Lecture 18

CS 6260 Automated Planning and Learning

Hierarchical Domain Definition Language

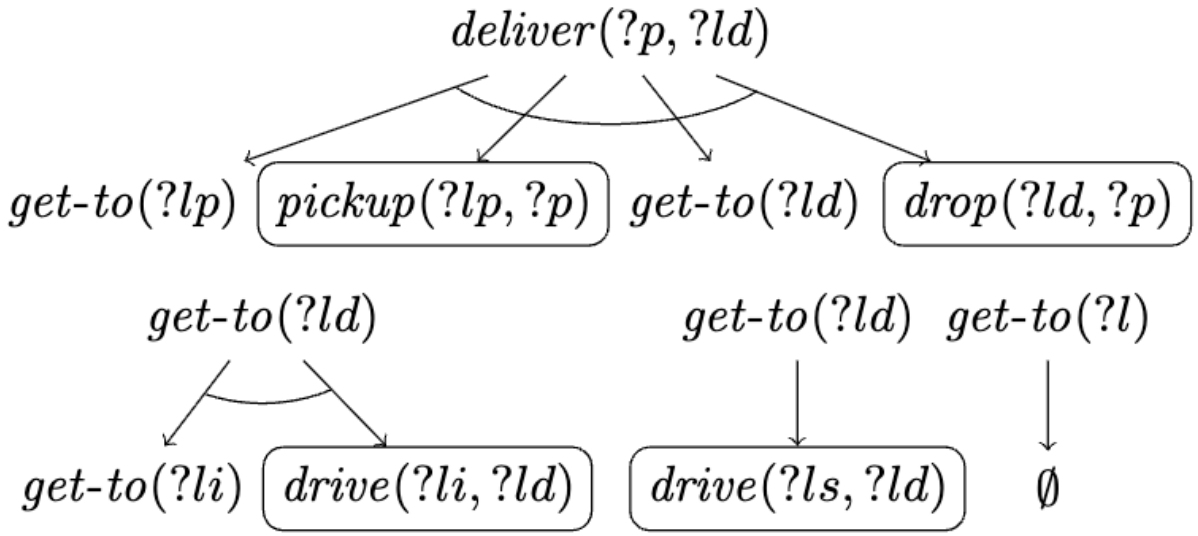
Department of Computer Science and Engineering

Indian Institute of Technology Madras

PDDL for Transport Domain

```
(define (domain transport)
  (:types location package - object)
  (:predicates
     (road ?l1 ?l2 - location)
     ...)
```

```
(:action drive
  :parameters (?l1 ?l2 - location)
  :precondition (and
    (tAt ?l1)
    (road ?l1 ?l2))
  :effect (and
    (not (tAt ?l1))
    (tAt ?l2)))
...)
```



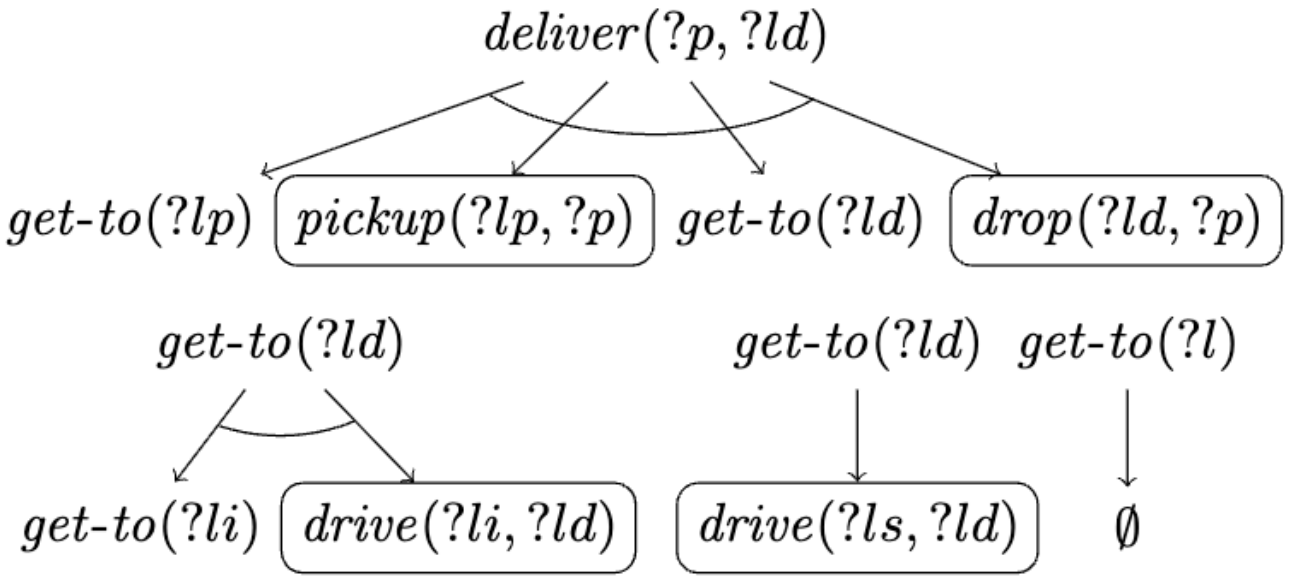
HDDL: PDDL Extended for Hierarchical Planning

```
(define (domain transport)
  (:types location package - object)
  (:predicates
     (road ?l1 ?l2 - location)
     ...)


(:action drive
  :parameters (?l1 ?l2 - location)
  :precondition (and
    (tAt ?l1)
    (road ?l1 ?l2))
  :effect (and
    (not (tAt ?l1))
    (tAt ?l2)))
...)
```

- Everything in PDDL, plus
  - Compound Task Definitions
  - Method Definitions
  - Initial Task Network
  - (Optional) Goal Description

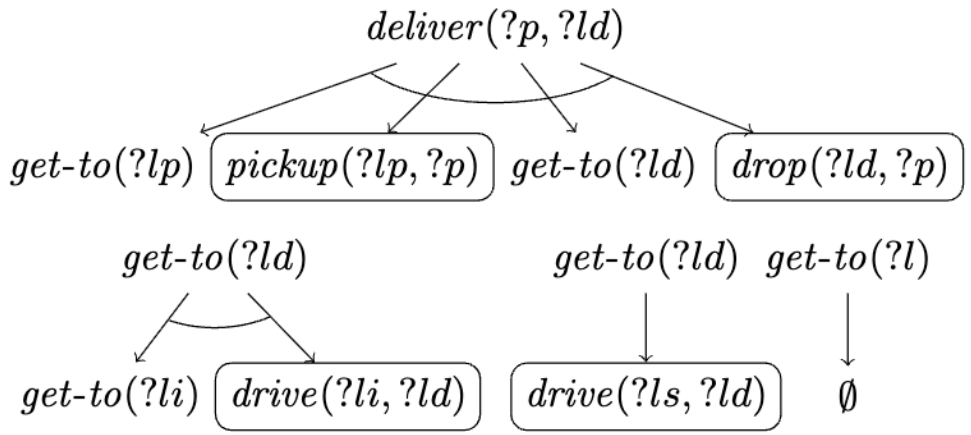
Tasks




```
(:task deliver :parameters (?p - package
    ?l - location))
(:task get-to :parameters (?l - location))
```

Method

```
(:method m-deliver
  :parameters (?p - package
       ?lp ?ld - location)
  :task (deliver ?p ?ld)
  :ordered-subtasks (and
    (get-to ?lp)
    (pick-up ?ld ?p)
    (get-to ?ld)
    (drop ?ld ?p)))
```



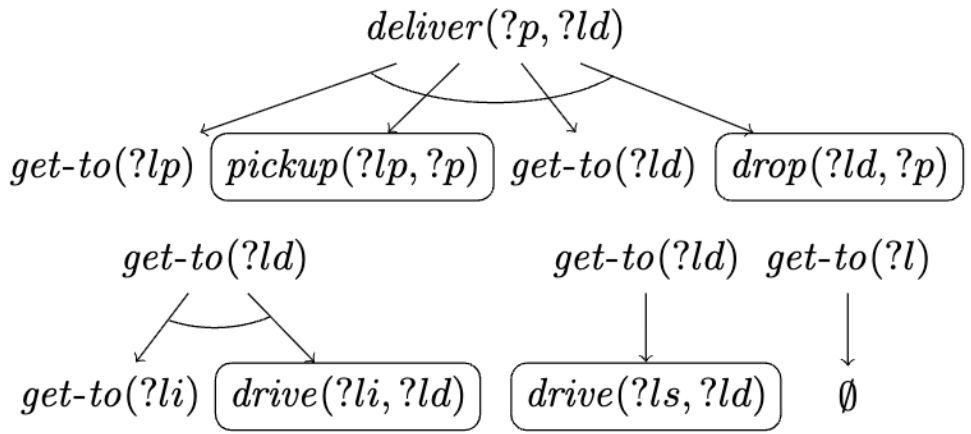
Method: Partial Ordering Support

```
(:method m-drive-to-via
  :parameters (?li ?ld - location)
  :task (get-to ?ld)
  :subtasks (and
    (t1 (get-to ?li))
    (t2 (drive ?li ?ld)))
  :ordering (and
    (< t1 t2)))
```

```
(:method m-deliver
  :parameters (?p - package
       ?lp ?ld - location)
  :task (deliver ?p ?ld)
  :ordered-subtasks (and
    (get-to ?lp)
    (pick-up ?ld ?p)
    (get-to ?ld)
    (drop ?ld ?p)))
```

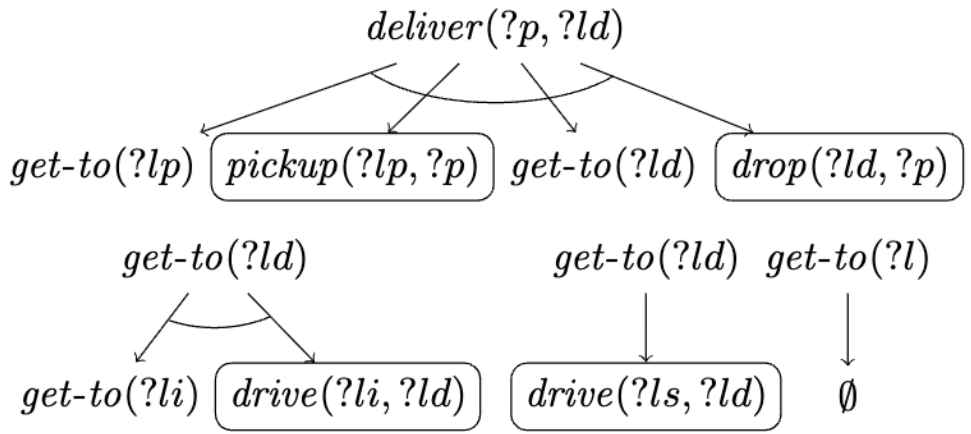
Constraints

```
(:method m-direct
  :parameters (?ls ?ld - location)
  :task (get-to ?ld)
  :constraints
    (not (= ?li ?ld))
  :subtasks (drive ?ls ?ld))
```



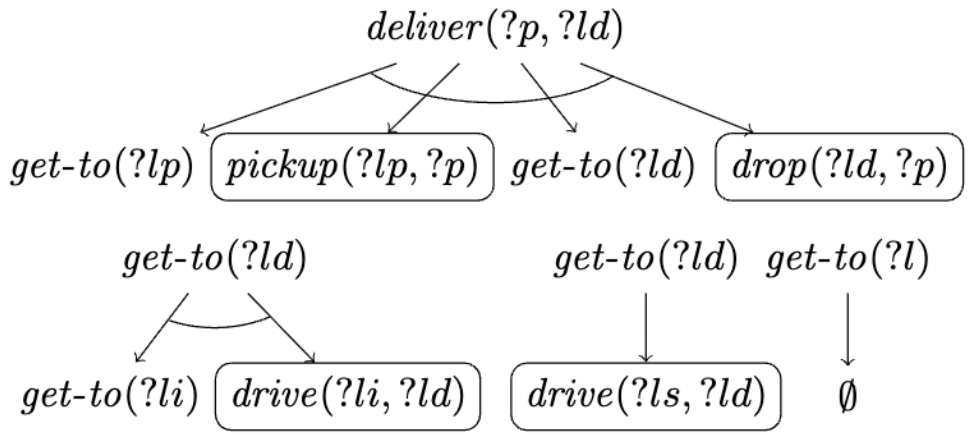
Problem

```
(define (problem p)
 (:domain transport)
 (:objects
  city-loc-0 city-loc-1 city-loc-2 -
      location
  package-0 package-1 - package)
```



HTN Specification

```
(:htn
 :tasks (and
  (deliver package-0 city-loc-0)
  (deliver package-1 city-loc-2))
 :ordering ()
 :constraints ())
(:init
 (road city-loc-0 city-loc-1)
 (road city-loc-1 city-loc-0)
 (road city-loc-1 city-loc-2)
 (road city-loc-2 city-loc-1)
 (at package-0 city-loc-1)
 (at package-1 city-loc-1)))
```



HDDL 2.1: Durative Actions

```
(:durative-action take_image
  :parameters (?ti_s - satellite
      ?ti_d - image_direction
      ?ti_i - instrument ?ti_m - mode)
  :duration (= ?duration 1.00000)
  :precondition (and
      (over all (observable ?ti_d))
      (at start (calibrated ?ti_i))
      (at start (pointing ?ti_s ?ti_d))
      (over all (on_board ?ti_i ?ti_s))
      (over all (power_on ?ti_i))
      (at start (supports ?ti_i ?ti_m)) )
  :effect (and
    (at end (have_image ?ti_d ?ti_m))
    (at end (increase
       (image-quality ?ti_s) 2))) )
```

HDDL 2.1: Durative Methods

```
:durative-method turn_precise
 :parameters (?tp_s - satellite
     ?tp_d_prev - image_direction
     ?tp_d - image_direction)
 :task (turn_to ?tp_s ?tp_d ?tp_d_prev)
 :duration (>= ?duration (* (/ (turn-tim
     (t_d_new t_d_prev)) 4) 5) )
 :ordered-subtasks (and
   (point_to ?tp_s ?tp_d ?tp_d_prev)
   (take_image ?tp_s ?tp_d ?tp_i ?tp_m)
 :constraints (and
   (not (= ?tp_d ?tp_d_prev))) )
```

```
:durative-method turn_approx
 :parameters (?ta_s - satellite
     ?ta_d_prev - image_direction
     ?ta_d - image_direction)
 :task (turn_to ?ta_s ?ta_d ?ta_d_prev)
 :duration (and
     (<= ?duration (* (/ (turn-time
         (t_d_new t_d_prev)) 4) 5))
     (>= ?duration (/ (turn-time
         (t_d_new t_d_prev)) 2) )
 :ordered-subtasks (and
   (point_to ?ta_s ?ta_d ?ta_d_prev)
   (take_image ?ta_s ?ta_d ?ta_i ?ta_m)
   (decrease_overall_quality ?ti_s) )
 :constraints (and
   (not (= ?ta_d ?ta_d_prev))) )
```

HDDL 2.1: Durative Methods

```
(:durative-method method_observe
  :parameters (?ta_d_prev - image_direction
     ?ta_s - satellite
     ?ta_d - image_direction
     ?ta_i - instrument ?m - mode)
  :task (do_observation ?ta_d ?m)
  :duration
   (< (duration task1) (calib-time ?ta_i))
  :subtasks (and
   (task0 (activate_instrument ?ta_s ?ta_i))
   (task1 (turn_to ?ta_s ?ta_d ?ta_d_prev))
   (task2 (take_image ?ta_s ?ta_d ?ta_i ?m))
  )
  :ordering (and
    (< task0 task2)
    (< task1 task2) )
  :constraints (and
    (not (= ?ta_d ?ta_d_prev))
    (hold-before task1 (power_on ?ta_i)))
)
```

HDDL and HTN References

- HDDL: https://arxiv.org/abs/1911.05499
- HDDL 2.1: https://arxiv.org/abs/2306.07353
  - Examples: https://github.com/pellierd/HDDL2.1
- HTN Tutorial:
  - Part 1: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2018/HTN-Tutorial-Part-I.pdf
  - Part 2: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2018/HTN-Tutorial-Part-II.pdf

Error Recovery in HTN Domains

- HTN methods require the solution plan to follow a particular trajectory
- Encode requirements that aren't explicit in the classical planning domain
  - Safety requirements:
    - Secure a container onto the robot before starting to move the robot
  - Commitments to other agents
    - Don't use a particular resource, because others may need it
  - A company's standard operating procedures
- HTN-Run-Lookahead and HTN-Run-Lazy-Lookahead don't know anything about the trajectory requirements
- That's OK if nothing goes wrong
- If unexpected events occur, need to recover in a way that still satisfies the trajectory requirements
- Three approaches
  1. Modify TO-HTN-Act to call an HTN planner
     - HTN planner returns a method selection
  2. Modify HTN planner to return a solution tree
     - Actor traverses the tree
  3. Actor calls HTN planner to do replanning in a modified domain