Lecture 26

CS 6380 Artificial Intelligence

Planning Domain Definition Language [PDDL]

Department of Computer Science and Engineering

Indian Institute of Technology Madras



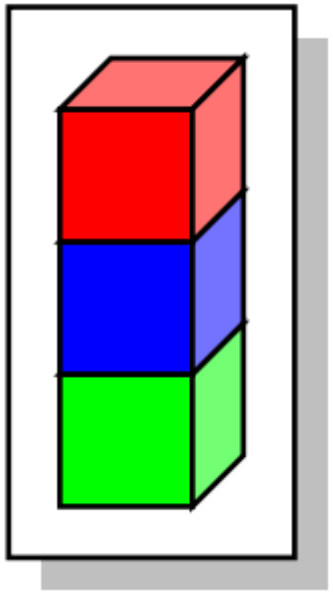
Some material based on lecture slides for ASU's CSE571 Artificial Intelligence by Siddharth Srivastava.

Blocks World*

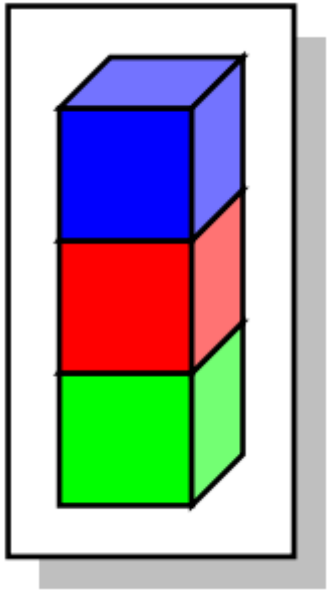
- A set of cube-shaped blocks on table.
- Blocks can be stacked on top of each other.
- Only one block can be stacked on top of another.
- A robot arm can move one block and
  - stack it on another block, or
  - keep the block on table
- Robot arm can pick up only one block at a time.
- Goal will be to stack blocks in some way.

* Source: **Section 10.1.3** of Russell and Norvig, Artificial Intelligence: A Modern Approach 3rd Ed.

Blocks World: Example



Initial State



Goal State

Image Source: Malte Helmert, Tutorial on Classical Planning, 3rd Summer School on Cognitive Robotics, 2019

Blocks World: Example

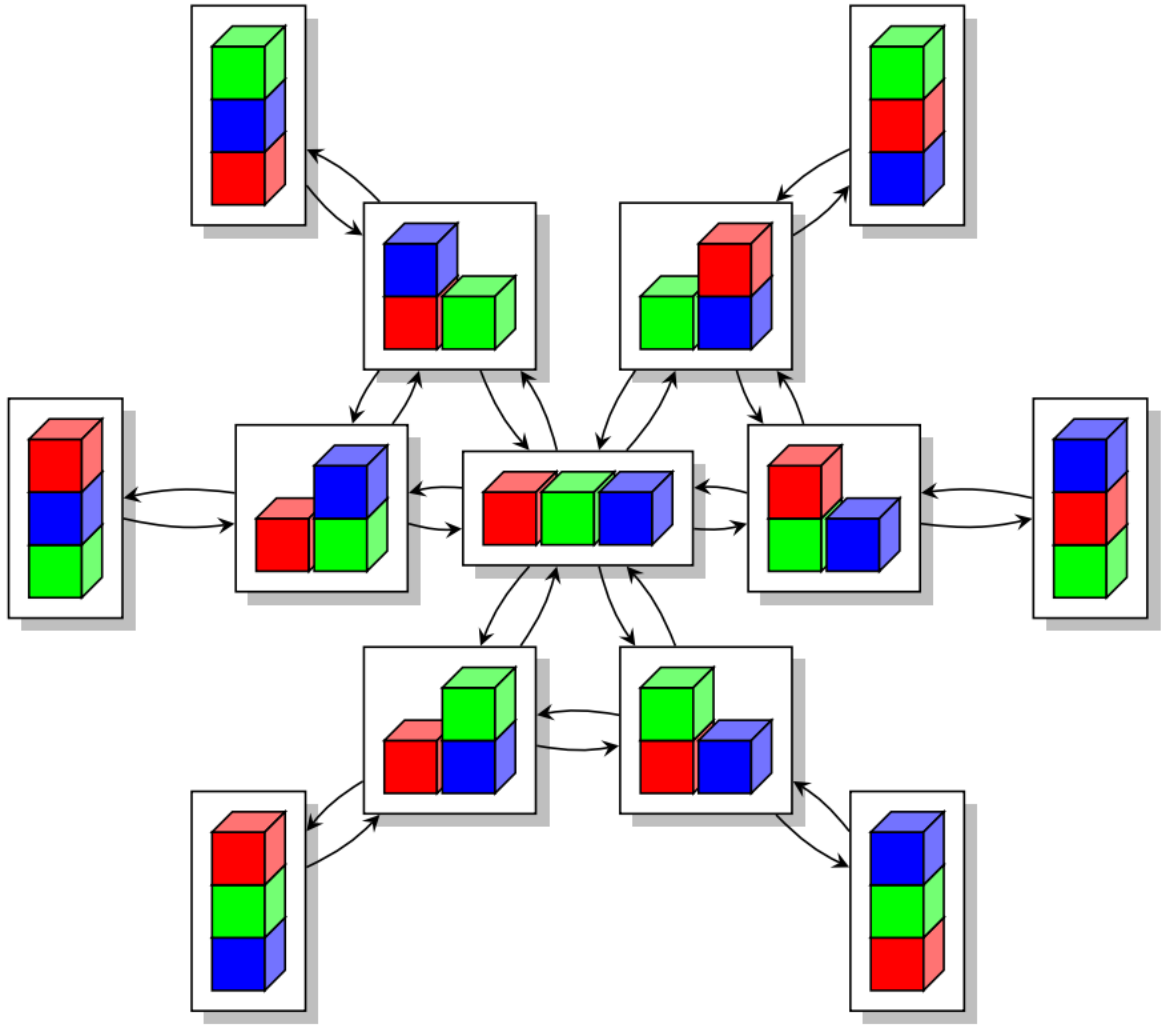
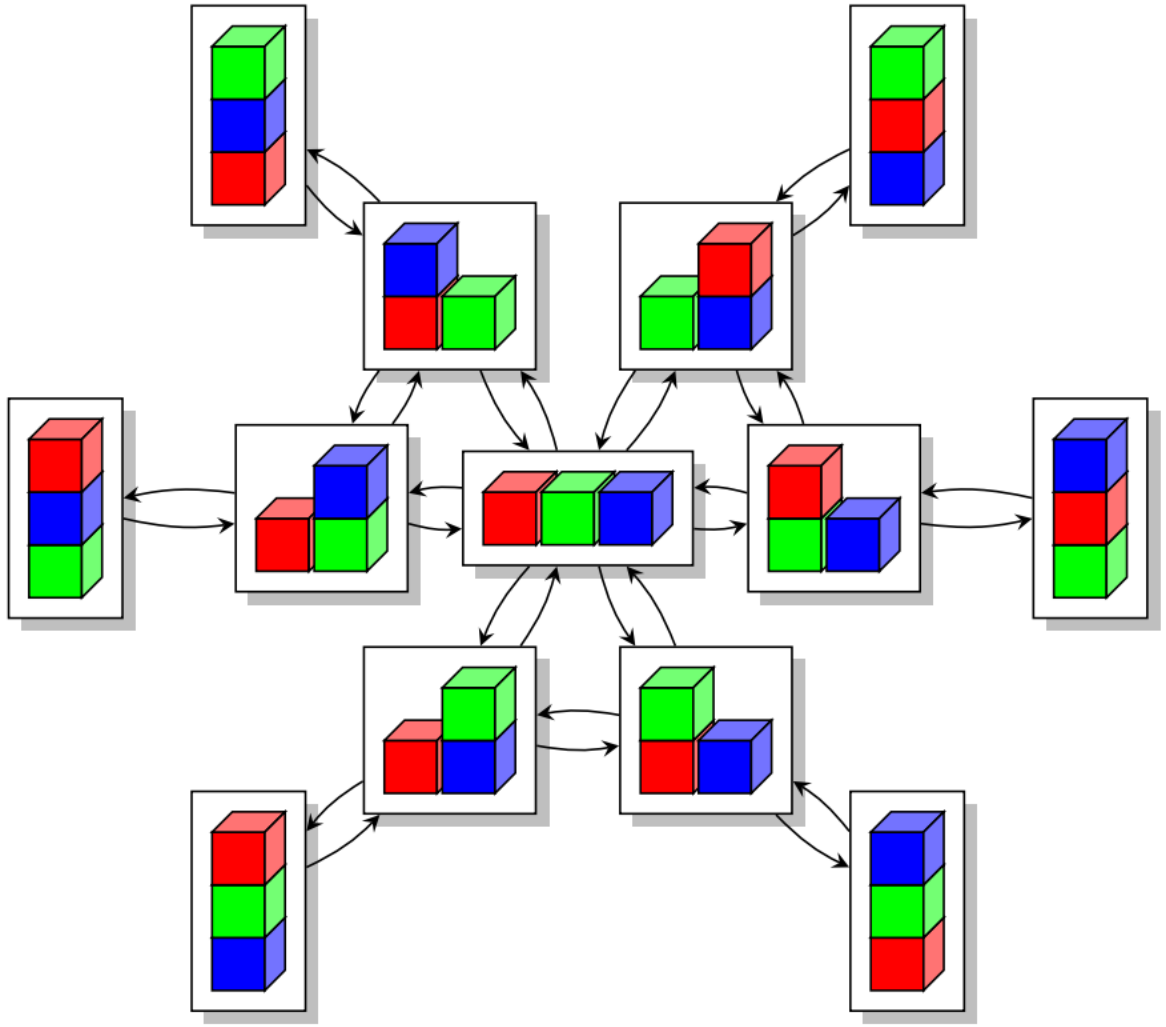


Image Source: Malte Helmert, Tutorial on Classical Planning, 3rd Summer School on Cognitive Robotics, 2019

Blocks World: Example



Side Note:
In automated planning, an **ergodic state space** means that every state is reachable from every other state, there's a path of actions that can take you from any state to any other state in the domain.

Blocks World is an example of ergodic state space.

Image Source: Malte Helmert, Tutorial on Classical Planning, 3rd Summer School on Cognitive Robotics, 2019

SHAKEY and STRIPS

- Original idea behind the STRIPS system language and system
  - Express states in FOL
  - Describe actions in a reduced language

***push*(*k*, *m*, *n*)**

Precondition: **ATR(*m*) ∧ AT(*k*, *m*)**

delete list **ATR(*m*); AT(*k*, *m*)**

add list **ATR(*n*); AT(*k*, *n*) .**

STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving[1]

Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

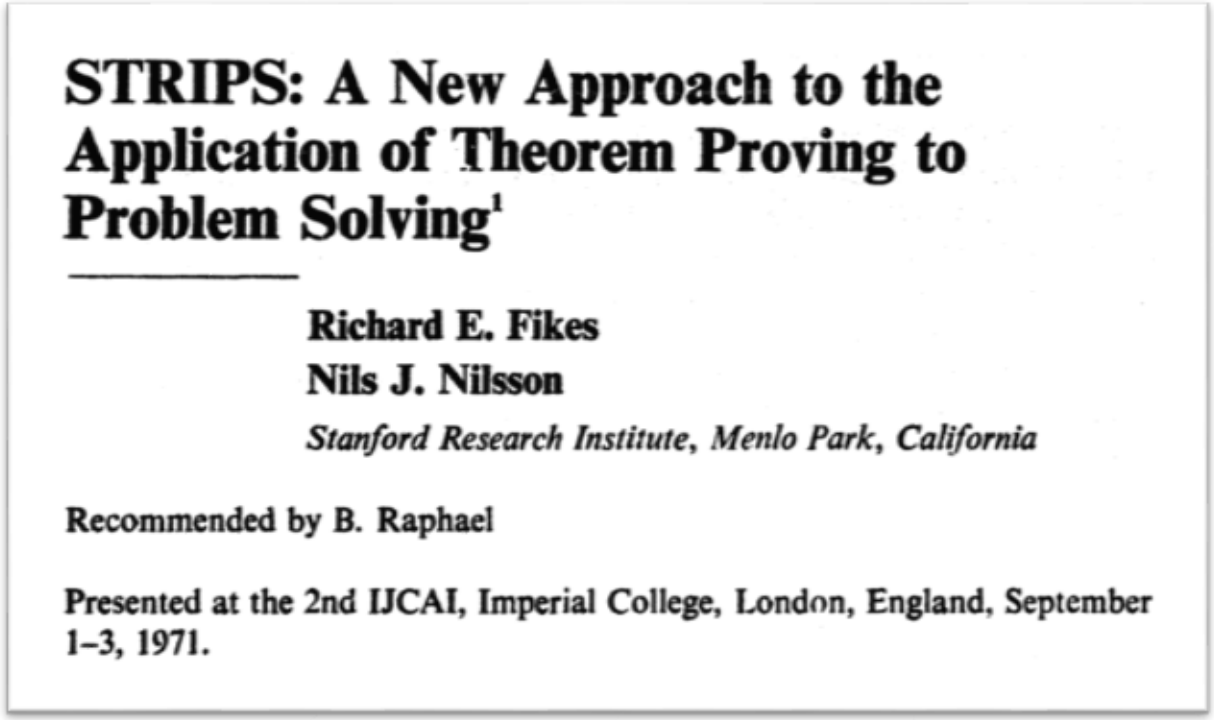
Recommended by B. Raphael

Presented at the 2nd IJCAI, Imperial College, London, England, September 1–3, 1971.

STanford Research Institute Problem Solver

SHAKEY and STRIPS

- Original idea behind the STRIPS system language and system
  - Express states in FOL
  - Describe actions in a reduced language
  - Use a theorem prover to select actions that may be useful
- The algorithmic approach was limited in scope, scalability
- The specification language turned out to be more influential



STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving[1]

Richard E. Fikes
Nils J. Nilsson
Stanford Research Institute, Menlo Park, California

Recommended by B. Raphael

Presented at the 2nd IJCAI, Imperial College, London, England, September 1–3, 1971.

STanford Research Institute Problem Solver

Issues with STRIPS

- Number of state variables is too high, each with a binary domain (True, False).

- For example: Consider blocks world,

state variables:

$$\{on(R,G),\quad on(R,B),$$
$$on(G,R),\quad on(G,B),$$
$$on(B,R),\quad on(B,G),$$
$$onTable(R),\quad onTable(G),$$
$$onTable(B),\quad clear(R),$$
$$clear(G),\quad clear(B),$$
$$handempty,\quad holding(R),$$
$$holding(G),\quad holding(B)\}$$

PDDL

- STRIPS based language.
- Components:
  - Objects
  - Predicates
  - Actions
  - Initial State
  - Goal State

Modern PDDL

- Over time, PDDL has evolved to accommodate much more than just STRIPS formulation.
- It now supports:
  - Ability to express a type structure for the objects in a domain.
  - Typing the parameters that appear in actions.
  - Constraining the types of arguments to predicates.
  - Actions with negative preconditions.
  - Conditional effects.

  And much more…

Modern Planning Domain Descriptions

Planning Domain Definition Language (PDDL)

- Domain:
  - Vocabulary
    - Constants, Predicates, Functions, Types
  - Action specifications
    - Precondition: FOL formula using the vocabulary
    - Effects: Facts that are added or removed by the action

```
(:predicates (on ?block1 ?block2)
             (ontable ?block)
             (clear ?block)
             (handempty)
             (holding ?block)
)
```

Modern Planning Domain Descriptions

Planning Domain Definition Language (PDDL)

- Domain:
  - Vocabulary
    - Constants, Predicates, Functions, Types
  - Action specifications
    - Precondition: FOL formula using the vocabulary
    - Effects: Facts that are added or removed by the action

```
(:predicates (on ?block1 ?block2)
             (ontable ?block)
             (clear ?block)
             (handempty)
             (holding ?block)
)
```

```
(:action pick-up
         :parameters

         :precondition

         :effect
)
```

"Robot should pick up a block from table and its hand should be empty. After picking up, robot is holding the block, the block is not on table, robot's hand is not empty, and block is not clear anymore."

Modern Planning Domain Descriptions

Planning Domain Definition Language (PDDL)

- Domain:
  - Vocabulary
    - Constants, Predicates, Functions, Types
  - Action specifications
    - Precondition: FOL formula using the vocabulary
    - Effects: Facts that are added or removed by the action

```
(:predicates (on ?block1 ?block2)
             (ontable ?block)
             (clear ?block)
             (handempty)
             (holding ?block)
)
```

```
(:action pick-up
         :parameters (?block)

         :precondition

         :effect
)
```

"Robot should pick up a block from table and its hand should be empty. After picking up, robot is holding the block, the block is not on table, robot's hand is not empty, and block is not clear anymore."

Modern Planning Domain Descriptions

Planning Domain Definition Language (PDDL)

- Domain:
  - Vocabulary
    - Constants, Predicates, Functions, Types
  - Action specifications
    - Precondition: FOL formula using the vocabulary
    - Effects: Facts that are added or removed by the action

"Robot should pick up a block from table and its hand should be empty. After picking up, robot is holding the block, the block is not on table, robot's hand is not empty, and block is not clear anymore."

```
(:predicates (on ?block1 ?block2)
             (ontable ?block)
             (clear ?block)
             (handempty)
             (holding ?block)
)
```

```
(:action pick-up
         :parameters (?block)
         :precondition
         (and (clear ?block)
              (ontable ?block)
              (handempty)
         )
         :effect
)
```

Modern Planning Domain Descriptions

Planning Domain Definition Language (PDDL)

- Domain:
  - Vocabulary
    - Constants, Predicates, Functions, Types
  - Action specifications
    - Precondition: FOL formula using the vocabulary
    - Effects: Facts that are added or removed by the action

```
(:predicates (on ?block1 ?block2)
             (ontable ?block)
             (clear ?block)
             (handempty)
             (holding ?block)
)
```

```
(:action pick-up
         :parameters (?block)
         :precondition
         (and (clear ?block)
              (ontable ?block)
              (handempty))
         :effect
         (and (not (ontable ?block))
              (not (clear ?block))
              (not (handempty))
              (holding ?block)))
)
```

"Robot should pick up a block from table and its hand should be empty. After picking up, robot is holding the block, the block is not on table, robot's hand is not empty, and block is not clear anymore."

Modern Planning Domain Descriptions

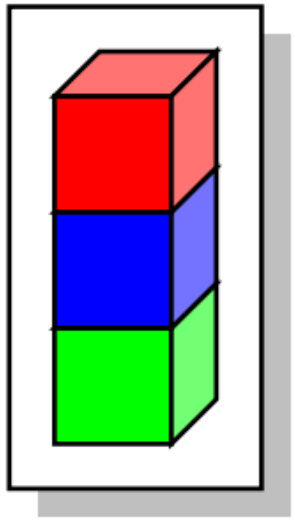
```
(:action put-down
    :parameters (?block)
    :precondition (holding ?block)
    :effect
    (and (not (holding ?block))
        (clear ?block)
        (handempty)
        (ontable ?block))
)
```

```
(:action stack
    :parameters (?block1 ?block2)
    :precondition
    (and (holding ?block1)
        (clear ?block2))
    :effect
    (and (not (holding ?block1))
        (not (clear ?block2))
        (clear ?block1)
        (handempty)
        (on ?block1 ?block2))
)
```

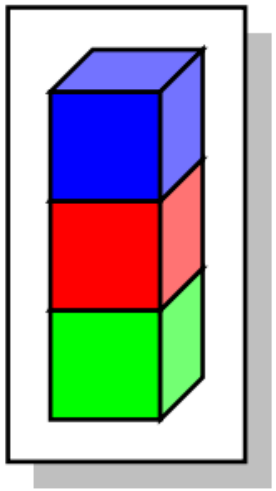
```
(:action unstack
    :parameters (?block1 ?block2)
    :precondition
    (and (on ?block1 ?block2)
        (clear ?block1)
        (handempty))
    :effect
    (and (holding ?block1)
        (clear ?block2)
        (not (clear ?block1))
        (not (handempty))
        (not (on ?block1 ?block2)))
)
```

PDDL Problem Description

```
(:objects R B G)
```



```
(:INIT   (CLEAR R)
         (ONTABLE G)
         (ON R B)
         (ON B G)
         (HANDEMPTY)
)
```



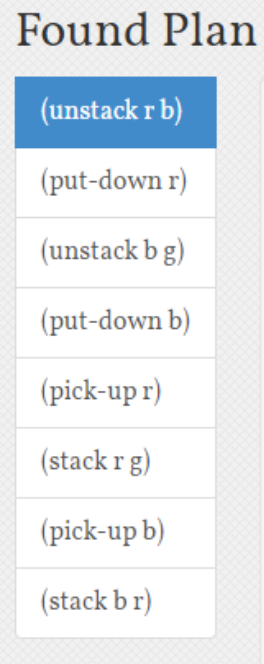
```
(:goal (AND (ONTABLE G)
            (ON R G)
            (ON B R))
)
```

- (unstack r b)
- (put-down r)
- (unstack b g)
- (put-down b)
- (pick-up r)
- (stack r g)
- (pick-up b)
- (stack b r)

PDDL Editors

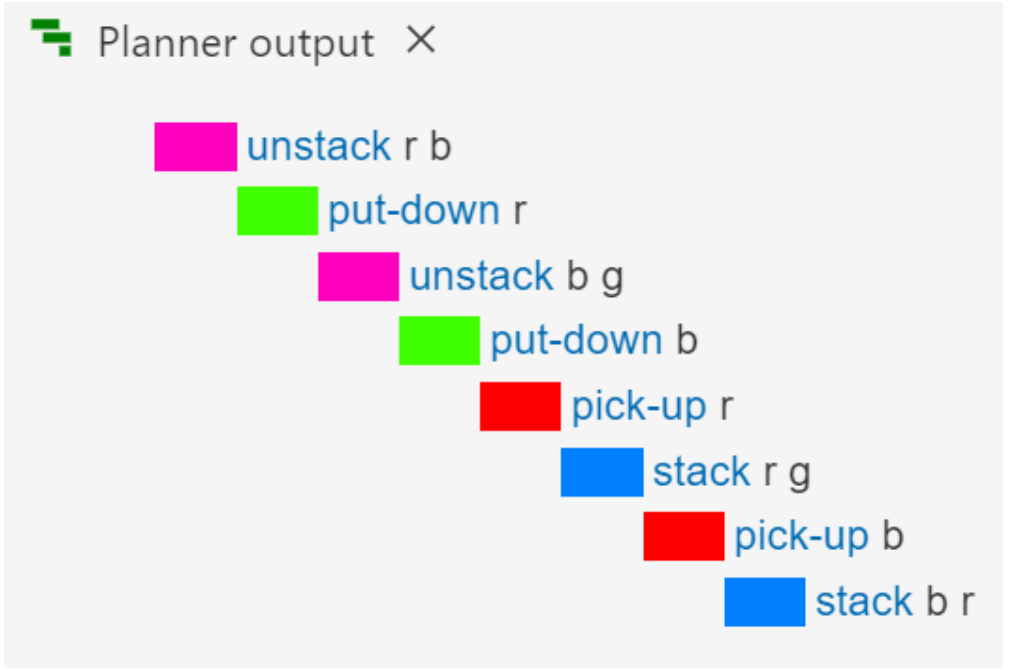
Planning.Domains

(http://planning.domains/)



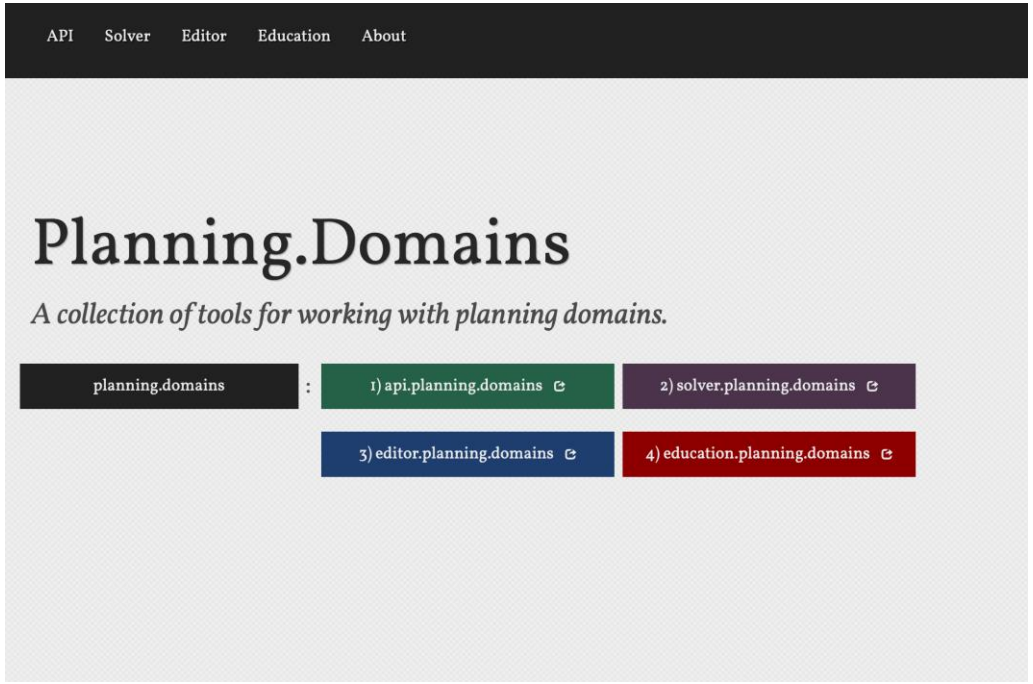
VS Code + PDDL Extension

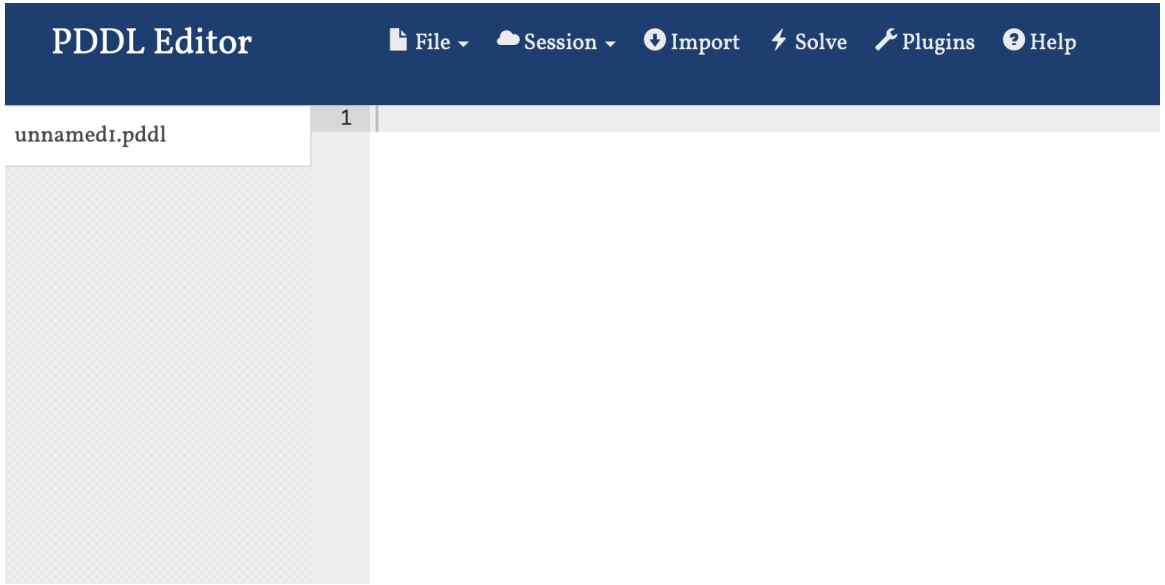
(https://code.visualstudio.com/download + https://marketplace.visualstudio.com/items?itemName=jan-dolejsi.pddl)



Using Planning.domains

- Use a browser to open the online planner/editor: http://planning.domains
- Go to “Editor” on top





Advantage of PDDL

- Independent Problem and Domain representation.

- Go to http://editor.planning.domains/
- Click on Import -> Choose IPC-2000 -> Choose Domain “blocks”
  - Single Domain File.
  - Multiple Problem Files. Each has a different set of objects, initial state, and goal condition/formula.

† Credits: 2nd International Planning Competition, 2000

Advantage of PDDL

- Independent Problem and Domain representation.

- Go to http://editor.planning.domains/
- Click on Import -> Choose IPC-2000 -> Choose Domain “blocks”
  - Single Domain File.
  - Multiple Problem Files. Each has a different set of objects, initial state, and goal condition/formula.

† Credits: 2nd International Planning Competition, 2000

Satisficing vs Optimal Plans

- Optimal Plans:
  - Least cost to reach the goal from the initial state
  - Relatively difficult to generate
  - Can be more than one

- Satisficing Plans:
  - Solve the problem (reaches the goal), but not optimal
  - Relatively easier to generate
  - Number of satisficing plans $\geq$ Number of optimal plans

Heuristics leveraging PDDL

Automated Search Control

- PDDL Domain + PDDL Problem
  = Search problem definition
- Most popular solution approach:
  - Informed search without hand-coded heuristic
  - *Pseudo-informed* search?
- Heuristics, pruning strategies derived automatically
  - Using domain independent algorithms

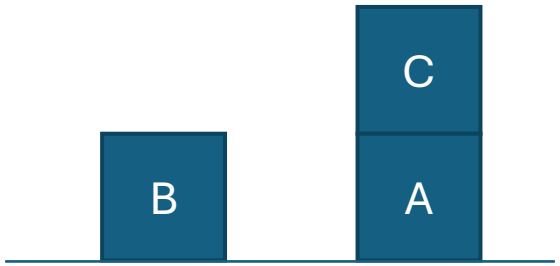
- A search problem is defined by
  - S set of possible states
    - Defined, but not enumerated a priori
  - $s_0 \in S$ start state (~vertex)
  - A set of actions
  - $N: S \rightarrow S$ successor function
    - $T: S \times A \rightarrow S$ deterministic transition function
    - $N(s) = \{s' : \exists a \, T(s,a) = s'\}$
  - $G: S \rightarrow \{True, False\}$ goal test
  - $C: S \times A \times S \rightarrow \mathbb{R}^+$ path cost

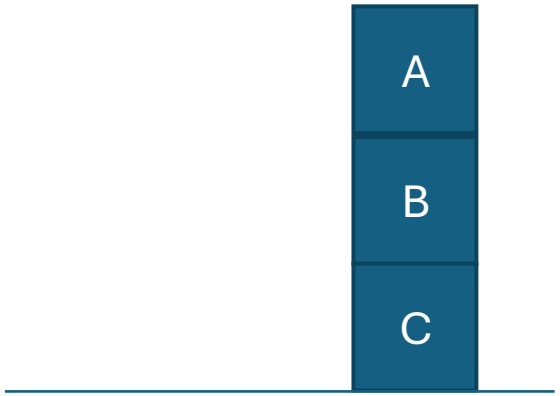
Ideas for Generating Heuristics and Search Control

- Goal decomposition (not quite)
- Planning graph
- Landmarks

Goal Decomposition

- Early approach motivated by the divide-and-conquer principle
- If the goal requires multiple properties, achieve each one independently
- What could go wrong?
- Sussman's anomaly

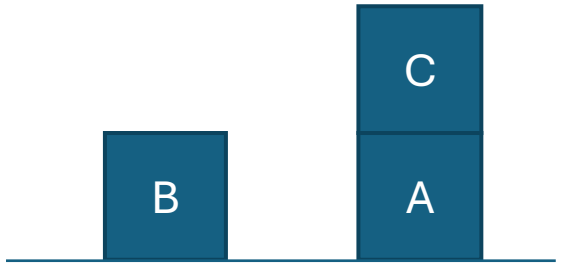




Goal requires: on(A, B); on(B,C)

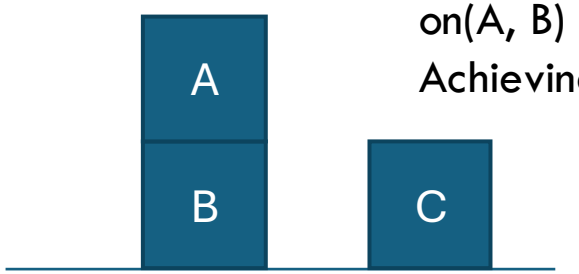
Goal Decomposition: Sussman's Anomaly

Goal requires: on(A, B); on(B,C)
What happens if we try to achieve them independently?

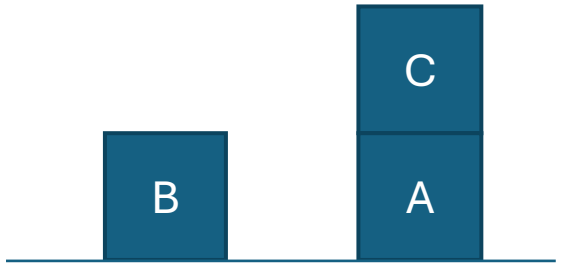


Try to achieve: on(A, B)

Move(C, table); Move(A, B)

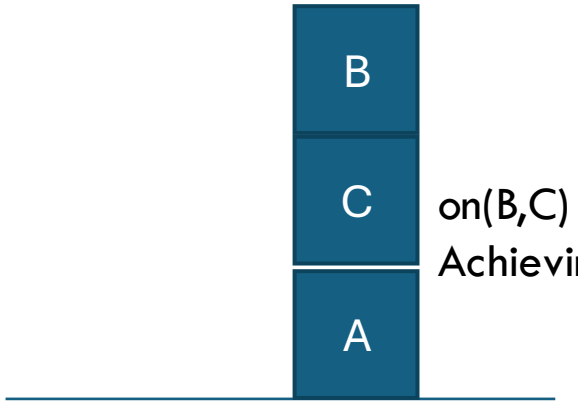


on(A, B) achieved ☺
Achieving on(B,C) will undo it ☹



Try to achieve: on(B, C)

Move(B, C)



on(B,C) achieved ☺
Achieving on(A,B) will undo it ☹

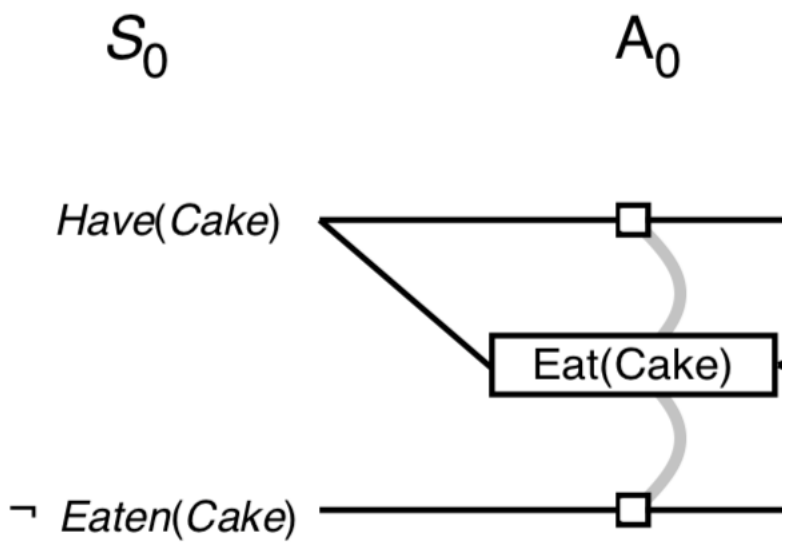
Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg\, Have(Cake)$
  EFFECT: $Have(Cake)$



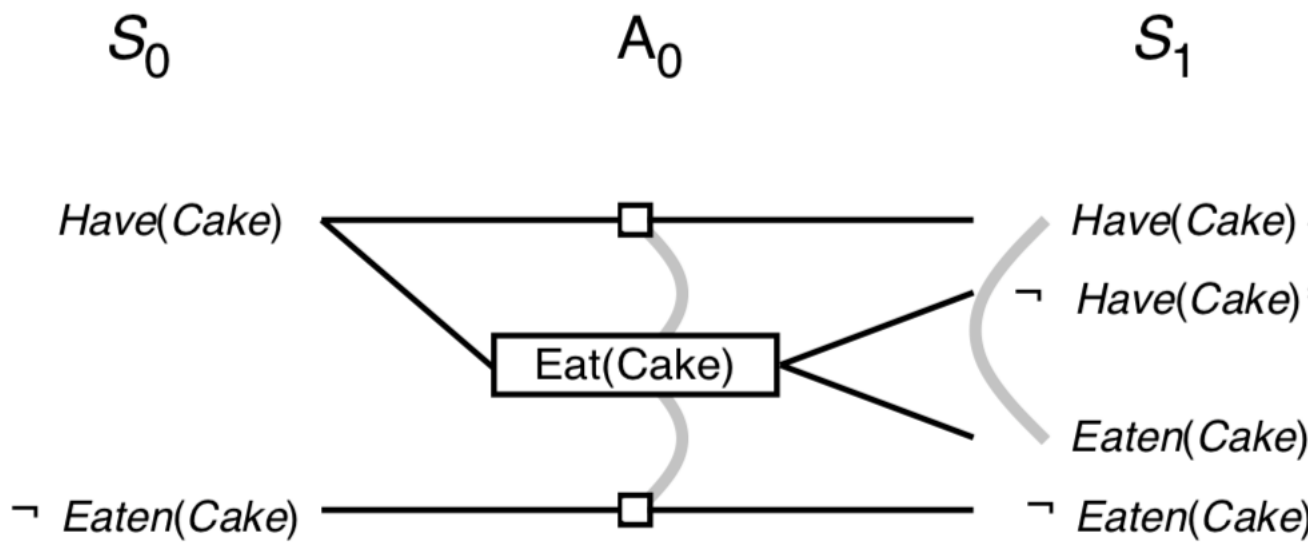
Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg\, Have(Cake)$
  EFFECT: $Have(Cake)$



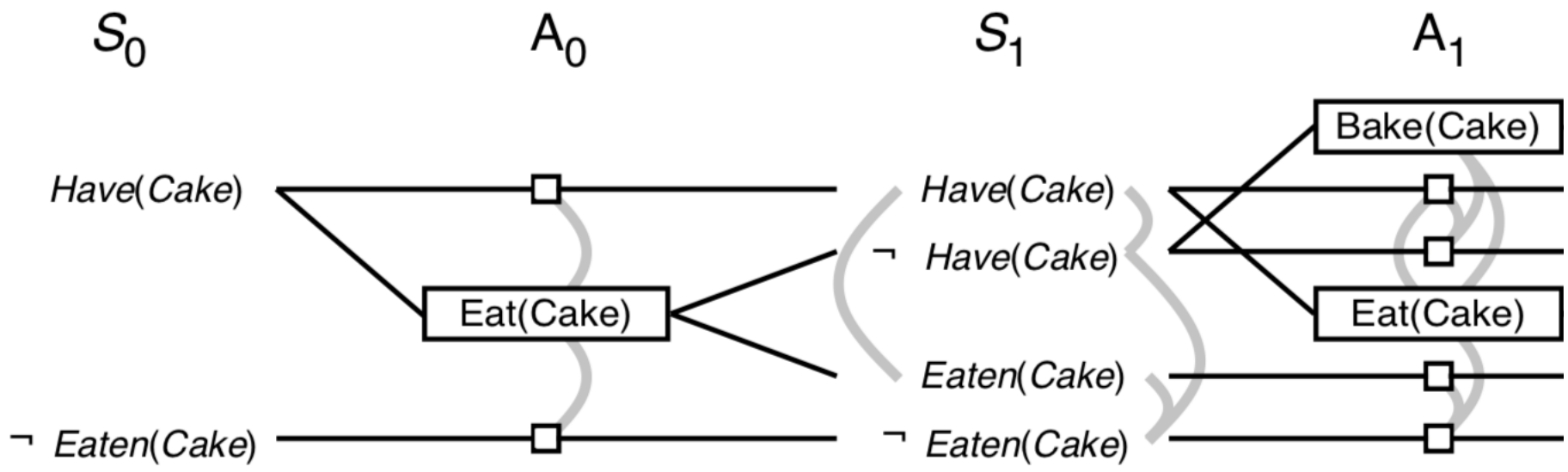
Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg\, Have(Cake)$
  EFFECT: $Have(Cake)$



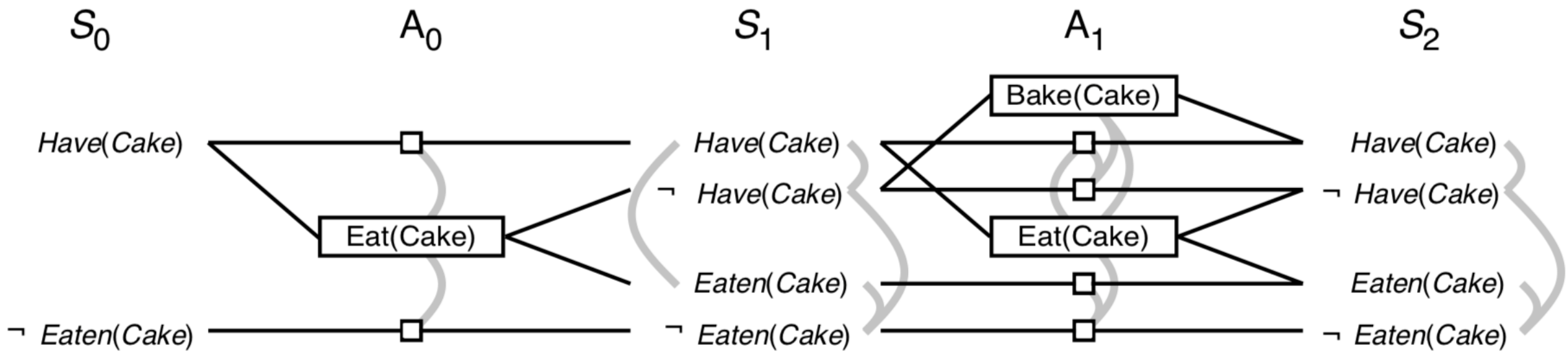
Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$



Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$



Planning Graph

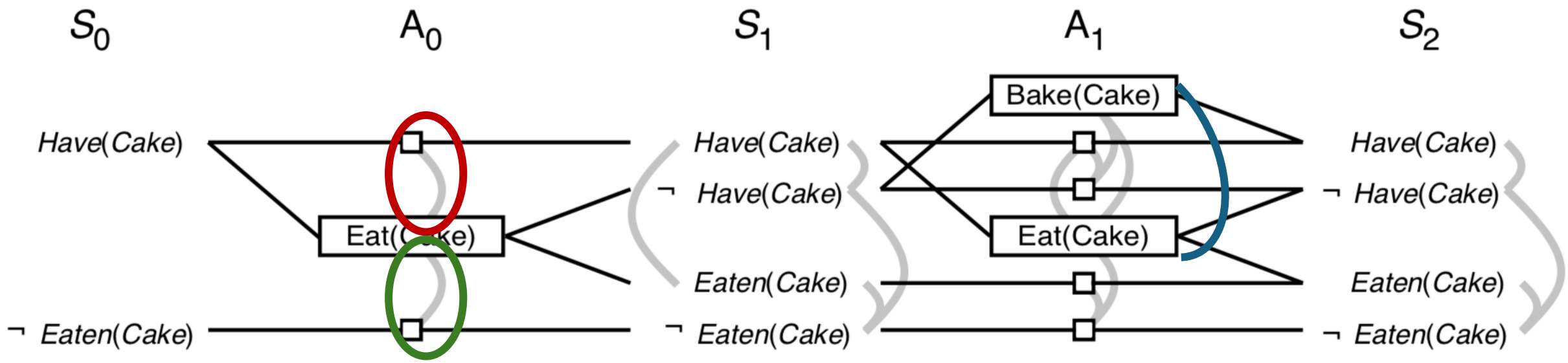
$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

Actions in a layer are *mutex* if:

Inconsistent Effects: one action's effect negates another's effect

Interference: one action negates another's precondition

Competing needs: one action requires negation of another's precondition



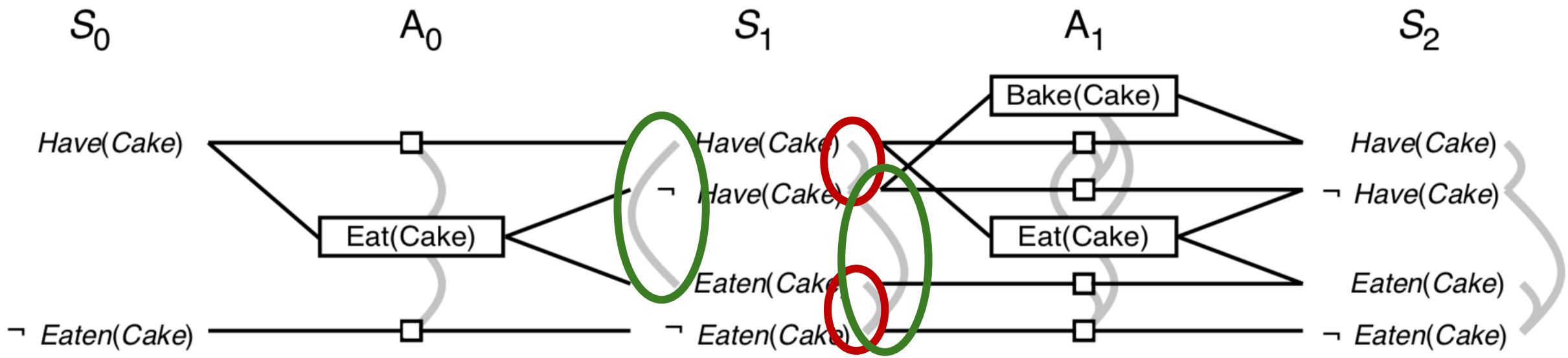
Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

Propositions in a layer are *mutex* if:

Negation: one negates the other

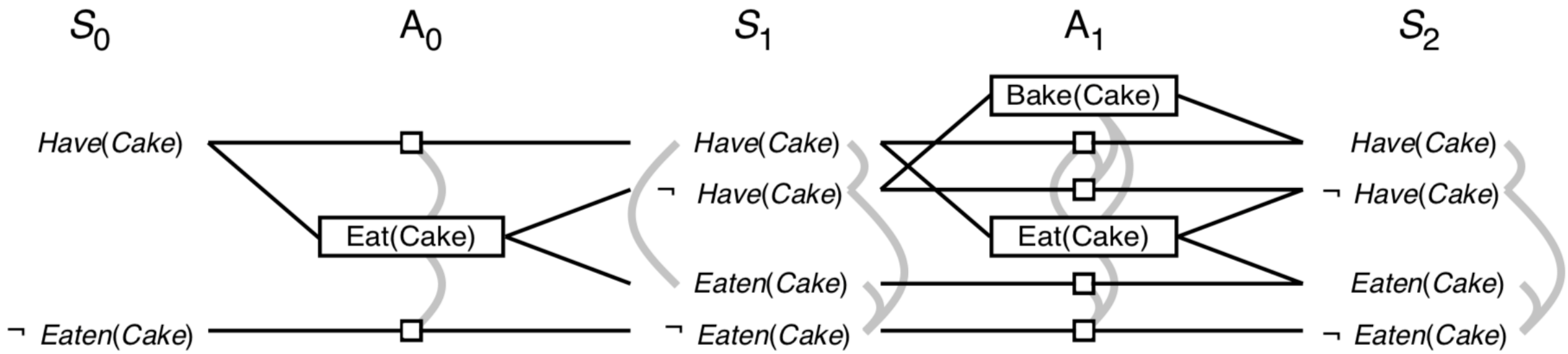
Inconsistent Support: all ways of achieving them are pairwise mutex



Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

- Mutexes are calculated for all pairs of actions/propositions at a level.



Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg\, Have(Cake)$
  EFFECT: $Have(Cake)$

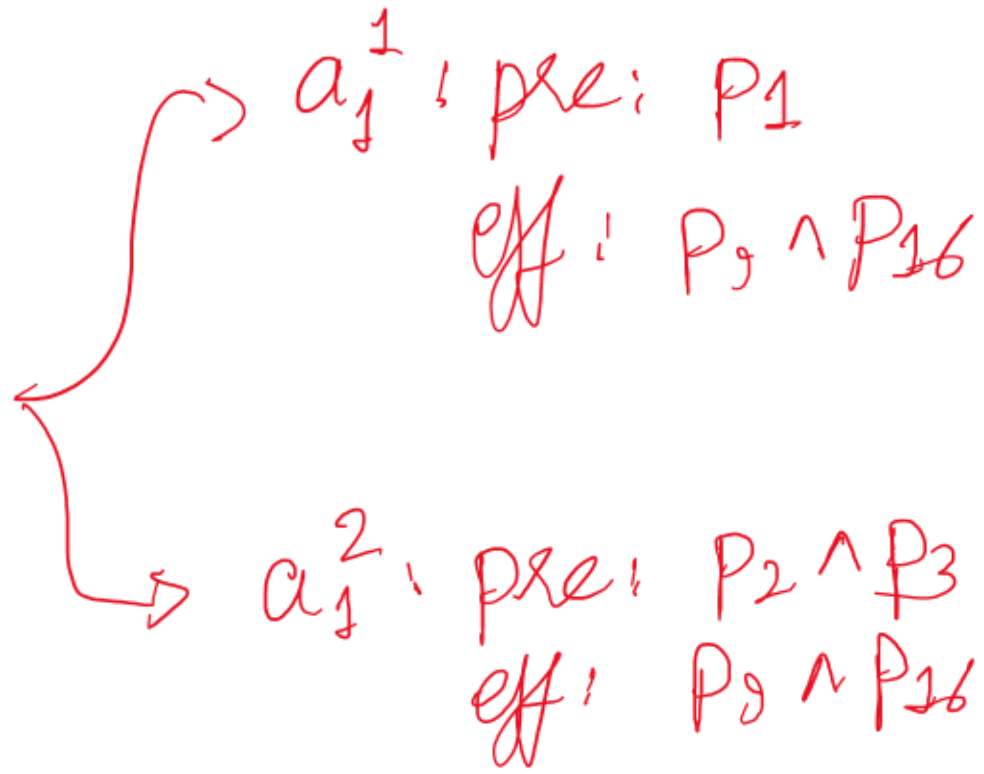
- If disjunction of propositions in an action's precondition?
  - Equivalent to 2 actions, each with a different part of the disjunction as its precondition.

Planning Graph

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
PRECOND: $Have(Cake)$
EFFECT: $\neg\, Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
PRECOND: $\neg\, Have(Cake)$
EFFECT: $Have(Cake)$

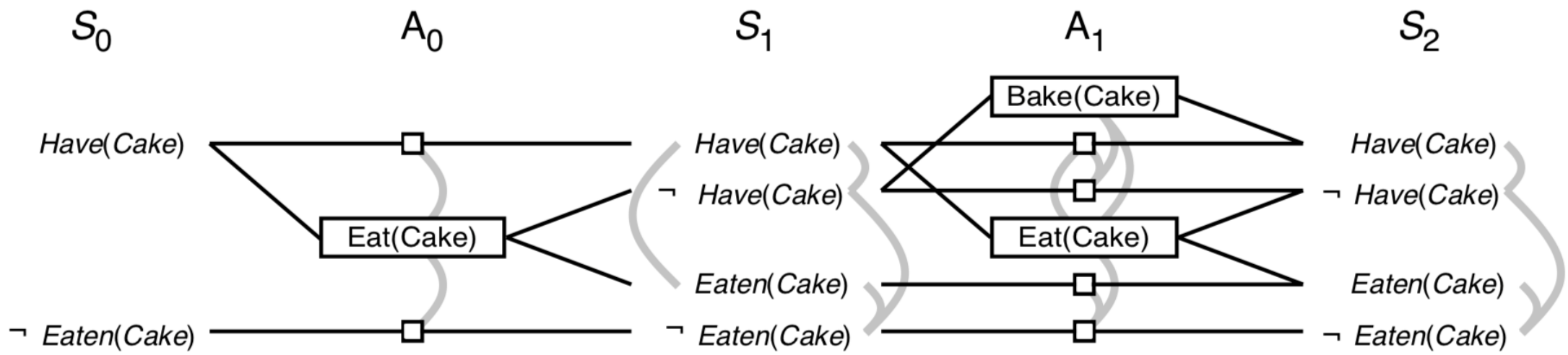
- If disjunction of propositions in an action's precondition?
  - Equivalent to 2 actions, each with a different part of the disjunction as its precondition.

a_1 : pre: $P_1 \vee (P_2 \wedge P_3)$
eff: $P_9 \wedge P_{16}$



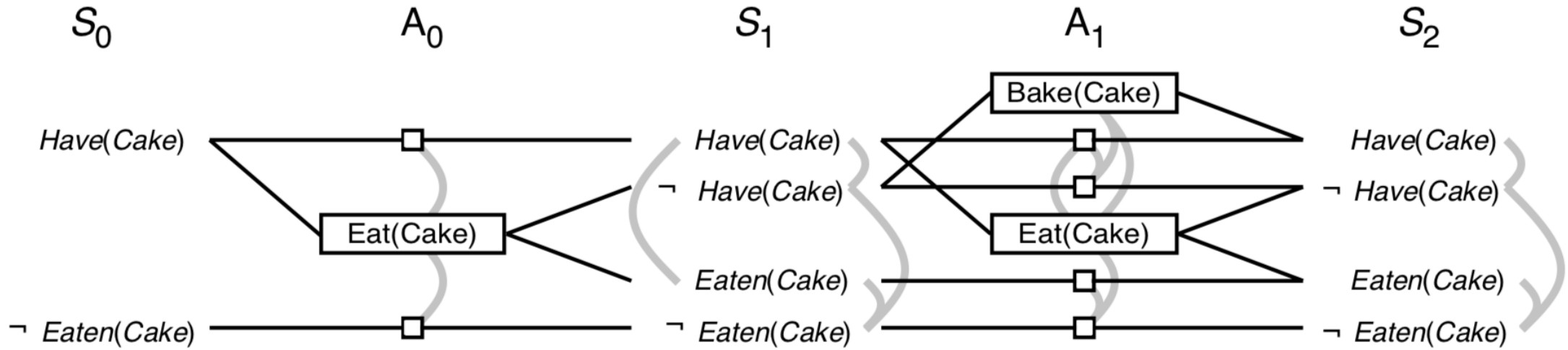
Extracting Heuristics From the Planning Graph

- Mutex relations can be used to try to extract plans that achieve the goal
- It is possible that a plan is not feasible within a given expansion of graphplan
  - Add another layer, try again
  - Until the graph levels off
  - If all the goal literals do not appear in the final level, goal is unreachable



Extracting Heuristics From the Planning Graph

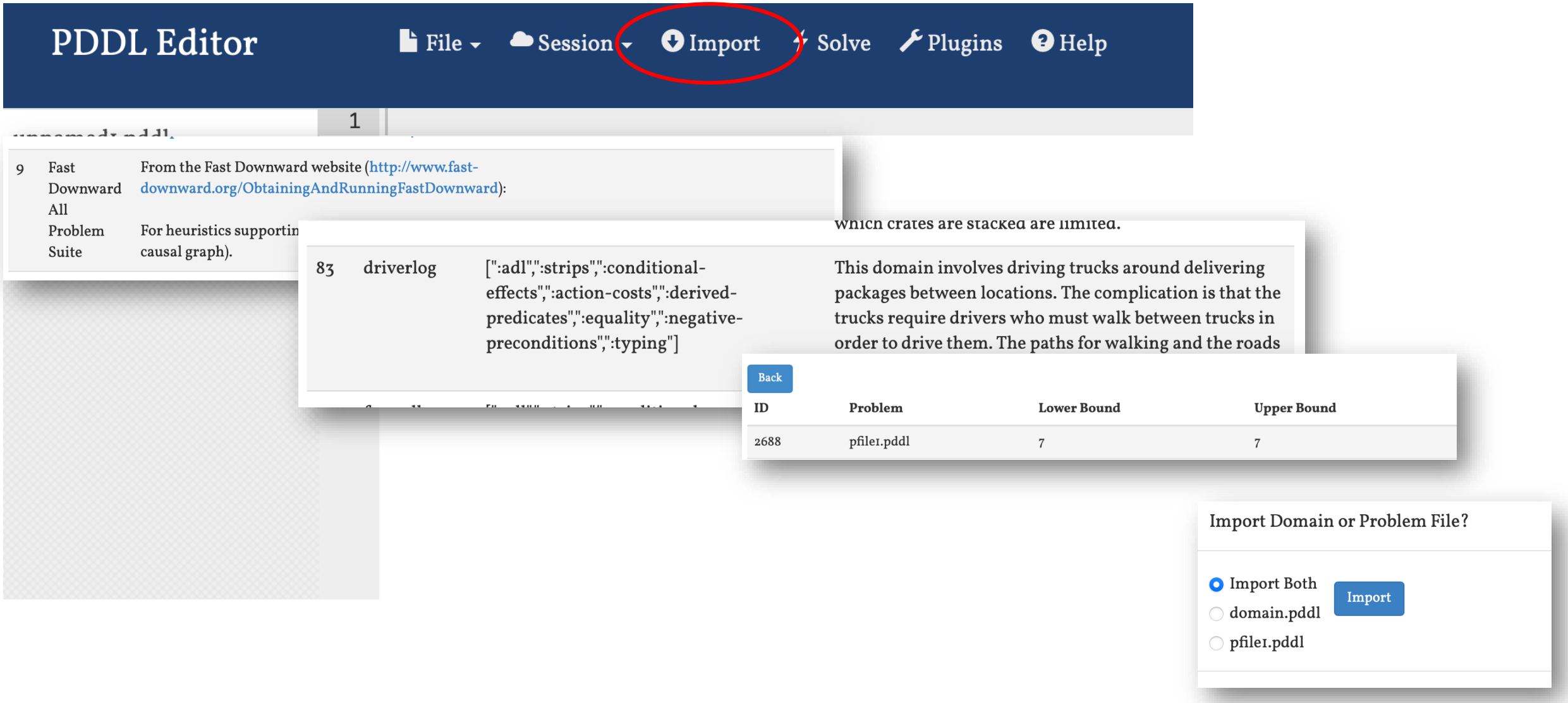
- Expand the graph until you reach a fixed point
- If a literal doesn't occur in the last layer, it cannot be achieved
- Level-cost heuristic
  - Estimated cost of achieving a goal literal = level at which it first appears
  - Is this admissible?
- What about goals that include multiple literals?
  - Max-level heuristic
  - Level-sum heuristic
    - Is this optimistic (admissible) or pessimistic (non-admissible)?



Additional PDDL Example

Another Example: Driverlog

Import→ Fast Downward All problem suite → driverlog → pfile 1→ import both



```
(define (problem DLOG-2-2-2)
    (:domain driverlog)
    (:objects
    driver1
    driver2
    truck1
    truck2
    package1
    package2
    s0
    s1
    s2
    p1-0
    p1-2
    )
    (:init
    (at driver1 s2)
    (DRIVER driver1)
    (at driver2 s2)
    (DRIVER driver2)
    (at truck1 s0)
    (empty truck1)
    (TRUCK truck1)
    (at truck2 s0)
    (empty truck2)
    (TRUCK truck2)
    (at package1 s0)
    (OBJ package1)
    (at package2 s0)
    (OBJ package2)
    (LOCATION s0)
    (LOCATION s1)
    (LOCATION s2)
    (LOCATION p1-0)
    (LOCATION p1-2)
    (path s1 p1-0)
    (path p1-0 s1)
    (path s0 p1-0)
    (path p1-0 s0)
    (path s1 p1-2)
    (path p1-2 s1)
    (path s2 p1-2)
    (path p1-2 s2)
    (link s0 s1)
    (link s1 s0)
    (link s0 s2)
    (link s2 s0)
    (link s2 s1)
    (link s1 s2)
)
    (:goal (and
    (at driver1 s1)
    (at truck1 s1)
    (at package1 s0)
    (at package2 s0)
    ))


)
```

-TRUCK

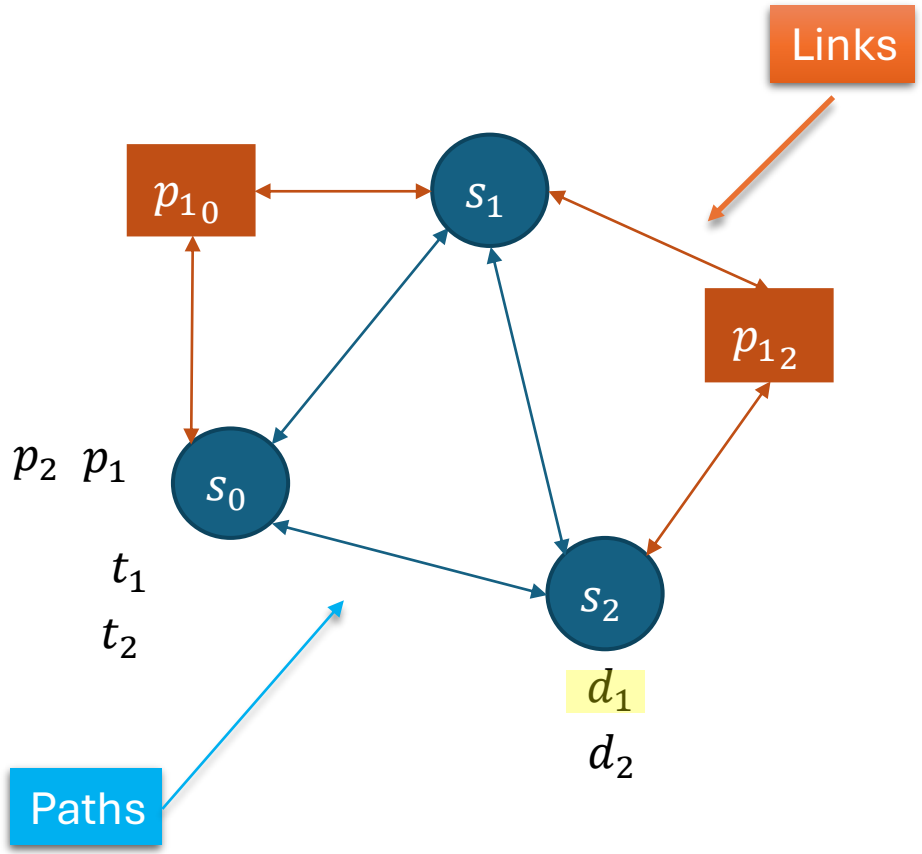
```
 ?truck) (LOCATION ?loc-from) (LOCATION ?loc-to) (DRIVER ?driver)
?loc-from)
-iver ?truck) (link ?loc-from ?loc-to))

at ?truck ?loc-from)) (at ?truck ?loc-to)))
```

LK
rs

om
)
tion
IVER ?driver) (LOCATION ?loc-from) (LOCATION ?loc-to)
iver ?loc-from) (path ?loc-from ?loc-to))

t (at ?driver ?loc-from)) (at ?driver ?loc-to)))



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

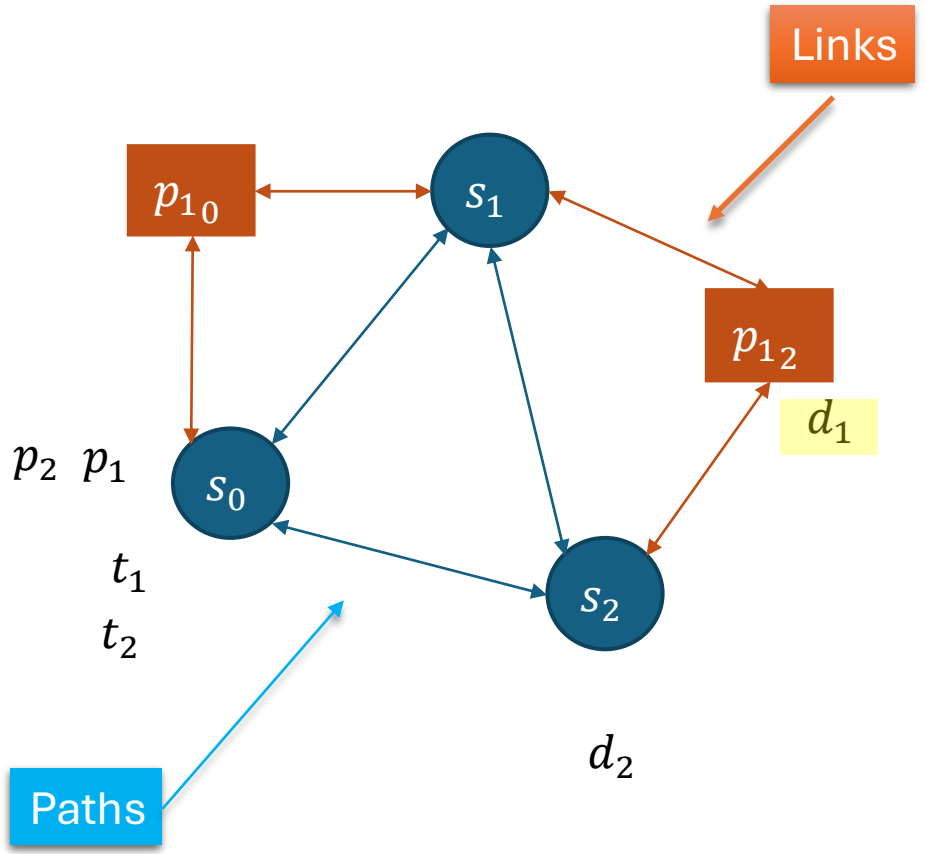
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action walk
  :parameters (driver1 s2 p1-2)
  :precondition
    (and
      (driver driver1)
      (location s2)
      (location p1-2)
      (at driver1 s2)
      (path s2 p1-2)
    )
  :effect
    (and
      (not
        (at driver1 s2)
      )
      (at driver1 p1-2)
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

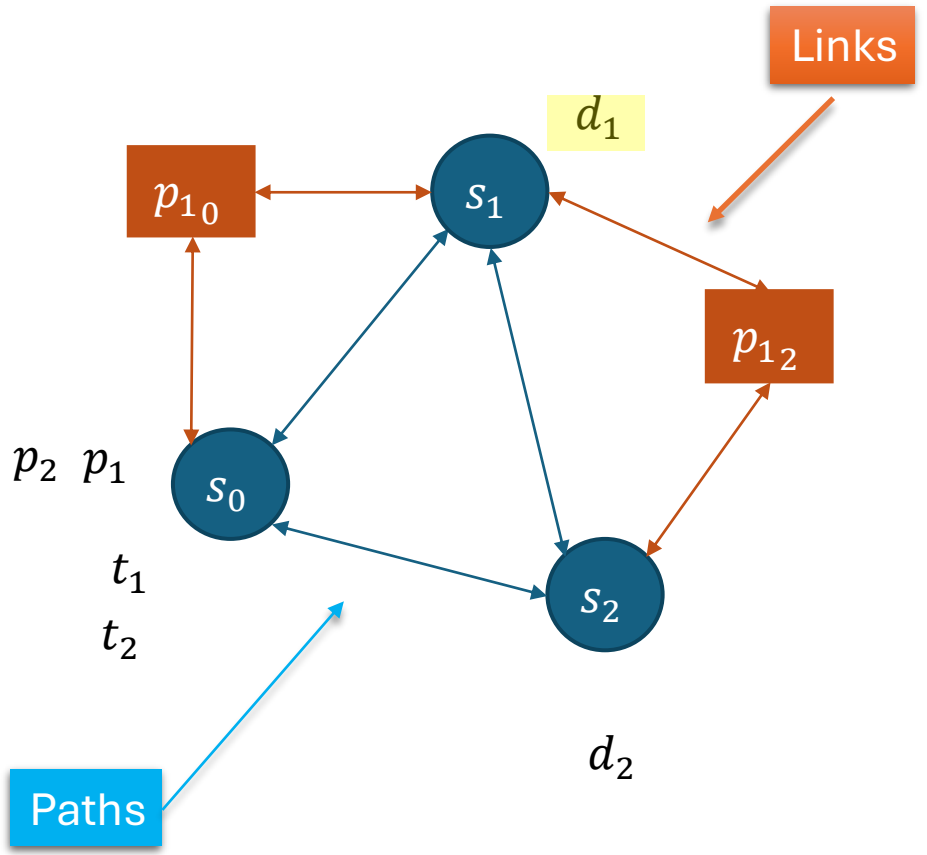
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action walk
  :parameters (driver1 p1-2 s1)
  :precondition
    (and
      (driver driver1)
      (location p1-2)
      (location s1)
      (at driver1 p1-2)
      (path p1-2 s1)
    )
  :effect
    (and
      (not
        (at driver1 p1-2)
      )
      (at driver1 s1)
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

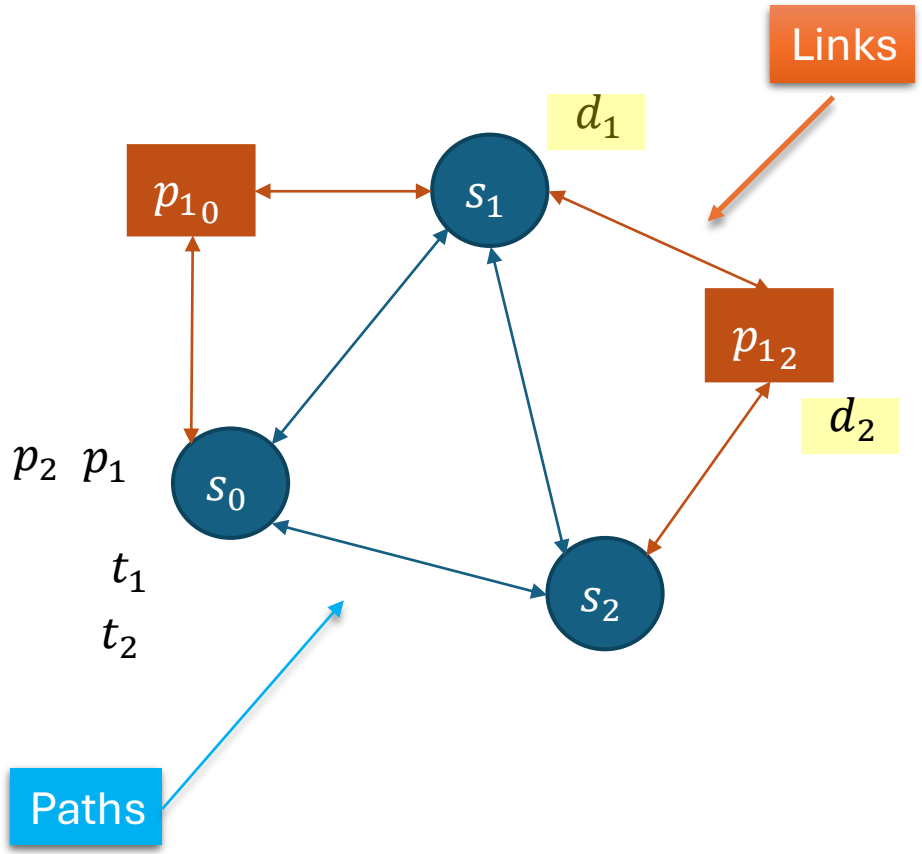
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action walk
  :parameters (driver2 s2 p1-2)
  :precondition
    (and
      (driver driver2)
      (location s2)
      (location p1-2)
      (at driver2 s2)
      (path s2 p1-2)
    )
  :effect
    (and
      (not
        (at driver2 s2)
      )
      (at driver2 p1-2)
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

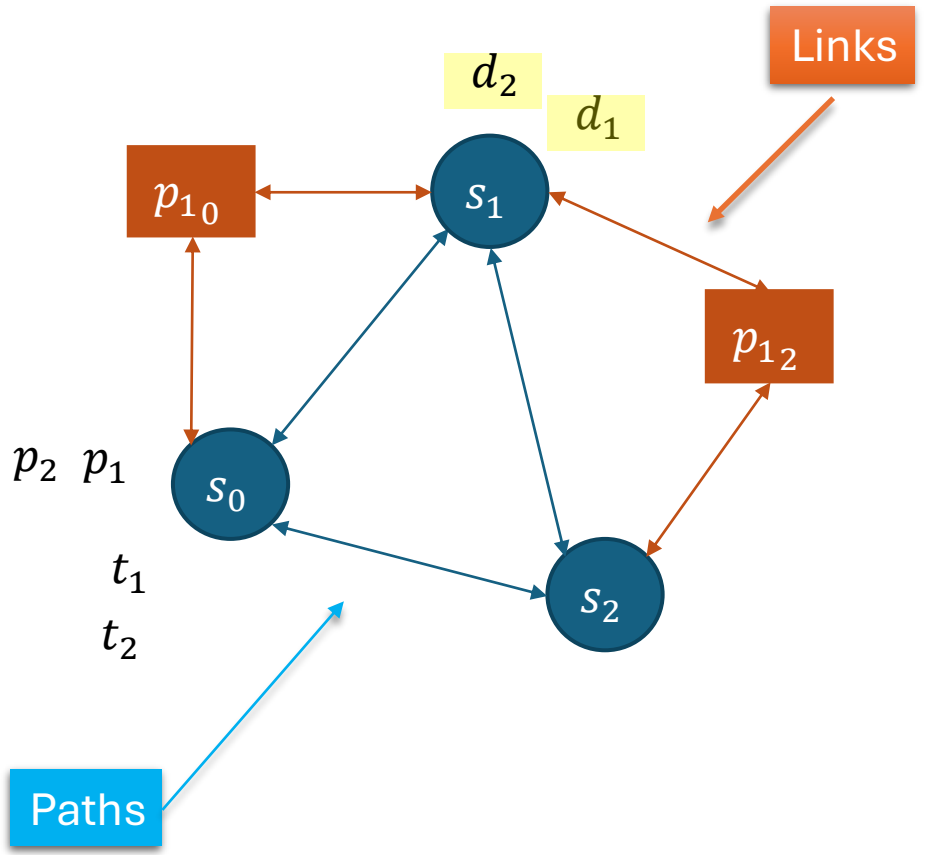
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action walk
  :parameters (driver2 p1-2 s1)
  :precondition
    (and
      (driver driver2)
      (location p1-2)
      (location s1)
      (at driver2 p1-2)
      (path p1-2 s1)
    )
  :effect
    (and
      (not
        (at driver2 p1-2)
      )
      (at driver2 s1)
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

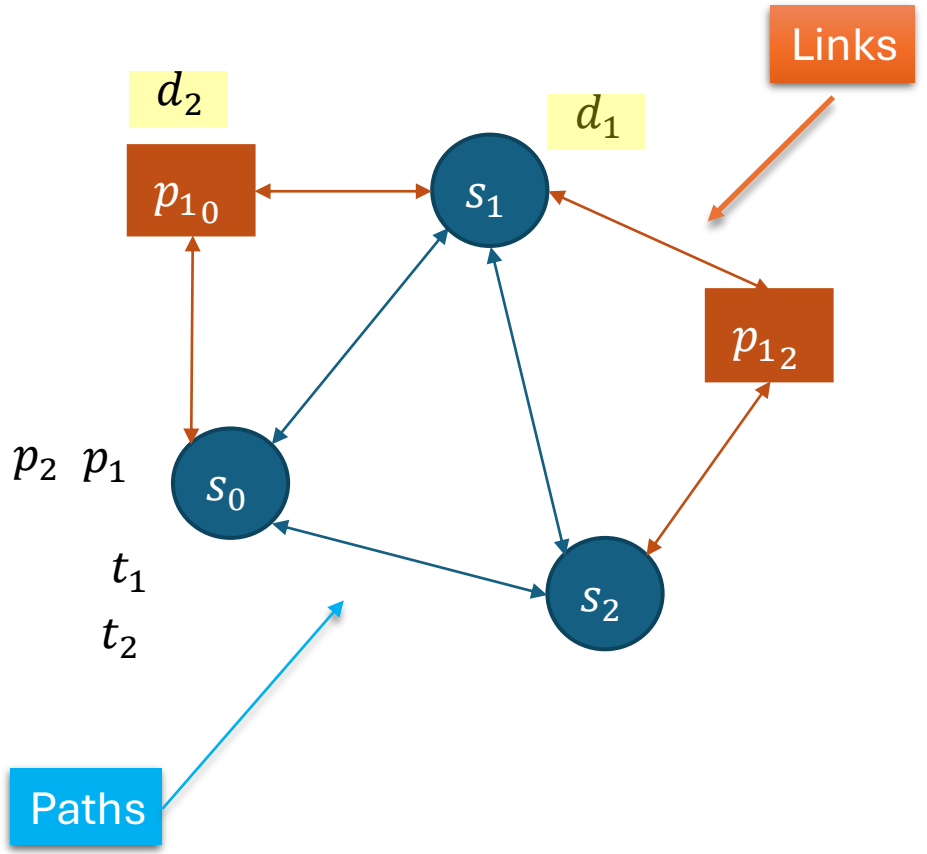
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action walk
  :parameters (driver2 s1 p1-0)
  :precondition
    (and
      (driver driver2)
      (location s1)
      (location p1-0)
      (at driver2 s1)
      (path s1 p1-0)
    )
  :effect
    (and
      (not
        (at driver2 s1)
      )
      (at driver2 p1-0)
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

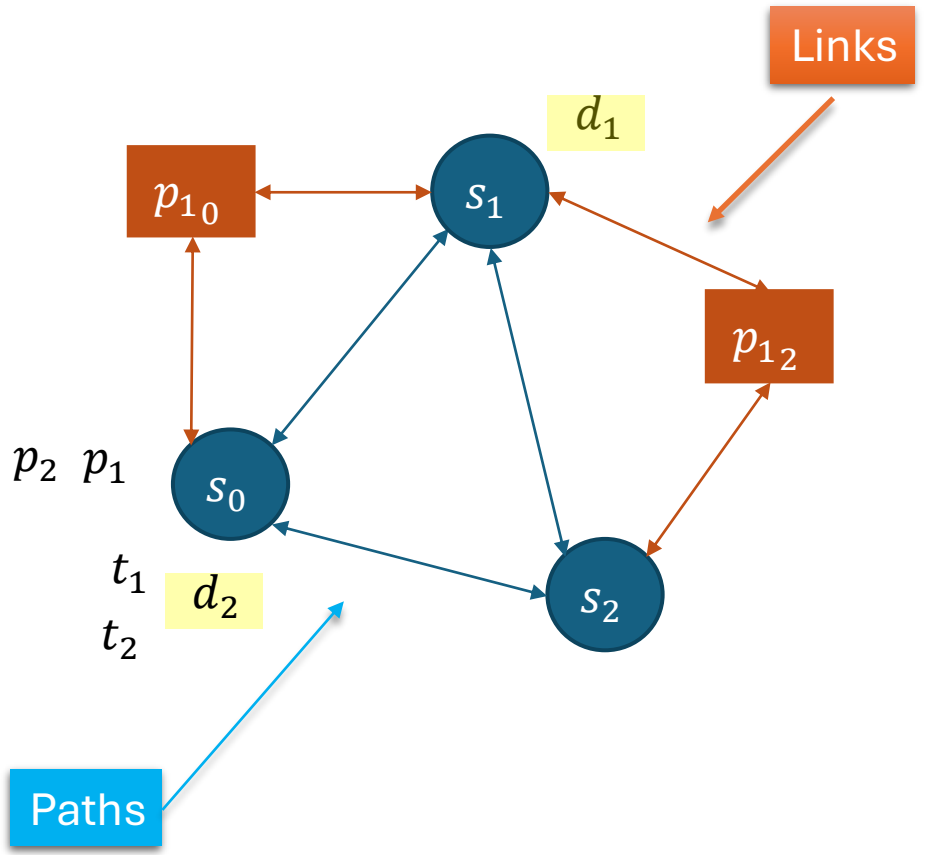
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action walk
  :parameters (driver2 p1-0 s0)
  :precondition
    (and
      (driver driver2)
      (location p1-0)
      (location s0)
      (at driver2 p1-0)
      (path p1-0 s0)
    )
  :effect
    (and
      (not
        (at driver2 p1-0)
      )
      (at driver2 s0)
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

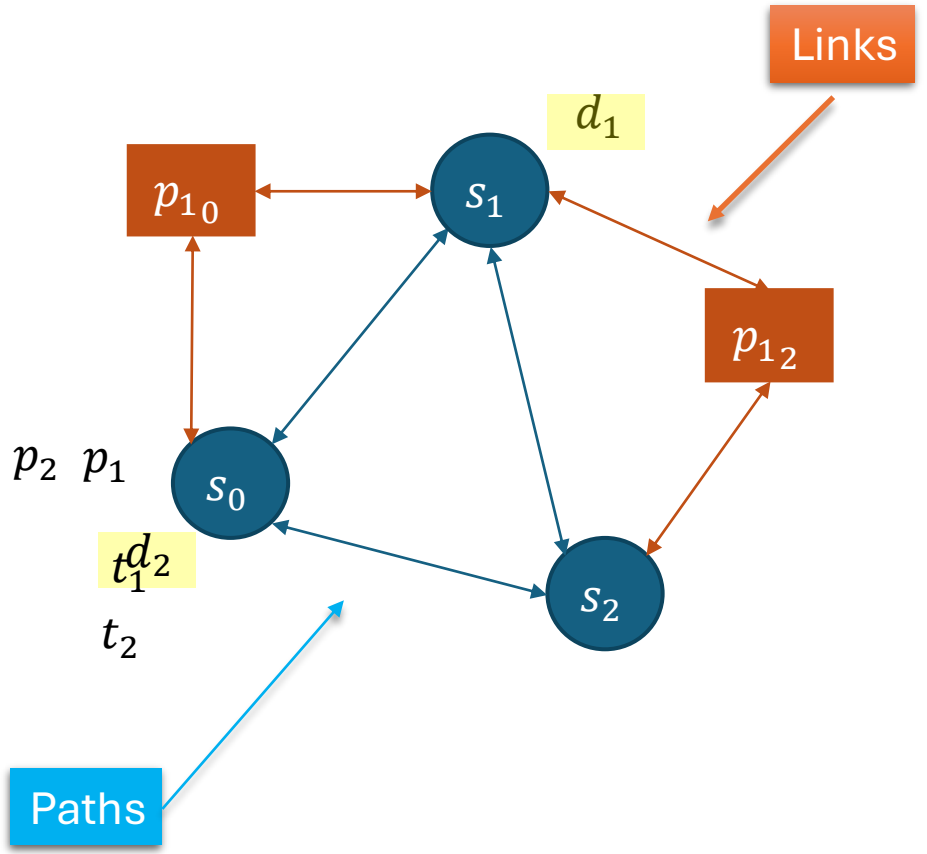
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action board-truck
  :parameters (driver2 truck1 s0)
  :precondition
    (and
      (driver driver2)
      (truck truck1)
      (location s0)
      (at truck1 s0)
      (at driver2 s0)
      (empty truck1)
    )
  :effect
    (and
      (not
        (at driver2 s0)
      )
      (driving driver2 truck1)
      (not
        (empty truck1)
      )
    )
)
```



(walk driver1 s2 p1-2)

(walk driver1 p1-2 s1)

(walk driver2 s2 p1-2)

(walk driver2 p1-2 s1)

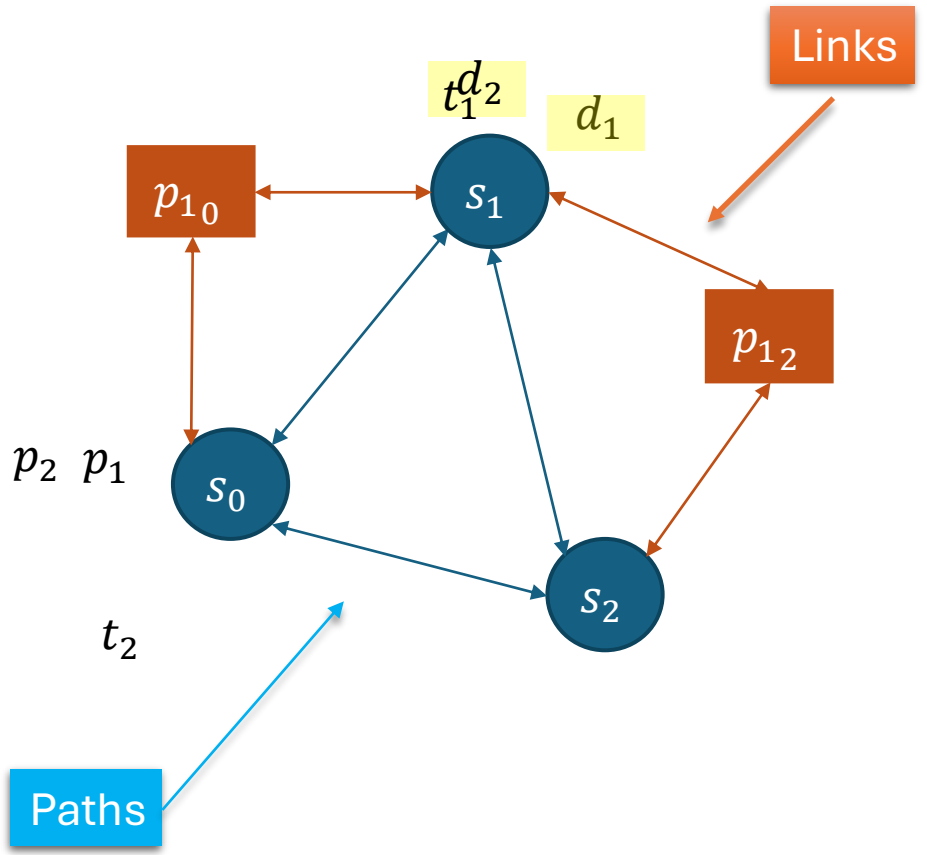
(walk driver2 s1 p1-0)

(walk driver2 p1-0 s0)

(board-truck driver2 truck1 s0)

(drive-truck truck1 s0 s1 driver2)

```
(:action drive-truck
  :parameters (truck1 s0 s1 driver2)
  :precondition
    (and
      (truck truck1)
      (location s0)
      (location s1)
      (driver driver2)
      (at truck1 s0)
      (driving driver2 truck1)
      (link s0 s1)
    )
  :effect
    (and
      (not
        (at truck1 s0)
      )
      (at truck1 s1)
    )
)
```



Is the plan optimal?

```
    (:init
    (at driver1 s2)
    (DRIVER driver1)
    (at driver2 s2)
    (DRIVER driver2)
    (at truck1 s0)
    (empty truck1)
    (TRUCK truck1)
    (at truck2 s0)
    (empty truck2)
    (TRUCK truck2)
    (at package1 s2)
    (OBJ package1)
    (at package2 s2)
    (OBJ package2)
    (LOCATION s0)
    (LOCATION s1)
    (LOCATION s2)
    (LOCATION p1-0)
    (LOCATION p1-2)
    (path s1 p1-0)
    (path p1-0 s1)
    (path s0 p1-0)
    (path p1-0 s0)
    (path s1 p1-2)
    (path p1-2 s1)
    (path s2 p1-2)
    (path p1-2 s2)
    (link s0 s1)
    (link s1 s0)
    (link s0 s2)
    (link s2 s0)
    (link s2 s1)
    (link s1 s2)
)
    (:goal (and
    (at driver1 s1)
    (at truck1 s1)
    (at package1 s1)
    (at package2 s1)
    ))
```

Minor Changes to Problem

1. Change the initial state to have both packages at s_2.
2. Change goal condition: *all* packages should be at s_1.

You can load this problem using this link (read-only):
https://editor.planning.domains/#read_session=hsiQUIpY2N

Try to trace the plan to verify if it is correct?

