Lecture 11

CS 6260 Automated Planning and Learning

Temporal Planning: Durative Actions 2

Department of Computer Science and Engineering

Indian Institute of Technology Madras

Most material based on David E. Smith's Tutorial on Temporal Planning at 3rd Summer School on Cognitive Robotics

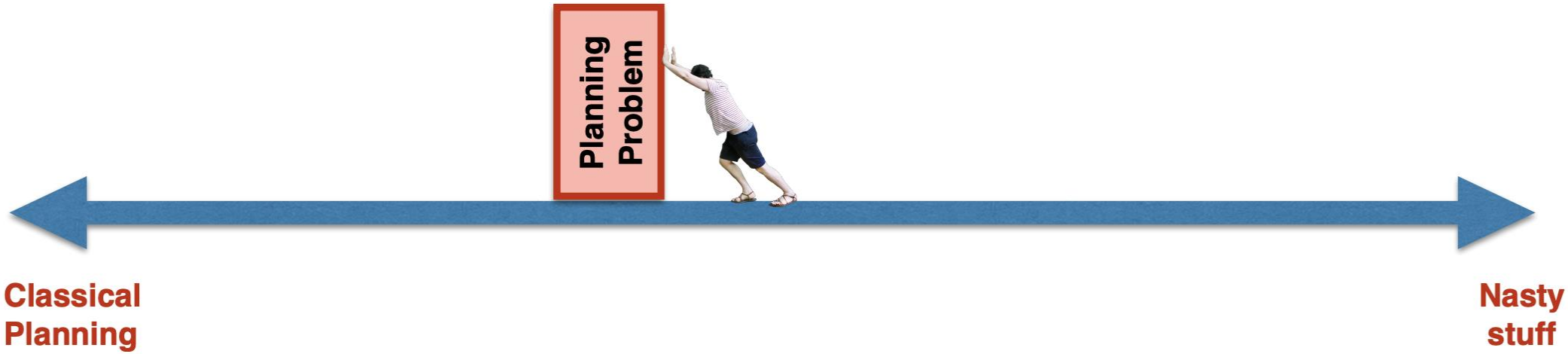
PDDL 2.1: Durative Actions

Refer to this document for PDDL 2.1
https://www.jair.org/index.php/jair/article/view/10352/24759

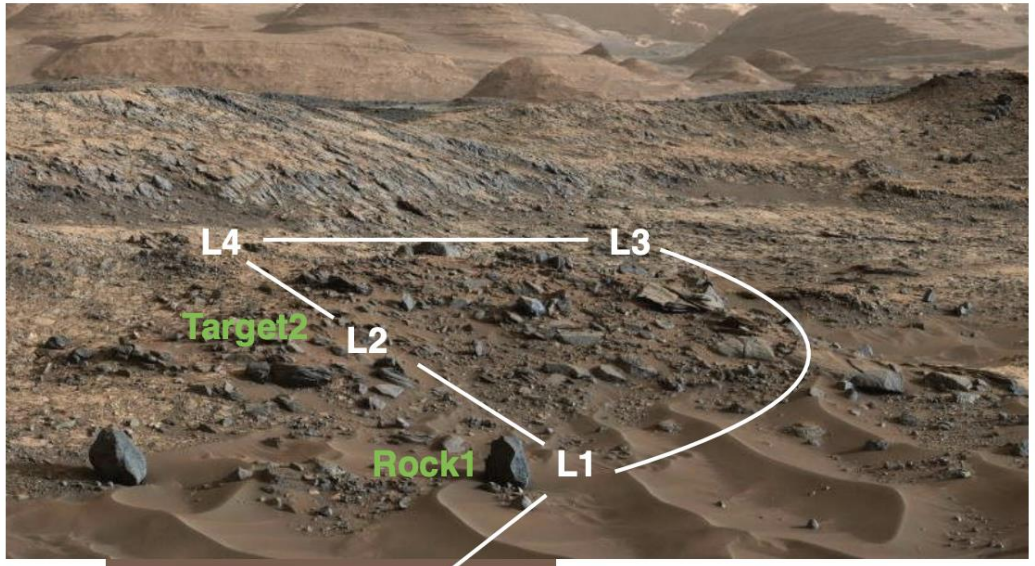
Additional Info: https://planning.wiki/ref/pddl

https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/smith03a-html/PDDL-commentary.html

Temporal Spectrum

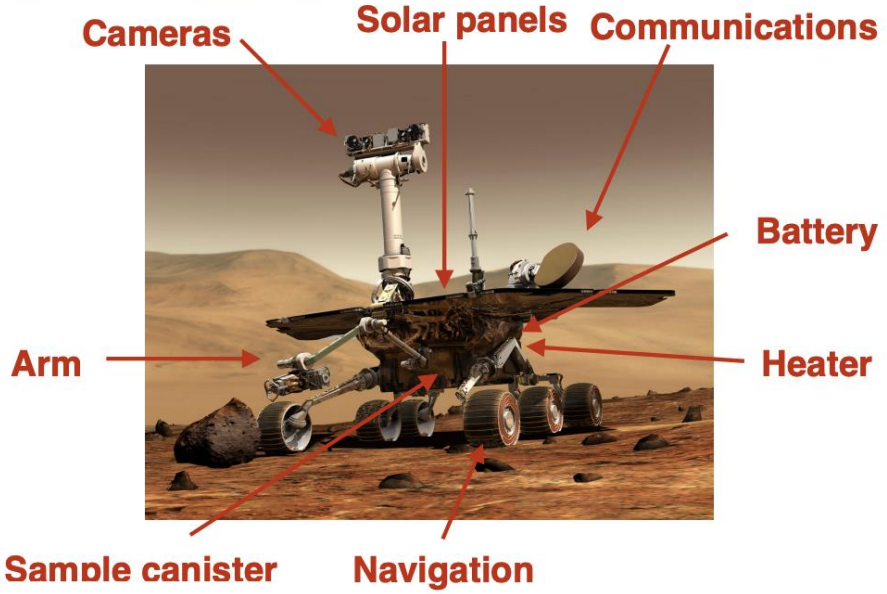


Rover Example



Path(L0,L1)
Visible(Rock1,L1)
Path(L1,L2)
Path(L2,L1)
Reachable(Target2,L2)
Path(L1,L3)
…

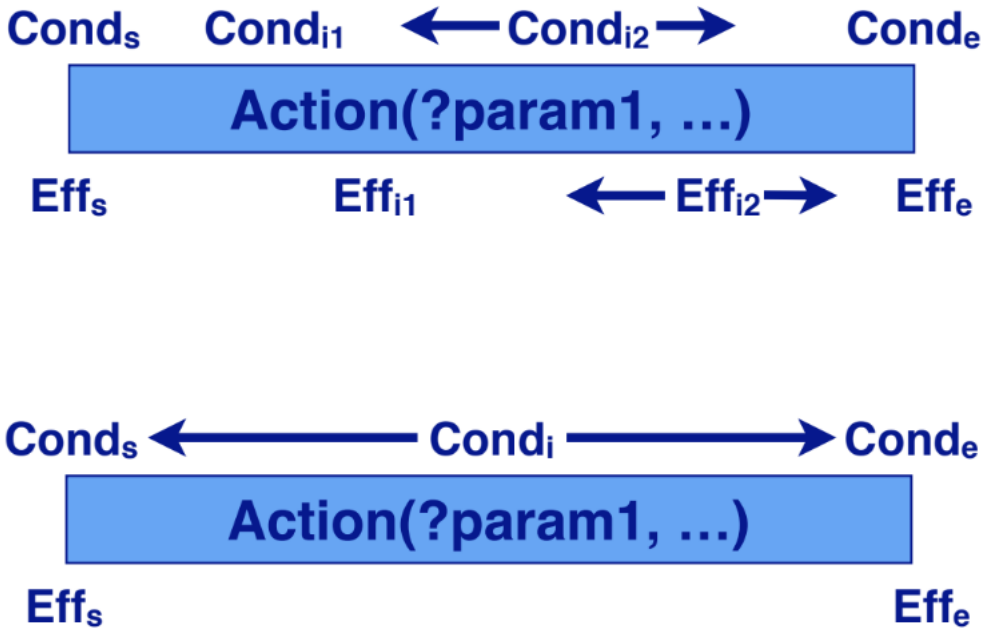
L0



At(L0)
Stowed
Empty
Charged

See also: https://github.com/potassco/pddl-instances/blob/master/ipc-2002/domains/rovers-time-automatic

Durative Actions with Discrete Effects



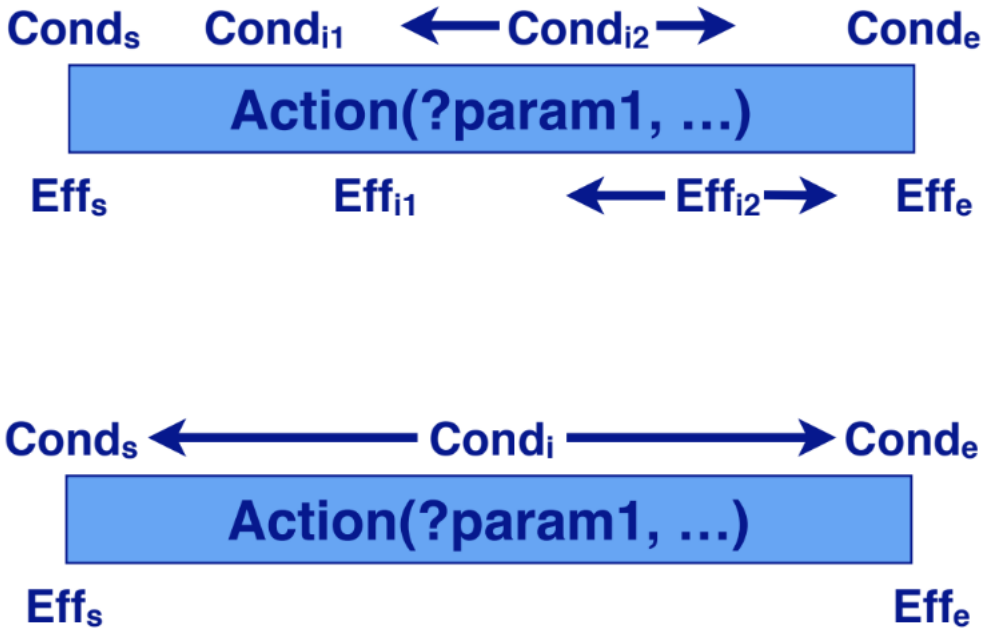

```
(:durative-action move
    :parameters (<arguments>)
    :duration (= ?duration 5)
    :condition (logical_expression)
    :effect (logical_expression)
)
```

:durative-actions

```
(:durative-action move
   :parameters (?r - rover ?fromwp - waypoint ?towp - waypoint)
   :duration (= ?duration 5)
   :condition (and
      (over all (can-move ?from-waypoint ?to-waypoint))
      (at start (at ?rover ?from-waypoint))
      (at start (> (battery-amount ?rover) 8)))
   :effect (and
      (at end (at ?rover ?to-waypoint))
      (at end (been-at ?rover ?to-waypoint))
      (at start (not (at ?rover ?from-waypoint)))
      (at start (decrease (battery-amount ?rover) 8))
      (at end (increase (distance-travelled) 5)))
)
```

See also: https://github.com/potassco/pddl-instances/blob/master/ipc-2002/domains/rovers-time-automatic

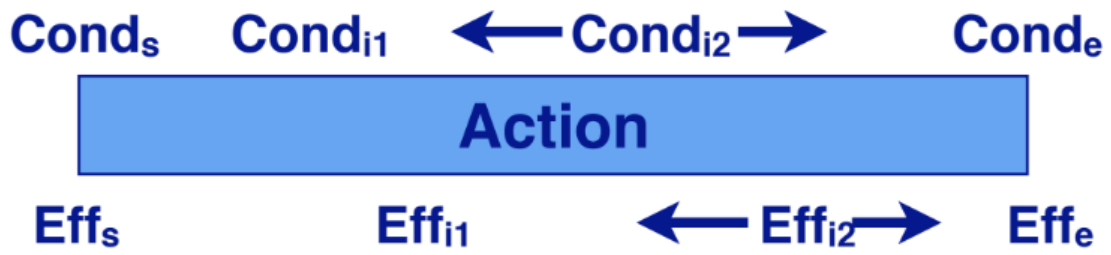
Durative Actions with Discrete Effects

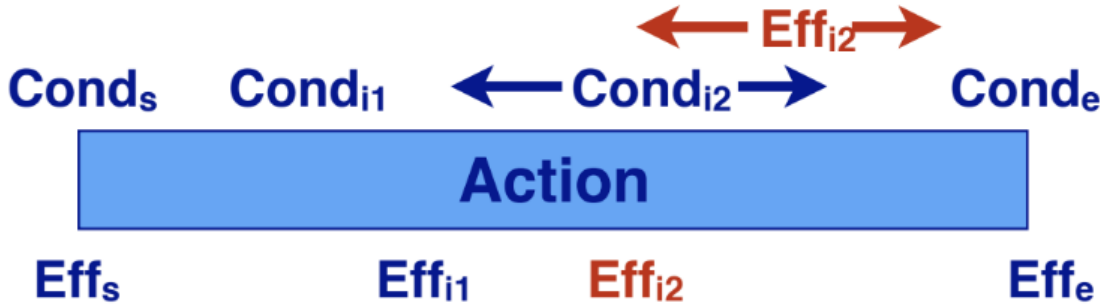



```
(:durative-action move
    :parameters (<arguments>)
    :duration (= ?duration 5)
    :condition (logical_expression)
    :effect (logical_expression)
)
```

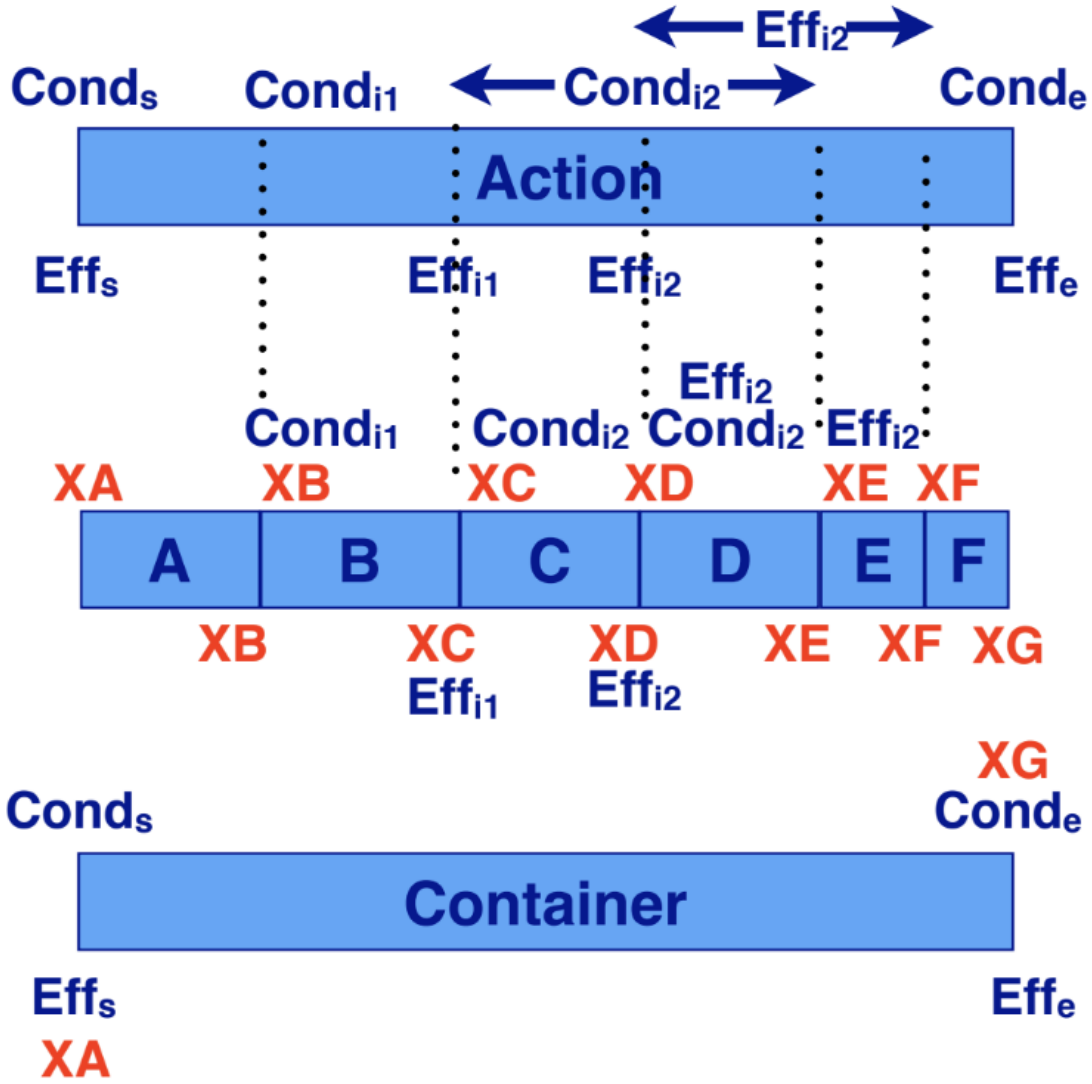
Intermediate Conditions and Effects?

Step 1: Eliminate Interval Effects





Step 2: Decomposing Intermediate Conditions and Effects



Simple Durative Planning

Initial Conditions

At(L0)
Stowed
Empty
Charged

Path(L0,L1)
Visible(Rock1,L1)
Path(L1,L2)
Path(L2,L1)
Reachable(Target2,L2)
Path(L1,L3)
…

Actions

Cond
Navigate(?loc1, ?loc2)
Eff

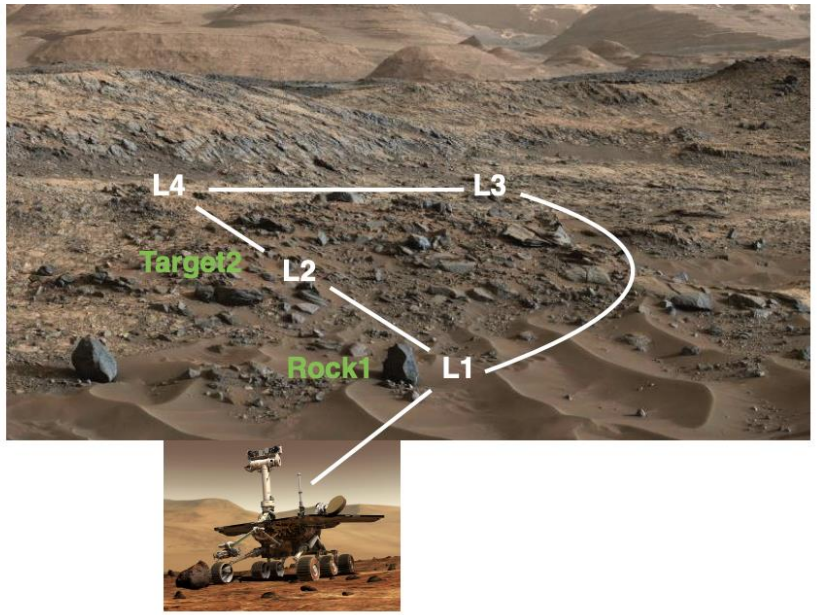
Cond
Snap(?target)
Eff

Cond
PlaceArm(?target)
Eff

Cond
CollectSample(?target)
Eff

Goal(s)

Image(Rock1)
Sample(Target2)
At(L3)



Simple Durative Actions

Charged
At(?loc1)

Stowed

Navigate(?loc1, ?loc2)

Dur=DriveTime(?loc1,?loc2)

¬At(?loc1)

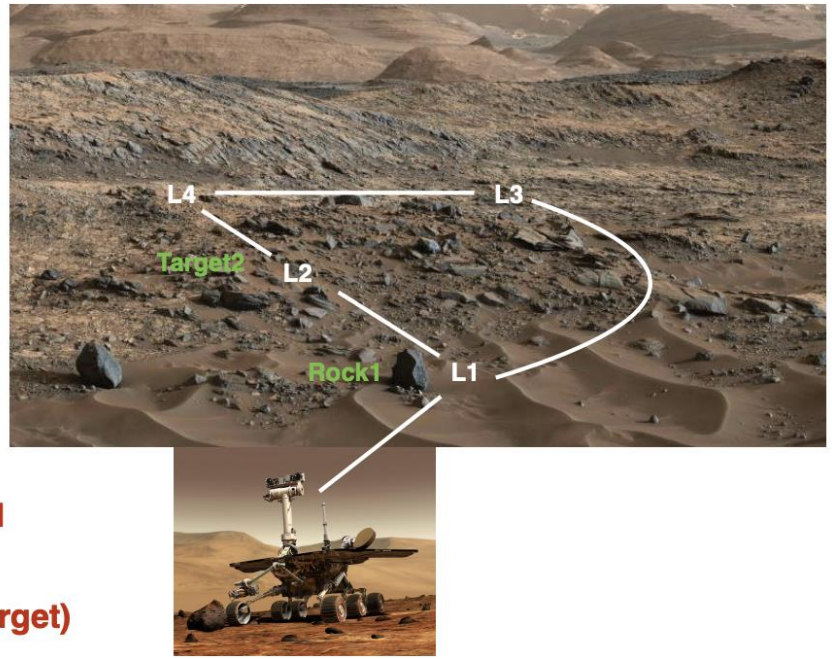
At(?loc2)

At(?loc)
Stowed
Viz(?target, ?loc)

Snap(?target)

Dur=1

Image(?target)



Stowed

At(?loc)
Reach(?target, ?loc)

PlaceArm(?target)

Dur=5

¬Stowed

Placed(?target)

Stow()

Dur=1

Stowed
¬Placed(?target)

Empty

Placed(?target)

CollectSample(?target)

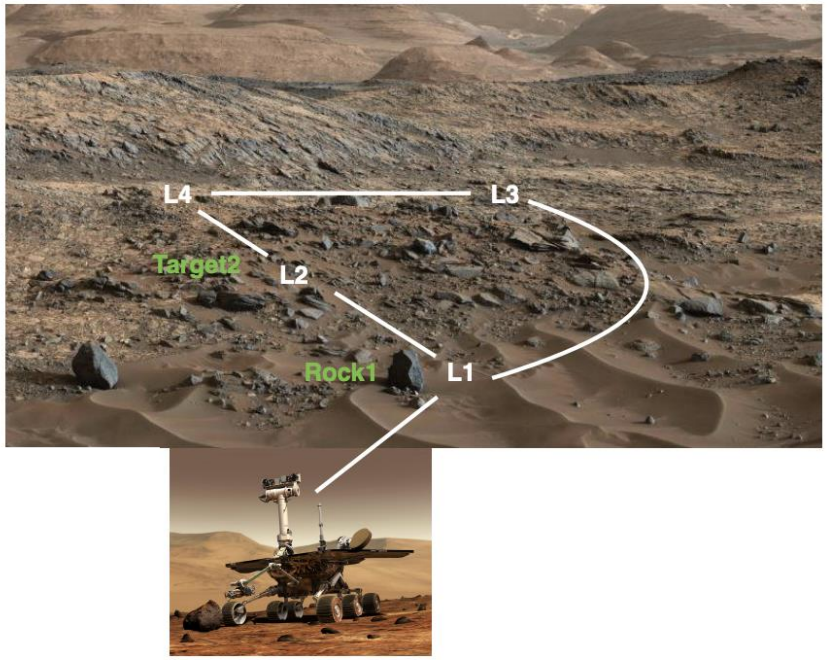
Dur=2

¬Empty

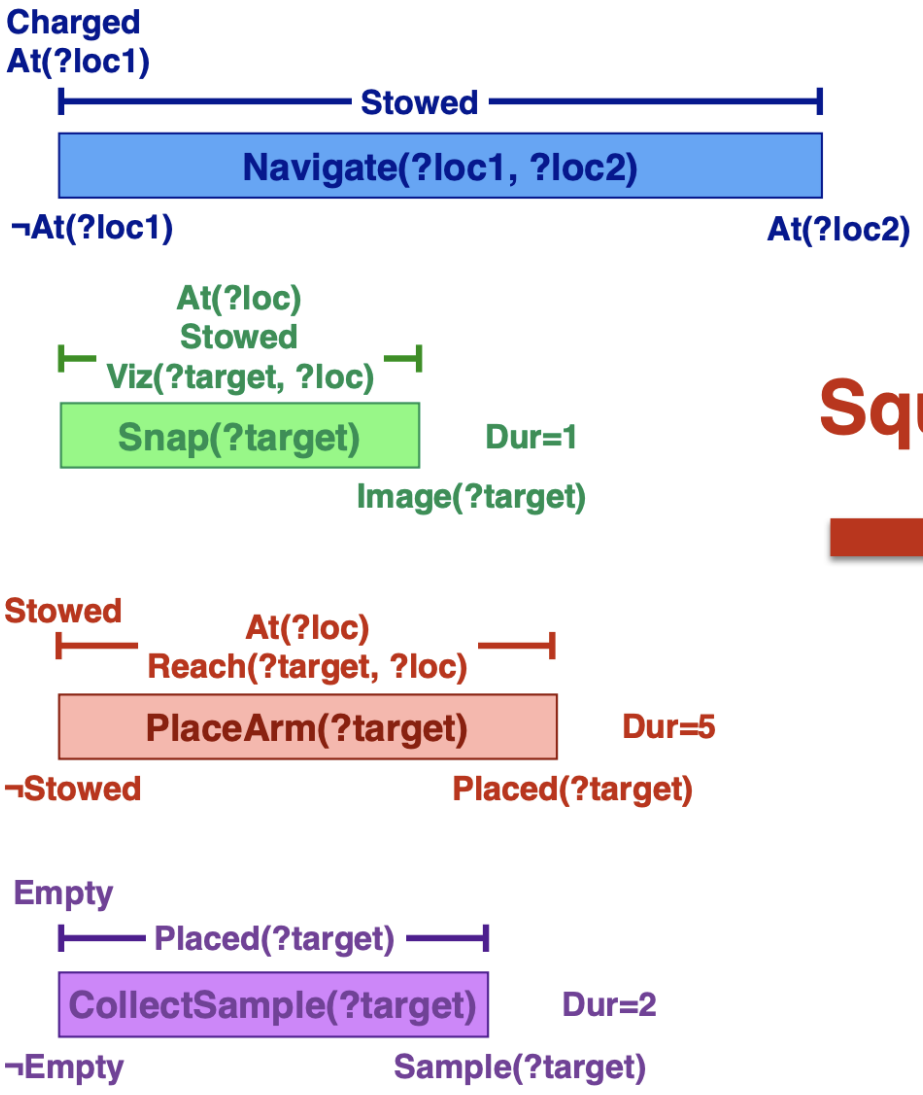
Sample(?target)

Rover Plan

Goal(s): Image(Rock1), Sample(Target2), At(L3)

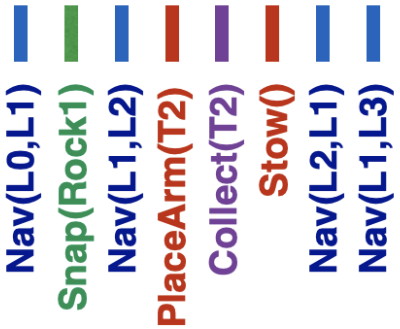


Reduction to Classical Planning



Reduction to Classical Planning

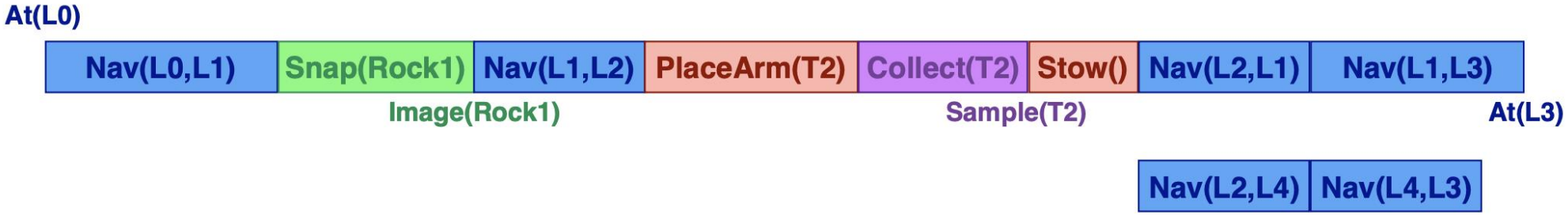
Run Classical Planner



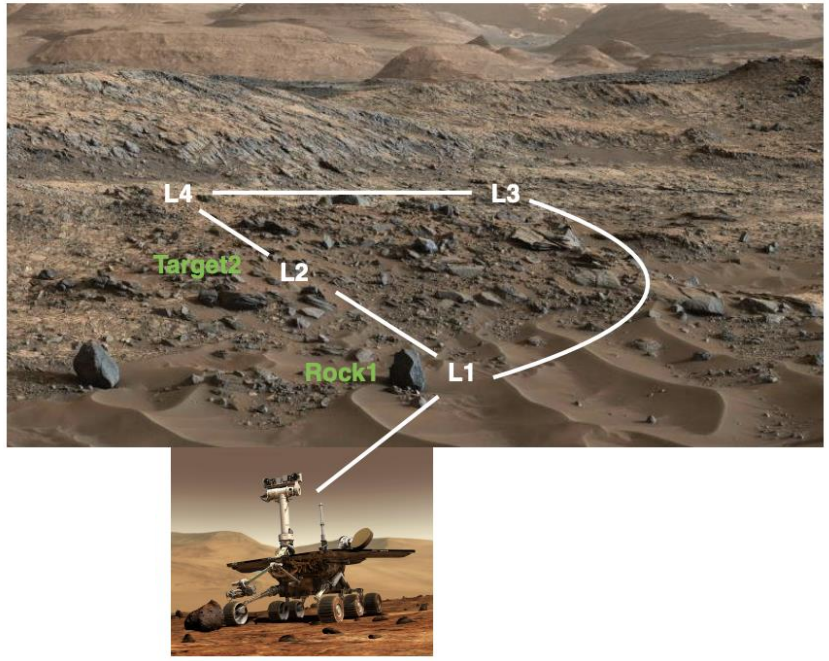
Expand

Solution: Minimizing Makespan

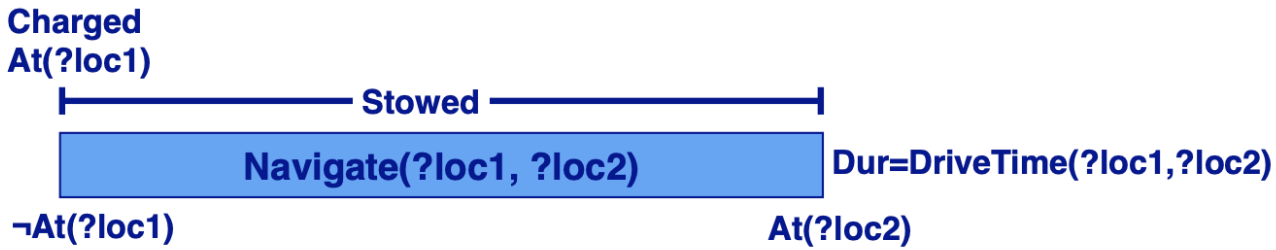
Goal(s): Image(Rock1), Sample(Target2), At(Loc3)

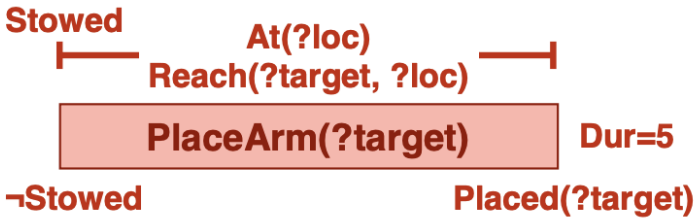


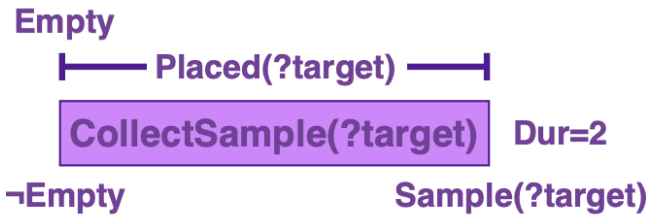
$$\textbf{find } \pi : \forall \pi', dur(\pi) \leq dur(\pi')$$



Reduction to Optimal Numeric Planning



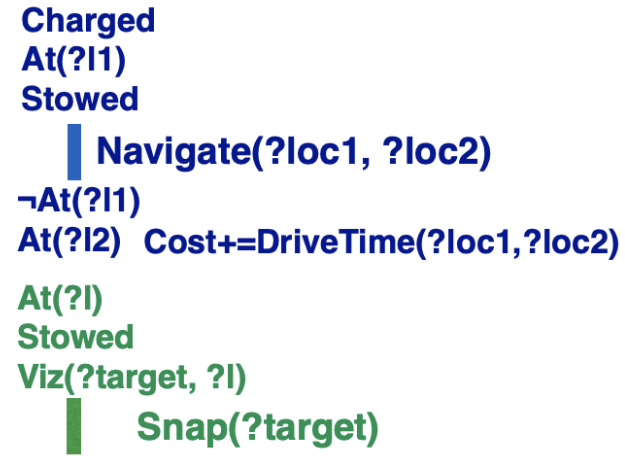




Squash

+ costs
+ metric

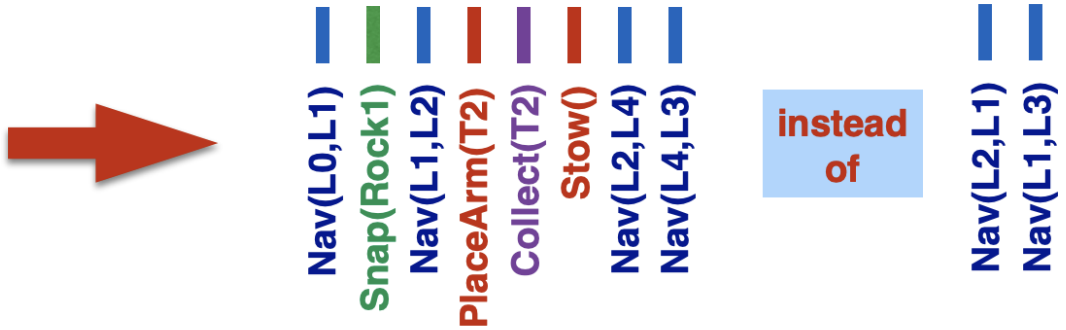
:functions (cost)
:init (= (cost) 0)
:metric minimize (cost)



:increase (cost) 2

Reduction to Optimal Numeric Planning

Run Numeric Planner
(minimizing cost)



Expand

Numeric planning
– minimize use of a resource (time)

Reality

Time constraints

- [0800,1100] Illumination(Rock1)
- [1530,1600] Comm-window
- [0900,1630] Temperature > -20

✓ **Actions take time** ONP

Continuous change

- location
- energy consumption
- solar recharging

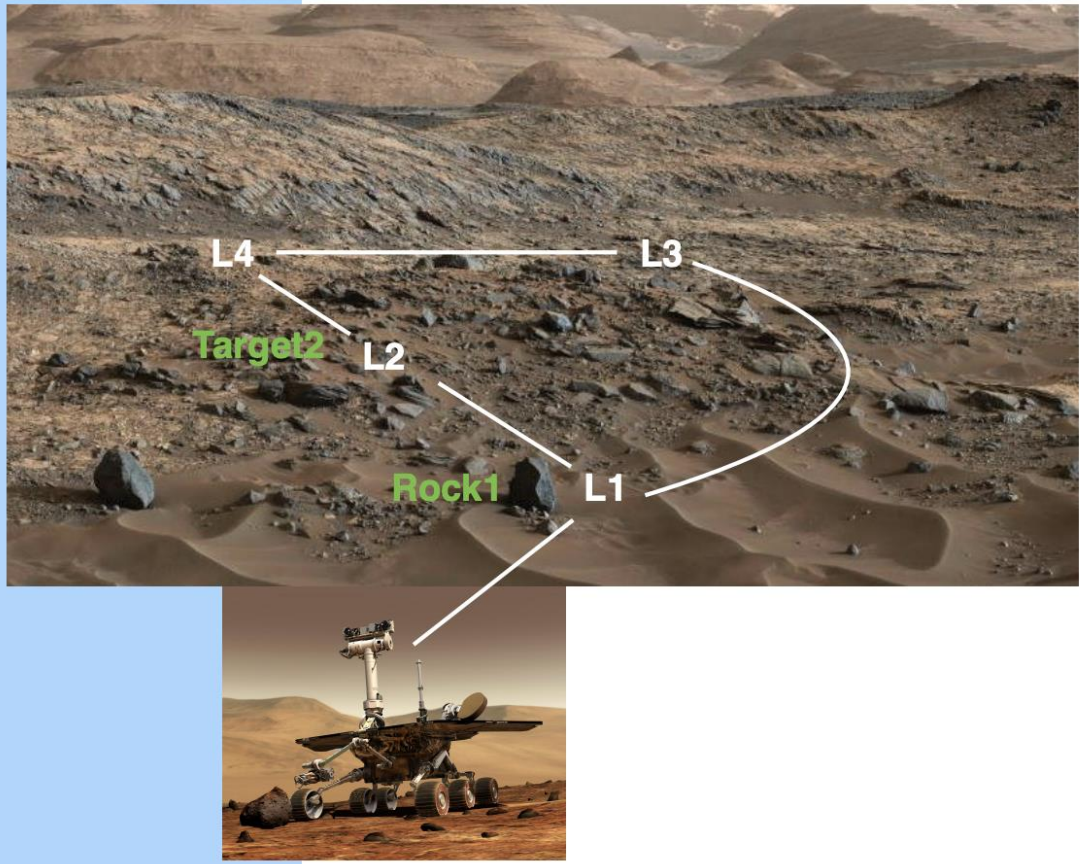
Concurrency

- heating
- sample processing
- communication

Goal constraints

- Deadlines: [...,1500] At(L3)
- Windows: [0900,1030] Image(Rock1)
- Ordering: Before(Image(Rock1), Sample(Target2))

Uncertainty!



Temporal Spectrum

Reality

Time constraints
- [0800,1100] Illumination(Rock1)
- [1530,1600] Comm-window
- [0900,1630] Temperature > -20

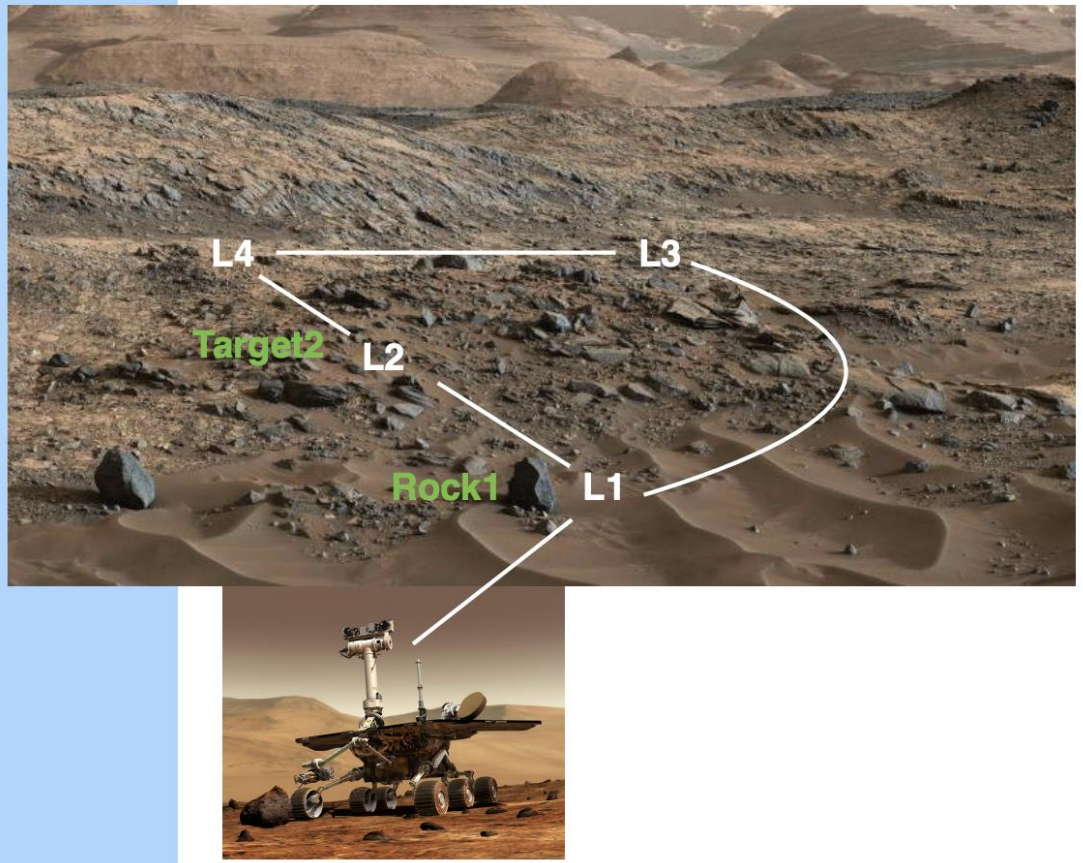
✓ **Actions take time** ONP

Continuous change
- location
- energy consumption
- solar recharging

Concurrency
- heating
- sample processing
- communication

→ **Goal constraints**
- Deadlines: [...,1500] At(L3)
- Windows: [0900,1030] Image(Rock1)
- Ordering: Image(Rock1) ≺ Sample(Target2)

Uncertainty!



Deadlines

Initial Conditions

At(L0)
Stowed
Empty
Charged

Path(L0,L1)
Visible(Rock1,L1)
Path(L1,L2)
Path(L2,L1)
Reachable(Target2,L2)
Path(L1,L3)
…

Actions

Cond
Navigate(?loc1, ?loc2)
Eff

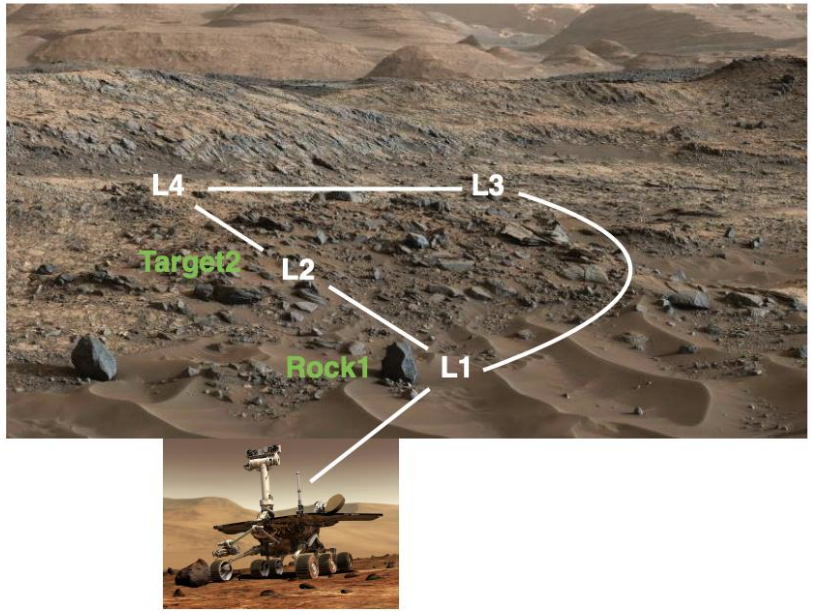
Cond
Snap(?target)
Eff

Cond
PlaceArm(?target)
Eff

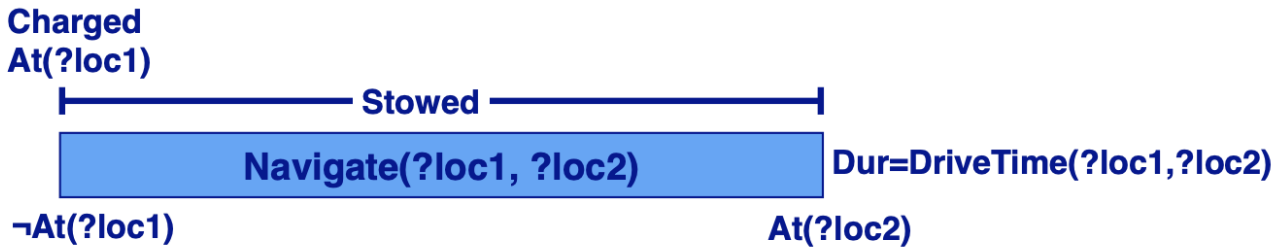
Cond
CollectSample(?target)
Eff

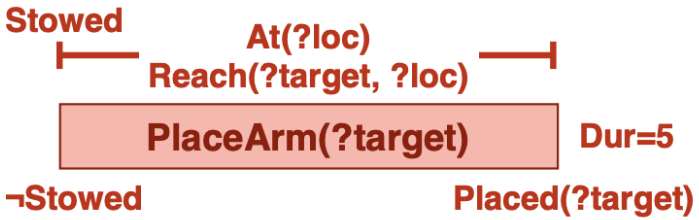
Goal(s)

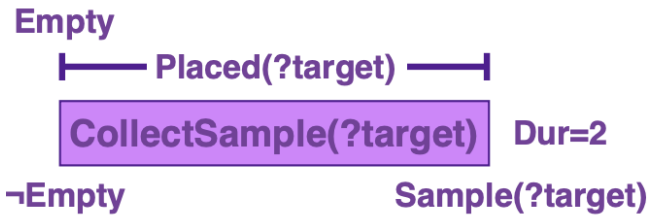
Image(Rock1)
Sample(Target2)
[…, 1500] At(L3)



Reduction to Optimal Numeric Planning



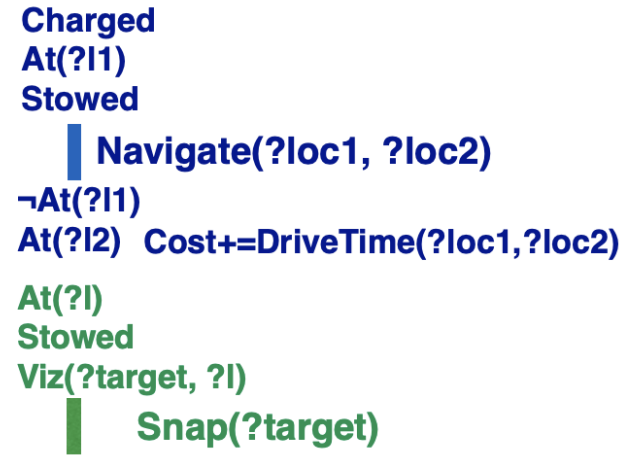




Squash

+ costs
+ metric

:functions (cost)
:init (= (cost) 0)
:metric minimize (cost)



:increase (cost) 2

Reduction to Optimal Numeric Planning

Run Numeric Planner
(minimizing cost)

Expand

Check makespan ≤ deadline!!

Numeric planning
– minimize use of a resource (time)

Works?

If makespan $\leq$ deadline

done!

Otherwise

not satisfiable

[…, 1500] At(L3)

Drawbacks

If makespan $\leq$ deadline

done!

Otherwise

not satisfiable

[…, 1500] At(L3)

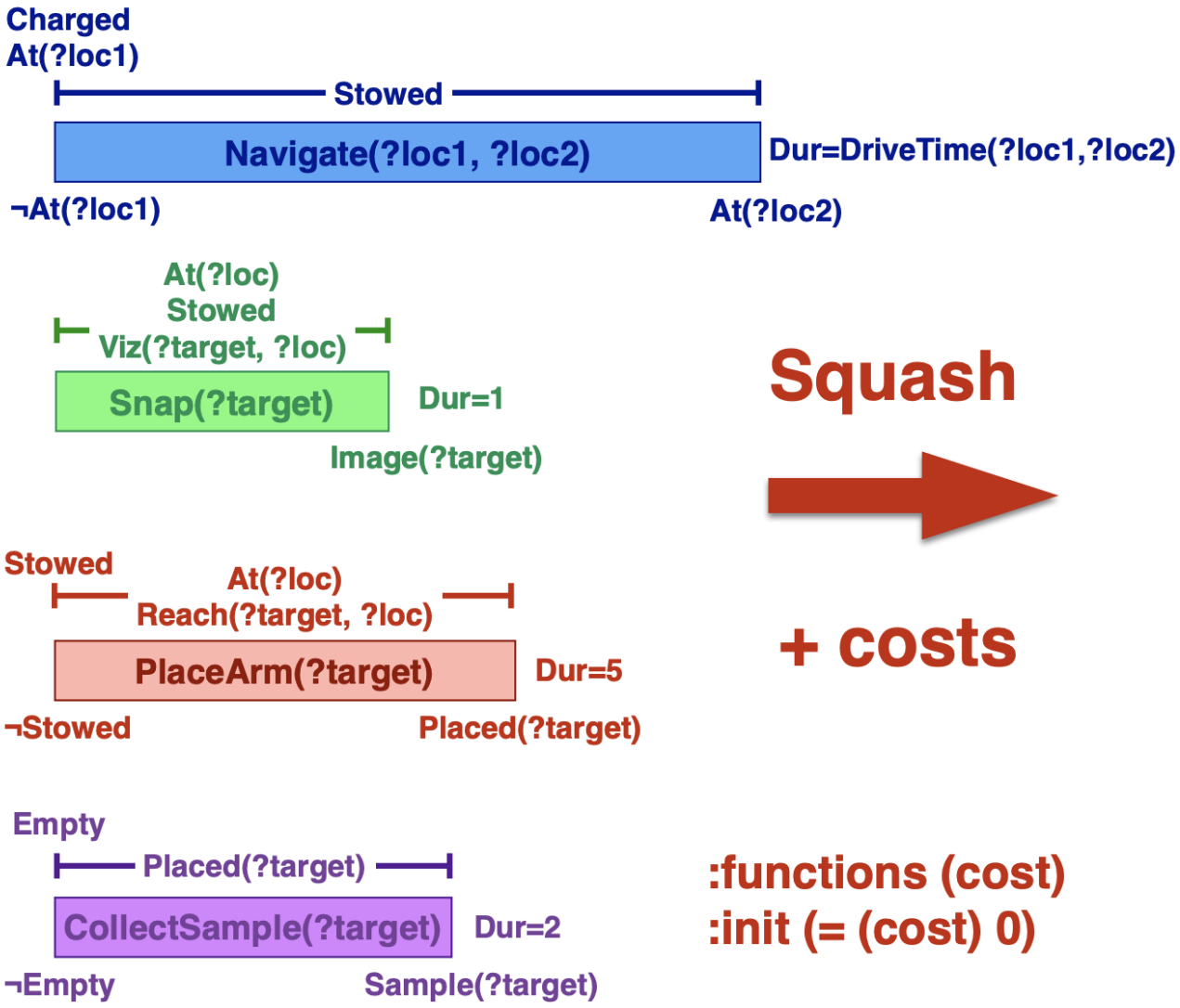
Overkill!

- **don't need minimal makespan**
- **deadline not on last goal**

$\leq$ 1500

[…, 1030] Snap(Rock1)

Numeric Planning + Pruning



Reduction to Numeric Planning

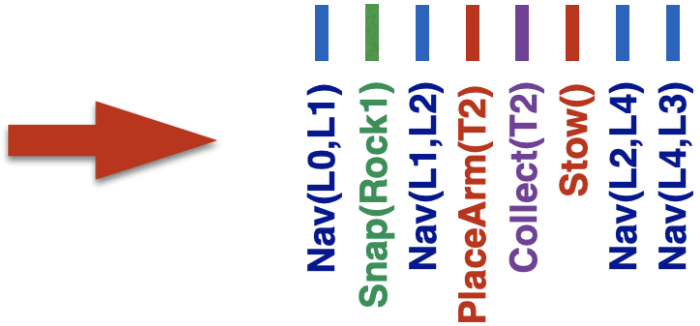
Expand

Numeric planning + Generate & test

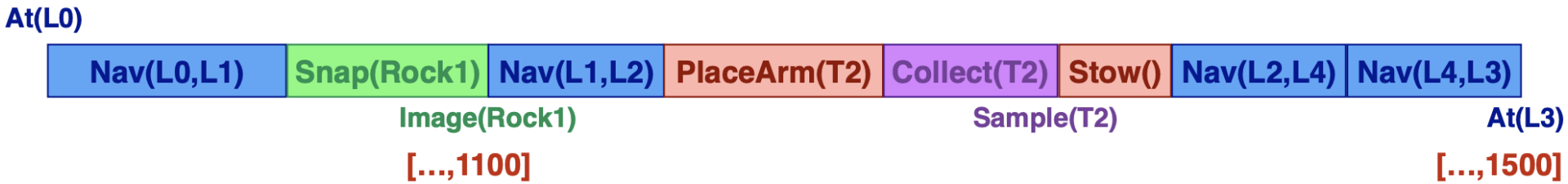
Reduction to Resource-constrained Numeric Planning

Run Numeric Planner

- Embedded checking
- Resource heuristics



Expand

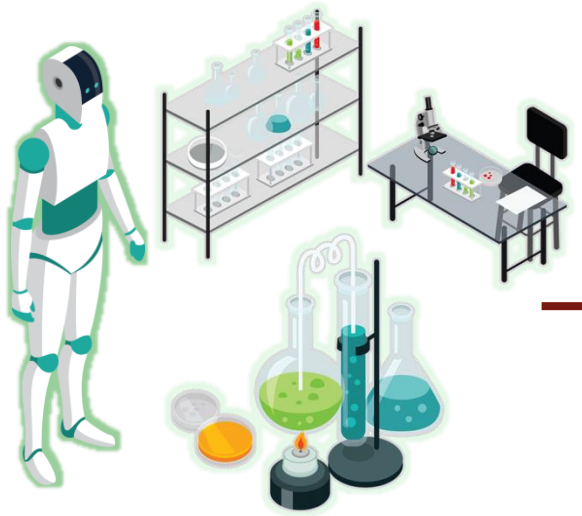


Numeric planning + smart resource checking

Learning PDDL Models

Reference: https://domain-learning.github.io/

Learning Action Models from Passive Observations



$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]

Learner

```
(:action pick-up-beaker
   :parameters (?x - beaker)
      :precondition (and (ontable ?x)
         (not-in-use ?x)
         (handempty))
      :effect (and (not (ontable ?x))
          (not (handempty))
          (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
      :precondition (holding ?x)
      :effect (and (not (holding ?x))
          (handempty)
          (onshelf ?x)))
```

[PDDL Example]

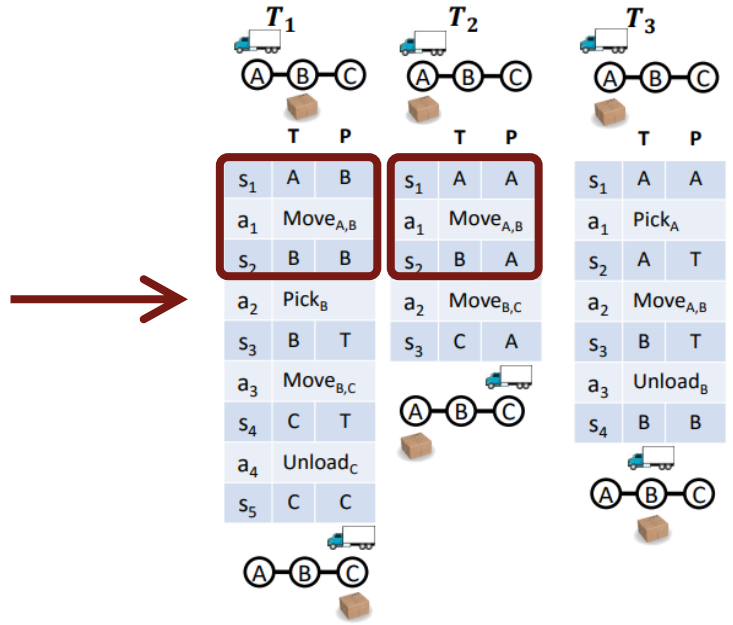
What kind of approaches these learners use?

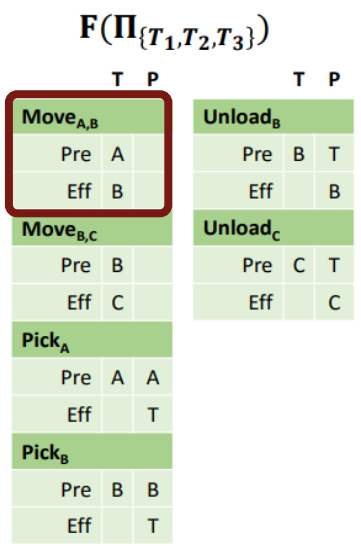
Inference Rules based Learners

- Take intersection of all states where an action is applicable to create precondition.
- Take intersection of all states after executing an action to create effect.

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]






```
(:action move-A-B
  :parameters ()
  :precondition (and (at A))
  :effect (and (not (at A)) (at B))
)

(:action move-B-C
  :parameters ()
  :precondition (and (at B))
  :effect (and (not (at B)) (at C))
)
```

[PDDL Example]

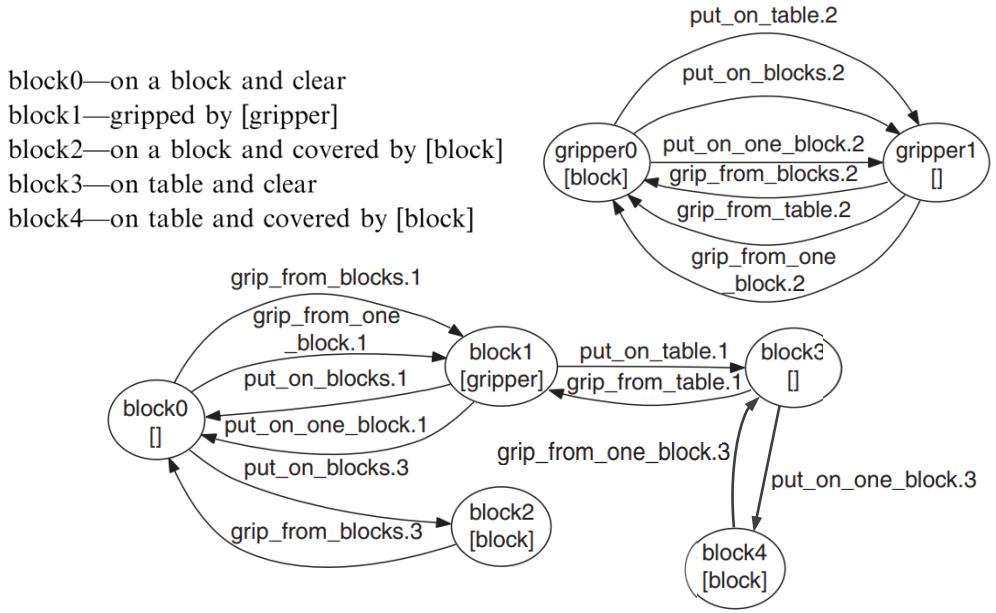
Stern et al. (IJCAI'17), SLAM - Juba et al. (KR'21)

Finite State Machine based Learners

- For each object type create a finite state machine.
- Create PDDL by combining them.

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]




```
(:action pick-up-beaker
   :parameters (?x - beaker)
      :precondition (and (ontable ?x)
         (not-in-use ?x)
         (handempty))
      :effect (and (not (ontable ?x))
          (not (handempty))
          (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
      :precondition (holding ?x)
      :effect (and (not (holding ?x))
          (handempty)
          (onshelf ?x)))
```

[PDDL Example]

LOCM, LOCM2 - Cresswell et al. (ICAPS'09 , ICAPS'11, Know. Engg. Rev.'13), LOP - Gregory et al. (ICAPS'15), NLOCM - Gregory et al. (ICAPS'16)

SAT based Learners

- Create a SAT problem using constraint axioms.
- Extract PDDL from the SAT problem's solution.

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]

1. $(par(p_k) \cap par(pri_i) = \phi) \wedge (par(p_k) \cap par(pre_{goal}) = \phi) \Rightarrow p_k \notin add_i \wedge p_k \notin del_i$
2. $pre_i \neq \phi \wedge add_i \neq \phi \wedge del_i \neq \phi$
3. $pre_i \cap add_i = \phi$
4. $del_i \subseteq pre_i$.

```
(:action pick-up-beaker
   :parameters (?x - beaker)
      :precondition (and (ontable ?x)
         (not-in-use ?x)
         (handempty))
      :effect (and (not (ontable ?x))
          (not (handempty))
          (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
      :precondition (holding ?x)
      :effect (and (not (holding ?x))
          (handempty)
          (onshelf ?x)))
```

[PDDL Example]

ARMS -Yang et al. (AIJ 2007), Zhuo et al. (IJCAI'13)

Planning based Learners

- Create Planning problem using SAT-like rules.
- Extract correct PDDL from solution to the planning problem.

$$\langle s_0, a_1, s_1 \rangle$$
$$\langle s_1, a_2, s_2 \rangle$$
$$\vdots$$
$$\langle s_{n-1}, a_n, s_n \rangle$$

[Input]

```
(:action apply_stack
  :parameters (?o1 - object ?o2 - object)
  :precondition
   (and (or (not (pre_stack_on_v1_v1)) (on ?o1 ?o1))
        (or (not (pre_stack_on_v1_v2)) (on ?o1 ?o2))
        (or (not (pre_stack_on_v2_v1)) (on ?o2 ?o1))
        (or (not (pre_stack_on_v2_v2)) (on ?o2 ?o2))
                    . . .
        (or (not (pre_stack_handempty)) (handempty)))
  :effect
   (and (when (del_stack_on_v1_v1) (not (on ?o1 ?o1)))
        (when (del_stack_on_v1_v2) (not (on ?o1 ?o2)))
        (when (del_stack_on_v2_v1) (not (on ?o2 ?o1)))
        (when (del_stack_on_v2_v2) (not (on ?o2 ?o2)))
                    . . .
        (when (add_stack_holding_v1) (holding ?o1))
        (when (add_stack_holding_v2) (holding ?o2))
        (when (add_stack_handempty) (handempty))
        (when (modeProg) (not (modeProg)))))
```

```
(:action pick-up-beaker
   :parameters (?x - beaker)
     :precondition (and (ontable ?x)
        (not-in-use ?x)
        (handempty))
     :effect (and (not (ontable ?x))
         (not (handempty))
         (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
     :precondition (holding ?x)
     :effect (and (not (holding ?x))
         (handempty)
         (onshelf ?x)))
```

[PDDL Example]

FAMA – Aineto et al. (ICAPS'18, AIJ'19)

Limitations of Learning from Passive Observations

- Susceptible to incorrect or incomplete model learning.
- E.g., if all packages are brown in color, a possible precondition will be that the package must be brown to unload them.
- Such methods don’t capture correct causal relationships.

Solution: Active/Online Acquisition of Observations