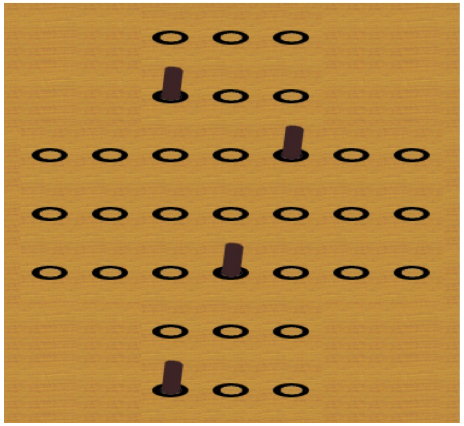
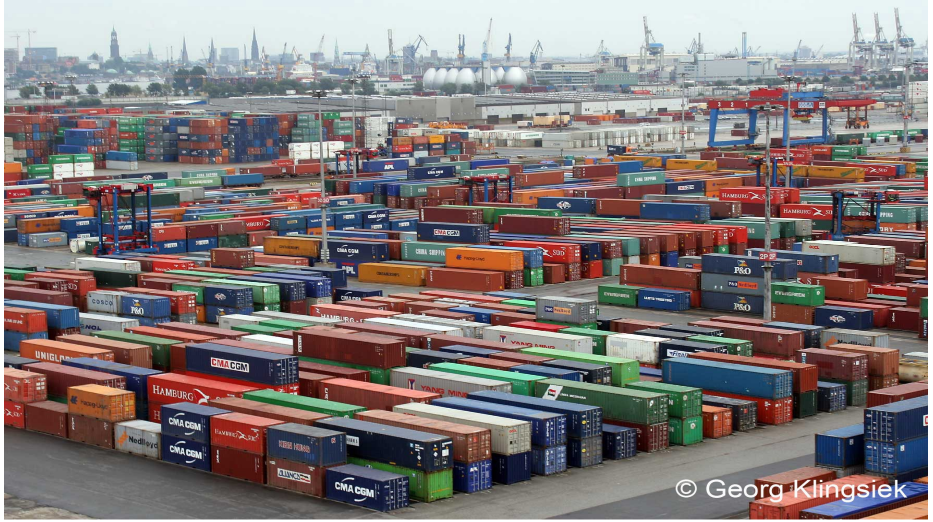
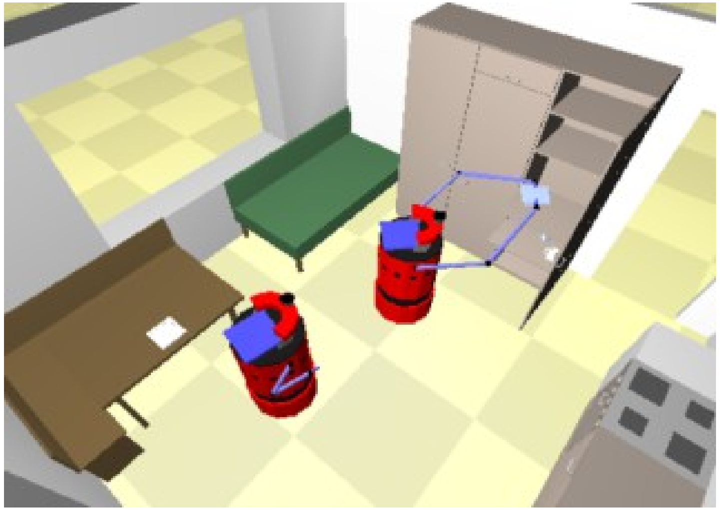
Outline

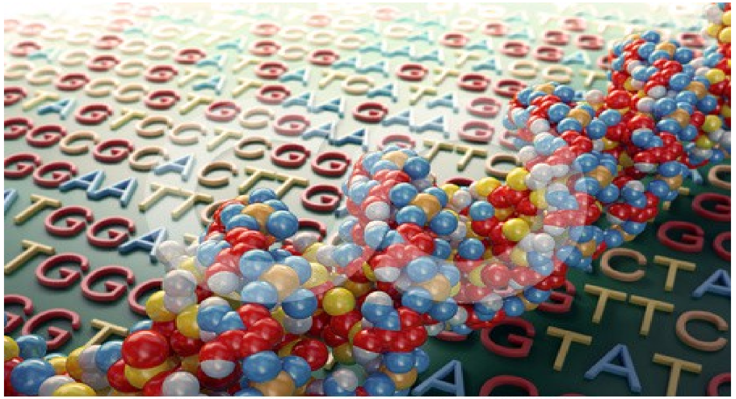
- The Planning Problem
- Planning with State-Based Search
- Partial-Order Planning
- Planning with Propositional Logic

Applications of Planning









© Mopic * www.ClipartOf.com/433340

Example: Single-Player "Game"

Legality

```
legal(you,rightShoe) <= true(rightSockOn)
legal(you,rightSock)
legal(you,leftShoe)  <= true(leftSockOn)
legal(you,leftSock)
```

Update

```
next(rightShoeOn) <= does(you,rightShoe)
next(rightSockOn) <= does(you,rightSock)
next(leftShoeOn)  <= does(you,leftShoe)
next(leftSockOn)  <= does(you,leftSock)
```

Termination and Goal

```
terminal      <= true(rightShoeOn)∧true(leftShoeOn)
goal(you,100) <= true(rightShoeOn)∧true(leftShoeOn)
```

A Simpler Description Language for Planning Problems

- Initial state: conjunction of variable-free atoms
- Actions: <Name, Precondition, Effect>
  - Name: Action name + parameter list
  - Precond: Conjunction of literals
  - Effect: Conjunction of literals
- Goal: logical sentence

A solution to a planning problem is an action sequence that, when executed in the initial state, results in a state that satisfies the goal.

Example

- Initial state

  `()`

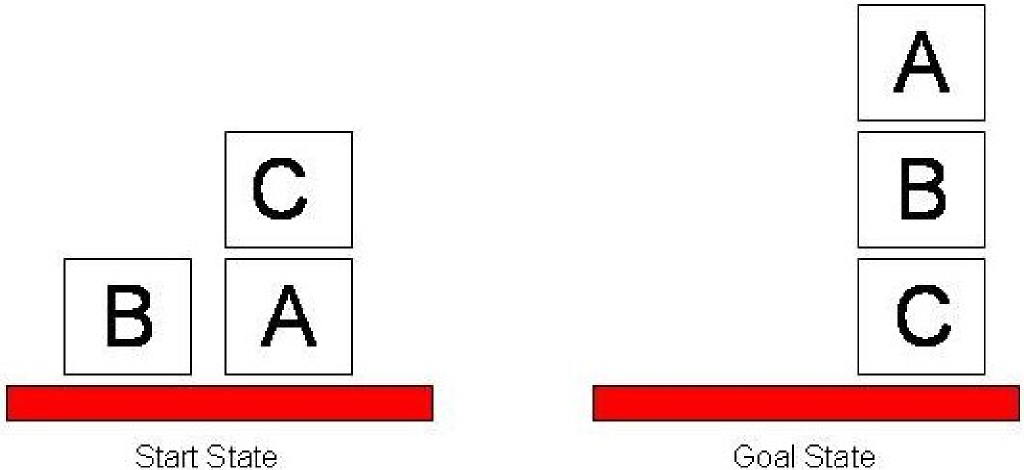
- Actions

`rightShoe`	Precond:	`rightSockOn`
	Effect:	`rightShoeOn`
`rightSock`	Effect:	`rightSockOn`
`leftShoe`	Precond:	`leftSockOn`
	Effect:	`leftShoeOn`
`leftSock`	Effect:	`leftSockOn`

- Goal

  $\mathtt{rightShoeOn} \land \mathtt{leftShoeOn}$

Another Example: Blocks World Planning



A robot arm can pick up a block and move it to another position.
The arm can only pick up one block at a time.

… Formalised in the Planning Description Language

- **Initial state**

  `on(a,table)∧on(b,table)∧on(c,a)∧clear(b)∧clear(c)`

- **Actions**

  Name: `move(X,Y,Z)`

  Precond: `on(X,Y)∧clear(X)∧clear(Z)∧X≠Z∧Y≠Z`

  Effect: `on(X,Z)∧clear(Y)∧¬on(X,Y)∧¬clear(Z)`

  Name: `moveToTable(X,Y)`

  Precond: `on(X,Y)∧clear(X)`

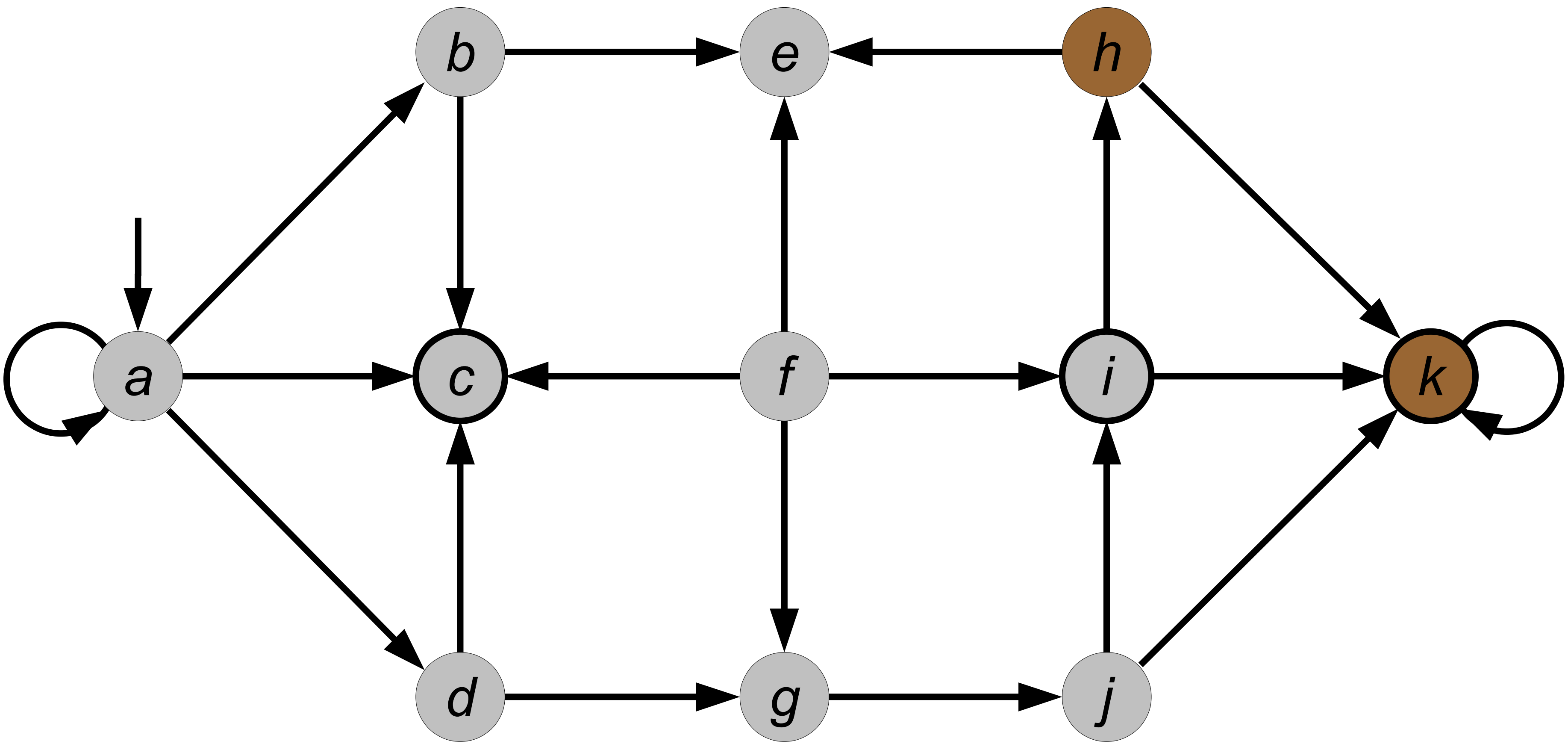
  Effect: `on(X,table)∧clear(Y)∧¬on(X,Y)`

- **Goal**

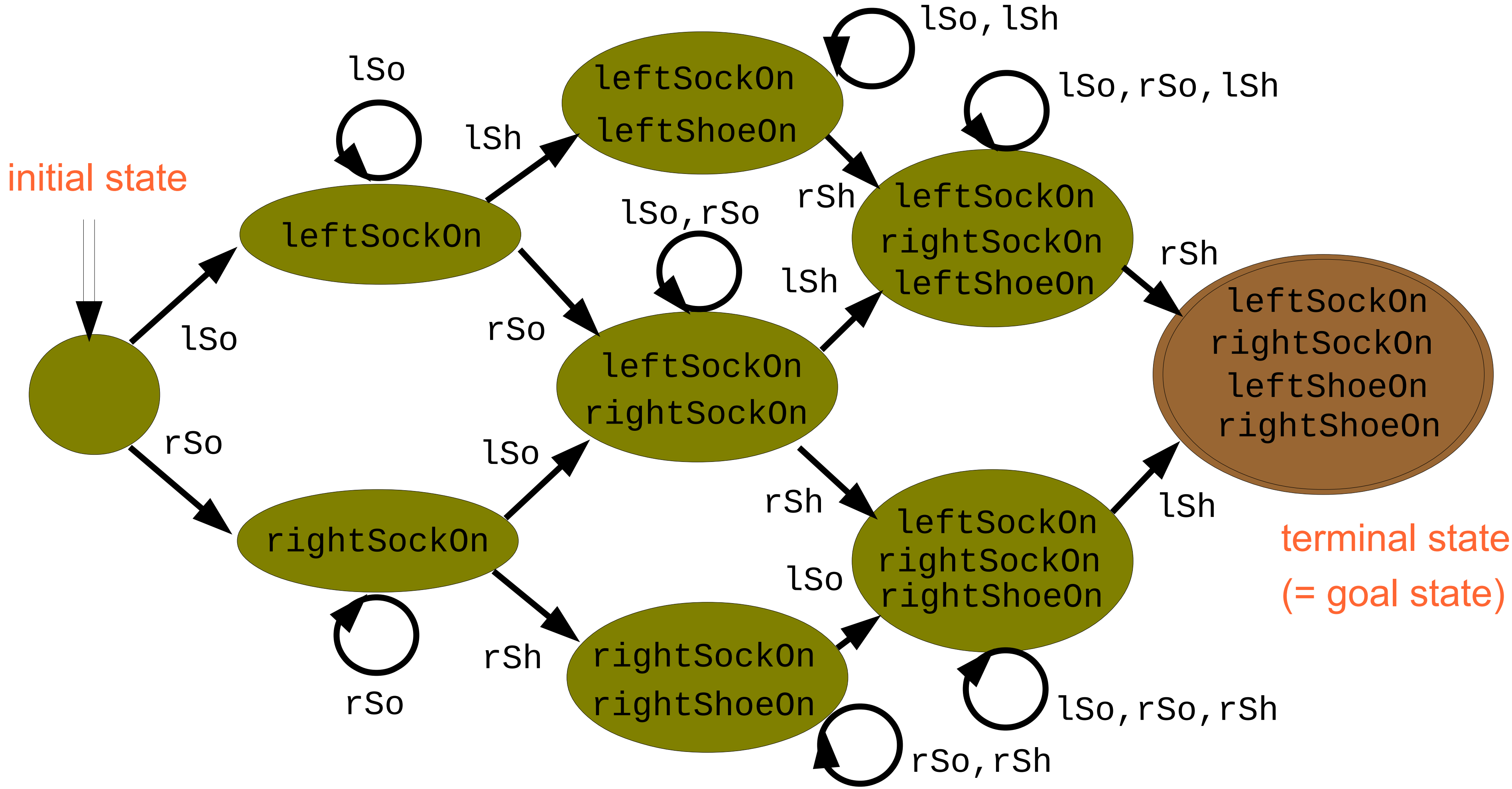
  `on(a,b) ∧ on(b,c)`

Planning by State-Based Search

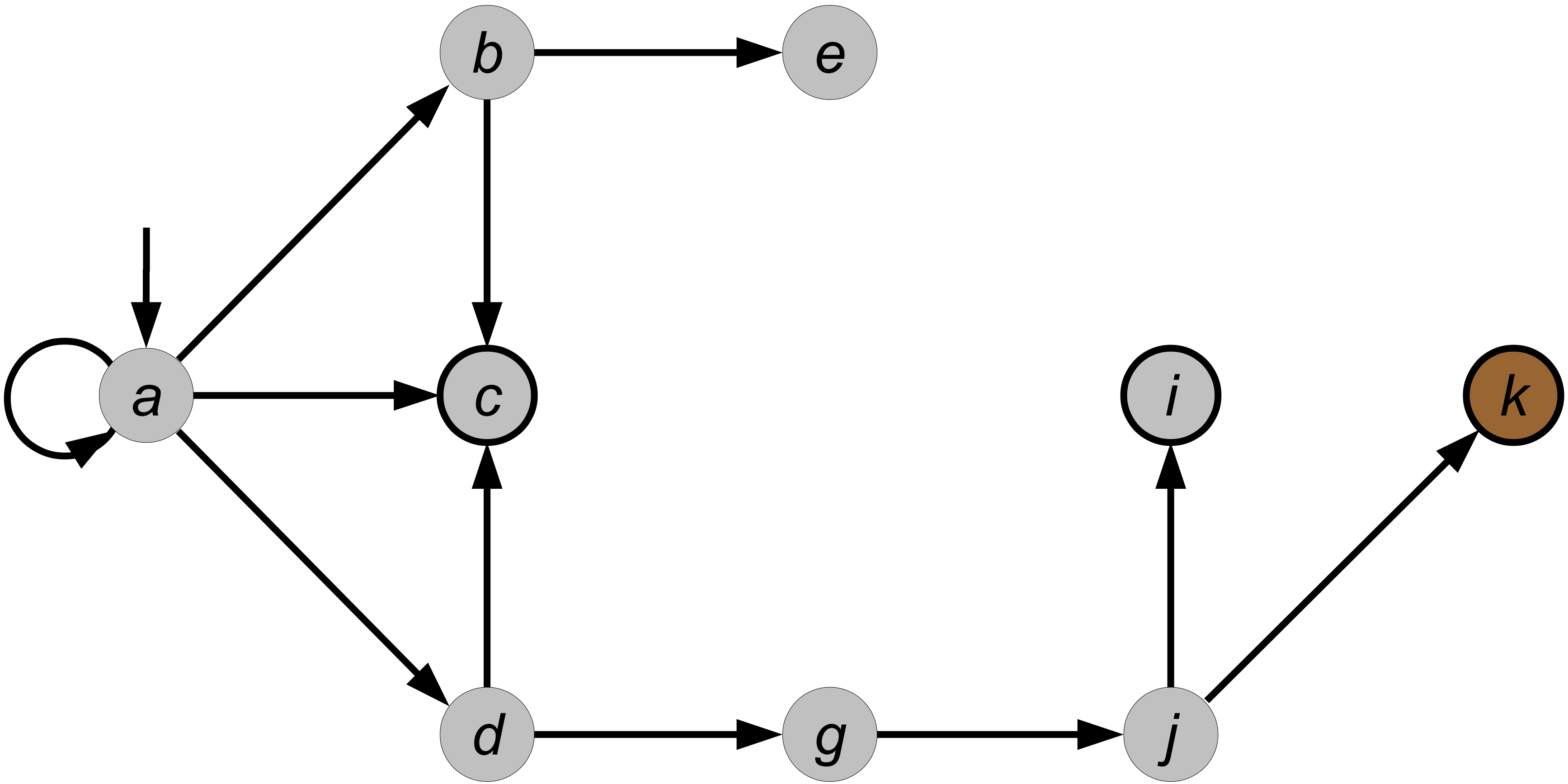
Recap: State Machines



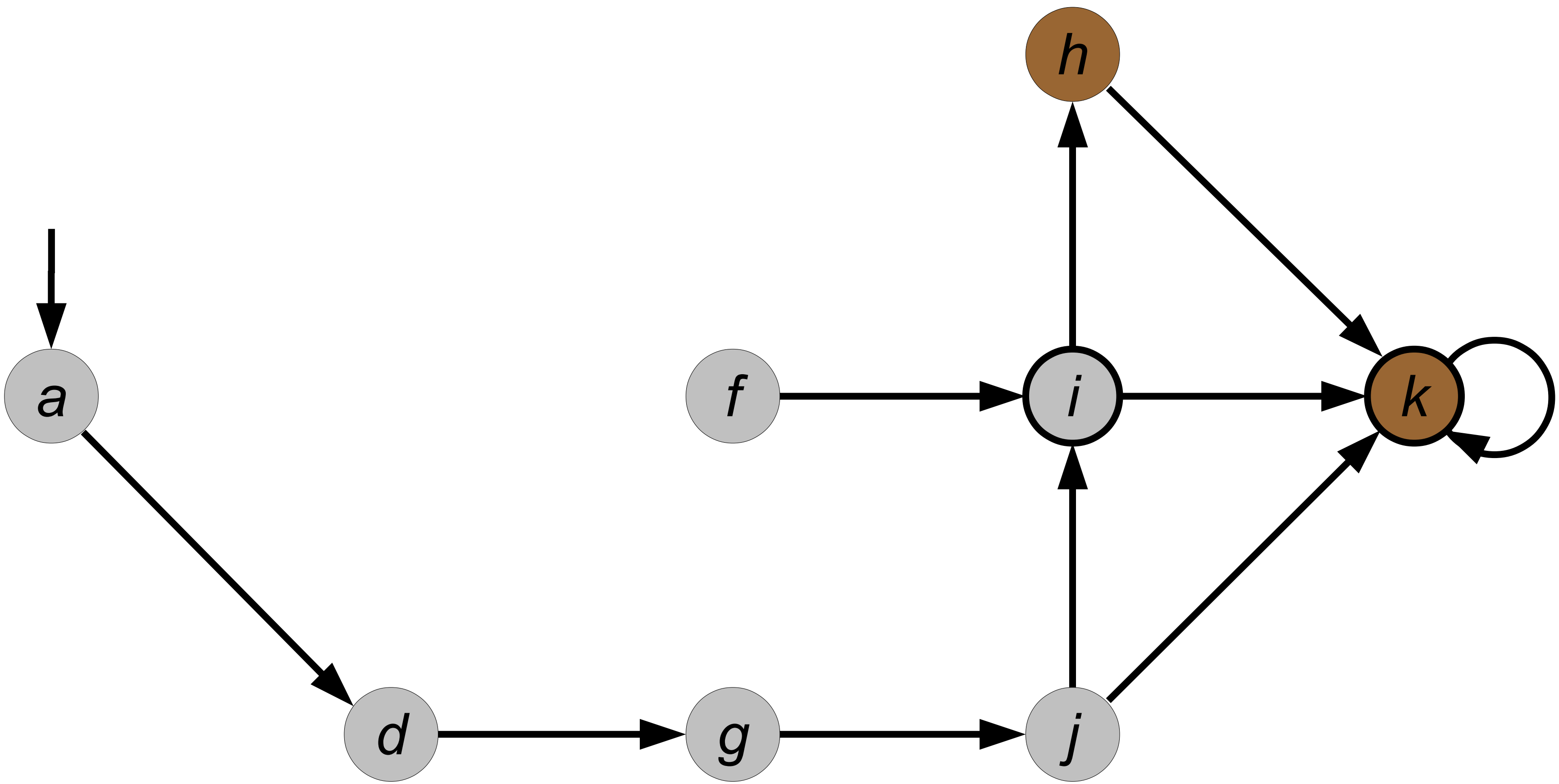
State Machine for the Example Game



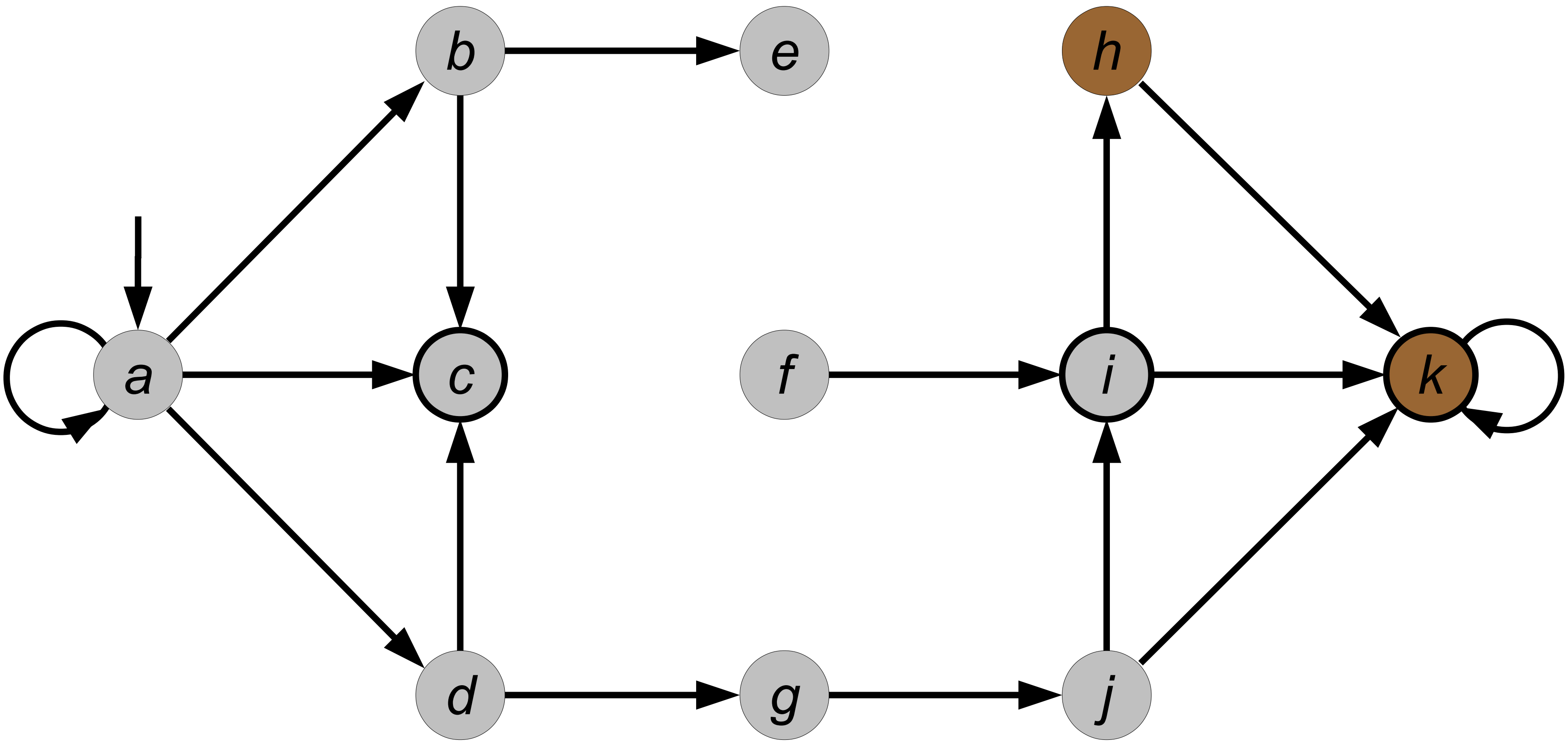
Forward Search



Backward Search

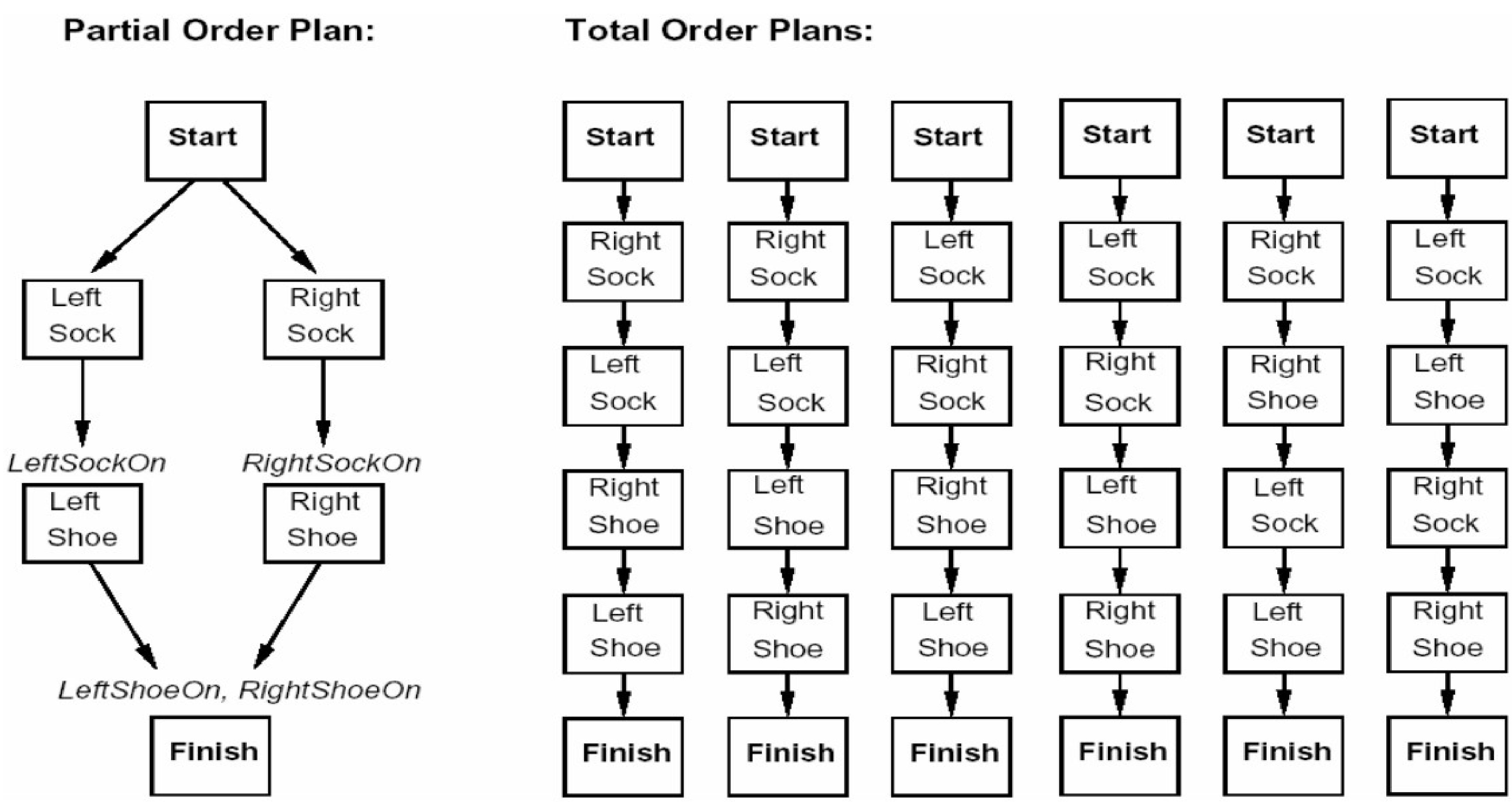


Bidirectional Search

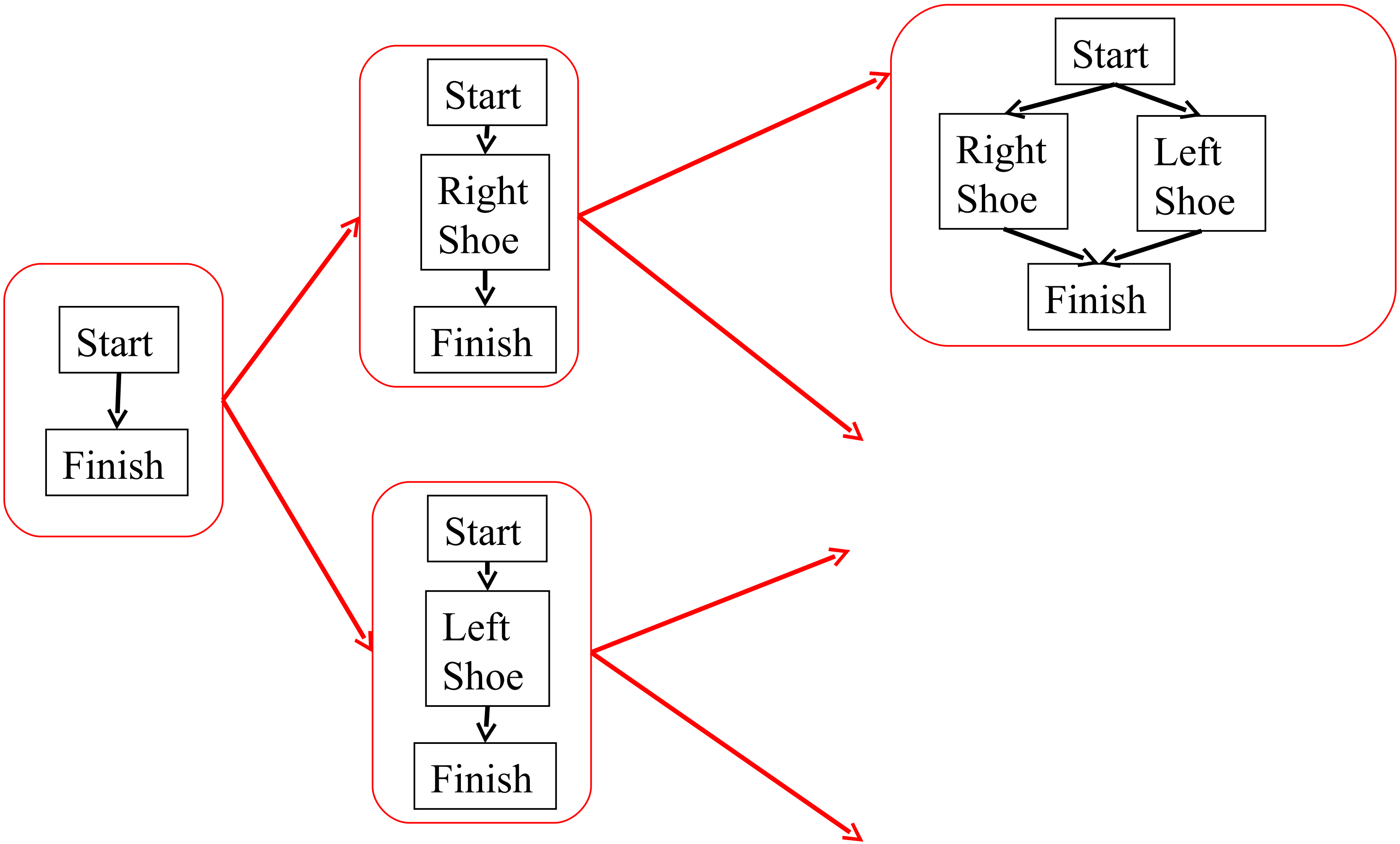


Partial-Order Planning

Partial-Order Plan (POP): Example



Plan-Space Search with POPs



Partial-Order Planning as Search Problem

- Search nodes are (mostly unfinished) partial-order plans
  The initial plan contains only *Start* and *Finish* action
- Plans have 4 components:
  - A set of actions (steps of the plan)
  - A set of ordering constraints A<B (A before B)
  - A set of causal links $A \xrightarrow{p} B$ (read: "A achieves p for B")
  - A set of open preconditions
- A plan is consistent if there are no cycles in the ordering constraints and no conflicts with the causal links.
- An action C conflicts with a causal link $A \xrightarrow{p} B$ if C has the effect ¬p and C could come after A and before B
- A consistent plan with no open preconditions is a solution.

Algorithm for Solving POPs

- Define effect of *Start* := initial state of the planning problem (no percond)
- Define precond of *Finish* := goal of the planning problem (no effect)
- The initial plan contains *Start* and *Finish*, the ordering constraint *Start*<*Finish*, no causal links. All preconditions of *Finish* are open.

- Repeat
  - Pick an open precondition p (of an action B in the plan)
  - Pick an action A with effect p
  - Add the causal link A $\xrightarrow{p}$ B and the ordering constraint A<B
    (if A is new to the plan, add *Start*<A and A<*Finish*)
  - If a conflict arises between a causal link A $\xrightarrow{p}$ B and an action C: add either B<C or C<A

- Retry (with different choices) if plan is inconsistent, stop if solution is found

Example: Flat Tire Problem

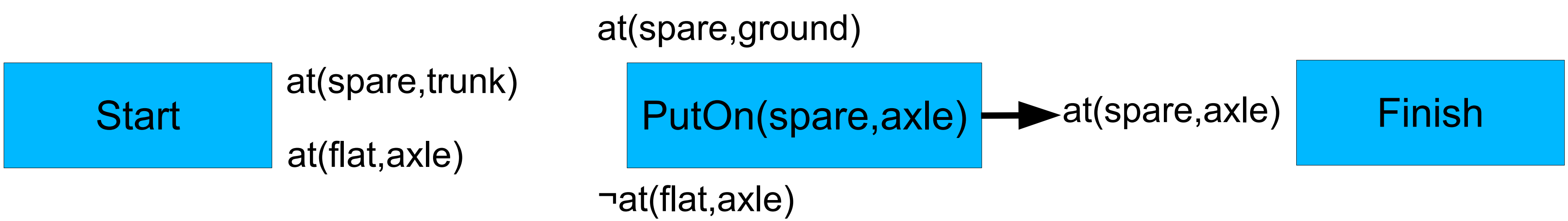
- **Initial state** `at(flat,axle)∧at(spare,trunk)`
- **Actions**

  Name: `remove(spare,trunk)`
  Precond: `at(spare,trunk)`
  Effect: `¬at(spare,trunk)∧at(spare,ground)`

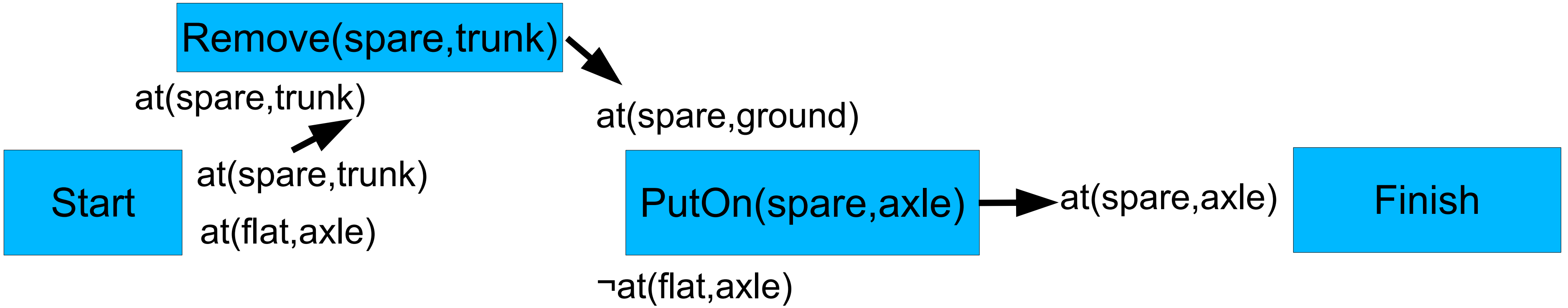
  Name: `remove(flat,axle)`
  Precond: `at(flat,axle)`
  Effect: `¬at(flat,axle)∧at(flat,ground)`

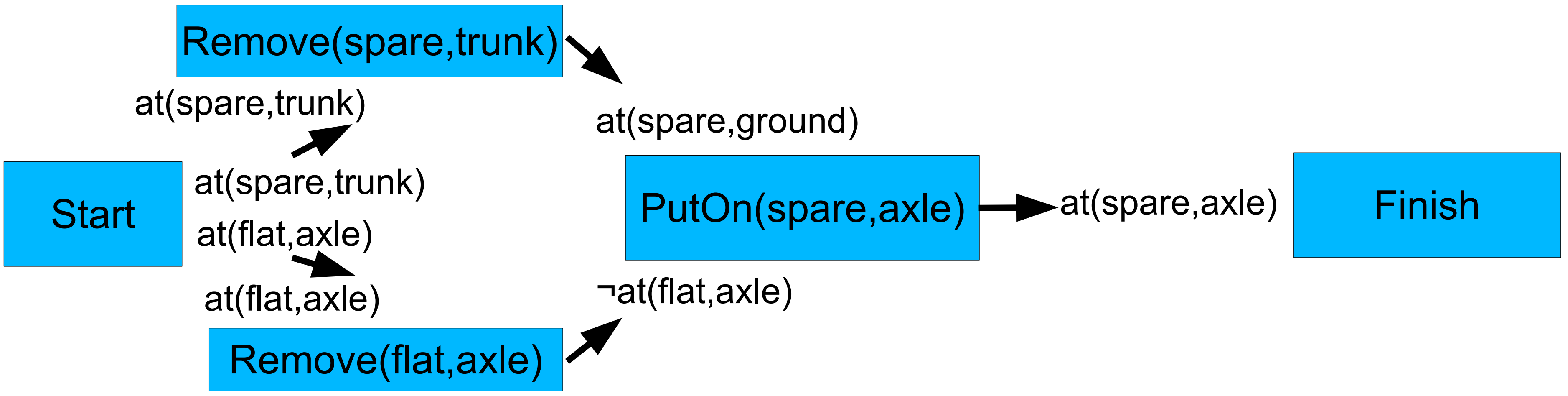
  Name: `putOn(spare,axle)`
  Precond: `at(spare,ground)∧¬at(flat,axle)`
  Effect: `¬at(spare,ground)∧at(spare,axle)`
- **Goal** `at(spare,axle)`

POP for the Flat Tire Problem (1)



POP for the Flat Tire Problem (2)





Planning with Propositional Logic

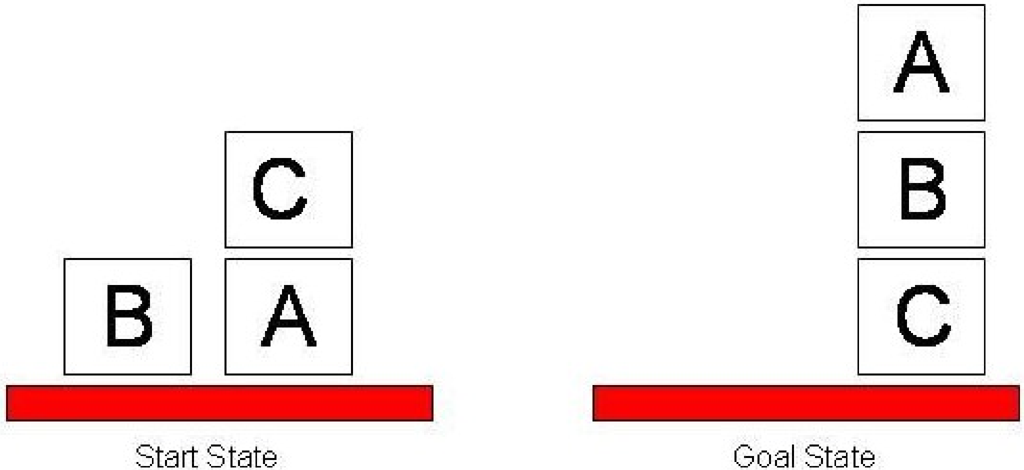
Encoding Planning Problems in Propositional Logic

- Planning can be done by testing the satisfiability of a logical sentence:

  `initial state` $\land$ `all possible actions` $\land$ `goal`

- This sentence contains propositions for every action occurrence
  - A model will assign *true* to an action A iff A is part of the correct plan
- An assignment that corresponds to an incorrect plan will not be a model because of inconsistency with the assertion that `goal` is true
- If the planning problem is unsolvable, there will be no model for the sentence
- Planners based on satisfiability can handle large planning problems

Recap: Blocks World Planning



A robot arm can pick up a block and move it to another position.
The arm can only pick up one block at a time.

Example: Blocks World Planning as Satisfiability (1)

- Encoding of the initial state

```
on(a,table)^0
on(b,table)^0
on(c,a)^0
clear(b)^0
clear(c)^0
```

- Encoding of action preconditions

```
move(X,Y,Z)^T => on(X,Y)^T ∧ clear(X)^T ∧ clear(Z)^T
moveToTable(X,Y)^T => on(X,Y)^T ∧ clear(X)^T
```

(for all `X,Y,Z∈{a,b,c,table}, T∈{0,1,2,...,max-1}, X≠Z, Y≠Z`)

- Action exclusion axioms

```
¬(move(X,Y,Z)^T ∧ moveToTable(X',Y')^T)
¬(moveToTable(X,Y)^T ∧ moveToTable(X',Y')^T)
¬(move(X,Y,Z)^T ∧ move(X',Y',Z')^T)
```

(for suitable `X,X'`,...)

Example: Blocks World Planning as Satisfiability (2)

- Encoding of action effects

```
move(X,Y,Z)^T => on(X,Z)^T+1 ∧ clear(Y)^T+1
move(X,Y,Z)^T => ¬on(X,Y)^T+1 ∧ ¬clear(Z)^T+1
moveToTable(X,Y)^T => on(X,table)^T+1 ∧ clear(Y)^T+1
moveToTable(X,Y)^T => ¬on(X,Y)^T+1
```

- Explanation closure axioms

```
on(X,Z)^T+1 => on(X,Z)^T ∨ move(X,Y,Z)^T ∨
               (moveToTable(X,Y)^T ∧ Z=table)
clear(Y)^T+1=> clear(Y)^T ∨
               move(X,Y,Z)^T ∨ moveToTable(X,Y)^T
```

- Encoding of the goal

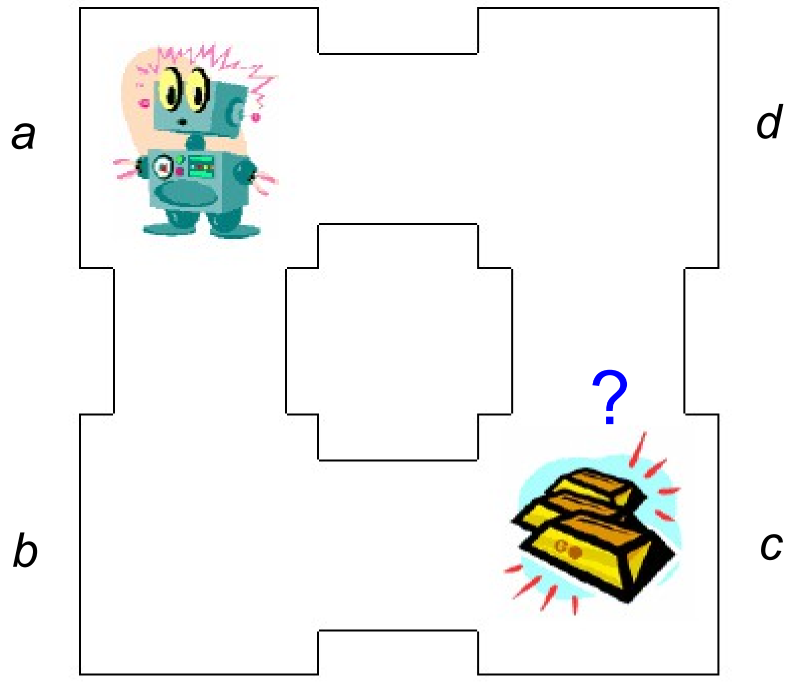
```
on(a,b)^max ∧ on(b,c)^max
```

Solution (`max=3`): a model that contains

```
moveTable(c,a)^0, move(b,table,c)^1, move(a,table,b)^2
```

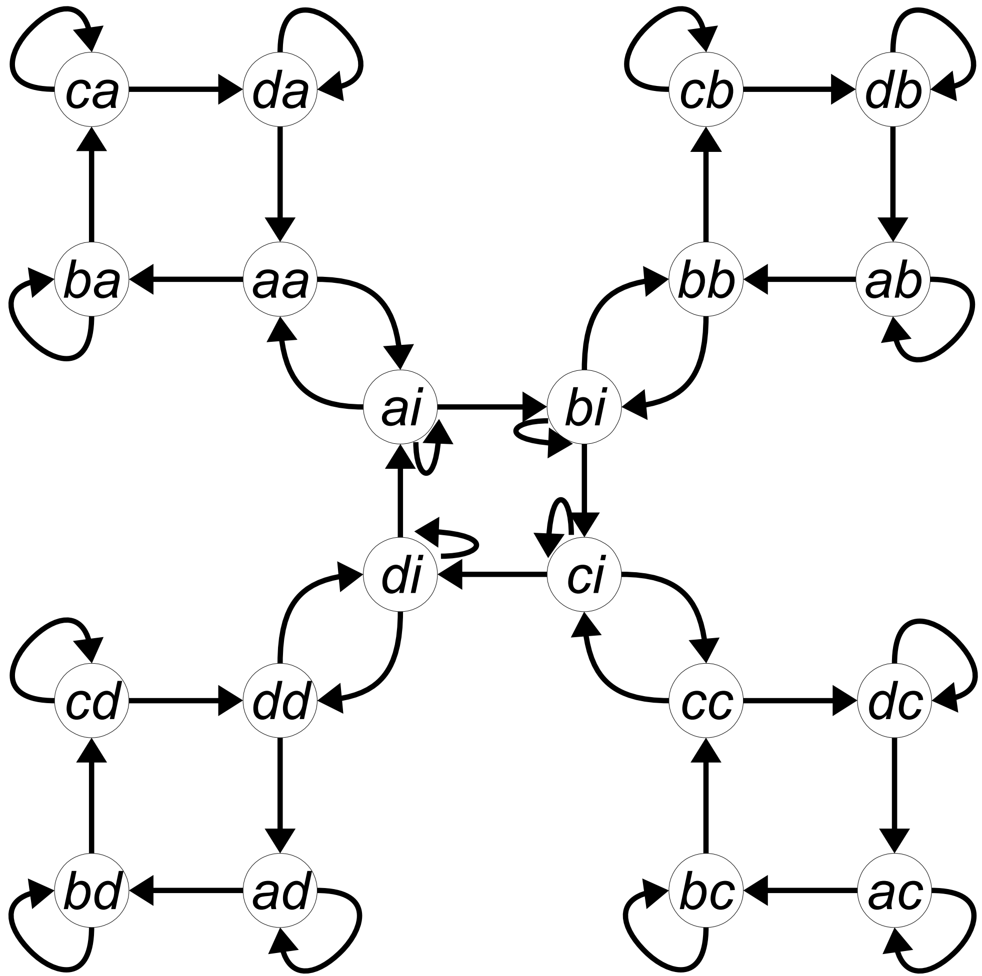
Conditional Planning

Planning Under Incomplete Information: Maze World



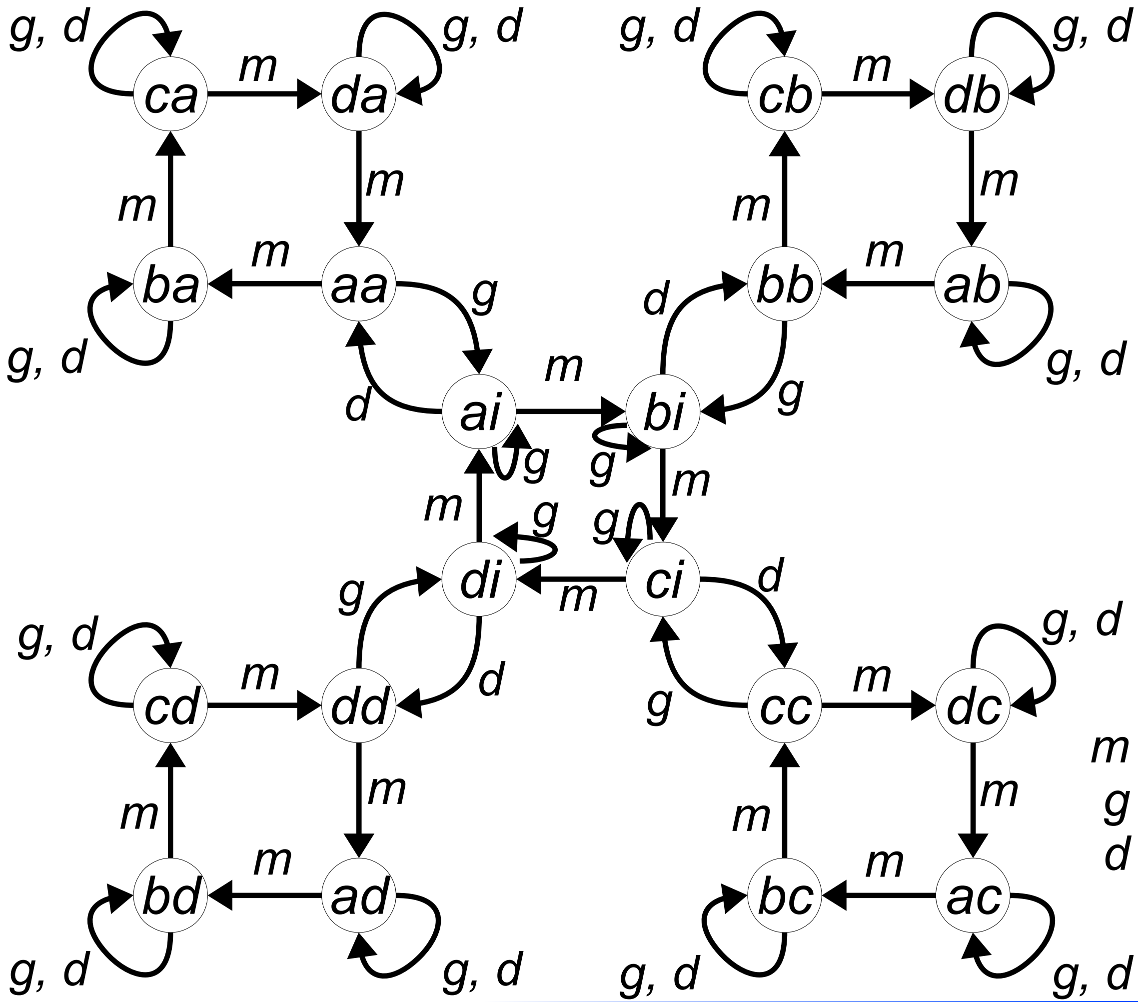
Initial State: *ac* (robot in *a*, gold in *c*)

Environment Model



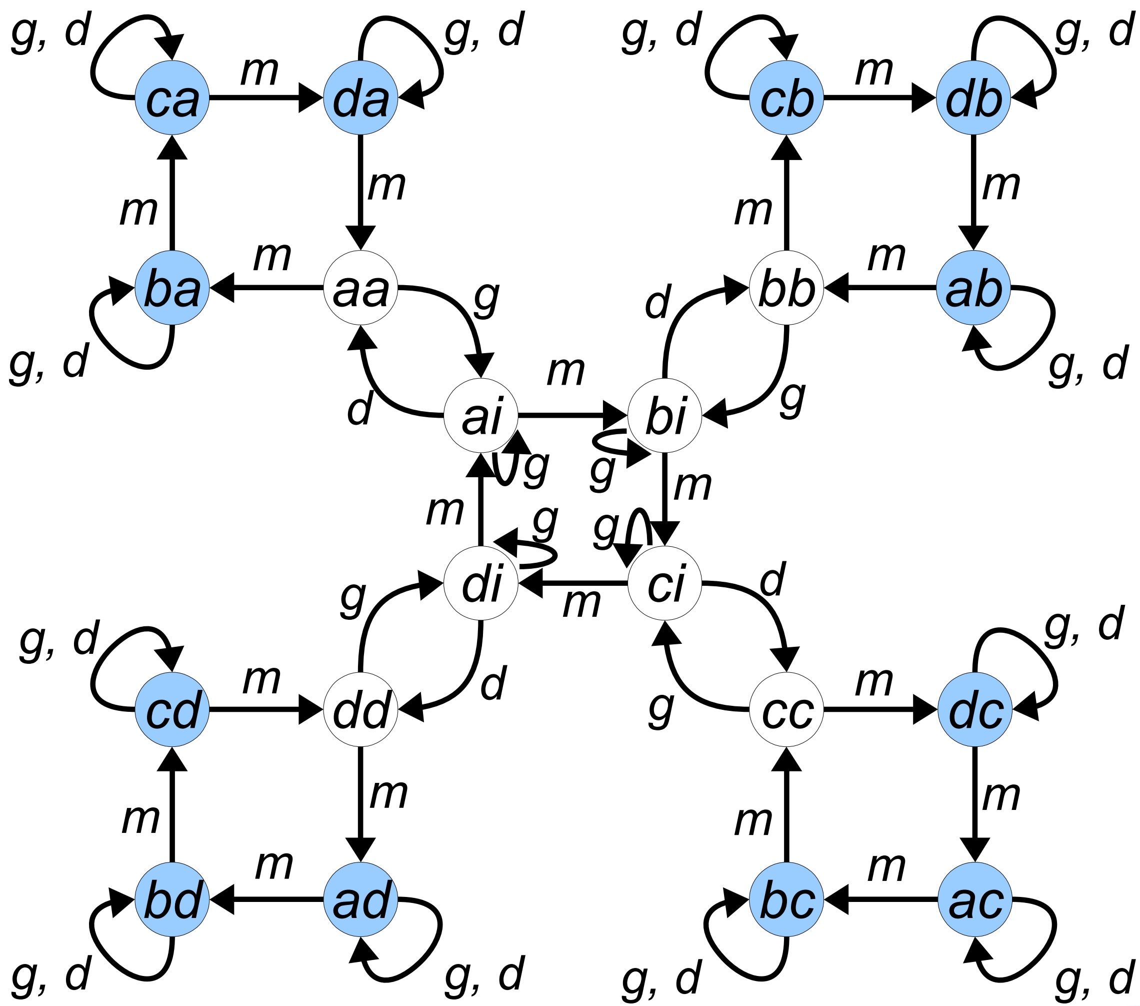
i = "in hand"

Agent Actions



m = "move clockwise"
g = "grab"
d = "drop"

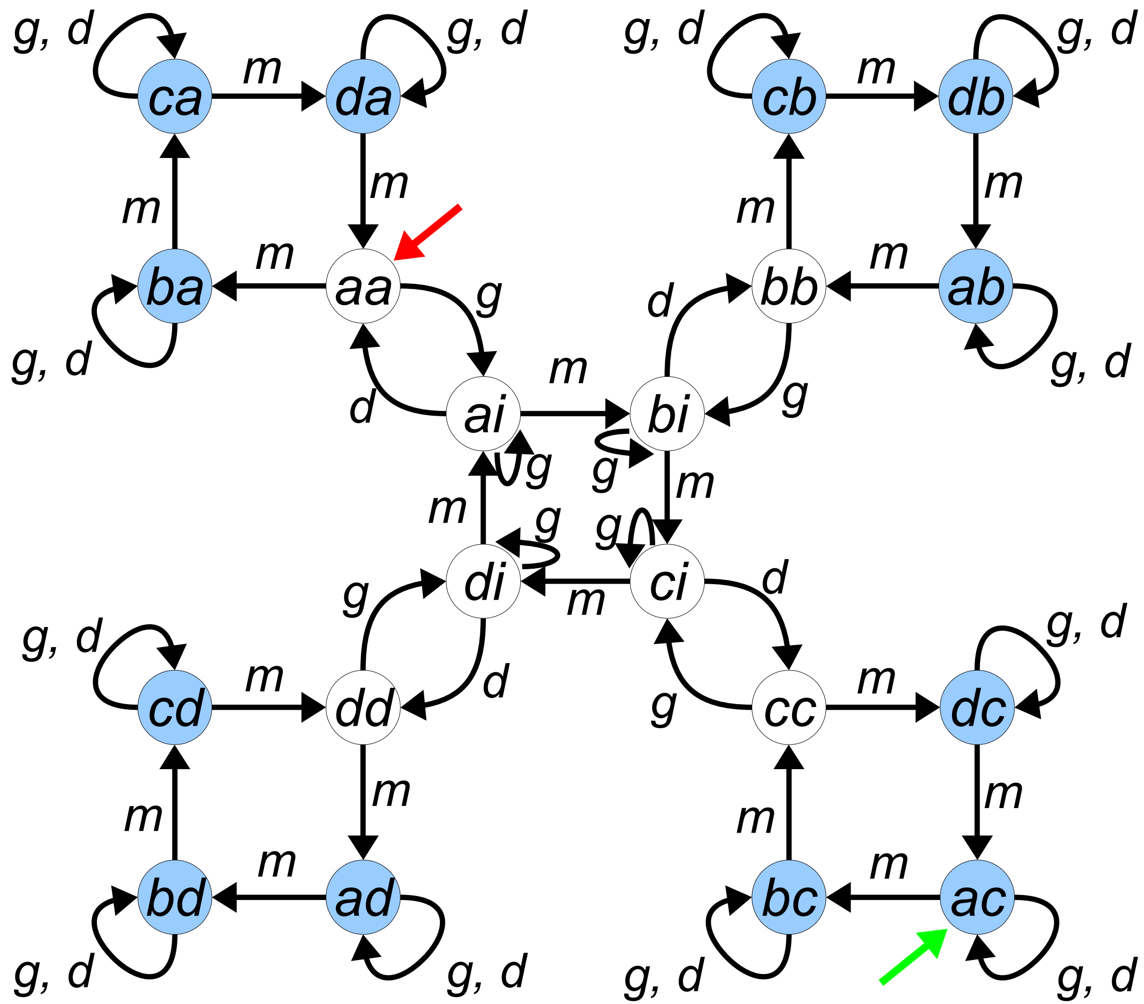
Agent Percepts



sees gold

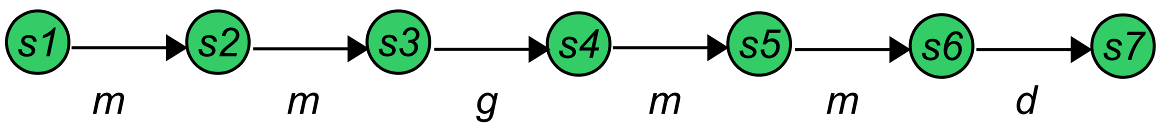
doesn't see gold

Initial State and Goal



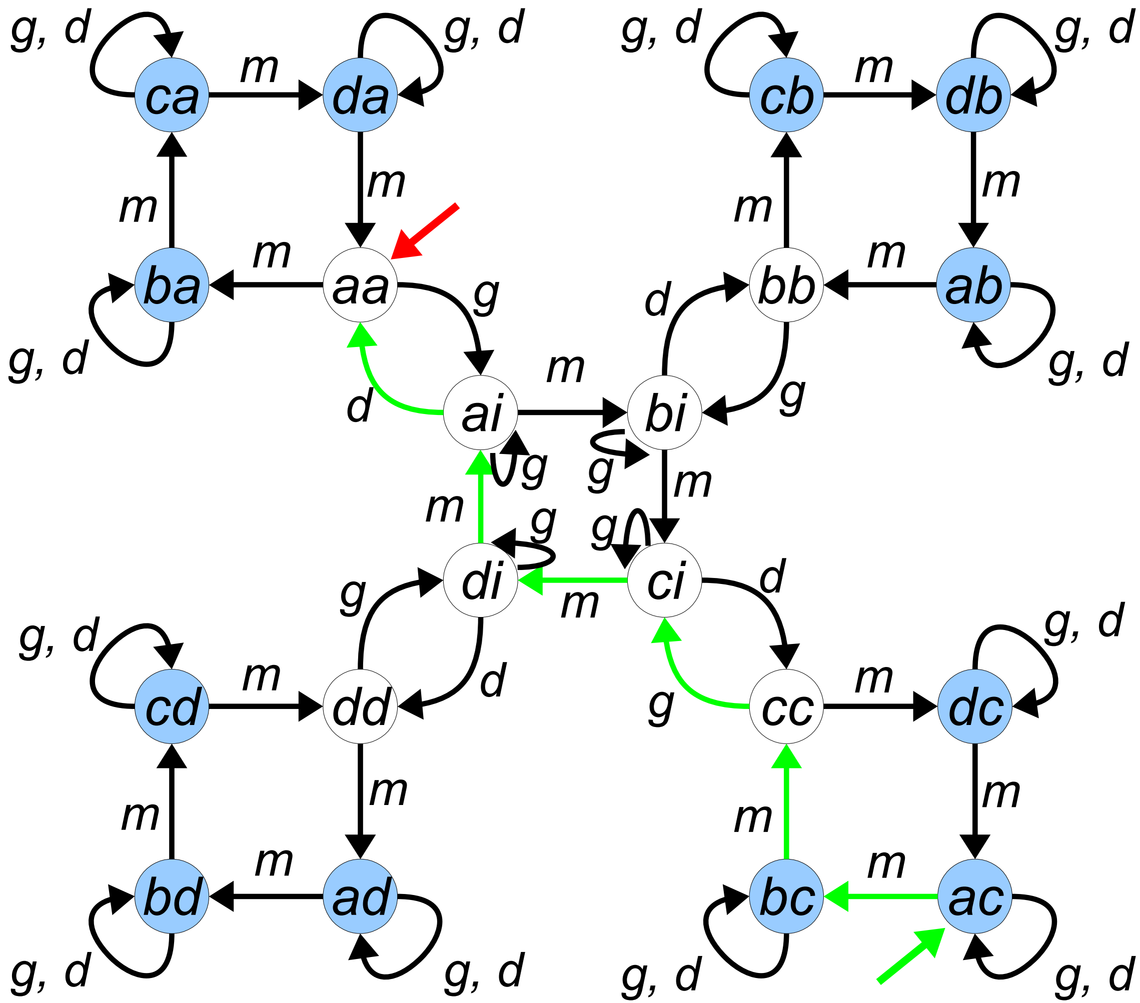
Planning

Planning is the process of finding a transition diagram *for our agent* that causes its environment to go from any initial state to a goal state.



Planning can be done offline and the resulting plan/program installed in the agent *or* the planning can be done online followed by execution.

State Space Planning



Incompleteness

Possible sources of incompleteness:

Partial knowledge of

- Initial state
- Transition diagram for environment
- Goal

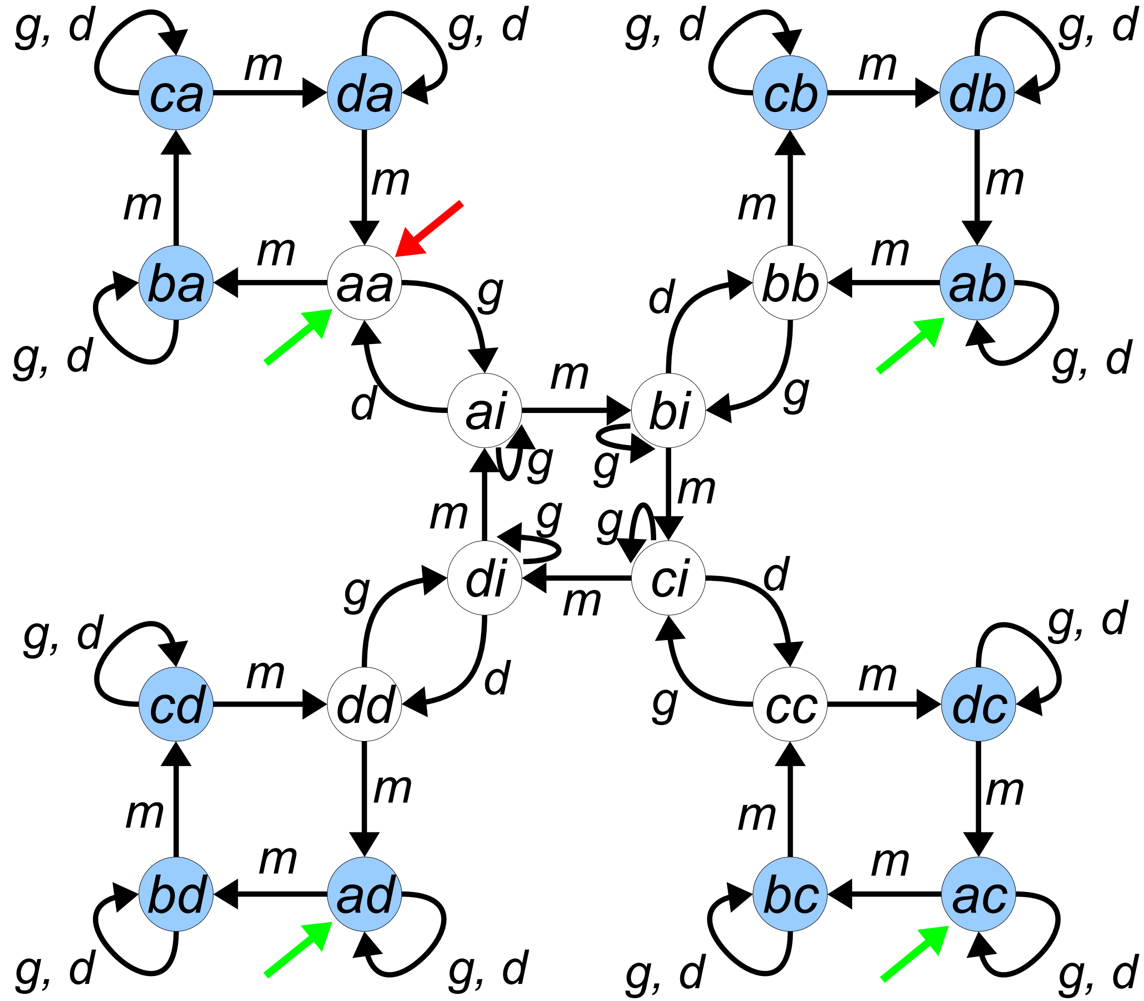
Complete Planning Techniques under incomplete information

- Coercion (e.g. do the *grab* action at all locations)
- Conditional plan (e.g. if see the gold grab it; else move)

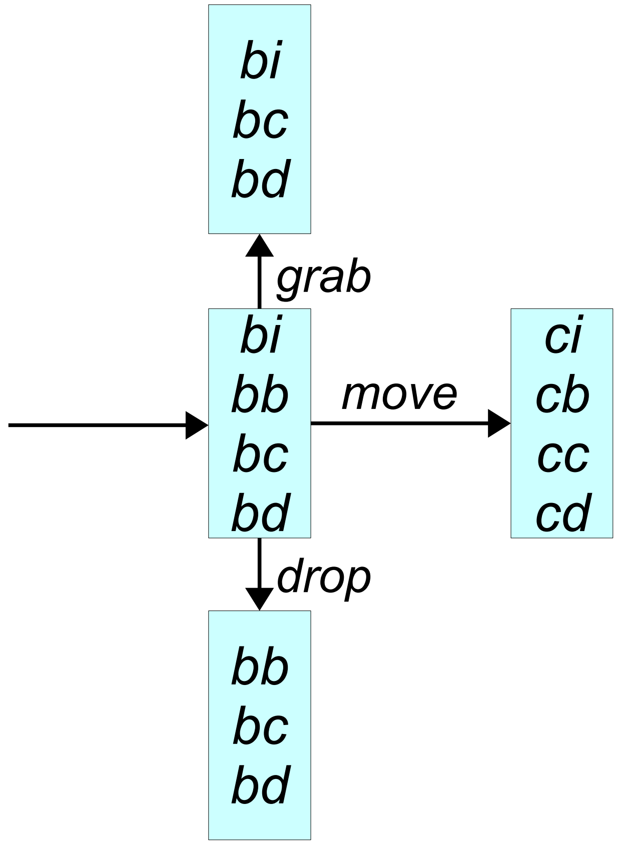
Postponement Techniques

- Delayed planning

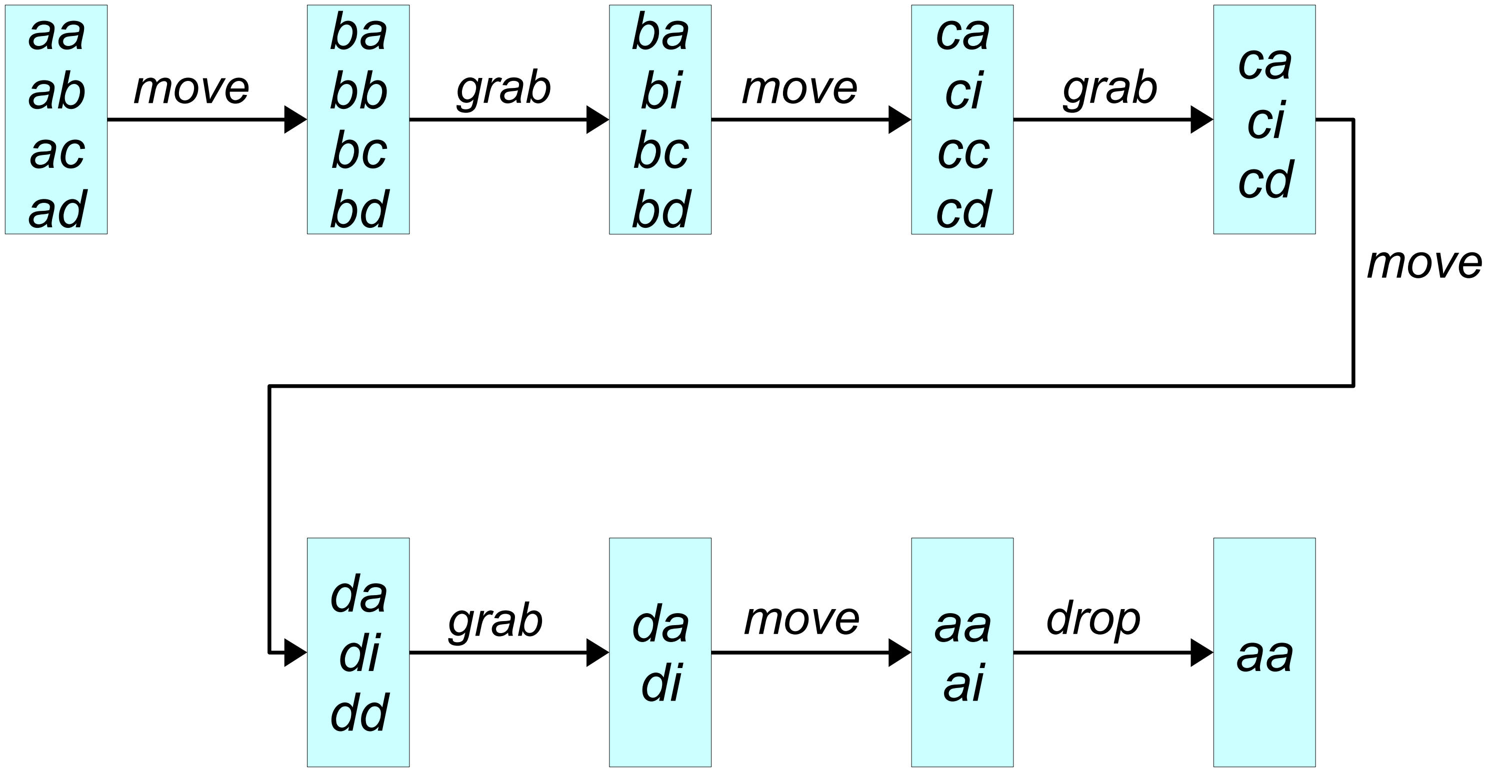
Initial State Uncertainty



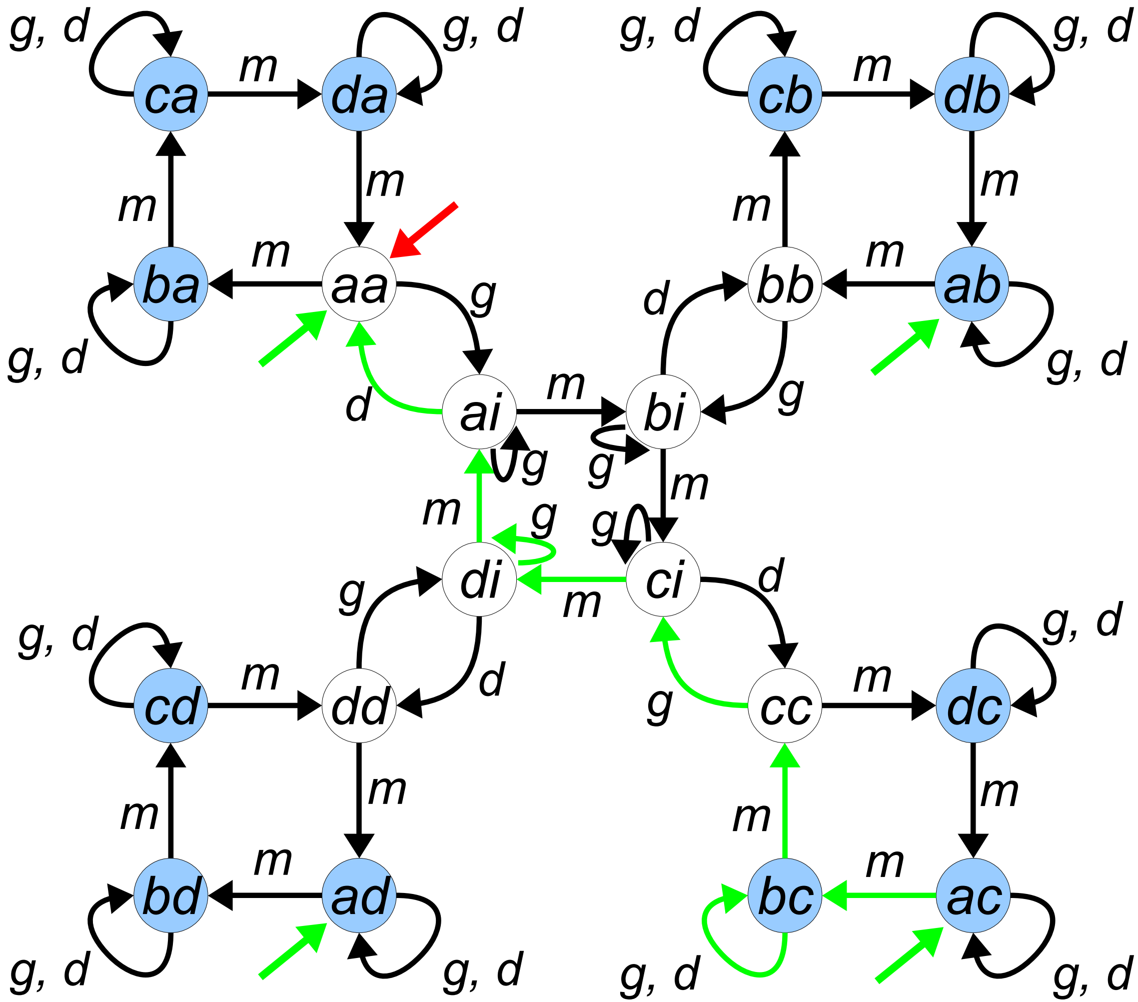
Sequential State Set Progression



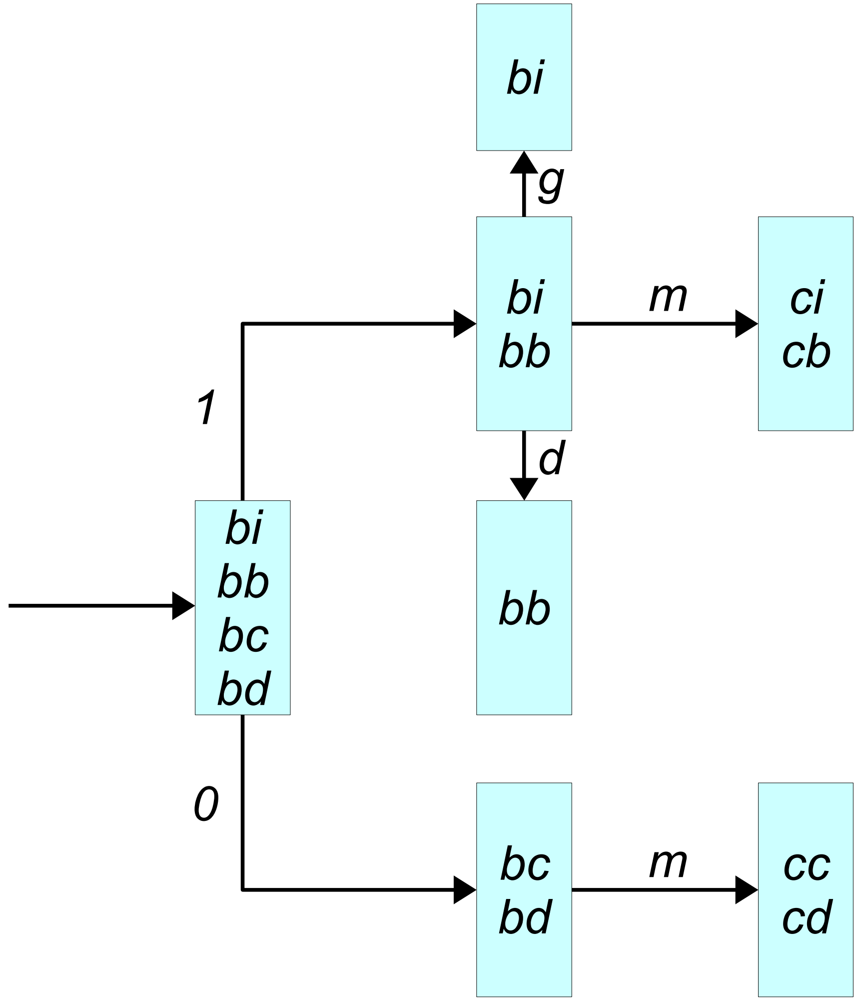
Sequential State Set Plan



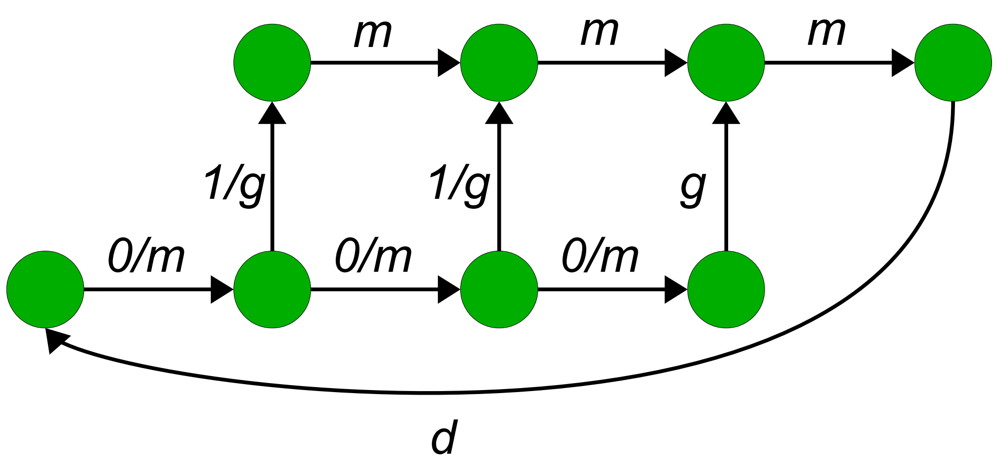
Plan Execution



Conditional State Set Progression



Conditional State Set Plan



Background Reading

Planning

- Russell & Norvig AIMA (3rd ed): Chapter 10
  (2nd edition: Chapter 11)

Comparison

Sequential plan

- possible that no plan exists
- plan may contain redundant moves

Conditional plan

- large search space

Delayed planning

- irreversibility problematic

As we can see from this analysis, it is sometimes desirable for an agent to do only a portion of its planning up front, secure in the knowledge that it can do more later as necessary.

Planning can be done offline and the resulting plan/program executed during play *or* the planning can be done online and interleaved with execution.