

## Problem Set IV: Modeling in STRIPS and PDDL

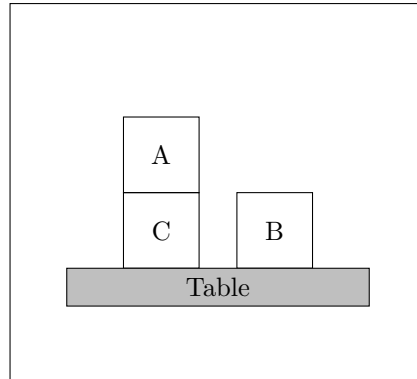


Figure 1: A blocks-world problem.

In blocks-world, the agent's aim is to stack the blocks in one tower with A on B and B on C. The agent can hold up to one block at a time and can put blocks down on the table, or another block.

1. Model Blocks-World as a STRIPS problem  $P = \langle F, O, I, G \rangle$ : define the set of facts  $F$ , the set of operators  $O$ , the goal facts  $G$  and the initial facts  $I$ . You must also define the *pre*, *add*, and *del* functions.
2. Implement your STRIPS model in PDDL. Use <http://editor.planning.domains> to test your model. Remember that a PDDL implementation is split between two files: a domain file (also known as an “operator” file) and a problem file (also known as a “fact” file).

For inspiration, the Australian TSP example from lectures is implemented in PDDL in the figures below. It is also accessible in [editor.planning.domains](http://editor.planning.domains) so you can try the solver on the cloud, and edit the TSP: [editor.planning.domains](http://editor.planning.domains) link with Domain and Problem file.

Or, you can directly fill in this template: [Block Worlds \(new\)](#)

See <http://www.hakank.org/pddl/> for more examples.

3. Blockworld can be modeled with only 2 actions instead of 4. The robot can pick up a block and put it down on another block (or the table) in a single action. You've got actions `Move(Block, FromTable, ToBlock)` and `Move(Block, FromBlock, ToTable)` You now no longer need to keep track of what the robot is holding or if the hand is empty.

Implement a STRIPS model of this “2-operation” blocks-world in PDDL. Use <http://editor.planning.domains> to write your model. The integrated solver is very limited, so you will want to download docker and install the planners in your computer by typing the command: `docker pull lapkt/lapkt-public`

See <https://hub.docker.com/r/lapkt/lapkt-public/> for more instructions on how to run the available planners

```

(define (domain tsp)
  (:requirements :typing)
  (:types node)

;; Define the facts in the problem
;; "?" denotes a variable, "-" a type

(:predicates (move ?from ?to - node)
  (at ?pos - node)
  (connected ?start ?end - node)
  (visited ?end - node))

;; Define the action(s)

(:action move
  :parameters (?start ?end - node)
  :precondition (and (at ?start)
    (connected ?start ?end))
  :effect (and (at ?end)
    (visited ?end)
    (not (at ?start))))

```

Figure 2: tsp-domain.pddl

```

(define (problem tsp-01)
  (:domain tsp)
  (:objects Sydney Adelaide Brisbane Perth Darwin - node)

;; Define the initial situation
(:init (connected Sydney Brisbane)
  (connected Brisbane Sydney)
  (connected Adelaide Sydney)
  (connected Sydney Adelaide)
  (connected Adelaide Perth)
  (connected Perth Adelaide)
  (connected Adelaide Darwin)
  (connected Darwin Adelaide)
  (at Sydney))

(:goal
  (and (at Sydney)
    (visited Sydney)
    (visited Adelaide)
    (visited Brisbane)
    (visited Perth)
    (visited Darwin))))

```

Figure 3: tsp-problem.pddl