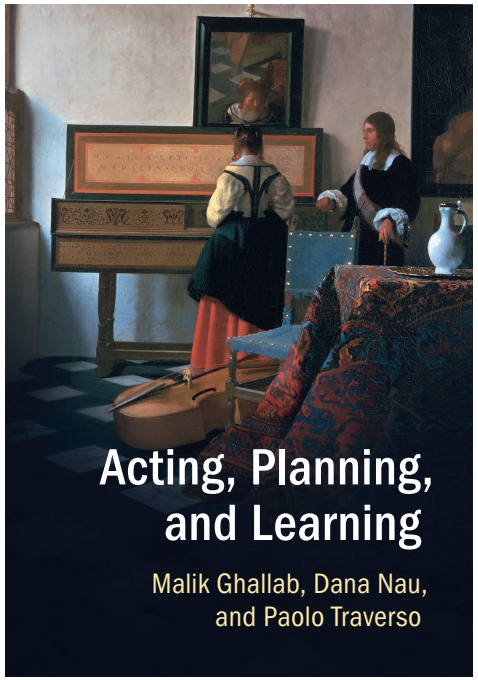
Last update: 11:09 PM, March 7, 2025



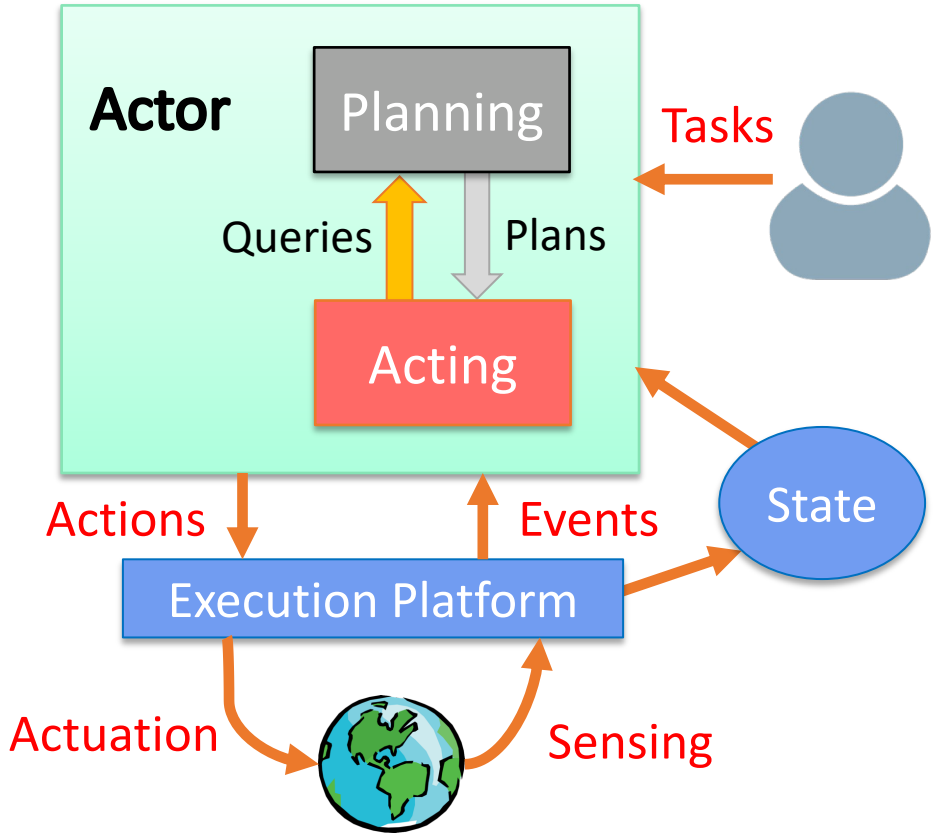
Chapter 14
Acting with Hierarchical Refinement

Dana S. Nau

University of Maryland

Planning and Acting

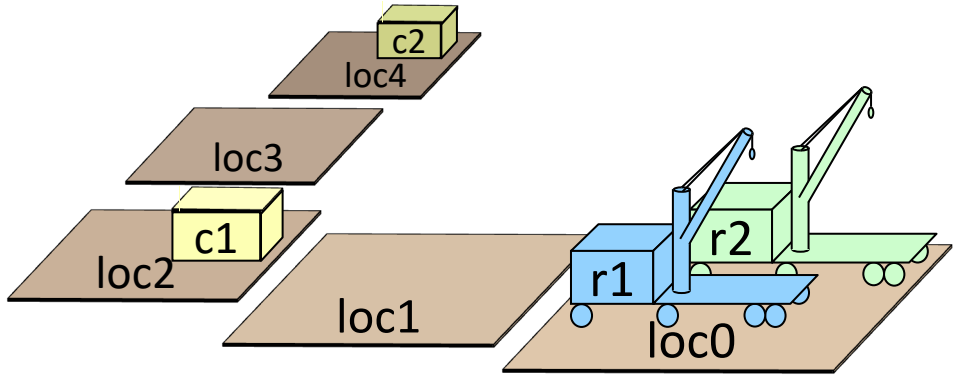
- **Planning**: *prediction + search*
  - Search over predicted states, possible organizations of tasks and actions
  - Uses *descriptive* models (e.g., PDDL)
    - predict *what* the actions will do
    - don't include instructions for performing it
- **Acting**: *performing*
  - Dynamic, unpredictable, partially observable environment
    - Adapt to context, react to events
  - Uses *operational* models
    - instructions telling *how* to perform the tasks
    - usually hierarchical



← We'll use *hierarchical refinement methods*
- Extended version of HTN methods

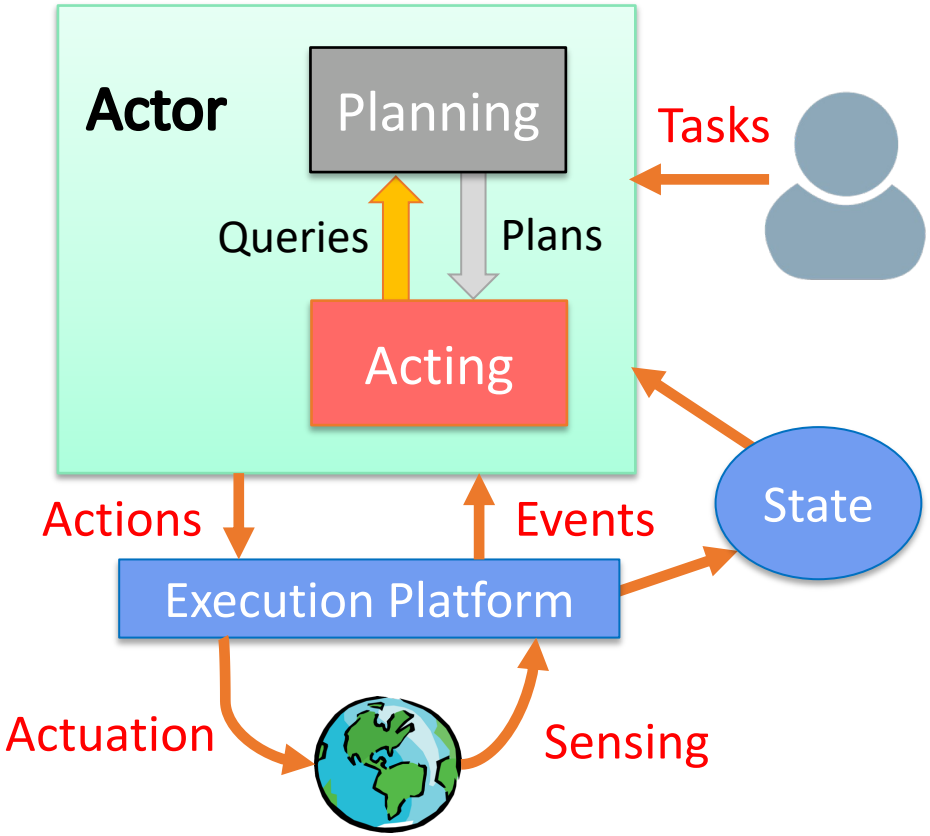
Example

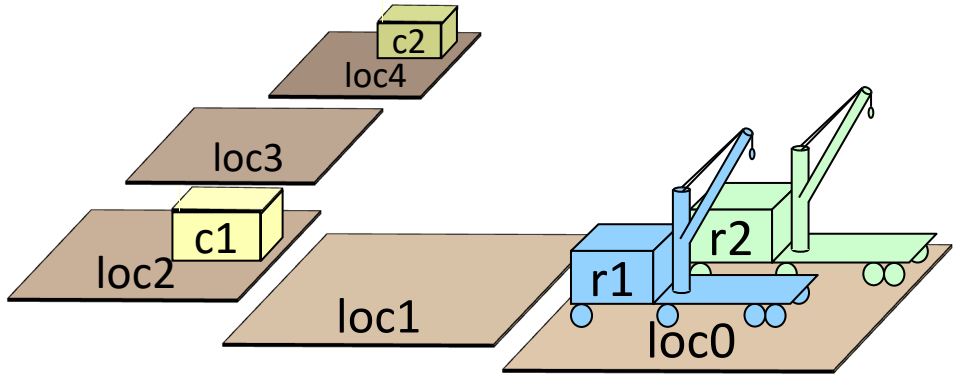
- Consider an actor that controls two robots
- Environment is *partially observable*
  - Each robot can only see what's at the current location



- Objects
  - *Robots* = {r1, r2}
  - *Containers* = {c1, c2}
  - *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 (fluents)
  - 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)

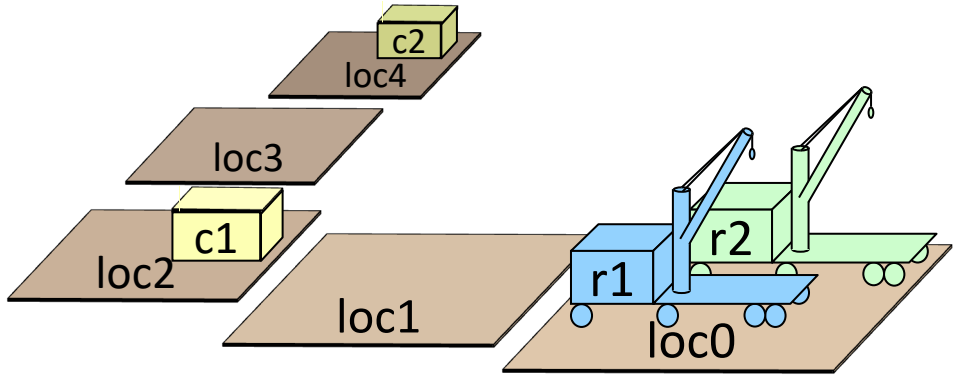




- Actions:
  - take(r,o,l): r takes object o at location l
  - put(r,o,l): r puts o at location l
  - perceive(r,l): robot r perceives what objects are at l
  - move-to(r,l): robot r moves to location l
- These are **not** classical actions
  - Commands to the execution platform

Tasks and Refinement 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
  - Like extended versions of HTN methods



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 to include events, goals
- actions

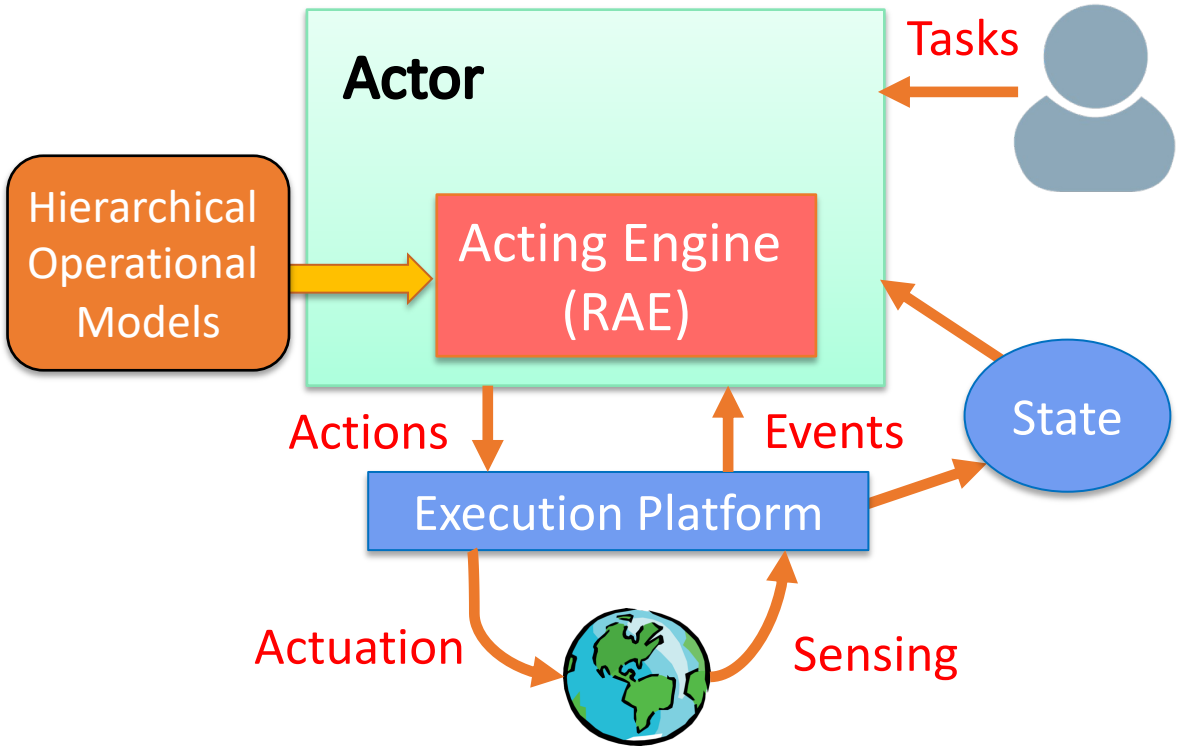
m-fetch1(r,c)
  task: fetch(r,c)
  pre: pos(c) = unknown
  body:
    if ∃l (view(l) = F) then
      move-to(r,l)
      perceive(r,l)
      if pos(c) = l then
        take(r,c,l)
      else fetch(r,c)
    else fail

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))

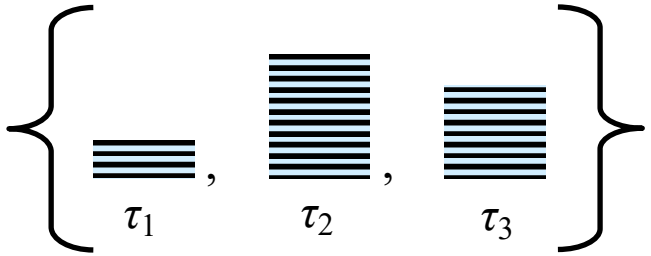
action

task

RAE (Refinement Acting Engine)



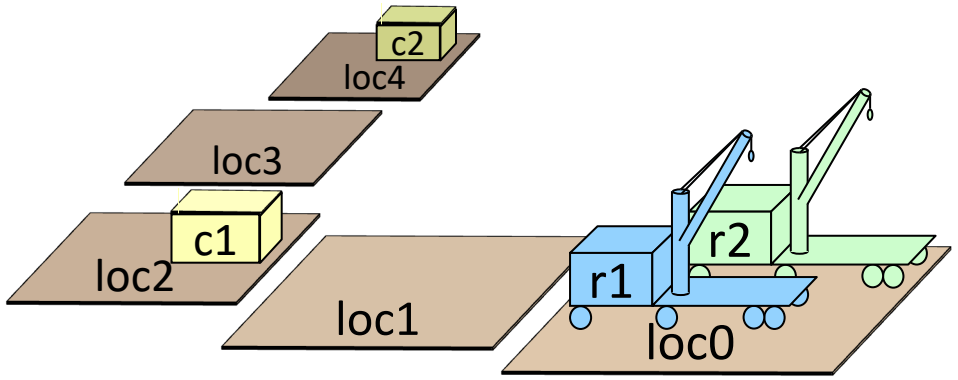
- Performs multiple tasks in parallel
  - Purely reactive, no lookahead
- For each task τ, a *refinement stack*
  - execution stack
- *Agenda* = {all current refinement stacks}



- Refinement stack for a task τ
  ⇔ current path in the refinement tree

```
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
```

Example (reminder)



- Objects
  - *Robots* = {r1, r2}
  - *Containers* = {c1, c2}
  - *Locations* = {loc1, loc2, loc3, loc4}

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

- State variables (fluents)
  - where $r \in$ *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

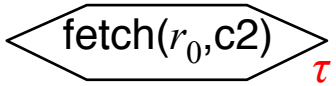
- Actions:
  - take(r,o,l): r takes object o at location l
  - put(r,o,l): r puts o at location l
  - perceive(r,l): robot r perceives what objects are at l
  - move-to(r,l): robot r moves to location l

Example

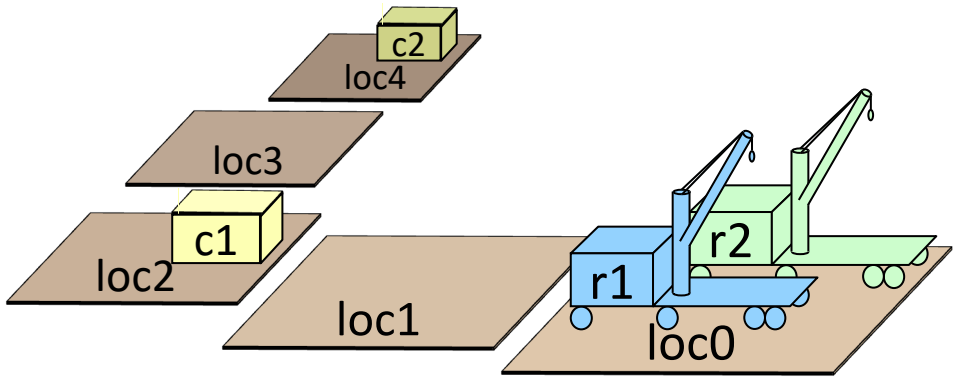
```
m-fetch1(r,c)
    task:   fetch(r,c)
    pre:    pos(c) = unknown
    body:
      if ∃l (view(l) = F) then
            move-to(r,l)
            perceive(r,l)
            if pos(c) = l then
                  take(r,c,l)
            else fetch(r,c)
      else fail
```

```
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))
```

Refinement tree



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




```
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
```

Example

m-fetch1(r,c) $r = r_0$, c = c2
- 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

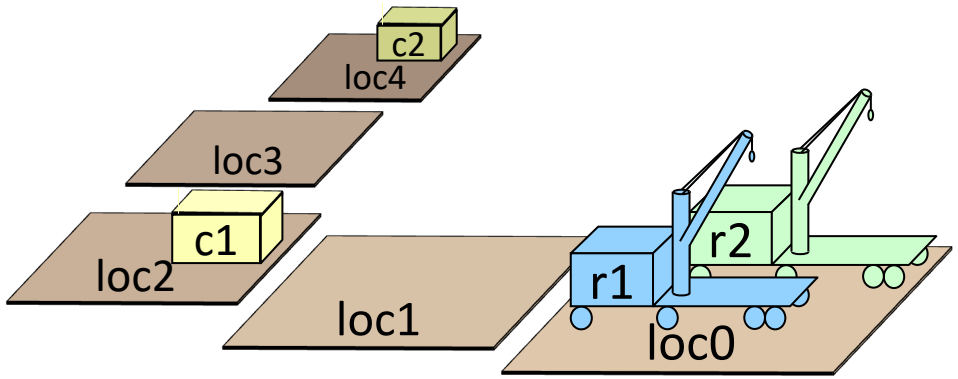
Refinement tree

fetch(r_0,c2) τ

Candidates
= {m-fetch1(r1,c2),
m-fetch1(r2,c2)}

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

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

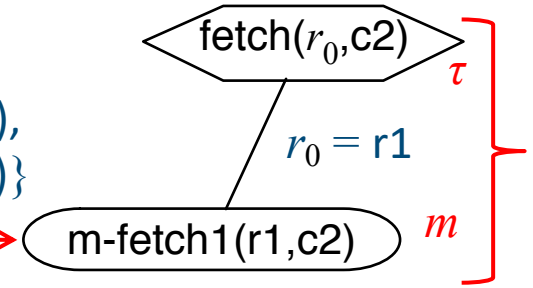


procedure RAE:
- loop:
  - for every new external task or event τ do
    - choose a method instance m for τ
    - create a refinement stack for τ, m
    - add the stack to *Agenda*
  - for each stack σ in *Agenda*
    - call Progress(σ)
    - if σ is finished then remove it

Example

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

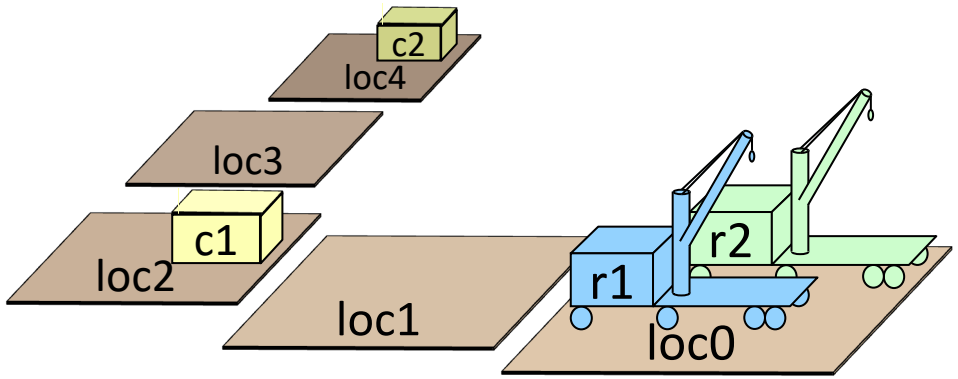
Refinement tree



Candidates
= {m-fetch1(r1,c2),
m-fetch1(r2,c2)}

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

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



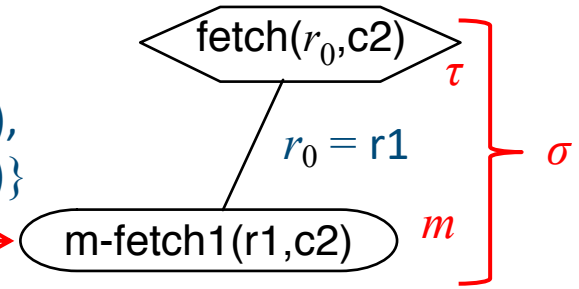

```
procedure RAE:
    loop:
        for every new external task or event τ do
            choose a method instance m for τ
            create a refinement stack for τ, m
            add the stack to Agenda
        for each stack σ in Agenda
            call Progress(σ)
            if σ is finished then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

Example

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

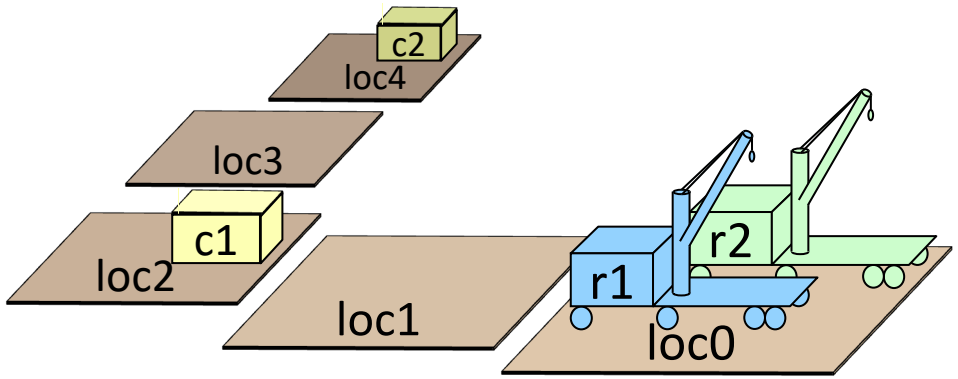
Candidates
= {m-fetch1(r1,c2),
m-fetch1(r2,c2)}

Refinement tree



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

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



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

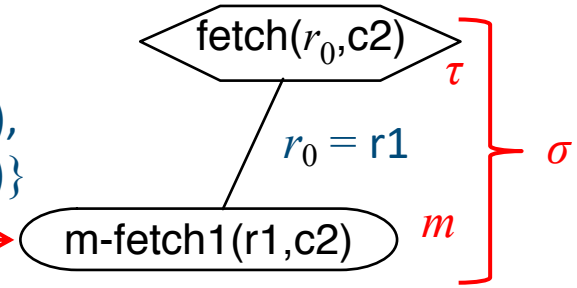
Example

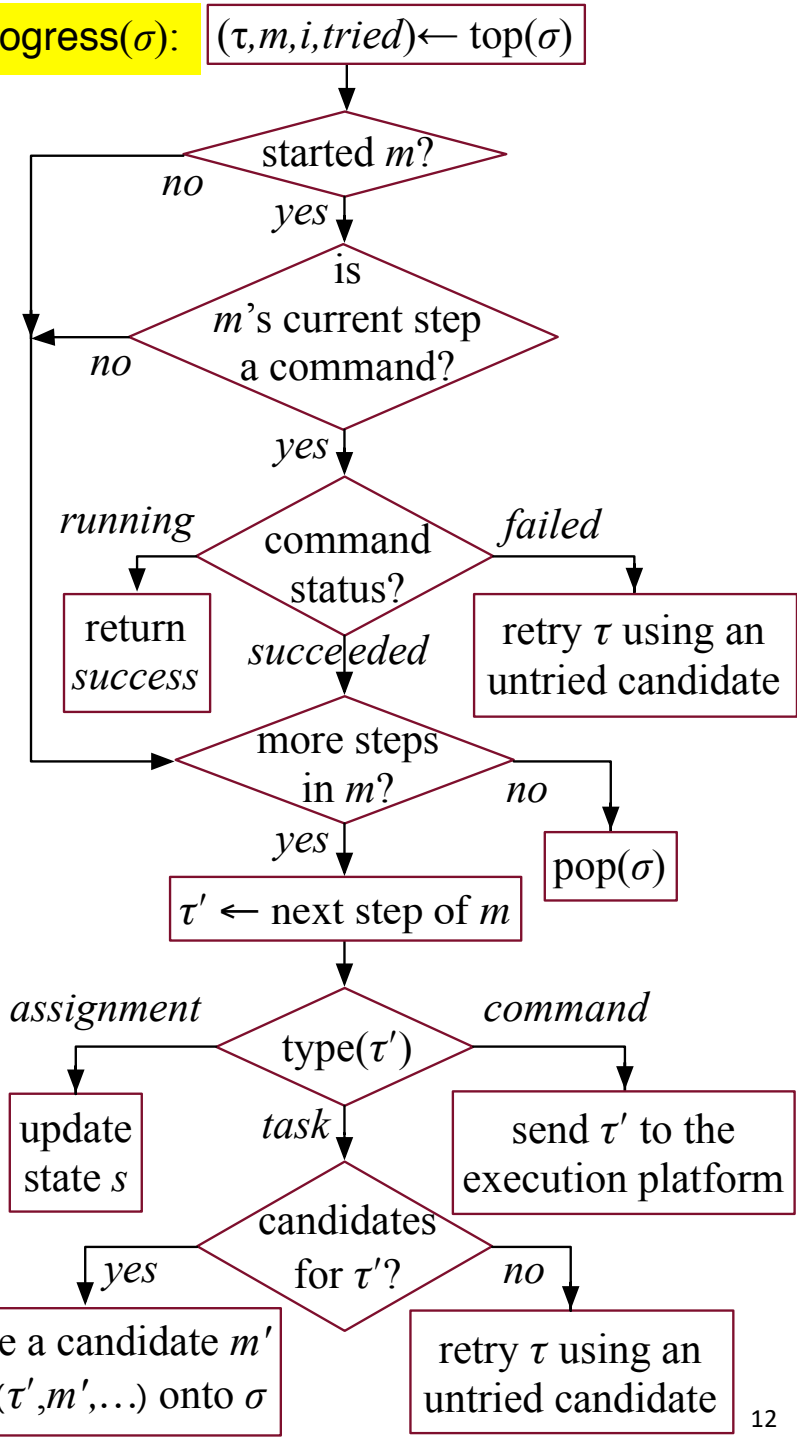
m-fetch1(r,c) r = r1, c = c2

```
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
```

Candidates = {m-fetch1(r1,c2), m-fetch1(r2,c2)}

Refinement tree

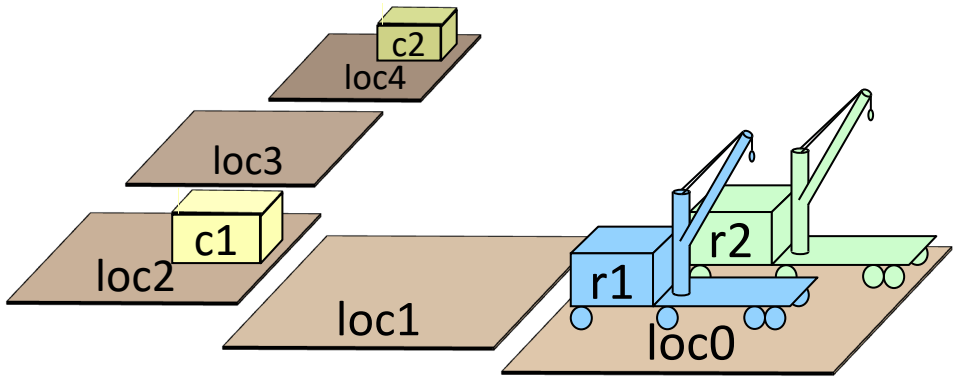




m-fetch2(r,c)

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

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

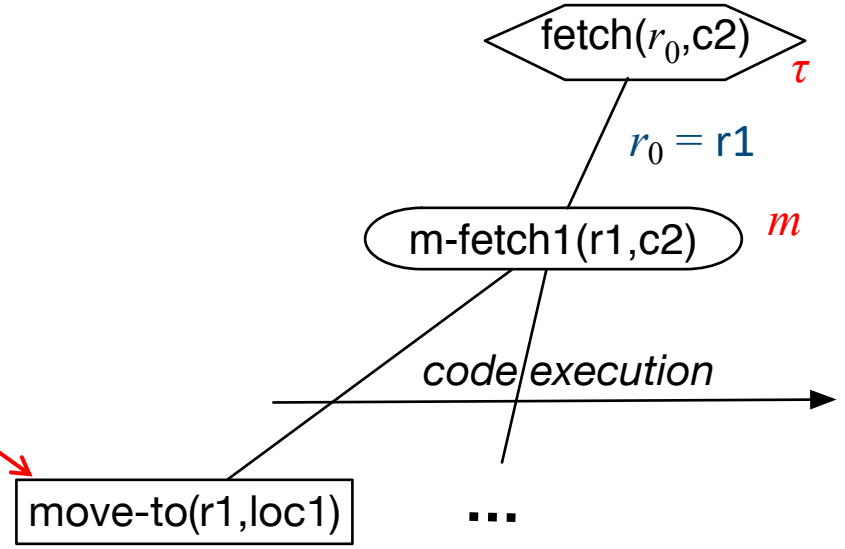


Example

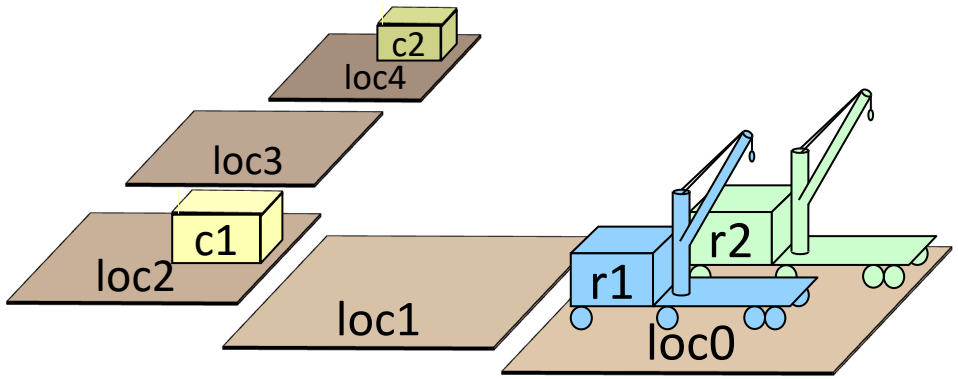
m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

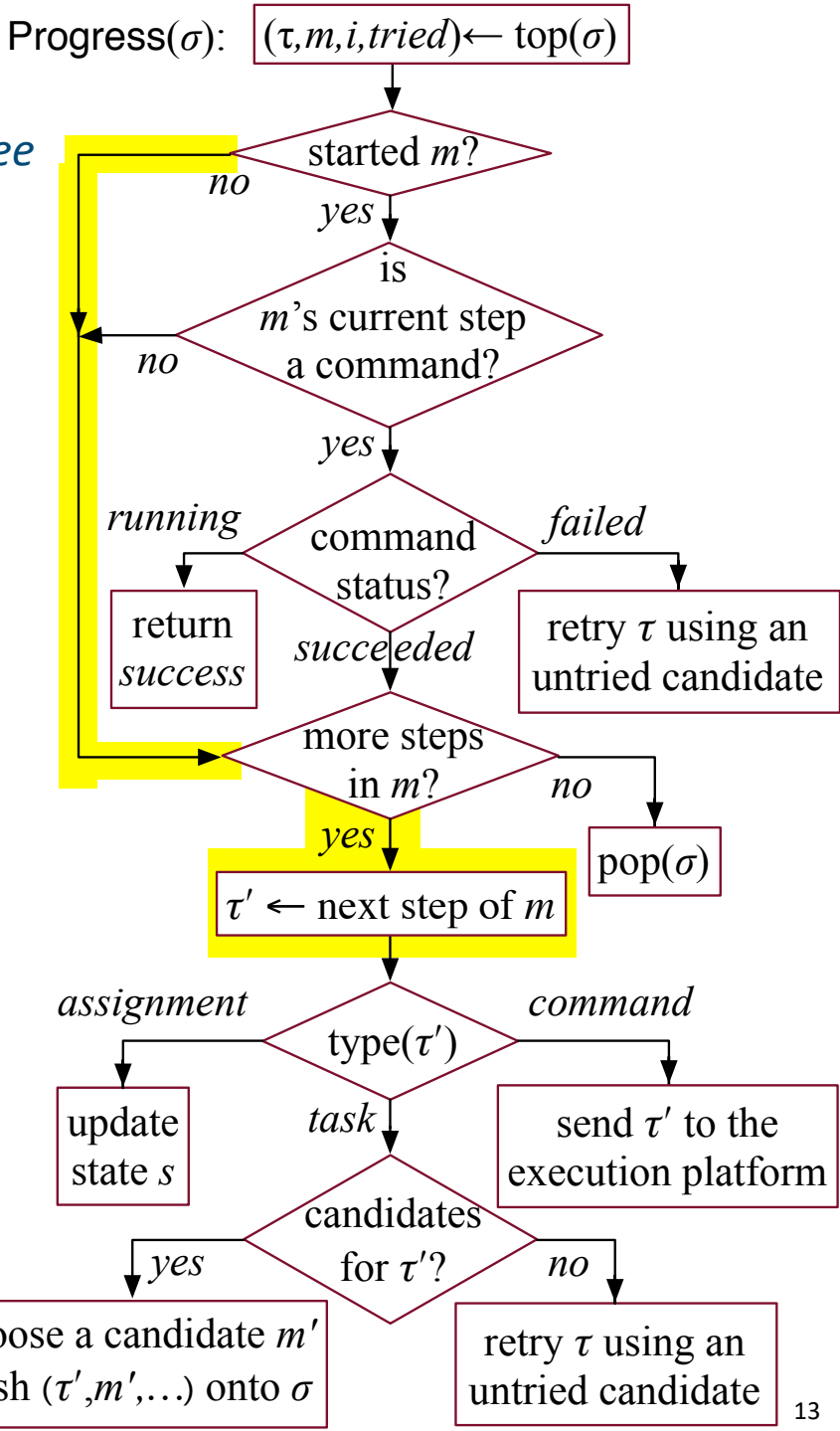
m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

Refinement tree



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



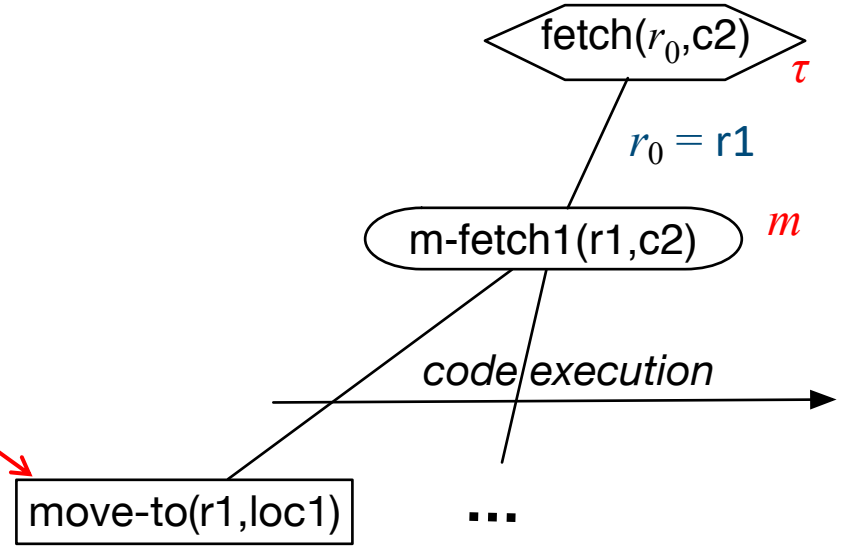


Example

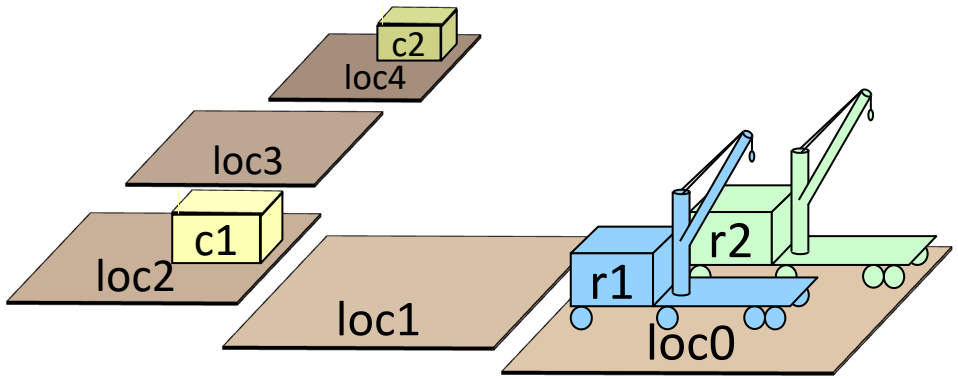
m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

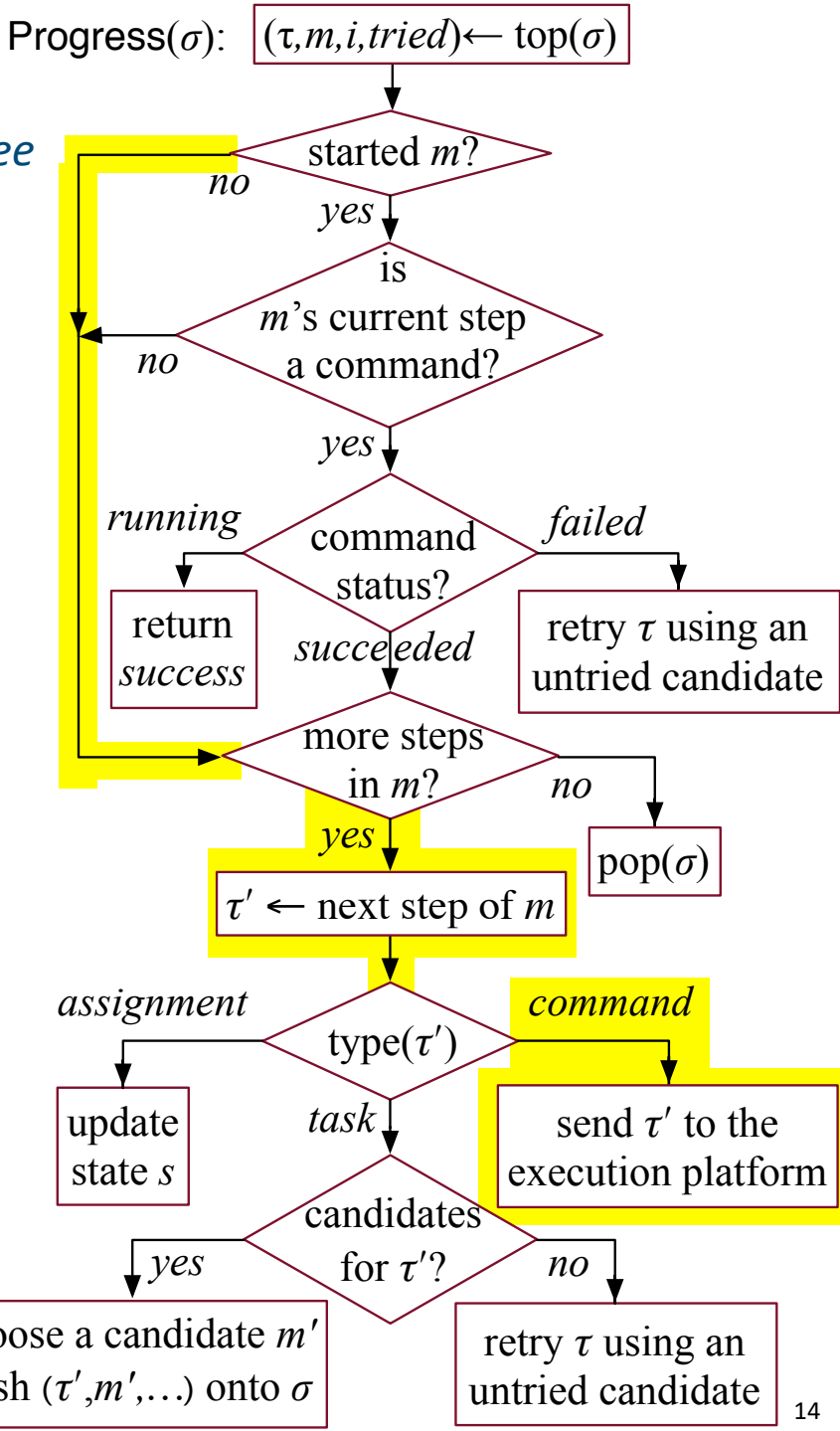
m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

Refinement tree



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

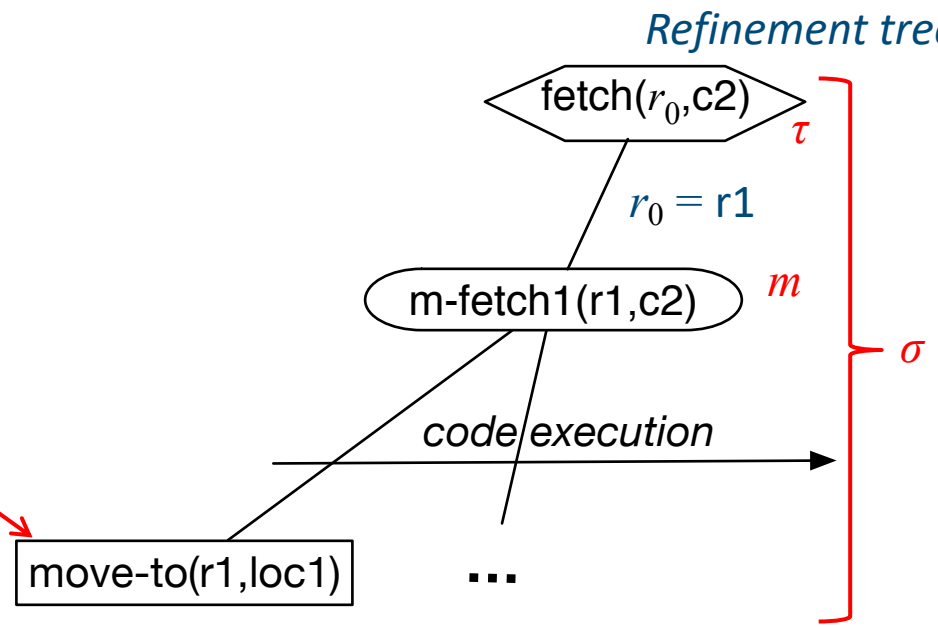




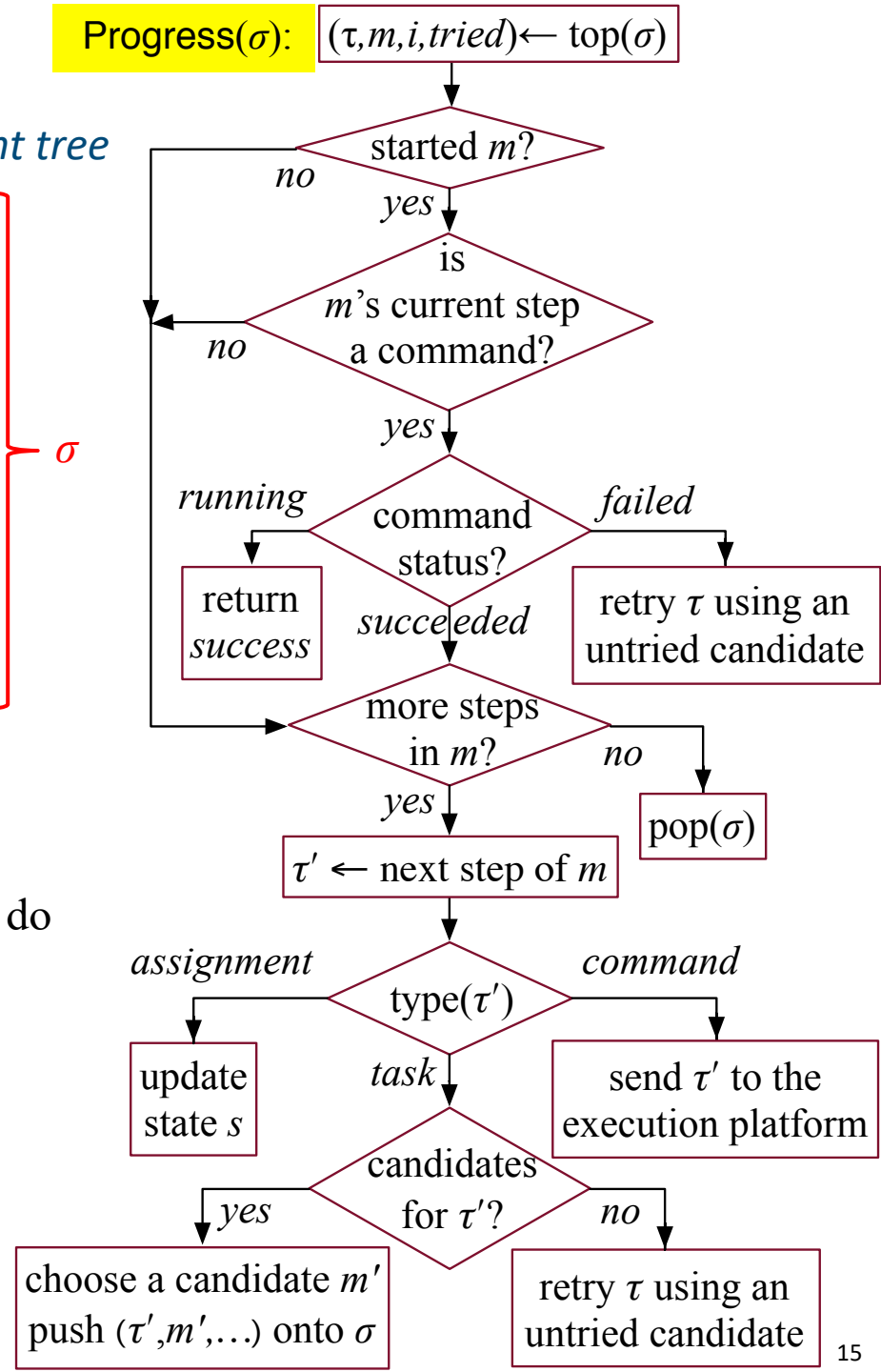
Example

m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - 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))



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

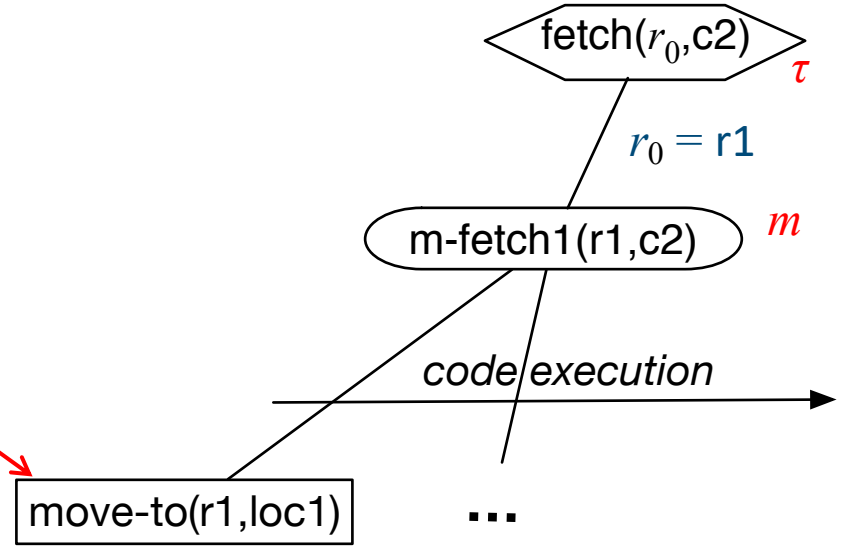


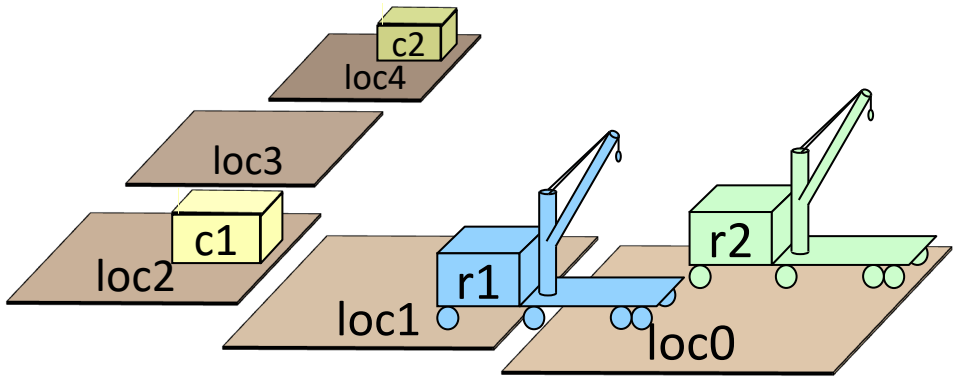
Example

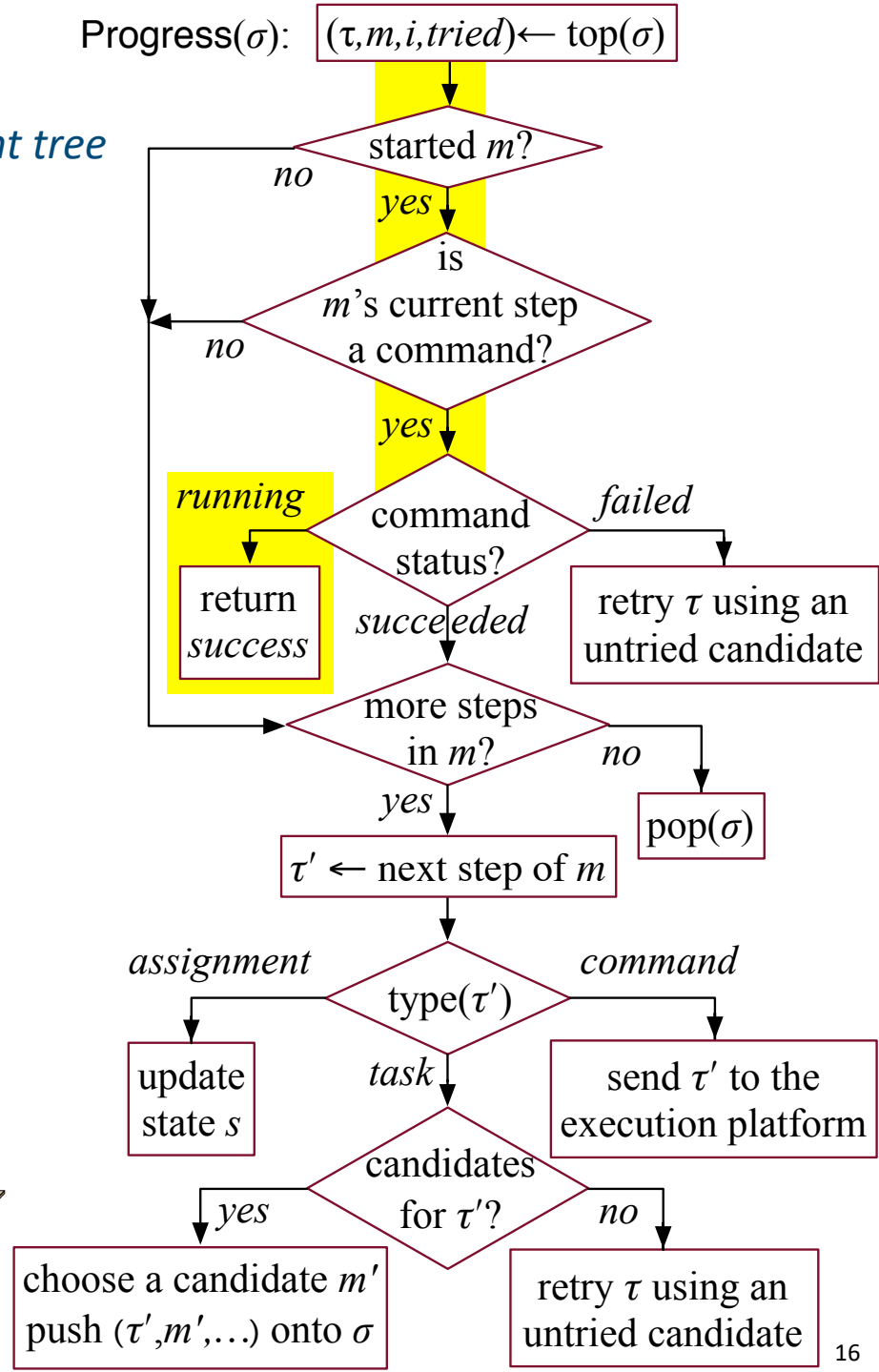
m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l) ← *running …*
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

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))

Refinement tree





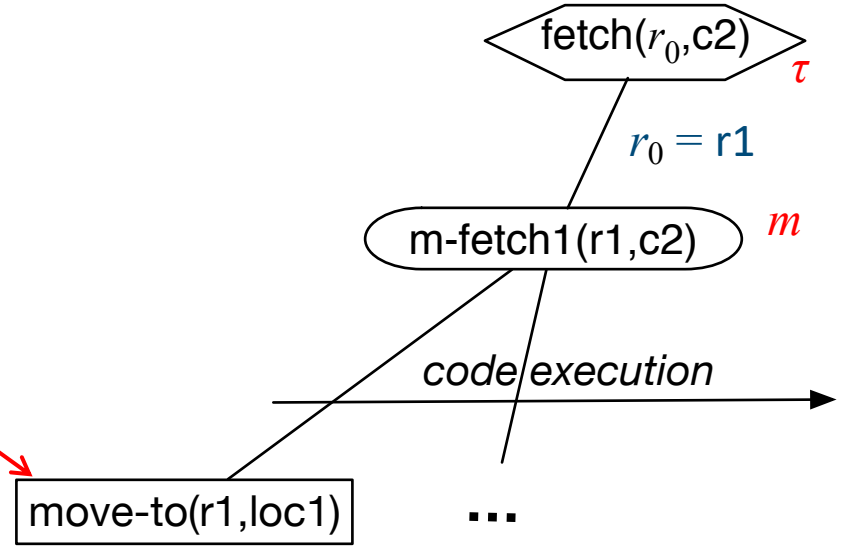


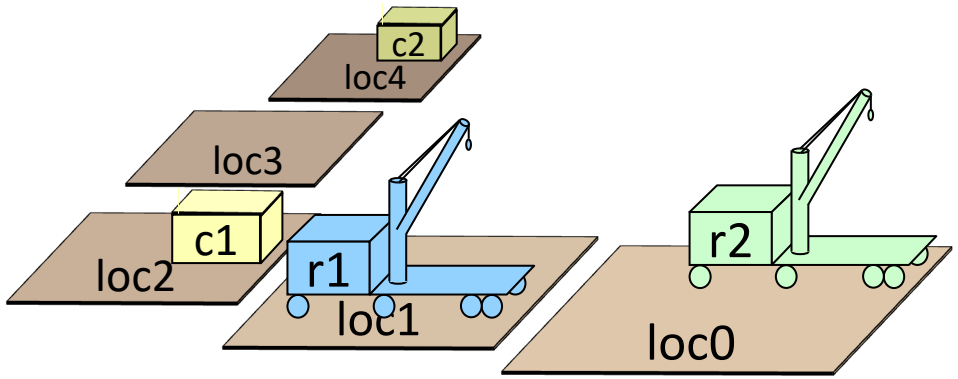
Example

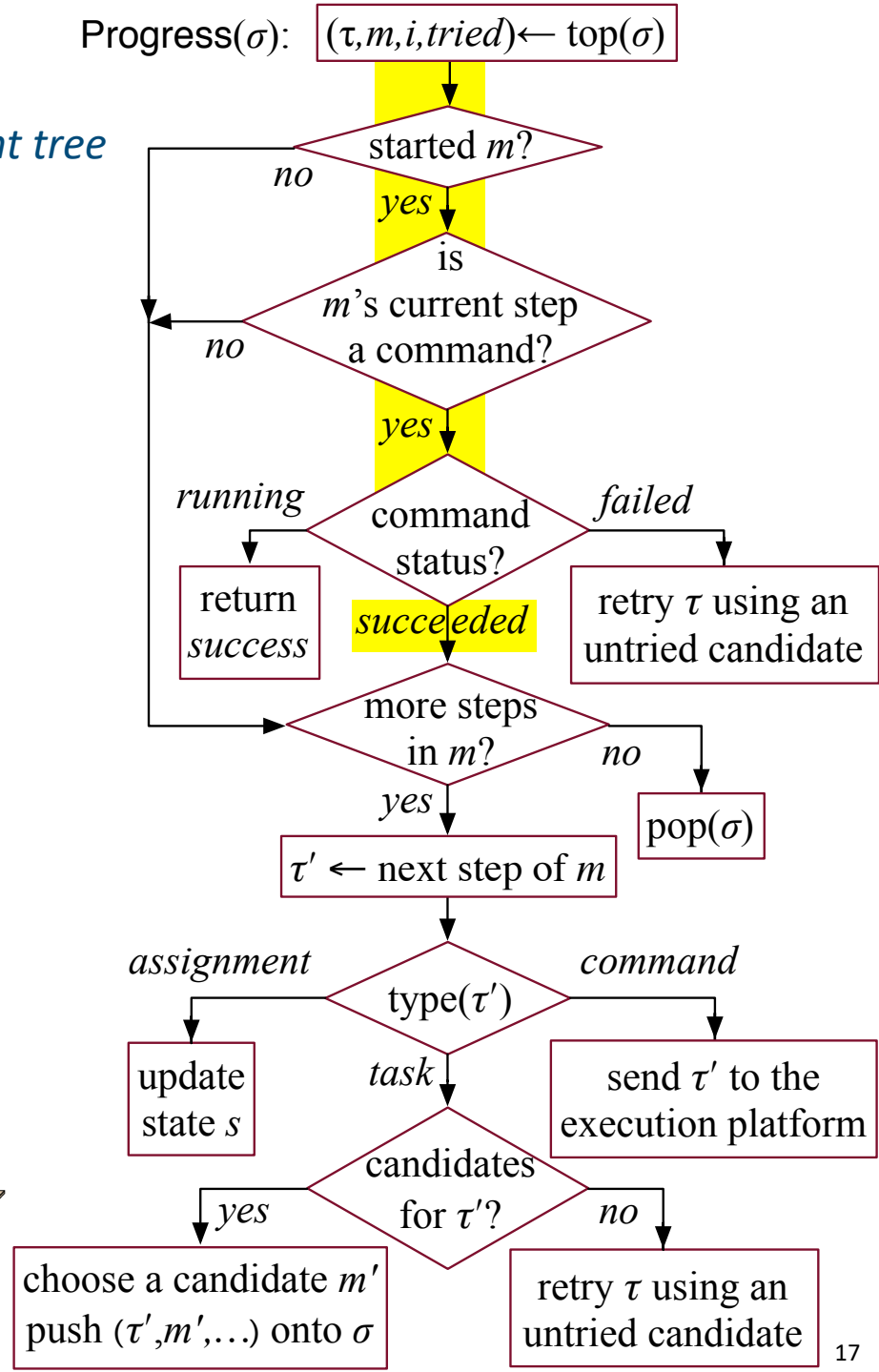
m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l) ← *succeeded*
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

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))

Refinement tree





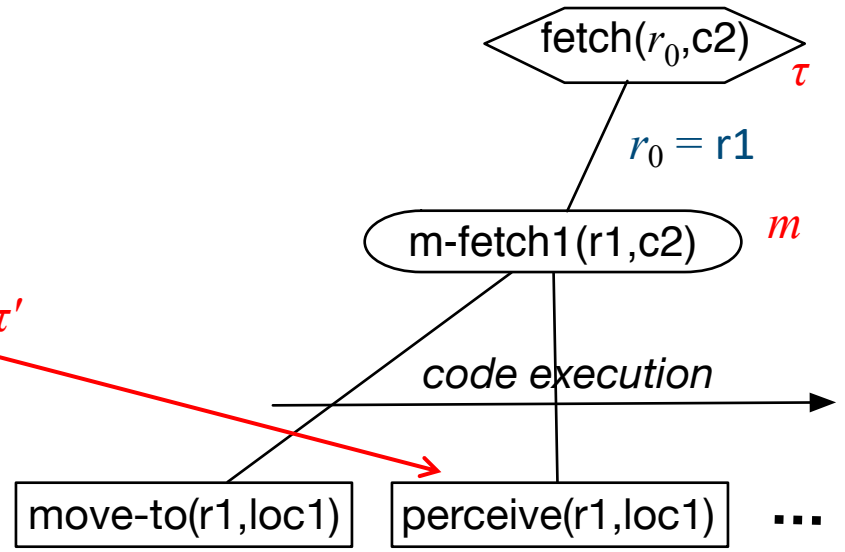


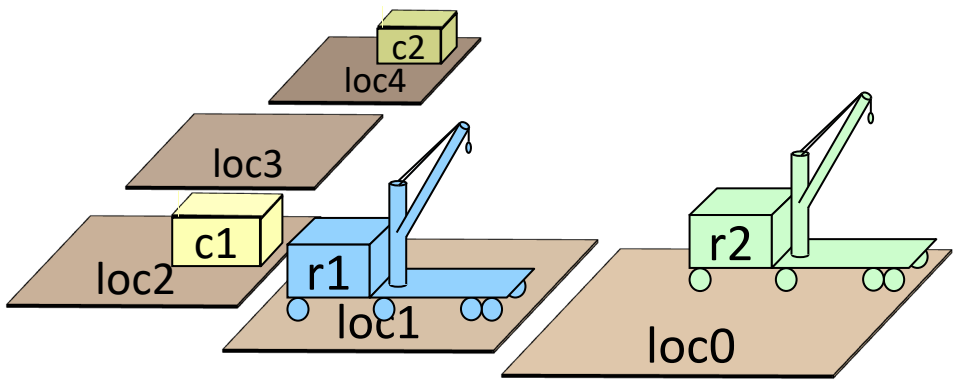
Example

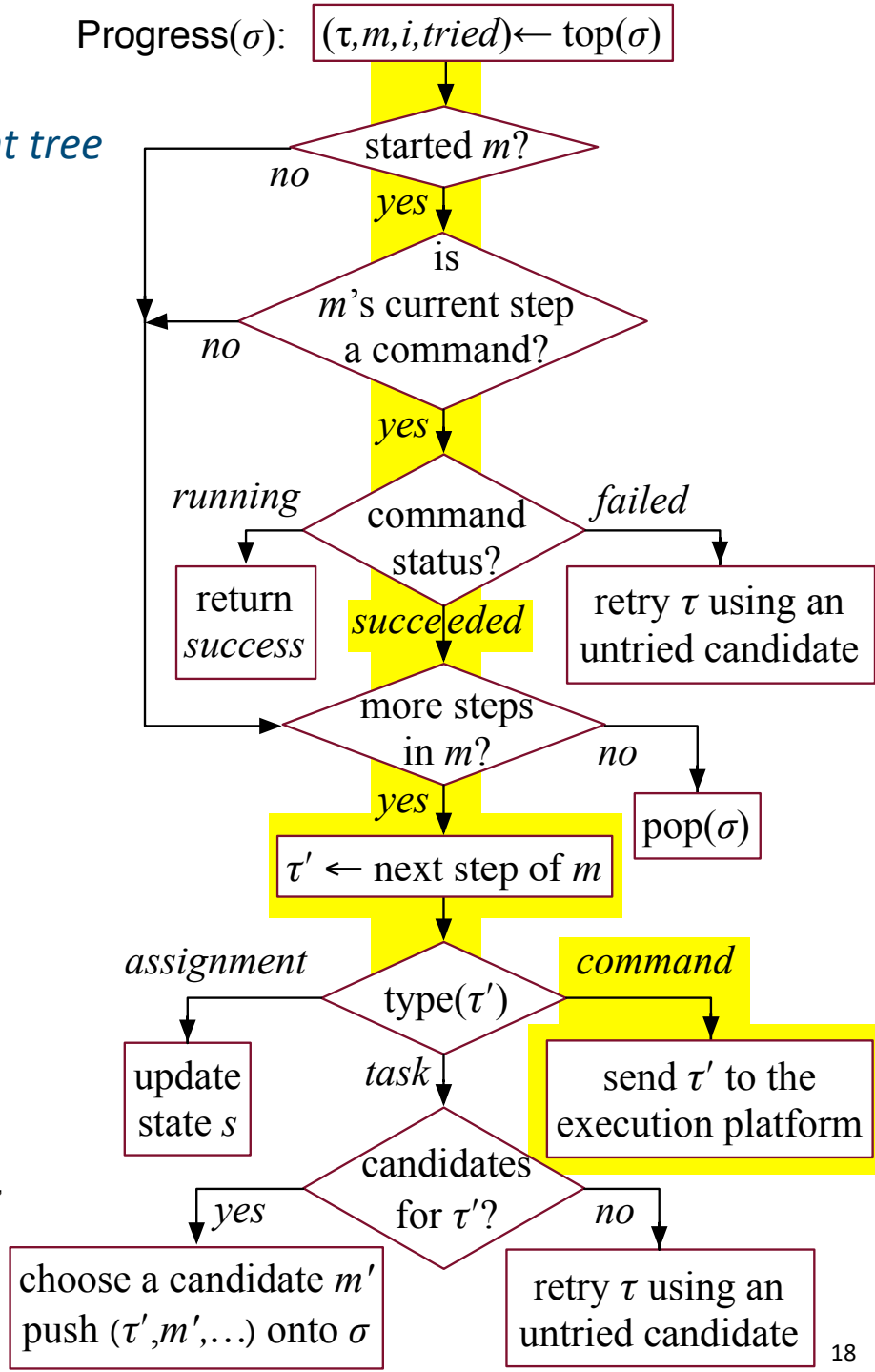
m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

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))

Refinement tree

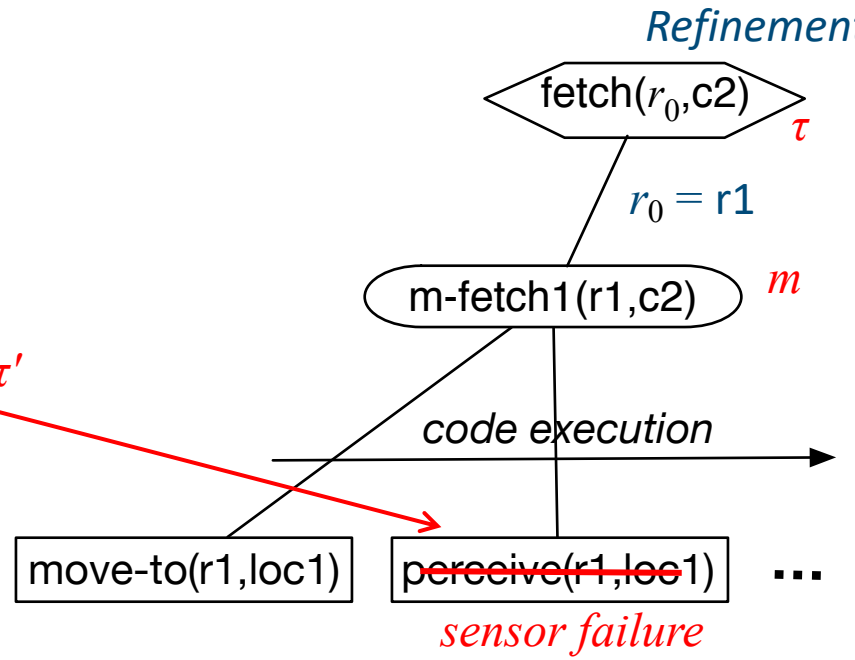




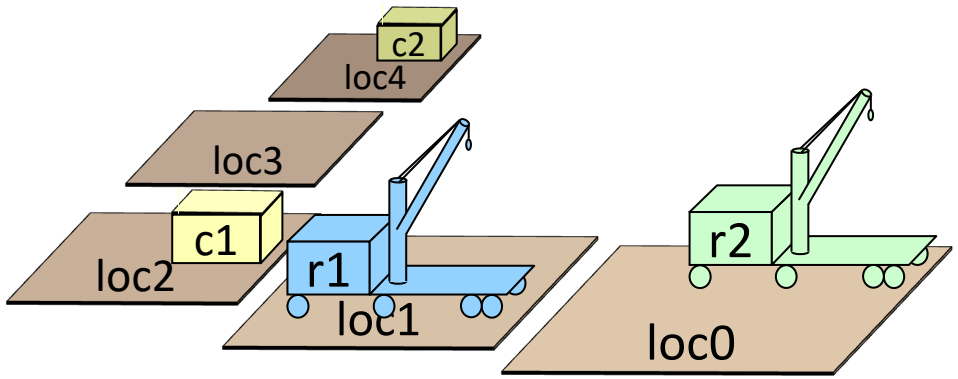


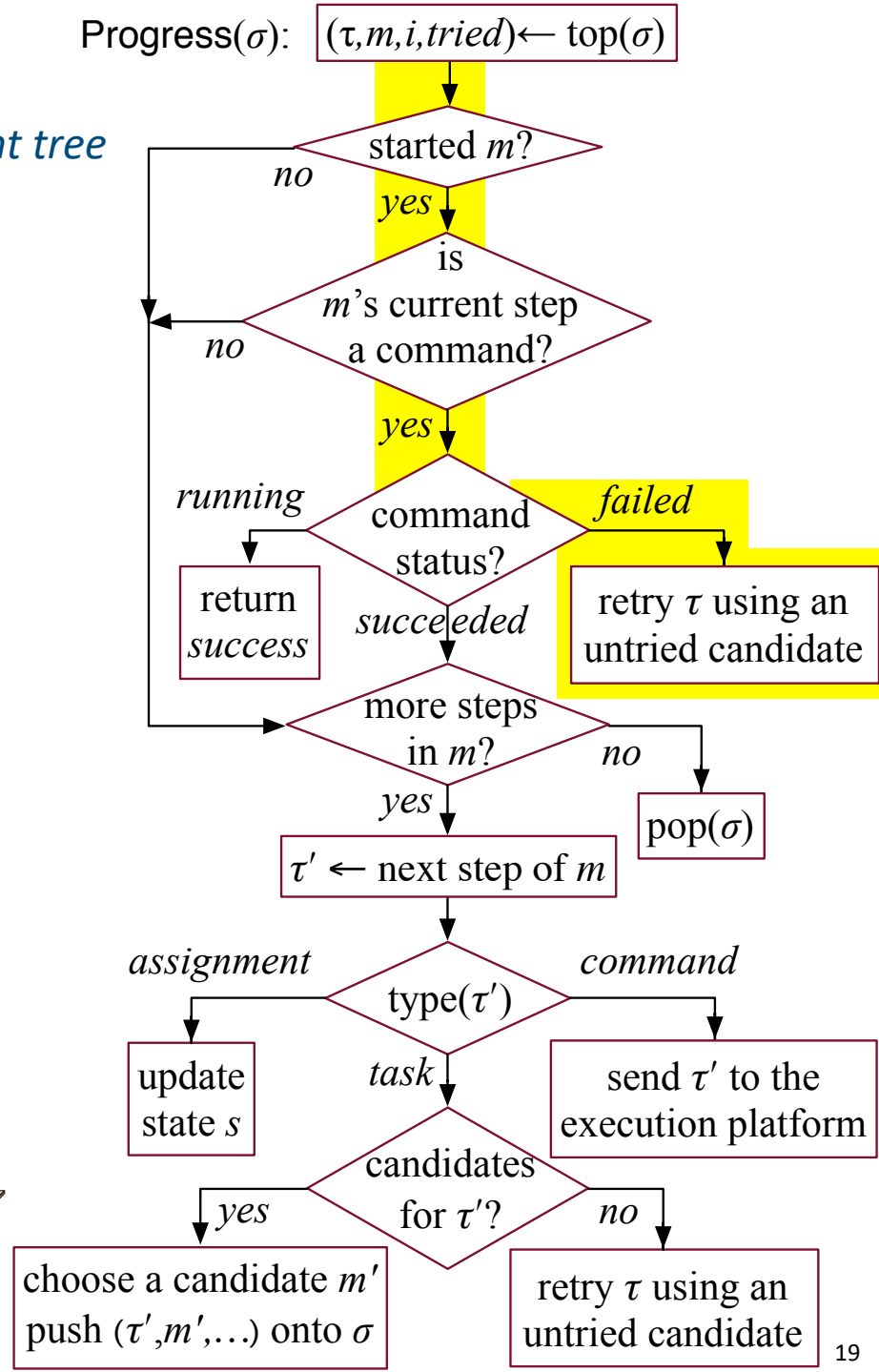
Example

m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l) ← *failed*
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail



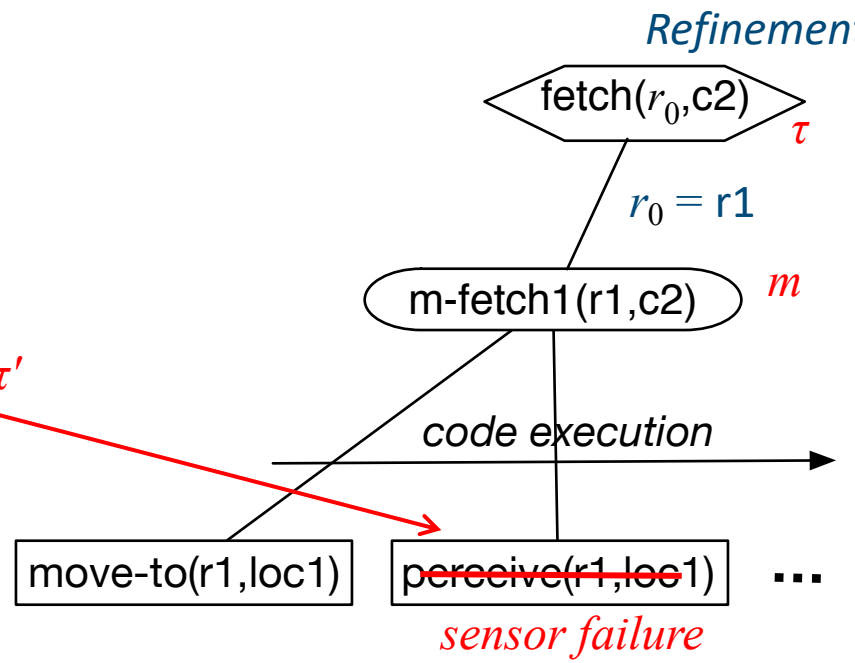
m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))



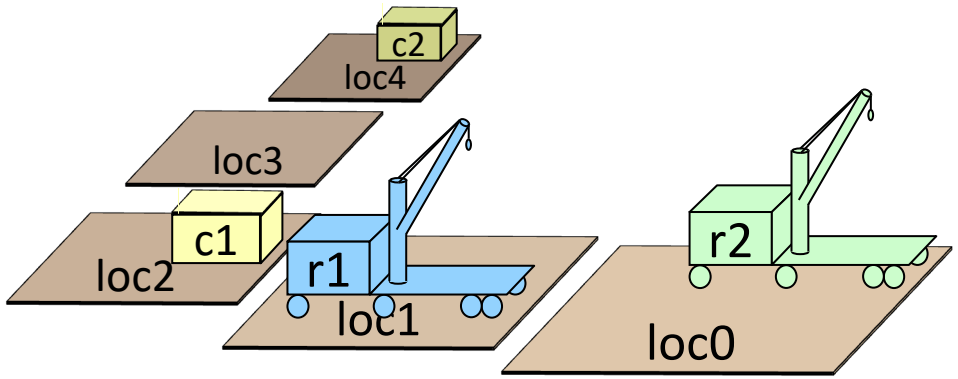


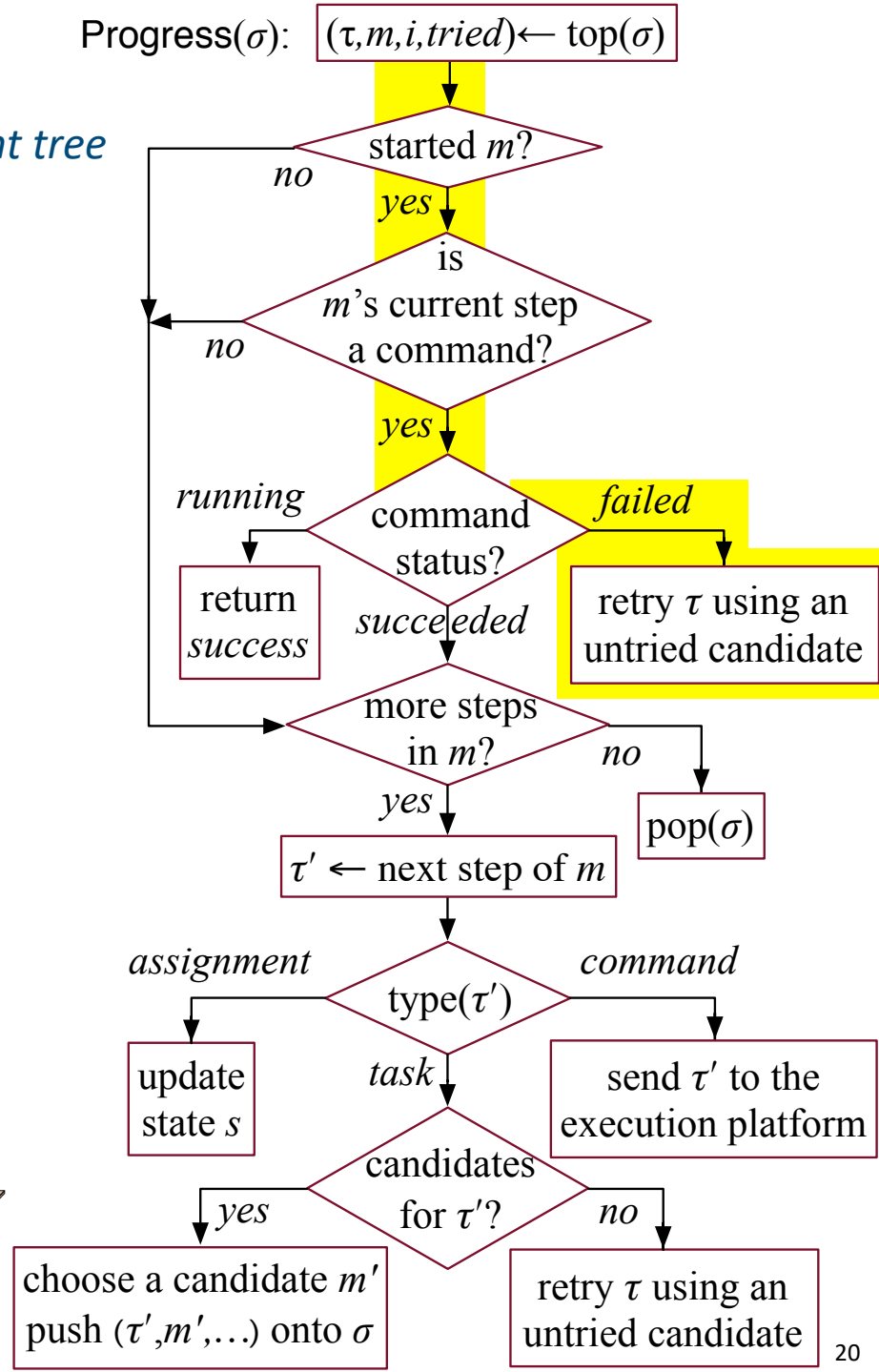
Example

m-fetch1(r,c) r = r1, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body: l = loc1
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l) ← *failed*
    - 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))



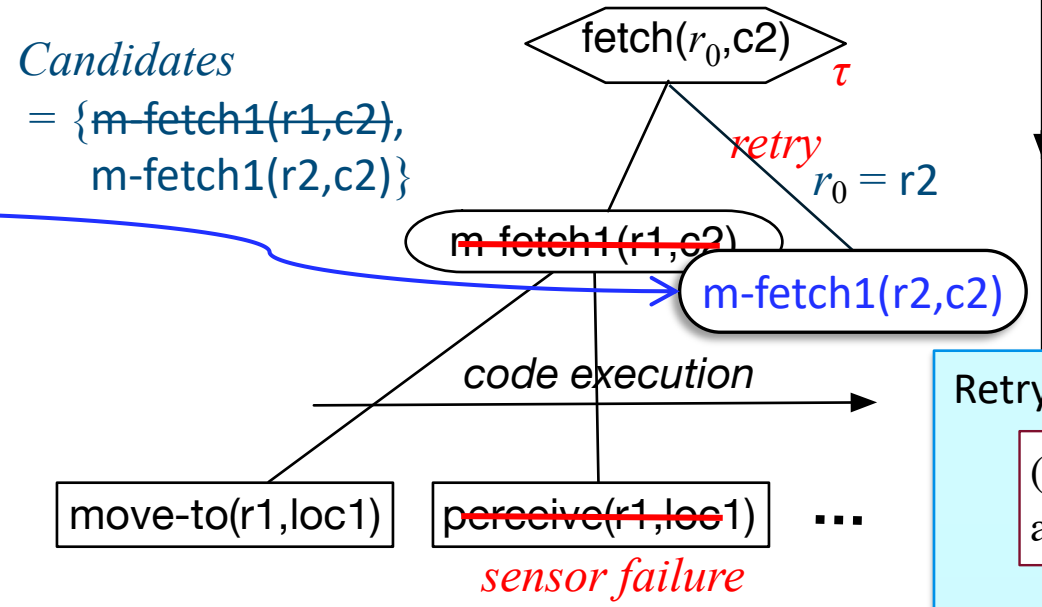


Example

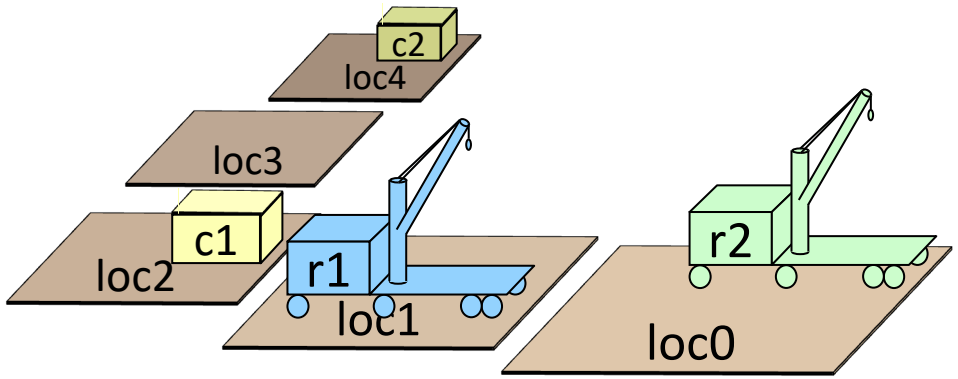
m-fetch1(r,c) r = r2, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body:
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

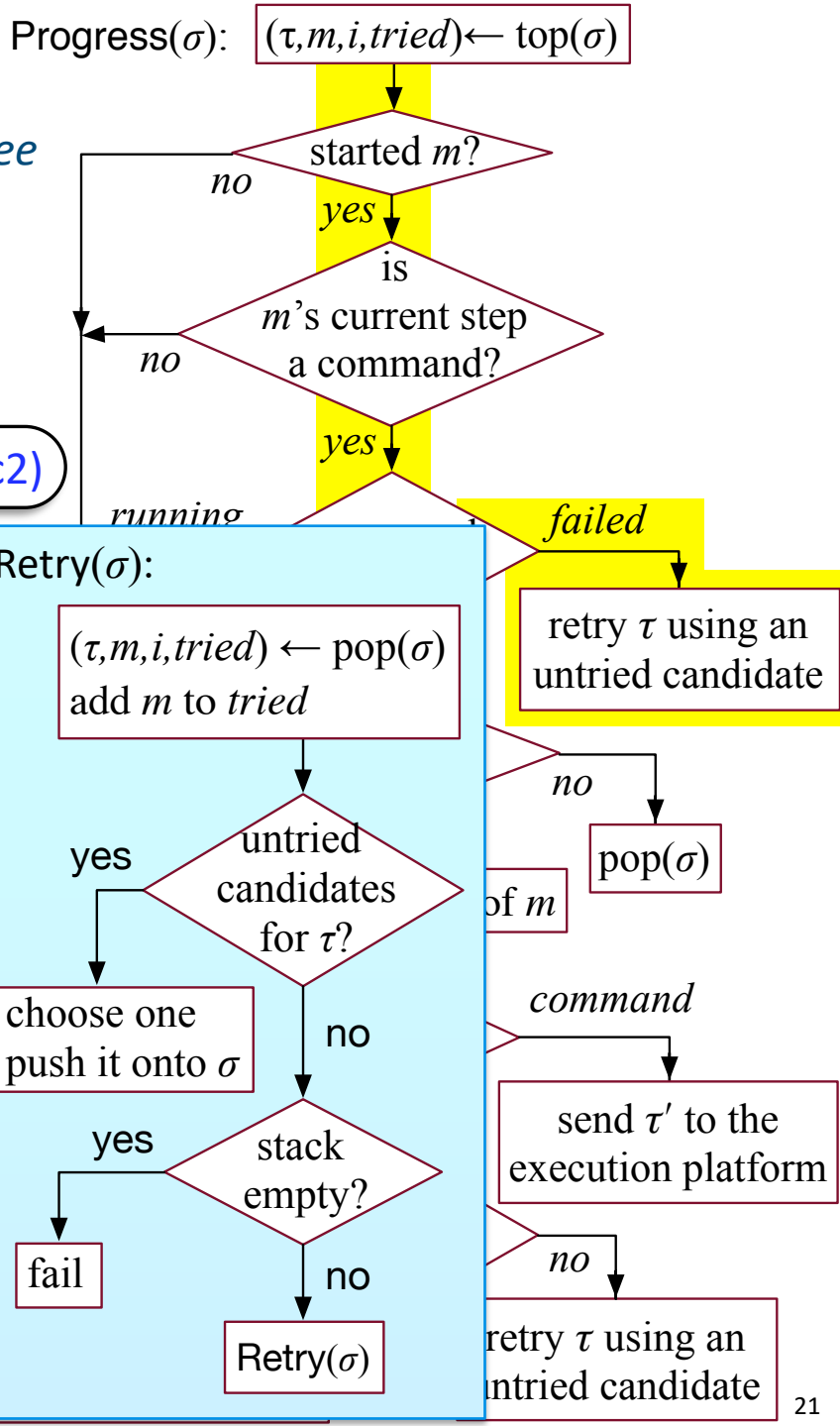
m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

Refinement tree



Candidates
= {~~m-fetch1(r1,c2)~~, m-fetch1(r2,c2)}



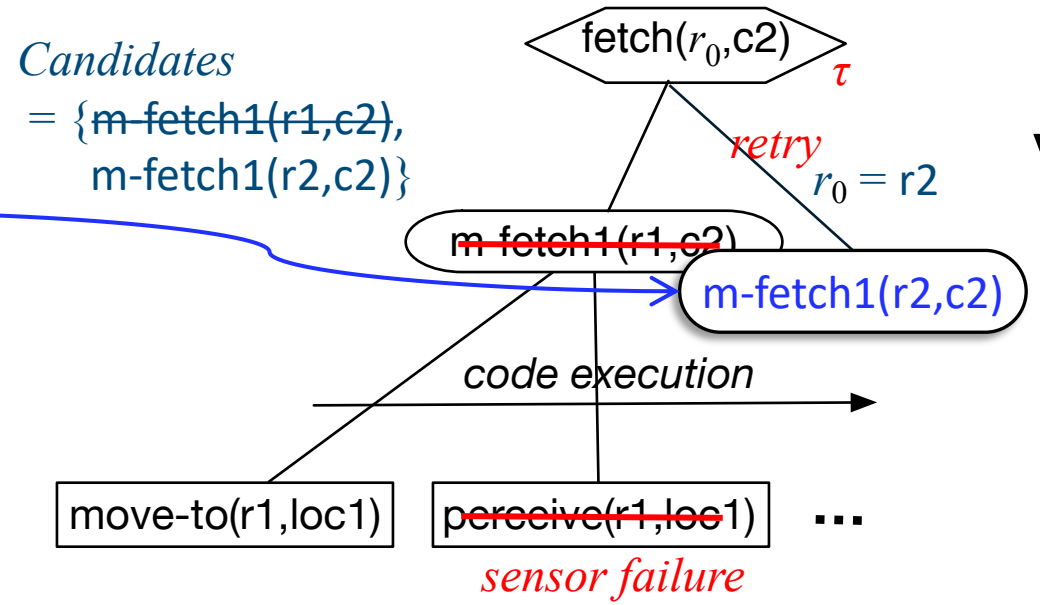


Example

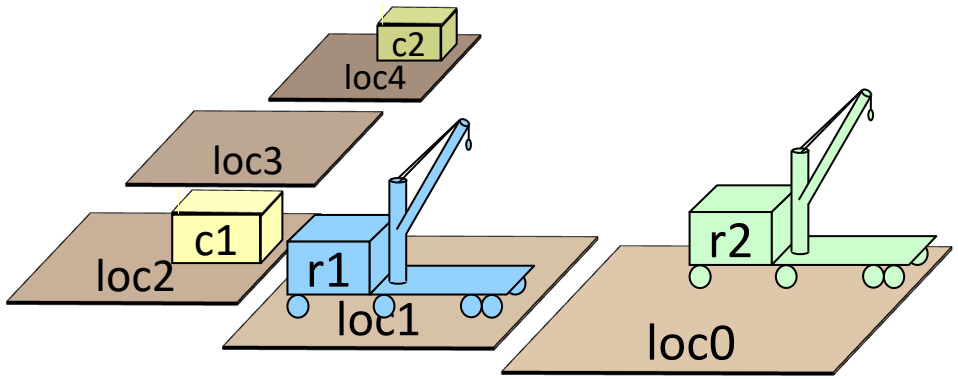
m-fetch1(r,c) r = r2, c = c2
- task: fetch(r,c)
- pre: pos(c) = unknown
- body:
  - if ∃l (view(l) = F) then
    - move-to(r,l)
    - perceive(r,l)
    - if pos(c) = l then
      - take(r,c,l)
    - else fetch(r,c)
  - else fail

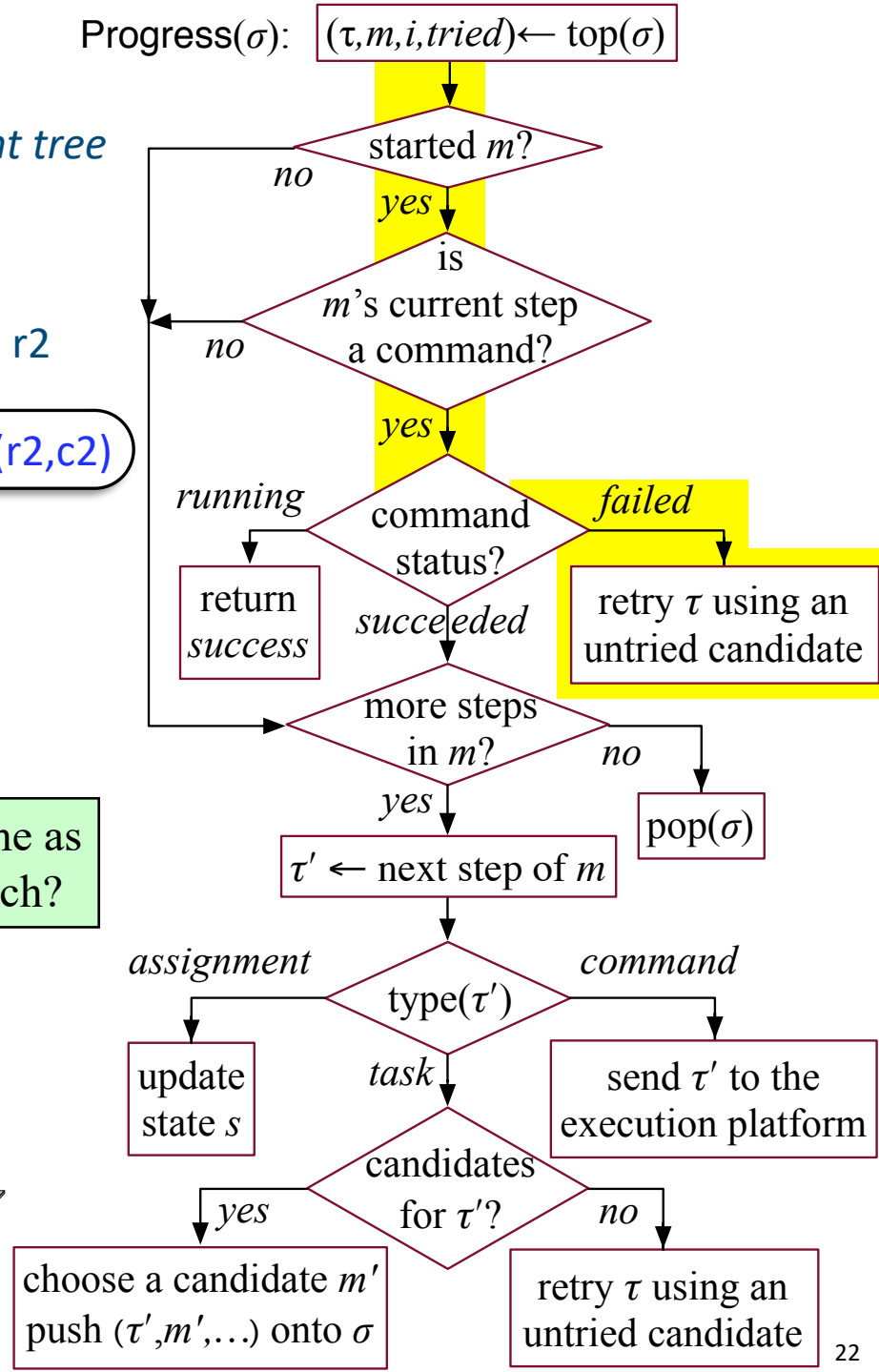
m-fetch2(r,c)
- task: fetch(r,c)
- pre: pos(c) ≠ unknown
- body:
  - if loc(r) = pos(c) then
    - take(r,c,pos(c))
  - else do
    - move-to(r,pos(c))
    - take(r,c,pos(c))

Refinement tree



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

Summary

- Section 14.1: Representation
  - Tasks, events, actions
  - Refinement methods
    - Extended version of HTN methods
    - name, task/event, preconditions, body
  - Example: fetch a container
- Sections 14.2-3: 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*
    - comparison to backtracking
  - Refinement trees