Lecture 13

CS 6260 Automated Planning and Learning

HTN Planning

Department of Computer Science and Engineering

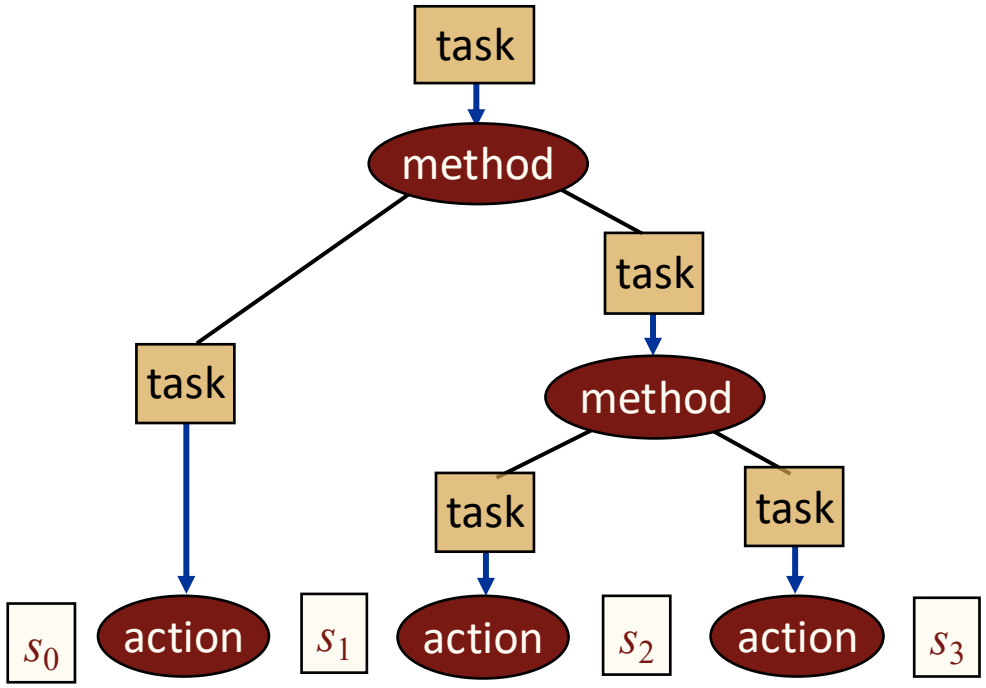
Indian Institute of Technology Madras

Some material based on lecture slides for Automated Planning and Acting by Nau/Roberts.

Recap: HTN Planning

Total-Order HTN Planning: Tasks

- Three kinds of tasks
  - *Primitive* task: head of an action
  - *Compound* task: *name*(*args*)
    - *name* is a *compound-task name*
  - *Goal task*: goal(*g*)
    - *g* is any classical goal formula



Total-Order HTN Planning: Tasks

- Method: a tuple
  ⟨*head*, *nonprimitive task*, *preconditions*, *subtasks*⟩

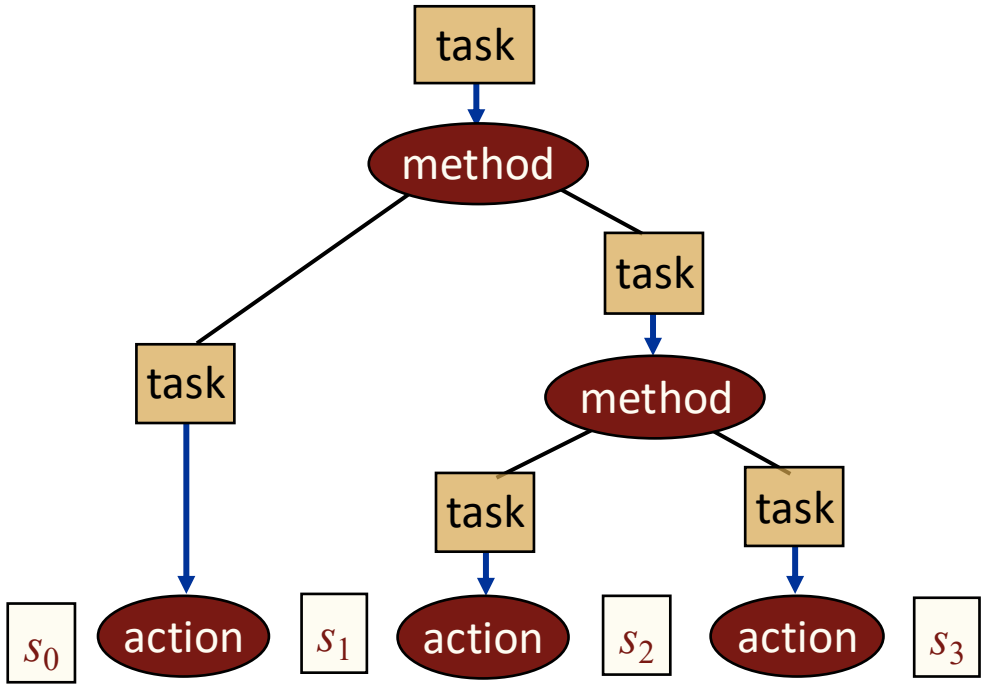
- Write it as pseudocode:

  method-name(*args*)

    Task: *nonprimitive task*

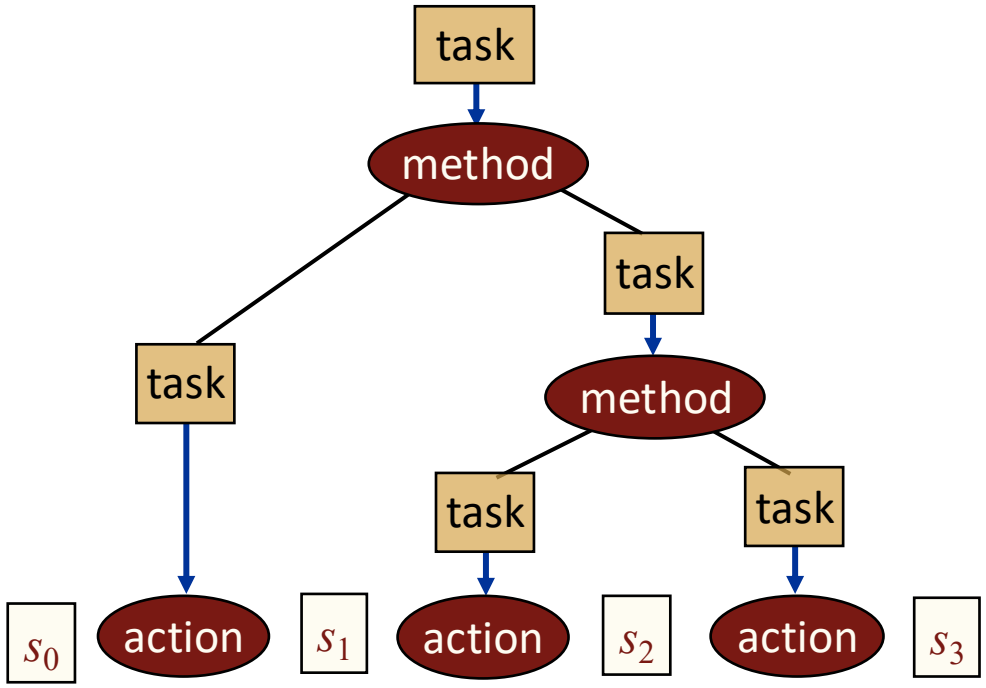
      Pre: *preconditions*

      Sub: *list of subtasks*

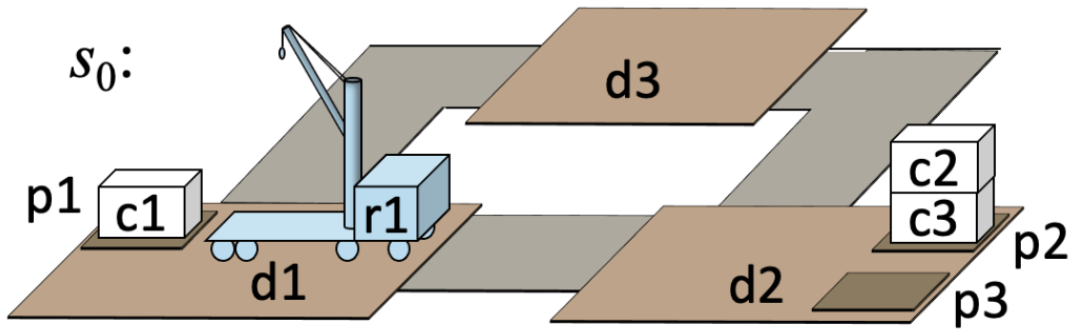


Total-Order HTN Planning

- TOHTN planning domain: a pair $(\Sigma, \mathcal{M})$
  - Σ: state-variable planning domain
  - $\mathcal{M}$: set of methods
- TOHTN planning problem $P = (\Sigma, \mathcal{M}, s_0, T)$
  - $T = \langle t_1, t_2, \ldots, t_k \rangle$
- *Solution* for P:
  - any executable plan that can be generated for T by applying
    - methods to nonprimitive tasks
    - actions to primitive tasks



Total-Order HTN Planning



- Objects:
  - robots r1, r2
  - loading docks d1, d2, d3
  - containers c1, c2, c3
  - piles p1, p2, p3
- Rigid relations:
  - adjacent = {(d1,d2), (d2,d1), (d2,d3), (d3,d2), (d3,d1), (d1,d3)};
  - at = {(p1, d1), (p2, d2), (p3, d2)}.
- State variables:
  - cargo(r) ∈ *Containers* ∪ {nil}
  - loc(r) ∈ *Docks*
  - occupied(d) ∈ {T, F}
  - pile(c) ∈ *Piles* ∪ {nil}
  - pos(c) ∈ *Robots* ∪ *Containers* ∪ {nil}
  - top(p) ∈ *Containers* ∪ {nil}
  - where r ∈ *Robots*, c ∈ *Containers*, p ∈ Piles

Ques 2: Complete the state s_0

s_0 = {
cargo(r1) = nil,
loc(r1) = d1,
occupied(d1)=T,
occupied(d2)=F,
occupied(d3)=F,
pile(c1) = p1,
pile(c2) = p2,
pile(c3) = p2,
pos(c1) = nil,
pos(c2) = c3,
pos(c3) = nil,
top(p1)= c1,
top(p2) = c2,
top(p3) = nil}

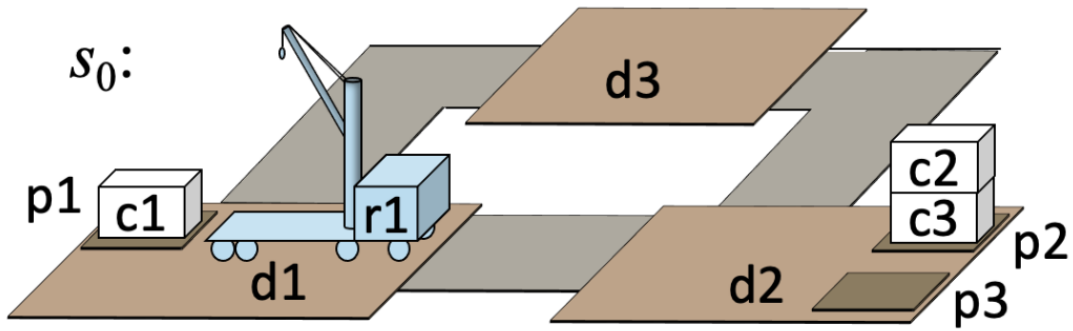
Total-Order HTN Planning

Action schemas:

- take(r, c, c', p, d)
  - pre: at(p,d), cargo(r) = nil, loc(r) = d, pos(c)= c', top(p) = c
  - eff: cargo(r) ← c, pile(c) ← nil, pos(c) ← r, top(p) ← c'
- put(r, c, c', p, d)
  - pre: at(p,d), pos(c) = r, loc(r) = d, top(p) = c'
  - eff: cargo(r) ← nil, pile(c) ← p, pos(c) ← c', top(p) ← c
- move(r, d, d')
  - pre: adjacent(d,d'), loc(r) = d, occupied(d') = F
  - eff: loc(r) ← d', occupied(d) ← F, occupied(d') ← T

where

- c ∈ *Containers*; c' ∈ *Containers* ∪ *Robots* ∪ {nil};
- d, d' ∈ *Docks*; p ∈ *Piles*; r ∈ *Robots*.

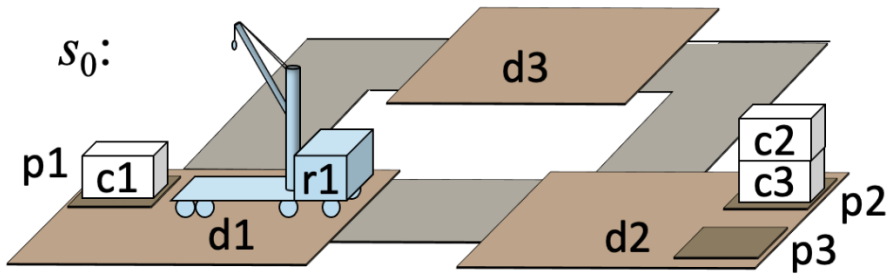


- State variables:
  - cargo(r) ∈ *Containers* ∪ {nil}
  - loc(r) ∈ *Docks*
  - occupied(d) ∈ {T, F}
  - pile(c) ∈ *Piles* ∪ {nil}
  - pos(c) ∈ *Robots* ∪ *Containers* ∪ {nil}
  - top(p) ∈ *Containers* ∪ {nil}
- where r ∈ *Robots*, c ∈ *Containers*, p ∈ Piles
- s_0 = {cargo(r1) = ?, cargo(r2) = ?,
  loc(r1) = ?, loc(r2) = ?,
  occupied(d1)=?, occupied(d2)=?, occupied(d3)=?,
  pile(c1) = ?, pile(c2) = ?, pile(c3) = ?,
  pos(c1) = ?, pos(c2) = ?, pos(c3) = ?,
  top(p1)= ?, top(p2) = ?, top(p3) = ?}

TOHTN Planning Domain

If I say "HTN" assume TOHTN unless stated otherwise

- TOHTN planning domain $\Sigma = (\Sigma_c, \mathcal{M})$
  - Σ_c = DWR domain on the previous pages
  - $\mathcal{M}$ = a set of eight methods:



- Compound task put-in-pile(*r*,*c*,*p*,*d*): put container c into pile *p* if it isn't there already

m1-put-in-pile(r, c, p, d)
task: $\{\text{pile}(c) = p\}$
pre: $\text{at}(p, d), \text{pile}(c) \neq p, \text{cargo}(r) = \text{nil}$
sub: $\text{get-container}(r, c), \text{navigate}(r, d), \text{put}(r, c, \text{top}(p), p, d)$

  - Preconditions:
    - 1st one ensures *d* has the correct value
    - Others check for applicability
  - Last subtask: one of the args is a state variable, top(p)
    - Violates a restriction in Chapter 2
    - But many HTN algorithms don't need the restriction

TOHTN Planning Domain (continued)

- Goal task: goal(cargo(*r*)=c)
  - Get *c* onto *r*
  - Subtask of m2-put-in-pile
- We aren't doing classical planning, so we need a method:

m1-get-container(r, c)
  task: get-container(r, c)
  pre: cargo$(r) = c$
  sub: *// no subtasks*

m2-get-container(r, c, p, d)
  task: get-container(r, c)
  pre: cargo(r) = nil, pile$(c) = p$, at(p, d)
  sub: navigate(r, d), uncover(c),
  take$(r, c,$ pos$(c), p, d)$

- Compound task uncover(c):
  - Subtask of m1-fetch
- Remove any containers that may be piled on top of c

m1-uncover(c)
  task: uncover(c)
  pre: top(pile$(c)) = c$
  sub: *// no subtasks*

m2-uncover(r, c, p, c', p', d)
  task: uncover(c)
  pre: pile$(c) = p$, top$(p) = c'$, $c' \neq c$,
  at(p, d), at(p', d), $p \neq p'$,
  loc$(r) = d$, cargo(r) = nil
  sub: take$(r, c',$ pos$(c'), p, d)$,
  put$(r, c',$ top$(p'), p', d)$,
  uncover(c)

TOHTN Planning Domain (continued)

- Compound task navigate(*r*,*d*):
  - Get robot *r* from current location to *d*
  - May require several move actions
- These methods are just for illustration, I don't recommend using them
  - Use a route planner instead
- Method for the case where loc(*r*) = *d*

m1-navigate(r, d)
task: navigate(r, d)
pre: loc$(r) = d$
sub: — // *no subtasks*

- Method for cases where loc(*r*) is adjacent to *d*

m2-navigate(r, d', d)
task: navigate(r, d)
pre: adjacent(d', d), loc$(r) = d'$
sub: move(r, d', d)

- Method for cases where loc(*r*) isn't adjacent to *d*

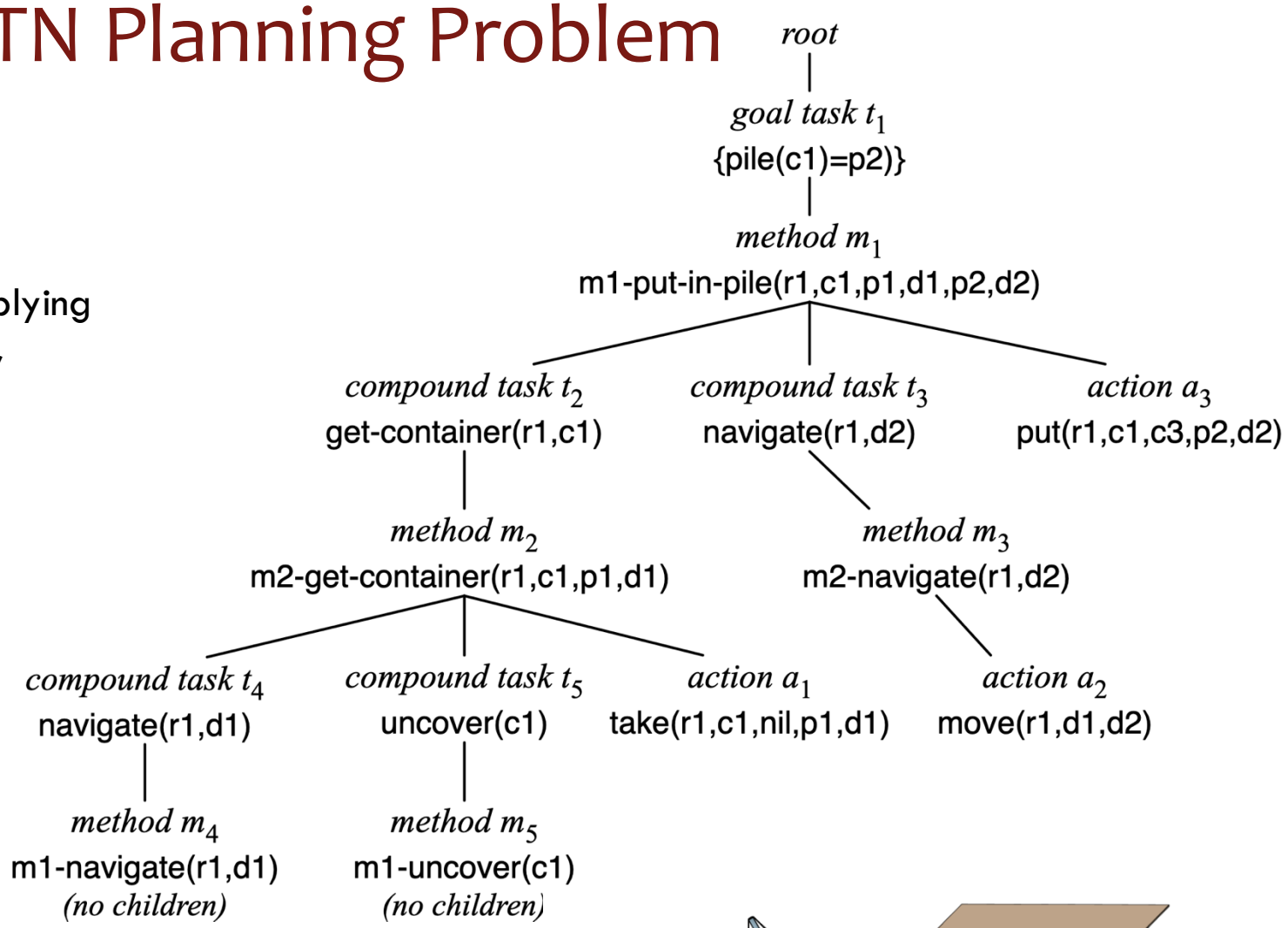
m3-navigate(r, d', d)
task: navigate(r, d)
pre: loc$(r) \neq d$, ¬adjacent(loc(r), d), adjacent(loc(r), d')
sub: move$(r,$ loc$(r), d')$ // *primitive task*
navigate(r, d) // *compound task*

- Methods in $\mathcal{M}$:
  - m1-put-in-pile, m2-put-in-pile, m1-get-container, m2-get-container, m1-uncover, m2-uncover, m1-navigate, m2-navigate, m3-navigate
- Tasks in Σ:
  - Primitive: all instances of
    - move(*r*,*l*,*m*), take(*r*,*c*,*l*), put(*r*,*c*,*l*)
  - Compound: all instances of
    - put-in-pile(*c*,*p*), uncover(*c*), and navigate(*r*,*d*)
  - Goal tasks: all instances of goal(cargo(*r*) = *c*)

HTN Planning Problem and Solution

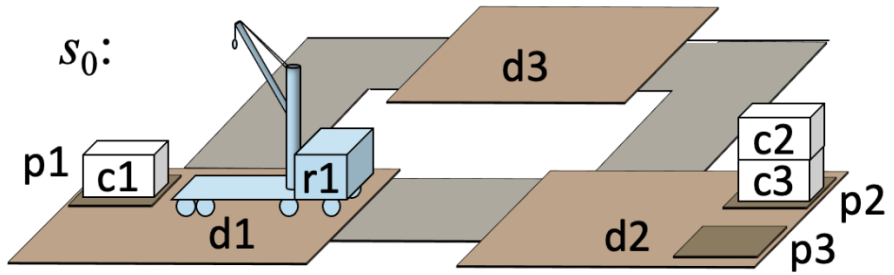
TOHTN Planning Problem

- *Planning problem:* $P = (\Sigma, s_0, T)$
- *Solution*
  - any executable plan produced by applying method instances to nonprimitive tasks, actions to primitive tasks



$P = (\Sigma, s_0, \langle\{\text{pile(c1)=p2}\}\rangle)$

$\pi = \langle$take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$



TOHTN Planning Problem

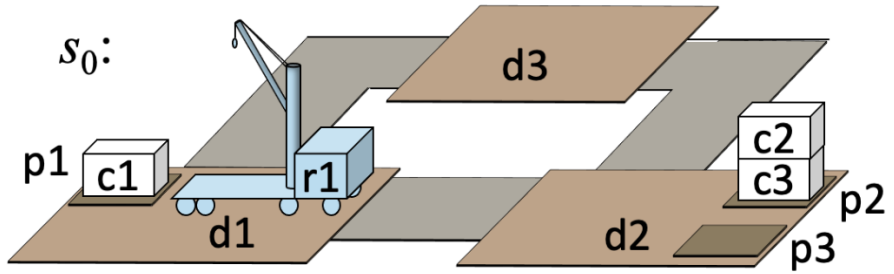
Planning problem: $P = (\Sigma, s_0, T)$

Solution

- If T is empty, then the empty plan $\langle\rangle$ is a solution for P.
- Else, let t be the first task of T, so that $T = t \cdot T'$
  where T' is a (possibly empty) sequence of tasks.
- If t is an action in Applicable($s0$), then for every solution π for the problem (Σ, $\gamma(s0, t)$, T'), the plan $t \cdot \pi$ is a solution for P.
- If t is a compound task or goal task and $m \in$ Methods($s0$, t, M), then every solution for the problem (Σ, $s0$, subtasks(m) $\cdot T'$) is also a solution for P.
- If t is a goal task and $a \in$ Actions($s0$, t), then for every solution π for the problem (Σ, $\gamma(s0, a)$, T'), the plan $a \cdot \pi$ is a solution for P.
- If t is a goal task and $s0 \models t$, then every solution for the problem (Σ, $s0$, T') is also a solution for P.

$P = (\Sigma, s_0, \langle\{\text{pile(c1)=p2}\}\rangle)$

$\pi = \langle$take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$



TOHTN Planning Problem

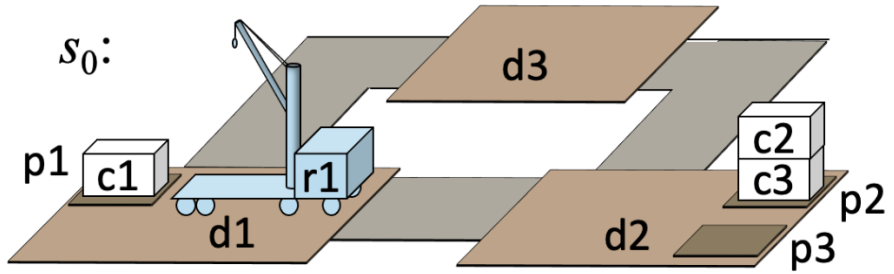
Planning problem: $P = (\Sigma, s_0, T)$

Solution

- If T is empty, then the empty plan $\langle\rangle$ is a solution for P.
- Else, let t be the first task of T, so that $T = t \cdot T'$
  where T' is a (possibly empty) sequence of tasks.
- If t is an action in Applicable($s0$), then for every solution π for the problem (Σ, $\gamma(s0, t)$, T'), the plan $t \cdot \pi$ is a solution for P.
- If t is a compound task or goal task and $m \in$ Methods($s0$, t, M), then every solution for the problem (Σ, $s0$, subtasks(m) $\cdot T'$) is also a solution for P.
- If t is a goal task and $a \in$ Actions($s0$, t), then for every solution π for the problem (Σ, $\gamma(s0, a)$, T'), the plan $a \cdot \pi$ is a solution for P.
- If t is a goal task and $s0 \models t$, then every solution for the problem (Σ, $s0$, T') is also a solution for P.

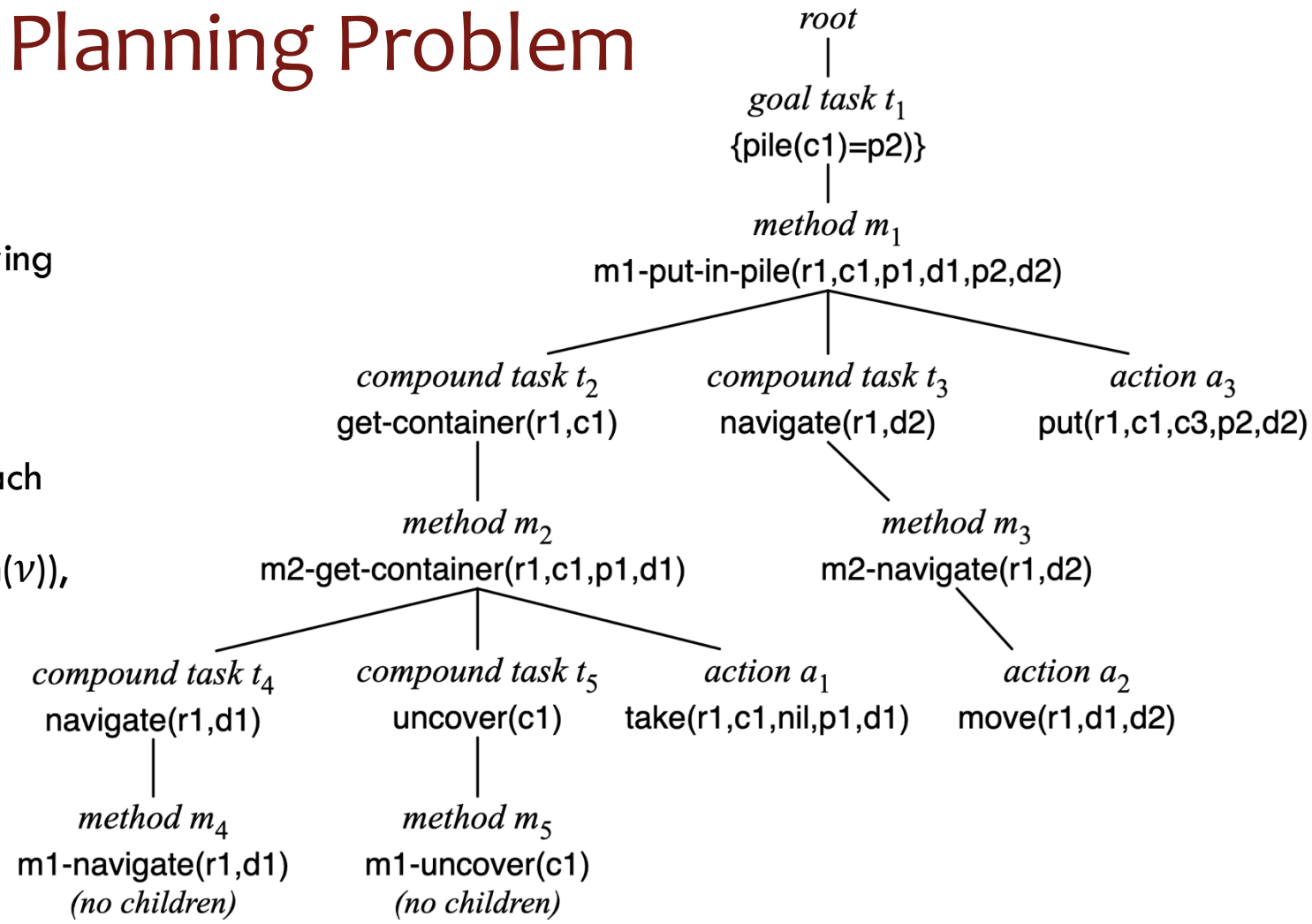
$P = (\Sigma, s_0, \langle\{\text{pile(c1)=p2}\}\rangle)$

$\pi = \langle$take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$



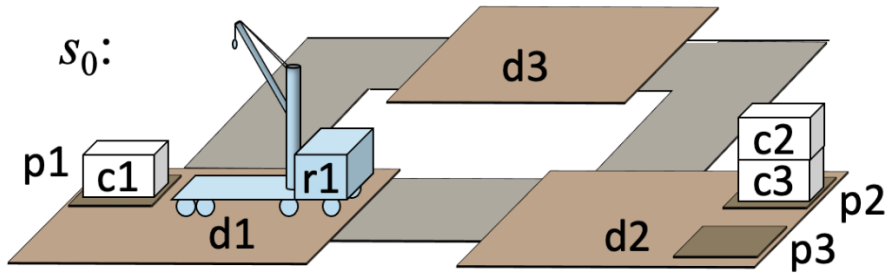
TOHTN Planning Problem

- *Planning problem:* $P = (\Sigma, s_0, T)$
- *Solution* any executable plan produced by applying method instances to nonprimitive tasks, actions to primitive tasks
- *Refinement tree*
  - A refinement tree for π is a tree in which each node is a tuple
    ν = (label(ν), content(ν), parent(ν), Children(ν)),
  - Nodes:
    root, compound tasks, goal tasks,
    method instances, actions
  - Edges:
    root → top-level tasks,
    task → method instance,
    method instance → subtasks



$P = (\Sigma, s_0, \langle\{\text{pile(c1)=p2}\}\rangle)$

$\pi = \langle$take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)$\rangle$

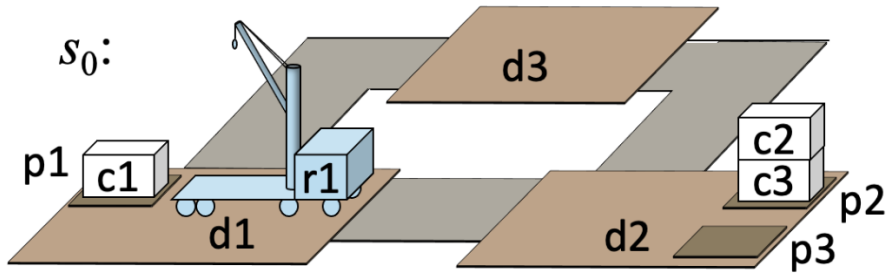


TOHTN Planning Problem

- *Planning problem:* $P = (\Sigma, s_0, T)$
- *Solution* any executable plan produced by applying method instances to nonprimitive tasks, actions to primitive tasks
- *Refinement tree*
  - A refinement tree for π is a tree in which each node is a tuple $v = (\text{label}(v), \text{content}(v), \text{parent}(v), \text{Children}(v))$,
  - Nodes:
    root, compound tasks, goal tasks,
    method instances, actions
  - Edges:
    root → top-level tasks,
    task → method instance,
    method instance → subtasks

$P = (\Sigma, s_0, \langle\{\text{pile(c1)=p2}\}\rangle)$

$\pi = \langle\text{take(r1,c1,c2,p1,d1), move(r1,d1,d2), put(r1,c1,c3,p2,d2)}\rangle$



Planning Algorithm

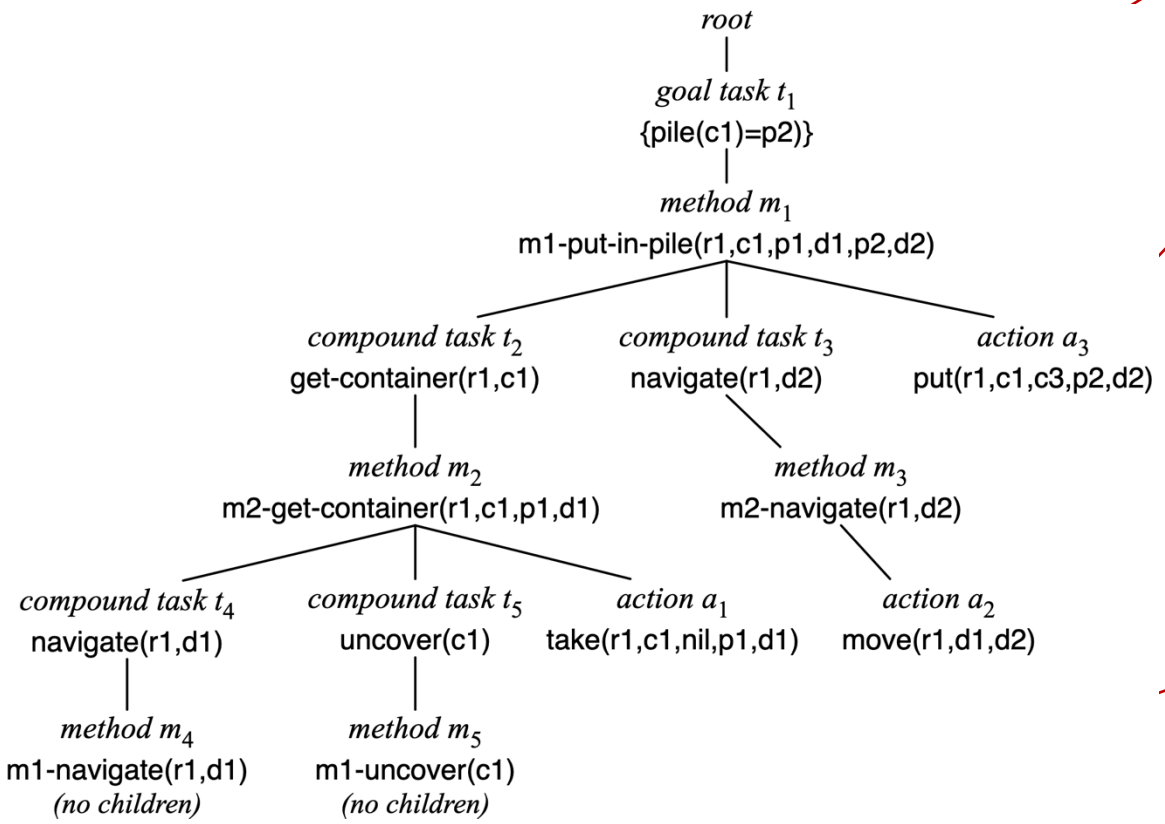
- Definitions

$$Achievers(s,t) = \{a \in Applicable(s) \mid \gamma(s,a) \models t\}$$

$$Ground(\mathcal{M}) = \{\text{all ground instances of methods in } \mathcal{M}\}$$

$$Refiners(s,t,\mathcal{M}) = \{m \in Ground(\mathcal{M}) \mid t \text{ is refinable by } m \text{ in } s\}$$

- Three cases: primitive, compound, goal task
  - Primitive task: apply action



TO-HTN-Forward($\Sigma_c, \mathcal{M}, s, T$)
  if T is empty **then return** $\langle\rangle$
  $t \leftarrow$ the first element of T; $T' \leftarrow$ the rest of T
1 $M \leftarrow$ HTN-Get-Candidates($\Sigma_c, \mathcal{M}, s, t$)
  if $M = \varnothing$ **then return** failure
  nondeterministically choose $m \in M$
  switch m **do**
2 **case** m is an action **do**
    $\pi \leftarrow$ TO-HTN-Forward($\Sigma_c, \mathcal{M}, \gamma(s,m), T'$)
    if $\pi \neq$ failure **then return** $m \cdot \pi$
    else return failure
3 **case** m is a method instance **do**
    return TO-HTN-Forward($\Sigma_c, \mathcal{M}, s, \text{sub}(m) \cdot T'$)

HTN-Get-Candidates($\Sigma_c, \mathcal{M}, s, t$)
  switch t **do**
4 **case** t is an action **do**
    if t is applicable in s **then** $M \leftarrow \{a\}$
    else $M \leftarrow \varnothing$
5 **case** t is a compound task **do** $M \leftarrow Refiners(s,t,\mathcal{M})$
6 **case** t is a goal task **do**
    $M \leftarrow Refiners(s,t,\mathcal{M}) \cup Achievers(s,t)$
7 **if** $s \models t$ **then** $M \leftarrow M \cup \{\text{null}\}$
  return $\mathcal{M}$

Planning Algorithm

- Most implementations do depth-first
  - Can use heuristic function, but the ones in Chapter 3 will probably need modification
  - Primitive task: apply action

state s; $T = \langle t, t_2, \ldots, t_k \rangle$

new state $\gamma(s,t)$; $T = \langle t_2, \ldots, t_k \rangle$

- Compound task: apply all method instances

state s; $T = \langle t, t_2, \ldots, t_k \rangle$

method instance m

$\langle u_1, \ldots, u_j, t_2, \ldots, t_k \rangle$

- Goal task: apply all method instances and actions

TO-HTN-Forward-Det($\Sigma_c, \mathcal{M}, s_0, T_0$)
  $Frontier \leftarrow \{(\langle\rangle, s_0, T_0)\}$ // {initial node}
  $Expanded \leftarrow \varnothing$
  while $Frontier \neq \varnothing$ **do**
    select a node $v = (\pi, s, T) \in Frontier$ 1
    remove v from $Frontier$ and add it to $Expanded$
    if $T = \langle\rangle$ **then return** π
    $t \leftarrow$ the first element of T; $T' \leftarrow$ the rest of T
    switch t **do**
      case t is an action **do**
        if t is applicable in s **then** $Children \leftarrow \{(\pi \cdot t, \gamma(s,t), T')\}$
        else $Children \leftarrow \varnothing$
      case t is a compound task **do**
        $Children \leftarrow \{(\pi, s, \text{sub}(m) \cdot T') \mid m \in Refiners(s,t,\mathcal{M})\}$
      case t is a goal task **do**
        $Children \leftarrow \{(\pi \cdot a, \gamma(s,a), T') \mid a \in Achievers(s,t)\} \cup$
                $\{(\pi, s, \text{sub}(m) \cdot T') \mid m \in Refiners(s,t,\mathcal{M})\}$
        if $s \models t$ **then** $Children \leftarrow Children \cup \{(\pi, s, T')\}$ 2
    prune 0 or more nodes from $Children$, $Frontier$ and $Expanded$
    $Frontier \leftarrow Frontier \cup Children$
  return failure