Lecture 12

CS 6260 Automated Planning and Learning

Hierarchical Task Networks (HTNs)

Department of Computer Science and Engineering

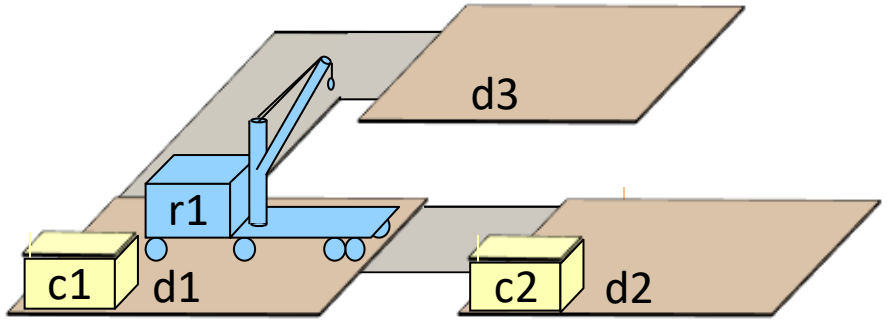
Indian Institute of Technology Madras

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

Recap:
State-Variable Representation

State-Variable Representation

- *Objects* = {names of objects in the environment}
- Organized into a *typed ontology*
  - sets of object types
- *Objects* = *Robots* ∪ *Containers* ∪ *Locs* ∪ {nil}
  - *Robots* = {r1}
  - *Containers* = {c1, c2}
  - *Locs* = {d1, d2, d3}

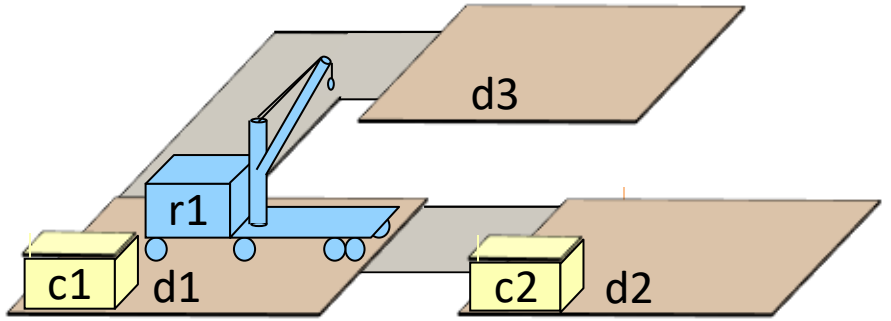


- *Objects* only needs to include objects that matter at the current level of abstraction
- Can omit lots of details
  - physical characteristics of robots, containers, loading docks, roads, …
- Objects have two kinds of properties
  - *rigid* and *varying*

States as Functions

- Represent each state *s* as a function that assigns values to state variables
  - For each state variable *x*, *s*(*x*) is one *x*'s possible values

s_1(loc(r1)) = d1, s_1(cargo(r1)) = nil,
s_1(loc(c1)) = d1, s_1(loc(c2)) = d2



- Mathematically, a function is a set of ordered pairs

s_1 = {(loc(r1), d1), (cargo(r1), nil), (loc(c1), d1) , (loc(c2), d2)}

- Equivalently, write it as a set of *ground positive literals* (or *ground atoms*):

s_1 = {loc(r1)=d1, cargo(r1)=nil, loc(c1)=d1, loc(c2)=d2}

  - Here, '=' is a predicate symbol
  - Assignment will be indicated with '←'

Suppose r1 can move between locations. What does the state need to include to represent that move(r1, d1, d2) is valid while move(r1, d2, d3) is invalid?

(short answer)

HTN Planning

HTN Planning

- For some planning problems, we may already have ideas for how to look for solutions

- Example: travel to a destination that's far away:
  - Brute-force search: many combinations of vehicles and routes

- Experienced human: small number of "recipes"

  e.g., flying:

  1. buy ticket from local airport to remote airport
  2. travel to local airport
  3. fly to remote airport
  4. travel to final destination

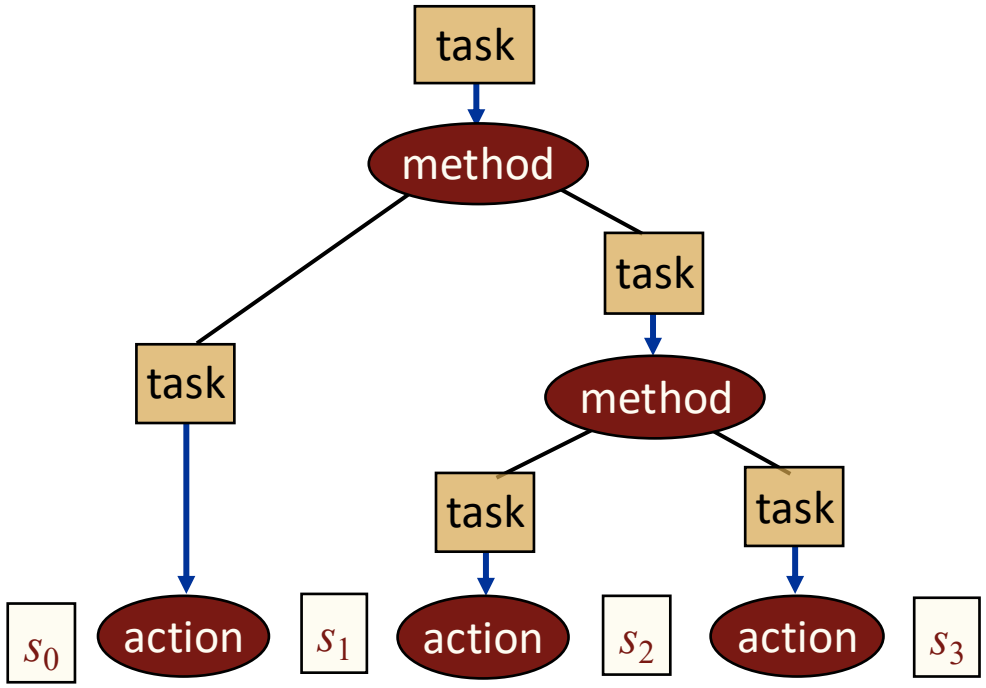
HTN Planning

- Two ways to put such information into a planner:
  - Domain-specific algorithm
  - Domain-independent planning engine + domain-specific planning information
    - HTN planning (this Part)

- Ingredients:
  - state-variable planning domain (Part I)
  - tasks: activities to perform
  - HTN methods: ways to perform tasks

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*⟩

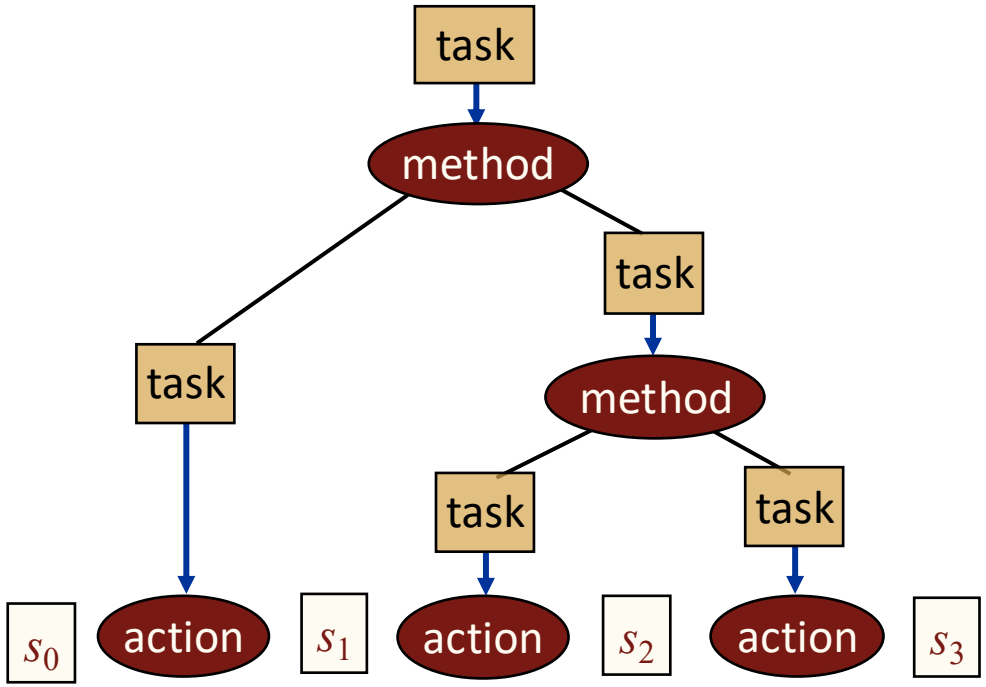
- Write it as pseudocode:

  method-name(*args*)

    Task: *nonprimitive task*

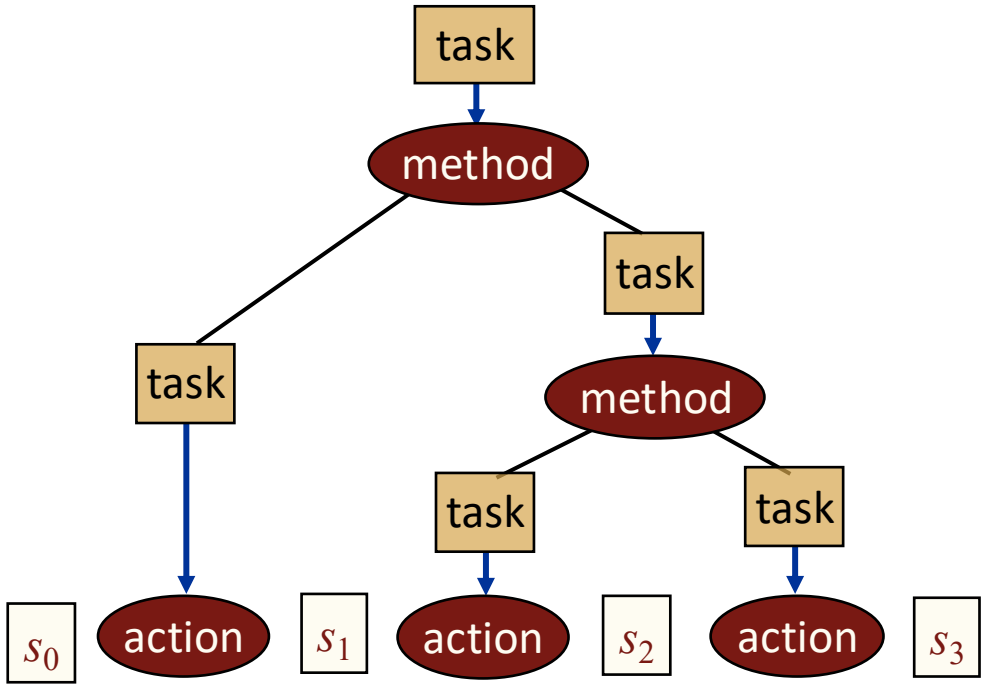
      Pre: *preconditions*

      Sub: *list of subtasks*

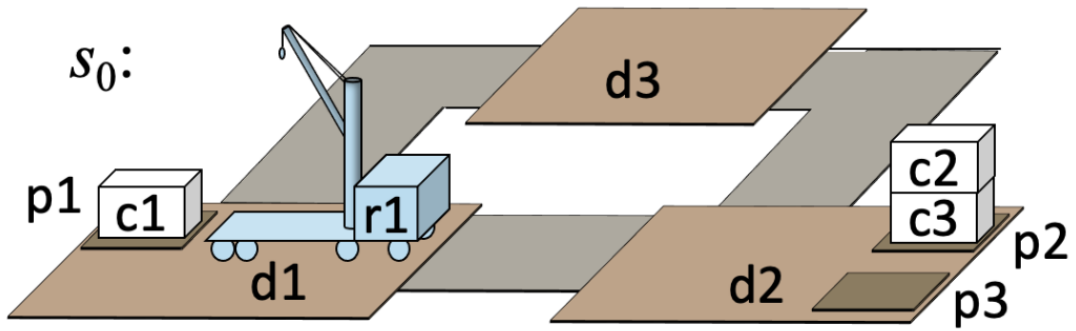


Total-Order HTN Planning

- 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

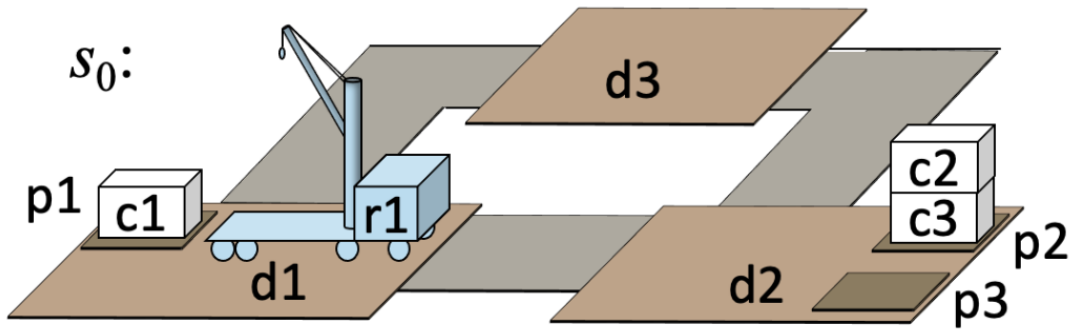
Action schemas:

- 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*.

Ques 3: Notice that cargo(r) = c iff pos(c) = r. Can we rewrite the domain to eliminate cargo(r)?
A. yes



- 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}

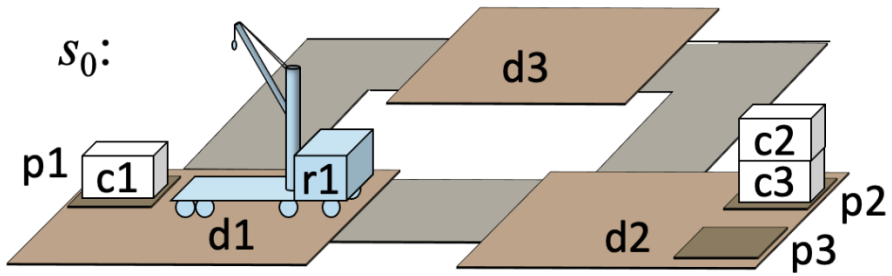
  where r ∈ *Robots*, c ∈ *Containers*, p ∈ Piles

- s_0 = {cargo(r1) = nil, cargo(r2) = nil,
  loc(r1) = d1, loc(r2) = d2,
  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}

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)

Ques4: Can we rewrite m2-uncover to eliminate c' ?
A. yes