Chapter 3
Deliberation with Refinement Methods

Dana S. Nau
University of Maryland
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*
  - Dynamic, unpredictable, partially observable environment
    - Adapt to context, react to events
  - Uses *operational* models
    - instructions telling *how* to perform the tasks
    - usually hierarchical
Outline

1. Motivation
2. Representation
3. Acting (Rae)
4. Planning for Rae
5. Acting with Planning (RAE+UPOM)
6. Learning
7. Evaluation, Application
Example

- Consider an actor that controls two robots
- Environment is partially observable
  - Each robot can only see what’s at the current location

- Objects
  - $Robots = \{r_1, r_2\}$
  - $Containers = \{c_1, c_2\}$
  - $Locations = \{loc0, loc1, loc2, loc3, loc4\}$

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

- Commands to the execution platform:
  - `take(r,o,l)`: robot `r` takes object `o` at location `l`
  - `put(r,o,l)`: robot `r` puts object `o` at location `l`
  - `perceive(r,l)`: robot `r` perceives what objects are at location `l`
  - `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**: a program
  - `method-name(arg_1, ..., arg_k)`
  - **task**: `task-identifier`
  - **pre**: `test`
  - **body**: `a program`

- Assignment statements
- Control constructs:
  - if-then-else, while, ...
- Tasks
  - can extend this to include events, goals
- Commands to the execution platform

```
m-fetch1(r,c)
  task: `fetch(r,c)`
  pre: `pos(c) = unknown`
  body: `if ∃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))`
```

command
task
Outline

1. Motivation
2. Representation
3. *Acting* (Rae)
4. Planning for Rae
5. Acting with Planning (RAE+UPOM)
6. Learning
7. Evaluation, Application
**RAE (Refinement Acting Engine)**

- Performs multiple tasks in parallel
  - Purely reactive, no lookahead
- For each task or event \( \tau \), a *refinement stack*
  - execution stack
- *Agenda* = {all current refinement stacks}

\[
\begin{align*}
\text{Agenda} &= \{\tau_1, \tau_2, \tau_3\} \\
\text{Refinement stack for a task } \tau &\iff \text{current path in RAE’s search tree for } \tau
\end{align*}
\]

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*
    call Progress(\( \sigma \))
    if \( \sigma \) is finished then remove it
```
Example (reminder)

- 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 $r \in Robots$, $c \in Containers$, $l \in Locations$
  - $loc(r) \in Locations$
  - $cargo(r) \in Containers \cup \{nil\}$
  - $pos(c) \in Locations \cup Robots \cup \{unknown\}$
  - $view(l) \in \{T, F\}$
    - 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:
  - $take(r, o, l)$: $r$ 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 $l$
### Example

**Refinement tree**

```
fetch(r_0,c2) \\
\tau
```

**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`

**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

---

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

m-fetch1(r, c)

\( r = r_0, c = c_2 \)

- 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 \( \text{pos}(c) = l \) then
      - take(r, c, l)
    - else fetch(r, c)
  - else fail

Candidates

\( \{ \text{m-fetch1}(r_1, c_2), \text{m-fetch1}(r_2, c_2) \} \)

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
    - take(r, c, pos(c))
  - else do
    - move-to(r, pos(c))
    - take(r, c, 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
  call Progress(\( \sigma \))
  if \( \sigma \) is finished then remove it
Example

m-fetch1\((r,c)\)  \(r = r_1, c = c_2\)

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

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) ≠ 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-fetch(r_1,c_2), m-fetch(r_2,c_2) \}

refinement tree

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

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

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

- **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

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

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

- **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\((r,c,\text{pos}(c))\)

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

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

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

\[ m\text{-}\text{fetch1}(r_1,c_2) \]
\[ r_0 = r_1 \]
\[ r_0 = r_1 \]
\[ m \]

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

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

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

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

<table>
<thead>
<tr>
<th>Progress(σ):</th>
<th>(τ, m, i, tried) ← top(σ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>started m?</td>
<td>yes</td>
</tr>
<tr>
<td>m’s current step a command?</td>
<td>no</td>
</tr>
<tr>
<td>command status?</td>
<td>yes</td>
</tr>
<tr>
<td>succeeded</td>
<td>running</td>
</tr>
<tr>
<td>no more steps in m?</td>
<td>no</td>
</tr>
<tr>
<td>yes</td>
<td>τ' ← next step of m</td>
</tr>
<tr>
<td>no</td>
<td>pop(σ)</td>
</tr>
<tr>
<td>yes</td>
<td>retry τ using an untried candidate</td>
</tr>
<tr>
<td>no</td>
<td>retry τ using an untried candidate</td>
</tr>
<tr>
<td>yes</td>
<td>succeeded</td>
</tr>
<tr>
<td>no</td>
<td>failed</td>
</tr>
</tbody>
</table>

- **Refinement tree**
- **Progress(σ)**
- **Assignment**
- **Command**
- **Task**
- **Send τ' to the execution platform**
- **Choose a candidate m'**
- **Push (τ', m', ...) onto σ**
- **Retry τ using an untried candidate**
Example

**Refinement 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:**
  - \( l = \text{loc} 1 \)
  - 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))

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

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

- no more steps in \( m \)?
  - yes
    - \( \tau' \) ← next step of \( m \)
  - no
    - pop(\( \sigma \))

**Assignment**

- type(\( \tau' \))
  - task
    - candidates for \( \tau' \)?
      - yes
        - send \( \tau' \) to the execution platform
      - no
        - retry \( \tau \) using an untried candidate
  - no

**Update state**

- choose a candidate \( m' \)
  - push (\( \tau', m', \ldots \)) onto \( \sigma \)
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
  call Progress($\sigma$)
  if $\sigma$ is finished then remove it

Example

Example:

$m$-fetch1($r,c$) $r = r_1$, $c = c_2$

  task: fetch($r,c$)
  pre: pos($c$) = unknown
  body:
    $l = l_0$
    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($r,c,pos(c)$)

Refinement tree

Progress($\sigma$):

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

Refinement tree

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

- task: fetch(r,c)
- pre: pos(c) = unknown
- body:
  - $l = loc1$
  - 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) $\neq$ unknown
- body:
  - if loc($r) = pos(c)$ then
  - take($r,c,\text{pos}(c)$)
  - else do
  - move-to($r;\text{pos}(c)$)
  - take($r,c,\text{pos}(c)$)

Progress($\sigma$): $(\tau,m,i,\text{tried}) \leftarrow \text{top}(\sigma)$

- started $m$?
- yes
  - $\tau$ is $m$’s current step a command?
  - yes
    - return
  - no
    - failed
      - retry $\tau$ using an untried candidate
      - yes
        - $\tau' \leftarrow \text{next step of } m$
        - $\text{more steps in } m$?
          - yes
            - $\text{more steps in } m$?
              - yes
                - $\tau' \leftarrow \text{next step of } m$
                - $\text{command status?}$
                  - succeeded
                    - $\text{more steps in } m$?
                      - yes
                        - $\text{more steps in } m$?
                          - yes
                            - $\text{more steps in } m$?
                              - yes
                                - $\text{more steps in } m$?
                                  - yes
                                    - $\text{more steps in } m$?
                                      - yes
                                        - $\text{more steps in } m$?
                                          - yes
                                            - $\text{more steps in } m$?
                                              - yes
                                                - $\text{more steps in } m$?
                                                  - yes
                                                    - $\text{more steps in } m$?
                                                      - yes
                                                        - $\text{more steps in } m$?
                                                          - yes
                                                            - $\text{more steps in } m$?
                                                              - yes
                                                                - $\text{more steps in } m$?
                                                                  - yes
                                                                    - $\text{more steps in } m$?
                                                                      - yes
                                                                        - $\text{more steps in } m$?
                                                                          - yes
                                                                            - $\text{more steps in } m$?
                                                                              - yes
                                                                                - $\text{more steps in } m$?
                                                                                  - yes
                                                                                    - $\text{more steps in } m$?
                                                                                      - yes
                                                                                       - $\text{more steps in } m$?
                                                                                        - yes
                                                                                         - $\text{more steps in } m$?
                                                                                           - yes
                                                                                            - $\text{more steps in } m$?
                                                                                         - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
                                                                                           - yes
                                                                                           - $\text{more steps in } m$?
Example

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

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

Progress$(\sigma)$:

- $(\tau,m,i,\text{tried}) \leftarrow \text{top}(\sigma)$
- $\text{started } m?$
  - *yes*
    - $m$’s current step a command?
      - *yes*
        - command status?
          - *reached*
          - $\text{return success}$
          - *no*
          - $\text{retry } \tau \text{ using an untried candidate}$
        - *yes*
          - $\text{more steps in } m?$
            - *no*
              - $\text{elasticity}$
            - *yes*
              - $\tau'$ $\leftarrow$ next step of $m$
                - $\text{type}(\tau')$
                  - *task*
                    - $\text{assignment}$
                      - *update state } s$
                        - $\text{candidates}$
                          - *for } \tau'$?
                            - *yes*
                              - $\text{send } \tau' \text{ to the execution platform}$
                                - *no*
                                - $\text{retry } \tau \text{ using an untried candidate}$
Example

Refinement tree

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

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

\[
\text{m-fetch2}(r,c) \quad r = r1, c = c2
\]

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

Progress(\(\sigma\)):

- \((\tau,m,i,\text{tried})\leftarrow \text{top}(\sigma)\)
- \(\text{started } m?\)
  - yes
  - \(m\text{'s current step a command?}\)
  - yes
    - return \(\text{success}\)
    - \(m\text{'s more steps in } m?\)
      - yes
        - \(\tau' \leftarrow \text{next step of } m\)
        - \(\text{command}\)
          - type(\(\tau'\))
            - task
              - send \(\tau'\) to the execution platform
        - \(\text{assignment}\)
          - update state \(s\)
            - \(\text{candidates for } \tau'?\)
              - yes
                - choose a candidate \(m'\)
                  - push \((\tau',m',\ldots)\) onto \(\sigma\)
              - no
                - retry \(\tau\) using an untried candidate
        - \(\text{retry } \tau\) using an untried candidate
      - no
        - \(\text{pop}(\sigma)\)
        - \(\text{no more steps in } m?\)
          - yes
            - \(\tau' \leftarrow \text{next step of } m\)
          - no
            - \(\text{failed}\)
        - \(\text{return } \text{failed}\)
    - \(\text{retry } \tau\) using an untried candidate
  - no
    - \(\text{no candidates for } \tau'\)
m-fetch1(r,c)  \( r = r1, c = c2 \)
task:  \( \text{fetch}(r,c) \)
pre:  \( \text{pos}(c) = \text{unknown} \)
body:  
\[
\begin{align*}
& l = \text{loc1} \\
& \text{if } \exists l \ (\text{view}(l) = F) \text{ then} \\
& \quad \text{move-to}(r,l) \\
& \quad \text{perceive}(r,l) \\
& \quad \text{if } \text{pos}(c) = l \text{ then} \\
& \quad \quad \text{take}(r,c,l) \\
& \quad \text{else } \text{fetch}(r,c) \\
& \text{else } \text{fail}
\end{align*}
\]

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

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

m\text{-}fetch1(r,c) \quad r = r2, \ c = c2
task: \text{fetch}(r,c)
pre: pos(c) = \text{unknown}
body:
  if \exists l \text{ (view}(l) = F) \text{ then}
    \text{move\text{-}to}(r,l)
  \text{perceive}(r,l)
  if pos(c) = l \text{ then}
    \text{take}(r,c,l)
  \text{else fetch}(r,c)
else fail

m\text{-}fetch2(r,c) \quad task: \text{fetch}(r,c)
pre: pos(c) \neq \text{unknown}
body:
  if loc(r) = pos(c) \text{ then}
    \text{take}(r,c,pos(c))
  \text{else do}
    \text{move\text{-}to}(r,pos(c))
    \text{take}(r,c,pos(c))
m-fetch1(r, c)  \( r = r_2, c = c_2 \)

**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**

**Candidates**

= \{ m-fetch1(r_1, c_2), m-fetch(r_2, c_2) \}

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

(\( \tau, m, i, tried \) ) \( \leftarrow \) top(\( \sigma \))

---

**Poll:** 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

1. Motivation
2. Representation
3. Acting (Rae)
4. *Planning for Rae*
5. Acting with Planning (RAE+UPOM)
6. Learning
7. Evaluation, Application
Planning for Rae?

Four places where Rae and Progress choose a method instance

- For a task
- Bad choice may lead to more costly solution
- Failure - need to recover, sometimes unrecoverable
- Bad choice may lead to failure - need to recover, sometimes unrecoverable

Solution:

- Call a planner, choose the method instance it suggests
- BAD CHOICE: Call a planner, choose the method instance it suggests

For every new external task or event τ do

- For each stack σ in Agenda
- Create a refinement stack for τ, m

Loop:

Procedure RAE:

Progress(σ):

\( (\omega, l, re\text{fined}) \rightarrow \text{top}(\omega) \)
Planning and Acting Integration

- Planner’s action models are abstractions
  - The planned actions are tasks for the actor to refine
- Consistency problem:
  - How to get action models that describe what the actor will do?
- One possible solution:
  - Actor and planner both use the same representation
    - Must be operational; descriptive models too abstract
    - Need planning algorithms that can use operational models
Planning and Acting Integration

- Planner’s action models are abstractions
  - The planned actions are tasks for the actor to refine
- Consistency problem:
  - How to get action models that describe what the actor will do?
- One possible solution:
  - Actor and planner both use the same representation
    - Must be operational; descriptive models too abstract
    - Need planning algorithms that can use operational models

- Idea 1:
  - Planner uses Rae’s tasks and refinement methods
  - For each of Rae’s commands, have a classical action model
  - DFS or GBFS search among alternatives to see which works best
SeRPE (Sequential Refinement Planning Engine)

\[ M = \{ \text{methods} \} \]
\[ A = \{ \text{action models} \} \]
\[ s = \text{initial state} \]
\[ \tau = \text{task or goal} \]

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

\[ \text{Progress-to-finish}(M, A, s, \tau, m) \]
\[ i \leftarrow \text{nil} \quad \text{// instruction pointer for body}(m) \]
\[ \pi \leftarrow \emptyset \quad \text{// plan produced from body}(m) \]
\[ \text{loop} \]
\[ \text{if } \tau \text{ is a goal and } s \models \tau \text{ then return } \pi \]
\[ \text{if } i \text{ is the last step of } m \text{ then} \]
\[ \text{if } \tau \text{ is a goal and } s \not\models \tau \text{ then return failure} \]
\[ \text{return } \pi \]
\[ i \leftarrow \text{nextstep}(m, i) \]
\[ \text{case type}(m[i]) \]
\[ \text{assignment: update } s \text{ according to } m[i] \]
\[ \text{command:} \]
\[ a \leftarrow \text{the descriptive model of } m[i] \text{ in } A \]
\[ \text{if } s \models \text{pre}(a) \text{ then} \]
\[ s \leftarrow \gamma(s, a); \quad \pi \leftarrow \pi.a \]
\[ \text{else return failure} \]
\[ \text{task or goal:} \]
\[ \pi' \leftarrow \text{SeRPE}(M, A, s, m[i]) \]
\[ \text{if } \pi' = \text{failure then return failure} \]
\[ s \leftarrow \gamma(s, \pi'); \quad \pi \leftarrow \pi.\pi' \]

- Like Rae with just one external task
  - Progress it all the way to the end, like Progress with a loop around it
  - Plan rather than act
    - For each command, apply a classical action model
- But SeRPE there are problems …
**Problems with SeRPE**

*Problem 1*: difficult to implement

- Each time a method invokes a subtask, SeRPE makes a nondeterministic choice
- To implement deterministically
  - Each path in the search space is an execution trace of the body of a method
  - Need to backtrack over code execution
- Need to write a compiler that can do backtracking
  - Is it worth the effort?

Example:
- Suppose that
  - Each task has two applicable methods
  - When $i=2$, the 1st method for $\text{baz}(2)$ fails
- Backtracking:
  - Try 2nd method for $\text{baz}(2)$
  - If it fails, try 2nd method for $\text{bar}(2)$
  - If it fails, backtrack to $i = 1$
    - Try 2nd method for $\text{baz}(1)$
    - If it fails, try 2nd method for $\text{bar}(1)$
    - If it fails, backtrack to task $\text{foo}(k)$
**Problems with SeRPE**

- *Problem 2*: limitations of classical action models
  - e.g., the *fetch* example
  - We don’t know in advance what perceive’s effects will be
    - If we did, perceive wouldn’t actually be needed

```
take(r,o,l)
// robot r takes object o at location l
pre: cargo(r) = nil, loc(r) = l, loc(o) = l
eff: cargo(r) ← o, loc(o) ← r

put(r,o,l)
// r puts o at location l
pre: loc(r) = l, loc(o) = r
eff: cargo(r) ← nil, loc(o) ← l

perceive(r,l):
// robot r sees what objects are at l
pre: loc(r) = l
eff: ?
```
Planning for Rae

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
      call Progress(\( \sigma \))
      if \( \sigma \) is finished then remove it

- Idea 2: simulation with multithreading or multiprocessing
  - Run Rae in simulated environment
    - Simulate the commands (see next page)
    - To choose among method instances, try all of them in parallel
- Planner returns the method instance \( m \) having the highest expected utility (\( \approx \) least expected cost)

Poll: is this a reasonable approach?
Simulating commands

- Simplest case:
  - probabilistic action template
    
    \[ a(x_1, \ldots, x_k) \]
    
    \( \text{pre: } \ldots \)
    
    \( (p_1) \text{ effects}_1: e_{11}, e_{12}, \ldots \)
    
    \( \ldots \)
    
    \( (p_m) \text{ effects}_m: e_{m1}, e_{m2}, \ldots \)
  
  - Choose effects\(_i\) at random with probability \( p_i \)
    and use it to update the current state

- More general:
  - Arbitrary computation, e.g., physics-based simulation
  - Run the code to get simulated effects
Planning for Rae

procedure RAE:
    loop:
        for every new external task or event τ do
            choose a method instance m for τ
            create a refinement stack for τ, m
            add the stack to Agenda
        for each stack σ in Agenda
            call Progress(σ)
            if σ is finished then remove it

- Idea 3: simulation with Monte Carlo rollouts
  - Multiple runs
    - Random choices and outcomes in each run
  - Maintain statistics to estimate each choice’s expected utility
  - Return the method instance m that has the highest estimated utility

Plan-with-UPOM (task $\tau$):

1. $Candidates \leftarrow \{\text{method instances relevant for } \tau\}$
2. for $i \leftarrow 1$ to $n$
   - call UPOM($\tau$)
     - update estimates of methods’ expected utility
3. return the $m \in Candidates$ that has
   - the highest estimated utility

UPOM($\tau$):

1. choose a method instance $m$ for $\tau$
2. create refinement stack $\sigma$ for $\tau$ and $m$
3. loop while $\text{Simulate-Progress}(\sigma) \neq \text{failure}$
   - if $\sigma$ is completed then return $(m, \text{utility})$
4. return $\text{failure}$

- Each call to UPOM does a Monte Carlo rollout
  - Simulated execution of RAE on $\tau$
Monte-Carlo rollouts

Plan-with-UPOM (task $\tau$):

- $Candidates \leftarrow \{\text{method instances relevant for } \tau\}$
- for $i \leftarrow 1$ to $n$
  - call UPOM($\tau$)
    - update estimates of methods’ expected utility
- return the $m \in Candidates$ that has
  - the highest estimated utility

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 ($m$, utility)
- return failure

- Each call to UPOM does a Monte Carlo rollout
  - Simulated execution of RAE on $\tau$

Plan-with-UPOM:

1. $Candidates \leftarrow \{\text{method instances relevant for } \tau\}$
2. for $i \leftarrow 1$ to $n$
   - call UPOM($\tau$)
     - update estimates of methods’ expected utility
3. return the $m \in Candidates$ that has
   - the highest estimated utility

UPOM($\tau$):

1. choose a method instance $m$ for $\tau$
2. create refinement stack $\sigma$ for $\tau$ and $m$
3. loop while Simulate-Progress($\sigma$) $\neq$ failure
   - if $\sigma$ is completed then return ($m$, utility)
4. return failure

- Each call to UPOM does a Monte Carlo rollout
  - Simulated execution of RAE on $\tau$
Digression: Monte Carlo rollouts

- Multi-arm bandit problem
  - Statistical model of sequential experiments
  - Name derived from one-armed bandit (slot machine)
- Multiple actions $a_1, a_2, \ldots, a_n$
  - Each $a_i$ provides a reward from an unknown probability distribution $p_i$
  - Assume each $p_i$ is stationary
    - Same every time, regardless of history
  - Objective: maximize expected utility of a sequence of actions
- Exploitation vs exploration dilemma:
  - **Exploitation**: choose an action that has given you high rewards in the past
  - **Exploration**: choose an action that’s less familiar, in hopes that it might produce a higher reward
UCB (Upper Confidence Bound) Algorithm

- Assume all rewards are between 0 and 1
  - If they aren’t, normalize them
- For each action \( a \), let
  - \( r(a) \) = average reward you’ve gotten from \( a \)
  - \( n(a) \) = number of times you’ve tried \( a \)
  - \( n_t = \sum_a n(a) \)
  - \( Q(a) = r(a) + \sqrt{2 \ln n_t / n(a)} \)

UCB:

if there are any untried actions:
  \( \tilde{a} \leftarrow \text{any untried action} \)
else:
  \( \tilde{a} \leftarrow \arg\max_a Q(a) \)
perform \( \tilde{a} \)
update \( r(\tilde{a}), n(\tilde{a}), n_t, Q(\tilde{a}) \)

- Theorem (given some assumptions):
  As the number of calls to UCB \( \to \infty \),
  UCB’s choice at each call \( \to \) optimal choice
UCT Algorithm

- MDP: state space in which each action has probabilistic outcomes
  - We’ll discuss this in Chapter 6
- UCT algorithm: Monte Carlo rollouts on an MDP
- At each state $s$,
  - Use UCB to choose an action at random
    - Balances exploration vs exploitation at $s$
    - Action’s outcome $\Rightarrow$ next state $s$

- How to use UCT:
  - Call it many times, return action with highest expected utility
- Theorem:
  As number of calls to UCT $\rightarrow \infty$, choice converges to optimal
Convergence

- UCT algorithm:
  - Monte Carlo rollouts on MDPs
  - Call it many times, choice converges to optimal

- UPOM search tree more complicated
  - tasks, method instances, commands, code execution

- If no exogenous events,
  - Can map it to UCT search of a complicated MDP
  - Proof of convergence to optimal
Outline

1. Motivation
2. Representation
3. Acting (Rae)
4. Planning for Rae
5. *Acting with Planning* (RAE+UPOM)
6. Learning
7. Evaluation, Application

Outline:

1. **Motivation**
2. **Representation**
3. **Acting (Rae)**
4. **Planning for Rae**
5. *Acting with Planning* (RAE+UPOM)
6. **Learning**
7. **Evaluation, Application**
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
        call Progress(\( \sigma \))
        if \( \sigma \) is finished then remove it

● Whenever RAE needs to choose a method instance
  ▷ call Plan-with-UPOM, use the method instance it returns

● Open-source Python implementation: [https://bitbucket.org/sunandita/RAE/](https://bitbucket.org/sunandita/RAE/)

```

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

- \( \text{started } m \)?
  - yes
    - is \( m \)'s current step a command?
      - yes
        - running command status?
          - yes
            - succeed
            - no
              - failed
                - retry \( \tau \) using an untried candidate
        - retry \( \tau \) using an untried candidate
      - no
        - no more steps in \( m \)?
          - yes
            - \( \tau' \leftarrow \text{next step of } m \)
          - no
            - pop(\( \sigma \))
  - no
    - no candidates for \( \tau' \)?
      - yes
        - \( \tau' \leftarrow \text{next step of } m \)
      - no
        - push \((\tau', m', \ldots)\) onto \( \sigma \)

**send \( \tau' \) to the execution platform**

**update state**

**candidates for \( \tau' \)?**

**choose a candidate \( m' \)**

**type(\( \tau' \))**

**task**

**assignment**

**retry \( \tau \) using an untried candidate**
Can we use UPOM with Run-Lookahead?

- Suppose we try to use Run-Lookahead with a modified version of UPOM (call it UPOM’)
  - Instead of returning method instance $m_1$, return the actions in the last Monte Carlo rollout
    - $\pi = \langle a_1, a_2, a_3, a_4, a_5 \rangle$
    - corresponding commands: $c_1, c_2, c_3, c_4, c_5$

- Problem
  - Run-lookahead calls UPOM’, gets $\pi$, executes $c_1$, then calls UPOM’ again
  - This time, UPOM’ needs to plan for $t_1$ in state $s_1$ rather than $s_0$
  - There might not be an applicable method

- If we want to use Run-Lookahead, we need to ensure that methods can work in unexpected states
Can we use UPOM with Run-Lazy-Lookahead?

- Run-Lazy-Lookahead calls UPOM’, UPOM’ returns $\pi = \langle a_1, a_2, a_3, a_4, a_5 \rangle$
- Run-Lazy-Lookahead executes $c_1, c_2, c_3, c_4, c_5$, won’t call UPOM’ again unless something unexpected happens, e.g.,
  - command $c_2$ has an execution failure
  - $c_2$ produces a state in which $c_3$ is inapplicable
  - or an exogenous event makes $c_3$ inapplicable
    - Method $m_2$ fails; we need to replan task $t_2$
- Need to modify Run-Lazy-Lookahead so that when a failure occurs, it knows which task to replan
  - Need to modify the methods to work in unexpected states
Comparison

- Rae + UPOM has tighter coupling between planning and acting
  - works better than Run-Lazy-Lookahead + UPOM’

- Example
  - Case 1: Run-Lazy-Lookahead calls UPOM’ for $t_1$ in state $s_0$
    - UPOM’ returns $\pi = \langle a_1, a_2, a_3, a_4, a_5 \rangle$
    - corresponding commands: $c_1, c_2, c_3, c_4, c_5$
    - Run-Lazy-Lookahead executes $c_1$, gets state $s_1'$ (not $s_1$)
      - Suppose this makes action $a_2$ redundant
      - Run-Lazy-Lookahead doesn’t have a way to detect this; continues with the rest of $\pi$
  - Case 2: Rae calls UPOM for $t_1$ in state $s_0$
    - UPOM returns $m_1$, Rae executes $c_1$, gets state $s_1'$
    - Rae calls UPOM for $t_2$ in state $s_1'$
      - UPOM might return a better method instance
      - Or maybe UPOM returns $m_2$, but $m_2$’s body includes an if-test to omit $a_2$ if it’s redundant
Outline

1. Motivation
2. Representation
3. Acting (Rae)
4. Planning for Rae
5. Acting with Planning (RAE+UPOM)
6. Learning
7. Evaluation, Application
Motivation

- Plan-with-UPOM is called by RAE, runs online
  - Time constraints might not allow complete search

- Case 1: no time to search at all
  - need a choice function

- Case 2: enough time to do partial search
  - Receding horizon
    - Cut off search at depth $d_{\text{max}}$ or when we run out of time
    - At leaf nodes, use heuristic function to estimated expected utility

- Learning algorithms:
  - Learn$\pi$: learns a choice function
  - LearnH: learns a heuristic function
Integration with Learning

- Gather training data from acting-and-planning traces of RAE and Plan-with-UPOM
- Train classifiers (multi-layered perceptrons)

Learn $\pi$
- Learns function for choosing a method
- Given current task and context (state and other information), choose $m$ from the set of available refinement methods
- Useful if there isn’t enough time to use UPOM

Actor:

Planning (UPOM) \hspace{1cm} Learn $\pi$

Task, context $\uparrow$ $m$ \hspace{1cm} Acting Engine (RAE)
Integration with Learning

- Gather training data from acting-and-planning traces of RAE and Plan-with-UPOM
- Train classifiers (multi-layered perceptrons)

LearnH
- Learns a heuristic function to guide UPOM’s search
- UPOM can use it to estimate expected utility at leaf nodes
- Useful if there isn’t enough time to search all the way to the end

Actor:
- Planning (UPOM)
- LearnH
- Learnπ

Acting Engine (RAE)
Outline

1. Motivation
2. Representation
3. Acting (Rae)
4. Planning for Rae
5. Acting with Planning (RAE+UPOM)
6. Learning
7. Evaluation, Application
Experimental Evaluation

- Five different domains, different combinations of characteristics
- Evaluation criteria: efficiency (reciprocal of cost), successes vs failures
- Result: Planning and learning help
  - RAE operates better with UPOM or learning than without
  - RAE’s performance improves with more planning
Prototype Application

- Software-defined networks
  - Decoupled control and data layers
  - Prone to high-volume, fast-paced online attacks
  - Need automated attack recovery
- Prototype solution using RAE+UPOM
  - Expert writes recovery procedures as refinement methods
- Experimental results
  - Improved efficiency, retry ratio, success ratio, resilience compared to human expert

Summary

3.1 Operational models
- $\xi$ versus $s$, tasks, events,
- Commands to the execution platform
- Extensions to state-variable representation
- Refinement method
  - name, task/event, preconditions, body
- Example: fetch a container

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 $\setminus$ tried
  - comparison to backtracking
- Refinement trees

3.3 Refinement planning
- plan by simulating Rae on a single external task/event/goal
- SeRPE uses classical action models
- UPOM simulates the actor’s commands, does Monte Carlo rollouts

3.4 Acting and planning
- Rae + UPOM
- Comparison: Run-Lazy-Lookahead + UPOM’
- A little about learning, experimental evaluation, prototype application

Open-source Python implementation of Rae and UPOM:
- https://bitbucket.org/sunandita/RAE/