Lecture 16

CS 6260 Automated Planning and Learning

Partial Order 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.

Logistics and Announcement

Logistics

- 3 Exams:
  1. [Quiz 1] Feb 19: 08:00 AM – 08:50 AM
  2. [Quiz 2] Apr 09: 08:00 AM – 08:50 AM
  3. [End Sem] May 12: 08:00 AM – 08:50 AM

- Office Hours: 1 hour after class every week Tu, W, Th
  - Due to the changed schedule, no office hours today

- Tomorrow's Class: 05:00 PM – 06:50 PM
  - A short break of 5 min in between

Talk Announcement

- **Title:** Integration of Structured Reasoning and Data-driven Learning for Acting and Planning
- **Date:** March 24, 2026
- **Time:** 09:30 AM – 10:30 AM
- Online link will be posted on Moodle

Dr. Sunandita Patra
J. P. Morgan AI Research, USA
https://sunanditapatra.wixsite.com/camp

Recap: Planners for HTN Planning

Pyhop

- A simple HTN planner written in Python
  - `import pyhop`
  - Depth-first version of TO-HTN-Forward with no goal tasks
  - Less than 150 lines of code, works in both Python 2 and 3
- State: Python object that contains state variables
  - `s = gtpyhop.State('Current state')`
  - To say r1 is at d1 in state *s*: `s.loc['r1'] = 'd1'`
- Actions and methods: ordinary Python functions
- Some limitations compared to most other HTN planners
  - I'll discuss later
- Open-source software, Apache license
  - http://bitbucket.org/dananau/pyhop

Comparison

Task: transport(c,y,z) – transport c from y to z

- TOHTN method:

Method m_transport(*r,x,c,y,z*)
  Task: transport(*c,y,z*)
  Pre: loc(*r*) = *x*, cargo(*r*) = nil, loc(*c*) = *y*
  Sub: move(*r,x,y*), take(*r,c,y*), move(*r,y,z*), put(*r,c,z*)

Most HTN planners:

- Write in a planning language the planner can read and analyze
- Can have parameters not mentioned in the task
  - robot *r*, location *x*
  - Backtrack over multiple possibilities
- Planner knows in advance what the subtasks are
  - Helps with implementing heuristic functions

- Pyhop method: ordinary Python function
  - Args: state *s* and the task parameters

```
def m_transport(s,c,y,z):
  (r,x) = find_suitable_robot('transport',s,c,y,z)
  if r != 'failure':
    return [('move',r,x,y), ('take',r,c,y), \
      ('move',r,y,z), ('put',r,c,z)]
  else: return False
```

- Advantages
  - Don't need to learn a planning language: write methods and actions in Python
- Disadvantages:
  - Planner doesn't know in advance what the subtasks are
    - How to implement a heuristic function?
  - What about parameters not mentioned in the task?

GTPyhop

- GTPyhop (2021):
- Like Pyhop, but has both compound tasks and goal tasks
  - declare *task methods* for compound tasks
  - declare *goal methods* for goal tasks
- Open-source: https://github.com/dananau/GTPyhop
- Mostly backward-compatible with Pyhop

Two kinds of goals:

- *Unigoal*: a single atom
  - represented as a triple (*name, arg, value*)

    ```
    ('pos', 'a', 'b')
    ```
  - goal: get to a state *s* in which

    ```
    s.pos['a']=='b'
    ```
- *Multigoal*: a conjunction of atoms
  - represented as a state-like object

    ```
    g = gtpyhop.Multigoal('Sussman goal')
    g.pos = {'a':'b', 'b':'c'}
    ```
  - goal: get to a state *s* in which

    ```
    s.pos['a']=='b' and s.pos['b']=='c'
    ```

Planners for HTN Planning

Socks and Shoes

- Suppose you have actions to put on socks or shoes.
  - A sock must be placed on the foot before a shoe, where $\prec$ indicates an ordering constraint
  - There are two feet
- Now suppose the following Partial Plan:
  - Left sock $\prec$ left shoe
  - Right sock $\prec$ right shoe

- If it helps, use the following naming:
  - Left Sock: LK
  - Left Shoe: LO
  - Right Sock: RK
  - Right Shoe: RO

Partial Order HTN Planning

POHTN (Partially Ordered HTN) Planning

- Sometimes we don't want to specify a total ordering on tasks
- Represent partially ordered tasks as a *task network*:
  - a pair $\mathcal{T} = (T, \prec)$
  - T is a set of task nodes
  - $\prec$ is a partial ordering of T
- *Task node:* a pair $\tau = (l, t)$
  - t is a task
  - l is a name that uniquely identifies τ
- Need labels so we can have multiple occurrences of t
- *POHTN Method*: a tuple
  (*head*, *task*, *pre*, *sub*, $\prec$)
- As usual, write POHTN methods as pseudocode:

  method-name(*args*)
  Task: *nonprimitive task*
  Pre: *preconditions*
  Sub: *subtask nodes*
  $\prec$: *partial ordering of the subtask nodes*

- TOHTN planning is a special case of POHTN planning
  - $\prec$ is a total ordering

Example POHTN Problem

- Σ_c : DWR with cranes attached to loading docks, not robots

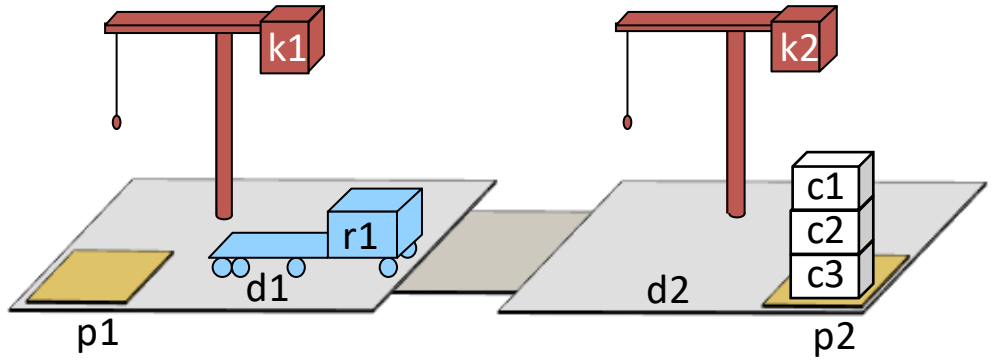
unstack(k, c, c', p, d) // *take container c from pile p*
pre: at(k, d), at(p, d), holding(k) = nil, pos$(c) = c'$, top$(p) = c$
eff: holding$(k) \leftarrow c$, pos$(c) \leftarrow k$, pile$(c) \leftarrow$ nil, top$(p) \leftarrow c'$

stack(k, c, c', p, d) // *put container c onto pile p*
pre: at(k, d), at(p, d), holding$(k) = c$, top$(p) \leftarrow c'$
eff: holding$(k) \leftarrow$ nil, pos$(c) = c'$, pile$(c) \leftarrow p$, top$(p) = c$

unload(k, c, r, d) // *take container c from robot r*
pre: at(k, d), holding$(k) = c$, loc$(r) = d$
eff: cargo$(r) \leftarrow c$, pos$(c) \leftarrow r$, holding$(k) \leftarrow$ nil

load(k, c, r, d) // *put container c onto robot r*
pre: at(k, d), holding(k) = nil, loc$(r) = d$, cargo$(r) = c$
eff: pos$(c) \leftarrow k$, holding$(k) \leftarrow c$, cargo$(r) \leftarrow$ nil

Poll. How many solution plans?
A. 1 B. 2 C. 3 D. 4 E. other



- $P = (\Sigma, s_0, (T, \prec))$
  - T = {put-on-robot(c1,r1)}; $\prec = \emptyset$

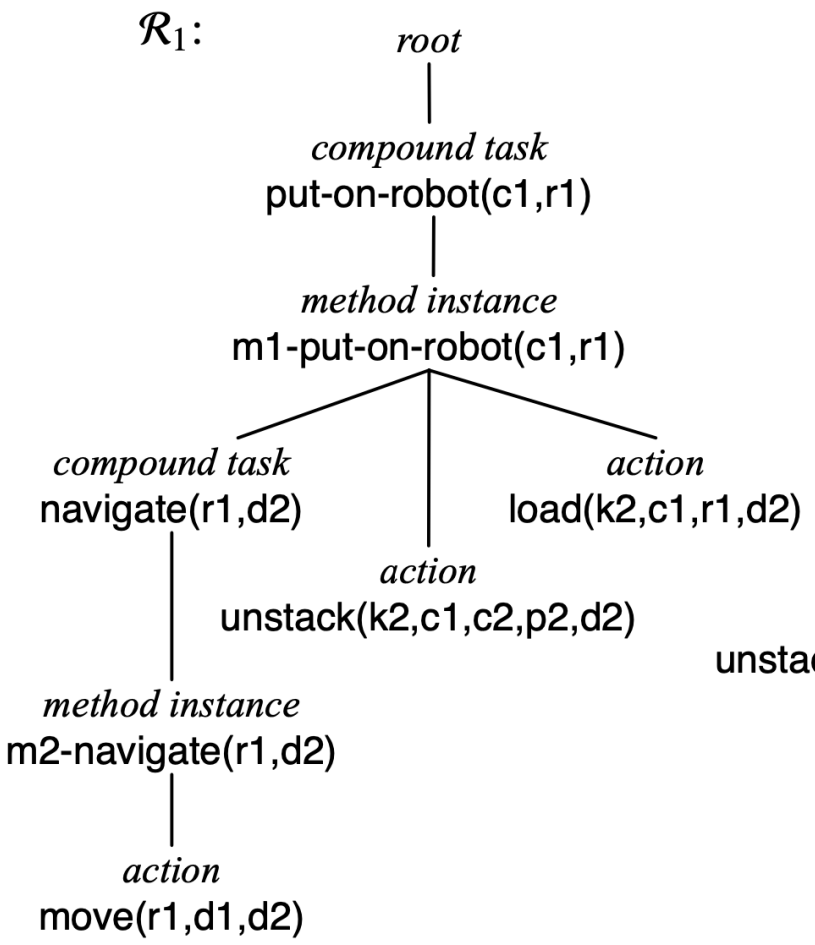
- $\Sigma = (\Sigma_c, \mathcal{M})$
  - $\mathcal{M}$: three methods

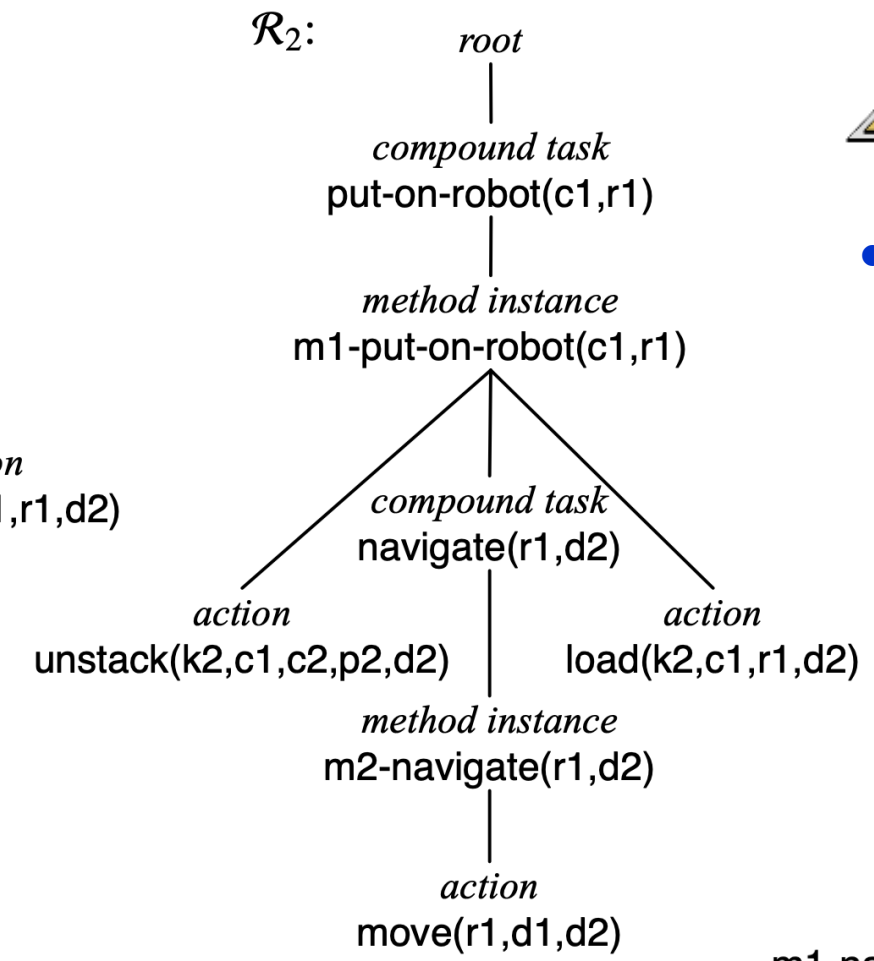
m1-put-on-robot(k, c, c', r, d, p)
task: put-on-robot(c, r)
pre: cargo(r) = nil, top$(p) = c$, at(p,d), attached(k,d), holding(k) = nil
sub: (t1, navigate(r, d))
(t2, unstack(k,c,c',p,d))
(t3, load(k,r,c,d))
$\prec$: t1 $\prec$ t3, t2 $\prec$ t3

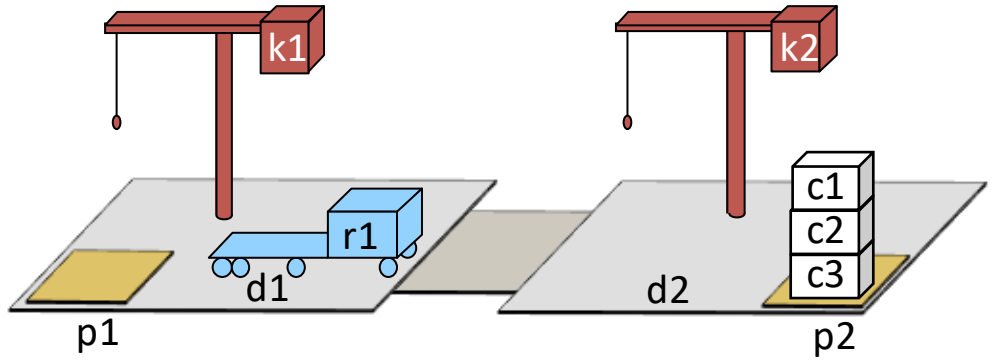
m1-navigate(r, d)
task: navigate(r, d)
pre: loc$(r) = d$
sub: // *none*
$\prec$: // *none*

m2-navigate(r, d', d)
task: navigate(r, d)
pre: adjacent(d', d), loc$(r) = d'$
sub: (t1, move(r, d', d))
$\prec$: // *none*

Solution Trees







- $P = (\Sigma, s_0, (T, \prec))$
  - $T = \{\text{put-on-robot(c1,r1)}\};\ \prec = \emptyset$

- π_1 = move(r1,d1,d2), unstack(k2,c1,c2,p2,d2), load(k2,c1,r1,d2)
- π_2 = unstack(k2,c1,c2,p2,d2), move(r1,d1,d2), load(k2,c1,r1,d2)

m1-put-on-robot(k, c, c', r, d, p)
task: put-on-robot(c, r)
pre: cargo(r) = nil, top(p) = c, at(p,d), attached(k,d), holding(k) = nil
sub: (t1, navigate(r, d))
(t2, unstack(k,c,c',p,d))
(t3, load(k,r,c,d))
$\prec$: t1 $\prec$ t3, t2 $\prec$ t3

m1-navigate(r, d)
task: navigate(r, d)
pre: loc(r) = d
sub: // *none*
$\prec$: // *none*

m2-navigate(r, d', d)
task: navigate(r, d)
pre: adjacent(d', d), loc(r) = d'
sub: (t1, move(r, d', d))
$\prec$: // *none*

Planning Algorithm

Three cases: primitive, compound, goal task

- Primitive task node: apply action

$\mathcal{T} = (T, \prec)$, $T = \{\tau, \tau_2, \ldots, \tau_k\}$, nothing precedes τ
state s_0 i.e., $\nexists \tau' \in T$ s.t. $\tau' \prec \tau$
new state $\gamma(s_0, \tau)$; $\{\tau_2, \ldots, \tau_k\}$

- Compound task node: apply method instance

$\mathcal{T} = (T, \prec)$, $T = \{\tau, \tau_2, \ldots, \tau_k\}$, nothing precedes τ
method instance m i.e., $\nexists \tau' \in T$ s.t. $\tau' \prec \tau$
state s_0 $\{ v_1, \ldots, v_j , \tau_2, \ldots, \tau_k\}$
make $v_1, \ldots, v_j$ precede everything τ preceded

PO-HTN-Forward($\Sigma_c, \mathcal{M}, s, \mathcal{T}$)
  if $\mathcal{T}$ is empty **then return** $\langle\rangle$
1 **nondeterministically choose** a node τ in $\mathcal{T}$ that has no predecessors in $\mathcal{T}$
  foreach τ' in $\mathcal{T}$ that has no predecessors in $\mathcal{T}$ **do**
2 **if** $\tau' \neq \tau$ **then** add ordering constraints to $\mathcal{T}$ to make $\tau \prec \tau'$
  $t \leftarrow$ task(τ)
  $M \leftarrow$ HTN-Get-Candidates($\Sigma_c, \mathcal{M}, s, t$)
  if $M \neq \varnothing$ **then**
    nondeterministically choose $m \in M$
    if m is an action **then**
      $\pi \leftarrow$ PO-HTN-Forward($\Sigma_c, \mathcal{M}, \gamma(s, a), \mathcal{T} \setminus \{\tau\}$)
      if $\pi \neq$ failure **then return** $a \cdot \pi$
    else if m is a ground method **then**
      return PO-HTN-Forward($\Sigma_c, \mathcal{M}, s, refine(\mathcal{T}, \tau, m)$)
  return failure

Using HTN Domains for Acting in the Environment

Using HTN Domain Models for Acting

- Unlike an HTN domain model, the actor's environment is not necessarily deterministic or static
  - Exogenous events, unanticipated action outcomes ⇒ current state may be different from what an HTN model would predict
- Actor can't backtrack to a previous state; prior actions are in the past

- HTN domain models still are very useful for providing *operational* models to the actor
  - How to carry out "standard operating procedures"
  - How to perform complex tasks without searching through a large state space
  - How to avoid situations where unanticipated events are likely to cause bad outcomes
  - How to recover when unanticipated events occur

Reactive HTN Actor

- Like TO-HTN-Forward but executes each action
  - Can similarly modify other Chapter 5 algorithms

Line

0 Return success or failure, not a plan

1 s isn't an argument, observe it instead

3 Instead of computing γ, execute action

2 Failure recovery: if m fails, try next one
  - if they all fail, return failure to next higher level in the recursion stack, to try other methods there

- At Line 2, a bad method instance can lead to non-optimal solution or failure
  - Can use a heuristic function
  - Can call an HTN planner – but other ways have less computational overhead

TO-HTN-Act$(\Sigma_c, \mathcal{M}, T)$

0 **if** T is empty **then return** success
  $t \leftarrow$ the first element of T; $T' \leftarrow$ the rest of T
1 $s \leftarrow$ observe current state
  $M \leftarrow$ HTN-Get-Candidates$(\Sigma_c, \mathcal{M}, s, t)$
2 **foreach** $m \in M$ **do**
    if m is a method instance **then**
      if TO-HTN-Act$(\Sigma, \text{sub}(m) \cdot T') =$ success **then return** success
    else if m is an action **then**
3 execute m
      if m executed successfully **then return** TO-HTN-Act(Σ, T')
  return failure