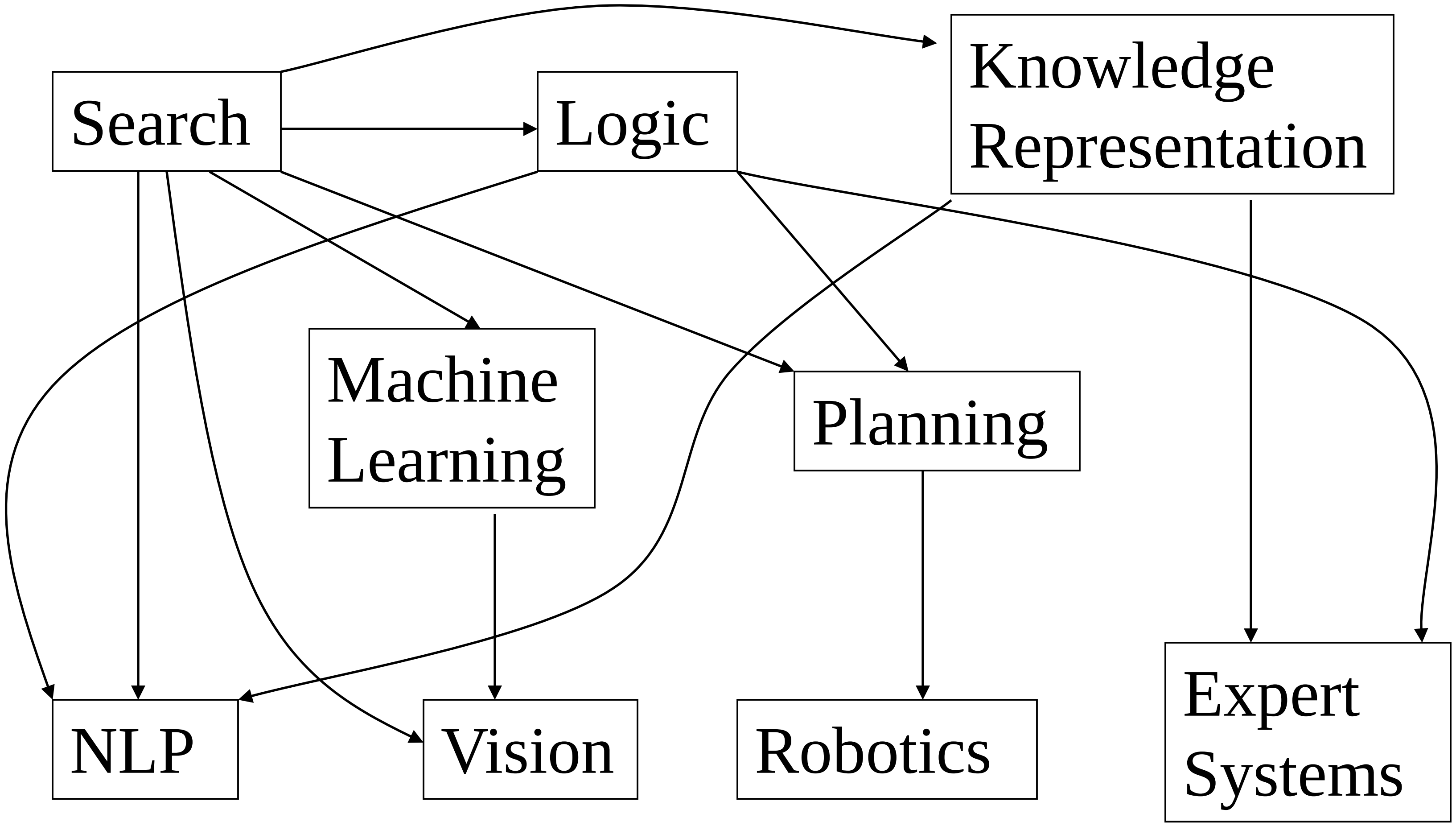
Artificial Intelligence

Areas of AI and Some Dependencies



What is Artificial Intelligence ?

- making computers that think?
- the automation of activities we associate with human thinking, like decision making, learning … ?
- the art of creating machines that perform functions that require intelligence when performed by people ?
- the study of mental faculties through the use of computational models ?

What is Artificial Intelligence ?

- the study of computations that make it possible to perceive, reason and act ?
- a field of study that seeks to explain and emulate intelligent behaviour in terms of computational processes ?
- a branch of computer science that is concerned with the automation of intelligent behaviour ?
- anything in Computing Science that we don't yet know how to do properly ? (!)

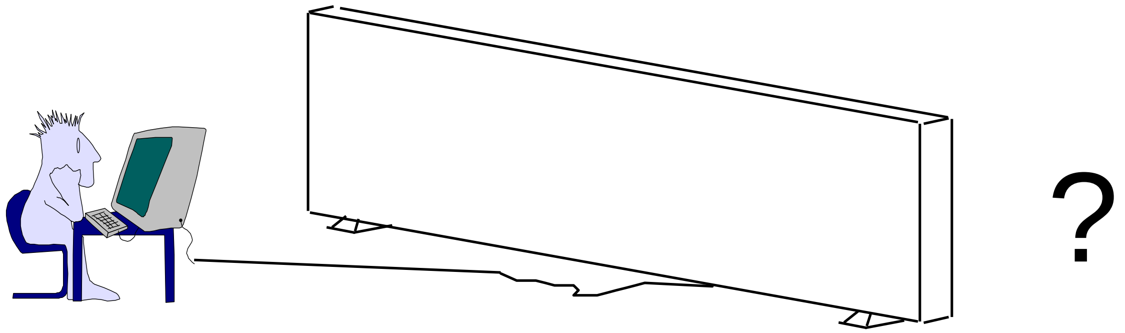
What is Artificial Intelligence ?

THOUGHT	**Systems that think like humans**	**Systems that think rationally**
BEHAVIOUR	**Systems that act like humans**	**Systems that act rationally**
	HUMAN	**RATIONAL**

Systems that act like humans: Turing Test

- “The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil)
- “The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight)

Systems that act like humans



- You enter a room which has a computer terminal. You have a fixed period of time to type what you want into the terminal, and study the replies. At the other end of the line is either a human being or a computer system.
- If it is a computer system, and at the end of the period you cannot reliably determine whether it is a system or a human, then the system is deemed to be intelligent.

Systems that act like humans

- The Turing Test approach
  - a human questioner cannot tell if
    - there is a computer or a human answering his question, via teletype (remote communication)
  - The computer must behave intelligently
- Intelligent behavior
  - to achieve human-level performance in all cognitive tasks

Systems that act like humans

- These cognitive tasks include:
  - *Natural language processing*
    - for communication with human
  - *Knowledge representation*
    - to store information effectively & efficiently
  - *Automated reasoning*
    - to retrieve & answer questions using the stored information
  - *Machine learning*
    - to adapt to new circumstances

The total Turing Test

- Includes two more issues:
  - *Computer vision*
    - to perceive objects (seeing)
  - *Robotics*
    - to move objects (acting)

What is Artificial Intelligence ?

THOUGHT	**Systems that think like humans**	**Systems that think rationally**
BEHAVIOUR	**Systems that act like humans**	**Systems that act rationally**
	HUMAN	**RATIONAL**

Systems that think like humans: cognitive modeling

- Humans as observed from 'inside'
- How do we know how humans think?
  - Introspection vs. psychological experiments
- Cognitive Science
- "The exciting new effort to make computers think … machines with *minds* in the full and literal sense" (Haugeland)
- "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning …" (Bellman)

What is Artificial Intelligence ?

	HUMAN	RATIONAL
THOUGHT	Systems that think like humans	Systems that think rationally
BEHAVIOUR	Systems that act like humans	Systems that act rationally

Systems that think ‘rationally’ "laws of thought"

- Humans are not always ‘rational’
- Rational - defined in terms of logic?
- Logic can’t express everything (e.g. uncertainty)
- Logical approach is often not feasible in terms of computation time (needs ‘guidance’)
- “The study of mental facilities through the use of computational models” (Charniak and McDermott)
- “The study of the computations that make it possible to perceive, reason, and act” (Winston)

What is Artificial Intelligence ?

THOUGHT	**Systems that think like humans**	**Systems that think rationally**
BEHAVIOUR	**Systems that act like humans**	**Systems that act rationally**
	HUMAN	**RATIONAL**

Systems that act rationally: “Rational agent”

- Rational behavior: doing the right thing
- The right thing: that which is expected to maximize goal achievement, given the available information
- Giving answers to questions is ‘acting’.
- I don't care whether a system:
  - replicates human thought processes
  - makes the same decisions as humans
  - uses purely logical reasoning

Systems that act rationally

- Logic → only *part* of a rational agent, not *all* of rationality
  - Sometimes logic cannot reason a correct conclusion
  - At that time, some *specific (in domain) human knowledge* or information is used
- Thus, it covers more generally different situations of problems
  - Compensate the incorrectly reasoned conclusion

Systems that act rationally

- Study AI as rational agent –

2 advantages:

  - It is more general than using logic only
    - Because: LOGIC + Domain knowledge
  - It allows extension of the approach with more scientific methodologies

Rational agents

- An agent is an entity that perceives and acts
- This course is about designing rational agents
- Abstractly, an agent is a function from percept histories to actions:
-

$$[f: P^* \rightarrow A]$$

- For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance
- Caveat: computational limitations make perfect rationality unachievable
  - → design best program for given machine resources
  -

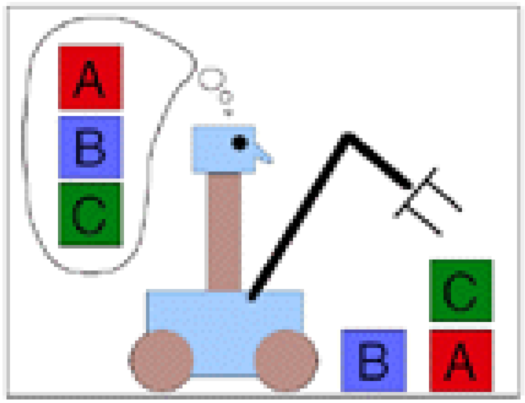
- Artificial
  - Produced by human art or effort, rather than originating naturally.
- Intelligence
- is the ability to acquire knowledge and use it" [Pigford and Baur]
- **So AI was defined as:**
  - AI is the study of ideas that enable computers to be intelligent.
  - AI is the part of computer science concerned with design of computer systems that exhibit human intelligence(From the Concise Oxford Dictionary)

From the above two definitions, we can see that AI has two major roles:

- Study the intelligent part concerned with humans.
- Represent those actions using computers.

Goals of AI

- To make computers more useful by letting them take over dangerous or tedious tasks from human
- Understand principles of human intelligence



The Foundation of AI

- ***Philosophy***
  - At that time, the study of human intelligence began with no formal expression
  - Initiate the idea of mind as a machine and its internal operations

The Foundation of AI

- Mathematics formalizes the three main area of AI: *computation*, *logic*, and *probability*
  - Computation leads to analysis of the problems that can be computed
    - *complexity theory*
  - Probability contributes the "*degree of belief*" to handle *uncertainty* in AI
  - *Decision theory* combines *probability theory* and *utility theory* (bias)

The Foundation of AI

- Psychology
  - How do humans think and act?
  - The study of human reasoning and acting
  - Provides reasoning models for AI
  - Strengthen the ideas
    - humans and other animals can be considered as information processing machines

The Foundation of AI

- Computer Engineering
  - How to build an efficient computer?
  - Provides the artifact that makes AI application possible
  - The power of computer makes computation of large and difficult problems more easily
  - AI has also contributed its own work to computer science, including: time-sharing, the linked list data type, OOP, etc.

The Foundation of AI

- **Control theory and Cybernetics**
  - How can artifacts operate under their own control?
  - The artifacts adjust their actions
    - To do better for the environment over time
    - Based on an objective function and feedback from the environment
  - Not limited only to linear systems but also other problems
    - as language, vision, and planning, etc.

The Foundation of AI

- Linguistics
  - For understanding natural languages
    - different approaches has been adopted from the linguistic work
  - Formal languages
  - Syntactic and semantic analysis
  - Knowledge representation

The main topics in AI

Artificial intelligence can be considered under a number of headings:

- Search (includes Game Playing).
- Representing Knowledge and Reasoning with it.
- Planning.
- Learning.
- Natural language processing.
- Expert Systems.
- Interacting with the Environment
  (e.g. Vision, Speech recognition, Robotics)

We won't have time in this course to consider all of these.

Some Advantages of Artificial Intelligence

- more powerful and more useful computers
- new and improved interfaces
- solving new problems
- better handling of information
- relieves information overload
- conversion of information into knowledge

The Disadvantages

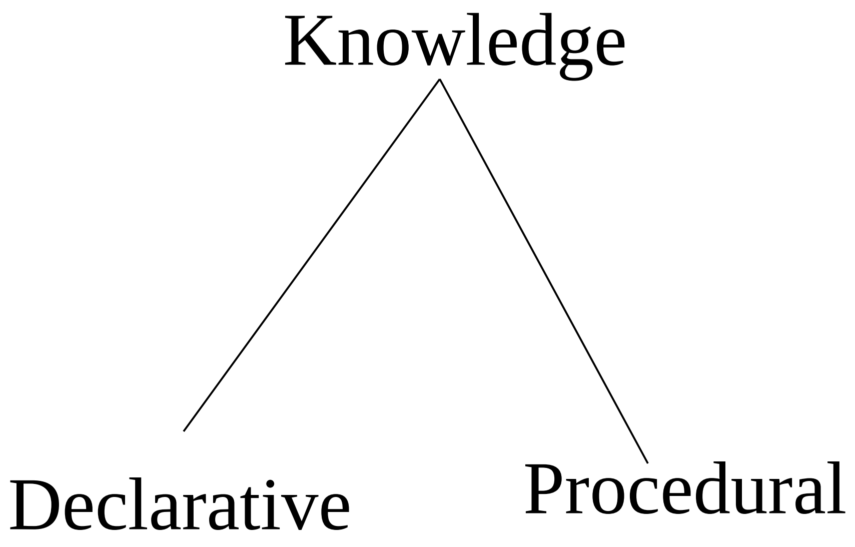
- increased costs
- difficulty with software development - slow and expensive
- few experienced programmers
- few practical products have reached the market as yet.

Search

- *Search* is the fundamental technique of AI.
  - Possible answers, decisions or courses of action are structured into an abstract space, which we then search.
- Search is either "blind" or “uninformed":
  - blind
    - we move through the space without worrying about what is coming next, but recognising the answer if we see it
  - informed
    - we guess what is ahead, and use that information to decide where to look next.
- We may want to search for the first answer that satisfies our goal, or we may want to keep searching until we find the best answer.

Knowledge Representation & Reasoning

- The <u>second</u> most important concept in AI
- If we are going to act rationally in our environment, then we must have some way of describing that environment and drawing inferences from that representation.
  - how do we describe what we know about the world ?
  - how do we describe it *concisely* ?
  - how do we describe it so that we can get hold of the right piece of knowledge when we need it ?
  - how do we generate new pieces of knowledge ?
  - how do we deal with *uncertain* knowledge ?



- Declarative knowledge deals with factoid questions (what is the capital of India? Etc.)
- Procedural knowledge deals with “How”
- Procedural knowledge can be embedded in declarative knowledge

Planning

Given a set of goals, construct a sequence of actions that achieves those goals:

- often very large search space
- but most parts of the world are independent of most other parts
- often start with goals and connect them to actions
- no necessary connection between order of planning and order of execution
- what happens if the world changes as we execute the plan and/or our actions don’t produce the expected results?

Learning

- If a system is going to act truly appropriately, then it must be able to change its actions in the light of experience:
  - how do we generate new facts from old ?
  - how do we generate new concepts ?
  - how do we learn to distinguish different situations in new environments ?

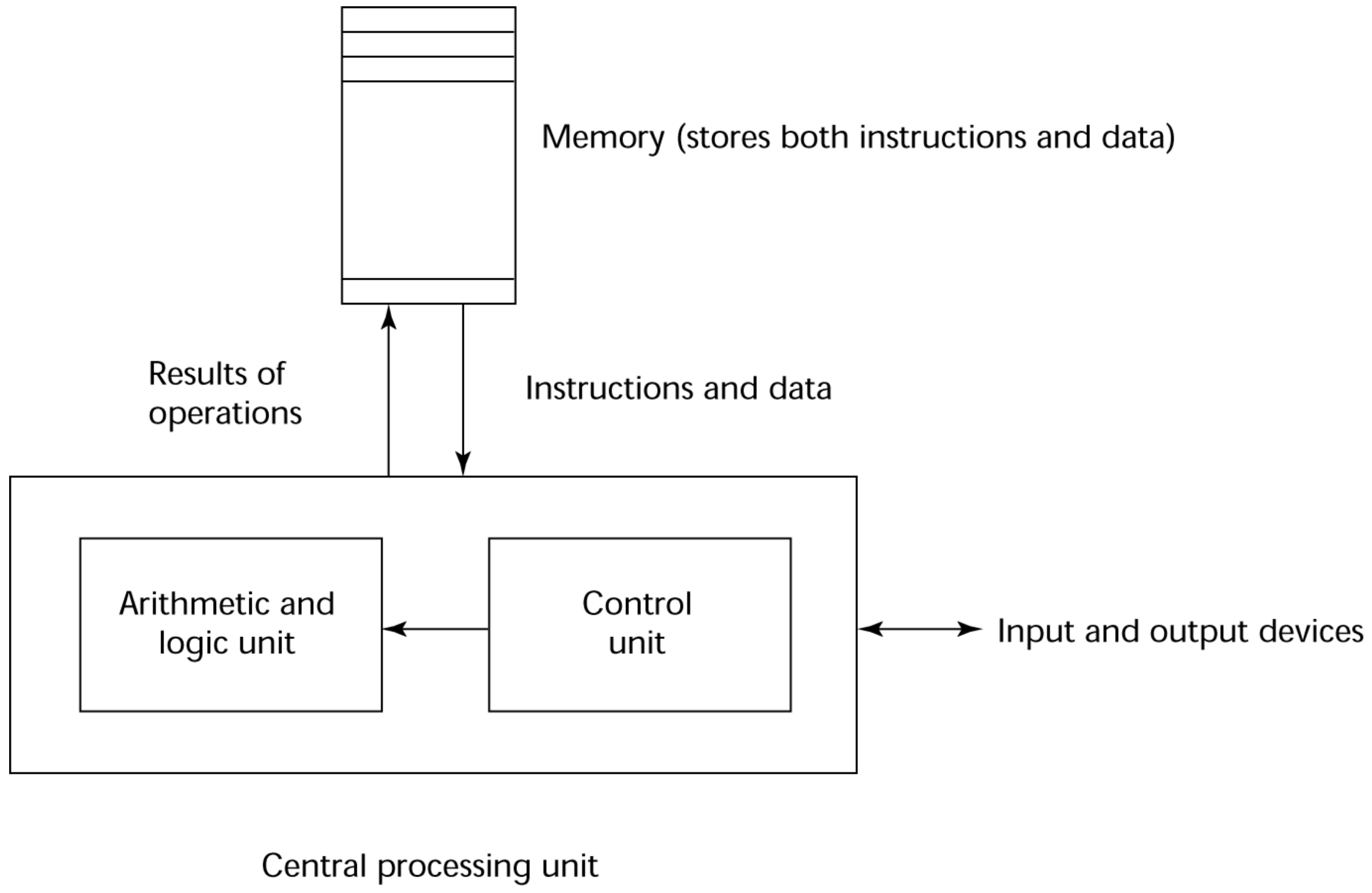
Interacting with the Environment

- In order to enable intelligent behaviour, we will have to interact with our environment.
- Properly intelligent systems may be expected to:
  - accept sensory input
    - vision, sound, …
  - interact with humans
    - understand language, recognise speech, generate text, speech and graphics, …
  - modify the environment
    - robotics

History of AI

- AI has a long history
  - Ancient Greece
    - Aristotle
  - Historical Figures Contributed
    - Ramon Lull
    - Al Khowarazmi
    - Leonardo da Vinci
    - David Hume
    - George Boole
    - Charles Babbage
    - John von Neuman
  - As old as electronic computers themselves (c1940)

The 'von Neuman' Architecture



History of AI

- Origins
  - The Dartmouth conference: 1956
    - John McCarthy (Stanford)
    - Marvin Minsky (MIT)
    - Herbert Simon (CMU)
    - Allen Newell (CMU)
    - Arthur Samuel (IBM)
- The Turing Test (1950)
- "Machines who Think"
  - By Pamela McCorckindale

Periods in AI

- Early period - 1950's & 60's
  - Game playing
    - brute force (calculate your way out)
  - Theorem proving
    - symbol manipulation
  - Biological models
    - neural nets
- Symbolic application period - 70's
  - Early expert systems, use of knowledge
- Commercial period - 80's
  - boom in knowledge/ rule bases

Periods in AI cont’d

- ? period - 90’s and New Millenium
- Real-world applications, modelling, better evidence, use of theory, ……?
- Topics: data mining, formal models, GA’s, fuzzy logic, agents, neural nets, autonomous systems
- Applications
  - visual recognition of traffic
  - medical diagnosis
  - directory enquiries
  - power plant control
  - automatic cars

Fashions in AI

Progress goes in stages, following funding booms and crises: Some examples:

1. Machine translation of languages

    1950's to 1966 - Syntactic translators

    1966 - all US funding cancelled

    1980 - commercial translators available

2. Neural Networks

    1943 - first AI work by McCulloch & Pitts

    1950's & 60's - Minsky's book on "Perceptrons" stops nearly all work on nets

    1986 - rediscovery of solutions leads to massive growth in neural nets research

The UK had its own funding freeze in 1973 when the Lighthill report reduced AI work severely - Lesson: Don't claim too much for your discipline!!!!

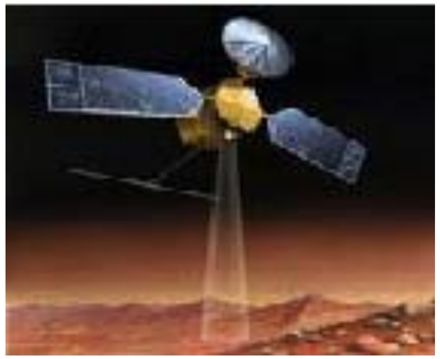
Look for similar stop/go effects in fields like genetic algorithms and evolutionary computing. This is a very active modern area dating back to the work of Friedberg in 1958.

Symbolic and Sub-symbolic AI

- Symbolic AI is concerned with describing and manipulating our knowledge of the world as explicit symbols, where these symbols have clear relationships to entities in the real world.
- Sub-symbolic AI (e.g. neural-nets) is more concerned with obtaining the correct response to an input stimulus without ‘looking inside the box’ to see if parts of the mechanism can be associated with discrete real world objects.
- This course is concerned with symbolic AI.

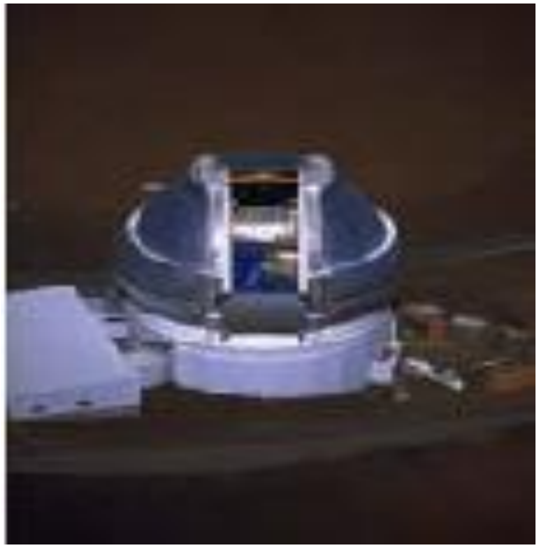
AI Applications

- Autonomous Planning & Scheduling:
  - Autonomous rovers.



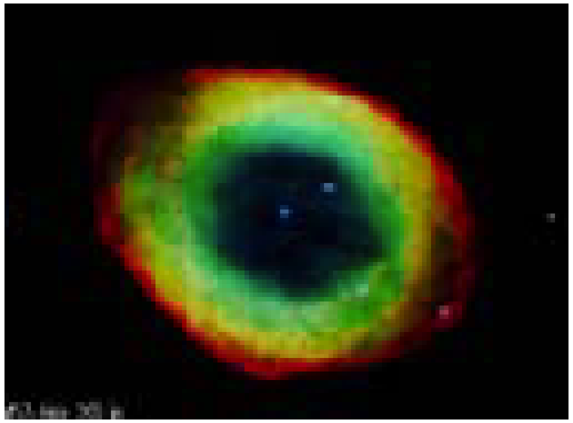
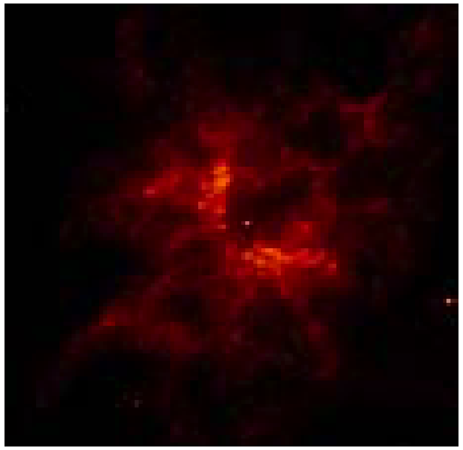
AI Applications

- Autonomous Planning & Scheduling:
  - Telescope scheduling



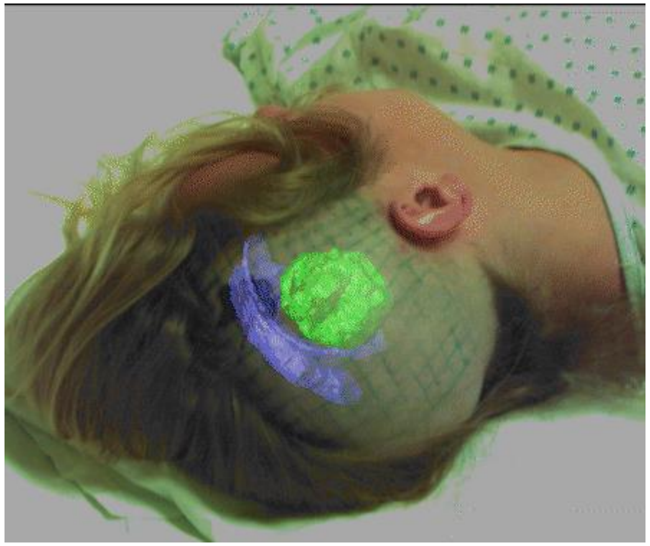
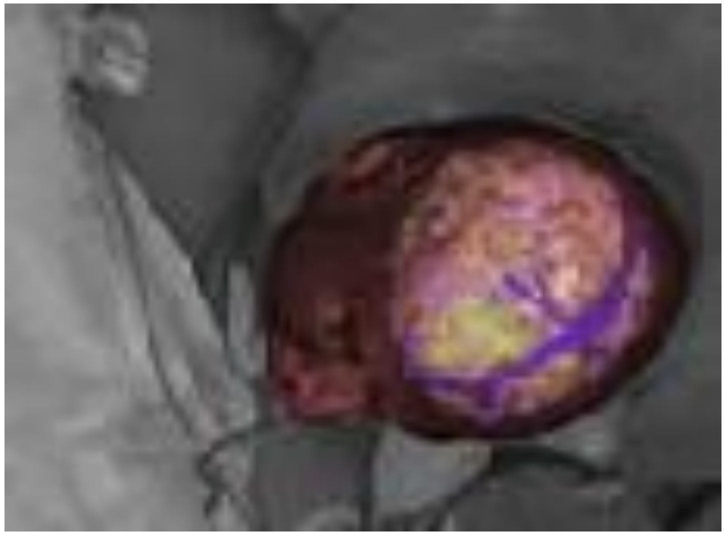
AI Applications

- Autonomous Planning & Scheduling:
  - Analysis of data:

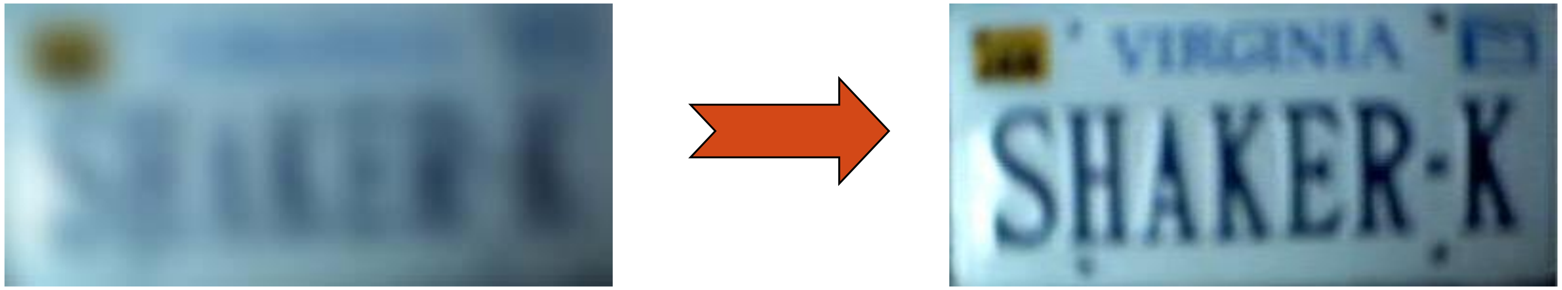
AI Applications

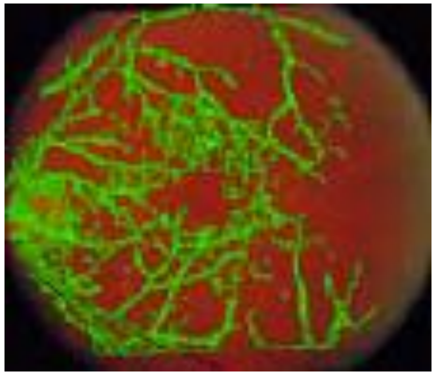
- **Medicine**:
  - Image guided surgery

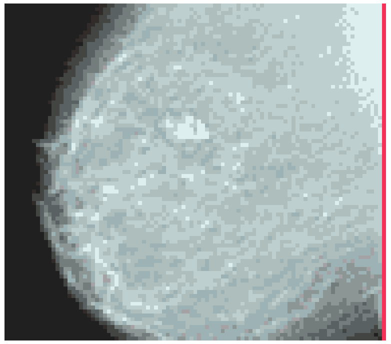



AI Applications

- **Medicine**:
  - Image analysis and enhancement







AI Applications

- **Transportation**:
  - **Autonomous vehicle control:**

AI Applications

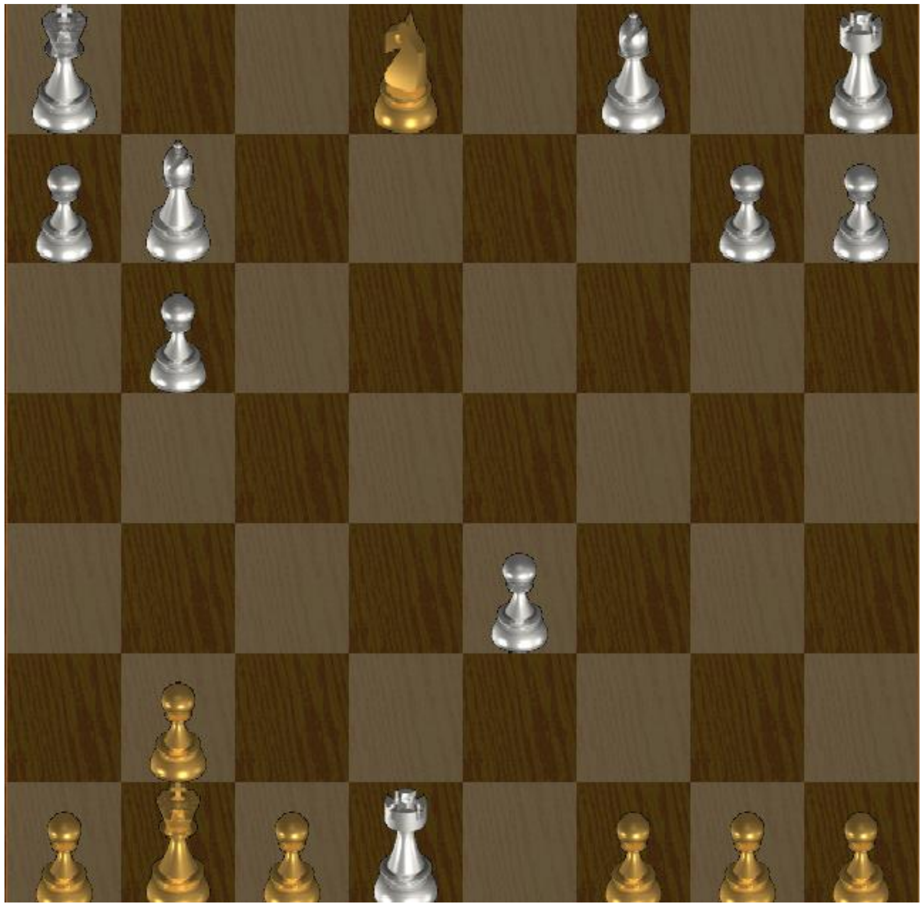
- **Transportation**:
  - **Pedestrian detection:**

AI Applications

Games:

AI Applications

- **Games**:



AI Applications

- **Robotic toys**:

AI Applications

Other application areas:

- **Bioinformatics:**
  - Gene expression data analysis
  - Prediction of protein structure
- **Text classification, document sortin**g:
  - Web pages, e-mails
  - Articles in the news
- **Video, image classification**
- **Music composition, picture drawing**
- **Natural Language Processing** .
- **Perception.**

Intelligent Agents

Chapter 2

PAGE

Must first specify the setting for intelligent agent design

Consider, e.g., the task of designing an automated taxi:

Percepts?? video, accelerometers, gauges, engine sensors, keyboard, GPS, . . .

Actions?? steer, accelerate, brake, horn, speak/display, . . .

Goals?? safety, reach destination, maximize profits, obey laws, passenger comfort, . . .

Environment?? US urban streets, freeways, traffic, pedestrians, weather, customers, . . .

Internet shopping agent

Percepts??

Actions??

Goals??

Environment??

Rational agents

Without loss of generality, "goals" specifiable by performance measure defining a numerical value for any environment history

Rational action: whichever action maximizes the expected value of the performance measure given the percept sequence to date

Rational $\neq$ omniscient
Rational $\neq$ clairvoyant
Rational $\neq$ successful

Environment types

	Solitaire	Backgammon	Internet shopping	Taxi
Accessible??	Yes	Yes	No	No
Deterministic??	Yes	No	Partly	No
Episodic??	No	No	No	No
Static??	Yes	Semi	Semi	No
Discrete??	Yes	Yes	Yes	No

The environment type largely determines the agent design

The real world is (of course) inaccessible, stochastic, sequential, dynamic, continuous

Agent functions and programs

An agent is completely specified by the agent function
mapping percept sequences to actions

(In principle, one can supply each possible sequence to see what it does.
Obviously, a lookup table would usually be immense.)

One agent function (or a small equivalence class) is rational

Aim: find a way to implement the rational agent function concisely

An agent program takes a single percept as input, keeps internal state:

```
function SKELETON-AGENT( percept) returns action
    static: memory, the agent's memory of the world

    memory ← UPDATE-MEMORY(memory, percept)
    action ← CHOOSE-BEST-ACTION(memory)
    memory ← UPDATE-MEMORY(memory, action)
    return action
```

AIMA code

The code for each topic is divided into four directories:

- `agents`: code defining agent types and programs
- `algorithms`: code for the methods used by the agent programs
- `environments`: code defining environment types, simulations
- `domains`: problem types and instances for input to algorithms

(Often run algorithms on domains rather than agents in environments.)

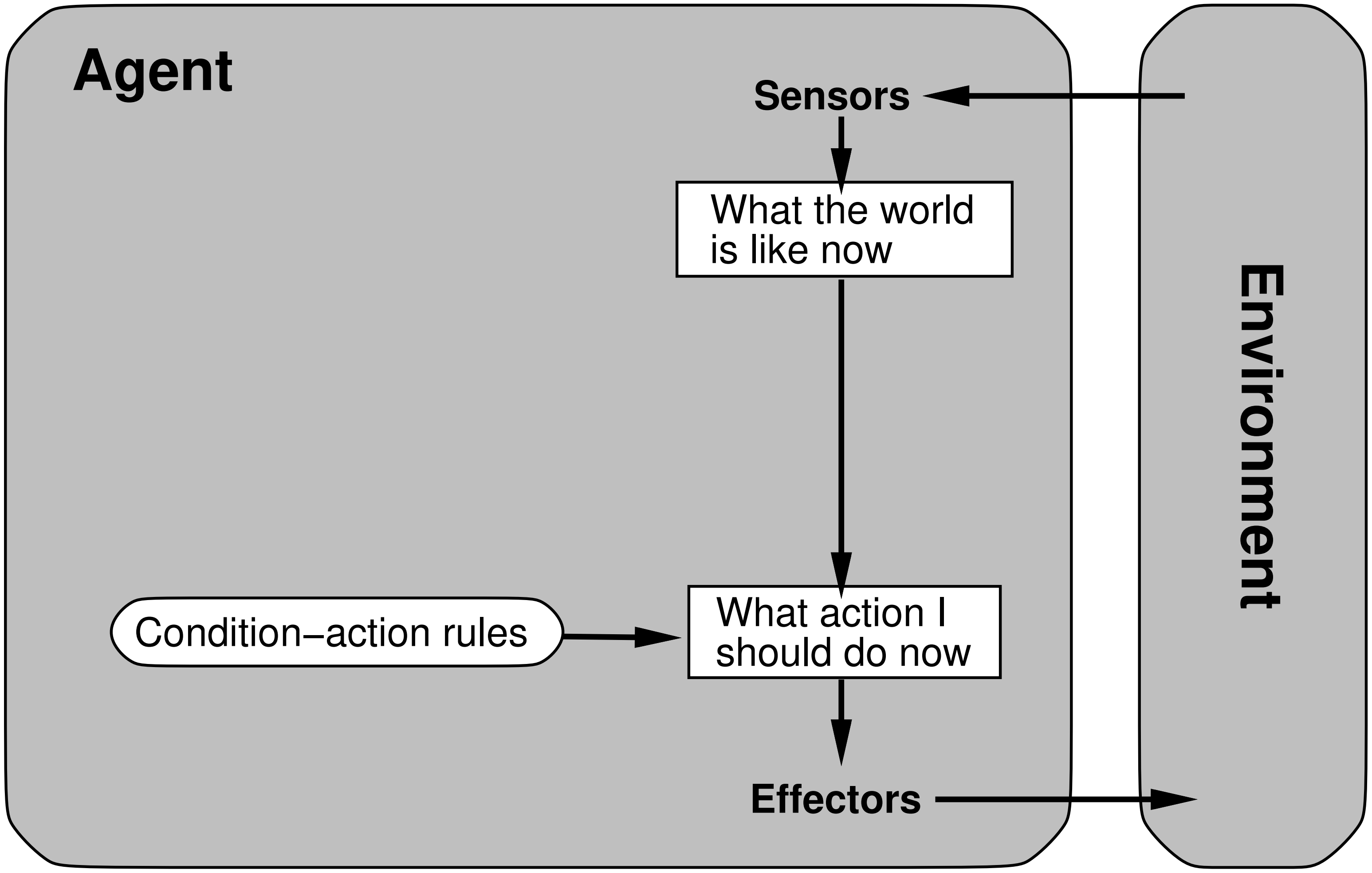
```
(setq joe (make-agent :name 'joe :body (make-agent-body)
                      :program (make-dumb-agent-program)))

(defun make-dumb-agent-program ()
  (let ((memory nil))
    #'(lambda (percept)
        (push percept memory)
        'no-op)))
```

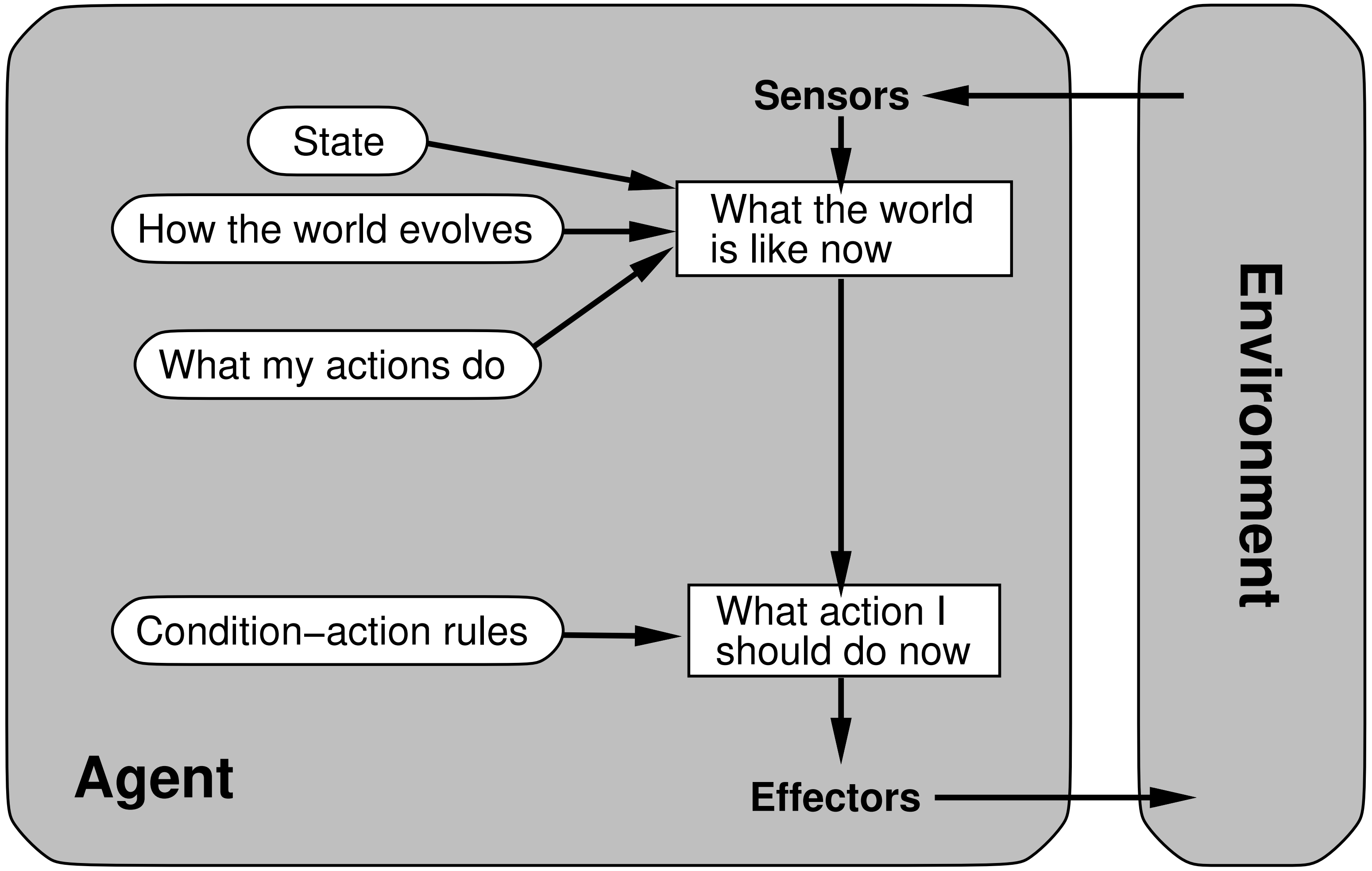
Agent types

Four basic types in order of increasing generality:
- simple reflex agents
- reflex agents with state
- goal-based agents
- utility-based agents

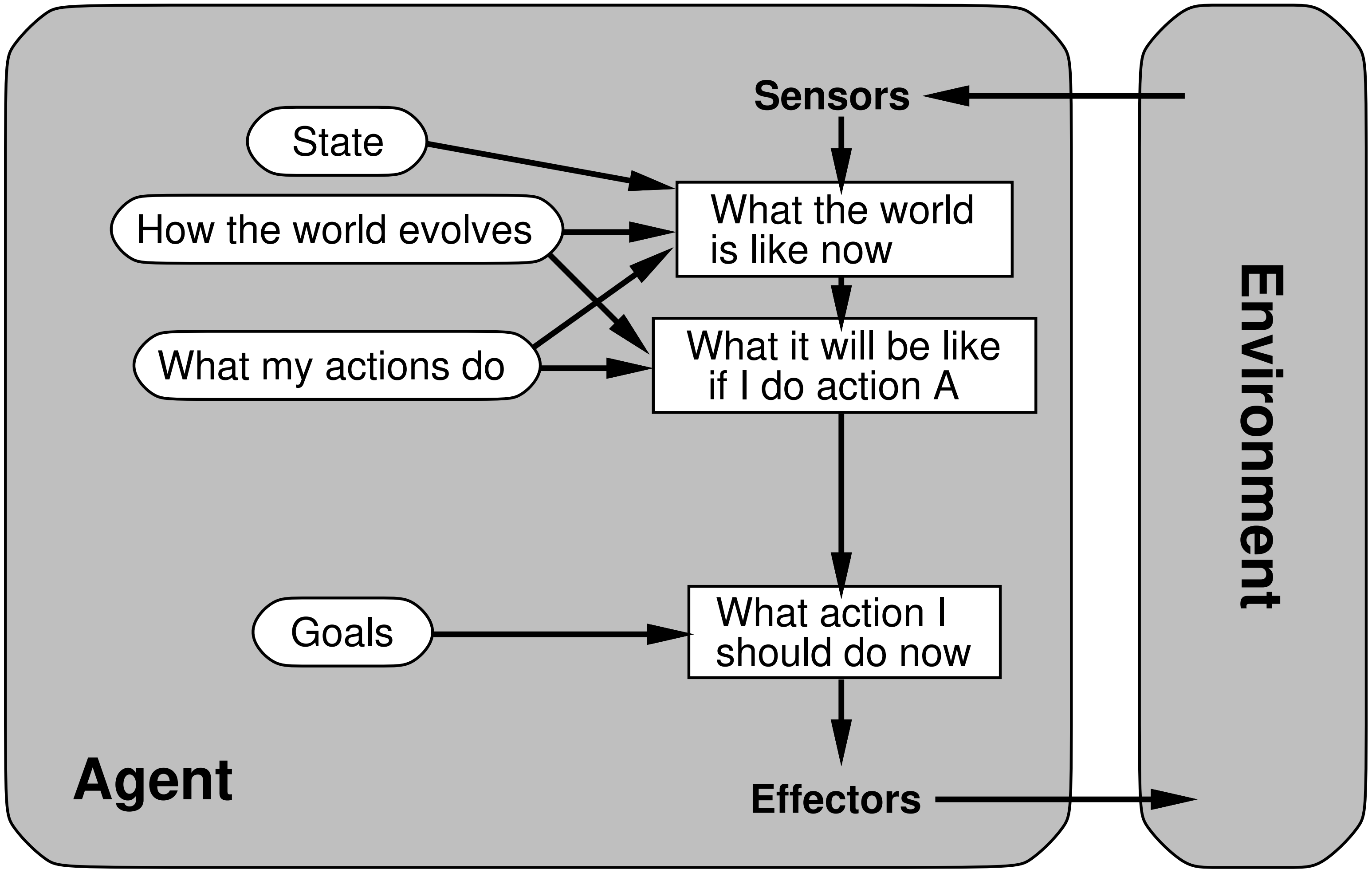
Simple reflex agents



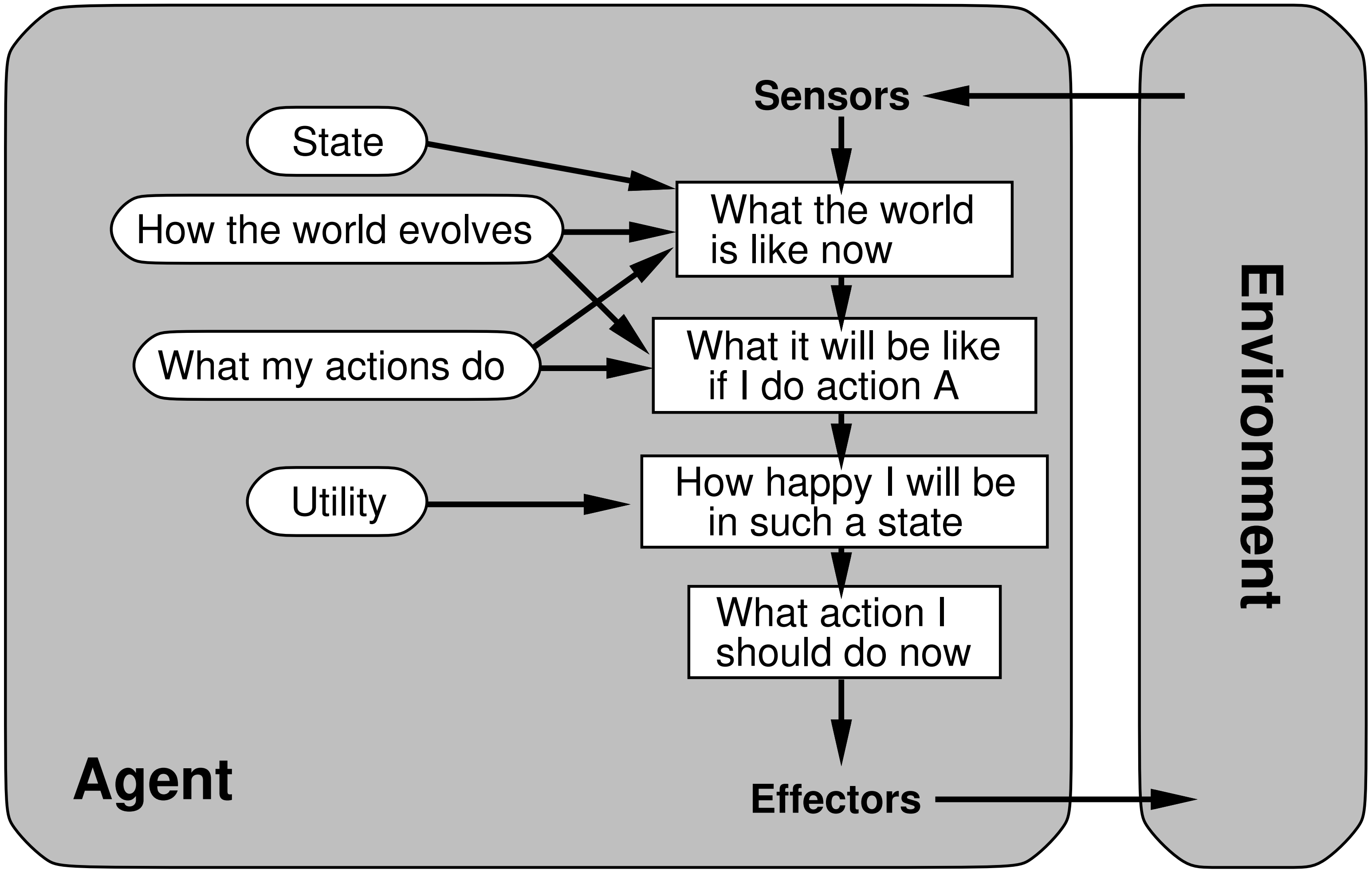
Reflex agents with state



Goal-based agents

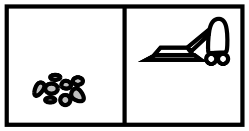

Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Goals
What action I should do now
Agent
Effectors
Environment

Utility-based agents


Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Agent
Effectors
Environment

The vacuum world

`code/agents/environments/vacuum.lisp`



Percepts (`<bump> <dirt> <home>`)

Actions `shutoff forward suck (turn left) (turn right)`

Goals (performance measure on environment history)
- +100 for each piece of dirt cleaned up
- -1 for each action
- -1000 for shutting off away from home

Environment
- grid, walls/obstacles, dirt distribution and creation, agent body
- movement actions work unless bump into wall
- suck actions put dirt into agent body (or not)

Accessible? Deterministic? Episodic? Static? Discrete?

Blind Search

Ref: Chapter 2

Review - State Space Problem Definition

- One widely used way to describe problems is by listing or describing all possible states.
- Solving a problem means moving through the state space from a start state to a goal state.
- Need to devise a set of operators that move from one state to another.

State Space Problem Definition

- Set of possible states.
- Set of possible operations that change the state.
- Specification of a starting state(s).
- Specification of a goal state(s).

State Space

- It is usually not possible to *list* all possible states:
  - use abstractions to describe legal states.
  - It may be easier to describe the illegal states.
  - Sometimes it is useful to provide a general description of the state space and a set of constraints.

Operations

- The problem solving system moves from one state to another according to well defined operations.
- Typically these operations are described as *rules*.
- A control system decides which rules are applicable at any state, and resolves conflicts and/or ambiguities.

Example: Water Jug Problem

Unlimited Water

Goal: 2 Gallons in the 4 Gal. Jug

Water Jug State Space

- The state can be represented by 2 integers x and y:
- x = gallons in the 4 gallon jug
- y = gallons in the 3 gallon jug

State Space = (x,y)

such that $x \in \{0,1,2,3,4\}$, $y \in \{0,1,2,3\}$

Water Jug Start and Goal States

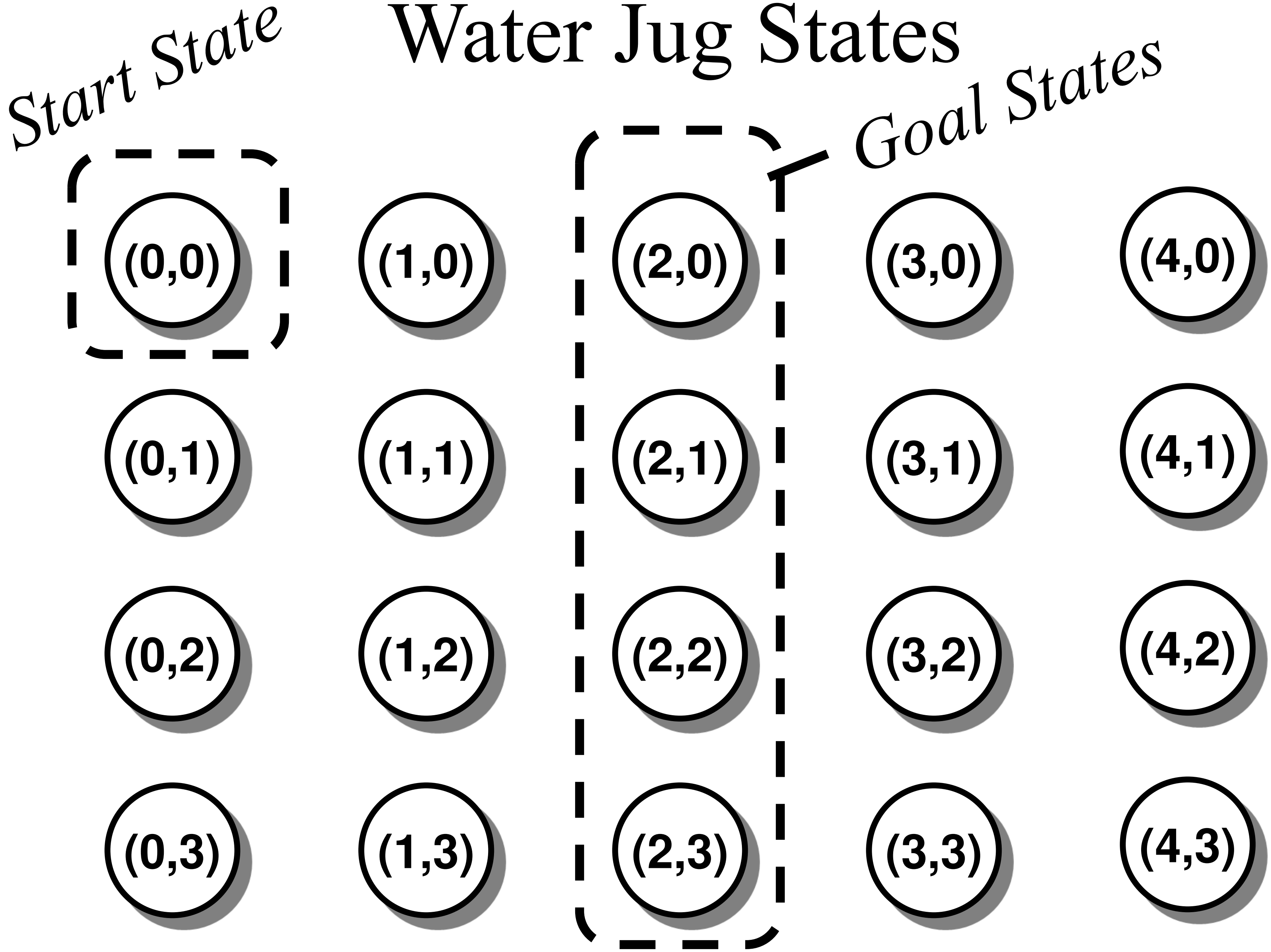
- The start state is when both jugs are empty:

  (0,0)

- The goal state is any state that has 2 gallons in the 4 gallon jug:

  (2,n) for any n

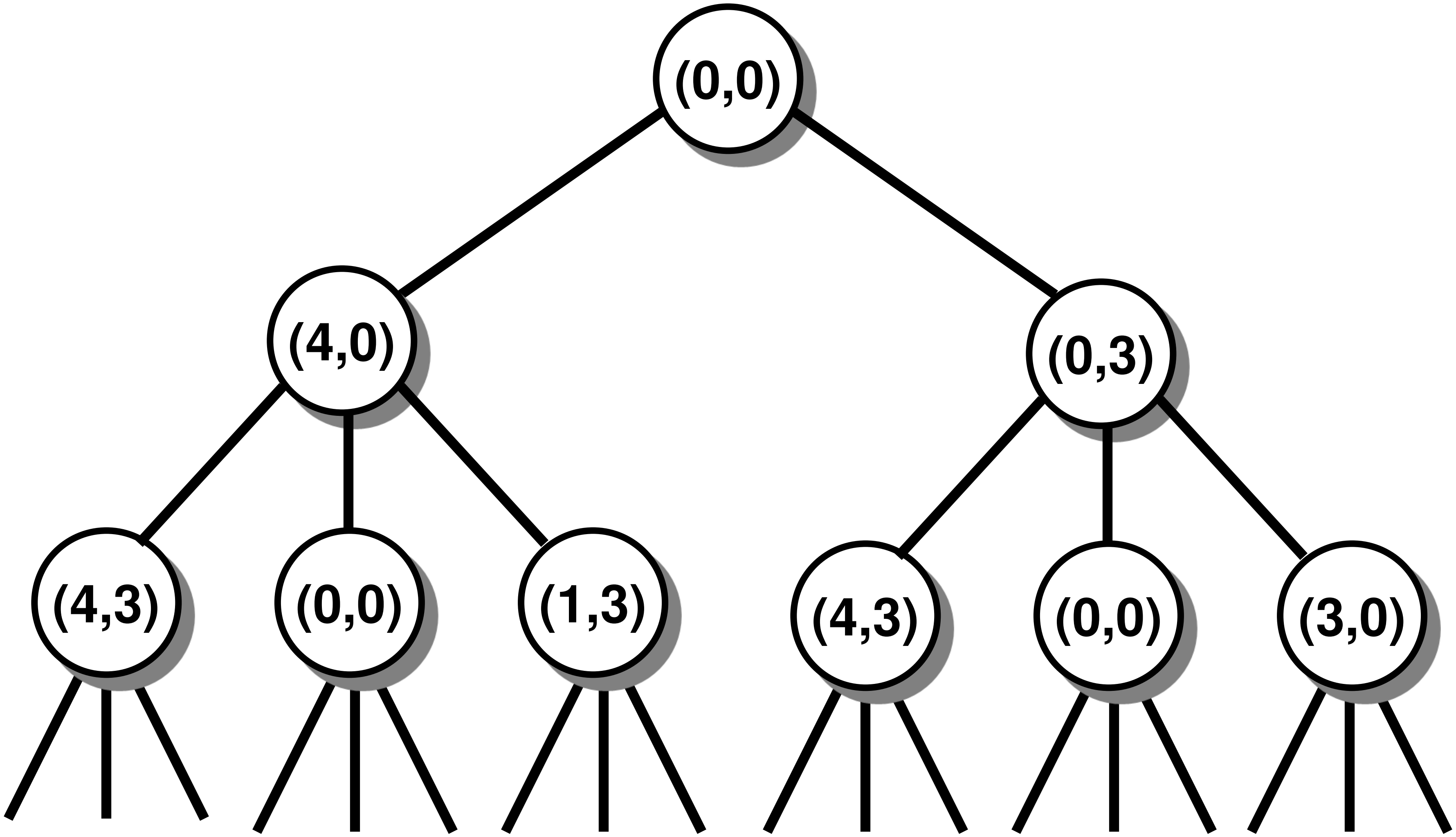
Water Jug States



Water Jug Operations

• Fill 3 gal. jug from pump	(x,y) -> (x,3)
• Fill 4 gal. jug from pump	(x,y) -> (4,y)
• Empty 3 gal. jug	(x,y) -> (x,0)
• Empty 4 gal. jug	(x,y) -> (0,y)
• Pour contents of 3 gal. jug into 4 gal. Jug	(x,y) -> (y+x mod 4,0)
• Others ...	

Water Jug Tree (partial)



Search Trees

- The search for a solution can be described by a tree - each node represents one state.
- The path from a parent node to a child node represents an operation.
- Search Trees provide a convenient description of the search space, they are not a data structure stored in memory!!!

Search Trees

- Each node in a search tree has child nodes that represent each of the states reachable by the parent state.
- Another way of looking at it: child nodes represent all available options once the search procedure reaches the parent node.
- There are a number of strategies for traversing a search tree.

Breadth-First-Search

- Breadth-First Search visits all nodes at depth *n* before visiting any nodes at depth *n+1*.
- General Algorithm for BFS:

```
create *nodelist* (a queue) and initialize to the start state.
repeat until a goal state is found or *nodelist* = {}
    remove the first element from nodelist:
        apply all possible rules and add resulting states to
          nodelist.
```

Breadth-First Search Characteristics

- If there is a solution, BFS will find it.
- BFS will find the minimal solution (shortest path length to the solution).
- BFS will not get caught in state space cycles.
- Requires space available to store the nodelist queue. This can be very large!!!

Depth-First Search

Depth-First processes all children (choices) of a node before considering any siblings (at the same depth).

Depth-First Search is similar to BFS, but instead of a queue we use a stack.

Depth-First Search can easily be described recursively.

DFS recursive definition

```
While applicable rules exist:
    apply the next rule and generate a new
      state.
    If new state is the goal - all done,
    Otherwise - do a depth-first search on the
      new state.
```

DFS

- General Algorithm for DFS:

```
create *nodelist* (a stack) and initialize to the start
   state.
repeat until a goal state is found or *nodelist* = {}
      pop the first element from nodelist:
            apply all possible rules and add (push)
               resulting states to nodelist.
```

Depth-First Search Characteristics

- Don’t need to keep track of a large list of states.
- May find a solution very fast (and might not).
- “Pruning” is possible
  - example: branch-and-bound
- Can easily get caught in loops.

Water Jug Problem Revisited

Rules

(x,y), $y<3$	->	(x,3)
(x,y), $x<4$	->	(4,y)
(x,y), $y\neq0$	->	(x,0)
(x,y), $x\neq0$	->	(0,y)
(0,y), $y\neq0$	->	(y,0)
(0,2)	->	(2,0)
(x,2)	->	(0,2)

Exercises:

1. Draw the BFS search

2. Draw the DFS search

Missionaries and Cannibals

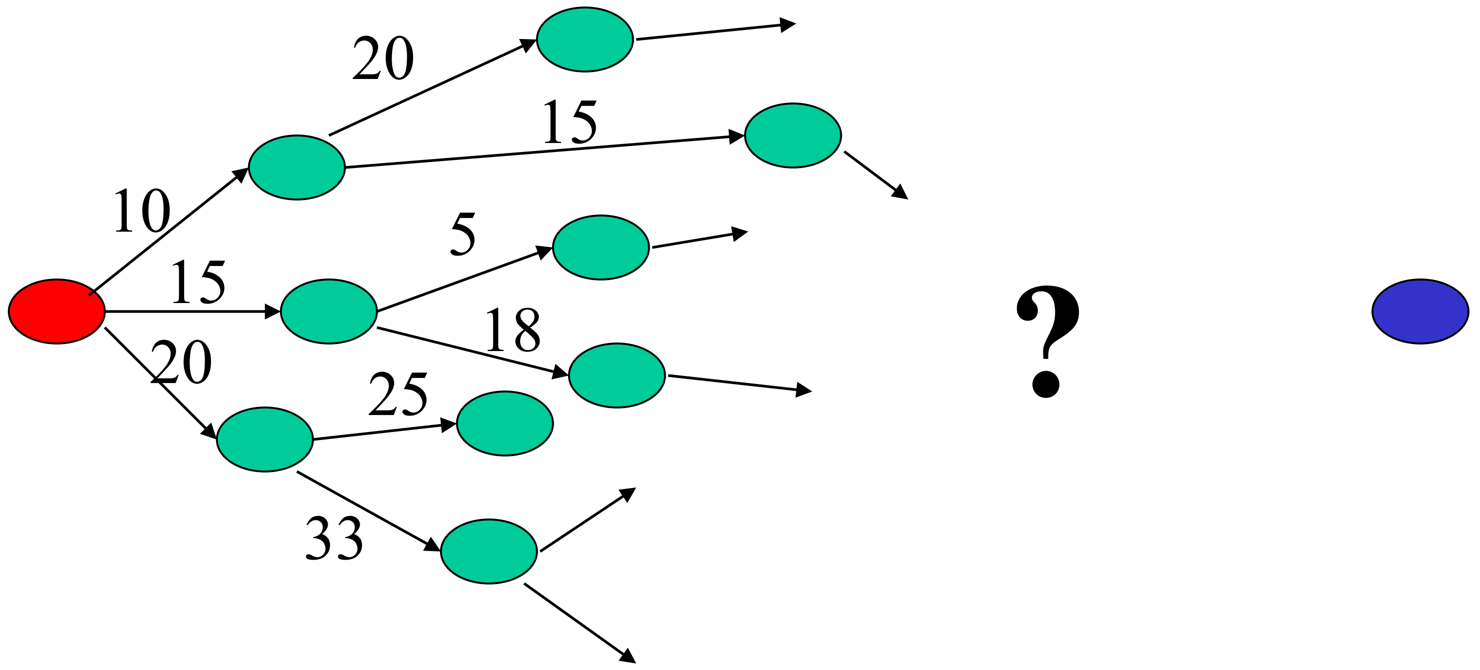
- 3 missionaries
- 3 cannibals
- 1 boat that holds (at most) 2 people.
- Want to get everyone across a river
- If cannibals outnumber the missionaries on either side of the river - ~~missionaries~~

The A* Algorithm

Héctor Muñoz-Avila

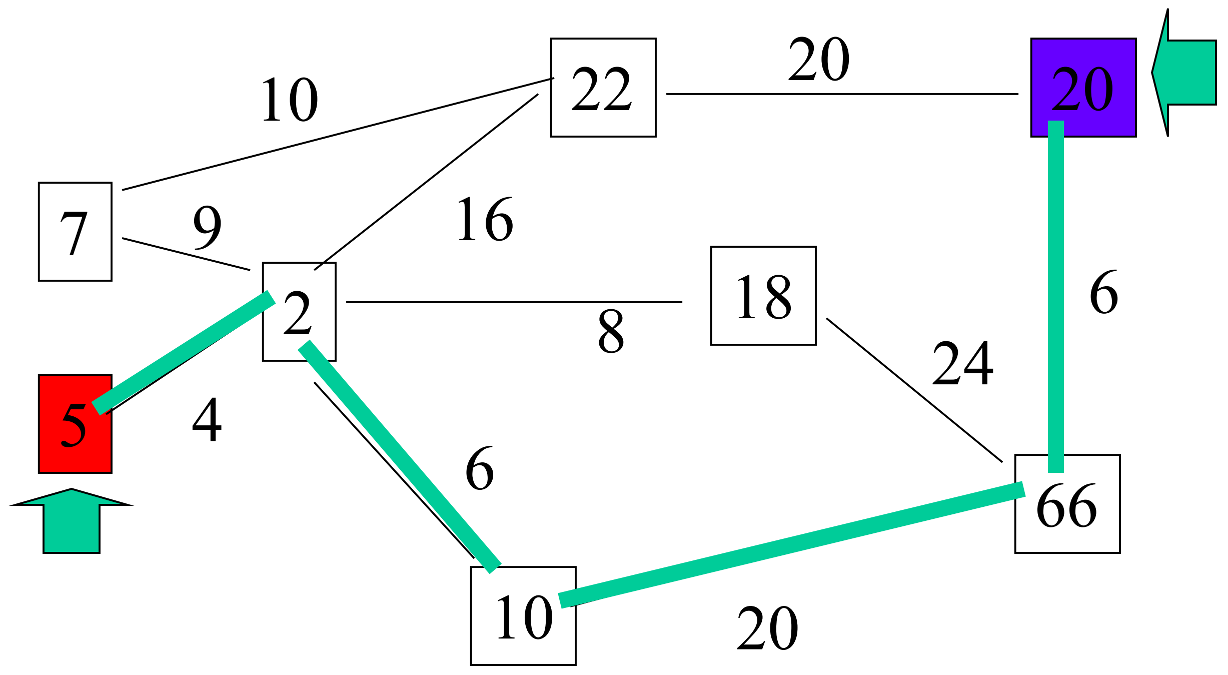
The Search Problem

Starting from a node n find the shortest path to a goal node g



Shortest Path

Consider the following weighted undirected graph:

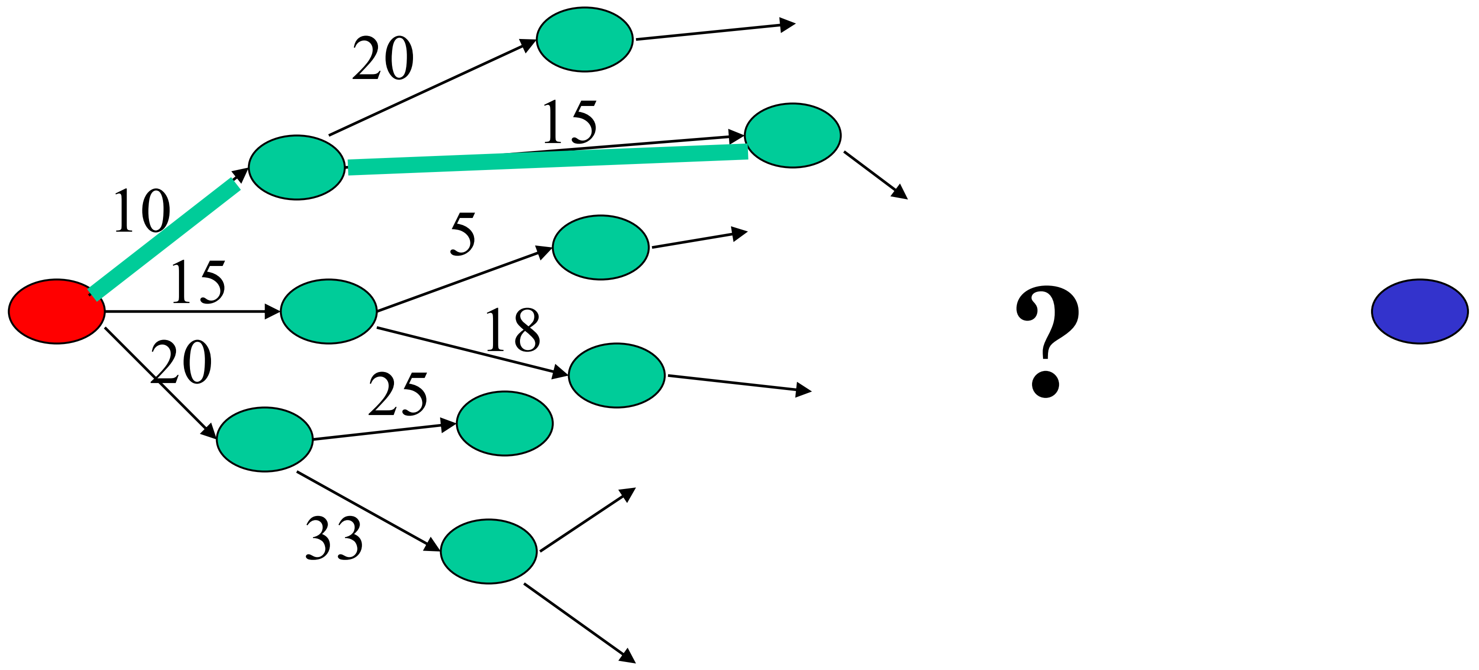


We want: A path 5 → v1 → v2 → … → 20

Such that g(20) = cost(5→v1) + cost(v1→ v2) + … + cost (→20) is minimum

Djikstra Algorithm

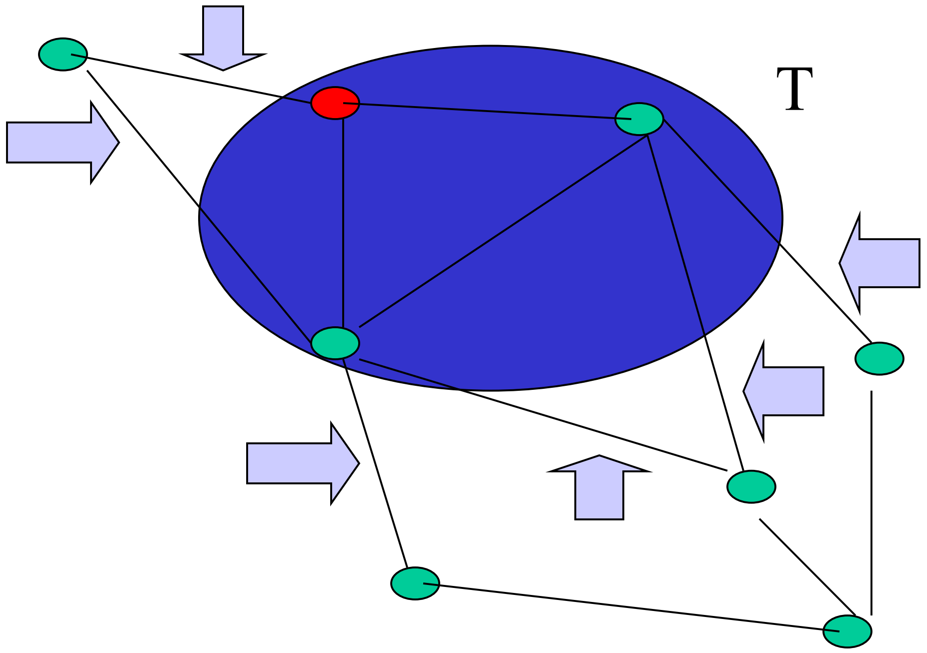
Greedy algorithm: from the candidate nodes select the one that has a path with minimum cost from the starting node



Djikstra Algorithm

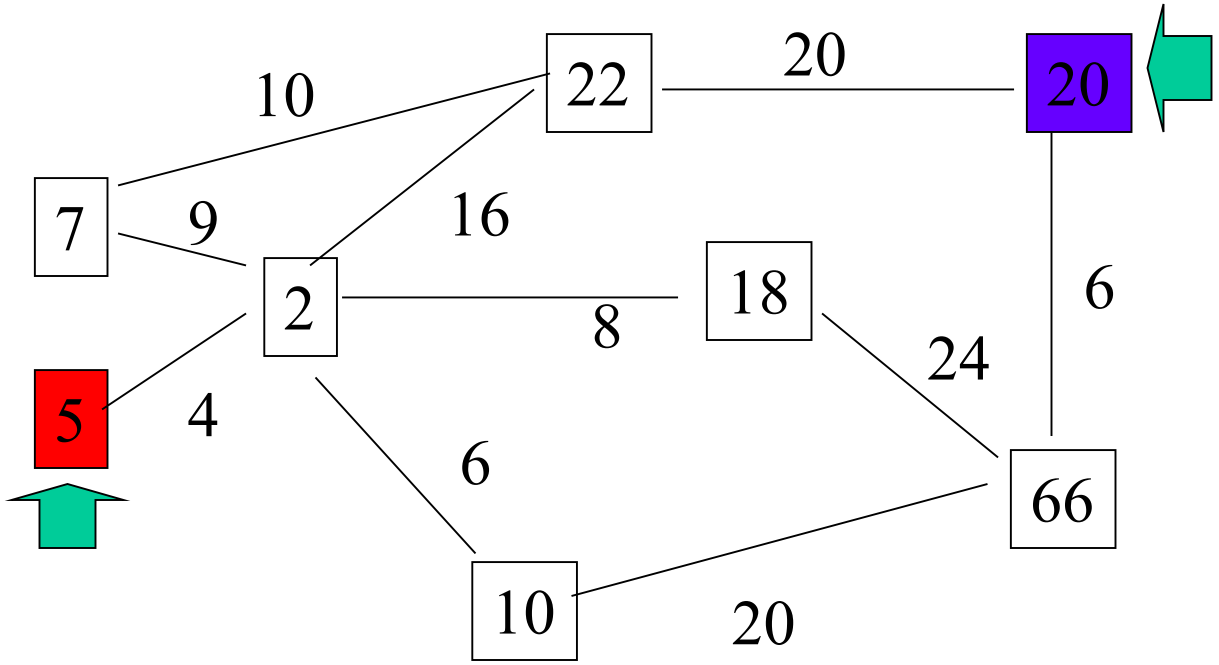
Given a Graph G = (V,E) and T a subset of V, the fringe of T, is defined as:

$$\text{Fringe}(T) = \{ (w,x) \text{ in } E : w \in T \text{ and } x \in V - T\}$$



Djikstra's algorithm pick the edge v in Fringe(T) that has minimum distance to the starting node 🔴 g(v) is minimum

Example

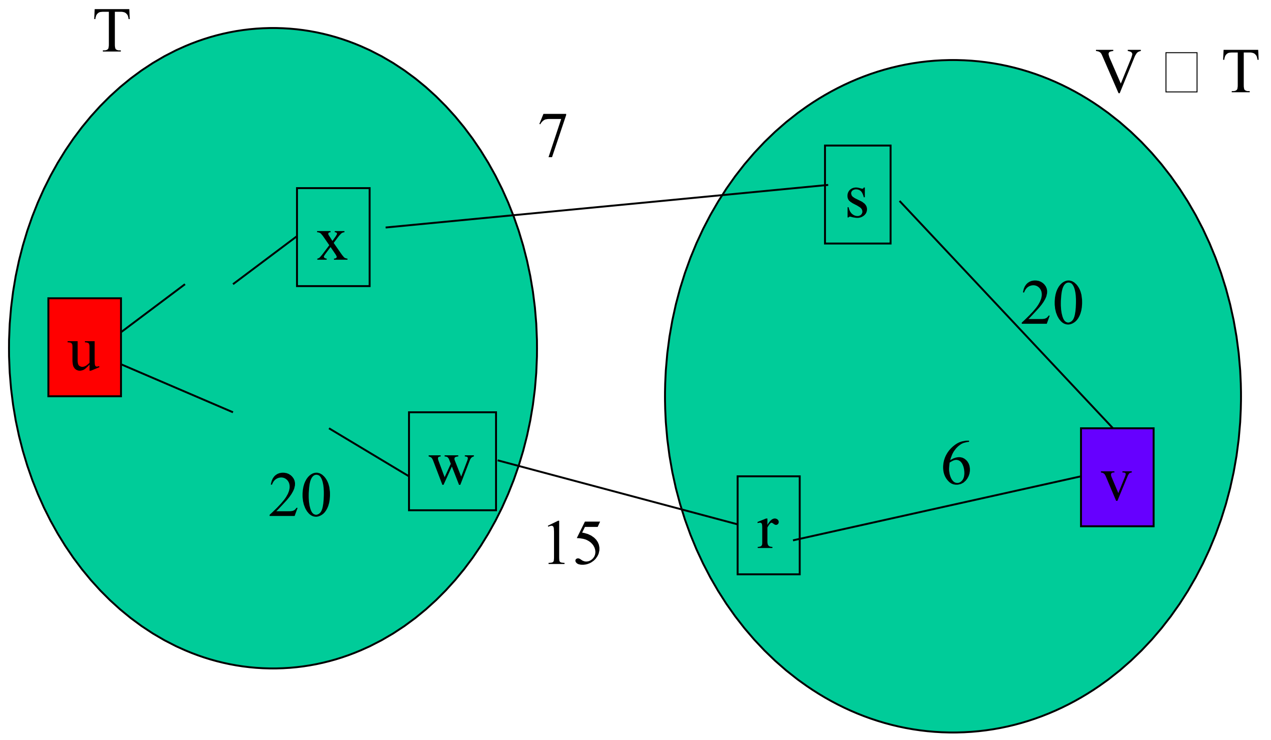


Properties

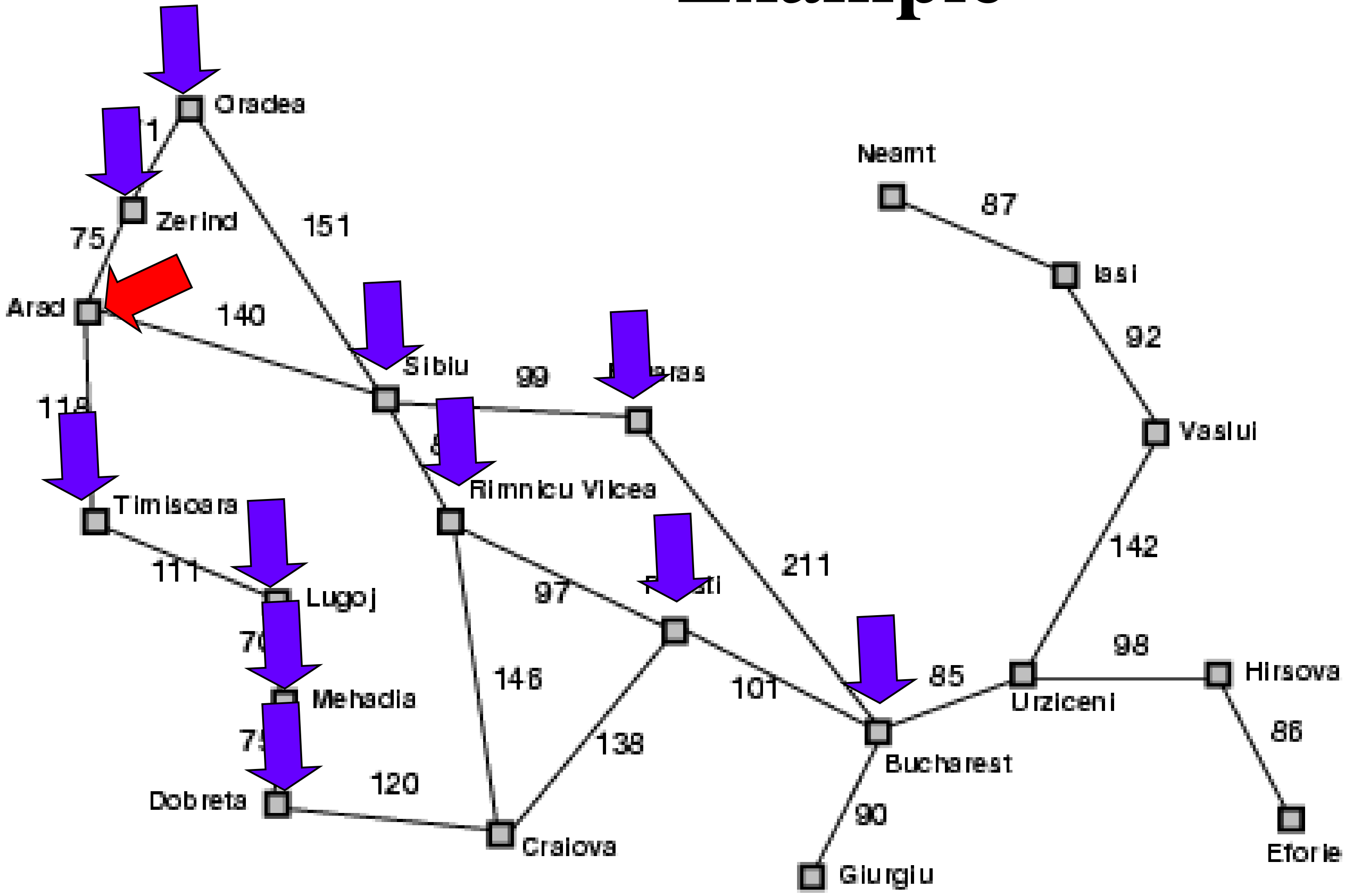
Dijkstra’s is a greedy algorithm

Why Dijkstra’s Algorithm works?

The path from u to every node in T is the minimum path



Example



Straight–line distance to Bucharest

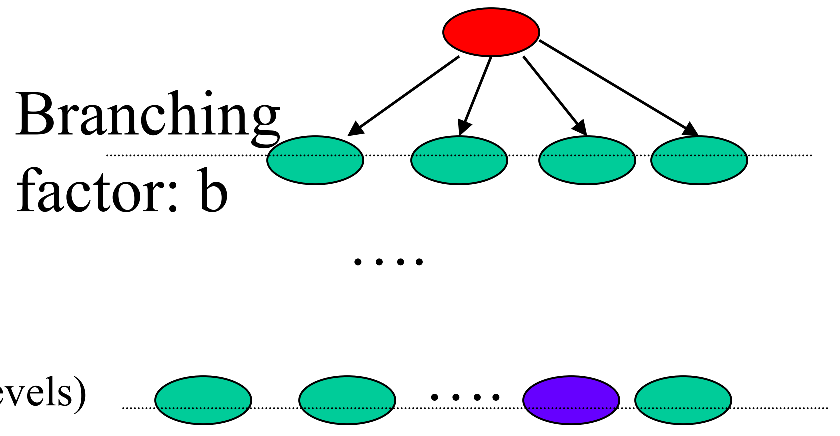
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

What does Djikstra’s algorithm will do? (minimizing g(n))

Problem: Visit too many nodes, some ***clearly*** out of the question

Complexity

- Actual complexity is $O(|E|\log_2 |E|)$
- Is this good?
  Actually it is bad for very large graphs!



# nodes = $b^{(\#\text{ levels})}$

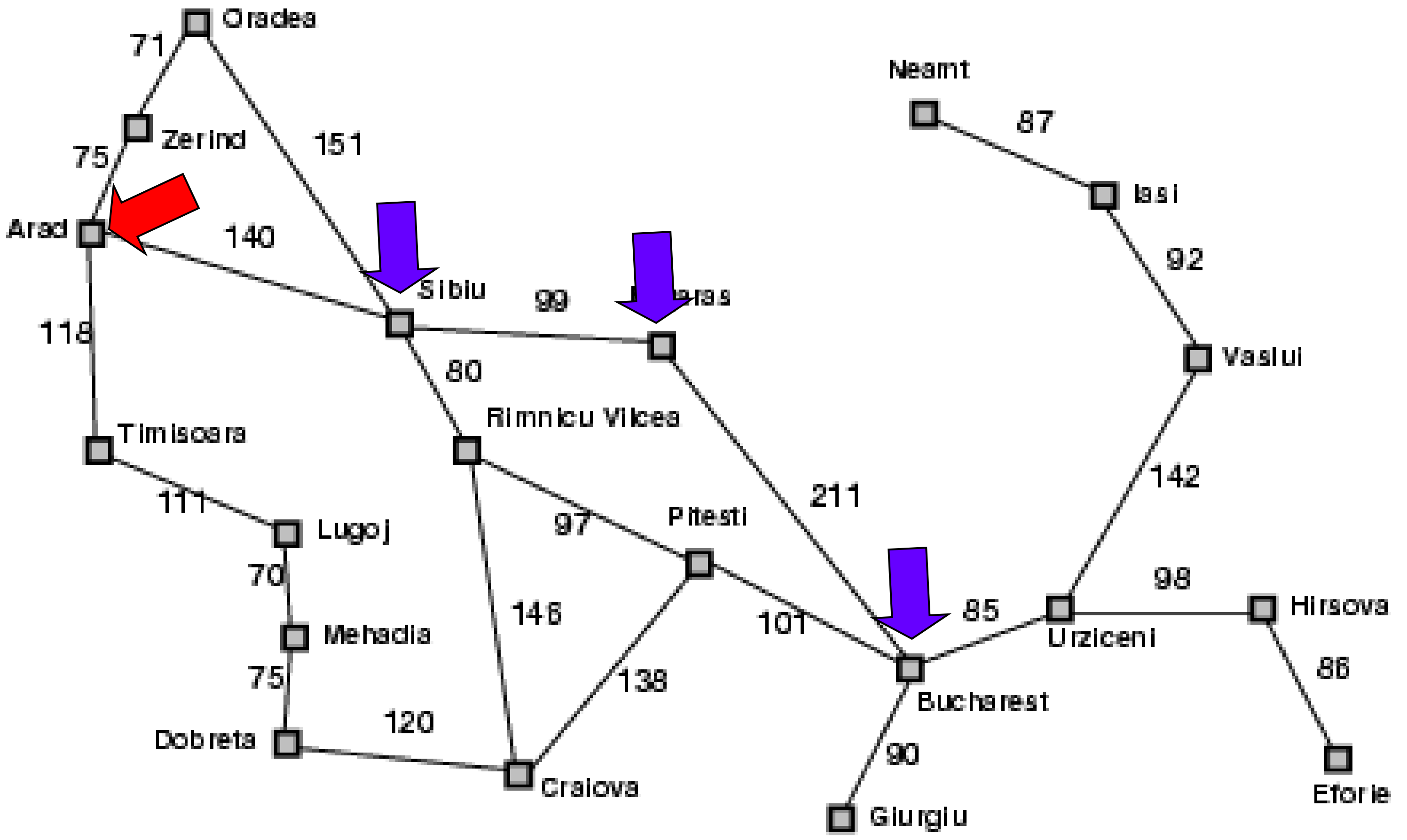
Another Example: think of the search space in chess

Better Solution: Make a ‘hunch”!

- Use ***heuristics*** to guide the search
  - **Heuristic**: estimation or “hunch” of how to search for a solution
- We define a heuristic function:

  h(n) = “estimate of the cost of the cheapest path from the starting node to the goal node”

Lets Try A Heuristic



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Heuristic: minimize h(n) = “Euclidean distance to destination”

Problem: not optimal (through Rimmici Viicea and Pitesti is shorter)

The A* Search

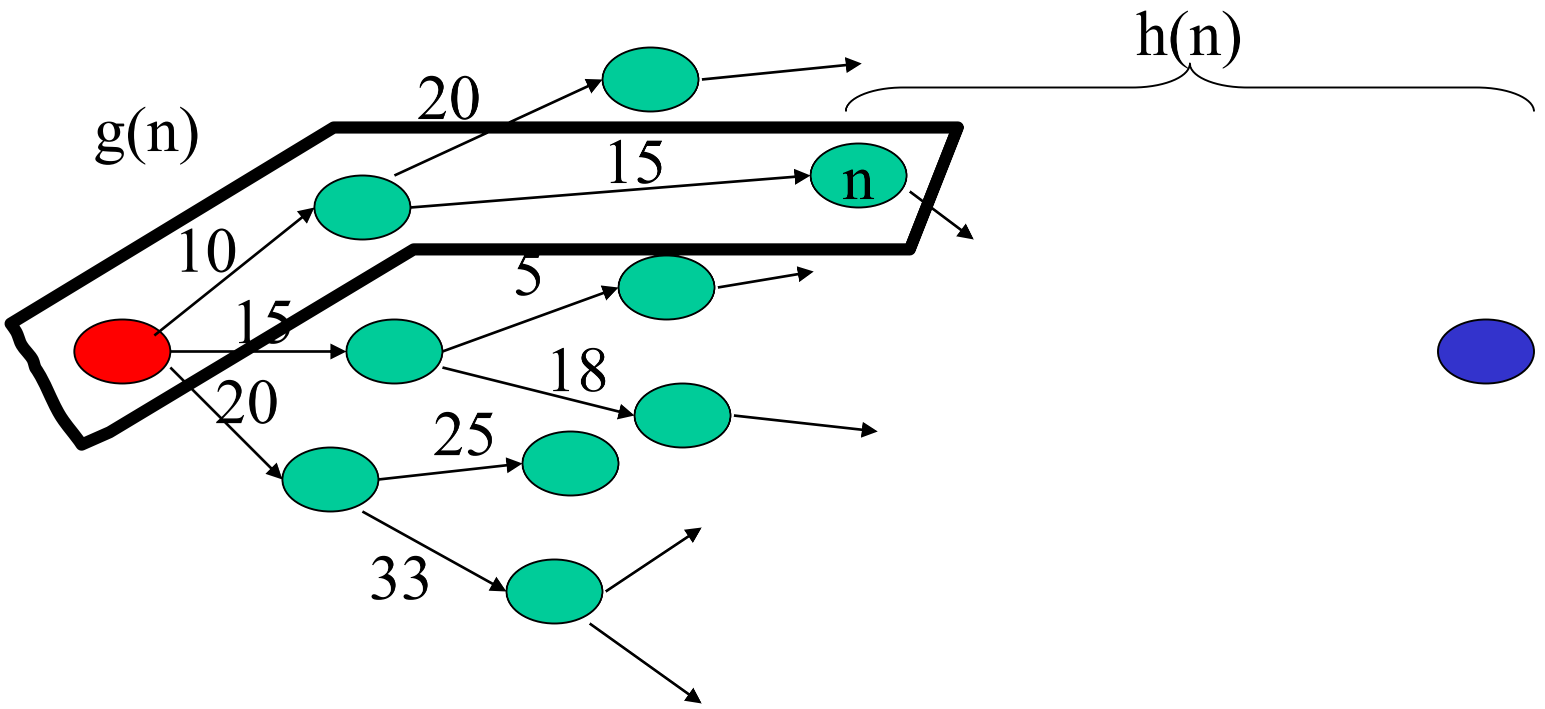
- **Difficulty**: we want to still be able to generate the path with minimum cost

- A* is an algorithm that:
  - Uses heuristic to guide search
  - While ensuring that it will compute a path with minimum cost

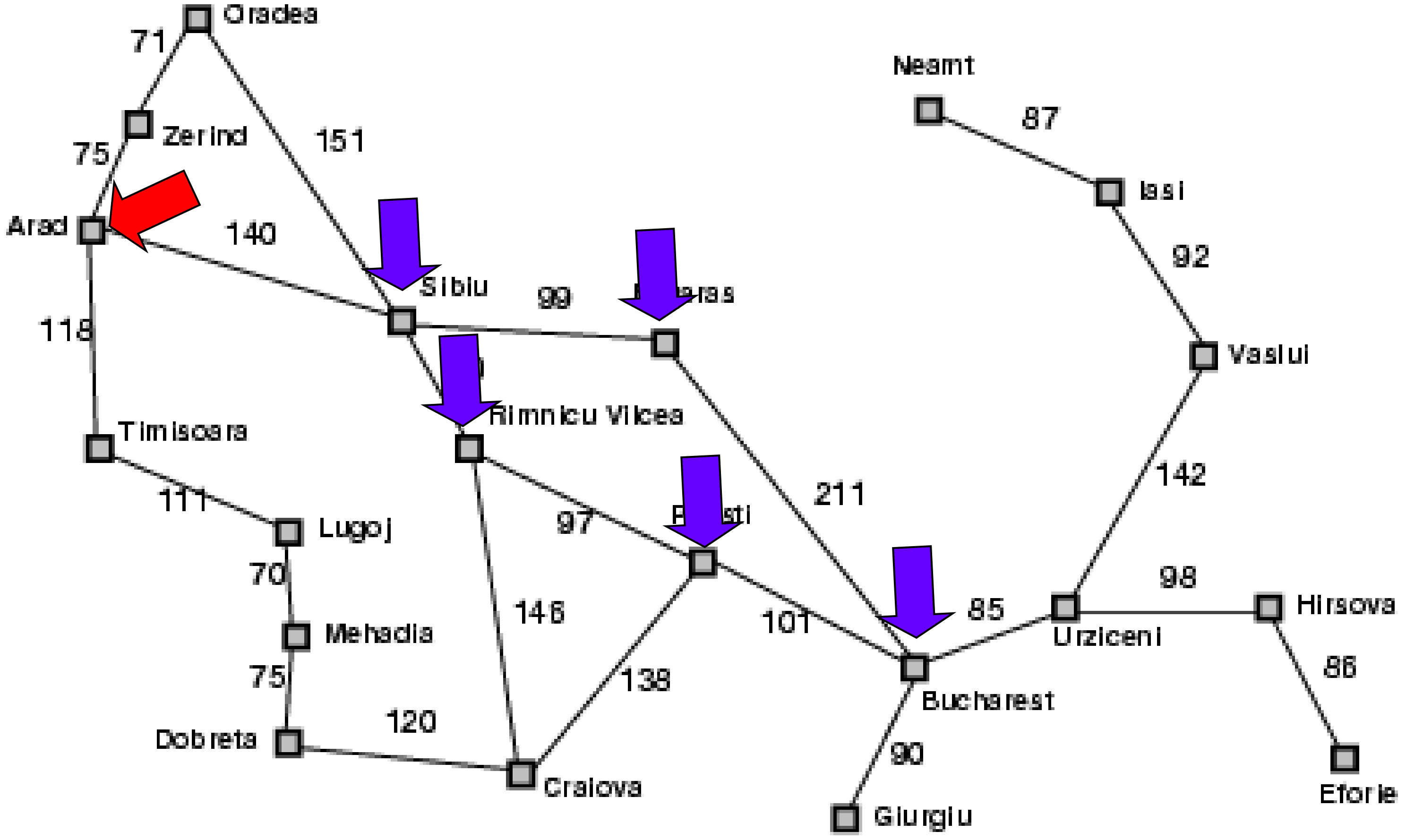
- A* computes the function $f(n) = \underbrace{g(n)}_{\text{“actual cost”}} + \overbrace{h(n)}^{\text{“estimated cost”}}$

A*

- f(n) = g(n) + h(n)
  - g(n) = "cost from the starting node to reach n"
  - h(n) = "estimate of the cost of the cheapest path from n to the goal node"



Example



Straight–line distance
to Bucharest

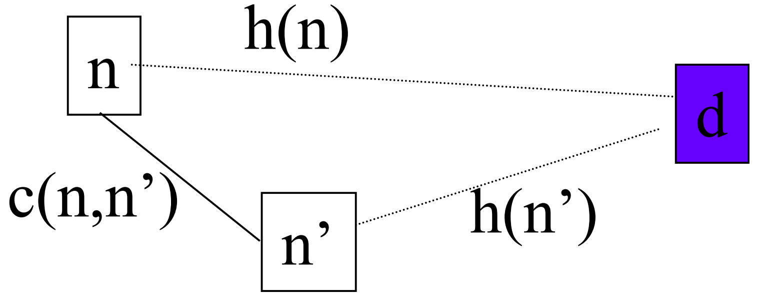
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A*: minimize f(n) = g(n) + h(n)

Properties of A*

- A* generates an optimal solution if h(n) is an admissible heuristic and the search space is a tree:
  - h(n) is **admissible** if it never overestimates the cost to reach the destination node
- A* generates an optimal solution if h(n) is a consistent heuristic and the search space is a graph:
  - h(n) is **consistent** if for every node n and for every successor node n' of n:

$$h(n) \leq c(n,n') + h(n')$$



- If h(n) is consistent then h(n) is admissible
- Frequently when h(n) is admissible, it is also consistent

Admissible Heuristics

- A heuristic is admissible if it is too optimistic, estimating the cost to be smaller than it actually is.

- Example:

  In the road map domain,

  h(n) = “Euclidean distance to destination”

  is admissible as normally cities are not connected by roads that make straight lines

How to Create Admissible Heuristics

- Relax the conditions of the problem
  - This will result in admissible heuristics!
- Lets look at an 8-puzzle game:

  http://www.permadi.com/java/puzzle8/

- Possible heuristics?

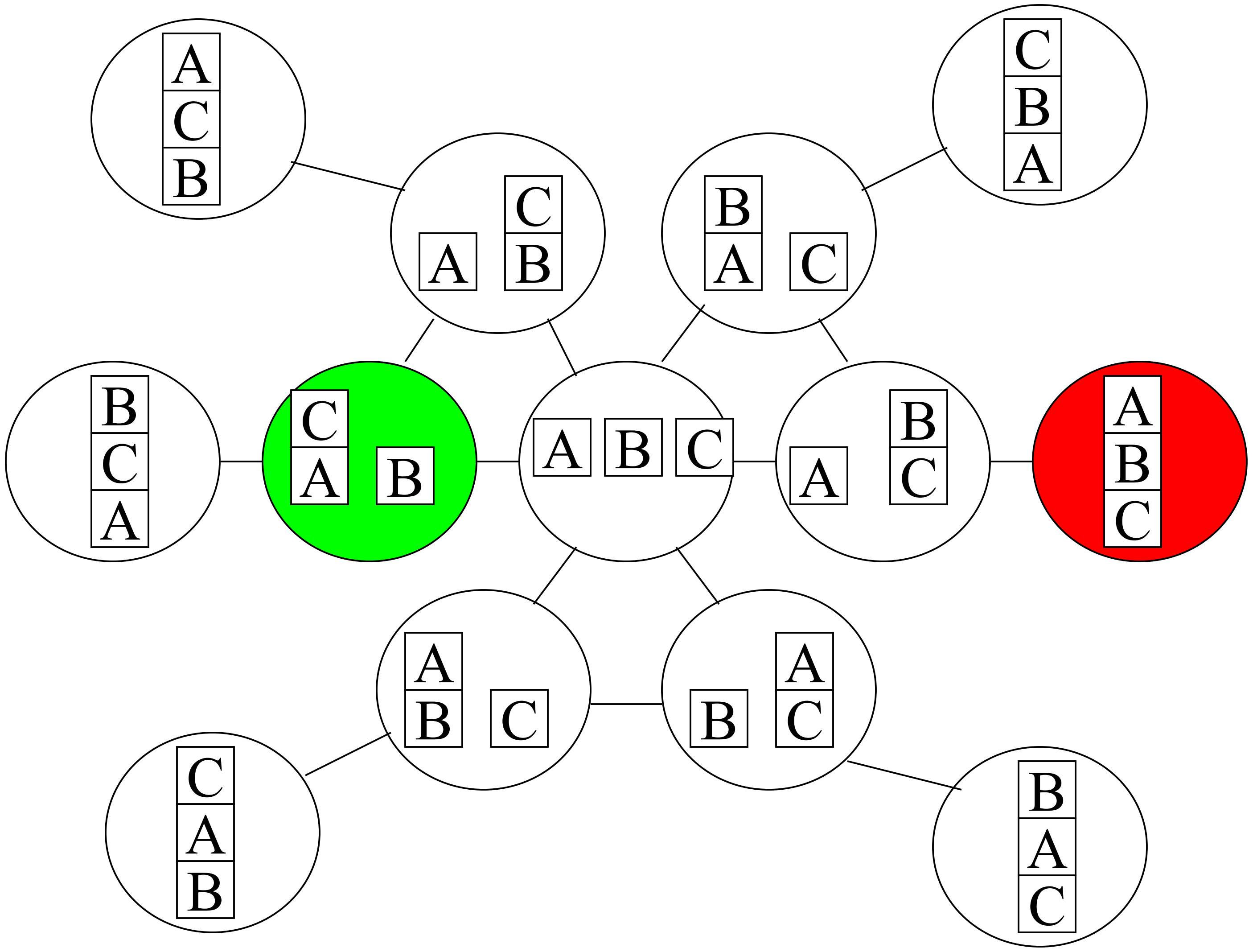
Example: Admissible Heuristics in 8-Puzzle Game

- Heuristic: a tile A can be moved to any tile B
  - H1(n) = "number of misplaced tiles in board n"

- Heuristic: a tile A can be moved to a tile B if B is adjacent to A
  - H2(n) = "sum of distances of misplaced tiles to goal positions in board n"

- Some experimental results reported in Russell & Norvig (2002):
  - A* with h2 performs up to 10 times better than A* with h1
  - A* with h2 performs up to 36,000 times better than a classical uninformed search algorithm (iterative deepening)

Using A* in Planning



h(n) = "# of goals remaining to be satisfied"   g(n) = "# of steps so far"

A* in Games

- Path finding (duh!)
  - We will have several presentations on the topic
  - We will see that sometimes even A* speed improvements are not sufficient
  - Additional improvements are required

- A* can be used for planning moves computer-controlled player (e.g., chess)

- F.E.A.R. uses A* to plan its search

Knowledge Representation

- We’ve discussed generic search techniques.
- Usually we start out with a generic technique and enhance it to take advantage of a specific domain.
- The representation of knowledge about the domain is a major issue.
- Picking a good representation can make a big difference.

Knowledge & Mappings

- Knowledge is a collection of “facts” from some domain.
- What we need is a representation of facts that can be manipulated by a program.
  - Some symbolic representation is necessary.
  - Need to be able to map facts to symbols.
  - Need to be able to map symbols to facts?

A.I. Problems & K.R.

- Game playing - need rules of the game, strategy, heuristic function(s).
- Expert Systems - list of rules, methods to extract new rules.
- Learning - the space of all things learnable (domain representation), concept representation.
- Natural Language - symbols, groupings, semantic mappings, ...

Representation Properties

<u>Representational Adequacy</u> - Is it possible to represent everything of interest ?

<u>Inferential Adequacy</u> - Can new information be inferred?

<u>Inferential Efficiency</u> - How easy is it to infer new knowledge?

<u>Acquisitional Efficiency</u> - How hard is it to gather information (knowledge)?

Using Logic to Represent Facts

- Logic representation is common in AI programs:

Spot is a dog	*dog(Spot)*
All dogs have tails	$\forall$*x:dogs(x)->hastail(x)*
Spot has a tail	*hastail(Spot)*

Relational Databases

- One way to store declarative facts is with a relational database:

Suspect	Height	Weight	Handed
OJ	6'2"	220	Right
Al	6'3"	240	Left
Kato	5'10"	170	Right

- Collection of attributes and values.

Inheritance

- It is often useful to provide a representation structure that directly supports inference mechanisms.
- *Property Inheritance* is a common inference mechanism.
- Objects belong to classes.
- Classes have properties that are inherited by objects that belong to the class.

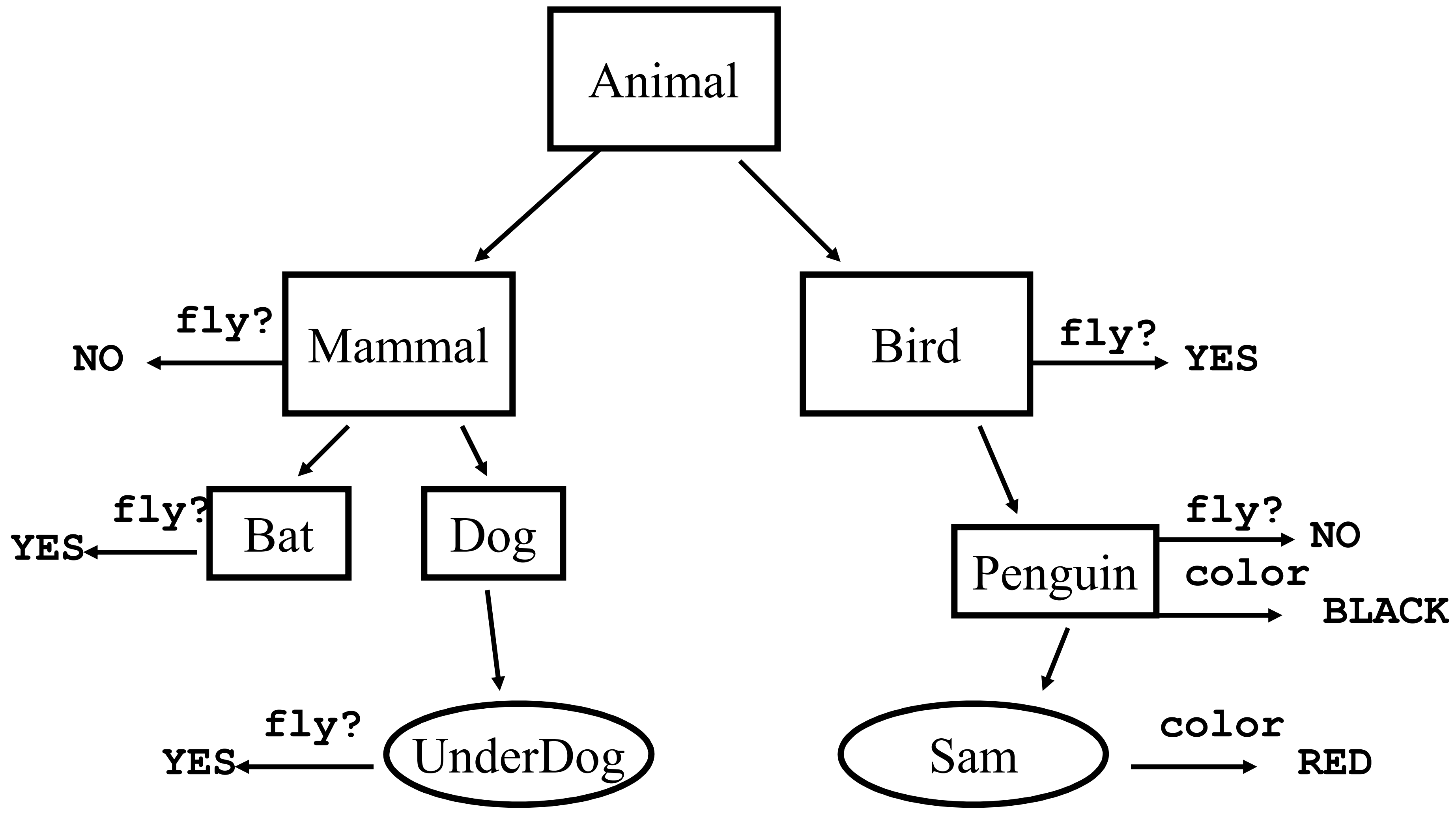
Class Hierarchy

- Classes are arranged in a hierarchy, so that some classes are members of more general classes.
- There are a variety of representation strategies used in AI that are based on inheritance:

slot-and-filler

semantic network

frame system



Animal
Mammal
fly?
NO
Bird
fly?
YES
Bat
fly?
YES
Dog
Penguin
fly?
NO
color
BLACK
UnderDog
fly?
YES
Sam
color
RED

Inferential Knowledge

- Inheritance is not the only inferential mechanism - logic formulas are often used:

$$\forall x, y : Batter(x) \wedge batted(x, y) \wedge Infield - Fly(y) \rightarrow Out(x)$$

- We will study logical based inference procedures soon.

Procedural Knowledge

- Some knowledge in contained in the code we write (for example, a hard coded chess strategy).
- How does procedural knowledge do with respect to the representation properties:
  - – Representational Adequacy
  - – Inferential Adequacy
  - – Inferential Efficiency
  - – Acquisitional Efficiency

Granularity of Representation

- High-level facts may require lots of storage if represented as a collection of low-level primitives.
- Most knowledge is available in a high-level form (English).
- It is not always clear what low-level primitives should be.

Representing Sets of Object

- Extensional definition: list all members of a set.
  - Dorks = {Bill, Bob, Dave, Jane}

- Intensional: use rules to define membership in a set:
  - Dork = {x: geek(x) and bald(x) }

Logic

- Use mathematical deduction to derive new knowledge.
- Predicate Logic is a powerful representation scheme used by many AI programs.
- Propositional logic is much simpler (less powerful).

Propositional Logic

- Symbols represent *propositions* (facts).
- A proposition is either *TRUE* or *FALSE*.
- Boolean connectives can join propositions together into complex *sentences*.
- Sentences are statements that are either *TRUE* or *FALSE*.

Propositional Logic Syntax

- The constants *TRUE* and *FALSE*.
- Symbols such as *P* or *Q* that represent propositions.
- Logical connectives:

$\wedge$	AND, conjunction
$\vee$	OR, disjunction
$\Rightarrow$	Implication , conditional (If then)
$\Leftrightarrow$	Equivalence , biconditional
$\neg$	Negation (unary)
( )	parentheses (grouping)

Sentences

- *True*, *False* or any proposition symbol is a sentence.
- Any sentence surrounded by parentheses is a sentence.
- The disjunction, conjunction, implication or equivalence of 2 sentences is a sentence.
- The negation of a sentence is a sentence.

Sentence Validity

- A propositional sentence is valid (TRUE) if and only if it is true under all possible interpretations in all possible domains.
- For example:

```
If Today_Is_Tuesday Then We_Have_Class
```

The truth does not depend on whether today is Tuesday but on whether the relationship is true.

Inference Rules

- There are many patterns that can be formally called *rules of inference* for propositional logic.
- These patterns describe how new knowledge can be derived from existing knowledge, both in the form of propositional logic sentences.
- Some patterns are common and have fancy names.

Inference Rule Notation

- When describing an inference rule, the *premise* specifies the pattern that must match our knowledge base and the *conclusion* is the new knowledge inferred.
- We will use the notation

premise ⇒ *conclusion*

Inference Rules

- Modus Ponens: $x \Rightarrow y, x \vdash y$
- And-Elimination: $x_1 \wedge x_2 \wedge \ldots \wedge x_n \vdash x_i$
- And-Introduction: $x_1, x_2, \ldots, x_n \vdash x_1 \wedge x_2 \wedge \ldots \wedge x_n$
- Or-Introduction: $x \vdash x \vee y \vee z \vee \ldots$
- Double-Negation Elimination: $\neg \neg x \vdash x$
- Unit Resolution: $x \vee y, \neg x \vdash y$

Resolution Inference Rule

$x \vee y, \neg y \vee z$ ↲ $x \vee z$

-or-

$\neg x \Rightarrow y, y \Rightarrow z$ ↲ $\neg x \Rightarrow z$

Logic & Finding a Proof

- Given
  - a knowledge base represented as a set of propositional sentences.
  - a goal stated as a propositional sentence
  - a list of inference rules
- We can write a program to repeatedly apply inference rules to the knowledge base in the hope of deriving the goal.

Example

It will snow OR there will be a test.

Dave is Darth Vader OR it will not snow.

Dave is not Darth Vader.

Will there be a test?

Solution

Snow = a Test = b Dave is Vader = c

Knowledge Base (these are all true):

$$a \vee b, \quad c \vee \neg a, \quad \neg c$$

By Resolution we know $b \vee c$ is true.

By Unit Resolution we know b is true.

There will be a test!

Developing a Proof Procedure

- Deriving (or refuting) a goal from a collection of logic facts corresponds to a very large search tree.
- A large number of *rules of inference* could be utilized.
- The selection of which rules to apply and when would itself be non-trivial.

Resolution & CNF

- *Resolution* is a single rule of inference that can operate efficiently on a special form of sentences.
- The special form is called *clause form* or *conjunctive normal form* (CNF), and has these properties:
  - Every sentence is a disjunction (or) of literals
  - All sentences are implicitly conjuncted (anded).

Propositional Logic and CNF

- Any propositional logic sentence can be converted to CNF. We need to remove all connectives other than OR (without modifying the meaning of a sentence)

Converting to CNF

- Eliminate implications and equivalence.
- Reduce scope of all negations to single term.
- Use associative and distributive laws to convert to a conjunct of disjuncts.
- Create a separate sentence for each conjunct.

Eliminate Implications and Equivalence

$$x \Rightarrow y \quad \text{becomes} \quad \neg x \vee y$$

$$x \Leftrightarrow y \quad \text{becomes} \quad (\neg x \vee y) \wedge (\neg y \vee x)$$

Reduce Scope of Negations

$\neg (\neg x)$ becomes x

$\neg(x \vee y)$ becomes $(\neg y \wedge \neg x)$

$\neg(x \wedge y)$ becomes $(\neg y \vee \neg x)$

deMorgans Laws

Convert to conjunct of disjuncts

Associative property:

(A v B) v C = A v (B v C)

Distributive property:

(A ^ B) v C = (A v C) ^ (B v C)

Using Resolution to Prove

- Convert all propositional sentences that are in the knowledge base to CNF.
- Add the <u>contradiction</u> of the goal to the knowledge base (in CNF).
- Use resolution as a rule of inference to prove that the combined facts can not all be true.

Proof by contradiction

- We assume that all original facts are TRUE.
- We add a new fact (the contradiction of sentence we are trying to prove is TRUE).
- If we can infer that FALSE is TRUE we know the knowledgebase is corrupt.
- The only thing that might not be TRUE is the negation of the goal that we added, so if must be FALSE. Therefore the goal is true.

Propositional Example: The Mechanics of Proof

Knowledge Base:

$$\left.\begin{array}{l}\mathbf{P} \\ (\boldsymbol{P} \wedge \boldsymbol{Q}) \Rightarrow \mathbf{R} \\ (\mathbf{S} \vee \mathbf{T}) \Rightarrow \mathbf{Q} \\ \mathbf{T}\end{array}\right\}$$

These represent the facts we know to be true.

Goal:

$$\left.\mathbf{R}\right\}$$

This is what we want to prove is true.

Conversion to CNF

Sentence	CNF
P	**P**
$(P \wedge Q) \Rightarrow \mathbf{R}$	$\neg \mathbf{P} \vee \neg \mathbf{Q} \vee \mathbf{R}$
$(\mathbf{S} \vee \mathbf{T}) \Rightarrow \mathbf{Q}$	$\neg \mathbf{S} \vee \mathbf{Q}$
	$\neg \mathbf{T} \vee \mathbf{Q}$
T	**T**

Add Contradiction of Goal

- The goal is **R**, so we add $\neg$ **R** to the list of facts, the new set is:

1. $\mathbf{P}$
2. $\neg \mathbf{P} \vee \neg \mathbf{Q} \vee \mathbf{R}$
3. $\neg \mathbf{S} \vee \mathbf{Q}$
4. $\neg \mathbf{T} \vee \mathbf{Q}$
5. $\mathbf{T}$
6. $\neg \mathbf{R}$

Resolution Rule of Inference

Recall the general form of resolution:

$$x_1 \lor x_2 \lor \ldots \lor x_n \lor z, \quad y_1 \lor y_2 \lor \ldots y_m \lor \neg z \Leftarrow$$

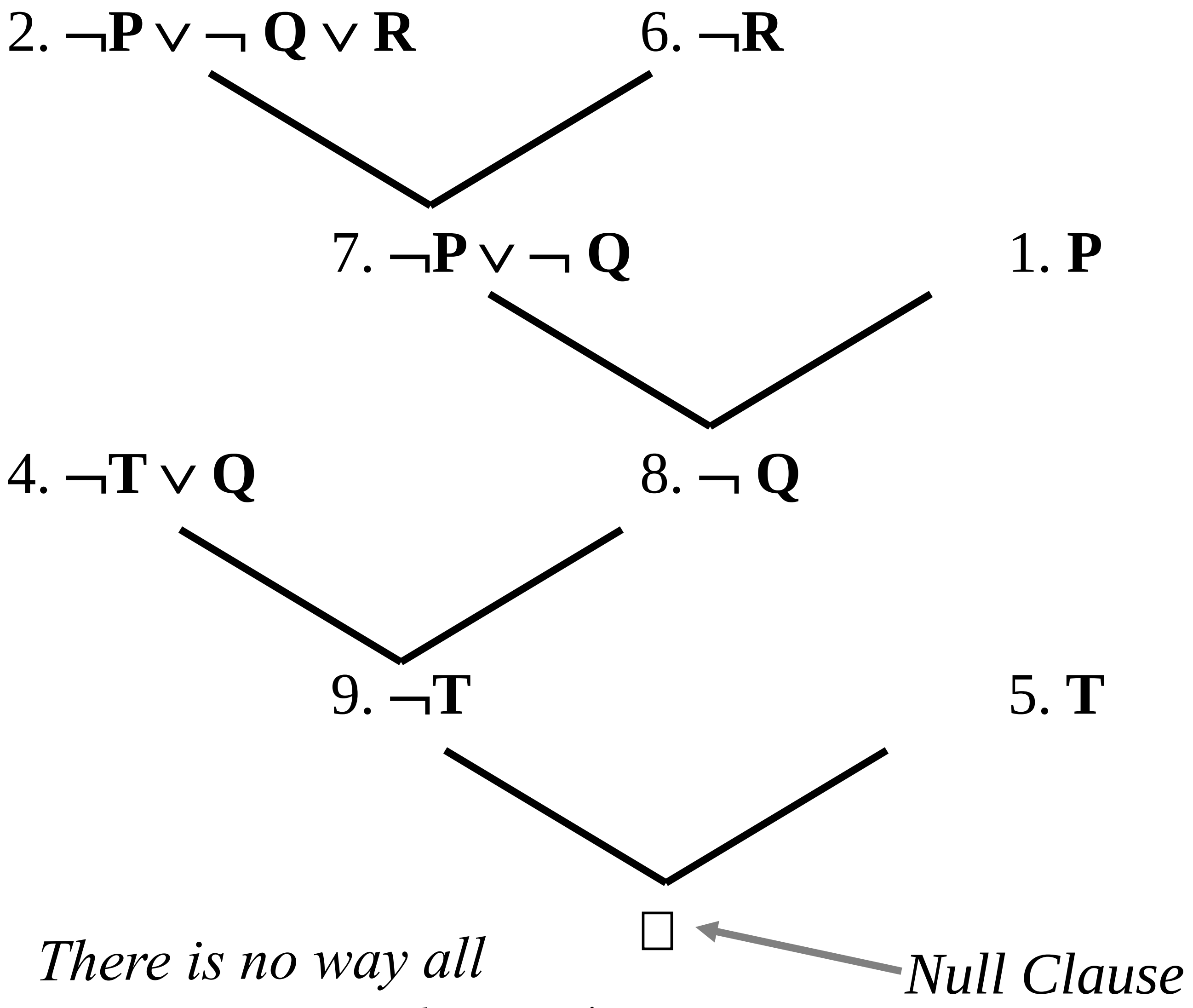
$$x_1 \lor x_2 \lor \ldots \lor x_n \lor y_1 \lor y_2 \lor \ldots y_m$$

Applying Resolution

Fact 2 can be *resolved* with fact 6, yielding a new fact:

$$\begin{tikzcd}
\neg \mathbf{P} \lor \neg \mathbf{Q} \lor \mathbf{R} \arrow[dr, no head] & & \neg \mathbf{R} \arrow[dl, no head] \\
& \neg \mathbf{P} \lor \neg \mathbf{Q} &
\end{tikzcd}$$

A new fact, call it fact 7.



There is no way all the clauses can be true!

A more intuitive look at the same example.

P: Joe is smart

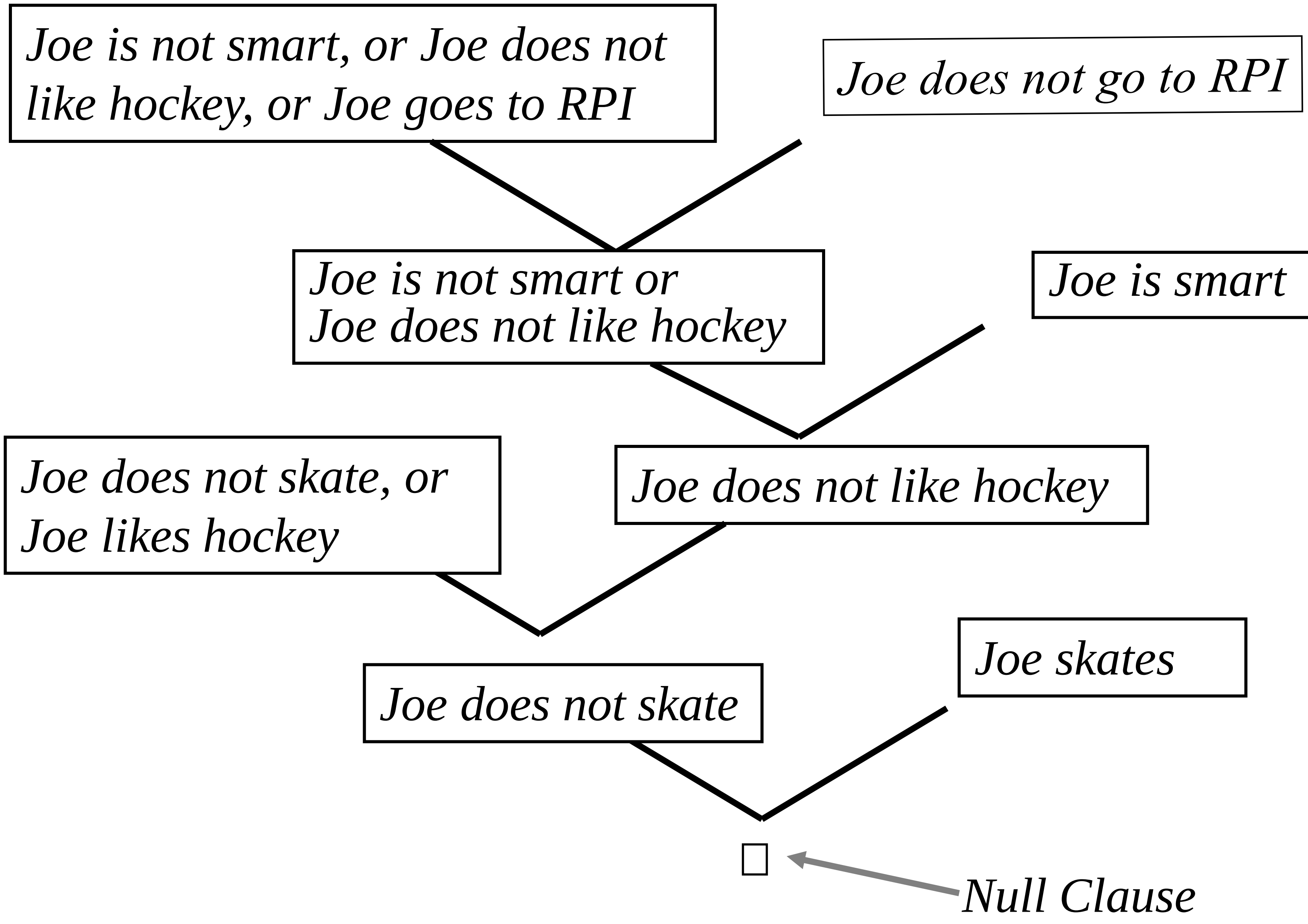
Q: Joe likes hockey

R: Joe goes to RPI

S: Joe is Canadian

T: Joe skates.

- Original Sentences:
  - Joe is smart
  - If Joe is smart and Joe likes hockey, Joe goes to RPI
  - If Joe is Canadian or Joe skates, Joe likes hockey.
  - Joe skates.
- After conversion to CNF:
  - Fact 2: Joe is not smart, or Joe does not like hockey, or Joe goes to RPI.
  - Fact 3: Joe is not Canadian or Joe likes hockey.
  - Fact 4: Joe does not skate, or Joe likes hockey.


Joe is not smart, or Joe does not like hockey, or Joe goes to RPI
Joe does not go to RPI
Joe is not smart or Joe does not like hockey
Joe is smart
Joe does not skate, or Joe likes hockey
Joe does not like hockey
Joe skates
Joe does not skate
□
Null Clause

Propositional Logic Limits

- The expressive power of propositional logic is limited. The assumption is that everything can be expressed by simple facts.
- It is much easier to model real world objects using *properties* and *relations*.
- Predicate Logic provides these capabilites more formally and is used in many AI domains to represent knowledge.

Propositional Logic Problem

If the unicorn is mythical, then it is immortal, but if it is not mythical then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

Q: Is the unicorn mythical? Magical? Horned?

Predicate Logic Background

Predicate Logic Review

- Logic is used to describe properties of objects in some “world”.
- Precise, mathematical syntax that makes it possible derive/discover new properties of the objects.
- Properties are described using a declarative language - statements are declarations, not lists of instructions.

Sentences

- Sentences in Predicate logic are statements of relations of objects.
- Sentences are based on relations (a.k.a. predicates) and connectives (conjuctions, disjuctions, implications, …).

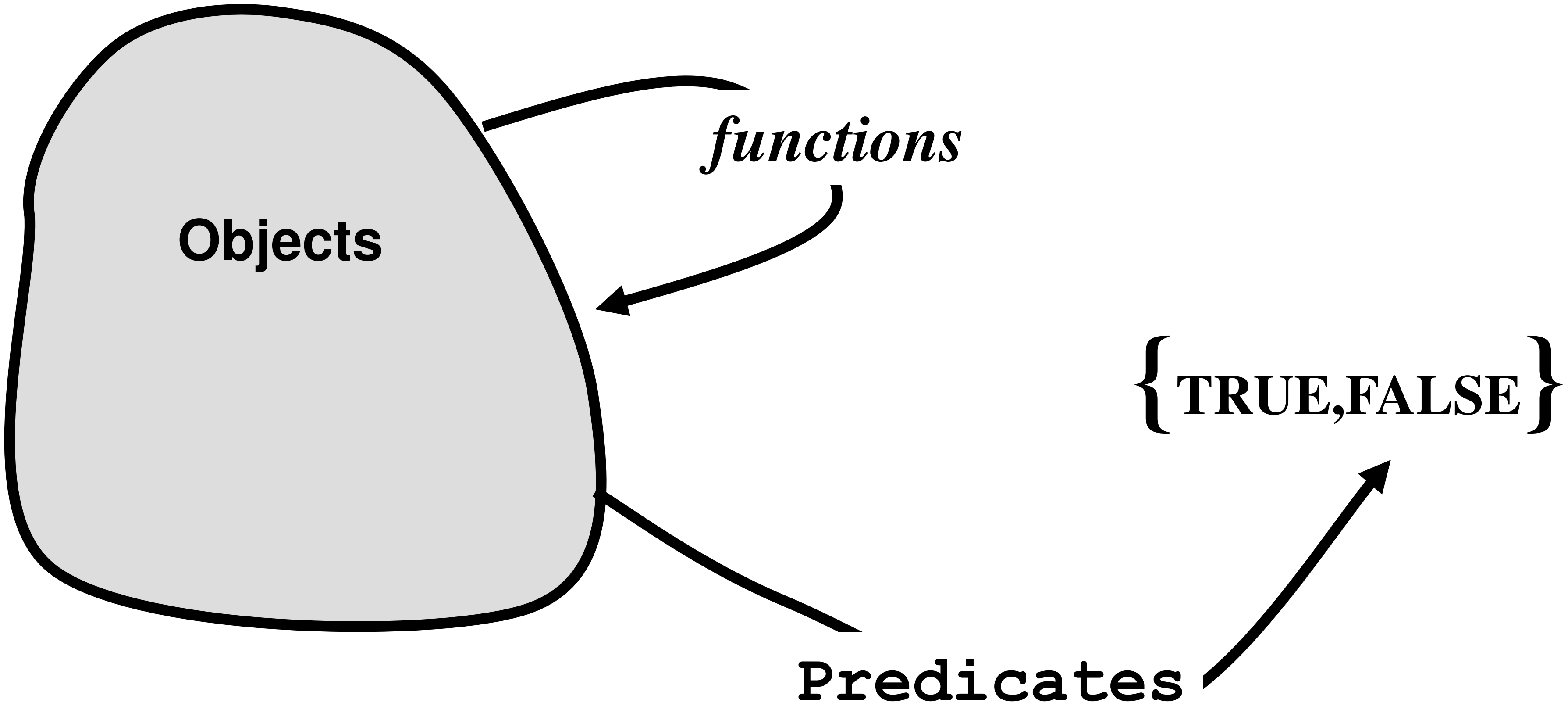
Objects

- Objects can be represented by:
  - constants (refer to a specific object)
  - variables
  - functions (mapping objects to objects)

Predicates (Relations)

- A Predicate is a statement about a property of object(s).
- Predicates can be used to partition objects into classes or to establish relationships between objects.

Predicate Logic



Examples

- Assume objects in our "world" are positive integers:
  - Some functions: *successor(x)*, *predecessor(x)*
  - Some predicates: `Even`(x), `Odd`(**x**)
  - Some simple statements:

    `Even`(2)

    `Even`(*successor*(1))

    `Odd`(*predecessor*(6))

More examples

`Equal`(x,y)

`Equal`$(\mathit{successor}(4),5)$

`Equal`$(\mathit{successor}(4),x)$

$\neg$ `Equal`$(\mathit{successor}(x),x)$

Connectives

- Conjuction (AND) $\wedge$
- Disjunction (OR) $\vee$
- Implication (If-Then) $\rightarrow$

  reminder:

  $A \rightarrow B$ is TRUE whenever A is FALSE!
- Variables are no different than variables we use in programming.
  - Name doesn't matter.

Examples

`Equal(`*successor(x),y*`)` $\rightarrow$ `Equal(`*x,predecessor(y)*`)`

`Equal(`*successor(x),x*`)` $\rightarrow$ `Equal(x,1024)`

`Loves(Ginsberg,`*dog-of*`(Ginsberg))`

Truth

- Sentences are statements about objects, these statements can be true or false!

$$\mathtt{A(x)} \rightarrow \neg \mathtt{A(x)}$$

$$\neg\mathtt{HadInappropriateRelationship(Bill,}\ \textit{intern-of}\mathtt{(Bill))}$$

Predicate Logic vs. First Order Logic

- First Order Logic includes quantifiers:
  – allow us to make statements about sets of objects.
- $\forall$ - universal quantification ("for all")
- $\exists$ - existential quantification ("there exists")

Quantifiers

- Quantifiers are applied to sentences, not to objects.

$(\forall x)$`IsStudent(x)`$\rightarrow$`Likes(x,pizza)`

OK

`Likes(`$(\forall x)$`IsStudent(`x`),pizza)`

Not a valid sentence!

Models

- Def: *Atom* - simple expression formed by applying a predicate to a set of objects.
- Def: *Model* - subset of the set of atoms in our language.
  - Partitioning of set of atoms into those that are TRUE and those that are FALSE.

Model Example

- Set of objects in our “world”:

  `Tom`

- Predicates:

  `Rich(Tom), Poor(Tom)`

- Four resulting Models:

  `¬Rich(Tom), ¬Poor(Tom)`

  `¬Rich(Tom), Poor(Tom)`

  `Rich(Tom), ¬Poor(Tom)`

  `Rich(Tom), Poor(Tom)`

<u>Models</u>

M1: `¬Rich(Tom), ¬Poor(Tom)`
M2: `¬Rich(Tom), Poor(Tom)`
M3: `Rich(Tom), ¬Poor(Tom)`
M4: `Rich(Tom), Poor(Tom)`

- Under which model(s) is the following sentence TRUE:

$$\texttt{Poor(Tom)} \rightarrow \neg\texttt{Rich(Tom)}$$

Modus Ponens Example

- John is a lawyer.
- Lawyers are rich.
- Rich people have big houses.
- Big houses are a lot of work.

- We would like to conclude that John’s house is a lot of work.

Logic Representation

- `Lawyer(John)`
- 🕕x `Lawyer`(x) → `Rich`(x)
- 🕕x `House`(*house-of*(x), x)
- 🕕x,h `House`(h, x) ∧ `Rich`(x) → `Big`(h)
- 🕕h `Big`(h) ∧ 🕖 x

  ~~House(h, x) →Work(h)~~

Representing facts with Predicate Logic - Example

- Marcus was a man
- Marcus was a Pompeian
- All Pompeians were Romans
- Caesar was a ruler.
- All Romans were either loyal to Caesar or hated him.
- Everyone is loyal to someone.
- Men only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar

Predicate Logic Knowledgebase

Man(Marcus)

Pompeian(Marcus)

$\forall\, x$ Pompeian(x) $\Rightarrow$ Roman(x)

Ruler(Caesar)

$\forall\, x$ Romans(x) $\Rightarrow$ Loyalto(x,Caesar) $\vee$ Hate(x,Caesar)

$\forall\, x\; \exists\, y$ Loyalto(x,y)

$\forall\, x\; \forall\, y$ Man(x) $\wedge$ Ruler(y) $\wedge$ Tryassassinate(x,y) $\Rightarrow$
$\neg$Loyalto(x,y)

Tryassassinate(Marcus,Caesar)

Questions (Goals)

Was Marcus a Roman?

Was Marcus loyal to Caesar?

Who was Marcus loyal to?

Was Marcus a ruler?

Will the test be easy?

Proof procedure for Predicate Logic

- Same idea, but a few added complexities:
  - conversion to CNF is much more complex.
  - Matching of literals requires providing a matching of variables, constants and/or functions.

$$\neg \text{Skates}(x) \lor \text{LikesHockey}(x)$$

$$\neg \text{LikesHockey}(y)$$

We can resolve these only if we assume x and y refer to the same object.

Predicate Logic and CNF

- Converting to CNF is harder - we need to worry about variables and quantifiers.
  1. Eliminate all implications $\Rightarrow$
  2. Reduce the scope of all $\neg$ to single term. *
  3. Make all variable names unique
  4. Move Quantifiers Left *
  5. Eliminate Existential Quantifiers *
  6. Eliminate Universal Quantifiers *
  7. Convert to conjunction of disjuncts
  8. Create separate clause for each conjunct.

Eliminate Existential Quantifiers

- Any variable that is existentially quantified means we are saying there is some value for that variable that makes the expression true.
- To eliminate the quantifier, we can replace the variable with a function.
- We don’t know what the function is, we just know it exists.

Skolem functions

$\exists\ y$ President*(y)*

We replace y with a new function func:

President*(func())*

func is called a skolem function.

In general the function must have the same number of arguments as the number of universal quantifiers in the current scope.

Skolemization Example

$\forall x\ \exists y$ Father(y,x)

create a new function named foo and replace y with the function.

$\forall\ x$ Father$(foo(x),x)$

Predicate Logic Resolution

- We have to worry about the arguments to predicates, so it is harder to know when 2 literals match and can be used by resolution.
- For example, does the literal Father(Bill,Chelsea) match Father(x,y) ?
- The answer depends on how we substitute values for variables.

Unification

- The process of finding a substitution for predicate parameters is called *unification*.
- We need to know:
  - that 2 literals can be matched.
  - the substitution is that makes the literals identical.
- There is a simple algorithm called the *unification algorithm* that does this.

The Unification Algorithm

1. Initial predicate symbols must match.
2. For each pair of predicate arguments:
   - different constants cannot match.
   - a variable may be replaced by a constant.
   - a variable may be replaced by another variable.
   - a variable may be replaced by a function as long as the function does not contain an instance of the variable.

Unification Algorithm

- When attempting to match 2 literals, all substitutions must be made to the entire literal.
- There may be many substitutions that unify 2 literals, the *most general unifier* is always desired.

Unification Example

"substitute x for y"

y for x, then z for y

- P(x) and P(y): substitution = (x/y)
- P(x,x) and P(y,z): (z/y)(y/x)
- P(x,f(y)) and P(Joe,z): (Joe/x, f(y)/z)
- P(f(x)) and P(x): can't do it!
- P(x) ∨ Q(Jane) and P(Bill) ∨ Q(y):

  (Bill/x, Jane/y)

Unification & Resolution Examples

Father(Bill,Chelsea)	¬ Father(Bill,x)∨Mother(Hillary,x)
Man(Marcus)	¬ Man(x) ∨ Mortal(x)
Loves(*father*(a),a)	¬ Loves(x,y) ∨ Loves(y,x)

$\uparrow$

This is a function

Predicate Logic Resolution Algorithm

- While no empty clause exists and there are clauses that can be resolved:
  - select 2 clauses that can be resolved.
  - resolve the clauses (after unification), apply the unification substitution to the result and store in the knowledge base.

Example:

$\neg$ Smart(x) $\lor$ $\neg$ LikesHockey(x) $\lor$ RPI(x)

$\neg$ Canadian(y) $\lor$ LikesHockey(y)

$\neg$ Skates(z) $\lor$ LikesHockey(z)

Smart(Joe)

Skates(Joe)

Goal is to find out if RPI(Joe) is true.

- Man(Marcus)
- Pompeian(Marcus)
- $\neg$ Pompeian(x_1) $\vee$ Roman(x_1)
- Ruler(Caesar)
- $\neg$ Romans(x_2) $\vee$ Loyalto(x_2,Caesar) $\vee$ Hate(x_2,Caesar)
- Loyalto(x_3, $f(x_3)$)
- $\neg$ Man(x_4) $\vee$ $\neg$ Ruler(y_1) $\vee$ $\neg$ Tryassassinate(x_4,y_1) $\vee$ $\neg$ Loyalto(x_4,y_1)
- PROVE: Tryassassinate(Marcus,Caesar)