Chapter 3
Deliberation with Refinement Methods

Section 3.1: Operational Models
Section 3.2: A Refinement Acting Engine

Dana S. Nau
University of Maryland
Motivation

- Hierarchically organized deliberation
  - At high levels, abstract actions
  - At lower levels, more detail

- Refine abstract actions into ways of carrying out those actions
  - How?

Diagram:
- 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

Actions:
- Identify type of door
- Move close to knob
- Grasp knob
- Turn knob
- Maintain
- Move back
- Ungrasp
- Pull
- Monitor
Opening a Door

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

Opening a Door

<table>
<thead>
<tr>
<th>open door</th>
</tr>
</thead>
<tbody>
<tr>
<td>identify type of door</td>
</tr>
<tr>
<td>move close to knob</td>
</tr>
<tr>
<td>grasp knob</td>
</tr>
<tr>
<td>turn knob</td>
</tr>
<tr>
<td>maintain</td>
</tr>
<tr>
<td>pull</td>
</tr>
<tr>
<td>monitor</td>
</tr>
<tr>
<td>move back</td>
</tr>
</tbody>
</table>

Nau – Lecture slides for Automated Planning and Acting
Opening a Door

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

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

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

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

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

- Identify type of door
- Move close to knob
- Grasp knob
- Turn knob
- Maintain pull
- Monitor pull
- Monitor move back
- Open door

- Get out
d- Close door

- Respond to user requests
- Bring object to room 2
- Go to hallway
- Deliver object
- Move to door
- Fetch object
- Navigate to room 2
- Navigate to room 1

Nau – Lecture slides for *Automated Planning and Acting*
Outline

3.1 Representation
   a. State variables, commands, refinement methods
   b. Example

3.2 Acting
   a. Rae (Refinement Acting Engine)
   b. Example
   c. Extensions

3.3 Planning
   a. Motivation and basic ideas
   b. Deterministic action models
   c. SeRPE (Sequential Refinement Planning Engine)

3.4 Using Planning in Acting
   a. Techniques
   b. Caveats
Quick review:

- **Objects**: $Robots = \{\text{rbt}\}$, $Containers = \{c1, c2, c3, \ldots\}$, $Locations = \{\text{loc0}, \text{loc1}, \text{loc2}, \ldots\}$

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

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

Details: *Automated Planning and Acting*, Sections 2.1 and 3.1.1
State-Variable Representation

Extensions:

- **Range(x)**
  - can be finite, infinite, continuous, discontinuous, vectors, matrices, other data structures

- **Assignment statement** $x \leftarrow expr$
  - expression that returns a ground value in $\text{Range}(x)$
  - and has no side-effects on the current state

- **Tests** (e.g., preconditions)
  - **Simple**: $x = v$, $x \neq v$, $x > v$, $x < v$
  - **Compound**: conjunction, disjunction, or negation of simple tests
**Commands**

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

---

**Diagram**

- **Acting**
  - **Environment**
  - **Execution Platform**
  - **tasks**
  - **commands**
  - **events**
Tasks and Methods

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

```plaintext
method-name(arg₁, ..., argₖ)
task:  task-identifier
pre:  test
body:  a program
```

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

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

**m1-unlatch(r,d,l,o)**

**task:** un latch(r, d)

**pre:** loc(r,l)∧toward-side(l,d)∧
side(d,left)∧type(d,rotate)∧handle(d,o)

**body:**
- grasp(r, o)
  - turn(r, o, alpha1)
  - pull(r, val1)
  - if door-status(d)=cracked then ungrasp(r, o)
  - else fail

**m1-throw-wide(r,d,l,o)**

**task:** throw-wide(r, d)

**pre:** loc(r,l)∧toward-side(l,d)∧
side(d,left)∧type(d,rotate)∧
handle(d,o)∧door-status(d)=cracked

**body:**
- grasp(r, o)∧door-status(d)=cracked
  - pull(r, val1)
  - move-by(r, val2)
Outline

3.1 Representation
   a. State variables, commands, refinement methods
   b. Example

3.2 Acting
   a. Rae (Refinement Acting Engine)
   b. Example
   c. Extensions

3.3 Planning
   a. Motivation and basic ideas
   b. Deterministic action models
   c. SeRPE (Sequential Refinement Planning Engine)

3.4 Using Planning in Acting
   a. Techniques
   b. Caveats
Rae (Refinement Acting Engine)

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

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

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

- For each task/event, a refinement stack
  - current path in Rae’s search tree for the task/event
- Agenda = \{all current refinement stacks\}
Rae (Refinement Acting Engine)

Basic idea:

\[ \text{Agenda} = \{\text{current refinement stacks}\} \]

\text{loop:}

if new external tasks/events then add them to \text{Agenda}

for each stack in \text{Agenda}

Progress it, and remove it if it’s finished

\[ \text{Rae}(\mathcal{M}) \]

\[ \text{Agenda} \leftarrow \emptyset \]

\text{loop}

\text{until the input stream of external tasks and events is empty do}

\text{read } \tau \text{ in the input stream}

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

\text{if Candidates } = \emptyset \text{ then output (“failed to address” } \tau)\]

\text{else do}

\text{arbitrarily choose } m \in \text{Candidates}

\text{Agenda } \leftarrow \text{Agenda } \cup \{((\tau, m, \text{nil}, \emptyset))\}

\text{for each stack } \in \text{Agenda do}

\text{Progress(stack)}

\text{if stack } = \emptyset \text{ then Agenda } \leftarrow \text{Agenda } \setminus \{\text{stack}\} \]

Stack element \((\tau,m,i,\text{tried})\)

\(\tau: \text{ task} \)

\(m: \text{ method} \)

\(i: \text{ instruction pointer} \)

\(\text{tried}: \text{ methods already tried} \)
Progress (subroutine)

Progress(\textit{stack})
(\tau, m, i, tried) \leftarrow \text{top}(\textit{stack})
if \(i \neq \text{nil}\) and \(m[i]\) is a command then do
\hspace{1em} \text{case status}(m[i])
\hspace{2em} \text{running}: \text{return}
\hspace{2em} \text{failure}: \text{Retry}(\textit{stack}); \text{return}
\hspace{2em} \text{done}: \text{continue}
if \(i\) is the last step of \(m\) then
\hspace{1em} \pop(\textit{stack}) \quad // \text{remove stack's top element}
else do
\hspace{1em} \(i \leftarrow \text{nextstep}(m, i)\)
\hspace{1em} \text{case type}(m[i])
\hspace{2em} \text{assignment}: \text{update } \xi \text{ according to } m[i]; \text{return}
\hspace{2em} \text{command}: \text{trigger command } m[i]; \text{return}
\hspace{2em} \text{task or goal}: \text{continue}
\hspace{2em} \tau' \leftarrow m[i]
Candidates \leftarrow \text{Instances}(\mathcal{M}, \tau', \xi)
if Candidates = \emptyset then \text{Retry}(\textit{stack})
else do
\hspace{1em} \text{arbitrarily choose } m' \in \text{Candidates}
\hspace{1em} \text{stack} \leftarrow \text{push}((\tau', m', \text{nil}, \emptyset), \text{stack})

Just a decision tree:

- \(m[i]\) finished?
  - yes: \return
  - no: \text{Retry}

- more steps?
  - yes: \text{pop stack}
  - no:
    - assignment: \(m[i]'\)s command type?
      - task or goal:
        - yes: candidates for \(m[i]\)?
          - yes: choose candidate \(m'\)
            - push \((m[i], m', \text{nil, } \emptyset)\) onto stack
          - no: \text{Retry}
        - trigger it
      - no: \text{Retry}
    - update \(\xi\)
  - \text{trigger it}
- **Objects**
  - $Robots = \{ rb t \}$
  - $Containers = \{ c_1, c_2, \ldots \}$
  - $Locations = \{ loc_1, loc_2, \ldots \}$

- **State variables**
  - $loc(r) \in Locations$
  - $load(r) \in Containers \cup \{ nil \}$
  - $pos(c) \in Locations \cup Robots \cup \{ unknown \}$
  - $view(l) \in \{ T, F \}$
    - Whether the robot has looked at location $l$ or not
    - If $view(l) = T$ then for every container $c$ at location $l$, $pos(c) = l$

- **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 location $l$
  - $move-to(r, l)$: robot $r$ moves to location $l$
Example

\texttt{m-fetch(r,c)}

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

\textbf{pre: }

\textbf{body: }

\begin{itemize}
\item if pos(c) = unknown then
  \begin{itemize}
  \item search(r,c)
  \end{itemize}
\item else if loc(r) = pos(c) then
  \begin{itemize}
  \item take(r,c,pos(c))
  \end{itemize}
\item else do
  \begin{itemize}
  \item move-to(r,pos(c))
  \item take(r,c,pos(c))
  \end{itemize}
\end{itemize}

\texttt{m-search(r,c)}

\textbf{task: } search(r,c)

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

\textbf{body: }

\begin{itemize}
\item if \( \exists l \) (view(r,l) = F) then
  \begin{itemize}
  \item move-to(r,l)
  \item perceive(l)
  \item if pos(c) = l then
    \begin{itemize}
    \item take(r,c,l)
    \end{itemize}
  \item else search(r,c)
  \end{itemize}
\item else fail
\end{itemize}
Example

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

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

Refinement stack

\begin{itemize}
  \item \(\tau\): \texttt{fetch}(r1,c2)
  \item \(m\): \texttt{m-fetch}(r1,c2)
  \item \(i\): (see method)
  \item \texttt{tried}: \emptyset
\end{itemize}
Example

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

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

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

\(\text{body: } \)

\(\text{if } \text{pos}(c) = \text{unknown} \text{ then } \text{search}(r,c)\)

\(\text{else if } \text{loc}(r) = \text{pos}(c) \text{ then } \text{take}(r,c,\text{pos}(c))\)

\(\text{else do } \)

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

\(\text{take}(r,c,\text{pos}(c))\)

m-search\((r,c)\)

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

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

\(\text{body: } \)

\(\text{if } \exists l (\text{view}(r,l) = F) \text{ then } \)

\(\text{move-to}(r,l)\)

\(\text{perceive}(l)\)

\(\text{if } \text{pos}(c) = l \text{ then } \text{take}(r,c,l)\)

\(\text{else } \text{search}(r,c)\)

\(\text{else } \text{fail}\)

\(\tau: \text{search}(r1,c2)\)

\(m: ?\)

\(i: \text{(see method)}\)

\(\text{tried: } \emptyset\)

\(\tau: \text{fetch}(r1,c2)\)

\(m: \text{m-fetch}(r1,c2)\)

\(i: \text{(see method)}\)

\(\text{tried: } \emptyset\)

Refinement stack

fetch\((r1,c2)\)

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

search\((r1,c2)\)
Example

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

m-search(r,c)

m-fetch(r1,c2)
task: fetch(r1,c2)
pre: pos(c) = unknown
body:
  if ∃l (view(r,l) = F) then
    move-to(r,l)
    perceive(l)
    if pos(c) = l then
      take(r,c,l)
    else search(r,c)
  else fail

m-search(r1,c2)
task: search(r1,c2)
pre:
body:
  if pos(c) = unknown then
    search(r1,c2)
  else if loc(r) = pos(c) then
    take(r1,c2,pos(c))
  else do
    move-to(r,pos(c))
    take(r1,c2,pos(c))

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

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

\( \tau: \text{search}(r1,c2) \)
\( m: \text{m-search}(r1,c2) \)
\( i: (\text{see method}) \)
\( \text{tried}: \emptyset \)

\( \tau: \text{fetch}(r1,c2) \)
\( m: \text{m-fetch}(r1,c2) \)
\( i: (\text{see method}) \)
\( \text{tried}: \emptyset \)
m-fetch(r,c)
  task: fetch(r,c)
  pre: 
  body:
    if pos(c) = unknown then
      \textbf{search(r,c)}
    else if loc(r) = pos(c) then
      take(r,c,pos(c))
    else do
      move-to(r,pos(c))
      take(r,c,pos(c))

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

\[ \tau: \text{search}(r1,c2) \]
\[ m: \text{m-search}(r1,c2) \]
\[ i: \text{(see method)} \]
\[ \text{tried: } \emptyset \]

\[ \tau: \text{fetch}(r1,c2) \]
\[ m: \text{m-fetch}(r1,c2) \]
\[ i: \text{(see method)} \]
\[ \text{tried: } \emptyset \]

\( \text{Refinement stack} \)

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

\( \text{m-search}(r,c) \)
\begin{align*}
\text{task: } & \text{search}(r,c) \\
\text{pre: } & \text{pos}(c) = \text{unknown} \\
\text{body: } & \\
& \text{if } \exists l \ (\text{view}(r,l) = F) \text{ then} \\
& \quad \text{move-to}(r,l) \\
& \quad \text{perceive}(l) \quad \text{sensor failure} \\
& \text{if } \text{pos}(c) = l \text{ then} \\
& \quad \text{take}(r,c,l) \\
& \text{else search}(r,c) \\
& \text{else fail} \\
\end{align*}
Example

m-fetch($r,c$)

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

- m-search($r,c$)

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

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

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

- Refinement stack

- If other candidates for search($r1,c2$), try them
Retry (subroutine)

\[ \text{Retry}(stack) \]
\[ (\tau, m, i, \text{tried}) \leftarrow \text{pop}(stack) \]
\[ \text{tried} \leftarrow \text{tried} \cup \{m\} \]
\[ \text{Candidates} \leftarrow \text{Instances}(M, \tau, \xi) \setminus \text{tried} \]
\[ \text{if Candidates} \neq \emptyset \text{ then do} \]
\[ \quad \text{arbitrarily choose } m' \in \text{Candidates} \]
\[ \quad \text{stack} \leftarrow \text{push}((\tau, m', \text{nil, tried}), \text{stack}) \]
\[ \text{else do} \]
\[ \quad \text{if stack} \neq \emptyset \text{ then Retry(stack)} \]
\[ \quad \text{else do} \]
\[ \quad \quad \text{output(“failed to accomplish” } \tau) \]
\[ \quad \quad \text{Agenda} \leftarrow \text{Agenda} \setminus \text{stack} \]

Poll: Is this the same as backtracking?
1. yes
2. no

Another decision tree:

pop top stack element \((\tau,m,i,\text{tried})\)
add \(m\) to \(\text{tried}\)

untried candidates for \(\tau\)?

yes

choose one push onto stack

no

stack empty?

yes

fail

no

Retry
Example

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

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

\(\tau: fetch(r1,c2)\)
\(m: m\text{-fetch}(r1,c2)\)
\(i: \text{(see method)}\)
\(\text{tried: } \emptyset\)
Example

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

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

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

Refinement stack
Extensions to Rae

Events:

- Example: an emergency
  - If \( r \) isn’t already handling another emergency, then
    - stop what it’s doing
    - go handle the emergency

\[
\text{method-name}(\text{arg}_1, \ldots, \text{arg}_k)
\]
\[
\text{event: event-identifier}
\]
\[
\text{pre: test}
\]
\[
\text{body: program}
\]
Extensions to Rae

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

Example:

\[
\text{m-fetch}(r,c) \\
task: \text{fetch}(r,c) \\
pre: \\
body: \\
\begin{align*}
&\text{if } \text{pos}(c) = \text{unknown} \text{ then} \\
&\hspace{1cm} \text{search}(r,c) \ \text{achieve}(\text{pos}(c) \neq \text{unknown}) \\
&\text{else if } \text{loc}(r) = \text{pos}(c) \text{ then} \\
&\hspace{1cm} \text{take}(r,c,\text{pos}(c)) \\
&\text{else do} \\
&\hspace{1cm} \text{move-to}(r,\text{pos}(c)) \\
&\hspace{1cm} \text{take}(r,c,\text{pos}(c))
\end{align*}
\]
Extensions to Rae

- Concurrent subtasks:
  - refinement stack for each one

- Controlling the progress of tasks:
  - e.g., suspend a task for a while
  - If there are multiple stacks, which ones get higher priority?
    - Application-specific heuristics

- For a task \( \tau \), which candidate to try first?
  - Refinement planning

\[
\text{body of a method:} \\
\ldots \\
\{\text{concurrent: } \tau_1, \tau_2, \ldots, \tau_n\} \\
\ldots
\]

\[
\text{Agenda} = \{\text{stack}_1, \text{stack}_2, \ldots, \text{stack}_n\}
\]

\[
\text{Candidates} = \text{Instances}(\tau, \mathcal{M}, \xi)
\]
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
- Examples: fetch container, emergency, opening a door

3.2 Refinement Acting Engine (RAE)

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