } \exists l \ (\text{view}(l) = F) \text{ then }
  \]
  \[
  \text{move-to}(r,l)
  \]
  \[
  \text{perceive}(r,l)
  \]
  \[
  \text{if } \text{pos}(c) = l \text{ then }
  \]
  \[
  \text{take}(r,c,l)
  \]
  \[
  \text{else fail}
  \]

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

- **Task:** fetch\((r,c)\)
- **Pre:** pos\((c) \neq \text{unknown} \)
- **Body:**
  \[
  \text{if } \text{loc}(r) = \text{pos}(c) \text{ then }
  \]
  \[
  \text{take}(r,c,\text{pos}(c))
  \]
  \[
  \text{else do}
  \]
  \[
  \text{move-to}(r,\text{pos}(c))
  \]
  \[
  \text{take}(r,c,\text{pos}(c))
  \]

\[ \text{r} = r_0 = r_1 \]

\[ \text{sensor failure} \]

\[ \text{Search tree} \]

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

- **No**
  \[
  \text{m's current step a command?}
  \]
  \[
  \text{Yes} \]
  \[
  \text{return success}
  \]
  \[
  \text{more steps in } m? \]
  \[
  \text{Yes} \]
  \[
  \text{retry } \tau \text{ using an untried candidate}
  \]
  \[
  \text{pop}(\sigma)
  \]
  \[
  \text{assignment}
  \]
  \[
  \text{type}(\tau')
  \]
  \[
  \text{command}
  \]
  \[
  \text{candidates for } \tau'?
  \]
  \[
  \text{Yes} \]
  \[
  \text{update state}
  \]
  \[
  \text{send } \tau' \text{ to the execution platform}
  \]
  \[
  \text{no}
  \]
  \[
  \text{retry } \tau \text{ using an untried candidate}
  \]
  \[
  \text{no}
  \]
  \[
  \text{no more steps in } m\?
  \]
  \[
  \text{Yes} \]
  \[
  \text{push } (\tau', m', \ldots) \text{ onto } \sigma
  \]
  \[
  \text{no}
  \]
  \[
  \text{no candidates for } \tau'?
  \]
  \[
  \text{Yes}
  \]
  \[
  \text{type}(\tau')
  \]
  \[
  \text{command}
  \]
  \[
  \text{candidates for } \tau'?
  \]
  \[
  \text{Yes}
  \]
  \[
  \text{update state}
  \]
  \[
  \text{send } \tau' \text{ to the execution platform}
  \]
  \[
  \text{no}
  \]
  \[
  \text{no}
  \]
  \[
  \text{no}
  \]
m-fetch1(r, c)  \( r = r_2, c = c_2 \)

<table>
<thead>
<tr>
<th>task:</th>
<th>fetch(r, c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre:</td>
<td>pos(c) = unknown</td>
</tr>
<tr>
<td>body:</td>
<td>if ( \exists l ) (view(l) = F) then move-to(r, l) perceive(r, l) if pos(c) = l then take(r, c, l) else fail</td>
</tr>
</tbody>
</table>

m-fetch2(r, c)

<table>
<thead>
<tr>
<th>task:</th>
<th>fetch(r, c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre:</td>
<td>pos(c) ( \neq ) unknown</td>
</tr>
<tr>
<td>body:</td>
<td>if loc(r) = pos(c) then take(r, c, pos(c)) else do move-to(r, pos(c)) take(r, c, pos(c))</td>
</tr>
</tbody>
</table>

Example

\[ m\text{-}fetch1(r_1, c_2) \]

| \( r = r_2, c = c_2 \) |

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

Progress(\( \sigma \)):

- started \( m \)?
- yes
- m’s current step a command?
- yes
- command status?
- succeeded
- yes
- more steps in \( m \)?
- no
- return success
- no
- \( \tau \) ← next step of \( m \)

Assignment:

| \( \tau \) ’ ← type(\( \tau \) ’)
| command |
| send \( \tau \) ’ to the execution platform |
| candidates for \( \tau \) ’? |
| yes |
| choose a candidate \( m \)’ push (\( \tau \) ’, \( m \)’,…) onto \( \sigma \) |
| no |
| retry \( \tau \) using a tried candidate |

Retry:

| retry \( \tau \) using an untried candidate |

Fail:

| no more steps in \( m \)? |
| yes |
| no |
| \( \tau \) ← next step of \( m \) |

Sensor failure:

| r_0 = r_2 |

Search tree

- \( \tau \)
- retry

Code execution

- move-to(r1, loc1)
- perceive(r1, loc1)
Example

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

Sensor failure

Is this the same as a backtracking search?
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
Why Plan?

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

Progress($\sigma$):

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

- assignment
- type($\tau'$)
- update state
- task
- send $\tau'$ to the execution platform
- candidates for $\tau'$?
  - yes
    - choose a candidate $m'$
      - push ($\tau'$, $m'$,...) onto $\sigma$
  - no
    - retry $\tau$ using an untried candidate

- succeeded
- no more steps in $m$?
Planner

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

\[
\text{UPOM}(\tau):
\begin{align*}
&\text{choose a method instance } m \text{ for } \tau \\
&\text{create refinement stack } \sigma \text{ for } \tau \text{ and } m \\
&\text{loop while Simulate-Progress}(\sigma) \neq \text{failure} \\
&\quad \text{if } \sigma \text{ is completed then return } (m, \text{utility of outcome}) \\
&\text{return failure}
\end{align*}
\]

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

- Simplest case:
  - probabilistic action template
    \[
    a(x_1, \ldots, x_k)
    \]
    pre: …
    \[
    (p_1) \text{ eff}_1: e_{11}, e_{12}, \ldots
    \]
    …
    \[
    (p_m) \text{ eff}_m: e_{m1}, e_{m2}, \ldots
    \]
  - Choose randomly, each \text{ eff}_i has probability \( p_i \)
  - Use \text{ eff}_i to update the current state

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

---

**Monte Carlo Rollouts**

- Rollouts on MDPs
- UCT algorithm
- Converges to optimal choice at root of tree

---

**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
RAE + UPOM

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

| Domain | $|\mathcal{T}|$ | $|\mathcal{M}|$ | $|\overline{\mathcal{M}}|$ | $|\mathcal{A}|$ | Dynamic events | Dead ends | Sensing | Robot collaboration | Concurrent tasks |
|--------|---------|---------|----------|---------|---------------|-----------|---------|-------------------|-----------------|
| S&R    | 8       | 16      | 16       | 14      | ✓             | ✓         | ✓       | ✓                 | ✓               |
| Explore| 9       | 17      | 17       | 14      | ✓             | ✓         | ✓       | ✓                 | ✓               |
| Fetch  | 7       | 10      | 10       | 9       | ✓             | ✓         | ✓       | –                 | ✓               |
| Nav    | 6       | 9       | 15       | 10      | ✓             | –         | ✓       | ✓                 | ✓               |
| Deliver| 6       | 6       | 50       | 9       | ✓             | ✓         | –       | ✓                 | ✓               |

- 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`