Planning I

PROF LIM KWAN HUI

50.021 Artificial Intelligence

Outline & Objectives

- Able to describe planning problems in a simple formalism - that is finding a sequence of actions.
- Able to formulate a planning problem as a STRIPS instance
- Understand the PDDL planning language
- Develop relaxed versions of planning problems and understand the various heuristics that can be used

Planning and Search

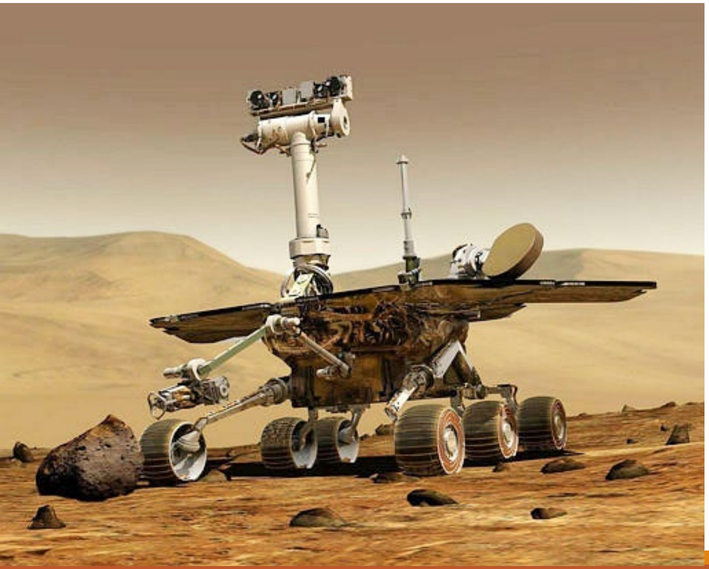
- Planning is the process of computing several steps of a problem-solving procedure before executing any of them
- This problem can be solved by search
- The main difference between search and planning is the representation of states
  - In search, states are represented as a single entity (which may be quite a complex object, but its internal structure is not used by the search algorithm)
  - In planning, states have structured representations (collections of properties) which are used by the planning algorithm

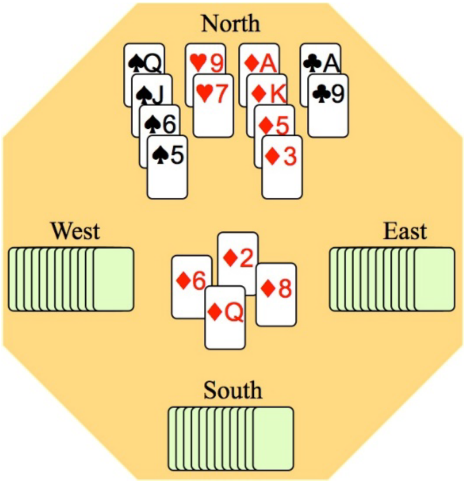
Types of Planners

1. Domain-specific
   - Made or tuned for a specific planning domain
   - Won’t work well (if at all) in other planning domains
2. Domain-independent
   - In principle, works in any planning domain
   - In practice, need restrictions on what kind of planning domain
3. Configurable
   - Domain-independent planning engine
   - Input includes info about how to solve problems in some domain

Domain-Specific Planners

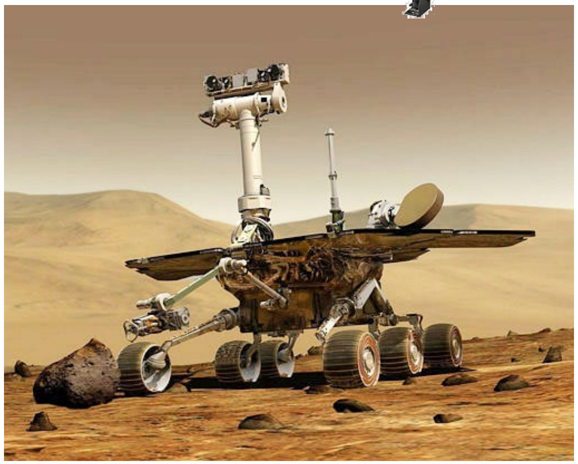
- Most successful real-world planning systems work this way
  - Mars exploration, sheet-metal bending, playing bridge, etc.
- Often use problem-specific techniques that are difficult to generalize to other planning domains





Domain-Independent Planners

- In principle, works in any planning domain
- No domain-specific knowledge except the description of the system
- In practice,
  - Not feasible to make domain-independent planners work well in all possible planning domains



- Make simplifying assumptions to restrict the set of domains
  - *Classical planning*
  - Historical focus of most research on automated planning

Configurable Planners

- In a fixed planning domain, a domain-independent planner usually won’t work as well as a domain-specific planner made for that domain
  - A domain-specific planner may be able to go directly toward a solution in situations where a domain-independent planner would explore many alternative paths
- But we don’t want to write a whole new planner for every domain
- Configurable planners
  - Domain-independent planning engine
  - Input includes info about how to solve problems in the domain
- Generally this means one can write a planning engine with fewer restrictions than domain-independent planners

Classical Planning

- Classical planning requires certain assumptions
  - Offline generation of action sequences for an environment that is fully observable, deterministic, finite, static and discrete
- Reduces to the following problem:
  - Given a planning problem $\mathcal{P} = (A, s_0, S_g)$
  - Find a sequence of actions $(a_1, a_2, \dots a_n)$ that produces a sequence of state transitions $(s_1, s_2, \dots, s_n)$ such that s_n is in S_g.
- This is just path-searching in a graph
  - Nodes = states
  - Edges = actions

STRIPS

- STRIPS = Stanford Research Institute Problem Solver (1971)
  - Originally a planner software, today mostly used to name a formal language to describe planning problems.
  - (Logic-based) Language expressive enough to describe a wide variety of problems, but restrictive enough to allow efficient algorithms to operate over it

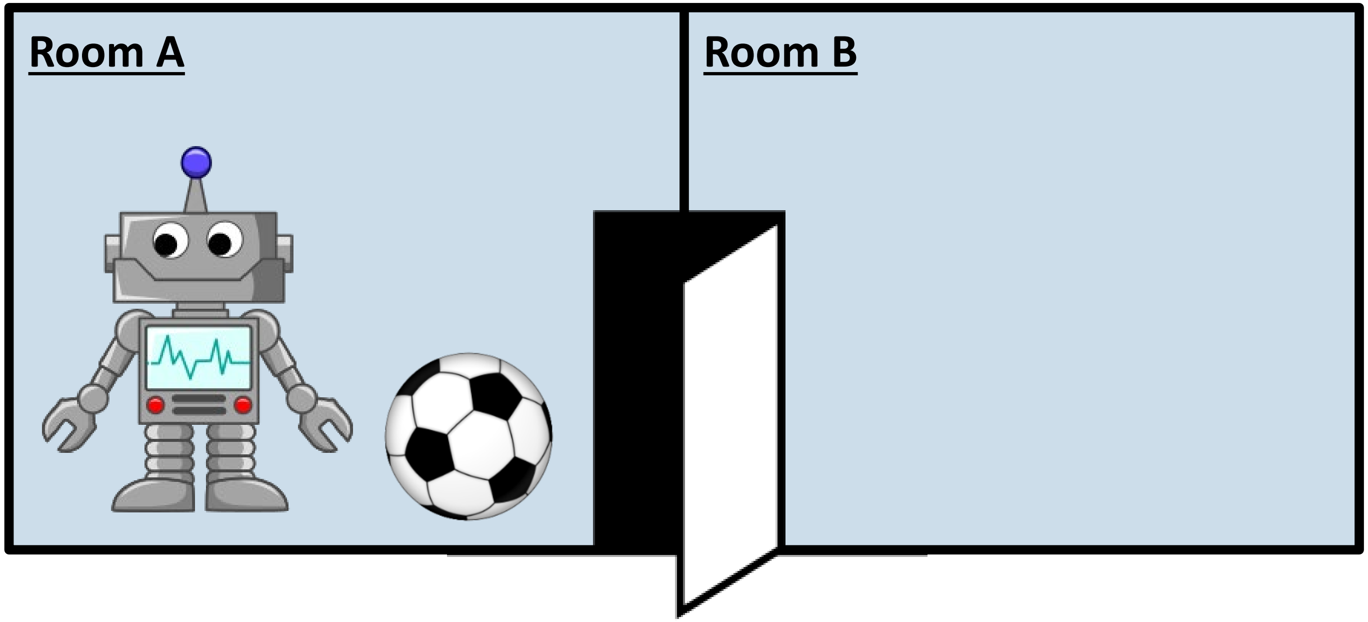
- Planning is about finding a sequence of actions to achieve a goal. how to describe such things?

STRIPS Instance

- An initial state;
- The specification of the goal states – situations which the planner is trying to reach
- A set of actions. For each action, the following are included:
  - preconditions: true and false facts that must hold so that an action can be performed
  - postcondition: facts that change when an action is performed.

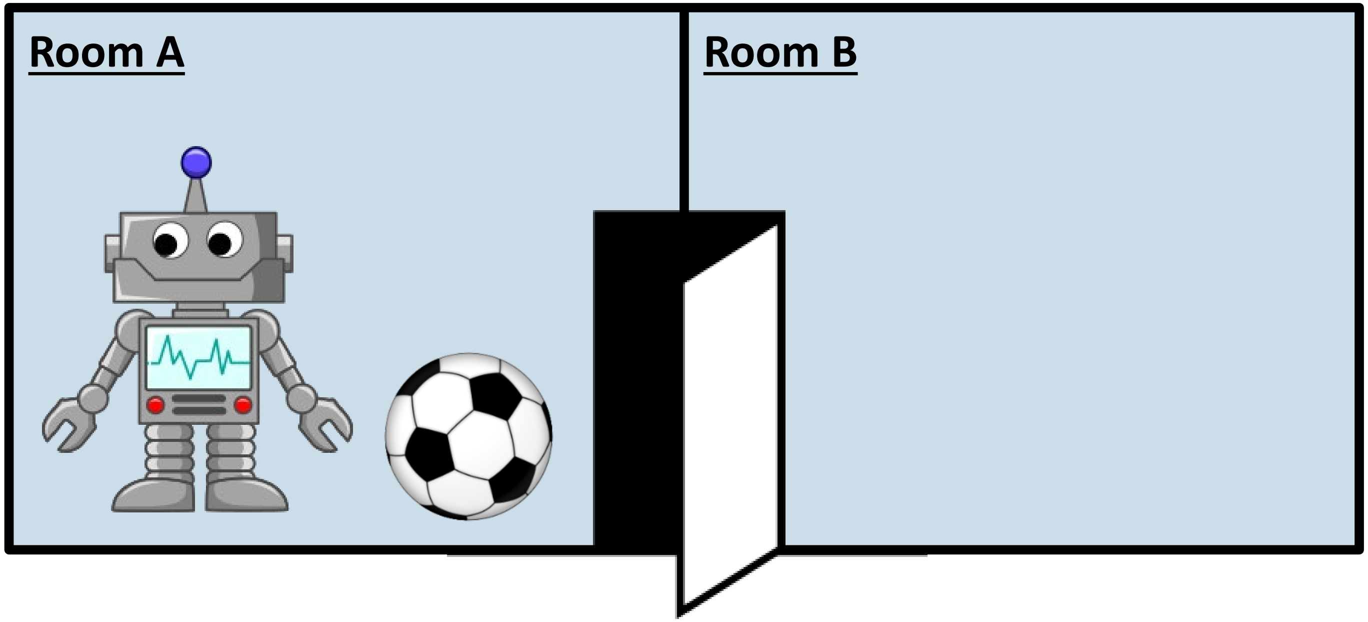
Example: Robot World

- A simple world with a robot, a ball and two rooms connected by a door
  - How would you model it?



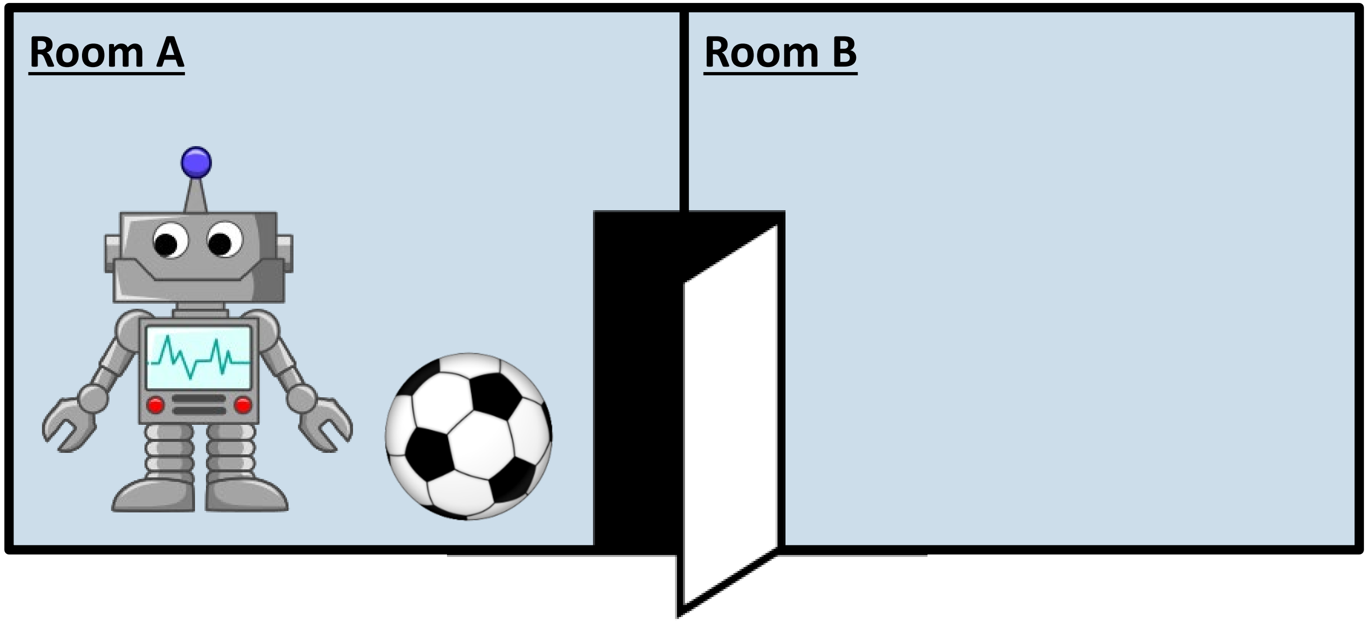
Example: Robot World

- Propositional variables or facts
  - True/false variables that model some aspect of the world
    - inB(ball): Is the ball in room B? False
    - inA(robot): Is the robot in room A? True



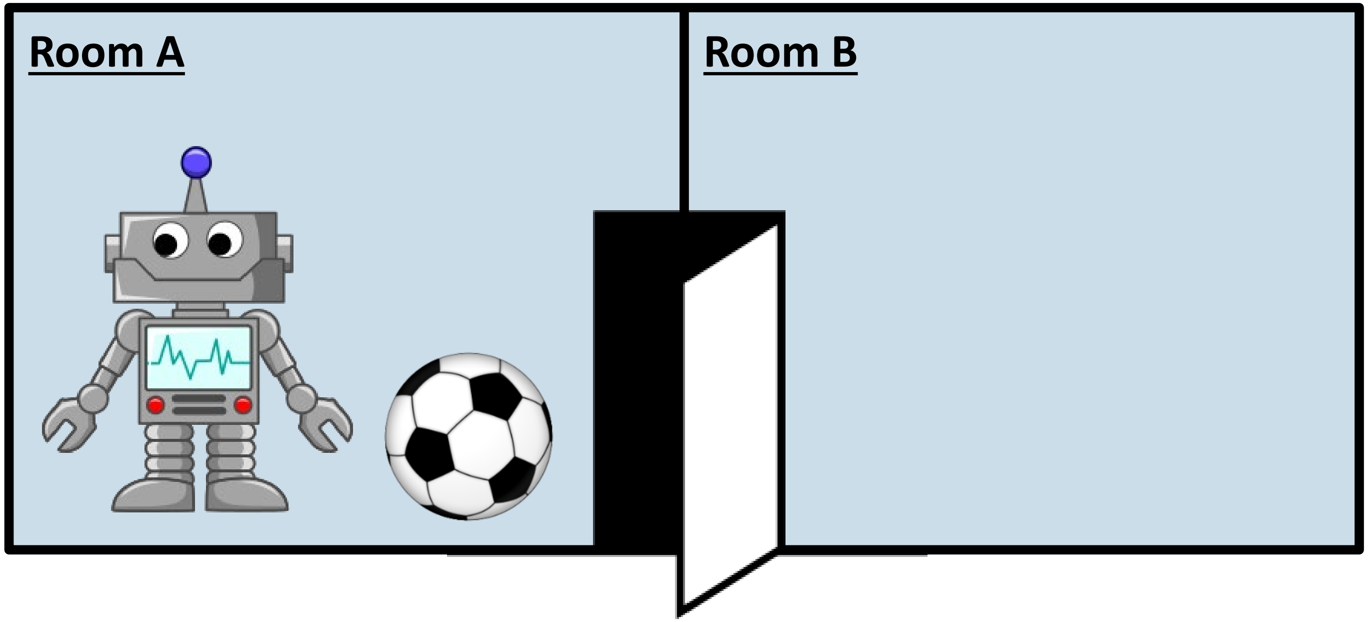
Example: Robot World

- There are various objects (robot, ball) in this world
  - How do we model actions on these objects, e.g., kick ball to other room?
  - What effect does it have on the variables representing these objects?



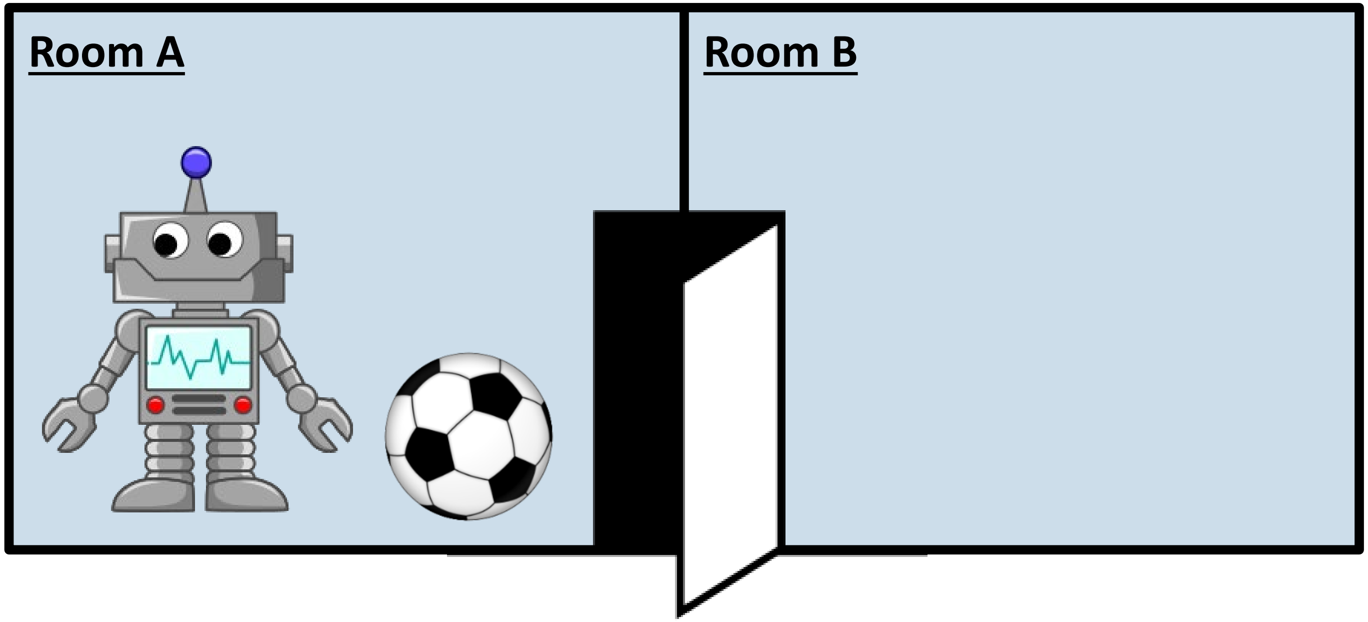
Example: Robot World

- There are various objects (robot, ball) in this world
  - Actions may change the value of these variables
  - Actions may have various preconditions of variable values



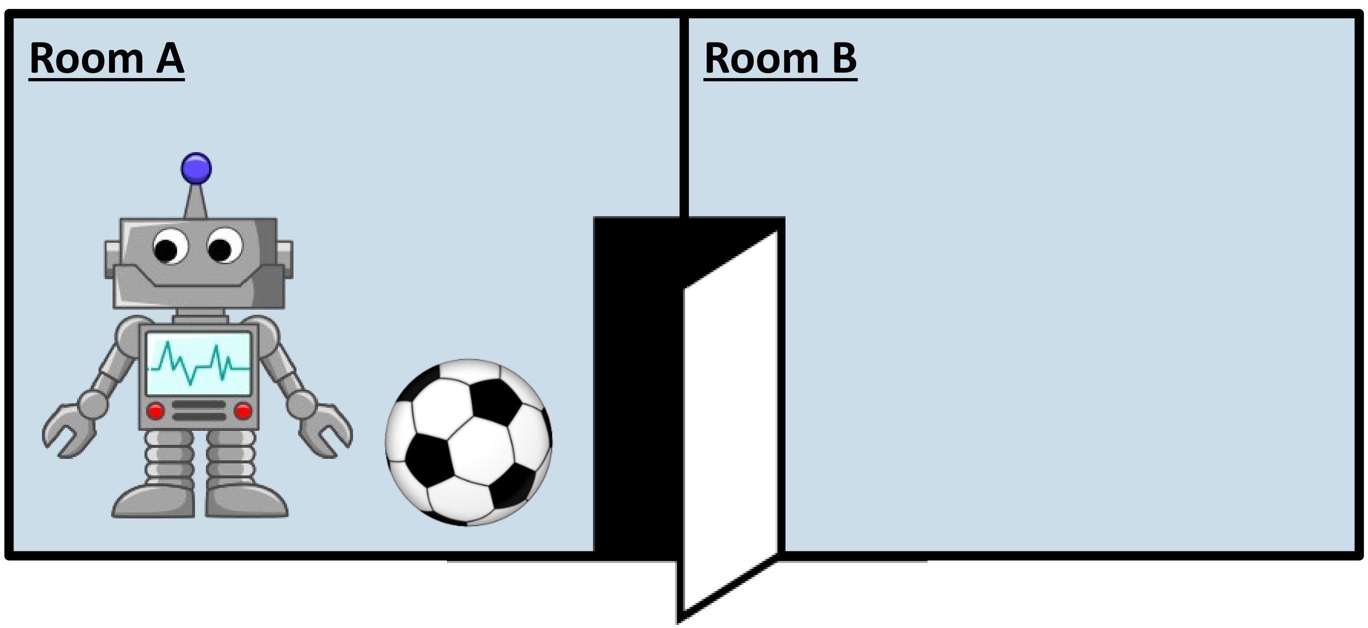
Example: Robot World

- For modelling the action of the robot kicking the ball to room B, kickball(a,b), what do we need check for first?



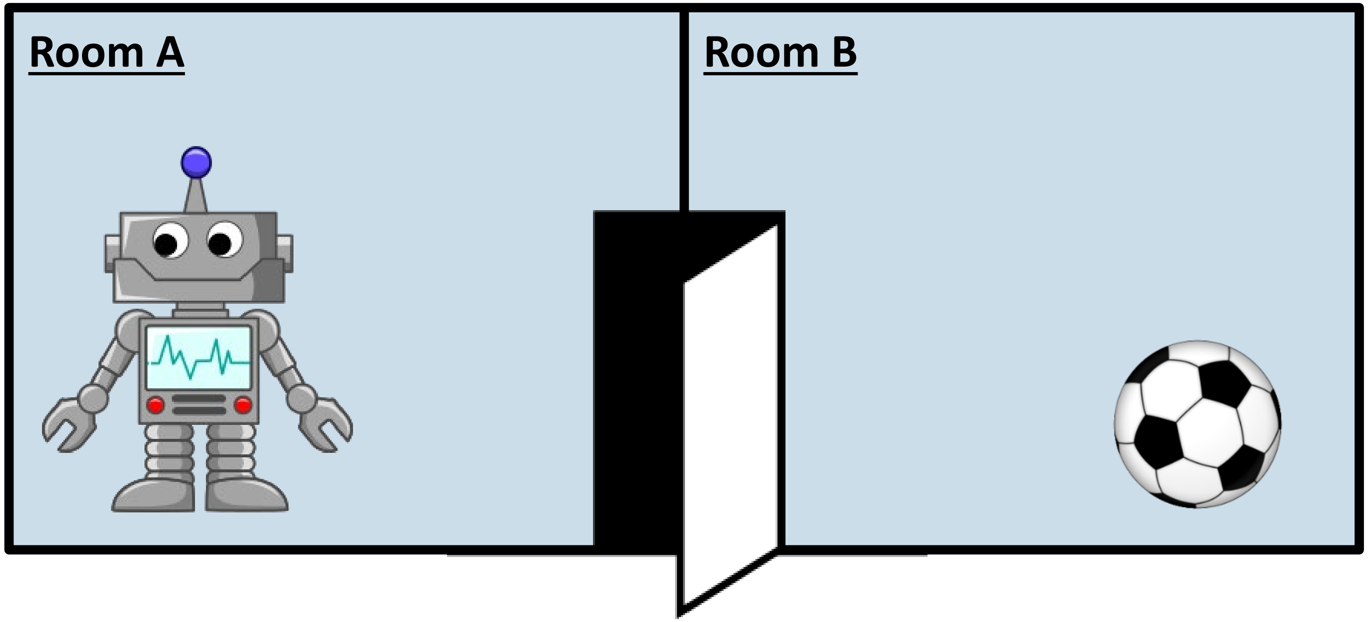
Example: Robot World

- For modelling the action of the robot kicking the ball to room B, kickball(a,b), what do we need check for first?
  - inA(robot), inA(ball), dooropen(A,B)



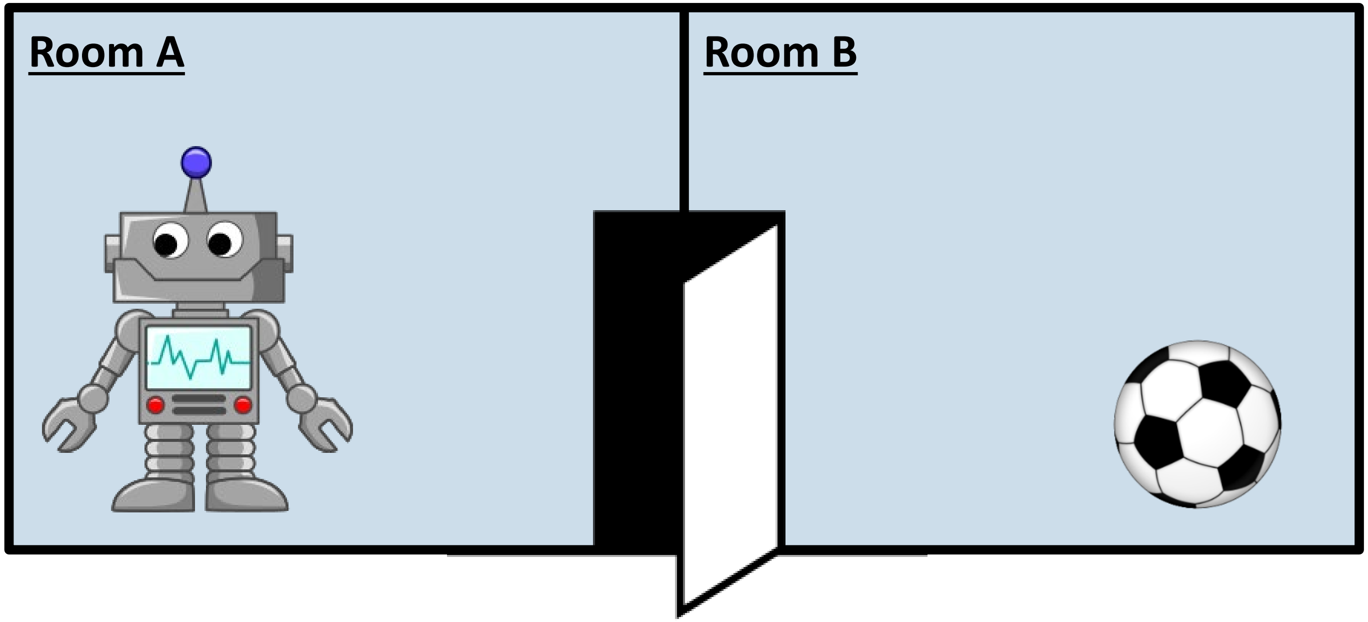
Example: Robot World

- For modelling the action of the robot kicking the ball to room B, kickball(a,b), what changes are there to the variables?



Example: Robot World

- For modelling the action of the robot kicking the ball to room B, kickball(a,b), what changes are there to the variables?
  - not inA(ball), inB(ball)
  - What about the robot? Still inA(robot)



STRIPS Instance

- An initial state;
- The specification of the goal states – situations which the planner is trying to reach
- A set of actions. For each action, the following are included:
  - preconditions: true and false facts that must hold so that an action can be performed
  - postcondition: facts that change when an action is performed.

STRIPS Instance

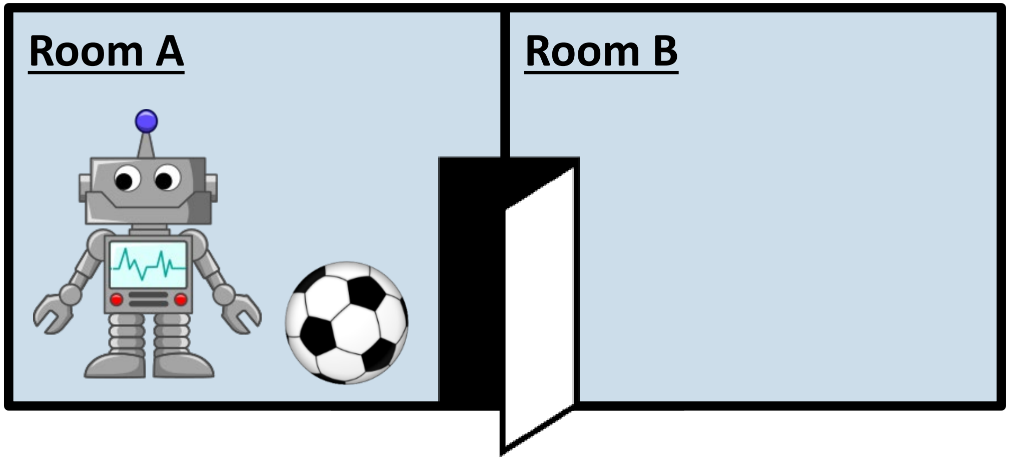
- STRIPS is formally defined as a 4-tuple ***(P,O, I, G)***
- ***P*** - a set of propositional variables – the facts that describe the state of the world
- ***O*** - a set of operators (i.e., actions). Each operator itself is a 3-tuple (pre_a, add_a, del_a) of
  - pre_a - facts that must be true before the action can be performed
  - add_a - facts that will change to true when/after the action can be performed
  - del_a - facts that will change to false when/after the action can be performed
  - requirement: $add_a \cap del_a = \emptyset$
- ***I*** - the initial state of the world, true/false assignments to variables from P
- ***G*** - the goal state of the world

STRIPS: Robot World

- STRIPS is formally defined as a 4-tuple ***(P,O, I, G)***
- ***P*** - a set of propositional variables. E.g., inA(x), inB(x), dooropen(x,y)
- ***O*** - a set of operators (i.e., actions). E.g., kickball(a,b)
  - pre_a - inA(robot), inA(ball), dooropen(A,B)
  - add_a - inB(ball)
  - del_a - inA(ball)
- ***I*** - the initial state of the world. E.g., inA(robot), inA(ball), dooropen(A,B)
- ***G*** - the goal state of the world. E.g., inB(ball)

State Representation

- World states are represented as sets of facts: conjunction of propositions (conditions)
  - E.g., Robot World: State 1 = { inA(robot) ∧ inA(ball) ∧ dooropen(A,B) }



- Closed World Assumption (CWA): Facts not listed in a state are assumed to be false. Under CWA the assumption the agent has full observability and only positive facts need to be stated

State Representation

- The world is represented through a set of features / objects (e.g., planes, people, cities) and each proposition states a fact that attributes “values” to features

Objects	State Propositions	Closed World Assumptions
Robot	inA(robot)	Not inB(robot)
Ball	inA(ball)	Not inB(ball)
Door	doorOpen(a,b)	

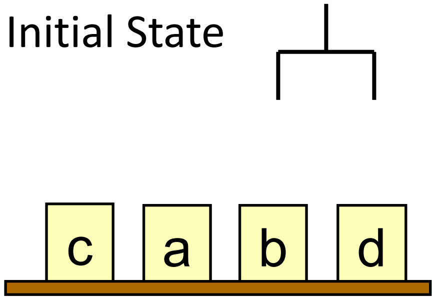
State Representation

- Goals states are also represented as sets of facts
  - E.g., the state { inB(ball) } can be defined as a goal state
- A goal state is any state that includes all the goal facts
  - The following are valid goals:
    - State 1 = { inA(robot) $\wedge$ inB(ball) }
    - State 2 = { inB(robot) $\wedge$ inB(ball) }
    - State 3 = { dooropen(a,b) $\wedge$ inB(robot) $\wedge$ inB(ball) }
  - The following are not:
    - State 1 = { inB(robot) $\wedge$ dooropen(a,b) }
    - State 2 = { inA(ball) $\wedge$ inB(robot) }

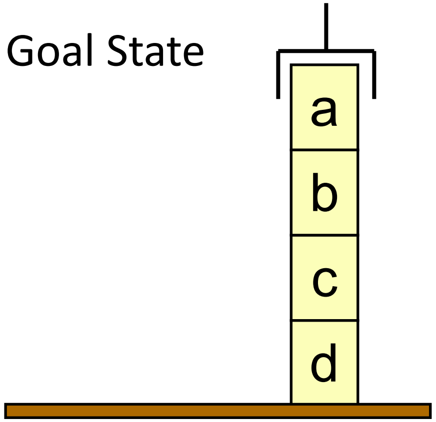
Exercise: Block World

- A simple world that comprises a set of blocks, a table, and a robot claw
- The aim is to stack the blocks on top of each other in alphabetical order
- Some constraints of this world:
  - The robot claw can only carry one block at a time
  - Only one block can be on another block
  - Any number of blocks can be on the table
- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let's start with P, I and G first.

Initial State

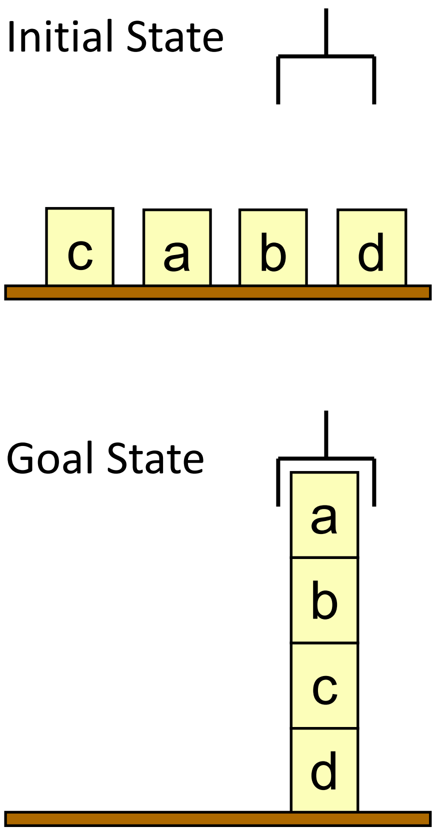


Goal State



Exercise: Block World

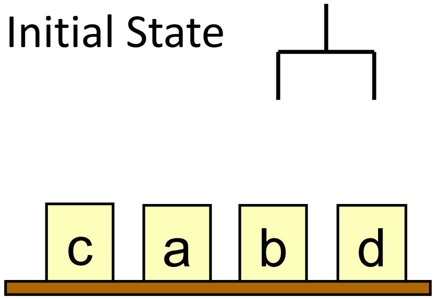
- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let’s start with P, I and G first.
- ***P*** - a set of propositional variables.
  - ?



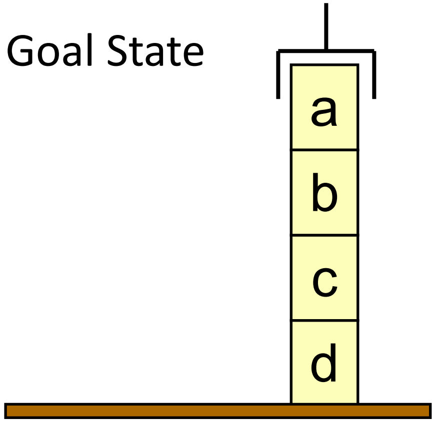
Exercise: Block World

- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let’s start with P, I and G first.
- ***P*** - a set of propositional variables.
  - ontable(block1): block1 is on the table
  - on(block1,block2): block1 is on block2
  - free(block1): top of block1 is free
  - clawempty: the robot claw is not holding any blocks

Initial State

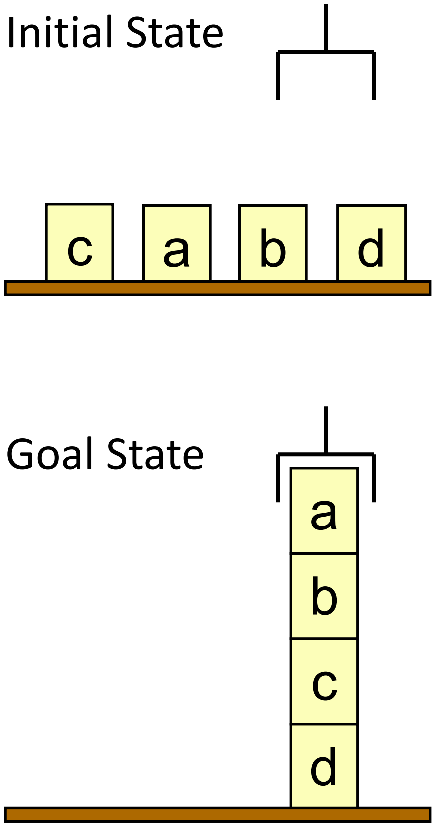


Goal State



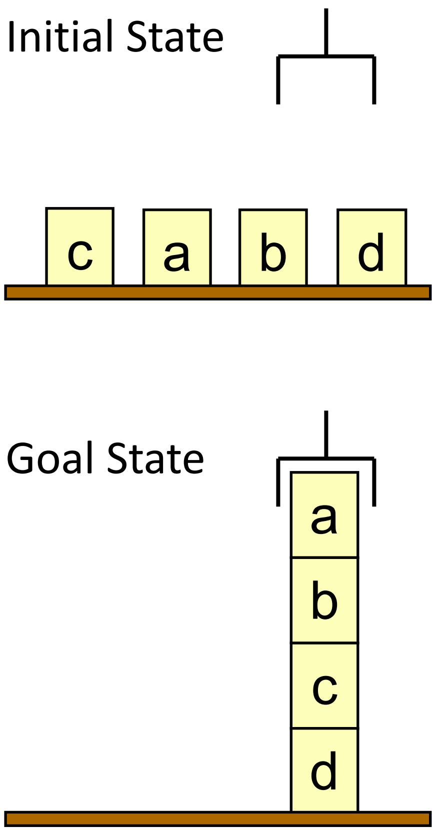
Exercise: Block World

- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let's start with P, I and G first.
- ***P*** - a set of propositional variables.
  - ontable(block1): block1 is on the table
  - on(block1,block2): block1 is on block2
  - free(block1): top of block1 is free
  - clawempty: the robot claw is not holding any blocks
- ***I*** - the initial state of the world.
  - ?



Exercise: Block World

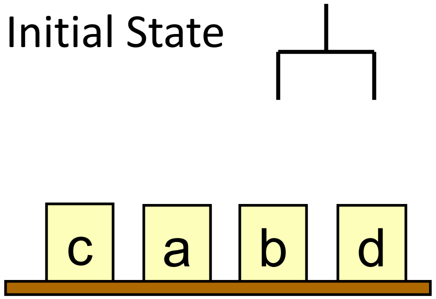
- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let's start with P, I and G first.
- ***P*** - a set of propositional variables.
  - ontable(block1): block1 is on the table
  - on(block1,block2): block1 is on block2
  - free(block1): top of block1 is free
  - clawempty: the robot claw is not holding any blocks
- ***I*** - the initial state of the world.
  - ontable(c), ontable(a), … , free(c), … , clawempty



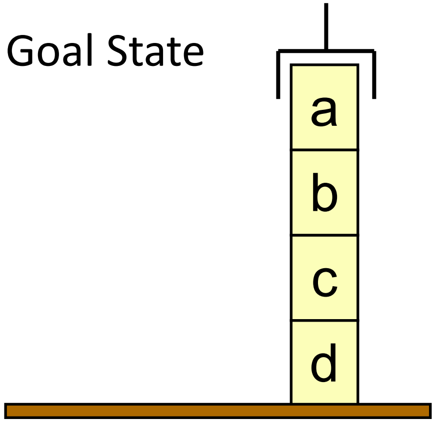
Exercise: Block World

- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let’s start with P, I and G first.
- ***P*** - a set of propositional variables.
  - ontable(block1): block1 is on the table
  - on(block1,block2): block1 is on block2
  - free(block1): top of block1 is free
  - clawempty: the robot claw is not holding any blocks
- ***I*** - the initial state of the world.
  - ontable(c), ontable(a), ... , free(c), ... , clawempty
- ***G*** - the goal state of the world.
  - ?

Initial State



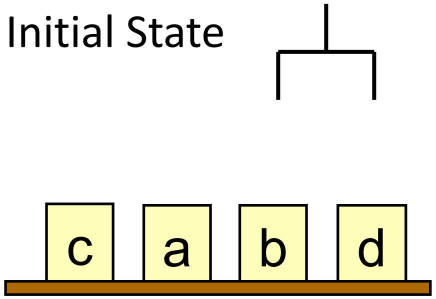
Goal State



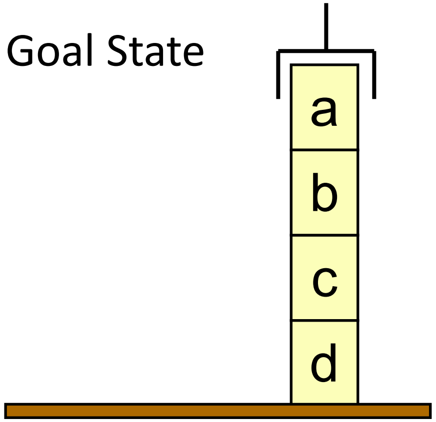
Exercise: Block World

- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Let's start with P, I and G first.
- ***P*** - a set of propositional variables.
  - ontable(block1): block1 is on the table
  - on(block1,block2): block1 is on block2
  - free(block1): top of block1 is free
  - clawempty: the robot claw is not holding any blocks
- ***I*** - the initial state of the world.
  - ontable(c), ontable(a), ... , free(c), ... , clawempty
- ***G*** - the goal state of the world.
  - on(a,b), on(b,c), on(c,d), ontable(d)

Initial State

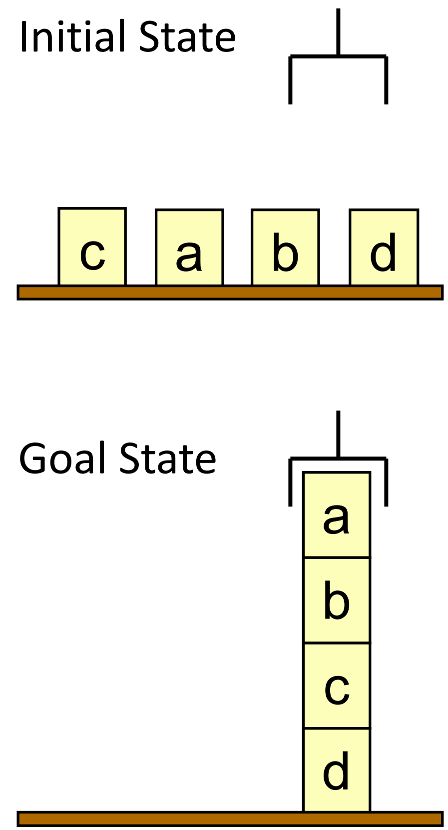


Goal State



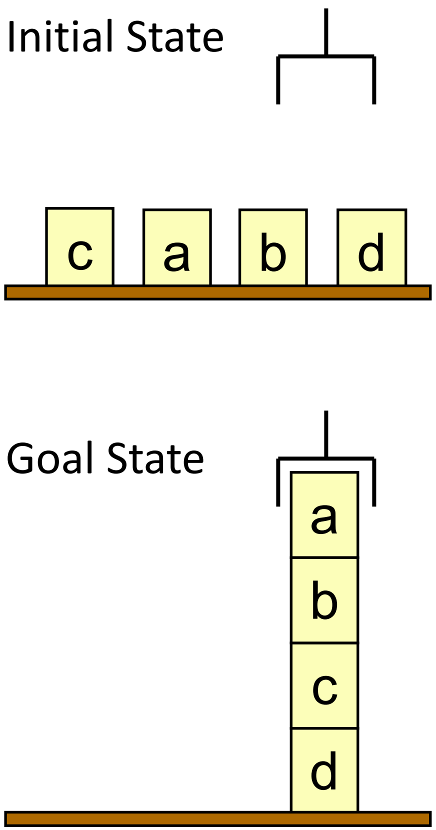
Exercise: Block World

- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Next, let's move on to O, specifically grabandstack(), which grabs a block and stack it in the last column.
- ***O*** - a set of operators (i.e., actions). E.g., for the grabandstack(block1,block2) operator:
  - pre_a -
  - add_a -
  - del_a -



Exercise: Block World

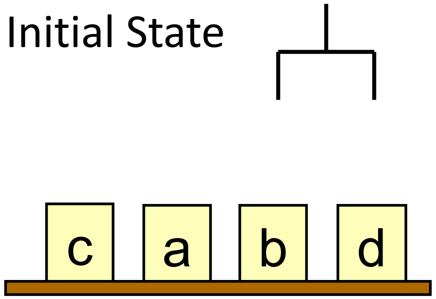
- Task: How would you model this world in terms of the STRIPS 4-tuple *(P, O, I, G)*?
  - Next, let's move on to O, specifically grabandstack(), which grabs a block and stack it in the last column.
- ***O*** - a set of operators (i.e., actions). E.g., for the grabandstack(block1,block2) operator:
  - pre_a – free(b1), free(b2), clawempty, ontable(b1)
  - add_a – on(b1,b2)
  - del_a – free(b2), ontable(b1)



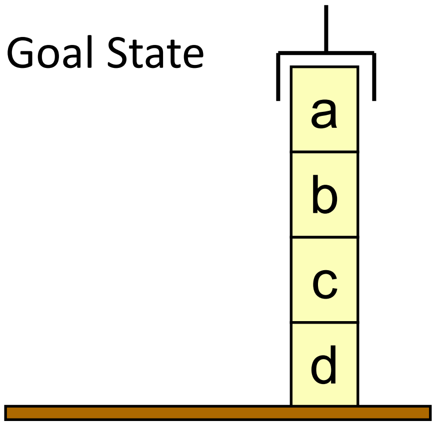
Exercise: Block World

- Task: Instead of grabandstack(), we want separate grab(block) and stack(block1, block2) operators.
  - What new propositional variables or facts do we need?
  - How do you define the operators?
- ***P*** - a set of propositional variables.
  - Existing: ontable(blk1), on(b1,b2), free(b1), clawempty
  - New:
- ***O*** - grab(block):
  - pre_a –
  - add_a –
  - del_a –
- ***O*** - stack(block1, block2):
  - pre_a –
  - add_a –
  - del_a –

Initial State



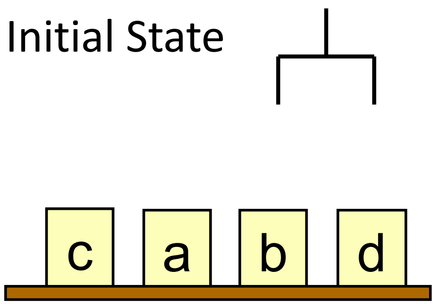
Goal State



Exercise: Block World

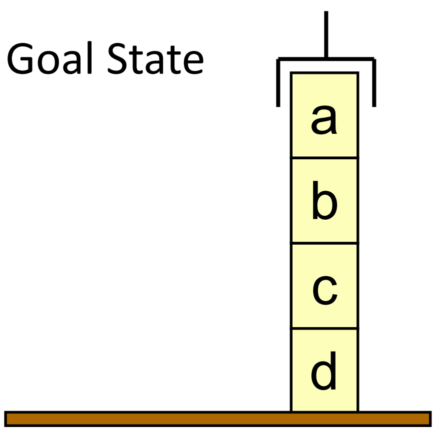
- Task: Instead of grabandstack(), we want separate grab(block) and stack(block1, block2) operators.
  - What new propositional variables or facts do we need?
  - How do you define the operators?

Initial State



- ***P*** - a set of propositional variables.
  - Existing: ontable(blk1), on(b1,b2), free(b1), clawempty
  - New: holding(block1)

Goal State



- ***O*** - grab(block):
  - pre_a – ontable(b), free(b), clawempty
  - add_a – holding(b)
  - del_a – ontable(b), free(b), clawempty
- ***O*** - stack(block1, block2):
  - pre_a – holding(b1), free(b2)
  - add_a – clawempty, free(b1) on(b1,b2)
  - del_a – holding(b1), free(b2)

Next

- Look at the PDDL planning language
- Simple coding exercise on using PDDL