CMSC 471
Fall 2012

Class #3

Thu 9/6/11
Problem Solving as Search

Kevin Winner, winnerk1@umbc.edu

Today’s class

- Uninformed Search
  - Goal-based agents
  - Representing states and operators
  - Example problems
  - Generic state-space search algorithm
- Hopefully a little more Lisp

Search

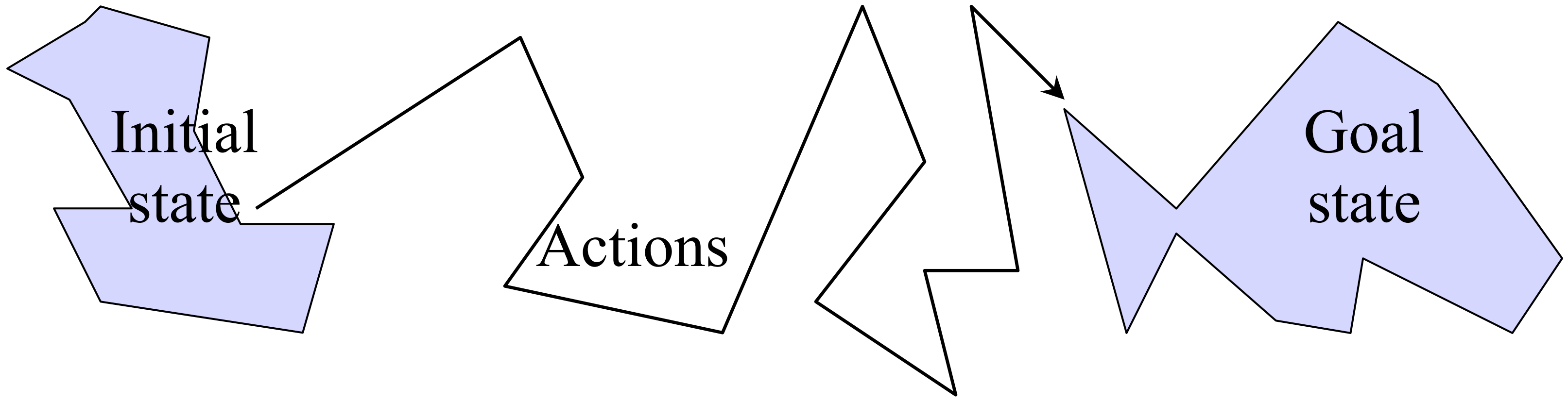
Sections 3.1-3.3

Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Building goal-based agents

To build a goal-based agent we need to answer the following questions about our problem:

- What *relevant* information is necessary to encode in order to describe the **state** of the world? (percepts)
- What are the **actions**? (actuators)
- What is the **goal** to be achieved?



Representing states

- A **state** must sufficiently encode all the important information about the environment at one point in time
  - The state our agent begins in is called the **start state**
  - We assume that our environment is **fully observable**
- The **state space** is the collection of all states which can be reached by applying any sequence of actions from our start state
  - Reducing the size of the state space is often very important
- The **size of a problem** is usually described in terms of the **number of states** that are possible.
  - Tic-Tac-Toe has 3^9 states.
  - A Rubik's Cube has about 10^{19} states.
  - Chess has about 10^{150} states in a typical game.

What are the actions?

- Characterize the **primitive actions** or events that are available for making changes in the world in order to achieve a goal.
- **Deterministic**, **known** world: no uncertainty in an action's effects. Given an action (a.k.a. **operator** or move) and a description of the **current world state**, the action completely specifies
  - whether that action *can* be applied to the current world (i.e., is it applicable and legal), and
  - what the exact state of the world will be after the action is performed in the current world (i.e., no need for "history" information to compute what the new world looks like).

Representing actions

- Note also that actions in this framework can all be considered as **discrete events** that occur at an **instant of time**.
  - For example, if "Mary is in class" and then performs the action "go home," then in the next situation she is "at home." There is no representation of a point in time where she is neither in class nor at home (i.e., in the state of "going home").
- The number of actions depends on the **representation** used in describing a state.
  - In the 8-puzzle, we could specify 4 possible moves for each of the 8 tiles, resulting in a total of **4*8=32 operators**.
  - On the other hand, we could specify four moves for the "blank" square and we would only need **4 operators**.
- Representational shift can greatly simplify a problem!

What is the goal to be achieved?

- Could describe a situation we want to achieve, a set of properties that we want to hold, etc.
- Typically we define a "**goal test**" so that we know what it means to have achieved/satisfied our goal.
  - any state in which the goal test is satisfied is called a **goal state**

Then the task is...

- Given a complete description of the problem, find a sequence of actions which, when executed in order, transition the agent from the start state to a goal state

Well-defined problems

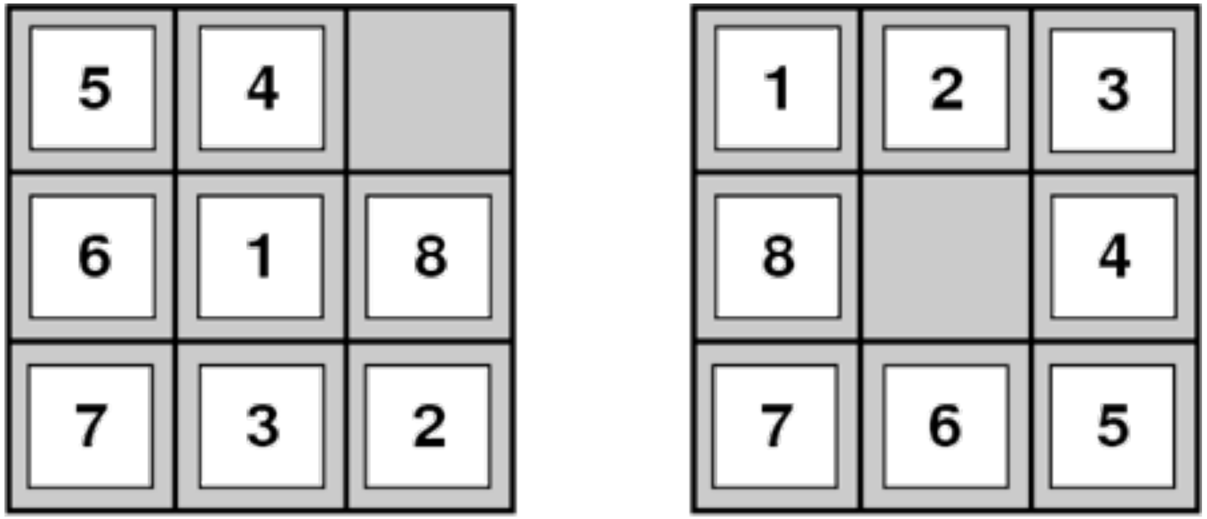
- Formally, a well-defined problem definition defines:
  - the **state space**
  - the possible **actions**
  - the **goal test**
  - an **initial state**
  - a **transition model**
  - a **path cost function**

Some example problems

- Toy problems and micro-worlds
  - 8-Puzzle
  - 8-Queens
  - Missionaries and Cannibals
  - Remove 5 Sticks
  - Water Jug Problem
- What is the problem definition for each of these puzzles?

8-Puzzle

Given an initial configuration of 8 numbered tiles on a 3 x 3 board, move the tiles in such a way so as to produce a desired goal configuration of the tiles.



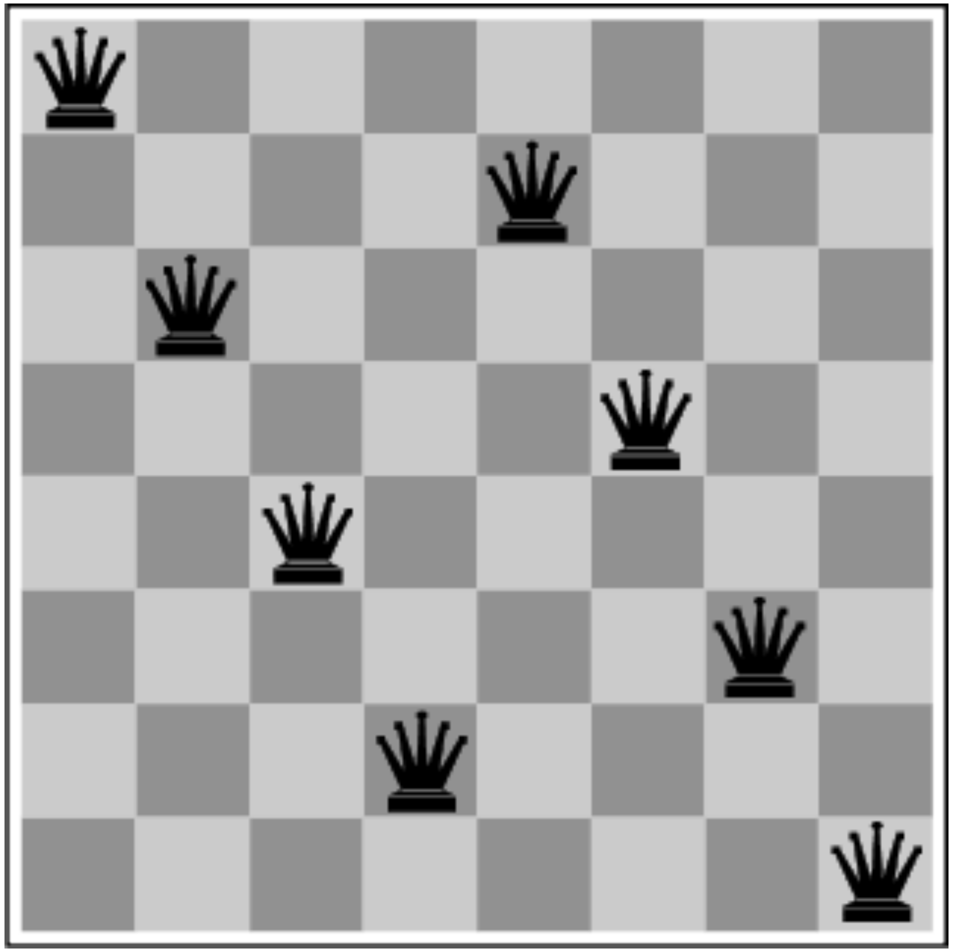
Start State Goal State

8-Puzzle

- **States:** 3 x 3 array configuration of the tiles on the board.
- **Actions:** Move Blank Square Left, Right, Up or Down.
  - This is a more efficient encoding of the operators than one in which each of four possible moves for each of the 8 distinct tiles is used.
- **Transition Model:** Swap the positions of the blank and the tile it moves into
- **Initial State:** Any arbitrary configuration of the board.
- **Goal:** One particular configuration of the board.
- **Path cost function:** Each action costs 1, so the total path cost is the number of actions taken

The 8-Queens Problem

Place eight queens on a chessboard such that no queen attacks any other!



8-queens

- **States:** Any arrangement of 0-8 queens
- **Actions:** Place a queen on the board
- **Transition Model:** Add a queen to the state representation
- **Initial State:** A board with 0 queens
- **Goal:** A board with 8 queens without any attacking each other
- **Path cost function:** The path cost is equal to the number of queens placed already
  – Yes, this means all paths which reach the goal have the same cost!

Missionaries and Cannibals

Three missionaries and three cannibals come to a river and find a [row]boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten.

How shall they cross?

Missionaries and Cannibals

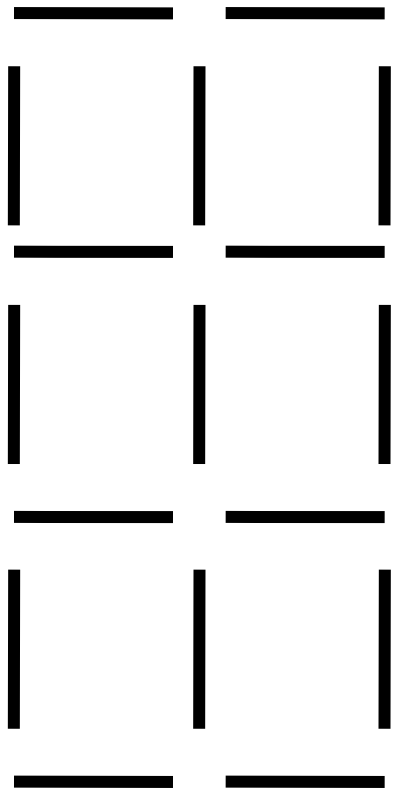
- **States:** Locations of all 6 people and the boat
- **Actions:** Send the boat across with 1 or 2 people
  - The legal actions will differ greatly in each state
- **Transition Model:** Move the boat and the people in it to the other side of the river
- **Initial State:** All 6 people and the boat on the same side of the river
- **Goal:** All 6 people are on the other side of the river
- **Path cost function:** Once again, each step costs 1, so the path cost is the number of steps

Missionaries and Cannibals Solution

	Near side			***Far side***
0 Initial setup:	MMMCCC	B		-
1 Two cannibals cross over:	MMMC		B	CC
2 One comes back:	MMMCC	B		C
3 Two cannibals go over again:	MMM		B	CCC
4 One comes back:	MMMC	B		CC
5 Two missionaries cross:	MC		B	MMCC
6 A missionary & cannibal return:	MMCC	B		MC
7 Two missionaries cross again:	CC		B	MMMC
8 A cannibal returns:	CCC	B		MMM
9 Two cannibals cross:	C		B	MMMCC
10 One returns:	CC	B		MMMC
11 And brings over the third:	-		B	MMMCCC

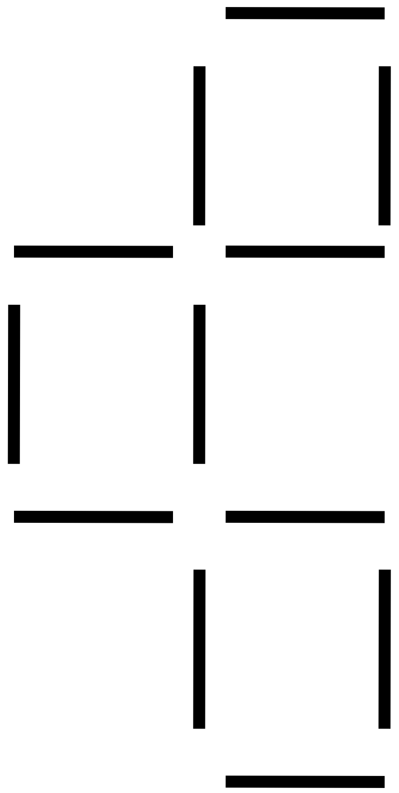
Remove 5 Sticks

- Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.



Remove 5 Sticks

- Given the following configuration of sticks, remove exactly 5 sticks in such a way that the remaining configuration forms exactly 3 squares.



Remove 5 Sticks

- **States:** The set of all remaining sticks and their orientations
- **Actions:** Remove a stick
- **Transition Model:** Remove that stick from the state representation
- **Initial State:** 17 sticks arranged as pictured
- **Goal:** 12 sticks arranged in 3 squares
- **Path cost function:** Path cost = # of sticks removed
  - Like the 8-queens problem, all paths which reach the goal will be the same length

Formalizing search in a state space

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **edges**, and each edge is directed from a node to another node
- Each **node** represents a state
  - In an implementation, a node will typically contain a list of the actions available at that state, along with other bookkeeping data
- Each **edge** corresponds to an action
  - The source of the edge is the state in which that action is valid
  - The destination of the edge is the state you transition to after executing that action
- One node is designated as the start state
- The nodes whose states satisfy the goal test are designated as goal nodes

Water Jug Problem

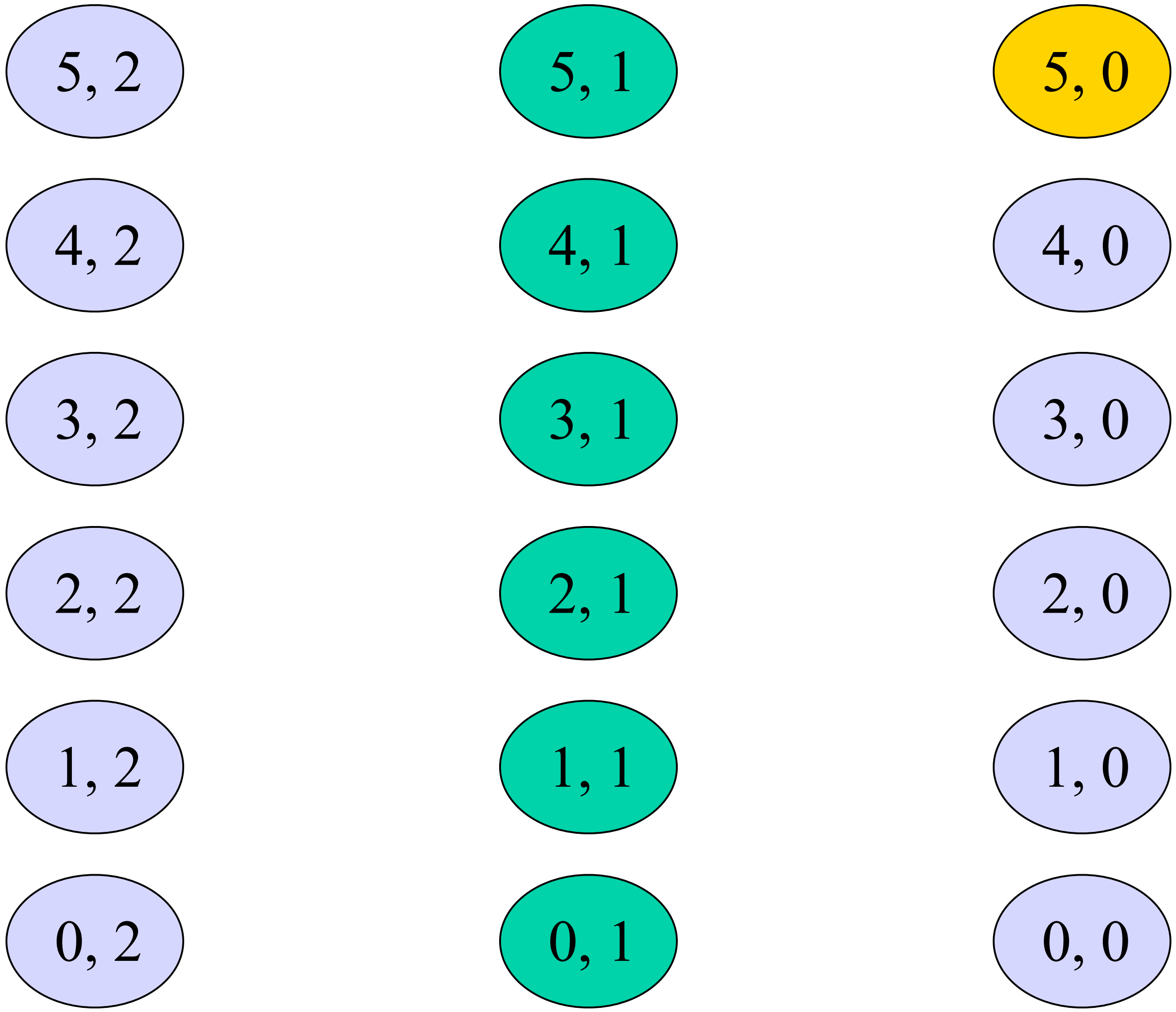
Given a full 5-gallon jug and an empty 2-gallon jug, the goal is to fill the 2-gallon jug with exactly one gallon of water.

- State = (x,y), where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug
- Initial State = (5,0)
- Goal State = (*,1), where * means any amount

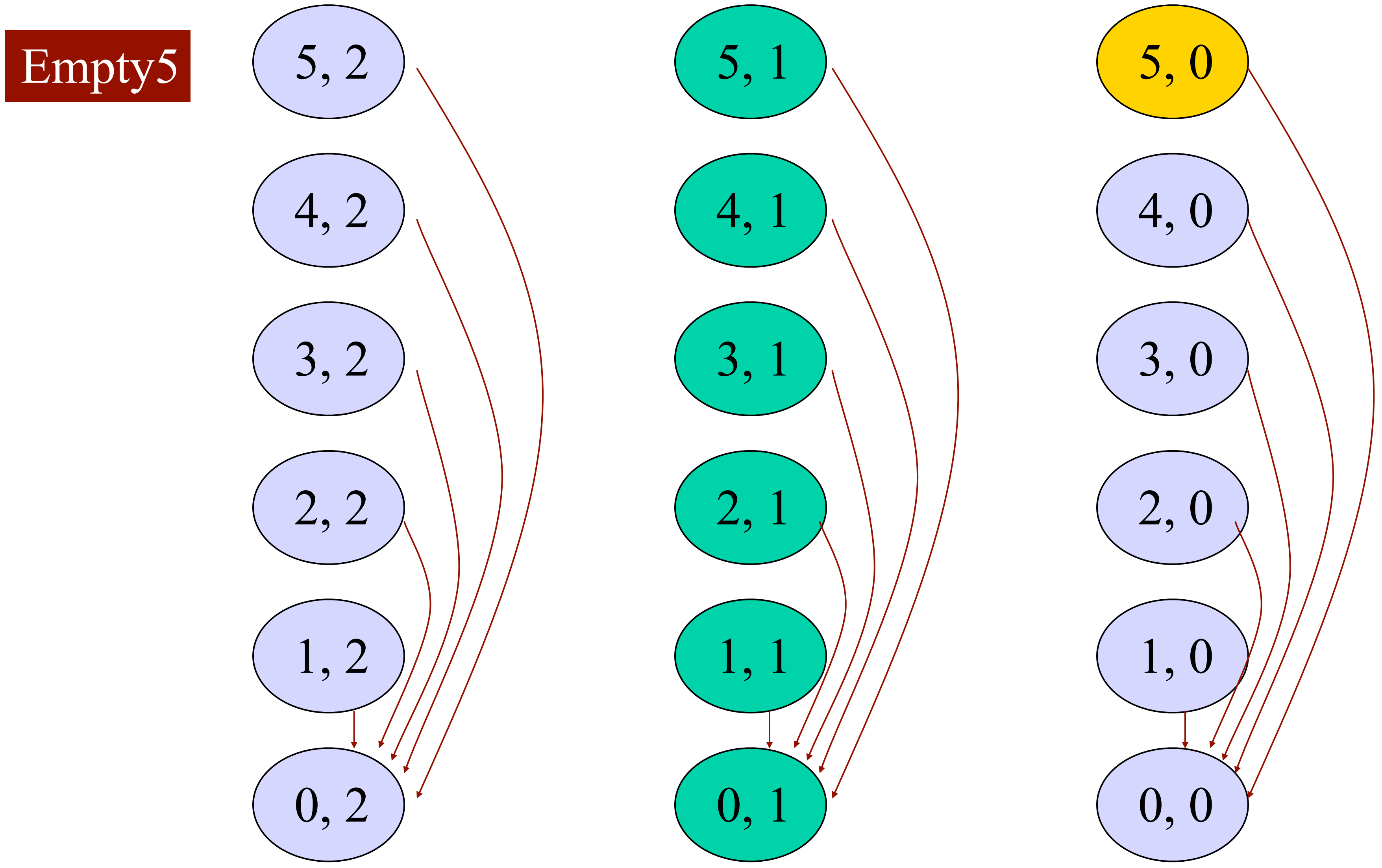
Action table

Name	Cond.	Transition	Effect
Empty5	–	$(x,y)\rightarrow(0,y)$	Empty 5-gal. jug
Empty2	–	$(x,y)\rightarrow(x,0)$	Empty 2-gal. jug
2to5	$x \leq 3$	$(x,2)\rightarrow(x+2,0)$	Pour 2-gal. into 5-gal.
5to2	$x \geq 2$	$(x,0)\rightarrow(x-2,2)$	Pour 5-gal. into 2-gal.
5to2part	$y < 2$	$(1,y)\rightarrow(0,y+1)$	Pour partial 5-gal. into 2-gal.

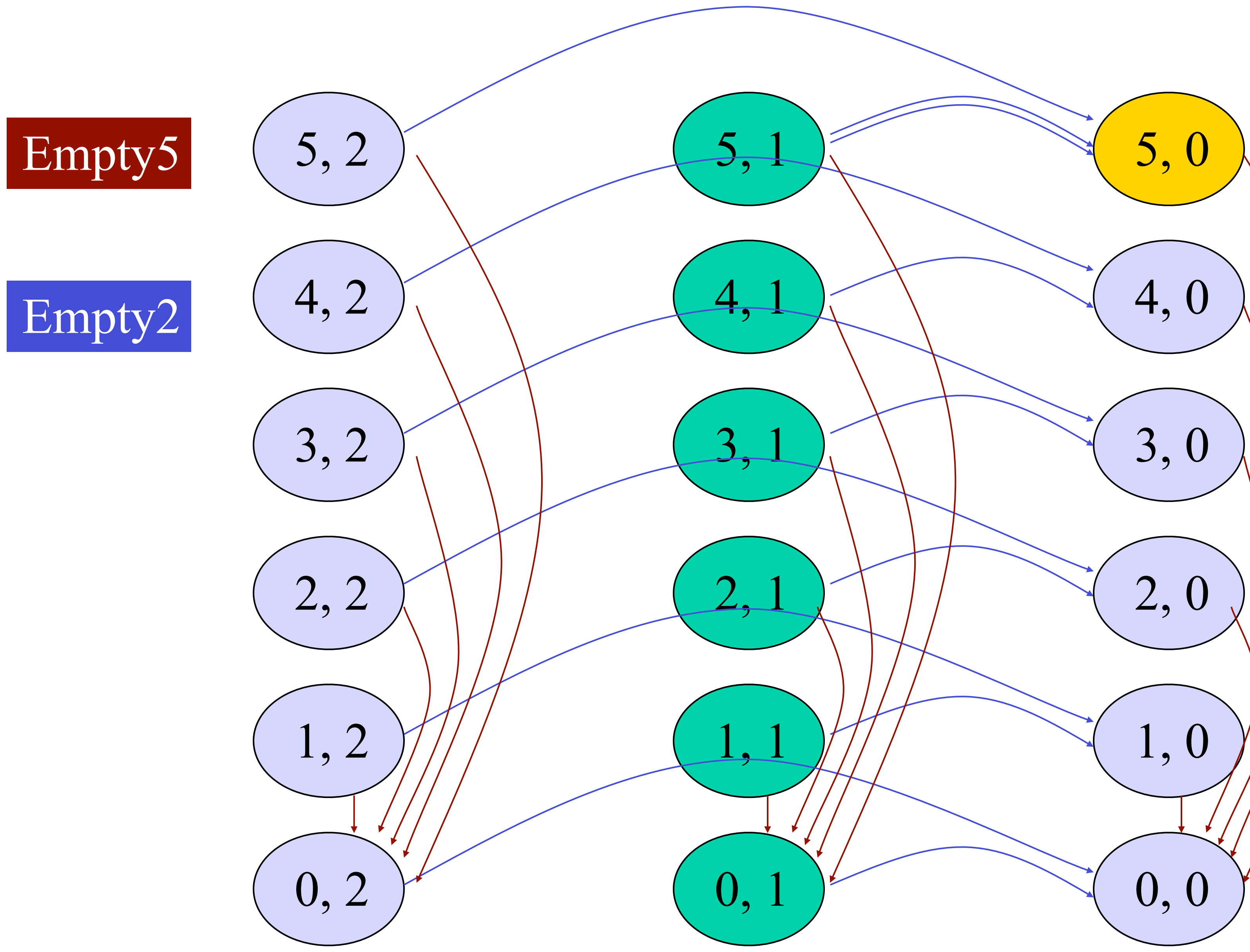
Water jug state space



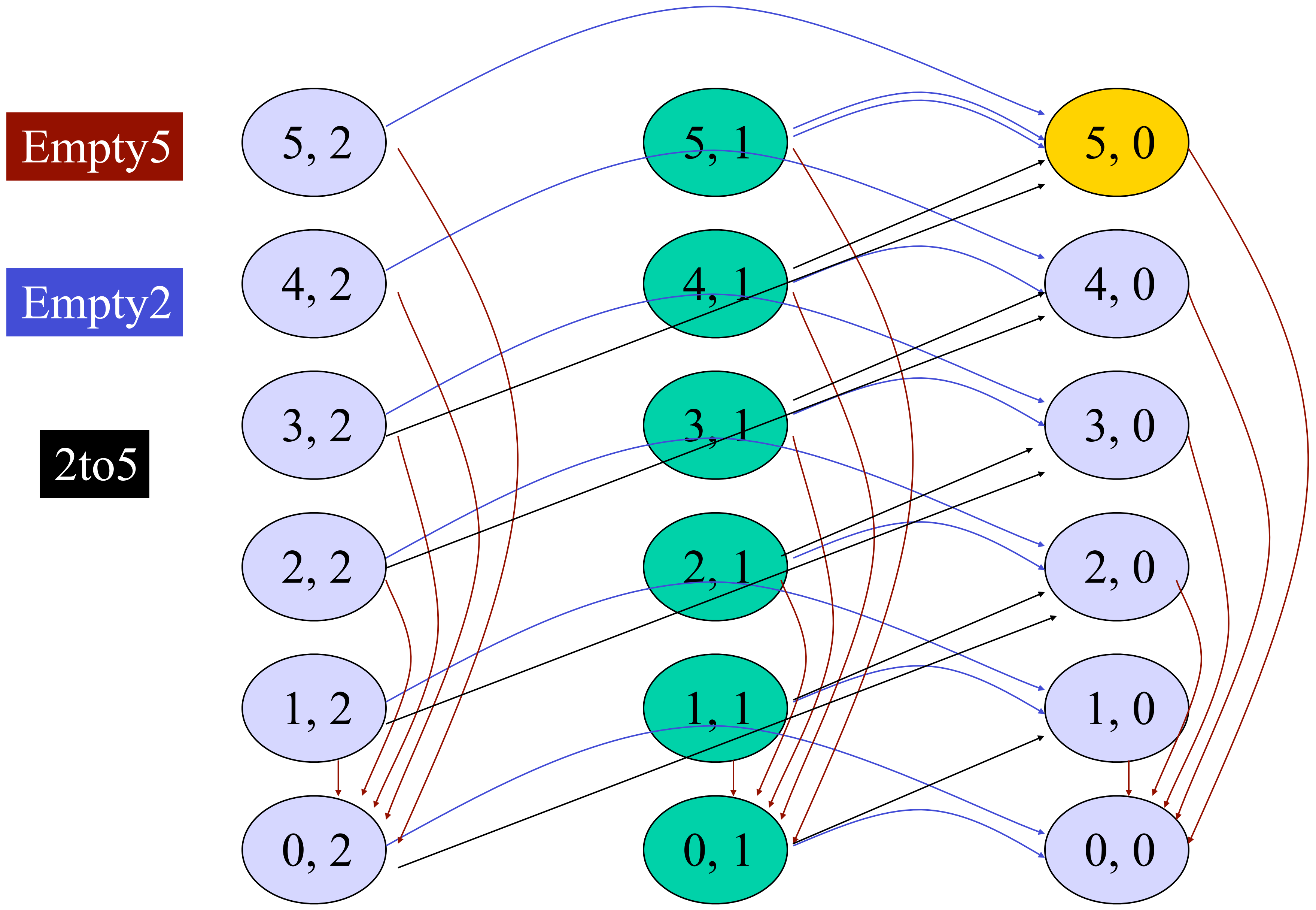
Water jug state space



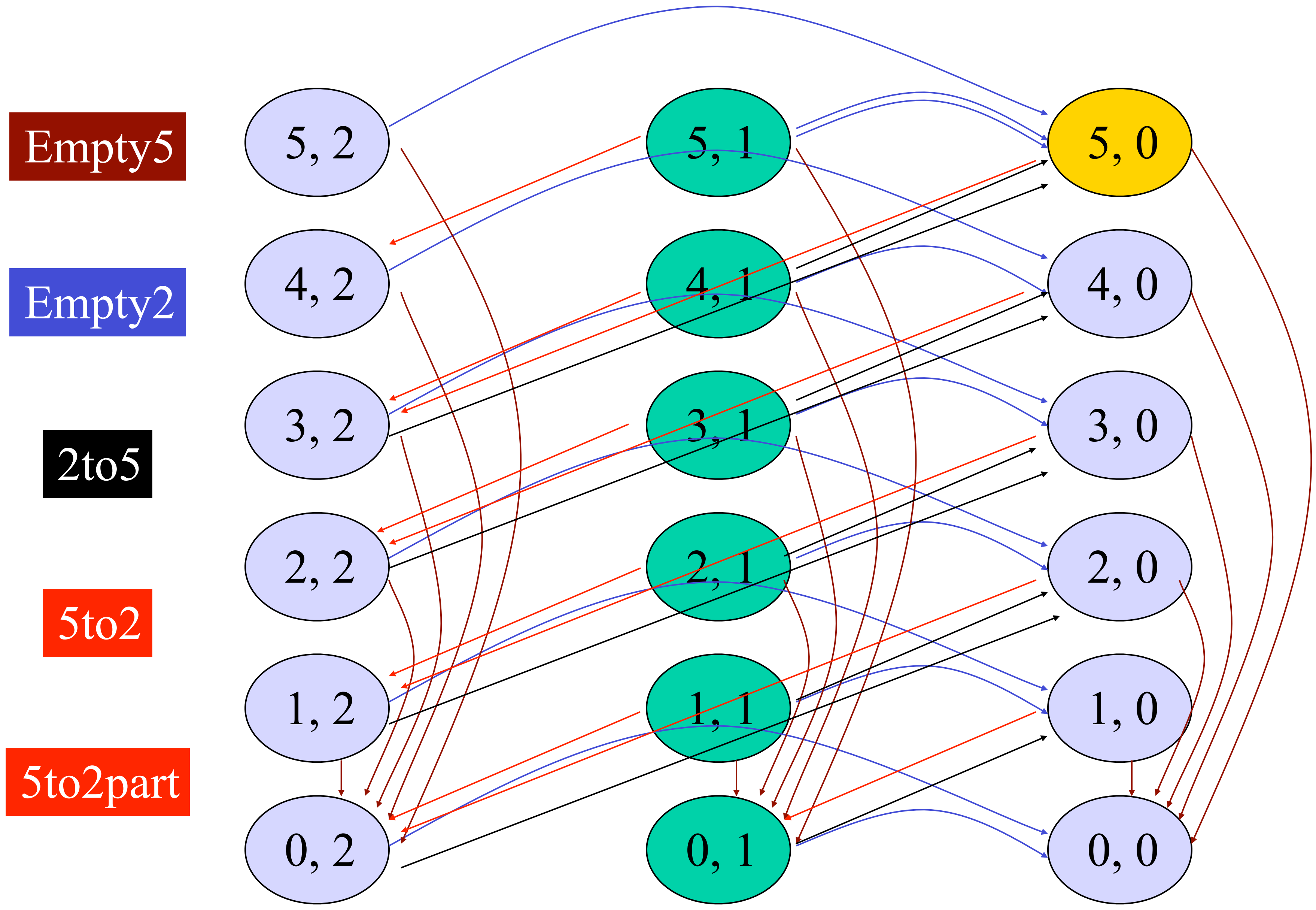
Water jug state space



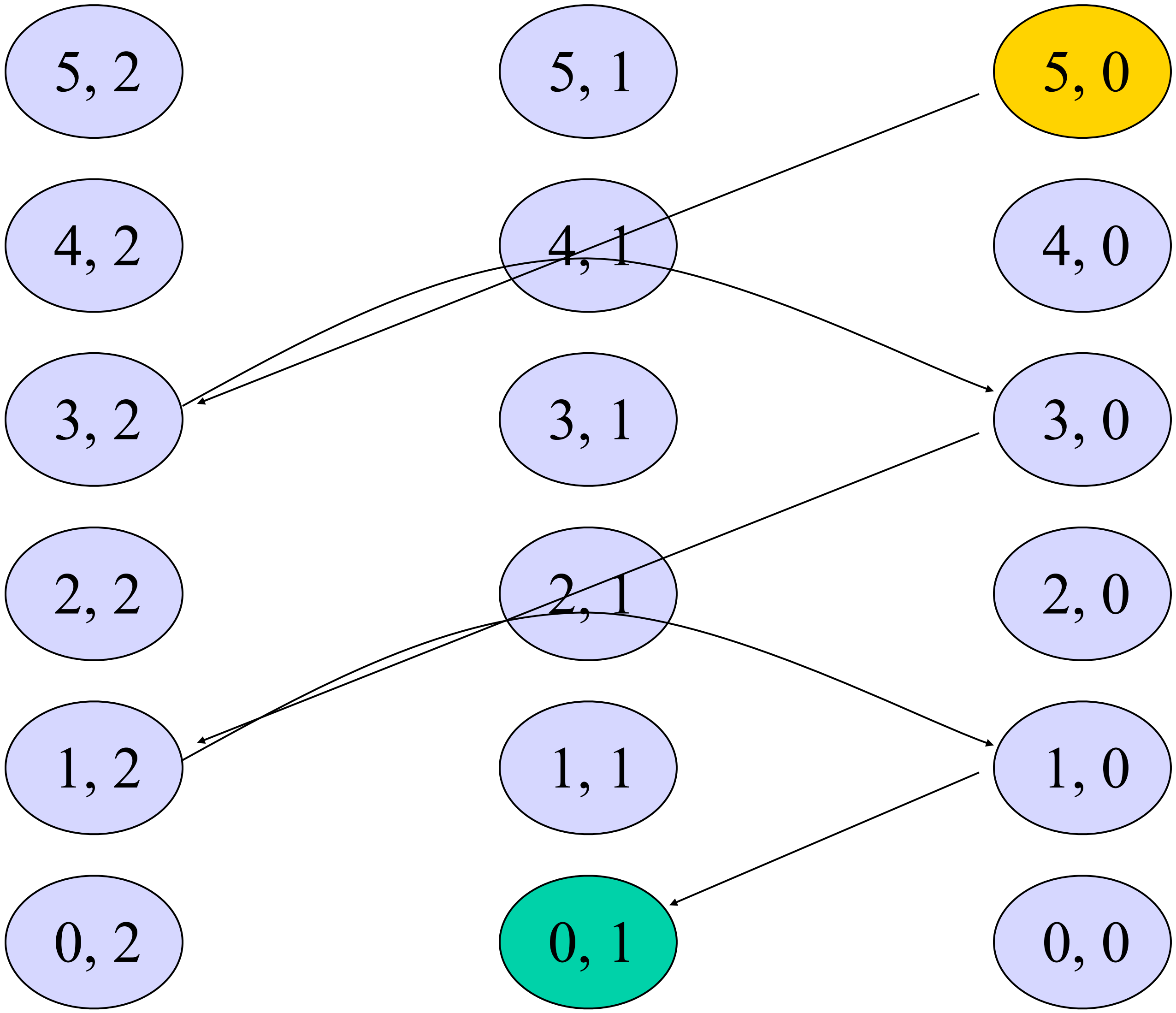
Water jug state space



Water jug state space



Water jug solution



Formalizing search III

- **State-space search** is the process of searching through a state space for a solution by **making explicit** a sufficient portion of an **implicit** state-space graph to find a goal node.
  - For large state spaces, it isn't practical to represent the whole space.
  - Initially V={S}, where S is the start node; when S is expanded, its successors are generated and those nodes are added to V and the associated arcs are added to E. This process continues until a goal node is found.

State-space search algorithm

```
function general-search (problem, QUEUEING-FUNCTION)
;; problem describes the start state, operators, goal test, and operator costs
;; queueing-function is a comparator function that ranks two states
;; general-search returns either a goal node or failure
nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
             problem.OPERATORS))
 end
  ;; Note: The goal test is NOT done when nodes are generated
  ;; Note: This algorithm does not detect loops
```

Key procedures to be defined

- EXPAND
  - Generate all successor nodes of a given node
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Used to maintain a ranked list of nodes that are candidates for expansion

Bookkeeping

- Typical node data structure includes:
  - State at this node
  - Parent node
  - Action applied to get to this node
  - Depth of this node (number of action applications since initial state)
  - Cost of the path (sum of each action application so far)

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space complexity**
  - How much space is used by the algorithm? Usually measured in terms of the maximum size of the "nodes" list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?