

CptS 440 / 540

Artificial Intelligence

Introduction

Why Study AI?

- AI makes computers more useful
- Intelligent computer would have huge impact on civilization
- AI cited as “field I would most like to be in” by scientists in all fields
- Computer is a good metaphor for talking and thinking about intelligence

Why Study AI?

- Turning theory into working programs forces us to work out the details
- AI yields good results for Computer Science
- AI yields good results for other fields
- Computers make good experimental subjects
- Personal motivation: mystery

What is the definition of AI?

What do you think?

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Bellman, 1978

“[The automation of] activities that we associate with human thinking, activities such as decision making, problem solving, learning”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Charniak & McDermott, 1985

“The study of mental faculties through the use of computational models”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Dean et al., 1995

“The design and study of computer programs that behave intelligently. These programs are constructed to perform as would a human or an animal whose behavior we consider intelligent”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Haugeland, 1985

“The exciting new effort to make computers think *machines with minds*, in the full and literal sense”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Kurzweil, 1990

“The art of creating machines that perform functions that require intelligence when performed by people”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Luger & Stubblefield, 1993

“The branch of computer science that is concerned with the automation of intelligent behavior”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Nilsson, 1998

“Many human mental activities such as writing computer programs, doing mathematics, engaging in common sense reasoning, understanding language, and even driving an automobile, are said to demand intelligence. We might say that [these systems] exhibit artificial intelligence”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Rich & Knight, 1991

“The study of how to make computers do things at which, at the moment, people are better”

What is the definition of AI?

Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Schalkoff, 1990

“A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes”

What is the definition of AI?

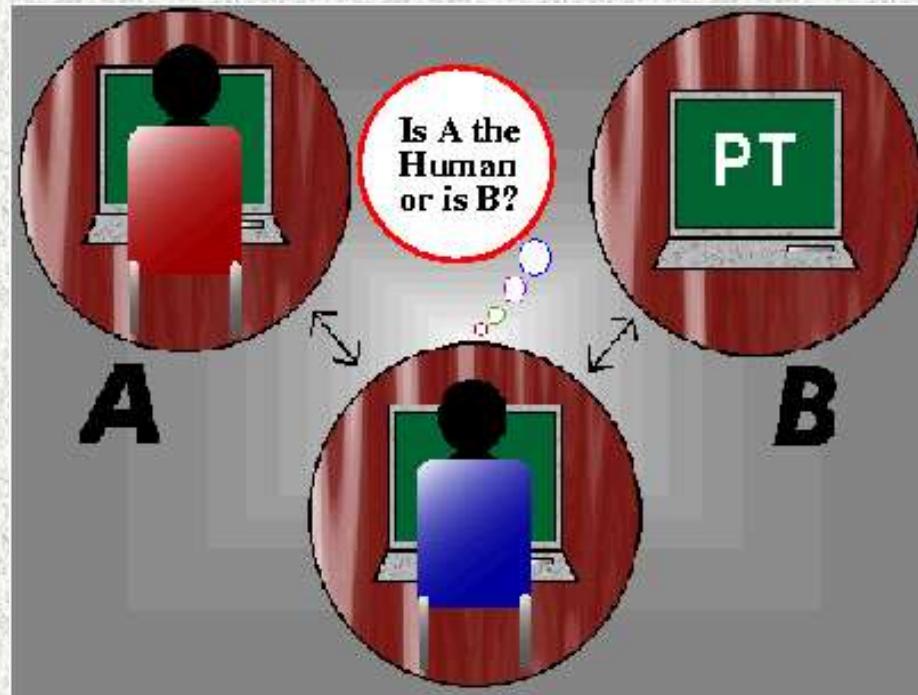
Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Winston, 1992

“The study of the computations that make it possible to perceive, reason, and act”

Approach 1: Acting Humanly

- Turing test: ultimate test for acting humanly
 - Computer and human both interrogated by judge
 - Computer passes test if judge can't tell the difference



How effective is this test?

- Agent must:
 - Have command of language
 - Have wide range of knowledge
 - Demonstrate human traits (humor, emotion)
 - Be able to reason
 - Be able to learn
- Loebner prize competition is modern version of Turing Test
 - Example: Alice, Loebner prize winner for 2000 and 2001

Chinese Room Argument



Imagine you are sitting in a room with a library of rule books, a bunch of blank exercise books, and a lot of writing utensils. Your only contact with the external world is through two slots in the wall labeled ``input'' and ``output''. Occasionally, pieces of paper with Chinese characters come into your room through the ``input'' slot. Each time a piece of paper comes in through the input slot your task is to find the section in the rule books that matches the pattern of Chinese characters on the piece of paper. The rule book will tell you which pattern of characters to inscribe the appropriate pattern on a blank piece of paper. Once you have inscribed the appropriate pattern according to the rule book your task is simply to push it out the output slot.

By the way, you don't understand Chinese, nor are you aware that the symbols that you are manipulating are Chinese symbols.

In fact, the Chinese characters which you have been receiving as input have been questions about a story and the output you have been producing has been the appropriate, perhaps even "insightful," responses to the questions asked. Indeed, to the outside questioners your output has been so good that they are convinced that whoever (or whatever) has been producing the responses to their queries must be a native speaker of, or at least extremely fluent in, Chinese.

Do you understand Chinese?



- Searle says NO
- What do you think?
- Is this a refutation of the possibility of AI?
- The Systems Reply
 - The individual is just part of the overall system, which does understand Chinese
- The Robot Reply
 - Put same capabilities in a robot along with perceiving, talking, etc. This agent would seem to have genuine understanding and mental states.

Approach 2: Thinking Humanly

- Requires knowledge of brain function
- What level of abstraction?
- How can we validate this
- This is the focus of Cognitive Science

Approach 3: Thinking Rationally

- Aristotle attempted this
- What are correct arguments or thought processes?
- Provided foundation of much of AI
- Not all intelligent behavior controlled by logic
- What is our goal? What is the purpose of thinking?

Approach 4: Acting Rationally

- Act to achieve goals, given set of beliefs
- Rational behavior is doing the “right thing”
 - Thing which expects to maximize goal achievement
- This is approach adopted by Russell & Norvig

Foundations of AI

- Philosophy
 - 450 BC, Socrates asked for algorithm to distinguish pious from non-pious individuals
 - Aristotle developed laws for reasoning
- Mathematics
 - 1847, Boole introduced formal language for making logical inference
- Economics
 - 1776, Smith views economies as consisting of agents maximizing their own well being (payoff)
- Neuroscience
 - 1861, Study how brains process information
- Psychology
 - 1879, Cognitive psychology initiated
- Linguistics
 - 1957, Skinner studied behaviorist approach to language learning

History of AI

- CS-based AI started with “Dartmouth Conference” in 1956
- Attendees
 - John McCarthy
 - LISP, application of logic to reasoning
 - Marvin Minsky
 - Popularized neural networks
 - Slots and frames
 - The Society of the Mind
 - Claude Shannon
 - Computer checkers
 - Information theory
 - Open-loop 5-ball juggling
 - Allen Newell and Herb Simon
 - General Problem Solver

AI Questions

- Can we make something that is as intelligent as a human?
- Can we make something that is as intelligent as a bee?
- Can we make something that is evolutionary, self improving, autonomous, and flexible?
- Can we save this plant \$20M/year by pattern recognition?
- Can we save this bank \$50M/year by automatic fraud detection?
- Can we start a new industry of handwriting recognition agents?

Which of these exhibits intelligence?

- You beat somebody at chess.
- You prove a mathematical theorem using a set of known axioms.
- You need to buy some supplies, meet three different colleagues, return books to the library, and exercise. You plan your day in such a way that everything is achieved in an efficient manner.
- You are a lawyer who is asked to defend someone. You recall three similar cases in which the defendant was guilty, and you turn down the potential client.
- A stranger passing you on the street notices your watch and asks, “Can you tell me the time?” You say, “It is 3:00.”
- You are told to find a large Phillips screwdriver in a cluttered workroom. You enter the room (you have never been there before), search without falling over objects, and eventually find the screwdriver.

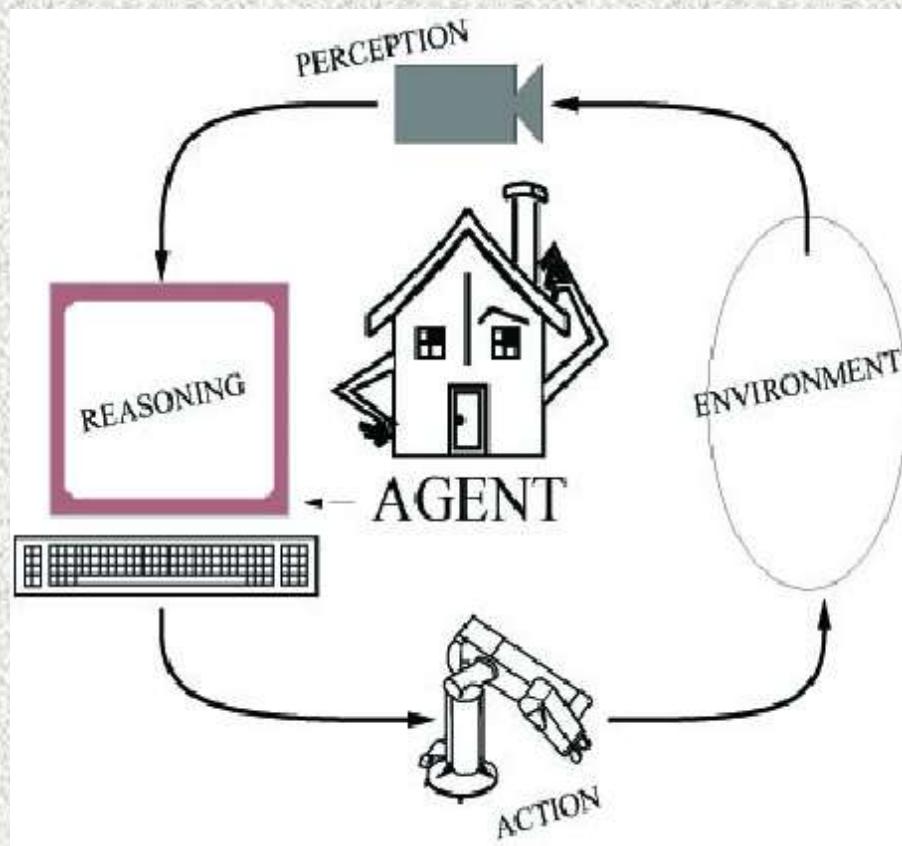
Which of these exhibits intelligence?

- You are a six-month-old infant. You can produce sounds with your vocal organs, and you can hear speech sounds around you, but you do not know how to make the sounds you are hearing. In the next year, you figure out what the sounds of your parents' language are and how to make them.
- You are a one-year-old child learning Arabic. You hear strings of sounds and figure out that they are associated with particular meanings in the world. Within two years, you learn how to segment the strings into meaningful parts and produce your own words and sentences.
- Someone taps a rhythm, and you are able to beat along with it and to continue it even after it stops.
- You are some sort of primitive invertebrate. You know nothing about how to move about in your world, only that you need to find food and keep from bumping into walls. After lots of reinforcement and punishment, you get around just fine.

Which of these can currently be done?

- Play a decent game of table tennis
- Drive autonomously along a curving mountain road
- Drive autonomously in the center of Cairo
- Play a decent game of bridge
- Discover and prove a new mathematical theorem
- Write an intentionally funny story
- Give competent legal advice in a specialized area of law
- Translate spoken English into spoken Swedish in real time
- Plan schedule of operations for a NASA spacecraft
- Defeat the world champion in chess

Components of an AI System



An **agent** perceives its environment through **sensors** and acts on the environment through **actuators**.

Human: sensors are eyes, ears, actuators (effectors) are hands, legs, mouth.

Robot: sensors are cameras, sonar, lasers, lidar, bump, effectors are grippers, manipulators, motors

The agent's behavior is described by its function that maps percept to action.

Rationality

- A rational agent does the **right thing** (what is this?)
- A fixed **performance measure** evaluates the sequence of observed action effects on the environment

PEAS

- Use PEAS to describe task
 - Performance measure
 - Environment
 - Actuators
 - Sensors

PEAS

- Use PEAS to describe task environment
 - Performance measure
 - Environment
 - Actuators
 - Sensors
- Example: Taxi driver
 - Performance measure: safe, fast, comfortable (maximize profits)
 - Environment: roads, other traffic, pedestrians, customers
 - Actuators: steering, accelerator, brake, signal, horn
 - Sensors: cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors

Environment Properties

- Fully observable vs. partially observable
- Deterministic vs. stochastic / strategic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock						
Chess without a clock						

Fully observable vs. partially observable
Deterministic vs. stochastic / strategic
Episodic vs. sequential
Static vs. dynamic
Discrete vs. continuous
Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker						

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples

Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon						



Fully observable vs. partially observable
 Deterministic vs. stochastic / strategic
 Episodic vs. sequential
 Static vs. dynamic
 Discrete vs. continuous
 Single agent vs. multiagent

Environment Examples

Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi



Fully observable vs. partially observable
 Deterministic vs. stochastic / strategic
 Episodic vs. sequential
 Static vs. dynamic
 Discrete vs. continuous
 Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

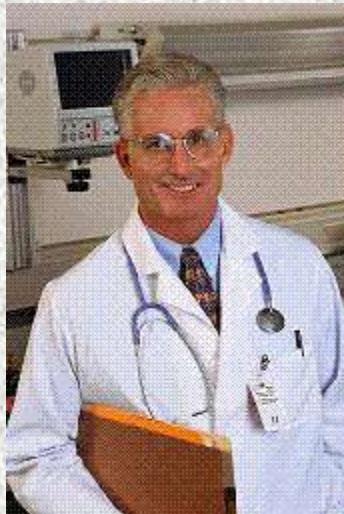
Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis						

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

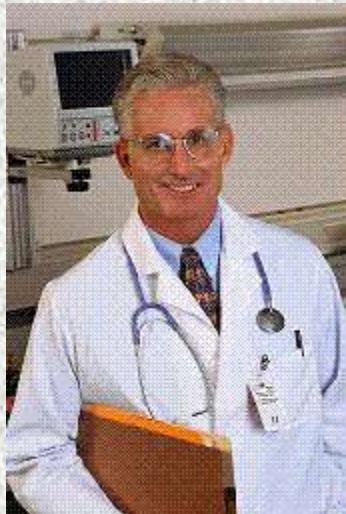
Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single
Image analysis						

Fully observable vs. partially observable

Deterministic vs. stochastic / strategic

Episodic vs. sequential

Static vs. dynamic

Discrete vs. continuous

Single agent vs. multiagent

Environment Examples



	Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Fully observable vs. partially observable	Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Deterministic vs. stochastic / strategic	Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Episodic vs. sequential Static vs. dynamic	Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Discrete vs. continuous	Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Single agent vs. multiagent	Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
	Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single
	Image analysis	Fully	Deterministic	Episodic	Semi	Discrete	Single

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single
Image analysis	Fully	Deterministic	Episodic	Semi	Discrete	Single
Robot part picking						

Fully observable vs.
partially observable

Deterministic vs.
stochastic / strategic

Episodic vs. sequential
Static vs. dynamic

Discrete vs. continuous

Single agent vs.
multiagent

Environment Examples



Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single
Image analysis	Fully	Deterministic	Episodic	Semi	Discrete	Single
Robot part picking	Fully	Deterministic	Episodic	Semi	Discrete	Single

Fully observable vs.
partially observable

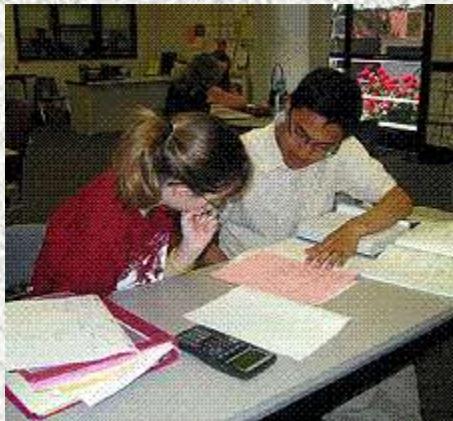
Deterministic vs.
stochastic / strategic

Episodic vs. sequential
Static vs. dynamic

Discrete vs. continuous

Single agent vs.
multiagent

Environment Examples



Fully observable vs.
partially observable

Deterministic vs.
stochastic / strategic

Episodic vs. sequential
Static vs. dynamic

Discrete vs. continuous

Single agent vs.
multiagent

Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single
Image analysis	Fully	Deterministic	Episodic	Semi	Discrete	Single
Robot part picking	Fully	Deterministic	Episodic	Semi	Discrete	Single
Interactive English tutor						

Environment Examples



Fully observable vs.
partially observable

Deterministic vs.
stochastic / strategic

Episodic vs. sequential
Static vs. dynamic

Discrete vs. continuous

Single agent vs.
multiagent

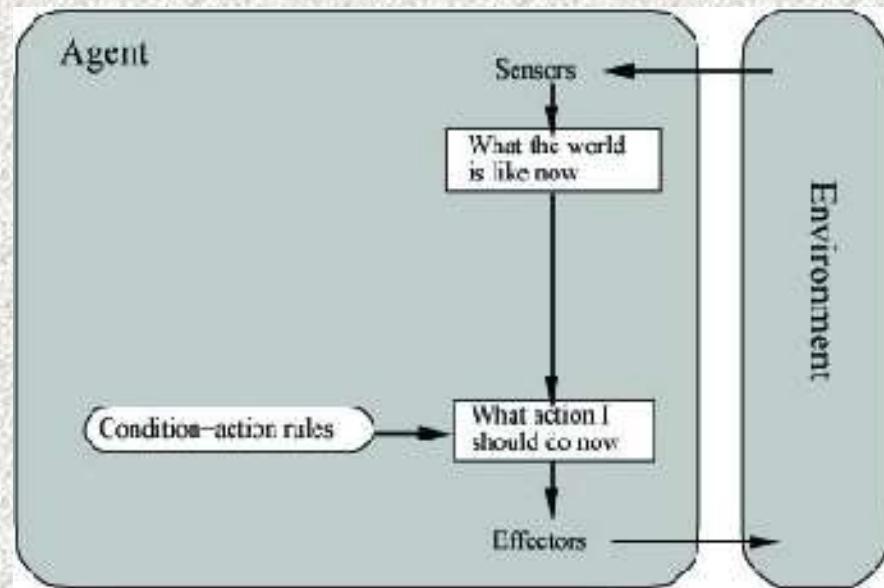
Environment	Observable	Deterministic	Episodic	Static	Discrete	Agents
Chess with a clock	Fully	Strategic	Sequential	Semi	Discrete	Multi
Chess without a clock	Fully	Strategic	Sequential	Static	Discrete	Multi
Poker	Partial	Strategic	Sequential	Static	Discrete	Multi
Backgammon	Fully	Stochastic	Sequential	Static	Discrete	Multi
Taxi driving	Partial	Stochastic	Sequential	Dynamic	Continuous	Multi
Medical diagnosis	Partial	Stochastic	Episodic	Static	Continuous	Single
Image analysis	Fully	Deterministic	Episodic	Semi	Discrete	Single
Robot part picking	Fully	Deterministic	Episodic	Semi	Discrete	Single
Interactive English tutor	Partial	Stochastic	Sequential	Dynamic	Discrete	Multi

Agent Types

- Types of agents (increasing in generality and ability to handle complex environments)
 - Simple reflex agents
 - Reflex agents with state
 - Goal-based agents
 - Utility-based agents
 - Learning agent

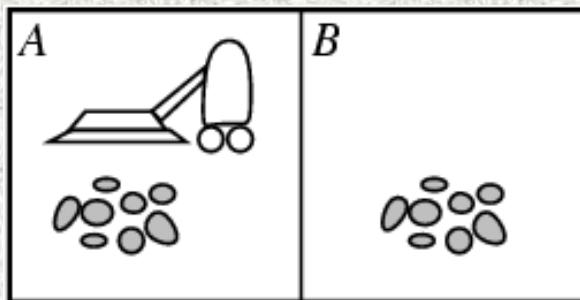
Simple Reflex Agent

- Use simple “if then” rules
- Can be short sighted



```
SimpleReflexAgent(percept)
state = InterpretInput(percept)
rule  = RuleMatch(state, rules)
action = RuleAction(rule)
Return action
```

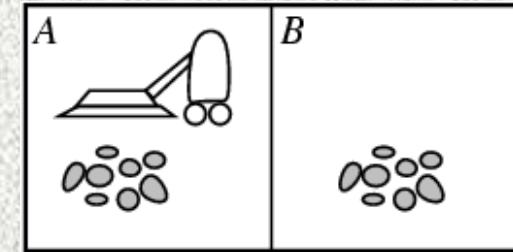
Example: Vacuum Agent



- Performance?
 - 1 point for each square cleaned in time T?
 - #clean squares per time step - #moves per time step?
- Environment: vacuum, dirt, multiple areas defined by square regions
- Actions: left, right, suck, idle
- Sensors: location and contents
 - [A, dirty]
- Rational is not omniscient
 - Environment may be partially observable
- Rational is not clairvoyant
 - Environment may be stochastic
- Thus Rational is not always successful

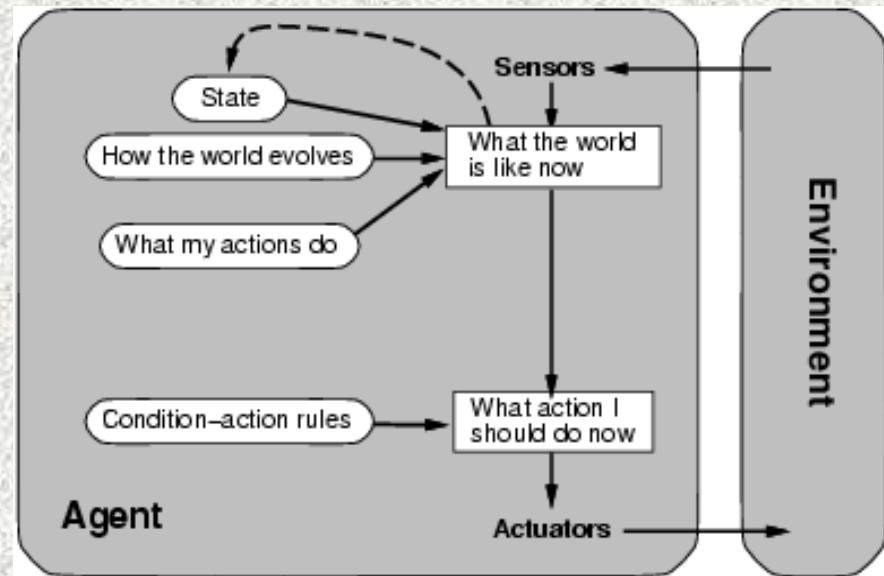
Reflex Vacuum Agent

- If status=Dirty then return Suck
else if location=A then return Right
else if location=B then right Left



Reflex Agent With State

- Store previously-observed information
- Can reason about unobserved aspects of current state



```
ReflexAgentWithState(percept)
```

```
state = UpdateDate(state,action,percept)
```

```
rule = RuleMatch(state, rules)
```

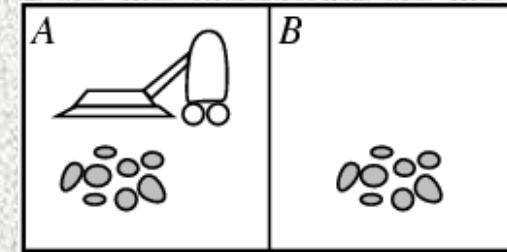
```
action = RuleAction(rule)
```

```
Return action
```

Reflex Vacuum Agent

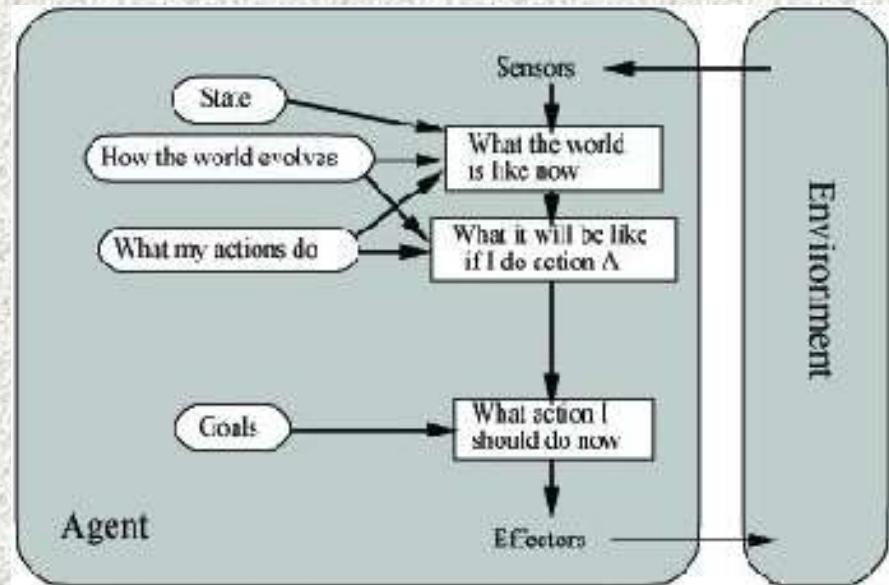
- If status=Dirty then Suck

else if have not visited other square in >3 time units, go there



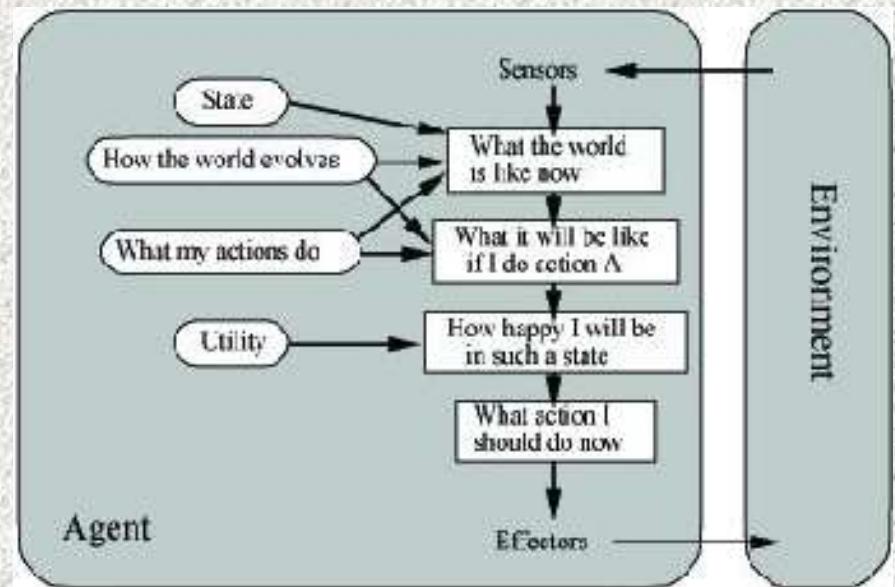
Goal-Based Agents

- Goal reflects desires of agents
- May project actions to see if consistent with goals
- Takes time, world may change during reasoning

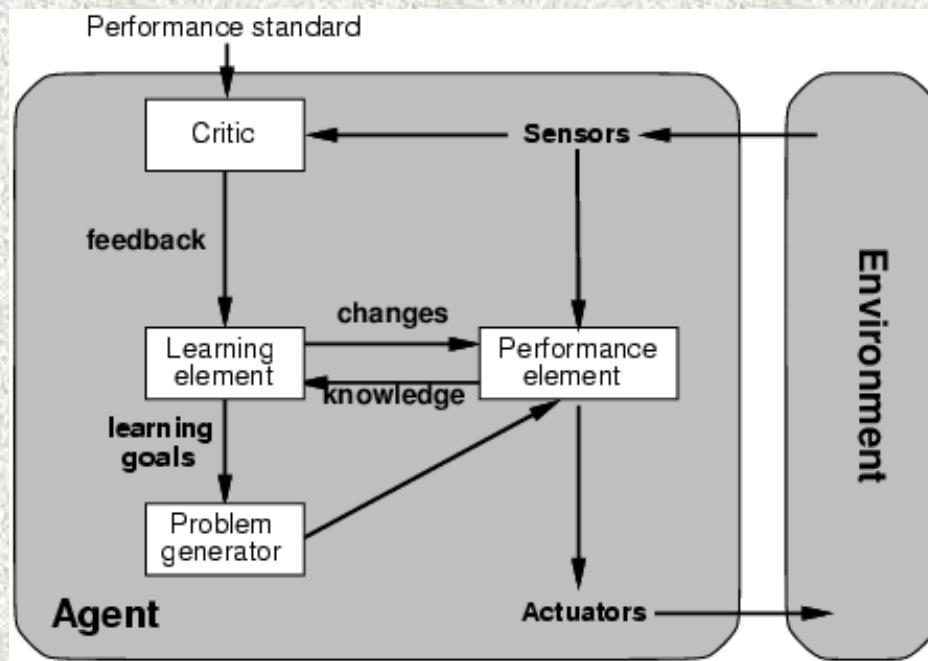


Utility-Based Agents

- Evaluation function to measure utility $f(state) \rightarrow \text{value}$
- Useful for evaluating competing goals

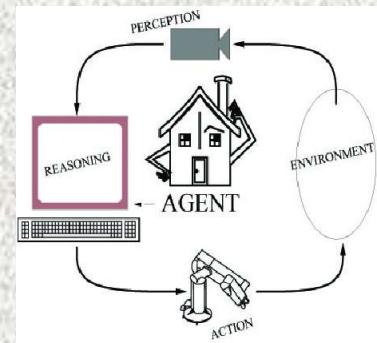


Learning Agents



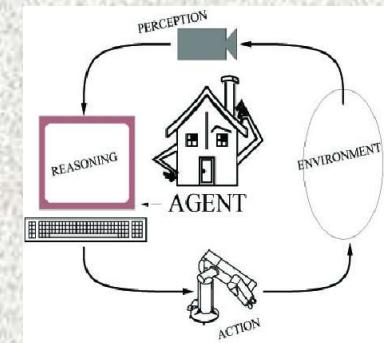
Xavier mail delivery robot

- Performance: Completed tasks
- Environment: See for yourself
- Actuators: Wheeled robot actuation
- Sensors: Vision, sonar, dead reckoning
- Reasoning: Markov model induction, A* search, Bayes classification



Pathfinder Medical Diagnosis System

- **Performance:** Correct Hematopathology diagnosis
- **Environment:** Automate human diagnosis, partially observable, deterministic, episodic, static, continuous, single agent
- **Actuators:** Output diagnoses and further test suggestions
- **Sensors:** Input symptoms and test results
- Reasoning: Bayesian networks, Monte-Carlo simulations

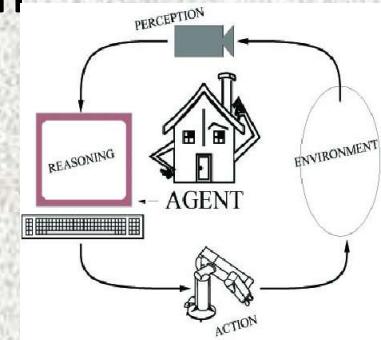


TDGammon

- **Performance:** Ratio of wins to losses
- **Environment:** Graphical output showing dice roll and piece movement, fully observable, stochastic, sequential, static, discrete, multiagent

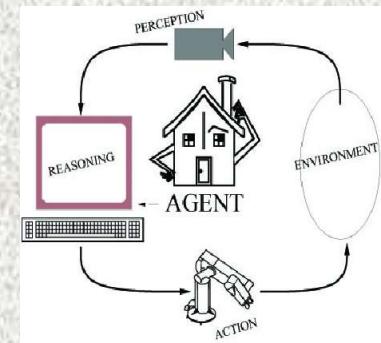
World Champion Backgammon Player

- **Sensors:** Keyboard input
- **Actuator:** Numbers representing moves of pieces
- **Reasoning:** Reinforcement learning, neural networks



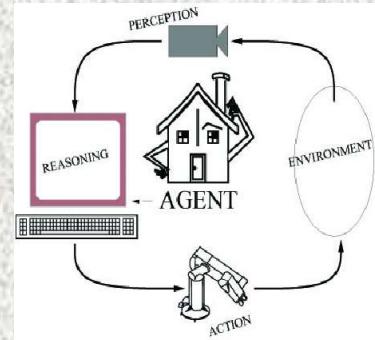
Alvinn

- **Performance:** Stay in lane, on road, maintain speed
- **Environment:** Driving Hummer on and off road without manual control (Partially observable, stochastic, episodic, dynamic, continuous, single agent), Autonomous automobile
- **Actuators:** Speed, Steer
- **Sensors:** Stereo camera input
- **Reasoning:** Neural networks



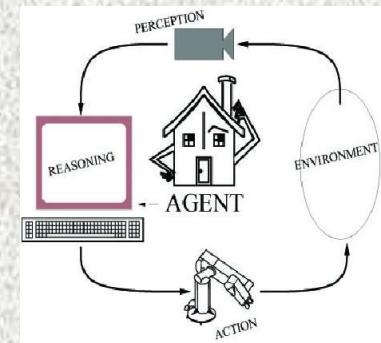
Talespin

- **Performance:** Entertainment value of generated story
- **Environment:** Generate text-based stories that are creative and understandable
 - One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe threatened to hit Irving if he didn't tell him where some honey was.
 - Henry Squirrel was thirsty. He walked over to the river bank where his good friend Bill Bird was sitting. Henry slipped and fell in the river. Gravity drowned. Joe Bear was hungry. He asked Irving Bird where some honey was. Irving refused to tell him, so Joe offered to bring him a worm if he'd tell him where some honey was. Irving agreed. But Joe didn't know where any worms were, so he asked Irving, who refused to say. So Joe offered to bring him a worm if he'd tell him where a worm was. Irving agreed. But Joe didn't know where any worms were, so he asked Irving, who refused to say. So Joe offered to bring him a worm if he'd tell him where a worm was...
- **Actuators:** Add word/phrase, order parts of story
- **Sensors:** Dictionary, Facts and relationships stored in database
- **Reasoning:** Planning



Webcrawler Softbot

- Search web for items of interest
- Perception: Web pages
- Reasoning: Pattern matching
- Action: Select and traverse hyperlinks



Other Example AI Systems

- Translation of Caterpillar truck manuals into 20 languages
- Shuttle packing
- Military planning (Desert Storm)
- Intelligent vehicle highway negotiation
- Credit card transaction monitoring
- Billiards robot
- Juggling robot
- Credit card fraud detection
- Lymphatic system diagnoses
- Mars rover
- Sky survey galaxy data analysis

Other Example AI Systems

- Knowledge Representation
- Search
- Problem solving
- Planning
- Machine learning
- Natural language processing
- Uncertainty reasoning
- Computer Vision
- Robotics



CptS 440 / 540

Artificial Intelligence

Search

Search

- Search permeates all of AI
- What choices are we searching through?
 - Problem solving
Action combinations (move 1, then move 3, then move 2...)
 - Natural language
Ways to map words to parts of speech
 - Computer vision
Ways to map features to object model
 - Machine learning
Possible concepts that fit examples seen so far
 - Motion planning
Sequence of moves to reach goal destination
- An intelligent agent is trying to find a set or sequence of actions to achieve a goal
- This is a **goal-based agent**

Problem-solving Agent

```
SimpleProblemSolvingAgent(percept)
state = UpdateState(state, percept)
if sequence is empty then
    goal = FormulateGoal(state)
    problem = FormulateProblem(state, g)
    sequence = Search(problem)
    action = First(sequence)
    sequence = Rest(sequence)
Return action
```

Assumptions

- Static or dynamic?

Environment is **static**

Assumptions

- Static or dynamic?
- Fully or partially observable?

Environment is **fully observable**

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?

Environment is discrete

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?

Environment is **deterministic**

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?

Environment is **sequential**

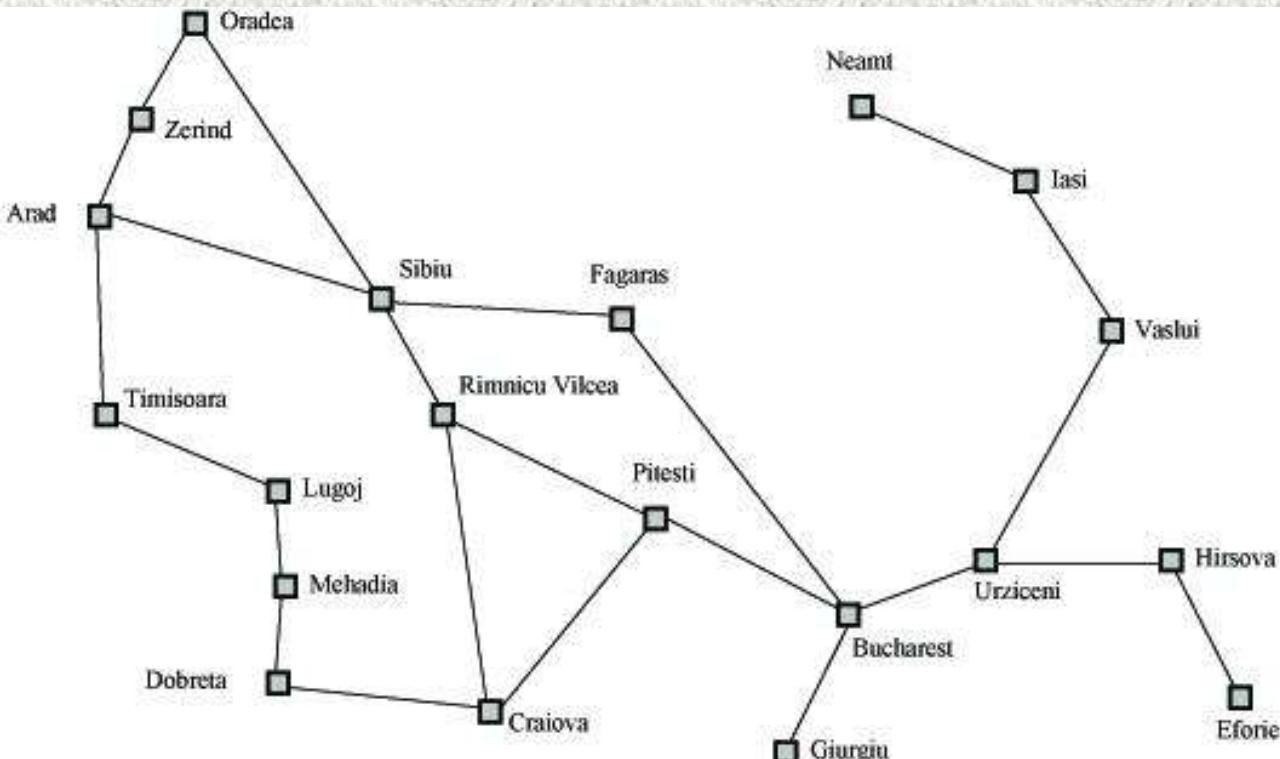
Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

Search Example



Formulate goal: Be in Bucharest.

Formulate problem: states are cities, operators drive between pairs of cities

Find solution: Find a sequence of cities (e.g., Arad, Sibiu, Fagaras, Bucharest) that leads from the current state to a state meeting the goal condition

Search Space Definitions

- **State**
 - A description of a possible state of the world
 - Includes all features of the world that are pertinent to the problem
- **Initial state**
 - Description of all pertinent aspects of the state in which the agent starts the search
- **Goal test**
 - Conditions the agent is trying to meet (e.g., have \$1M)
- **Goal state**
 - Any state which meets the goal condition
 - Thursday, have \$1M, live in NYC
 - Friday, have \$1M, live in Valparaiso
- **Action**
 - Function that maps (transitions) from one state to another

Search Space Definitions

- Problem formulation
 - Describe a general problem as a search problem
- Solution
 - Sequence of actions that transitions the world from the initial state to a goal state
- Solution cost (additive)
 - Sum of the cost of operators
 - Alternative: sum of distances, number of steps, etc.
- Search
 - Process of looking for a solution
 - Search algorithm takes problem as input and returns solution
 - We are searching through a space of possible states
- Execution
 - Process of executing sequence of actions (solution)

Problem Formulation

A search problem is defined by the

1. Initial state (e.g., Arad)
2. Operators (e.g., Arad -> Zerind, Arad -> Sibiu, etc.)
3. Goal test (e.g., at Bucharest)
4. Solution cost (e.g., path cost)

Example Problems – Eight Puzzle

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

States: tile locations

Initial state: one specific tile configuration

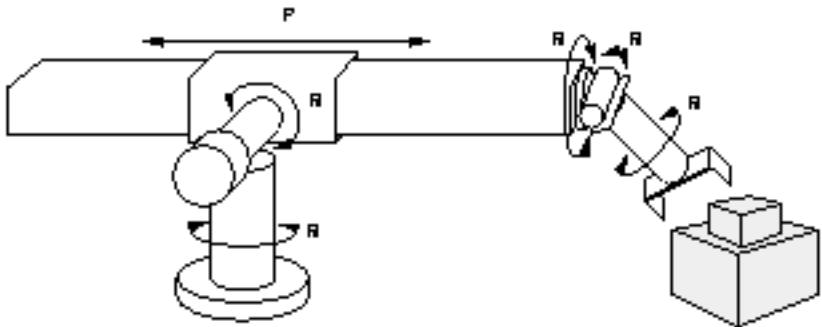
Operators: move blank tile left, right, up, or down

Goal: tiles are numbered from one to eight around the square

Path cost: cost of 1 per move (solution cost same as number of moves or path length)

[Eight puzzle applet](#)

Example Problems – Robot Assembly



States: real-valued coordinates of

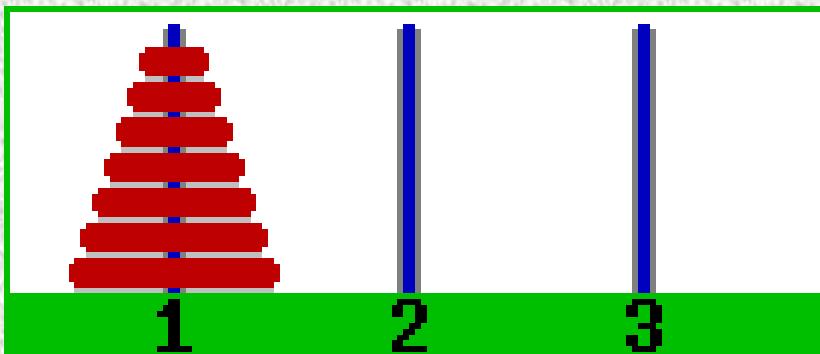
- robot joint angles
- parts of the object to be assembled

Operators: rotation of joint angles

Goal test: complete assembly

Path cost: time to complete assembly

Example Problems – Towers of Hanoi



States: combinations of poles and disks

Operators: move disk x from pole y to pole z
subject to constraints

- cannot move disk on top of smaller disk
- cannot move disk if other disks on top

Goal test: disks from largest (at bottom) to smallest on goal pole

Path cost: 1 per move

[Towers of Hanoi applet](#)

Example Problems – Rubik's Cube



States: list of colors for each cell on each face

Initial state: one specific cube configuration

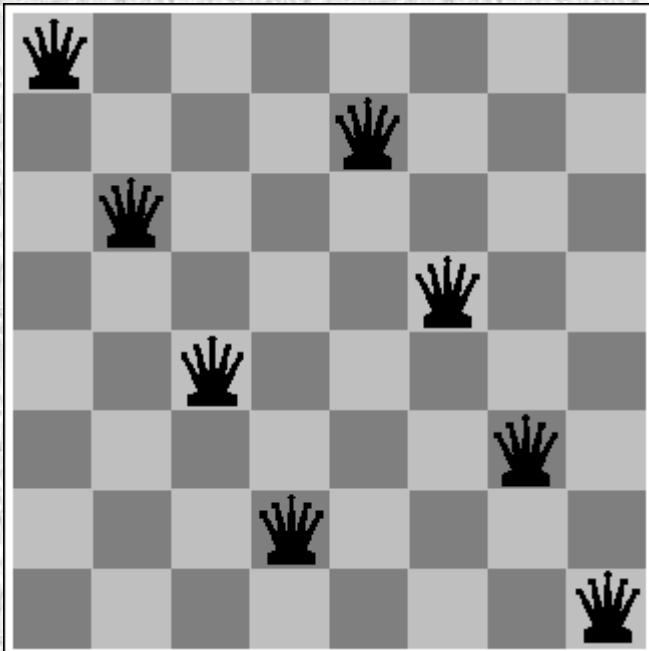
Operators: rotate row x or column y on face z direction a

Goal: configuration has only one color on each face

Path cost: 1 per move

[Rubik's cube applet](#)

Example Problems – Eight Queens



States: locations of 8 queens on chess board

Initial state: one specific queens configuration

Operators: move queen x to row y and column z

Goal: no queen can attack another (cannot be in same row, column, or diagonal)

Path cost: 0 per move

[Eight queens applet](#)

Example Problems – Missionaries and Cannibals



States: number of missionaries, cannibals, and boat on near river bank

Initial state: all objects on near river bank

Operators: move boat with x missionaries and y cannibals to other side of river

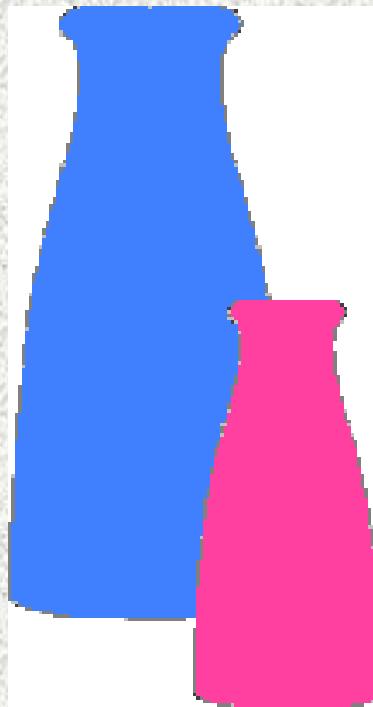
- no more cannibals than missionaries on either river bank or in boat
- boat holds at most m occupants

Goal: all objects on far river bank

Path cost: 1 per river crossing

[Missionaries and cannibals applet](#)

Example Problems –Water Jug



States: Contents of 4-gallon jug and 3-gallon jug

Initial state: (0,0)

Operators:

- fill jug x from faucet
- pour contents of jug x in jug y until y full
- dump contents of jug x down drain

Goal: (2,n)

Path cost: 1 per fill

[Saving the world, Part I](#)

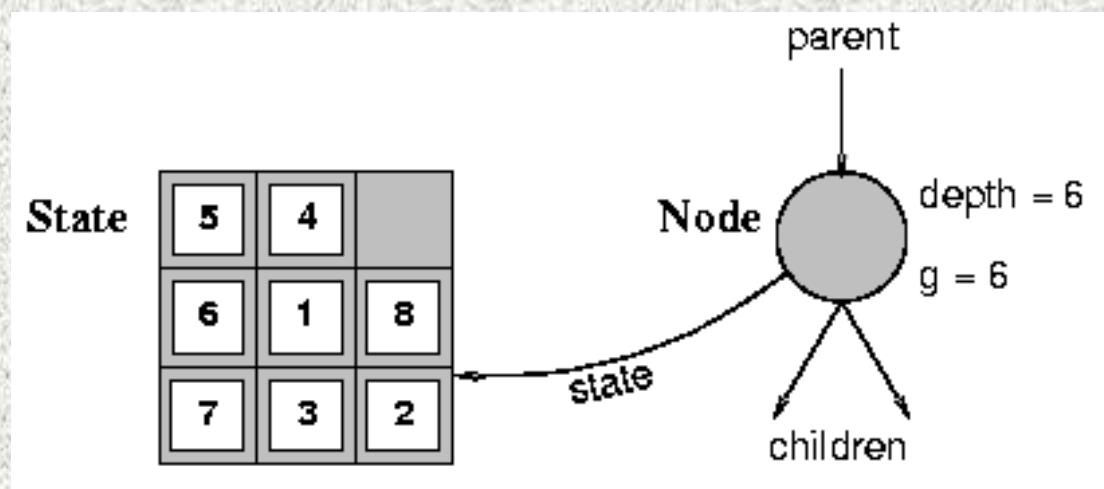
[Saving the world, Part II](#)

Sample Search Problems

- Graph coloring
- Protein folding
- Game playing
- Airline travel
- Proving algebraic equalities
- Robot motion planning

Visualize Search Space as a Tree

- States are nodes
- Actions are edges
- Initial state is root
- Solution is path from root to goal node
- Edges sometimes have associated costs
- States resulting from operator are children

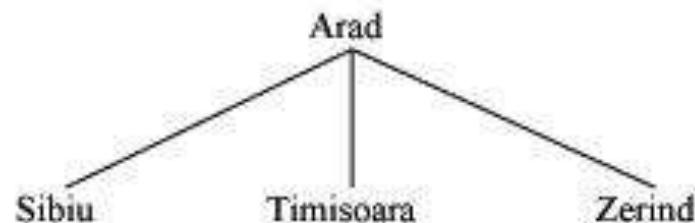


Search Problem Example (as a tree)

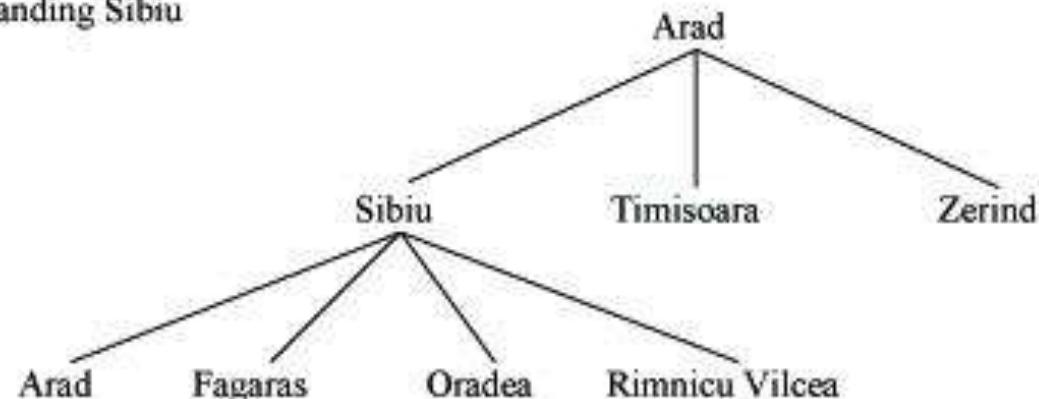
(a) The initial state

Arad

(b) After expanding Arad



(c) After expanding Sibiu



Search Function – Uninformed Searches

```
Open = initial state                                // open list is all generated states  
While open not empty                                // that have not been “expanded”  
    state = First(open)                            // one iteration of search algorithm  
    Pop(open)                                     // current state is first state in open  
    if Goal(state)                                // remove new current state from open  
        return “succeed”                           // test current state for goal condition  
    else open = QueueFunction(open, Expand(state)) // search is complete  
    Return “fail”                                 // else expand the current state by  
                                                // generating children and  
                                                // reorder open list per search strategy
```

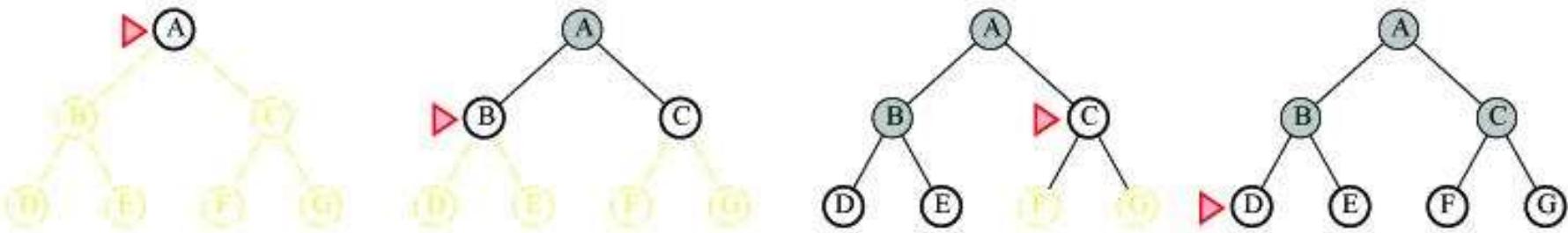
Search Strategies

- Search strategies differ only in QueuingFunction
- Features by which to compare search strategies
 - Completeness (always find solution)
 - Cost of search (time and space)
 - Cost of solution, optimal solution
 - Make use of knowledge of the domain
 - “uninformed search” vs. “informed search”

Breadth-First Search

- Generate children of a state, QueueingFn adds the children to the end of the open list
- Level-by-level search
- Order in which children are inserted on open list is arbitrary
- In tree, assume children are considered left-to-right unless specified differently
- Number of children is “branching factor” b

BFS Examples



$b = 2$

[Example trees](#)

[Search algorithms applet](#)

Analysis

- Assume goal node at level d with constant branching factor b
- Time complexity (measured in #nodes generated)
 - $1 \text{ (1}^{\text{st}} \text{ level)} + b \text{ (2}^{\text{nd}} \text{ level)} + b^2 \text{ (3}^{\text{rd}} \text{ level)} + \dots + b^d \text{ (goal level)} + (b^{d+1} - b)$
 $= O(b^{d+1})$
- This assumes goal on far right of level
- Space complexity
 - At most majority of nodes at level d + majority of nodes at level $d+1$ = $O(b^{d+1})$
 - Exponential time and space
- Features
 - Simple to implement
 - Complete
 - Finds shortest solution (not necessarily least-cost unless all operators have equal cost)

Analysis

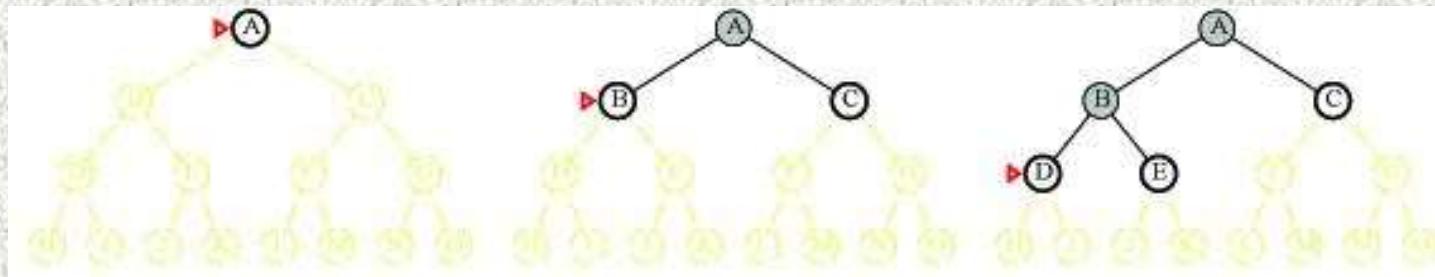
- See what happens with $b=10$
 - expand 10,000 nodes/second
 - 1,000 bytes/node

Depth	Nodes	Time	Memory
2	1110	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
15	10^{15}	3,523 years	1 exabyte

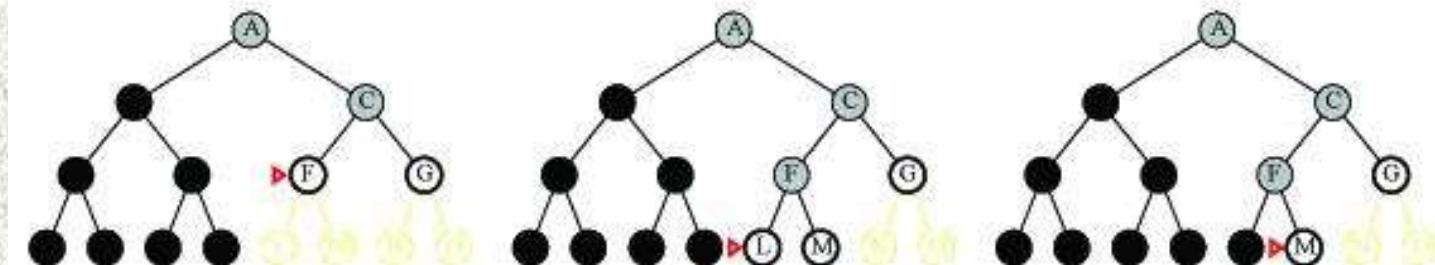
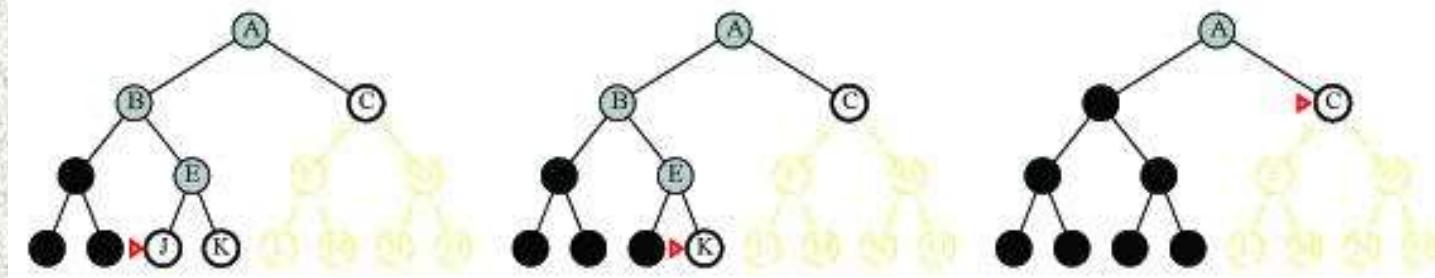
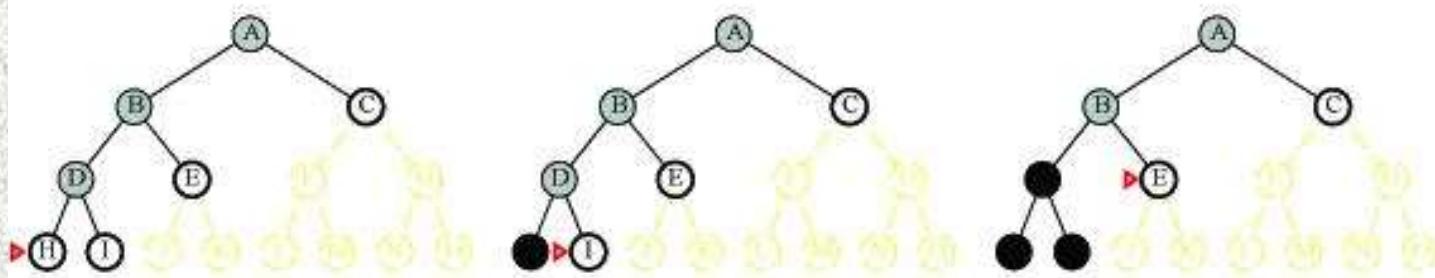
Depth-First Search

- QueueingFn adds the children to the **front** of the open list
- BFS emulates FIFO queue
- DFS emulates LIFO stack
- Net effect
 - Follow leftmost path to bottom, then backtrack
 - Expand deepest node first

DFS Examples



Example trees



Analysis

- Time complexity
 - In the worst case, search entire space
 - Goal may be at level d but tree may continue to level m , $m \geq d$
 - $O(b^m)$
 - Particularly bad if tree is infinitely deep
- Space complexity
 - Only need to save one set of children at each level
 - $1 + b + b + \dots + b$ (m levels total) = $O(bm)$
 - For previous example, DFS requires 118kb instead of 10 petabytes for $d=12$ (10 billion times less)
- Benefits
 - May not always find solution
 - Solution is not necessarily shortest or least cost
 - If many solutions, may find one quickly (quickly moves to depth d)
 - Simple to implement
 - Space often bigger constraint, so more usable than BFS for large problems

Comparison of Search Techniques

	DFS	BFS
Complete	N	Y
Optimal	N	N
Heuristic	N	N
Time	b^m	b^{d+1}
Space	bm	b^{d+1}



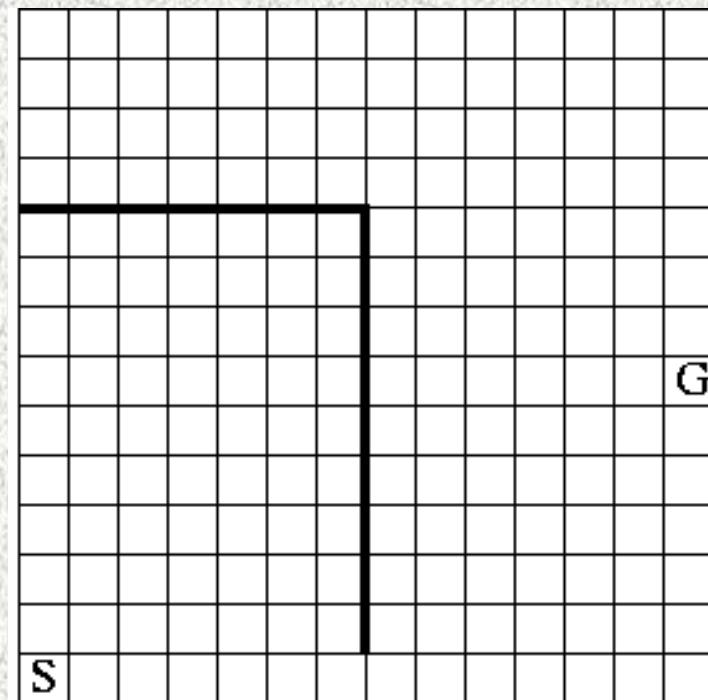
Avoiding Repeated States

Can we do it?

- Do not return to parent or grandparent state
 - In 8 puzzle, do not move up right after down
- Do not create solution paths with cycles
- Do not generate repeated states (need to store and check potentially large number of states)

Maze Example

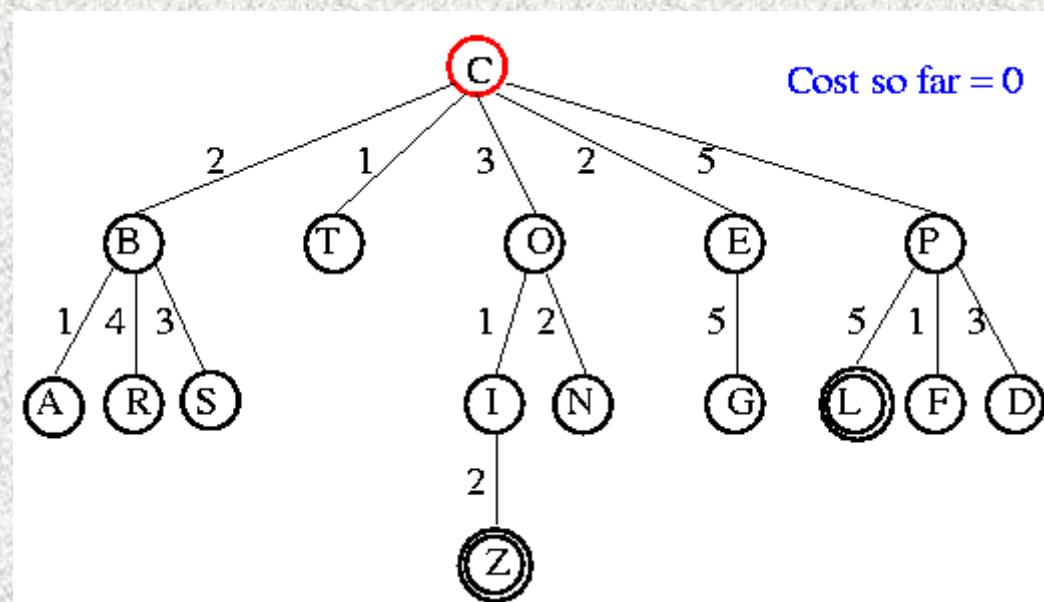
- States are cells in a maze
- Move N, E, S, or W
- What would BFS do
(expand E, then N, W, S)?
- What would DFS do?
- What if order changed to
N, E, S, W and loops are
prevented?



Uniform Cost Search (Branch&Bound)

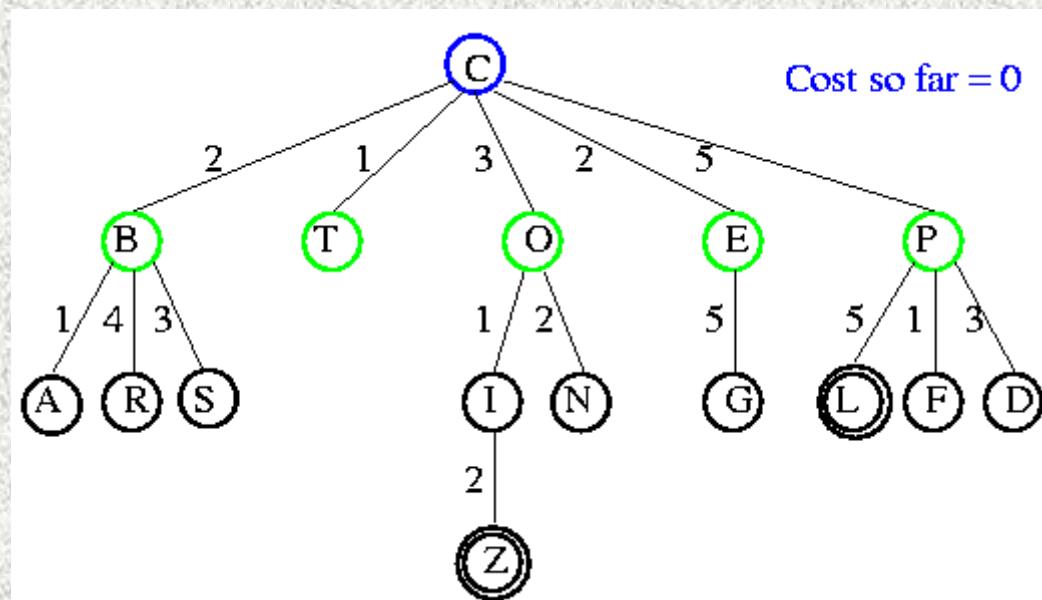
- QueueingFn is SortByCostSoFar
- Cost from root to current node n is $g(n)$
 - Add operator costs along path
- First goal found is least-cost solution
- Space & time can be exponential because large subtrees with inexpensive steps may be explored before useful paths with costly steps
- If costs are equal, time and space are $O(b^d)$
 - Otherwise, complexity related to cost of optimal solution

UCS Example



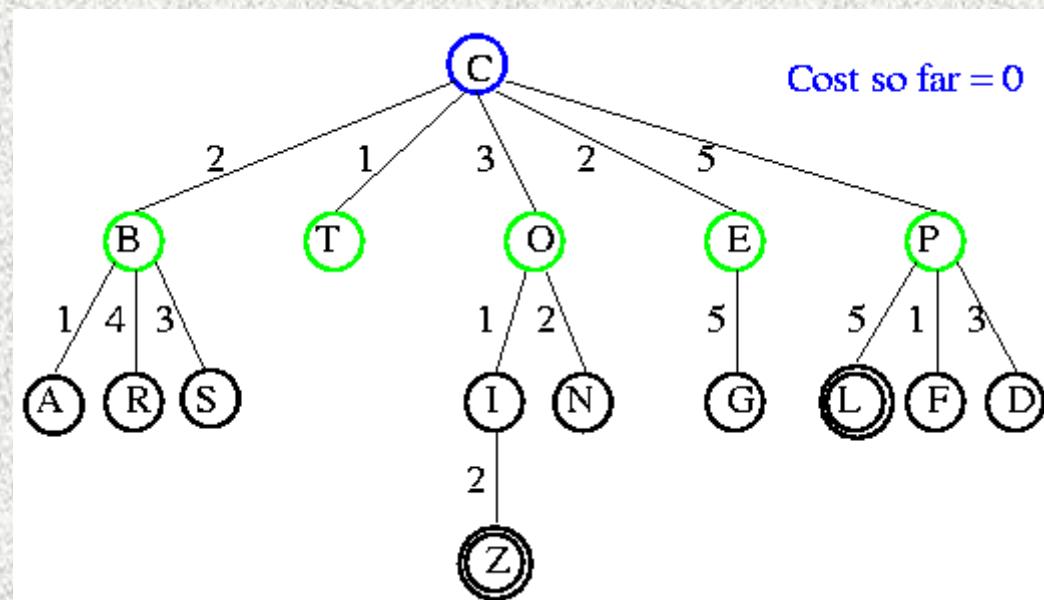
Open list: C

UCS Example



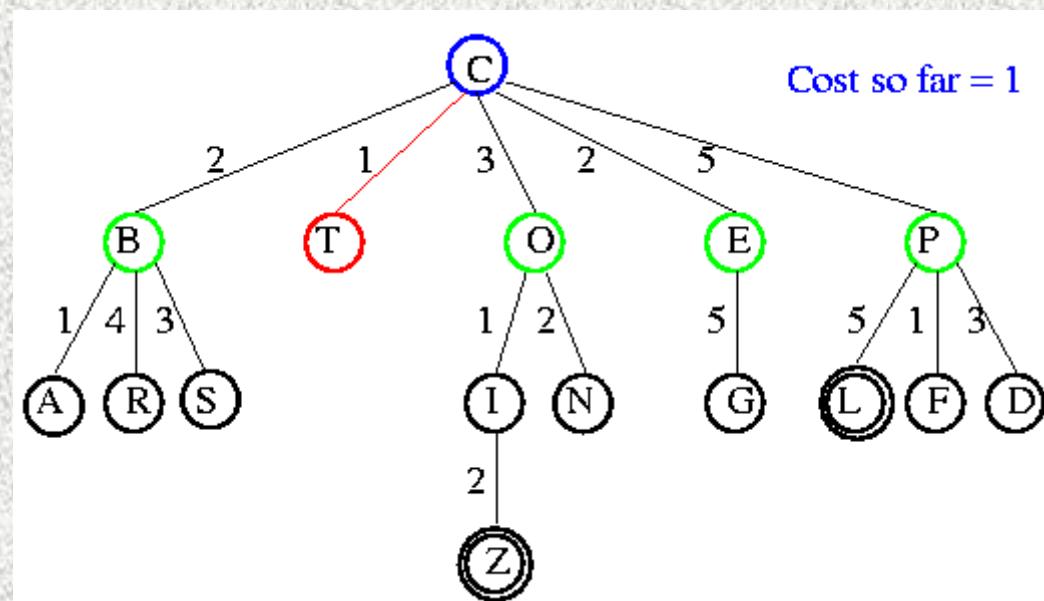
Open list: B(2) T(1) O(3) E(2) P(5)

UCS Example



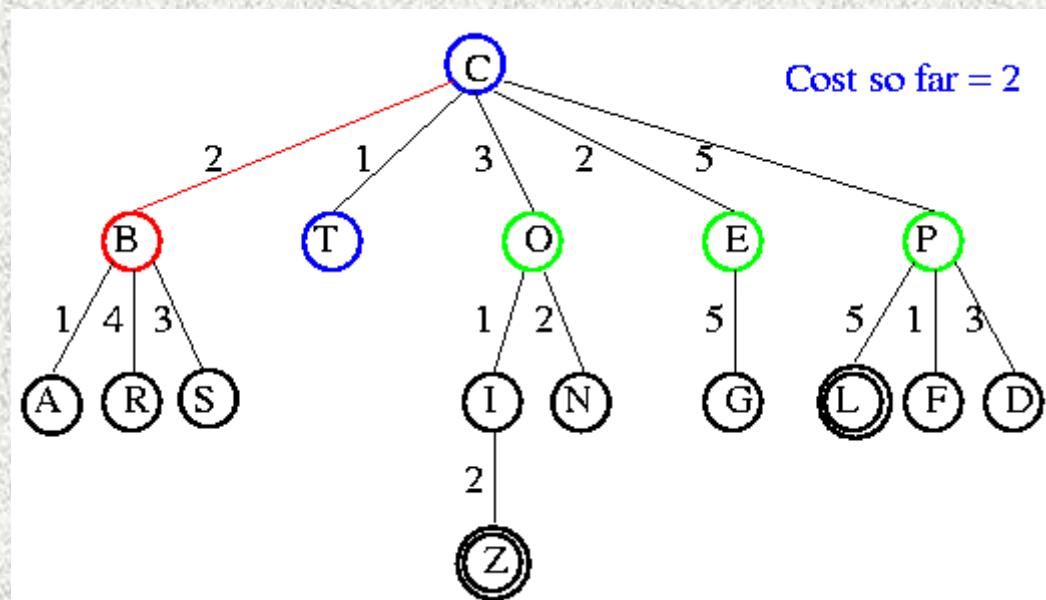
Open list: T(1) B(2) E(2) O(3) P(5)

UCS Example



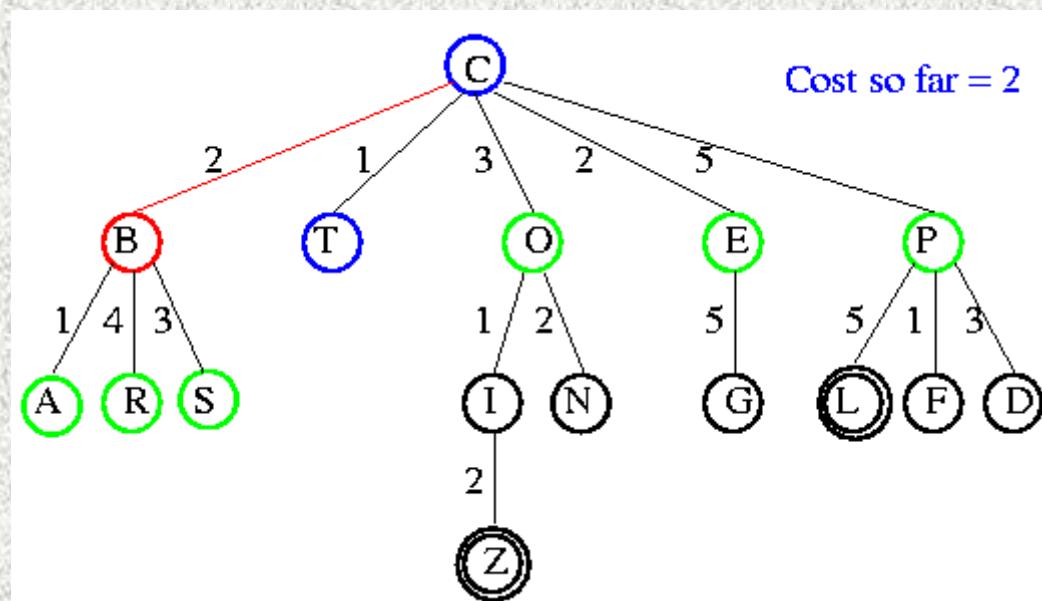
Open list: B(2) E(2) O(3) P(5)

UCS Example



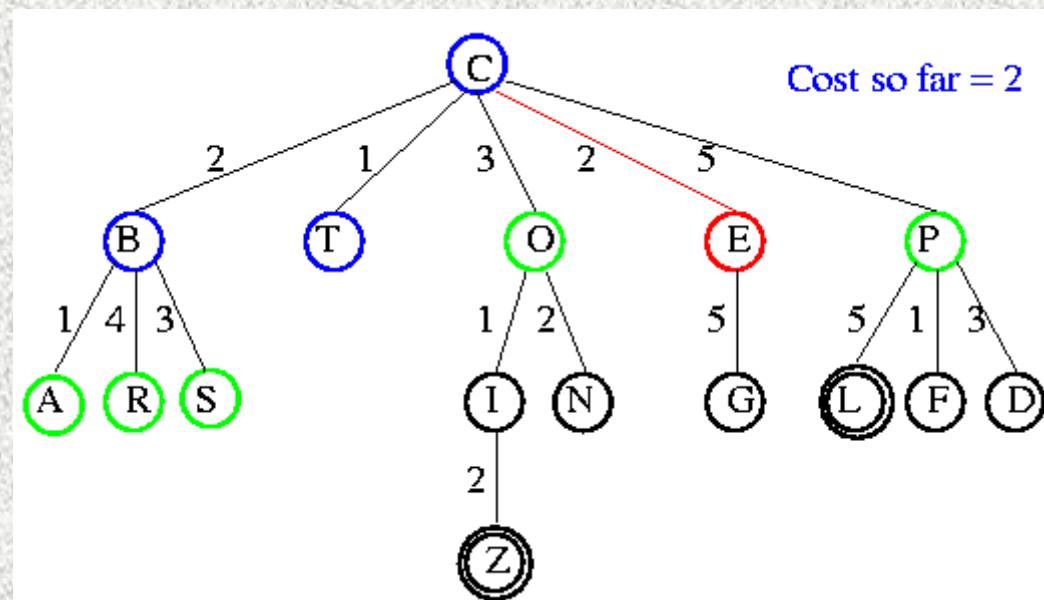
Open list: E(2) O(3) P(5)

UCS Example



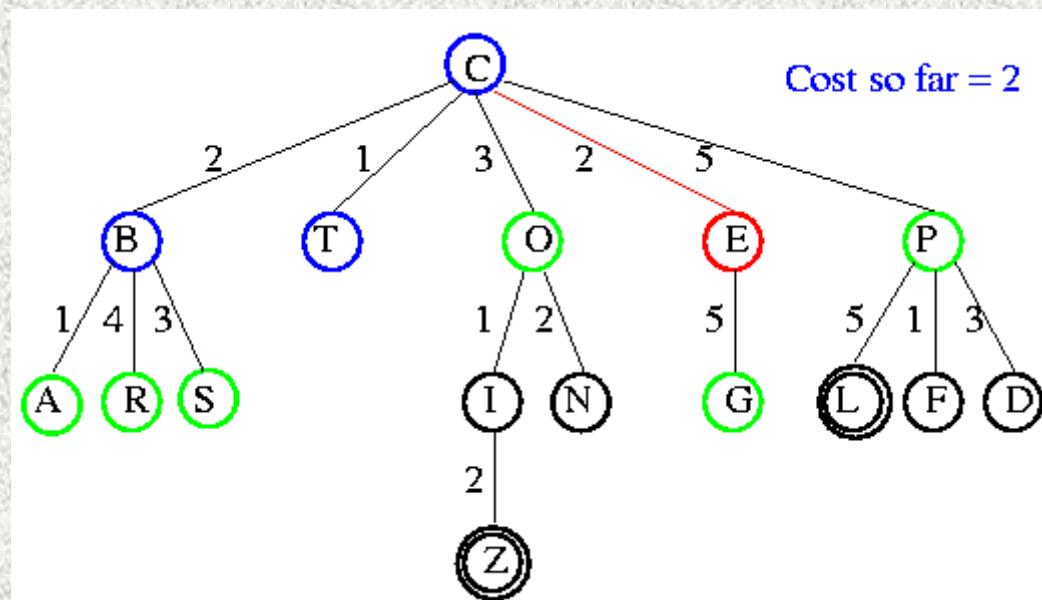
Open list: E(2) O(3) A(3) S(5) P(5) R(6)

UCS Example



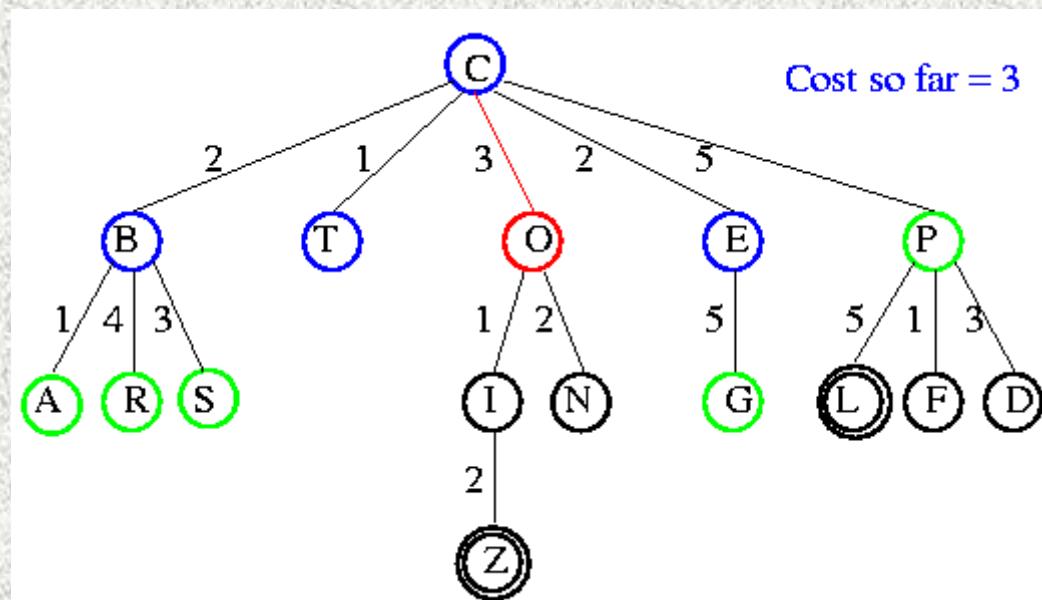
Open list: O(3) A(3) S(5) P(5) R(6)

UCS Example



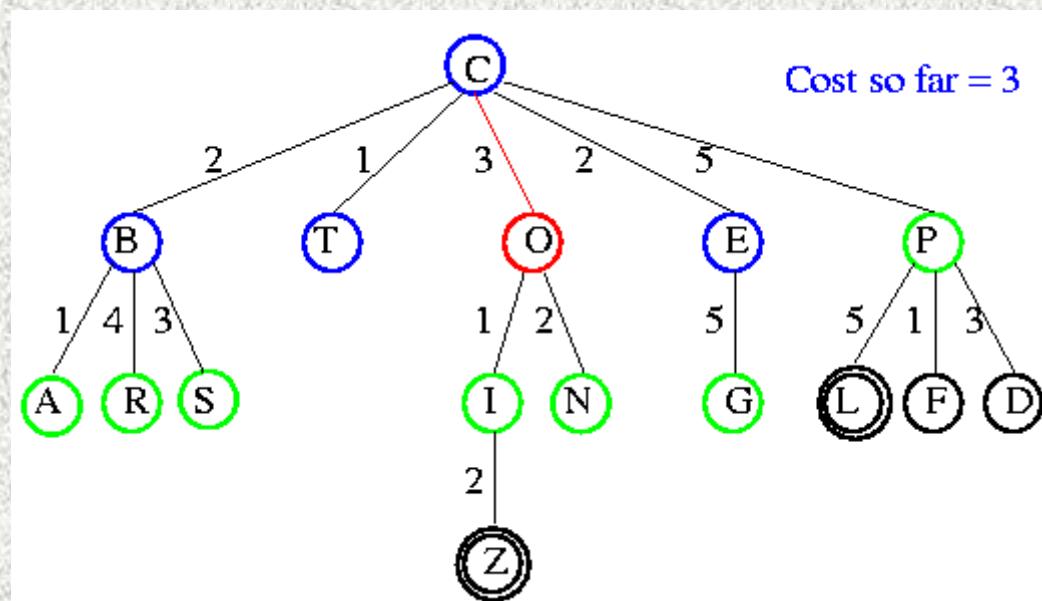
Open list: O(3) A(3) S(5) P(5) R(6) G(10)

UCS Example



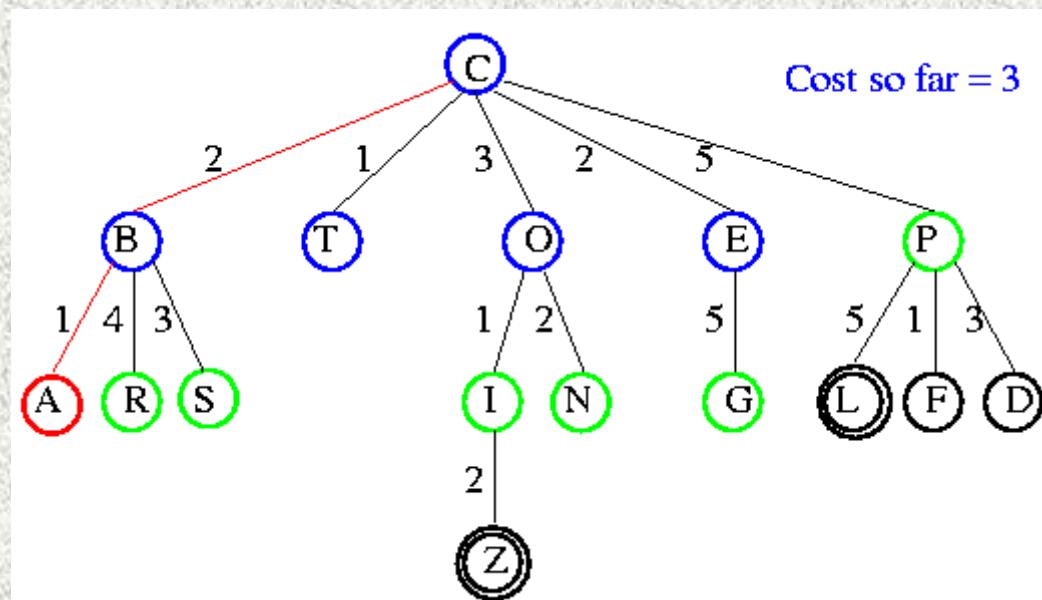
Open list: A(3) S(5) P(5) R(6) G(10)

UCS Example



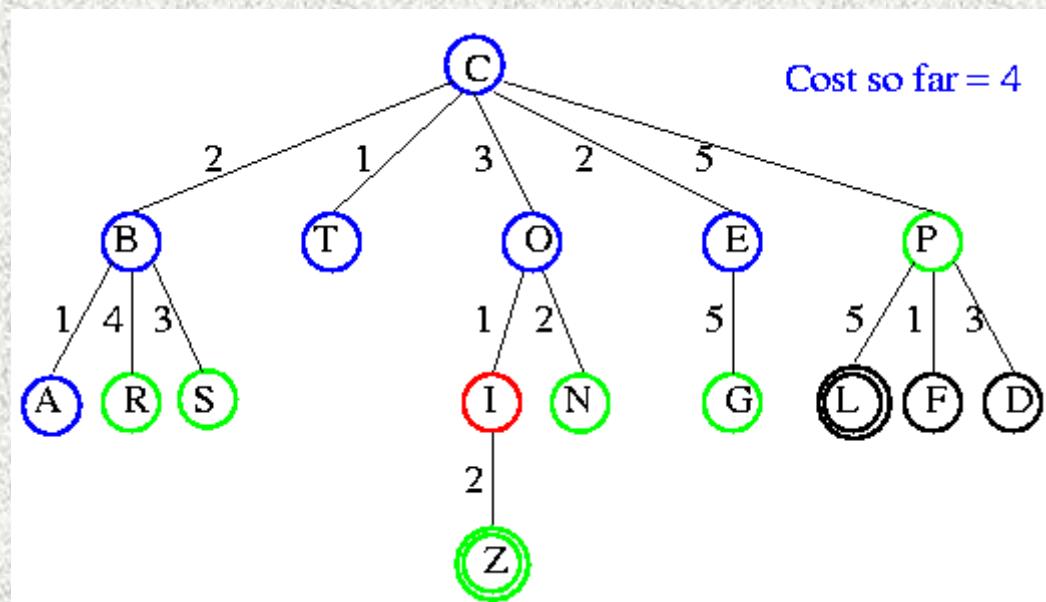
Open list: A(3) I(4) S(5) N(5) P(5) R(6) G(10)

UCS Example



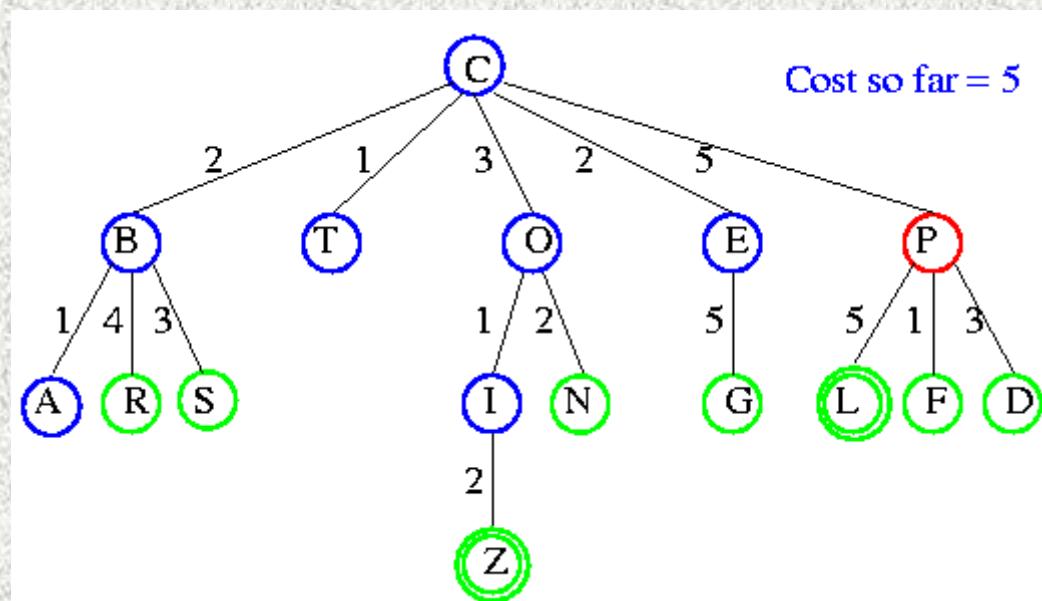
Open list: I(4) P(5) S(5) N(5) R(6) G(10)

UCS Example



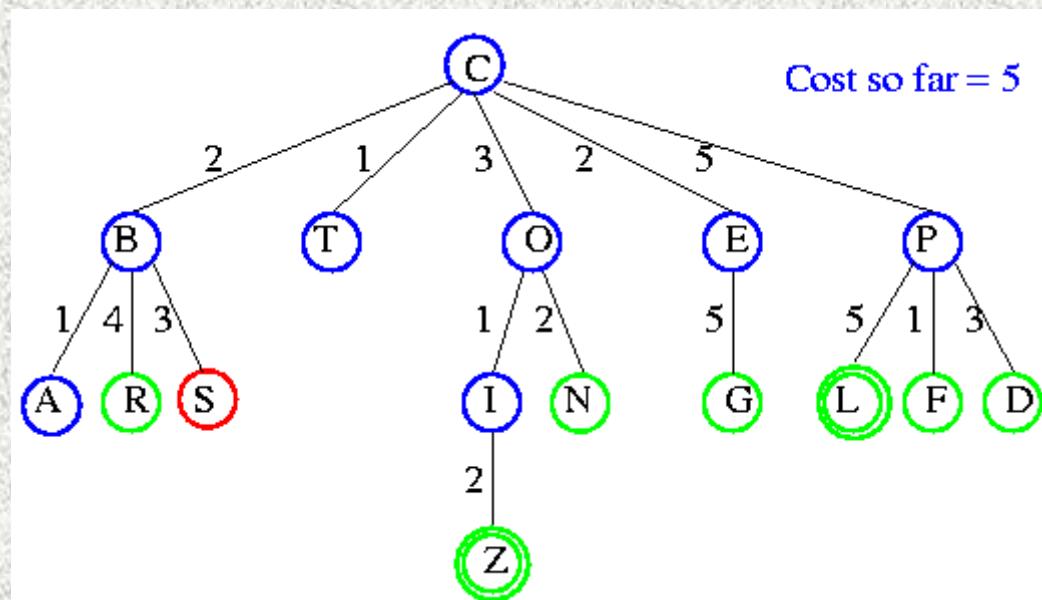
Open list: P(5) S(5) N(5) R(6) Z(6) G(10)

UCS Example



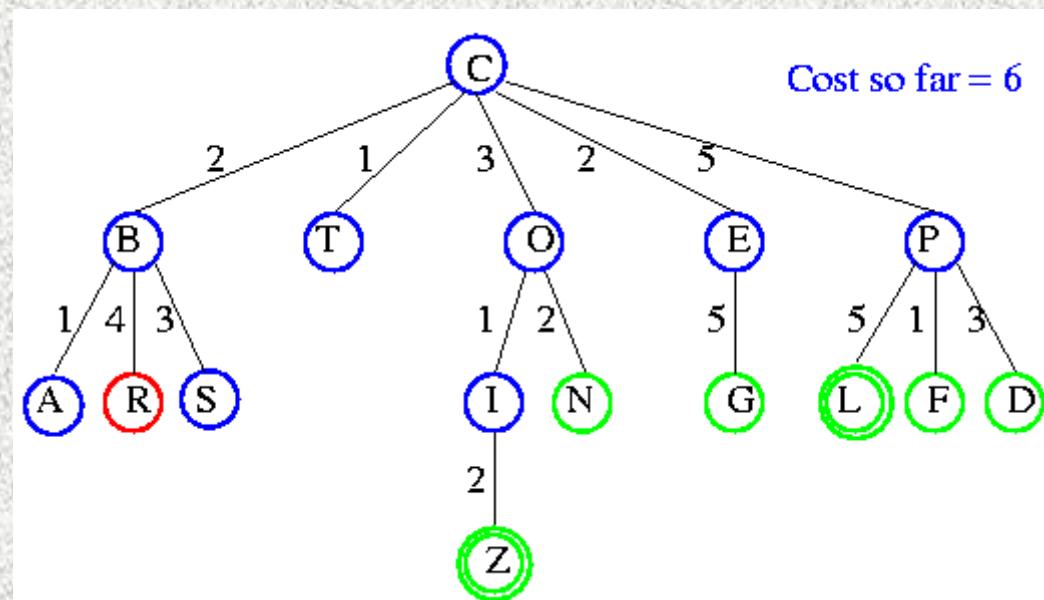
Open list: S(5) N(5) R(6) Z(6) F(6) D(8) G(10) L(10)

UCS Example



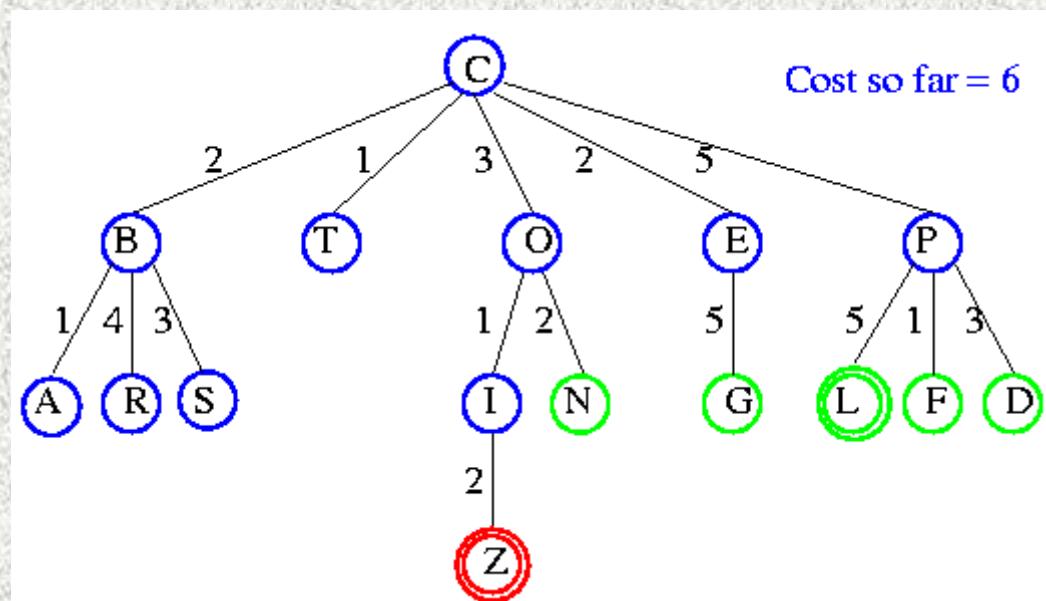
Open list: N(5) R(6) Z(6) F(6) D(8) G(10) L(10)

UCS Example



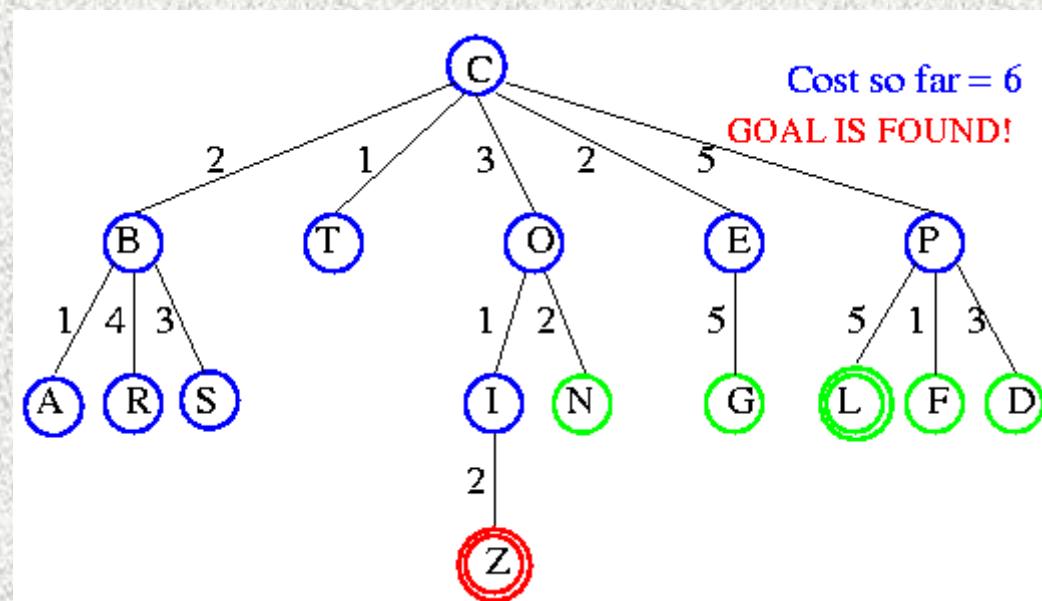
Open list: Z(6) F(6) D(8) G(10) L(10)

UCS Example



Open list: F(6) D(8) G(10) L(10)

UCS Example



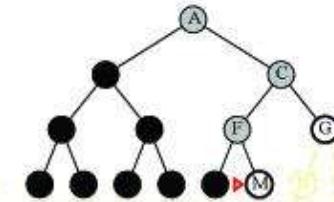
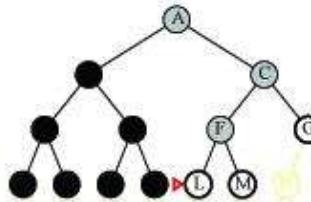
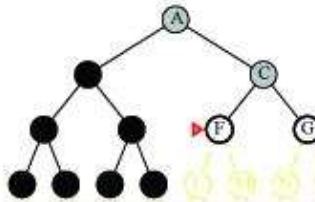
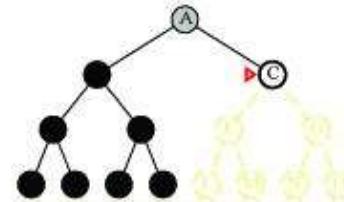
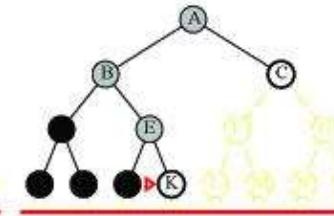
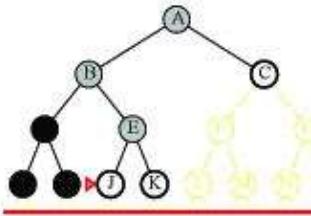
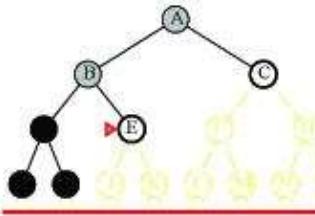
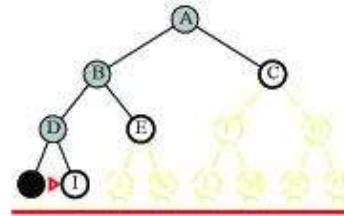
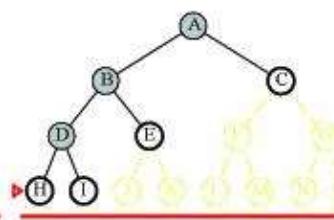
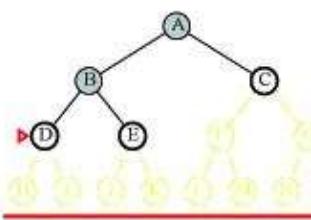
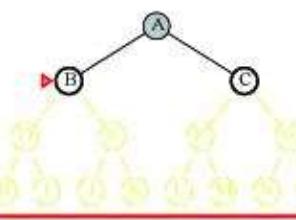
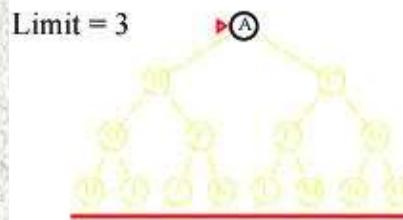
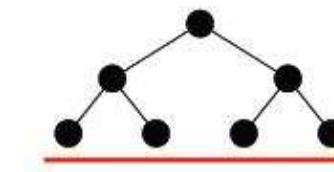
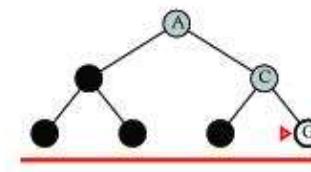
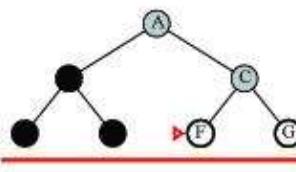
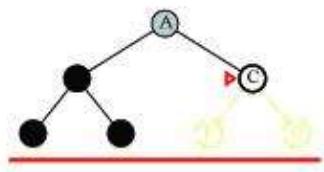
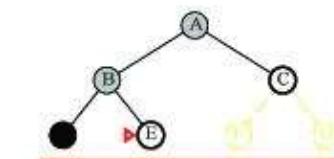
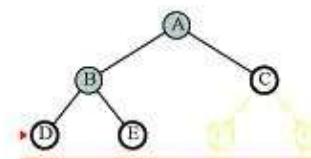
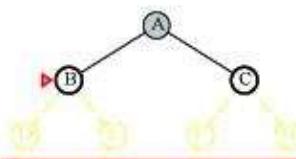
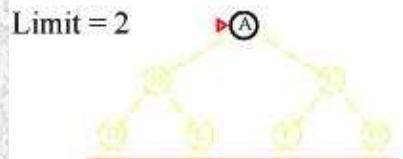
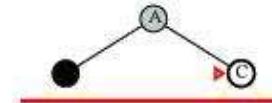
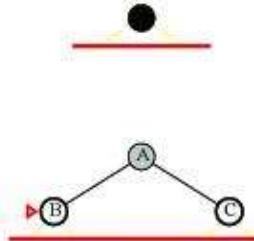
Comparison of Search Techniques

	DFS	BFS	UCS
Complete	N	Y	Y
Optimal	N	N	Y
Heuristic	N	N	N
Time	b^m	b^{d+1}	b^m
Space	bm	b^{d+1}	b^m



Iterative Deepening Search

- DFS with depth bound
- QueuingFn is enqueue at front as with DFS
 - Expand(state) only returns children such that $\text{depth}(\text{child}) \leq \text{threshold}$
 - This prevents search from going down infinite path
- First threshold is 1
 - If do not find solution, increment threshold and repeat



Analysis

- What about the repeated work?
- Time complexity (number of generated nodes)
 - $[b] + [b + b^2] + \dots + [b + b^2 + \dots + b^d]$
 - $(d)b + (d-1) b^2 + \dots + (1) b^d$
 - $O(b^d)$

Analysis

- Repeated work is approximately $1/b$ of total work
 - Negligible
 - Example: $b=10, d=5$
 - $N(\text{BFS}) = 1,111,100$
 - $N(\text{IDS}) = 123,450$
- Features
 - Shortest solution, not necessarily least cost
 - Is there a better way to decide threshold? (**IDA***)

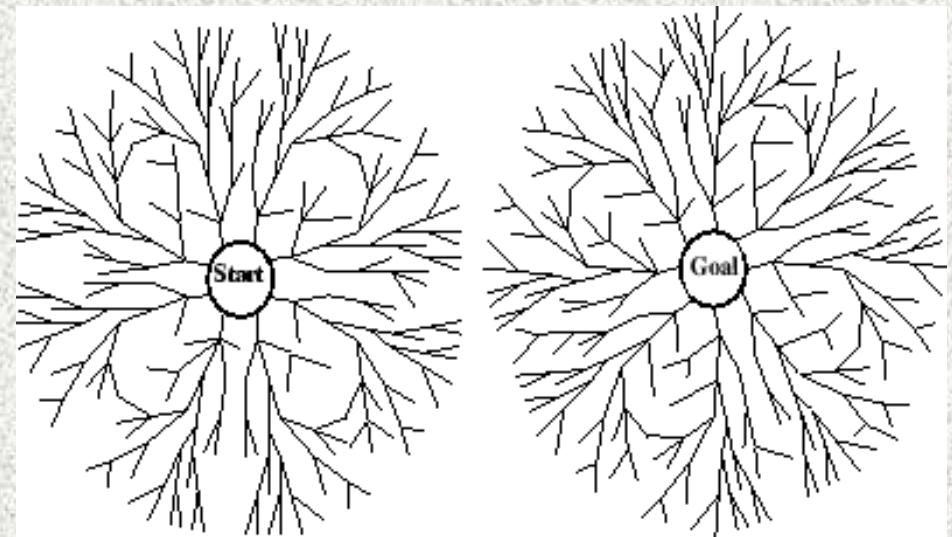
Comparison of Search Techniques

	DFS	BFS	UCS	IDS
Complete	N	Y	Y	Y
Optimal	N	N	Y	N
Heuristic	N	N	N	N
Time	b^m	b^{d+1}	b^m	b^d
Space	bm	b^{d+1}	b^m	bd



Bidirectional Search

- Search forward from initial state to goal AND backward from goal state to initial state
- Can prune many options
- Considerations
 - Which goal state(s) to use
 - How determine when searches overlap
 - Which search to use for each direction
 - Here, two BFS searches
- Time and space is $O(b^{d/2})$



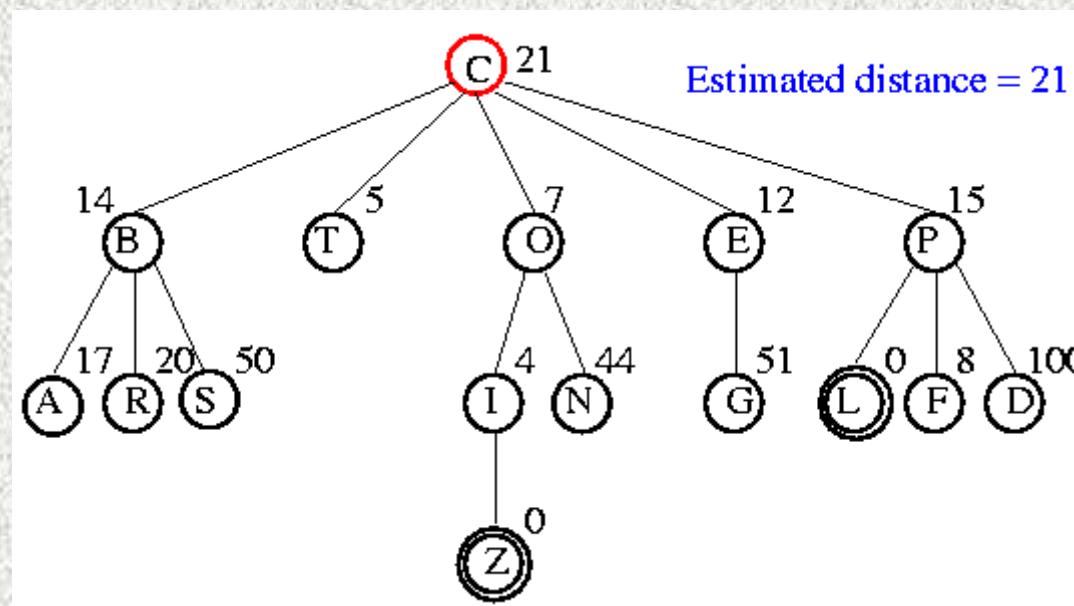
Informed Searches

- Best-first search, Hill climbing, Beam search, A*, IDA*, RBFS, SMA*
- New terms
 - Heuristics
 - Optimal solution
 - Informedness
 - Hill climbing problems
 - Admissibility
- New parameters
 - $g(n)$ = estimated cost from initial state to state n
 - $h(n)$ = estimated cost (distance) from state n to closest goal
 - $h(n)$ is our heuristic
 - Robot path planning, $h(n)$ could be Euclidean distance
 - 8 puzzle, $h(n)$ could be #tiles out of place
- Search algorithms which use $h(n)$ to guide search are **heuristic search algorithms**

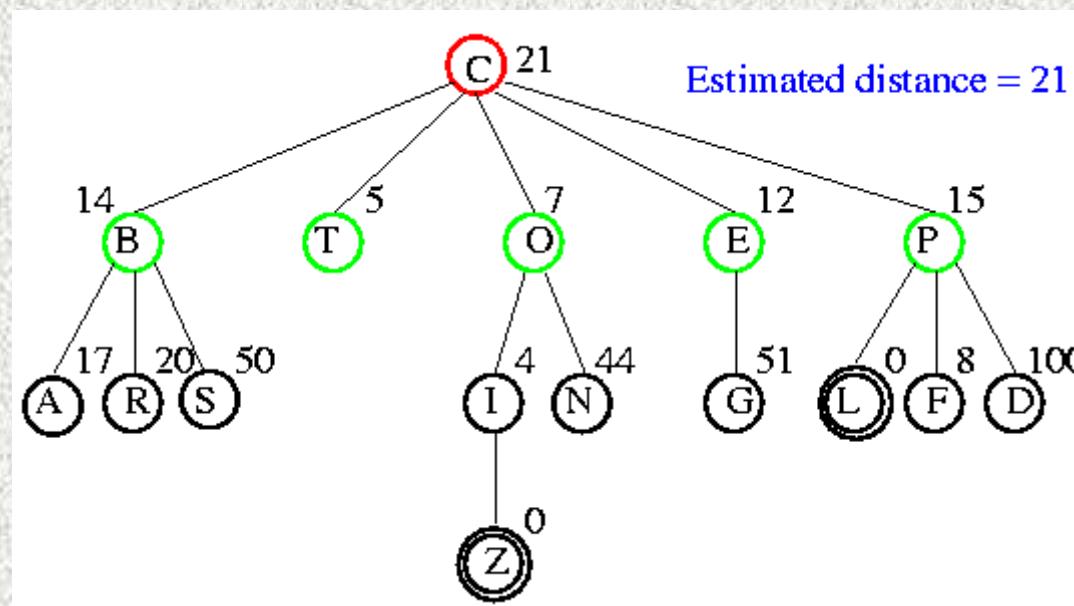
Best-First Search

- QueueingFn is sort-by-h
- Best-first search only as good as heuristic
 - Example heuristic for 8 puzzle:
Manhattan Distance

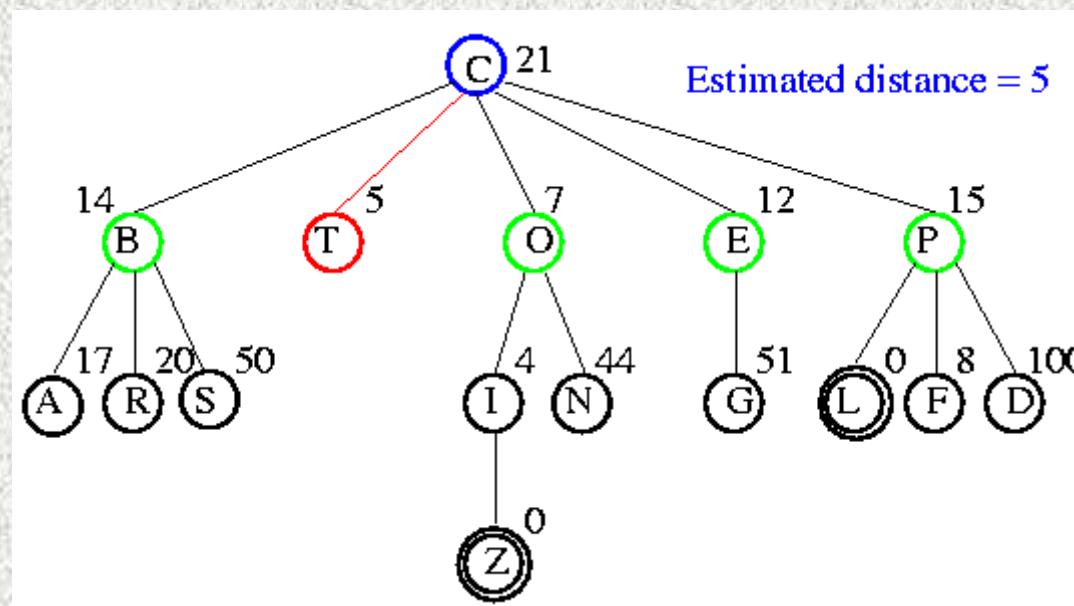
Example



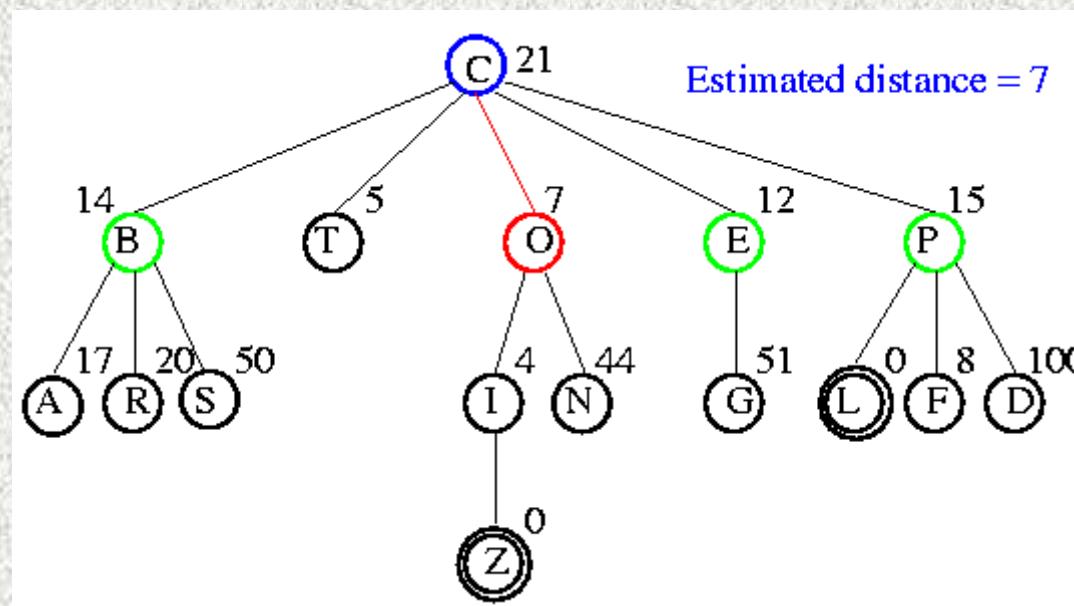
Example



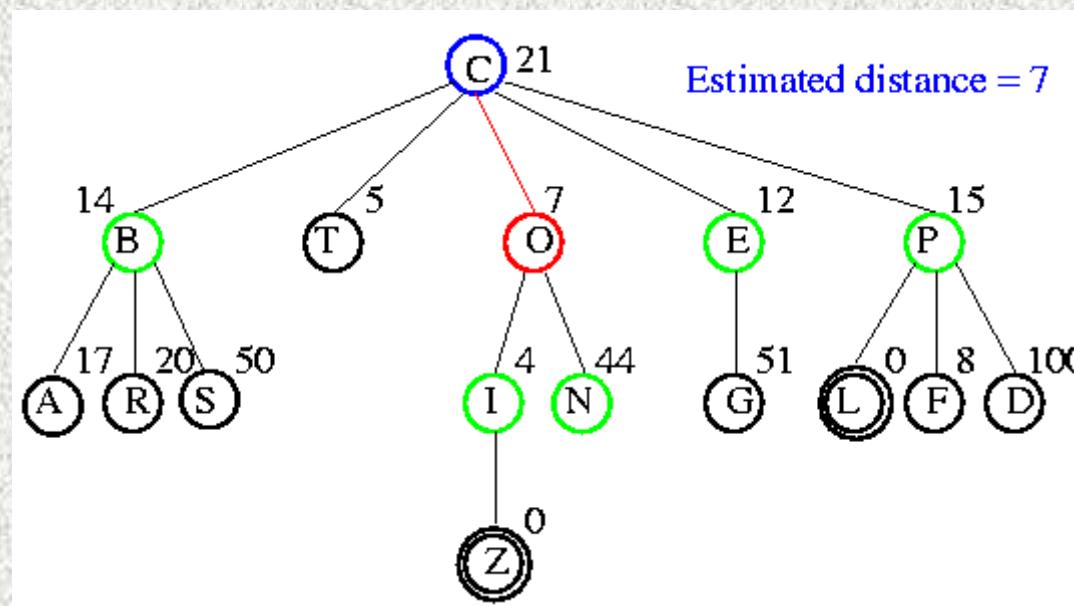
Example



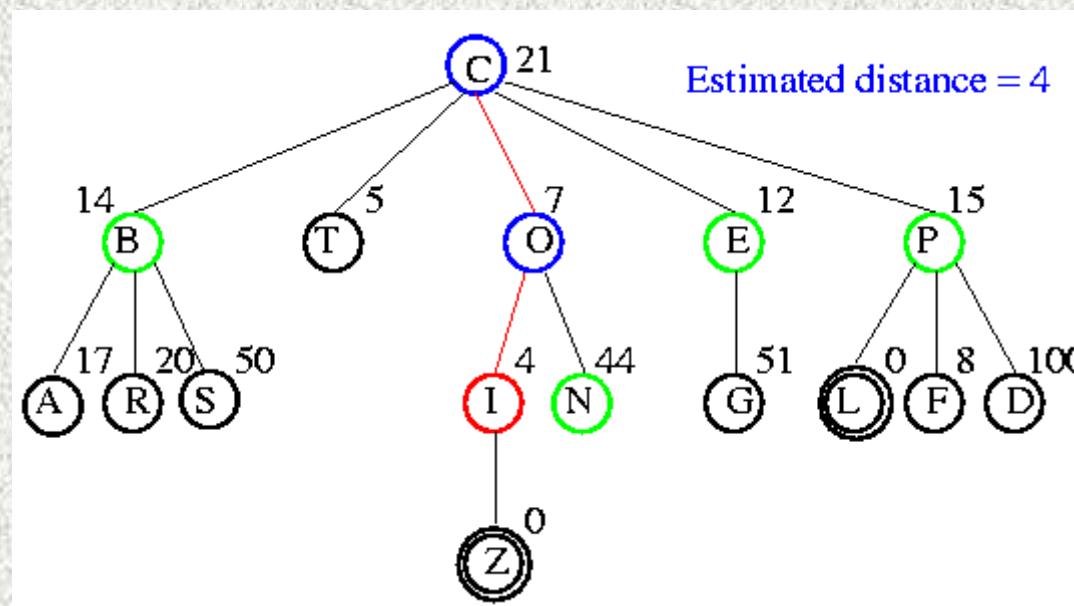
Example



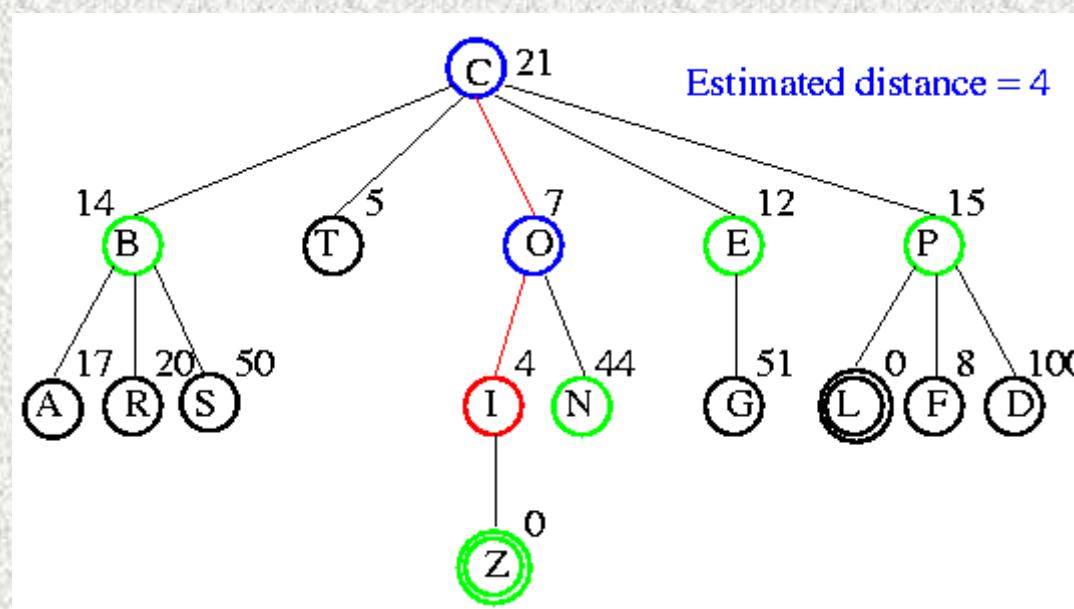
Example



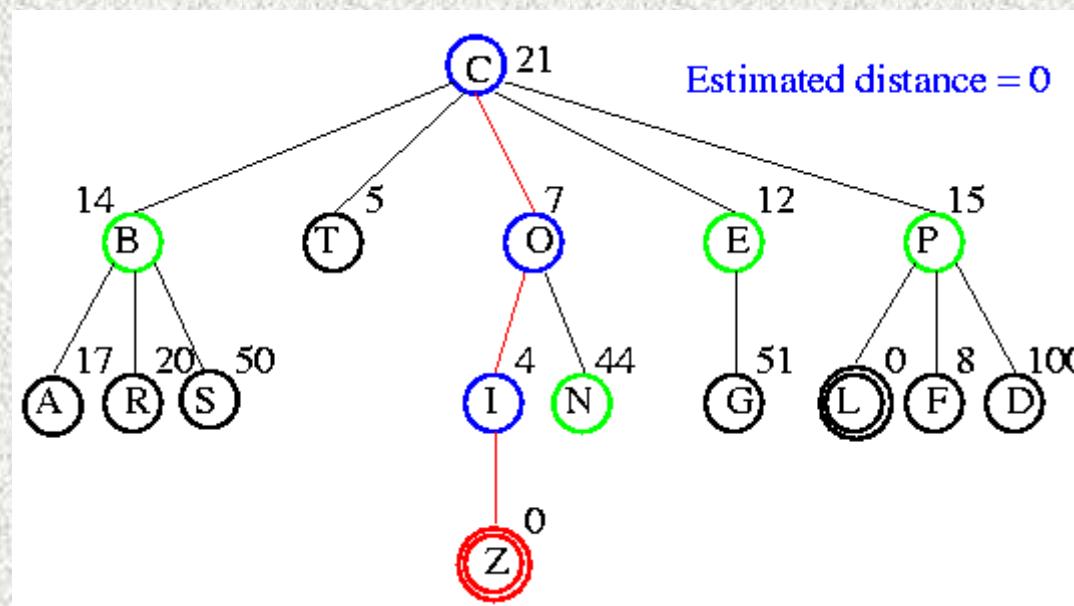
Example



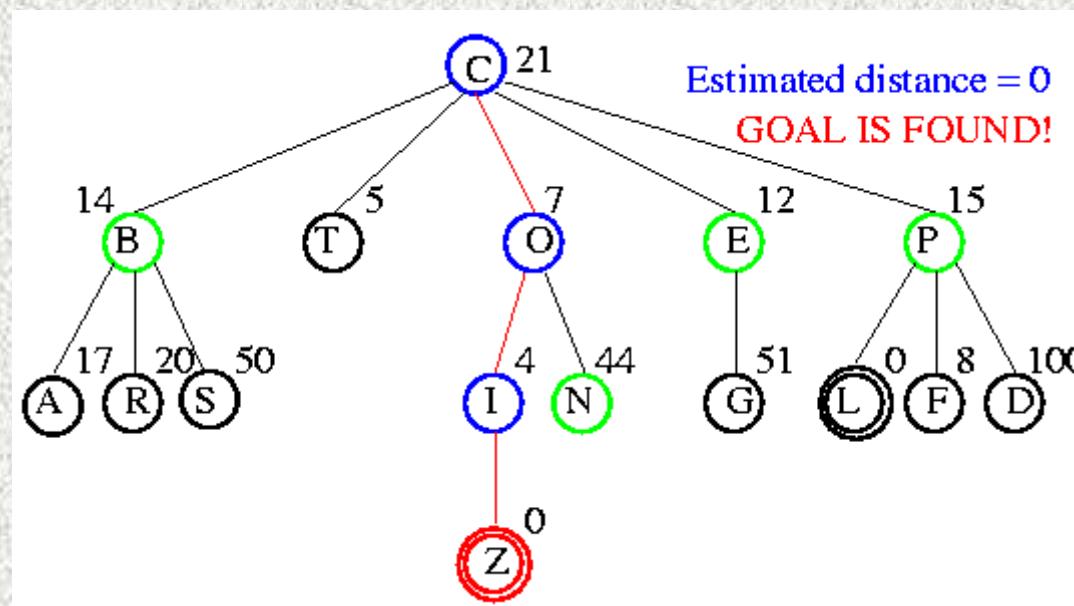
Example



Example



Example



Comparison of Search Techniques

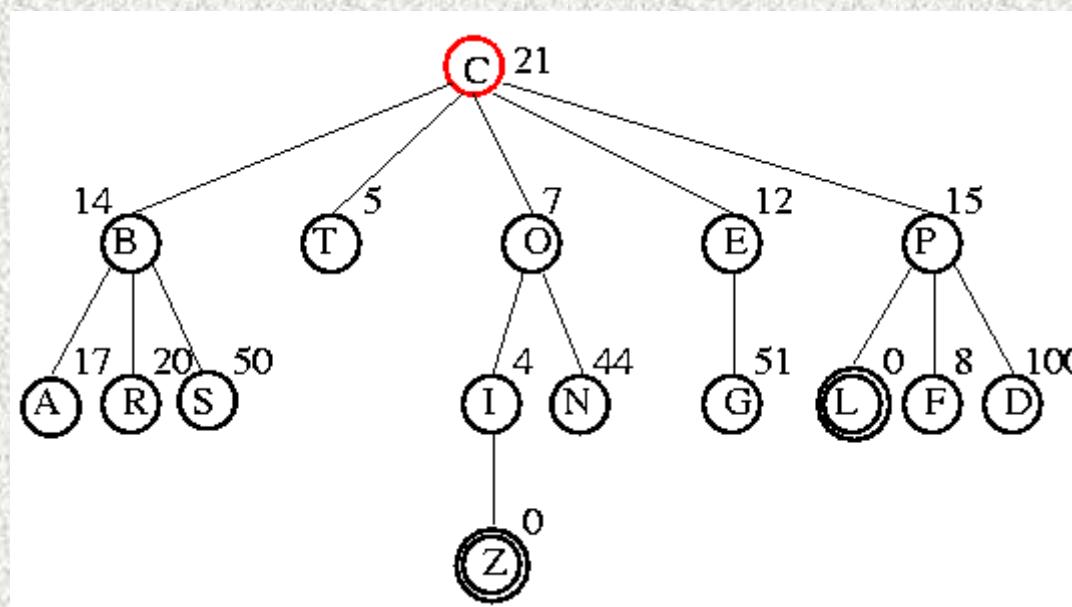
	DFS	BFS	UCS	IDS	Best
Complete	N	Y	Y	Y	N
Optimal	N	N	Y	N	N
Heuristic	N	N	N	N	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m
Space	bm	b^{d+1}	b^m	bd	b^m



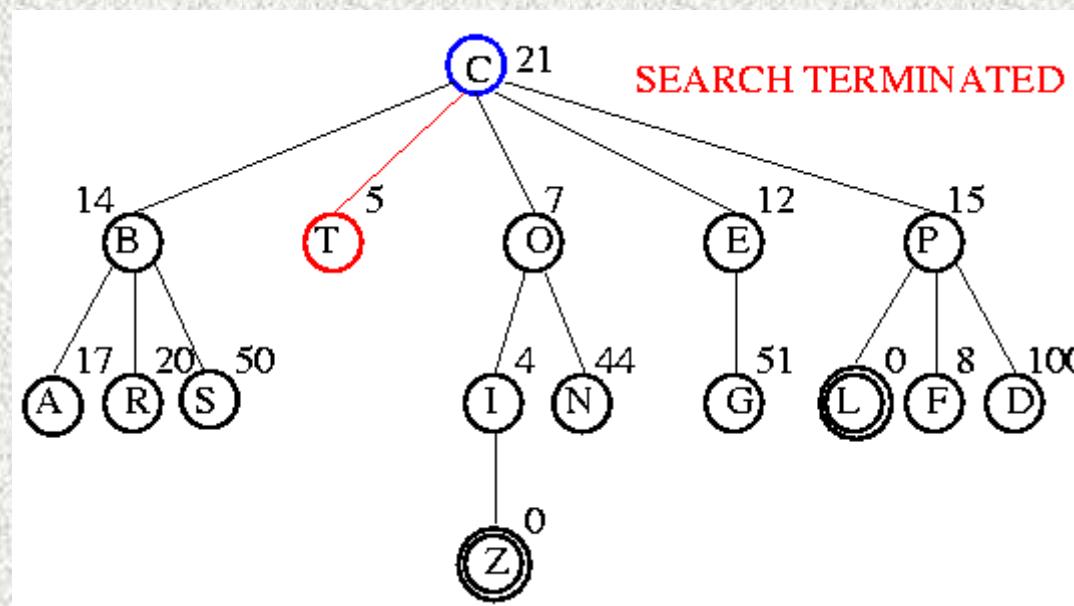
Hill Climbing (Greedy Search)

- QueueingFn is sort-by-h
 - Only keep lowest-h state on open list
- Best-first search is **tentative**
- Hill climbing is **irrevocable**
- Features
 - Much faster
 - Less memory
 - Dependent upon $h(n)$
 - If bad $h(n)$, may prune away all goals
 - Not complete

Example

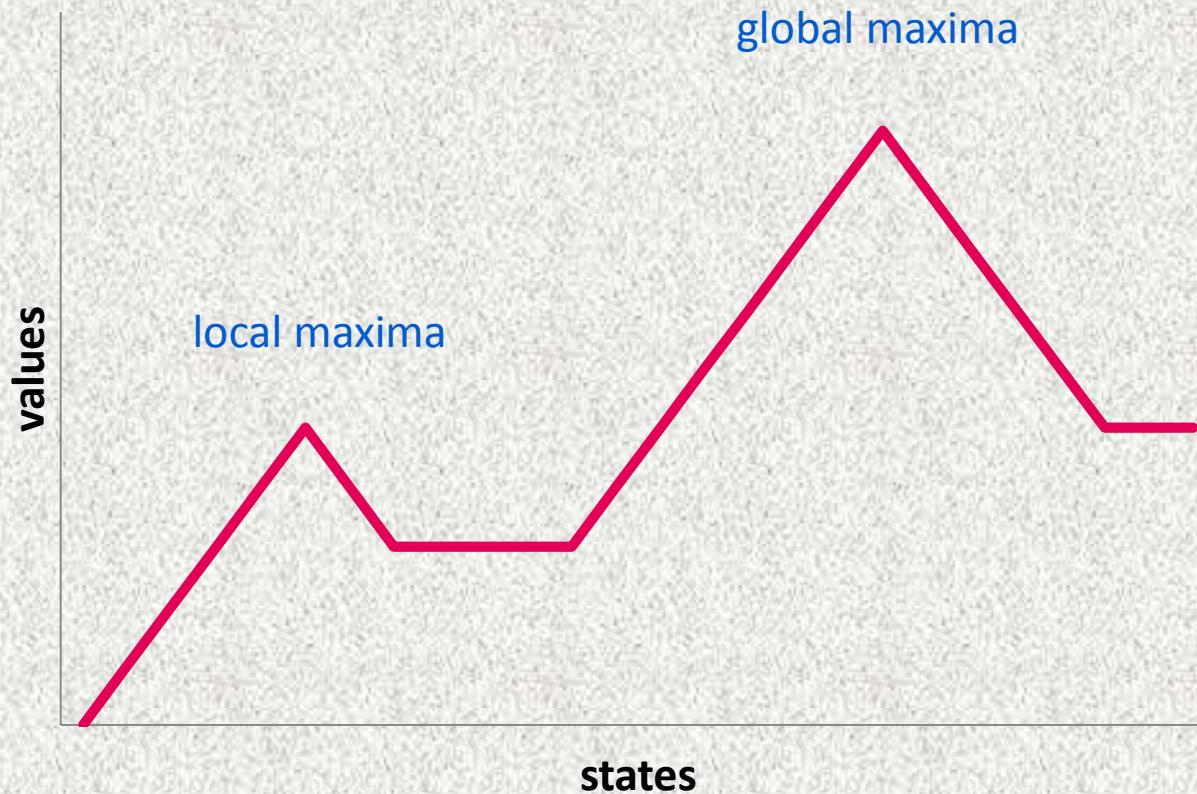


Example



Hill Climbing Issues

- Also referred to as gradient descent
- Foothill problem / local maxima / local minima
- Can be solved with random walk or more steps
- Other problems: ridges, plateaus



Comparison of Search Techniques

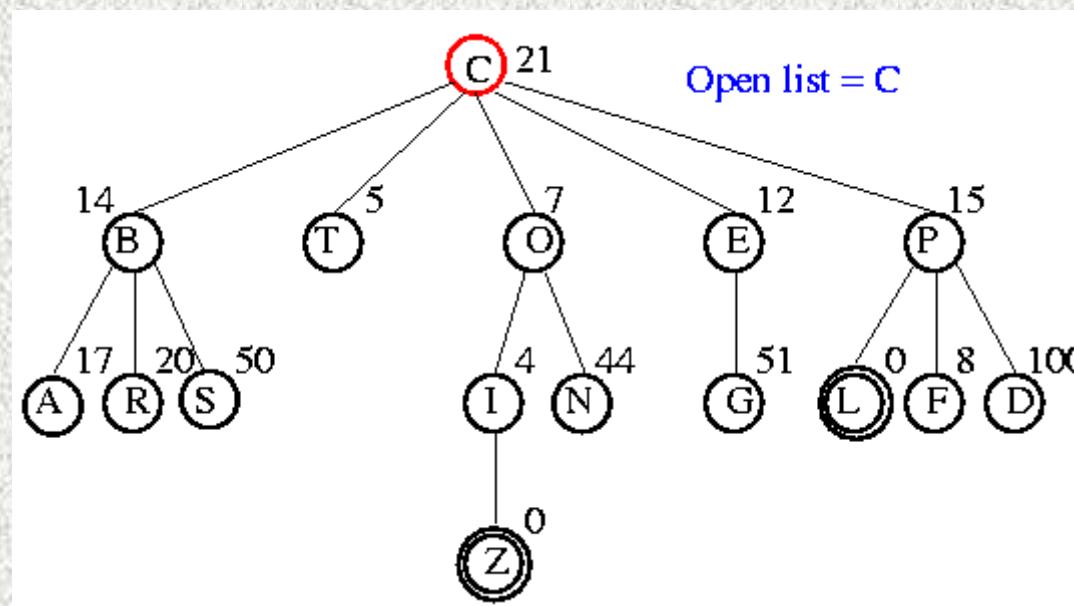
	DFS	BFS	UCS	IDS	Best	HC
Complete	N	Y	Y	Y	N	N
Optimal	N	N	Y	N	N	N
Heuristic	N	N	N	N	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	mn
Space	bm	b^{d+1}	b^m	bd	b^m	b



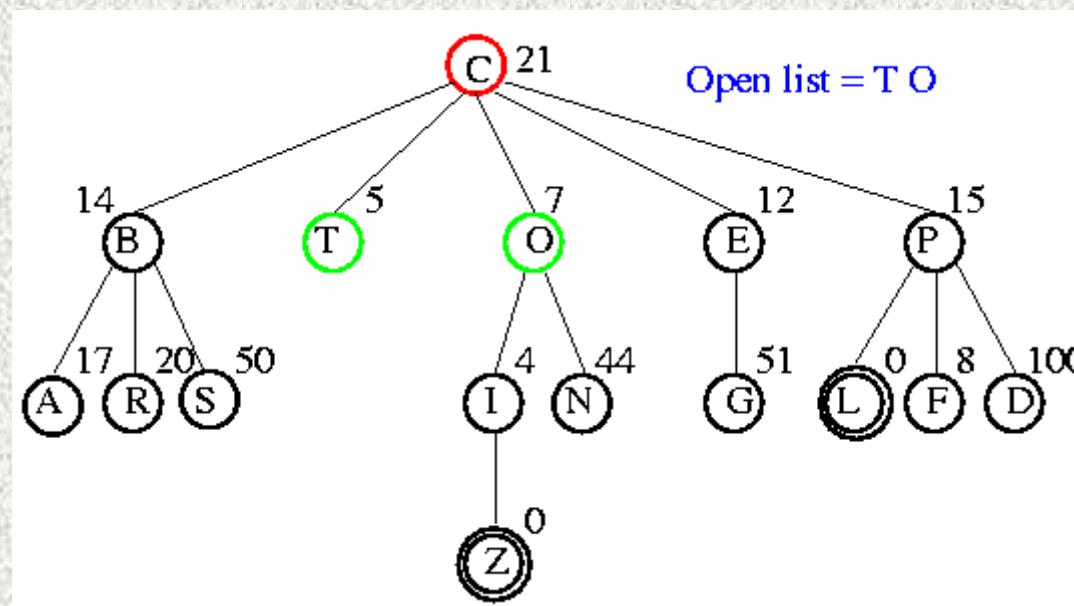
Beam Search

- QueueingFn is sort-by-h
 - Only keep best (lowest-h) n nodes on open list
- n is the “beam width”
 - n = 1, Hill climbing
 - n = infinity, Best first search

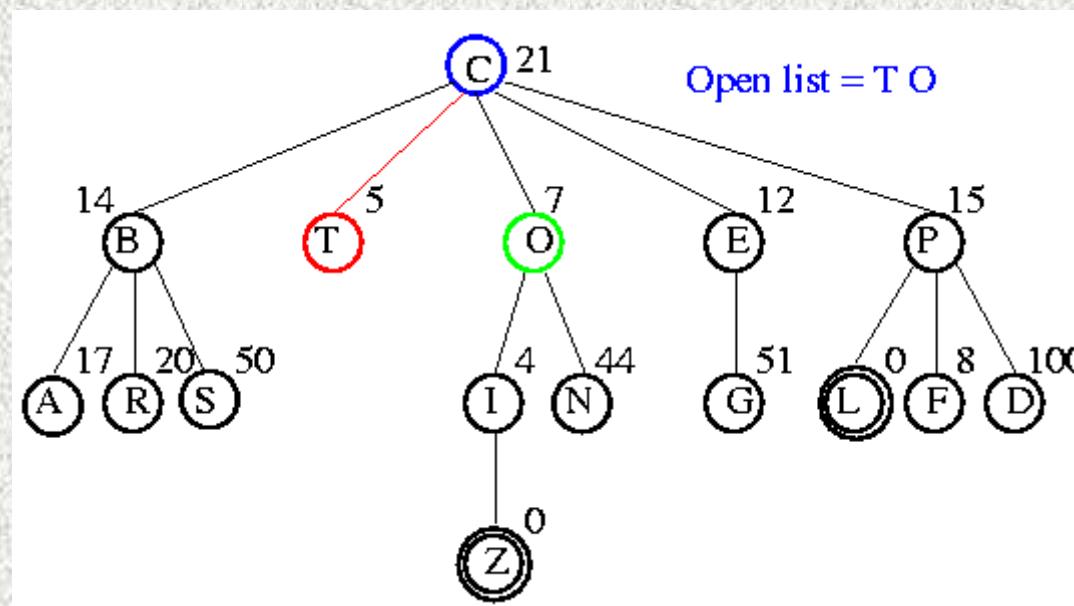
Example



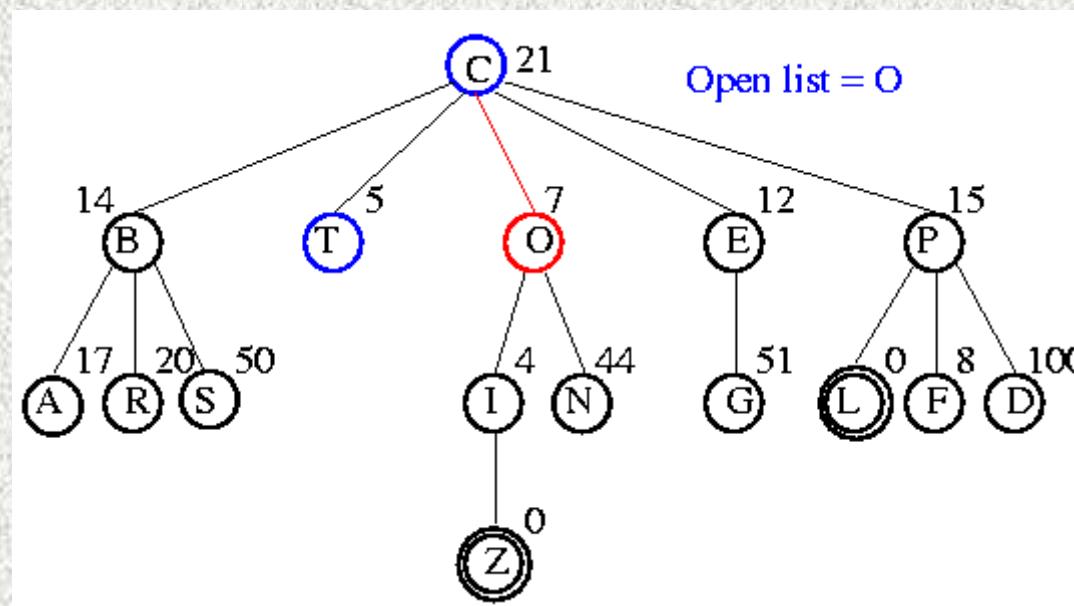
Example



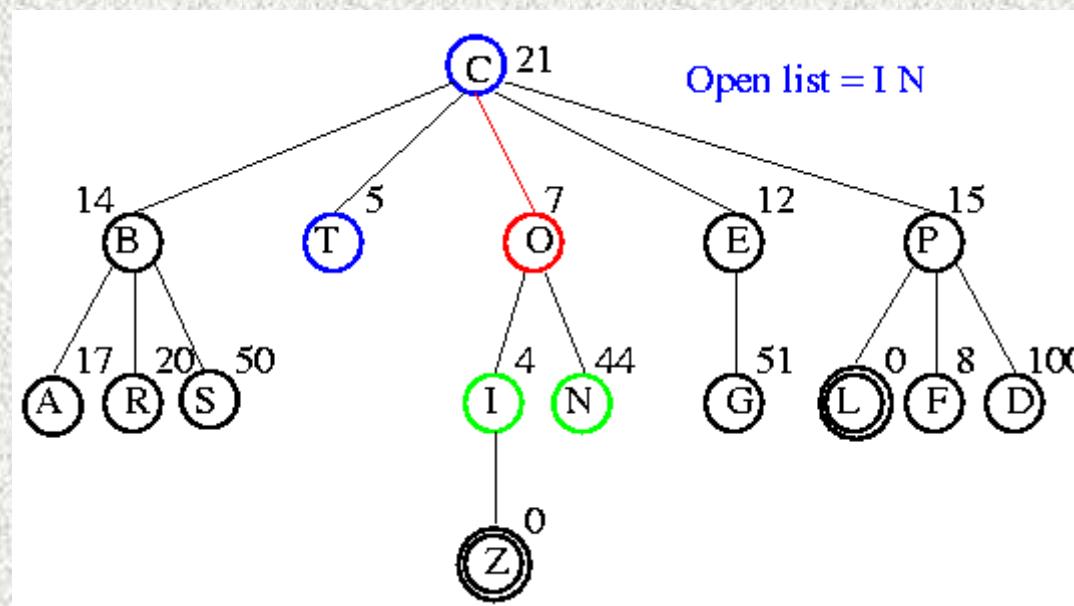
Example



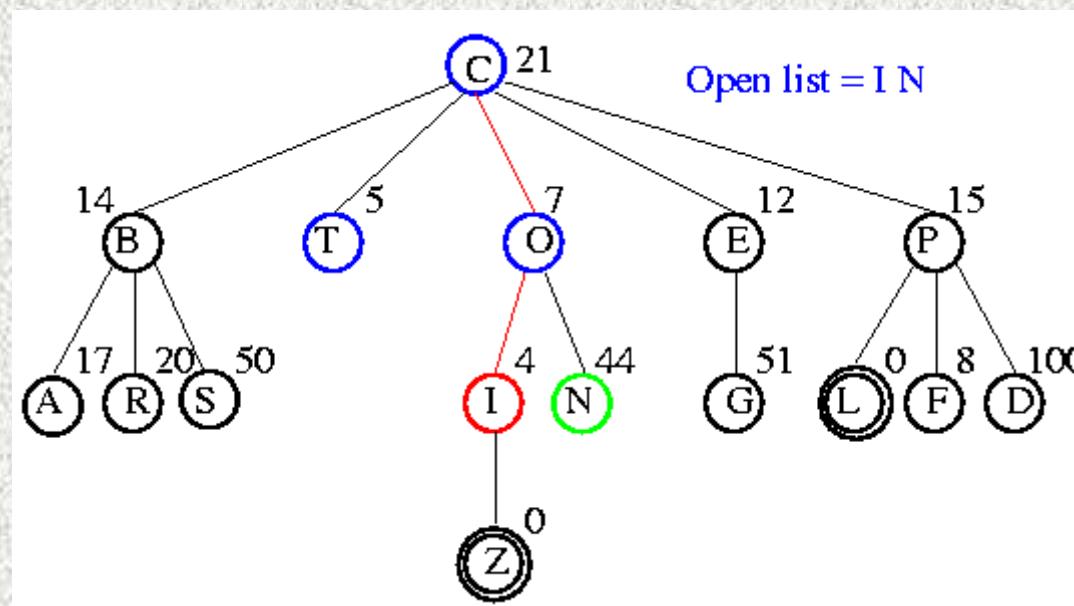
Example



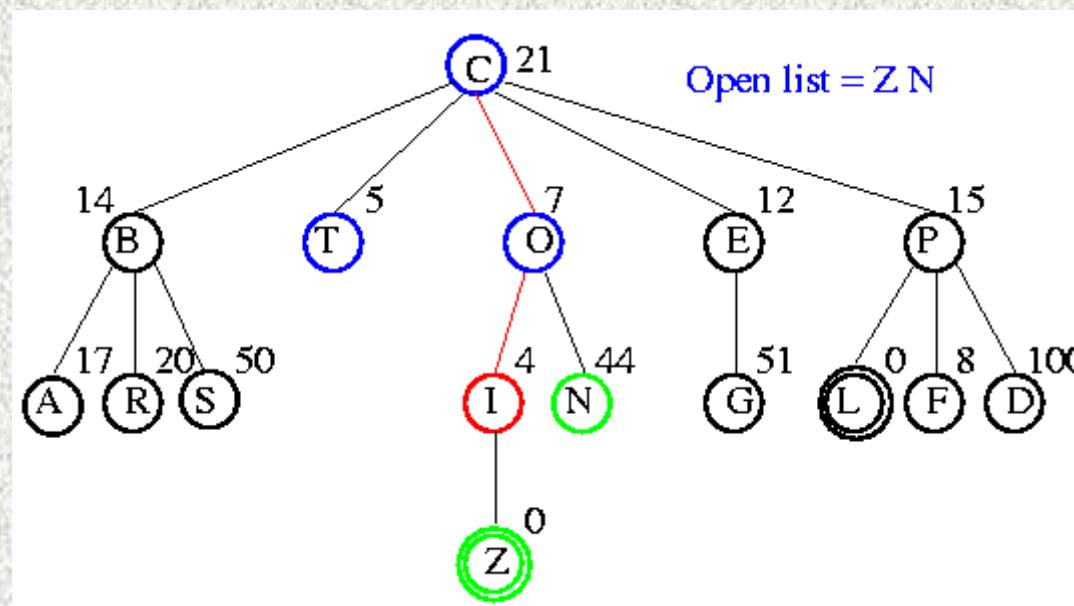
Example



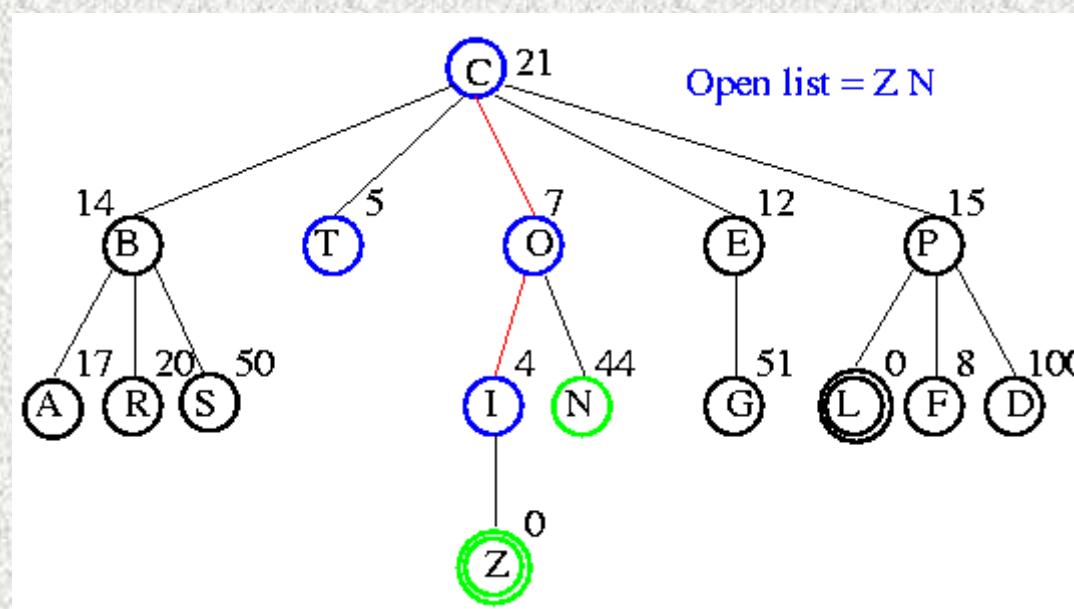
Example



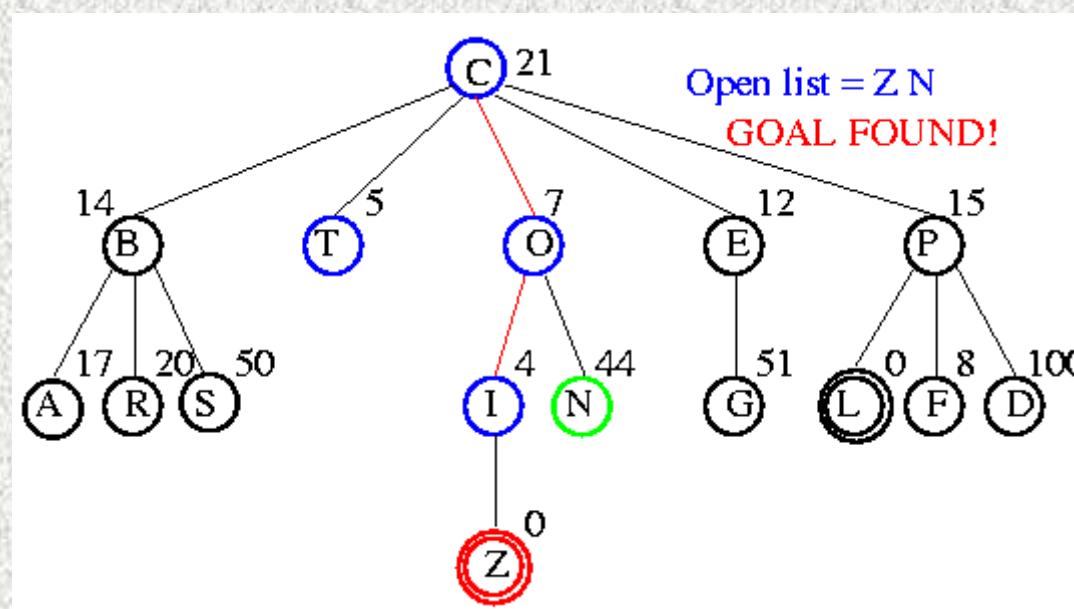
Example



Example



Example



Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best	HC	Beam
Complete	N	Y	Y	Y	N	N	N
Optimal	N	N	Y	N	N	N	N
Heuristic	N	N	N	N	Y	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm	nm
Space	bm	b^{d+1}	b^m	bd	b^m	b	bn



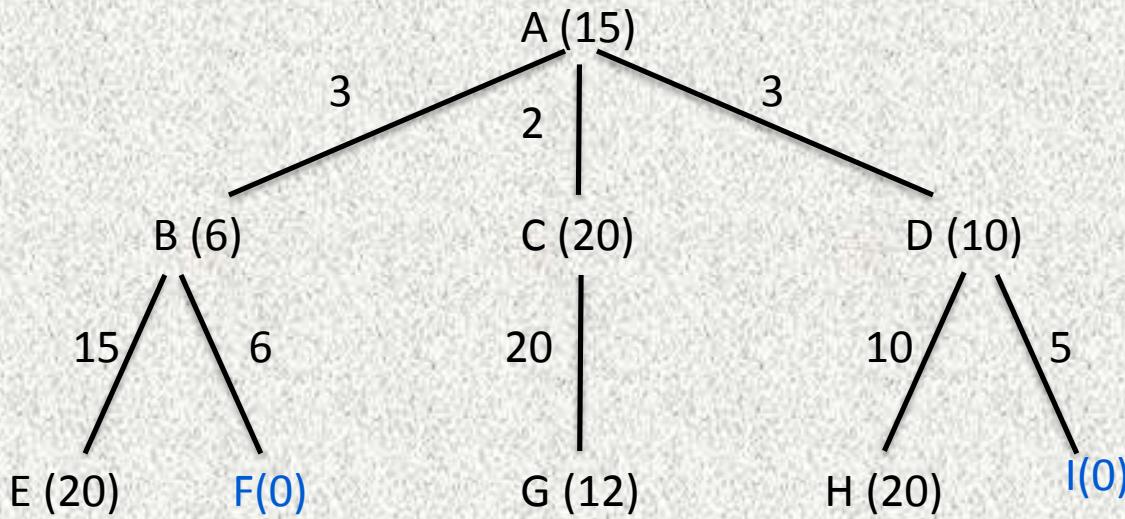
A*

- QueueingFn is sort-by-f
 - $f(n) = g(n) + h(n)$
- Note that UCS and Best-first both improve search
 - UCS keeps solution cost low
 - Best-first helps find solution quickly
- A* combines these approaches

Power of f

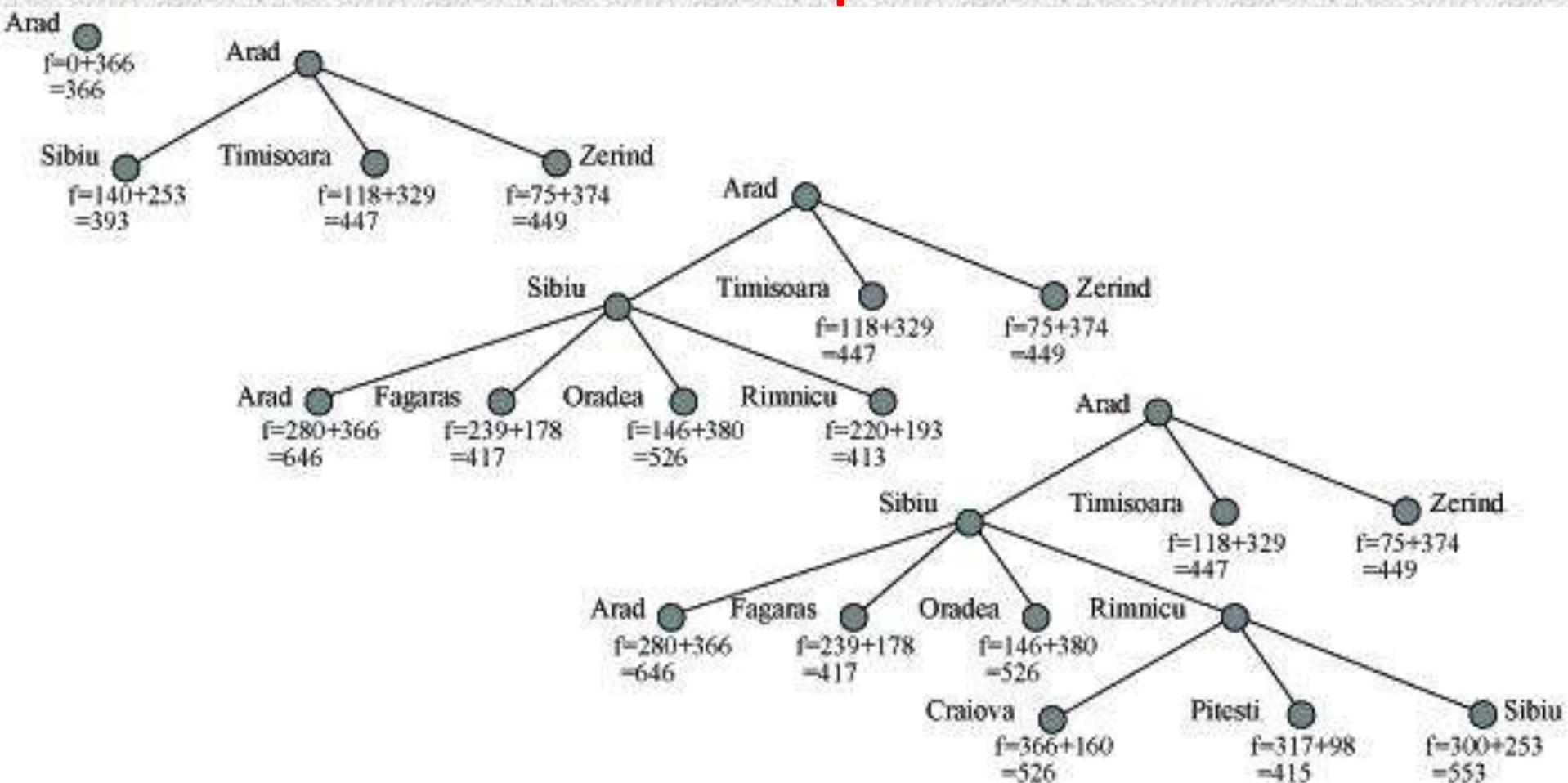
- If heuristic function is wrong it either
 - overestimates (guesses too high)
 - underestimates (guesses too low)
- Overestimating is worse than underestimating
- A* returns optimal solution if $h(n)$ is **admissible**
 - heuristic function is **admissible** if never overestimates true cost to nearest goal
 - if search finds optimal solution using admissible heuristic, the search is **admissible**

Overestimating



- Solution cost:
 - ABF = 9
 - ADI = 8
- Open list:
 - A (15) B (9) F (9)
- Missed optimal solution

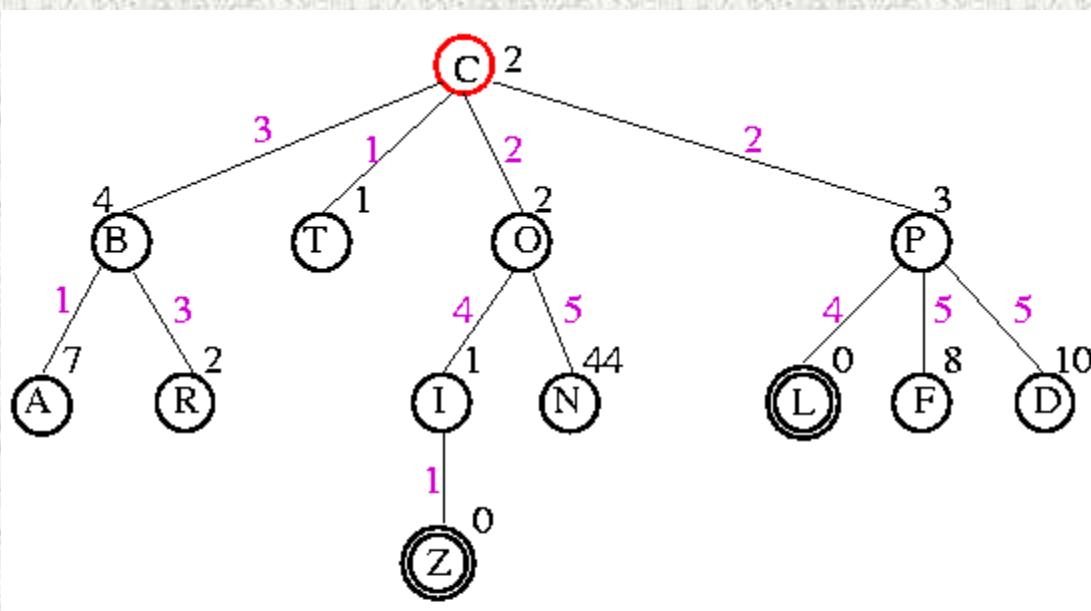
Example



[A* applied to 8 puzzle](#)

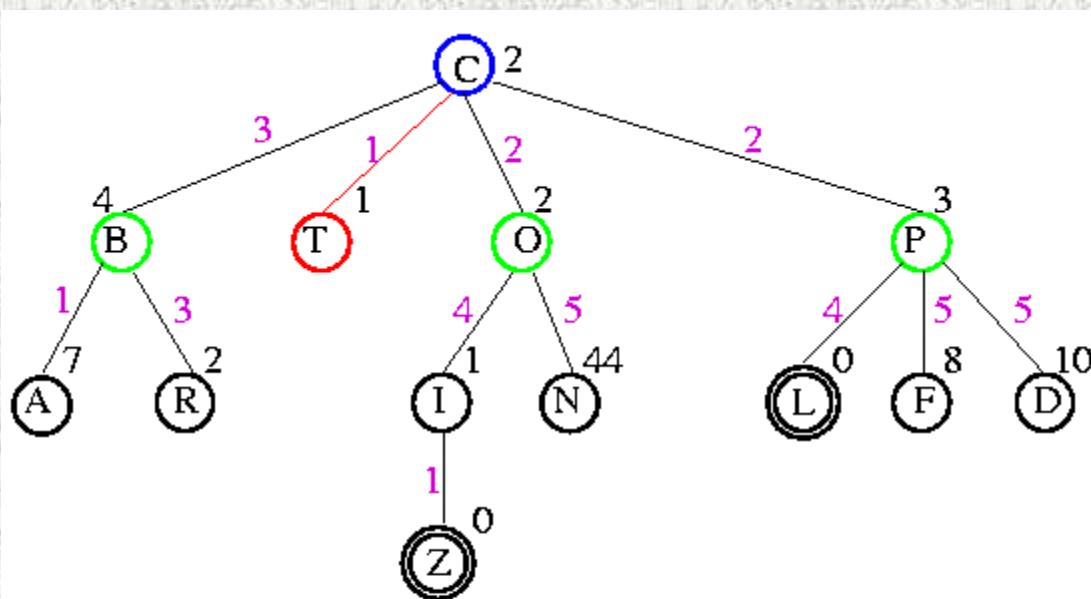
[A* search applet](#)

Example



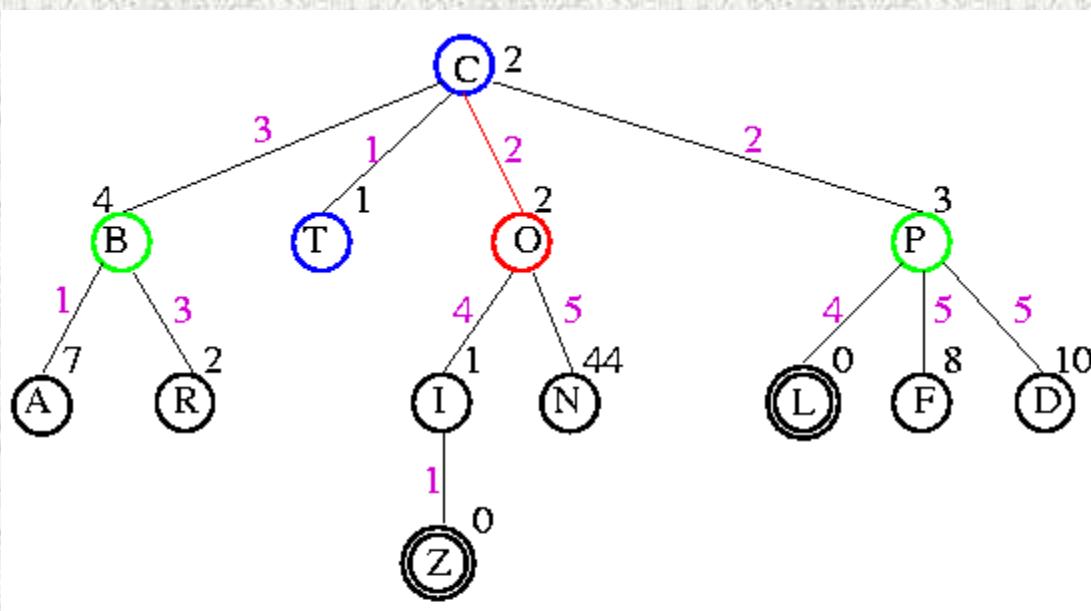
Open List = C (0+2=2)

Example



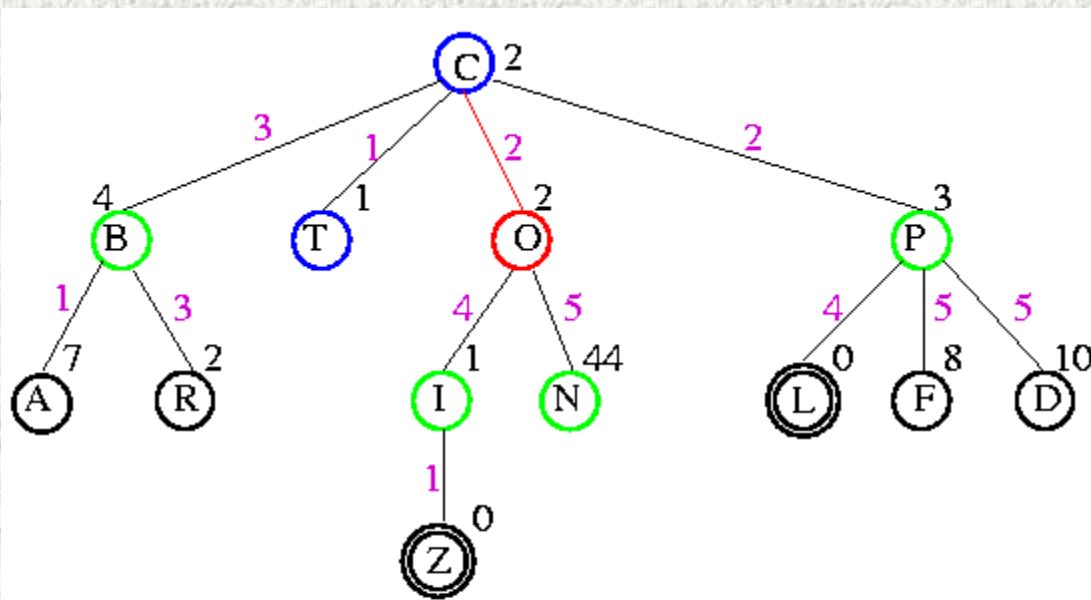
Open List = T ($1+1=2$), O ($2+2=4$), P ($2+3=5$), B ($3+4=7$)

Example



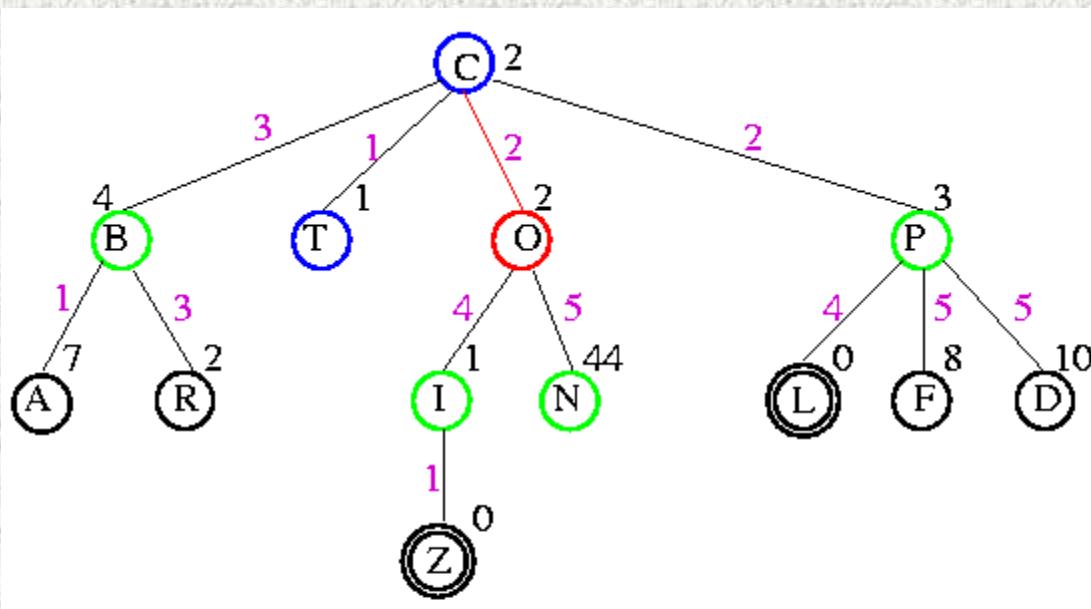
Open List = O ($2+2=4$), P ($2+3=5$), B($3+4=7$)

Example



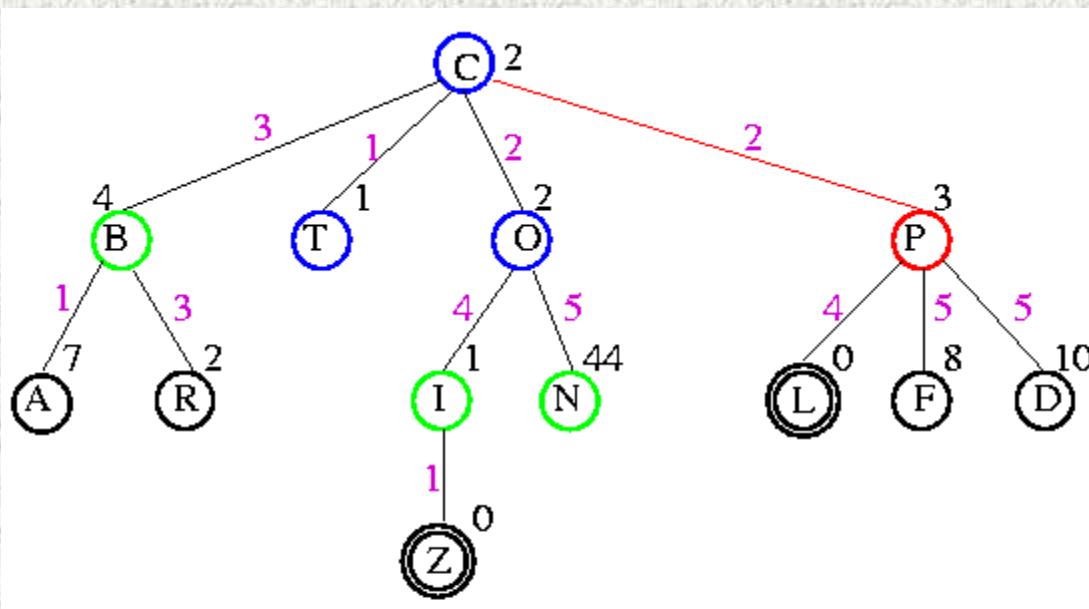
Open List = O (2+2=4), P (2+3=5), B(3+4=7)

Example



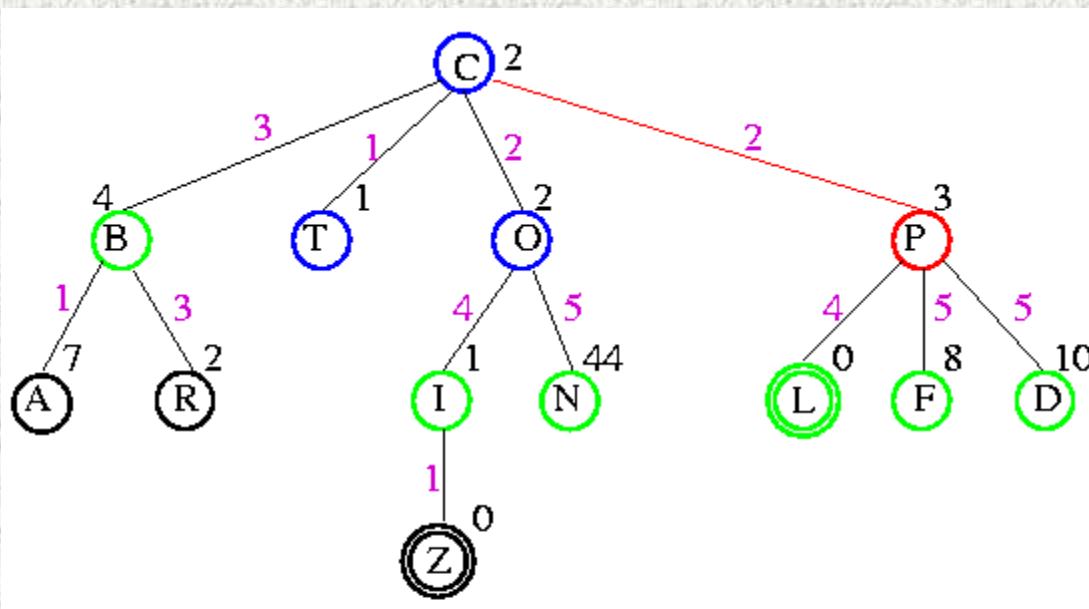
Open List = O (2+2=4), P (2+3=5), B(3+4=7)
I (6+1=7), N (7+44=51)

Example



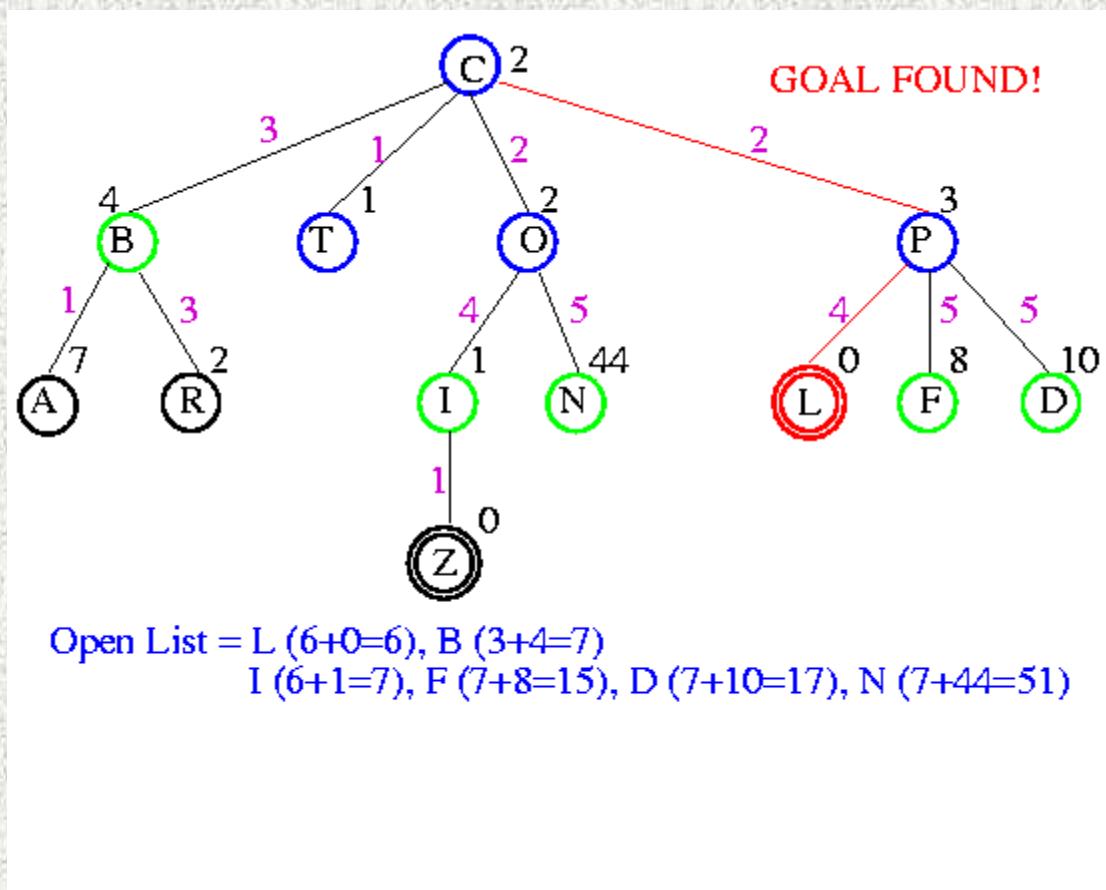
Open List = P ($2+3=5$), B($3+4=7$)
I ($6+1=7$), N ($7+44=51$)

Example



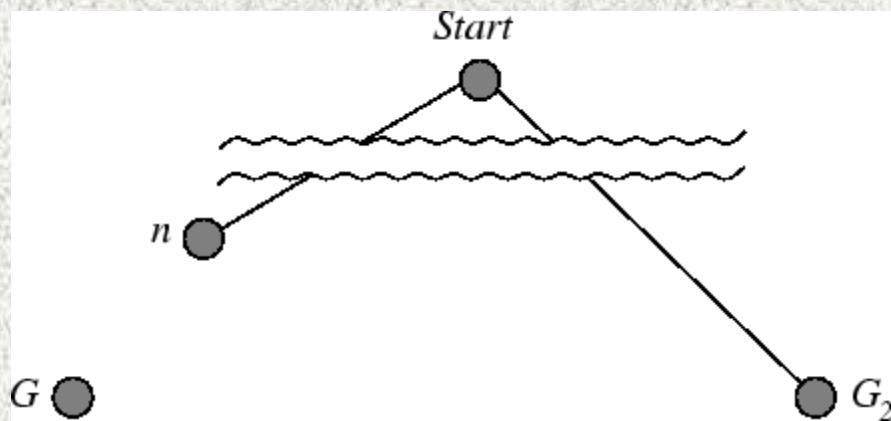
Open List = P ($2+3=5$), L ($6+0=6$), B ($3+4=7$)
I ($6+1=7$), F ($7+8=15$), D ($7+10=17$), N ($7+44=51$)

Example



Optimality of A*

- Suppose a suboptimal goal G_2 is on the open list
- Let n be unexpanded node on smallest-cost path to optimal goal G_1



$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\ &\geq g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Comparison of Search Techniques

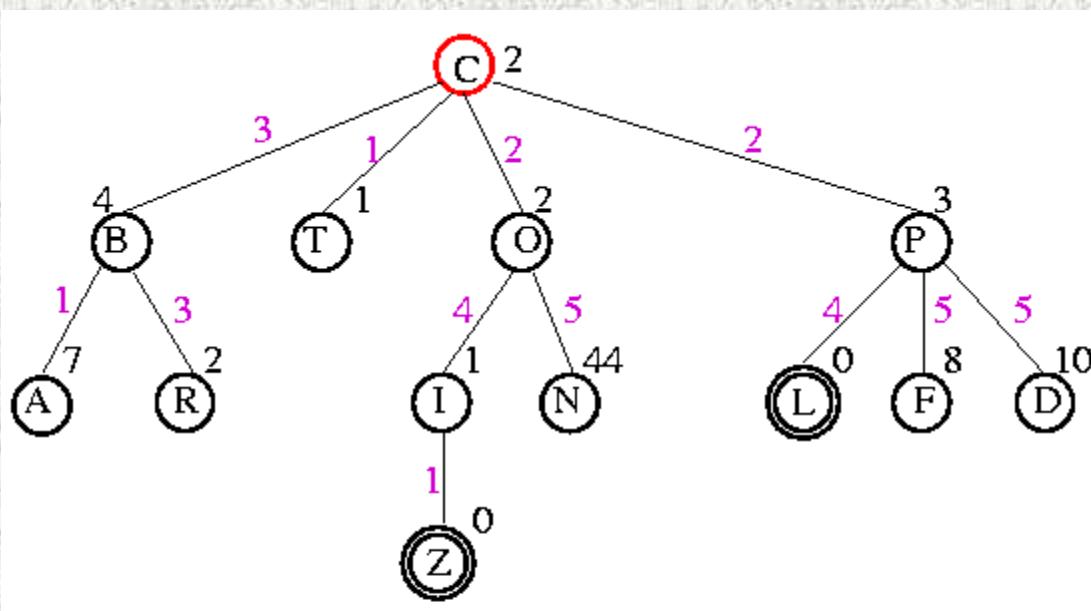
	DFS	BFS	UCS	IDS	Best	HC	Beam	A*
Complete	N	Y	Y	Y	N	N	N	Y
Optimal	N	N	Y	N	N	N	N	Y
Heuristic	N	N	N	N	Y	Y	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm	nm	b^m
Space	bm	b^{d+1}	b^m	bd	b^m	b	bn	b^m



IDA*

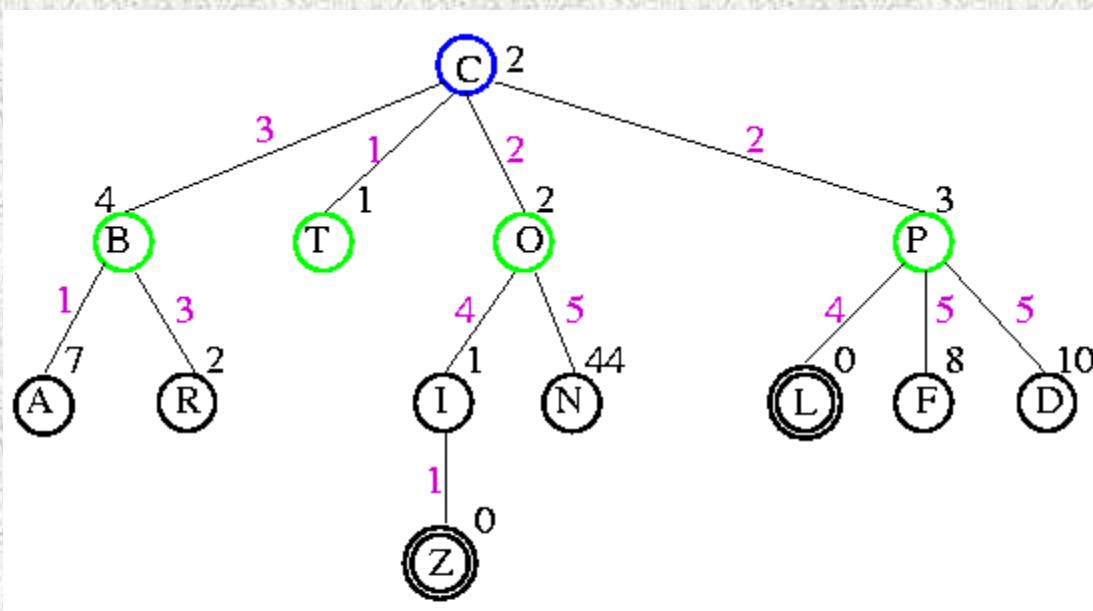
- Series of Depth-First Searches
- Like Iterative Deepening Search, except
 - Use A* cost threshold instead of depth threshold
 - Ensures optimal solution
- QueuingFn enqueues at front if $f(\text{child}) \leq \text{threshold}$
- Threshold
 - $h(\text{root})$ first iteration
 - Subsequent iterations
 - $f(\text{min_child})$
 - min_child is the cut off child with the minimum f value
 - Increase always includes at least one new node
 - Makes sure search never looks beyond optimal cost solution

Example



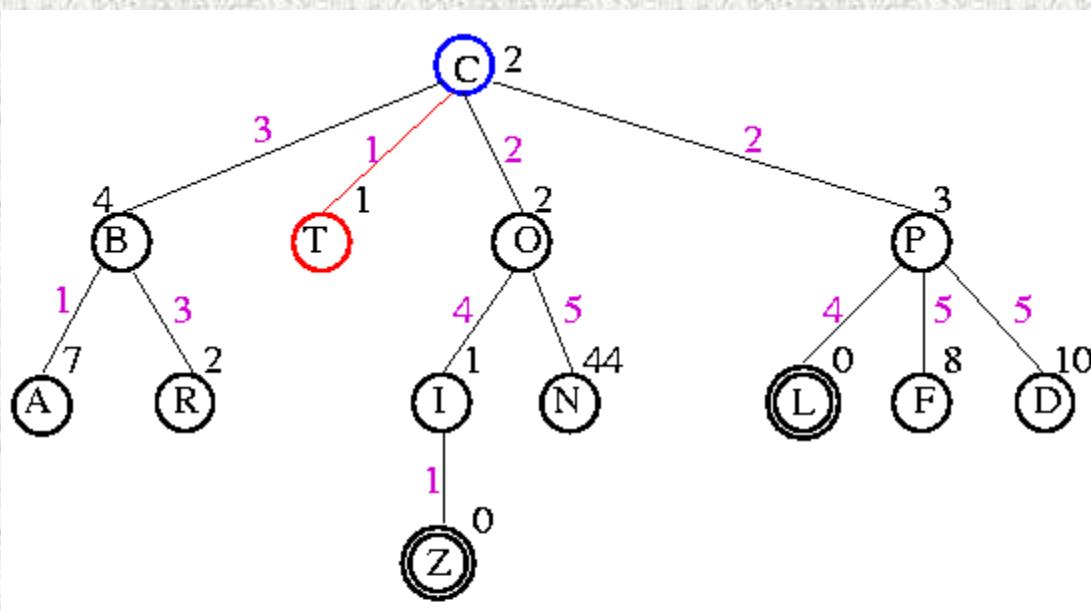
limit = $f(C) = 2$

Example



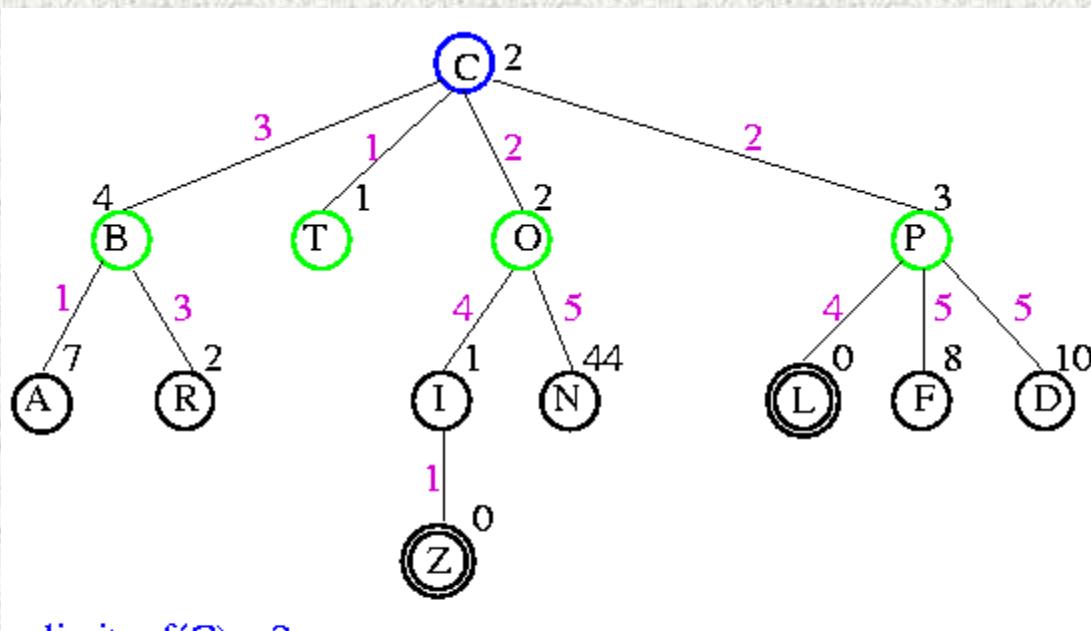
limit = $f(C) = 2$

Example

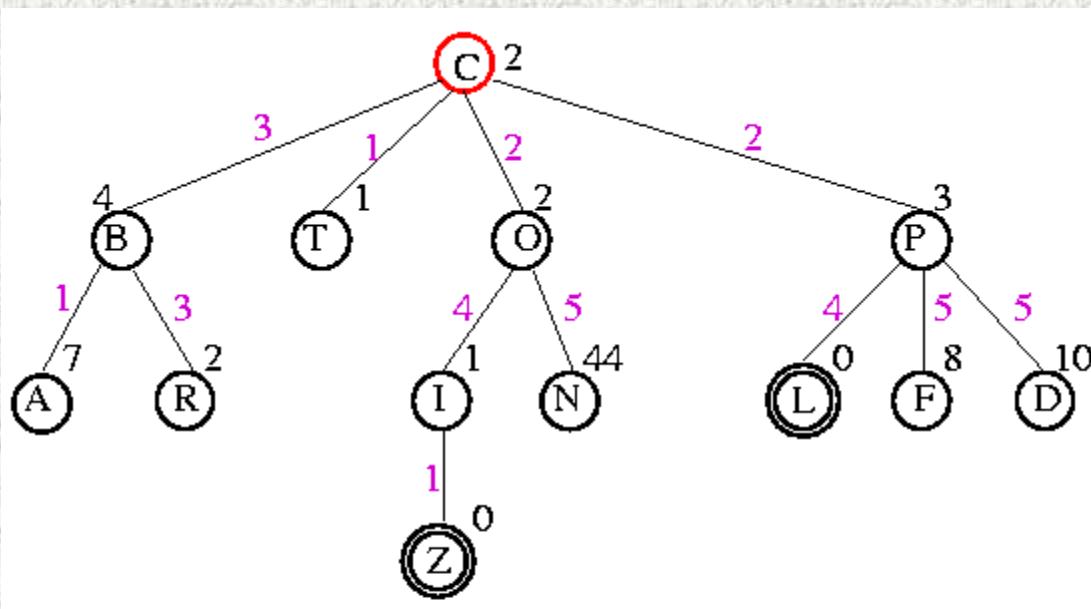


limit = $f(C) = 2$

Example

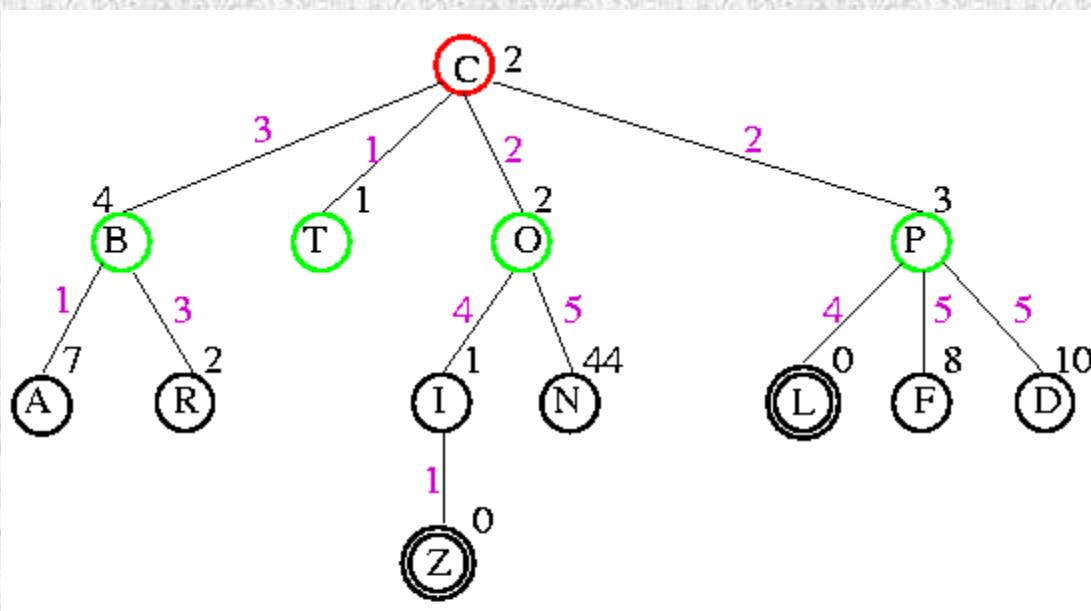


Example

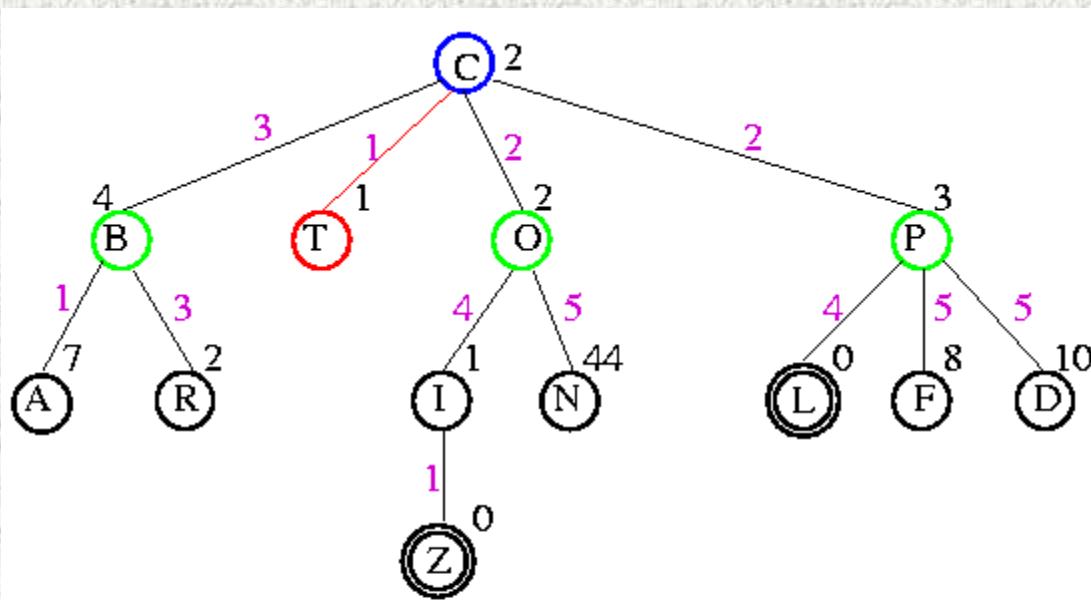


limit = $f(O) = 4$

Example

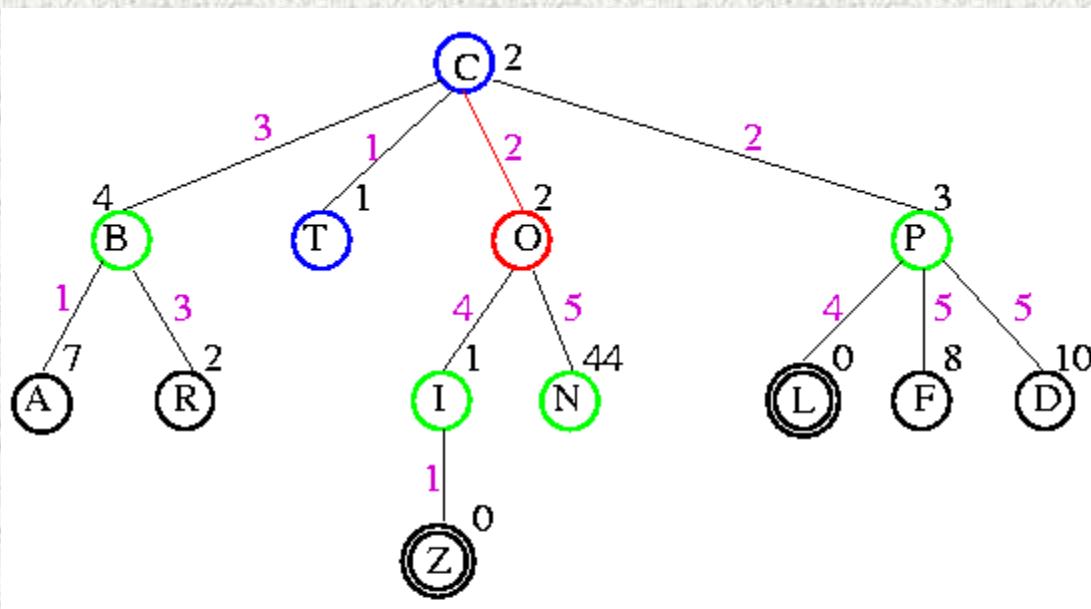


Example



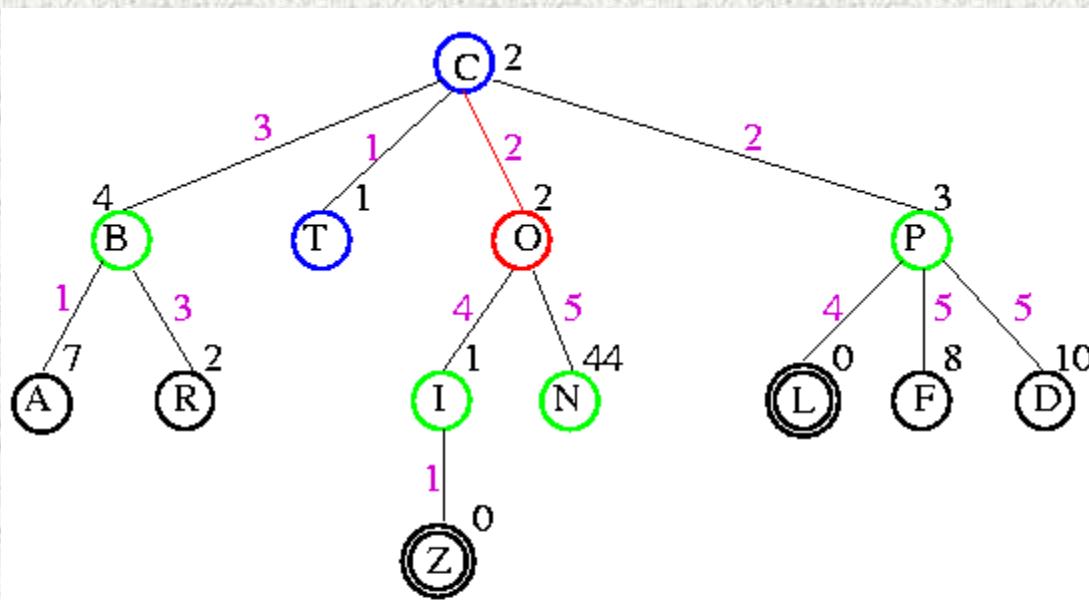
limit = $f(O) = 4$

Example



limit = $f(O) = 4$

Example

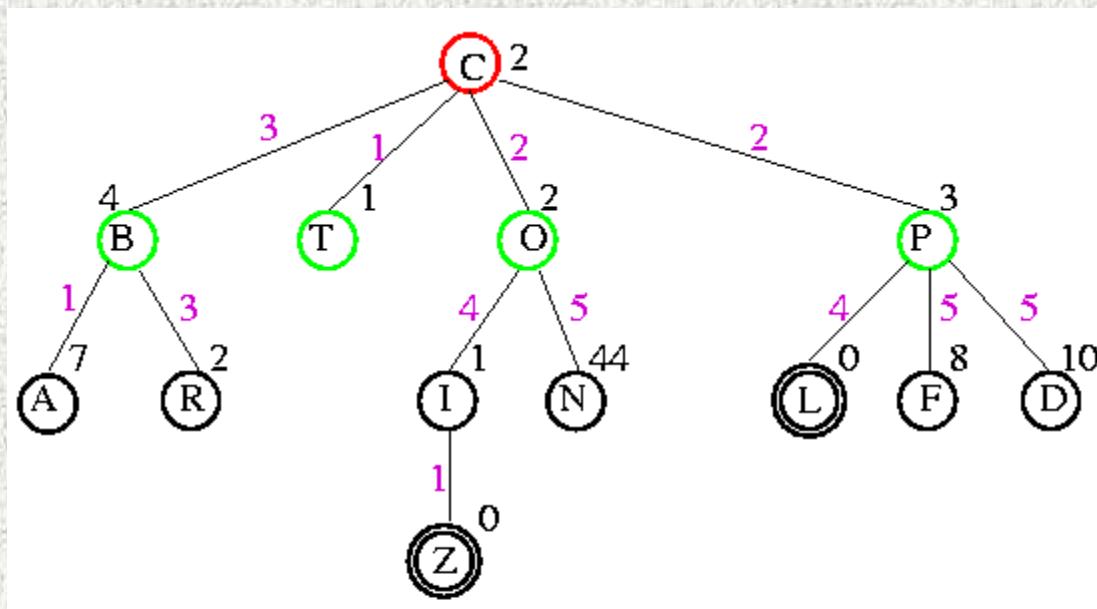


$$\text{limit} = f(O) = 4$$

Nodes on frontier: B ($3+4=7$), P ($2+3=5$)
I ($6+1=7$), N ($7+44=51$)

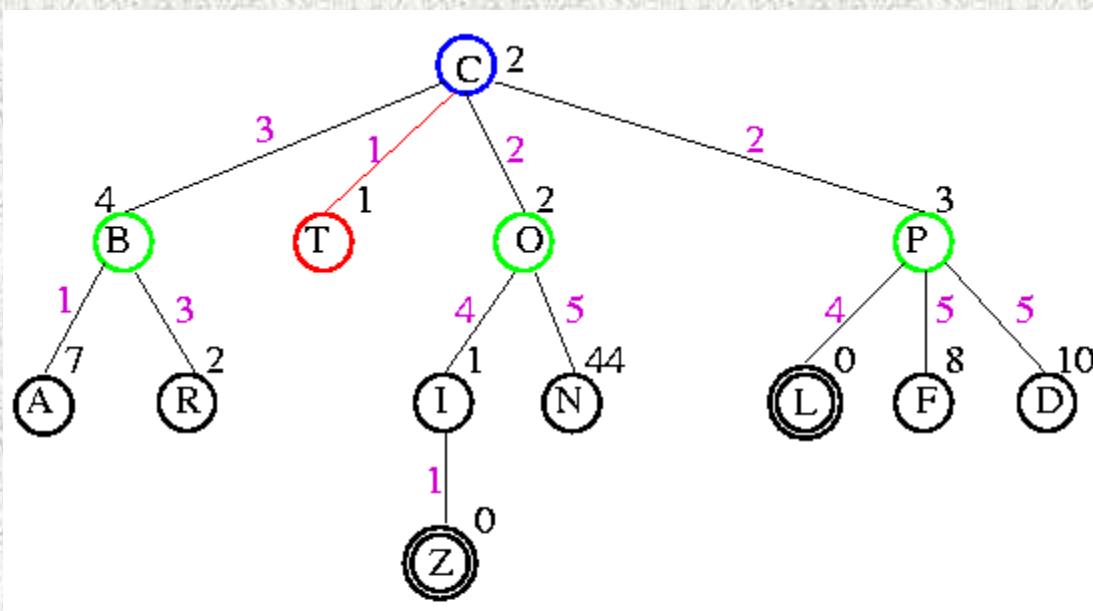
$$\text{New limit} = f(P) = 5$$

Example



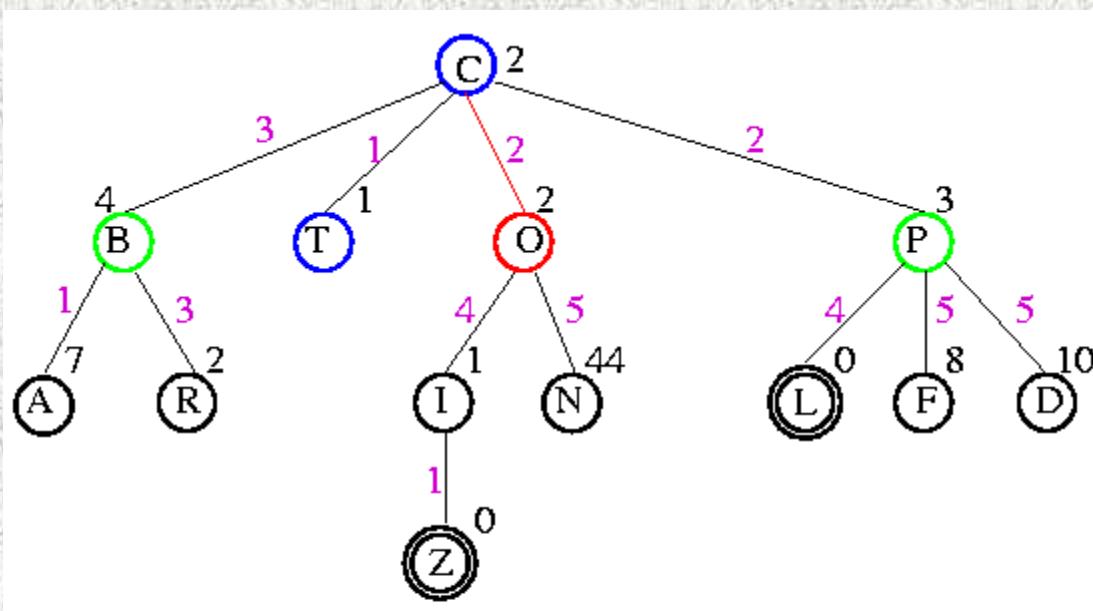
limit = $f(P) = 5$

Example



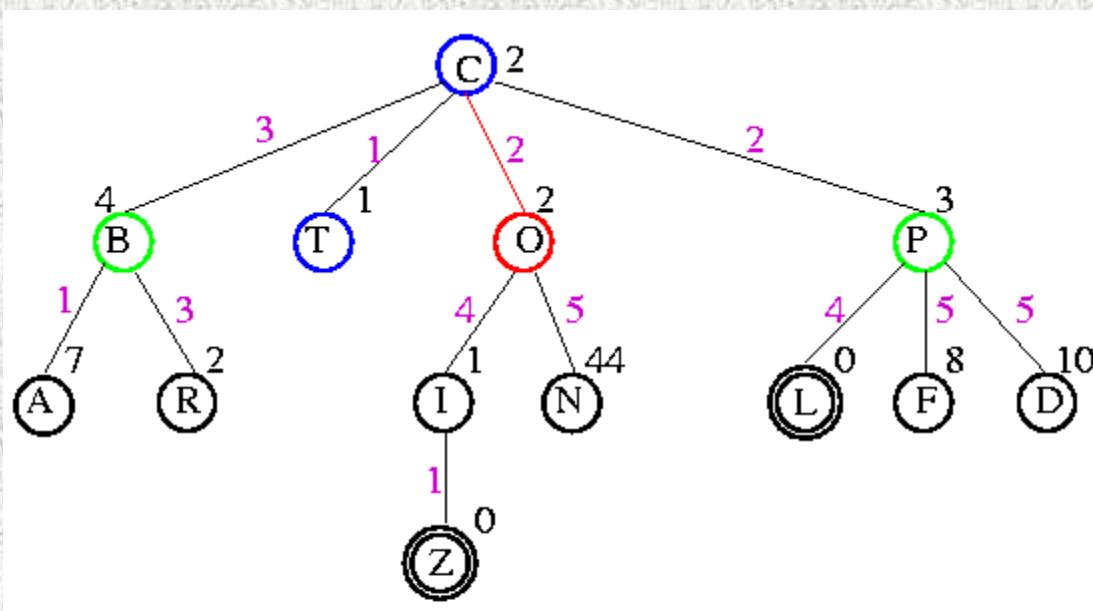
limit = $f(P) = 5$

Example



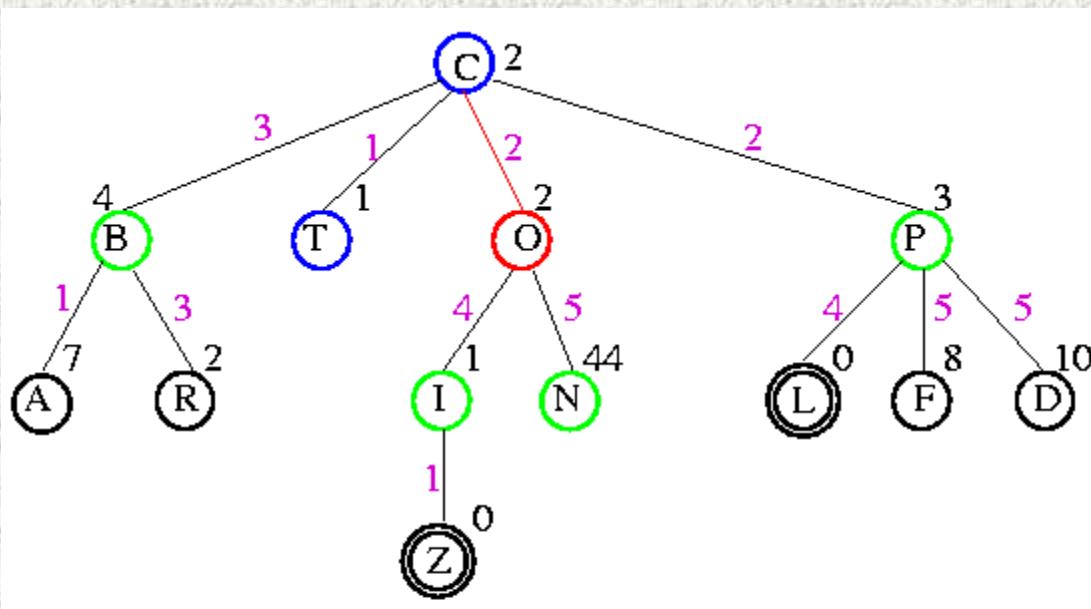
limit = $f(P) = 5$

Example

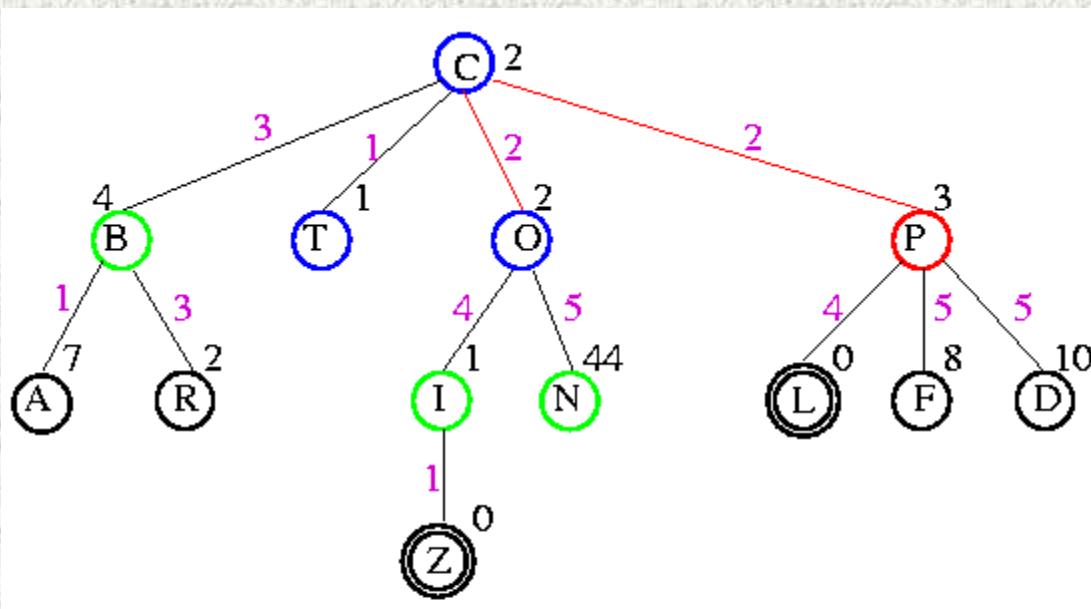


limit = $f(P) = 5$

Example

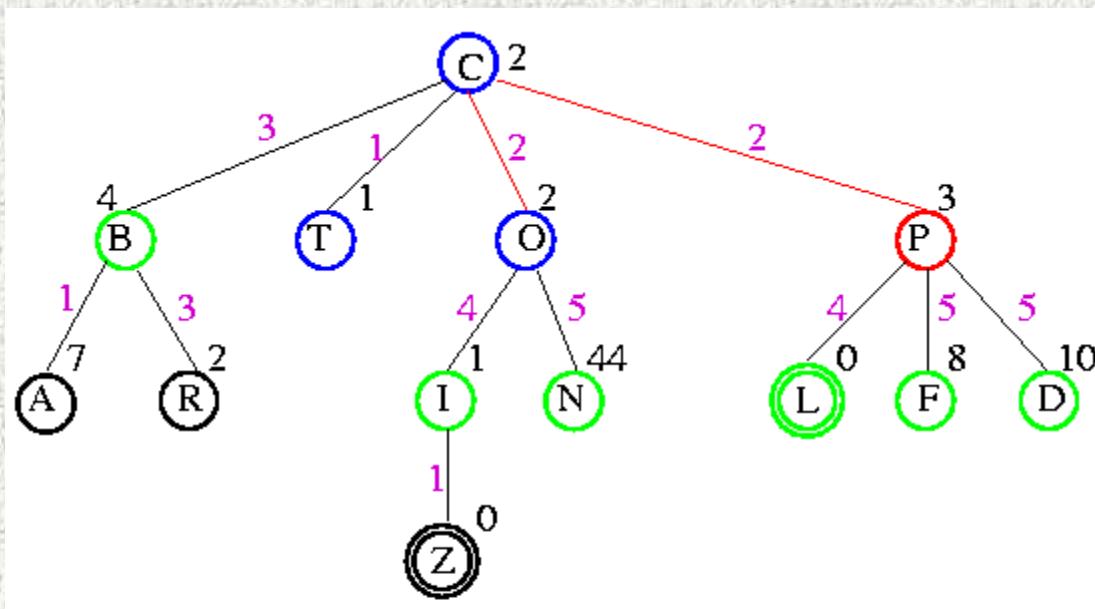


Example



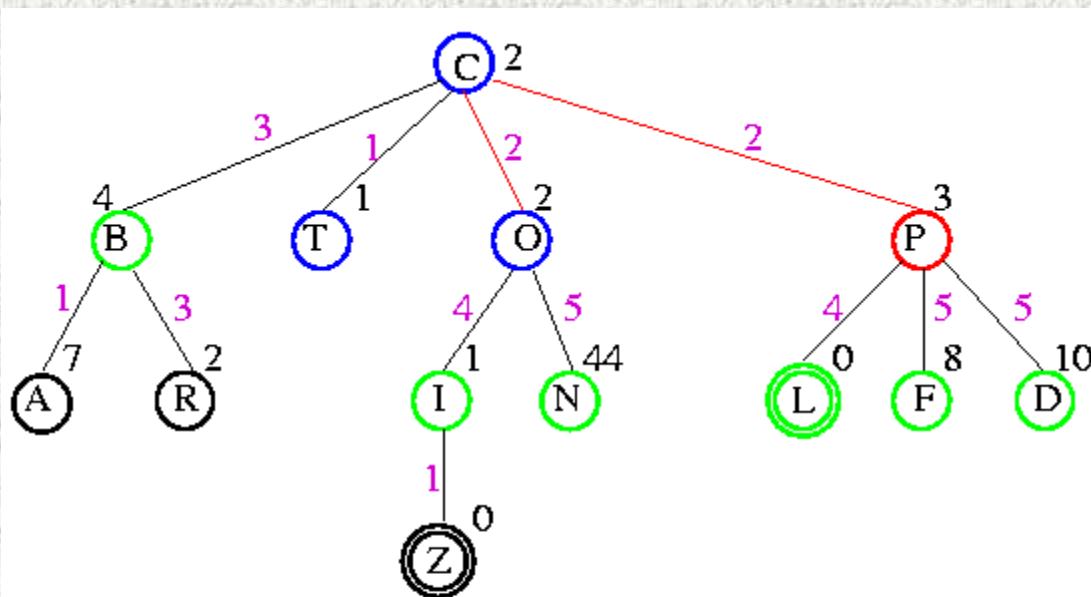
limit = $f(P) = 5$

Example



limit = $f(P) = 5$

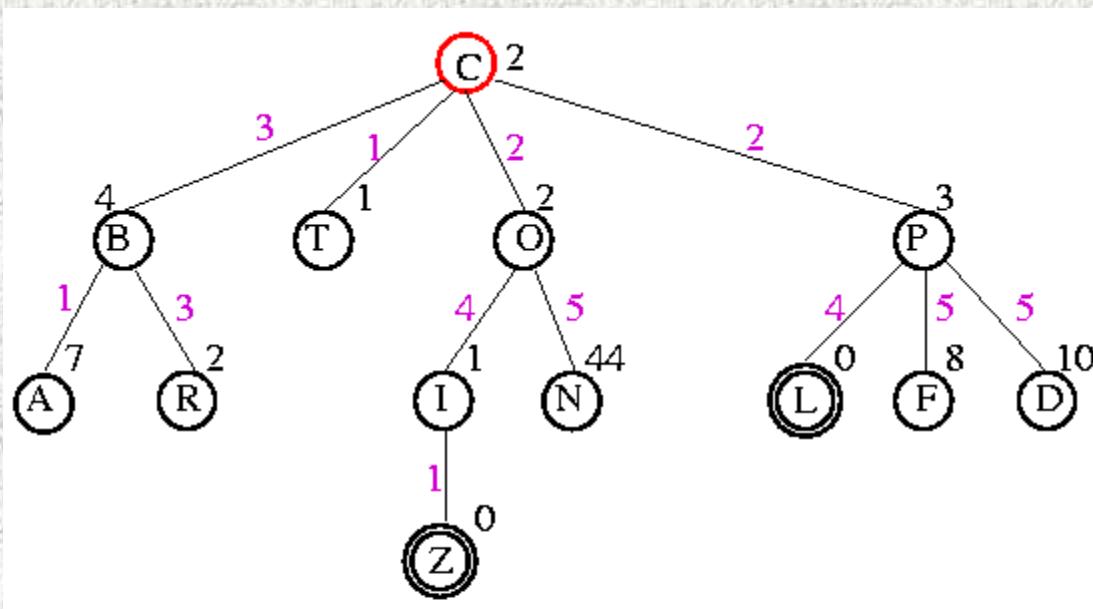
Example



$$\text{limit} = f(L) = 6$$

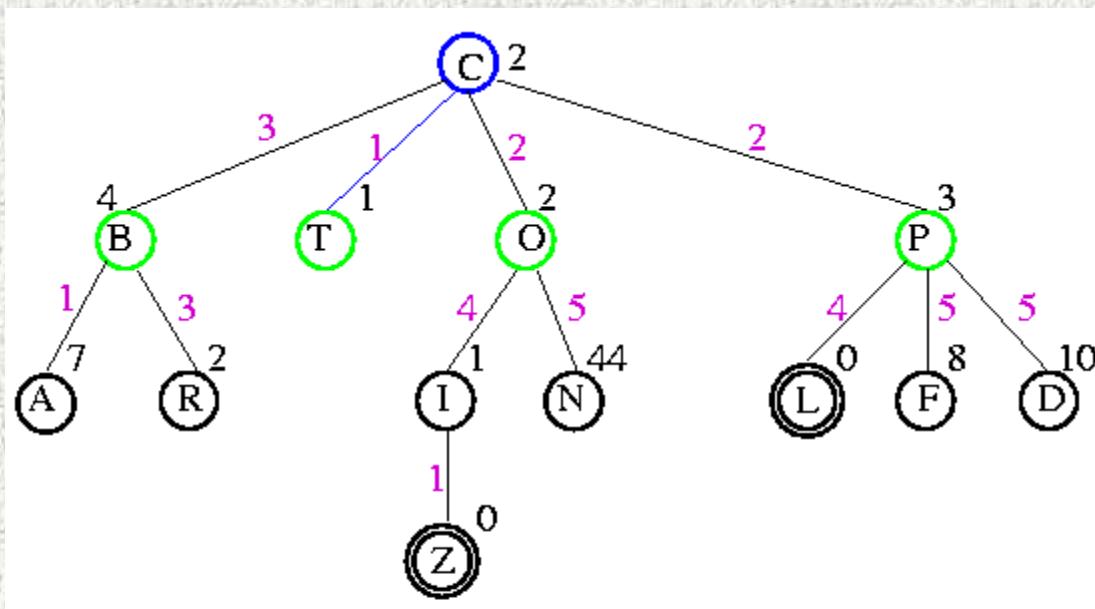
Nodes on frontier: B ($3+4=7$), I ($6+1=7$), N ($7+44=51$)
L ($6+0=6$), F ($7+8=15$), D ($7+10=17$)

Example



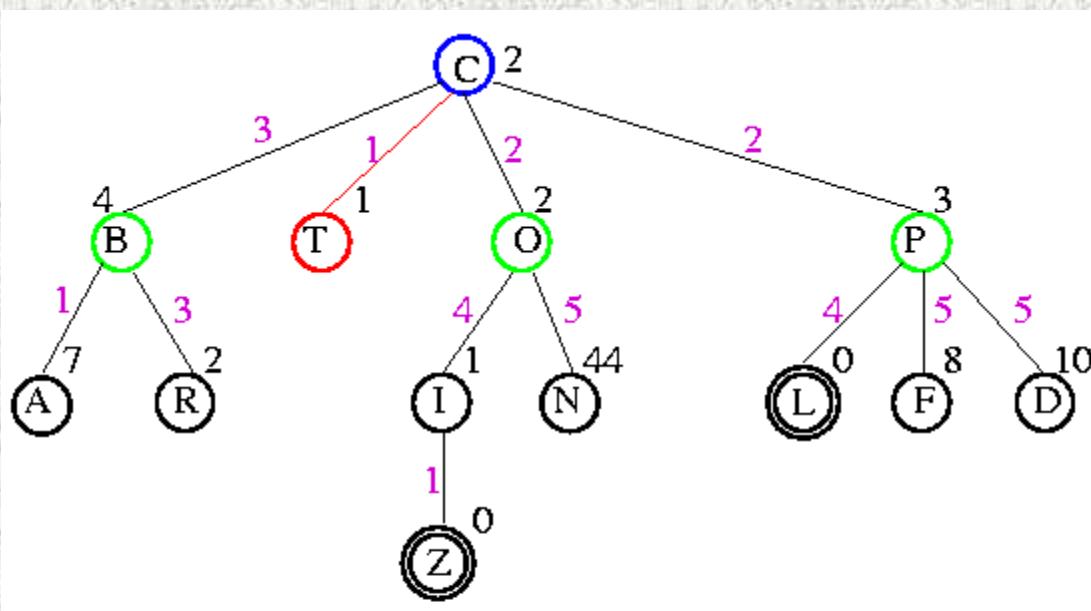
limit = $f(L) = 6$

Example



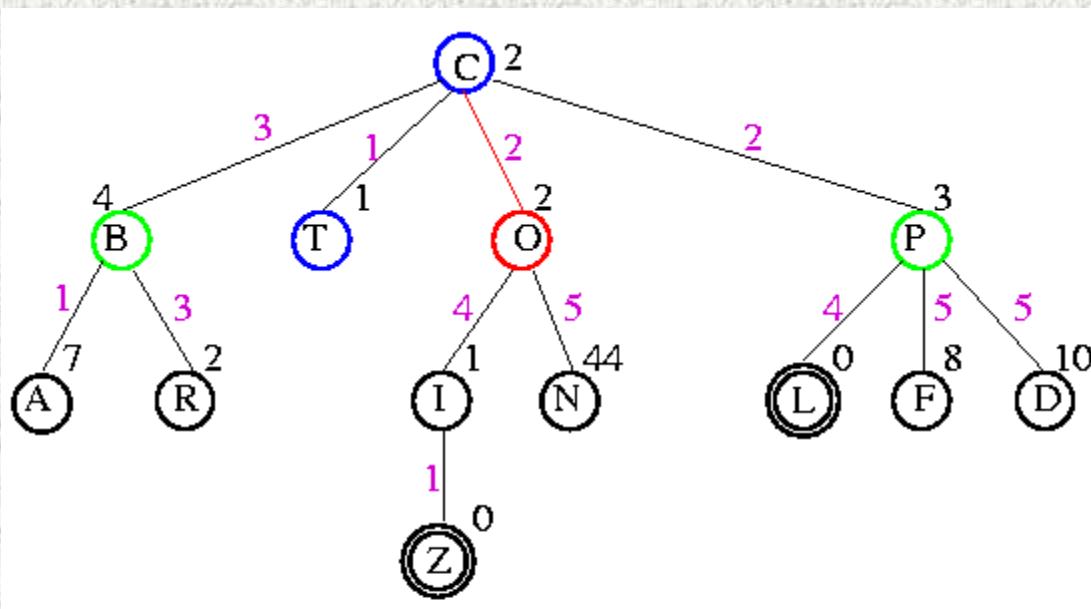
limit = $f(L) = 6$

Example



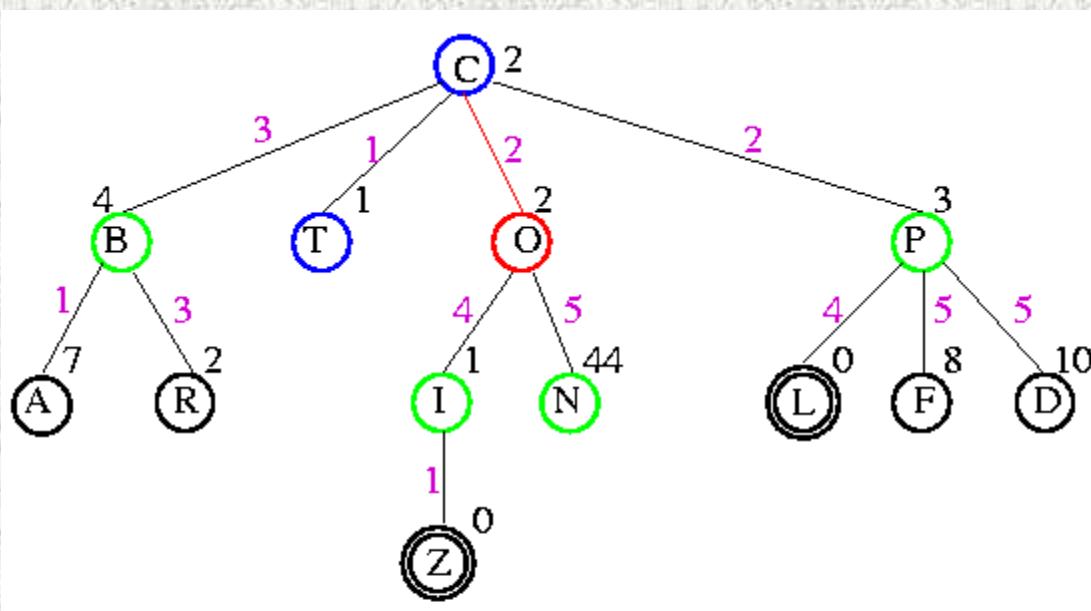
limit = $f(L) = 6$

Example



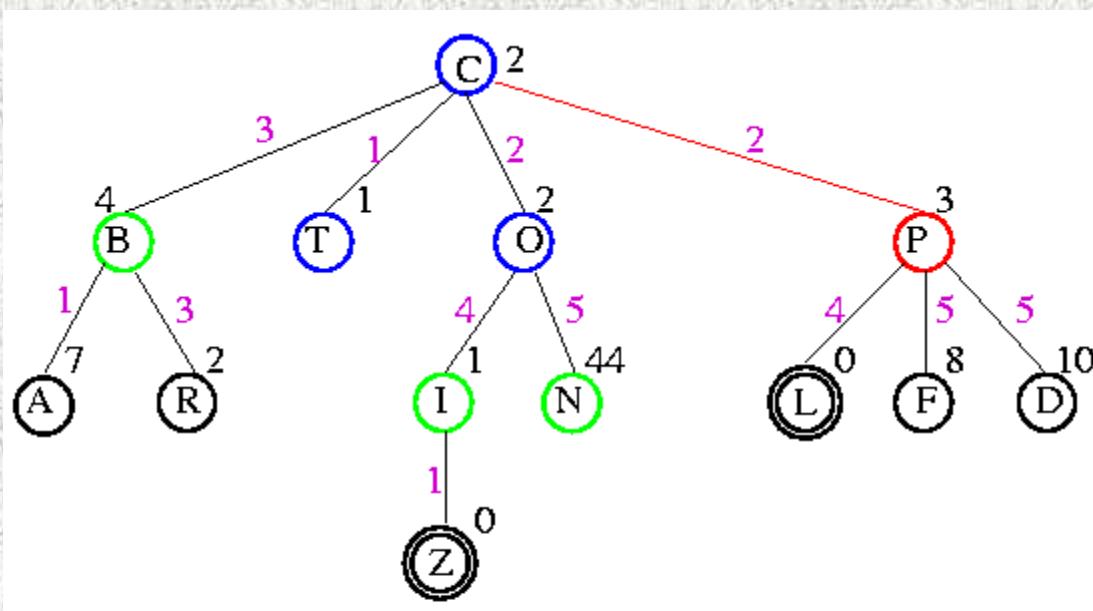
limit = $f(L) = 6$

Example

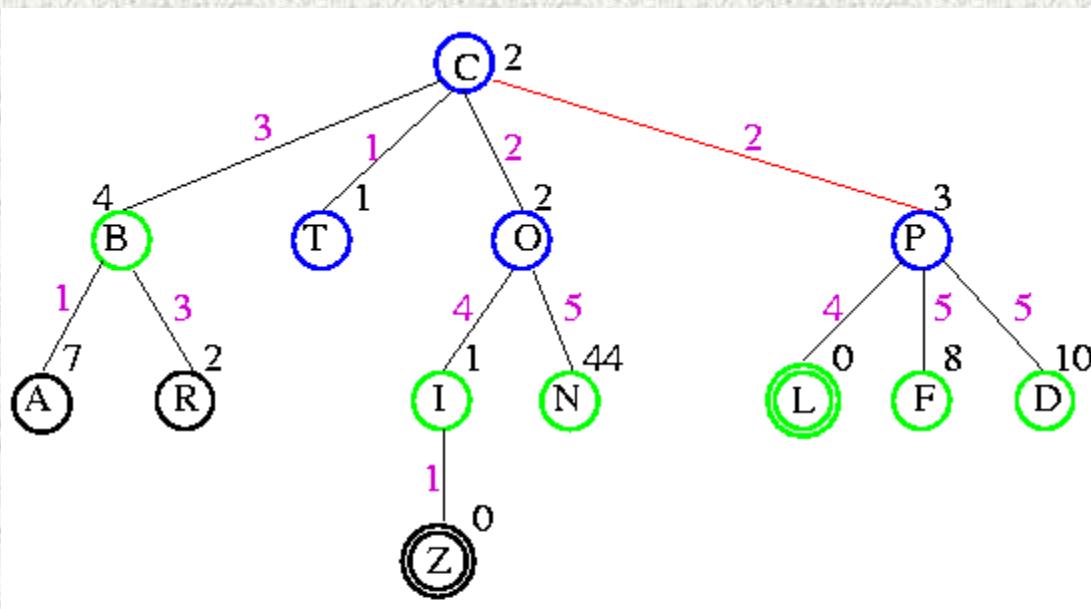


limit = $f(L) = 6$

Example

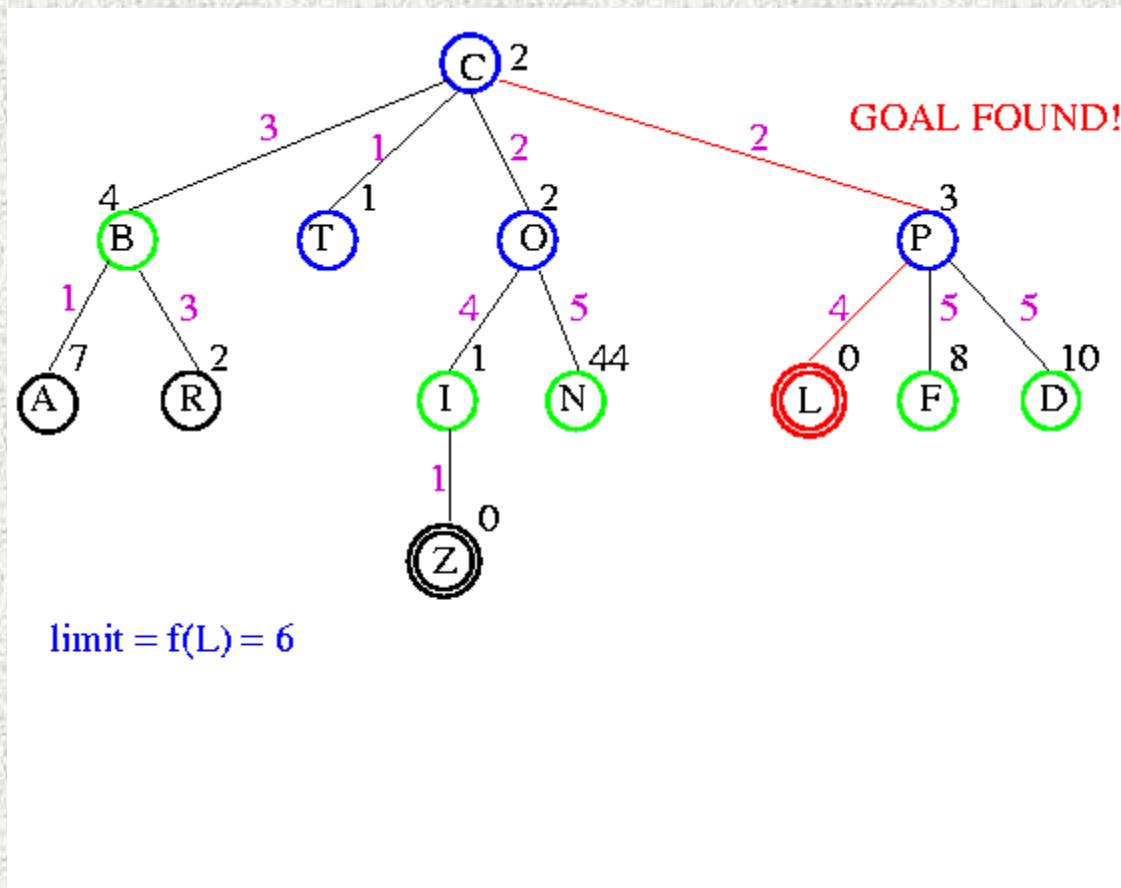


Example

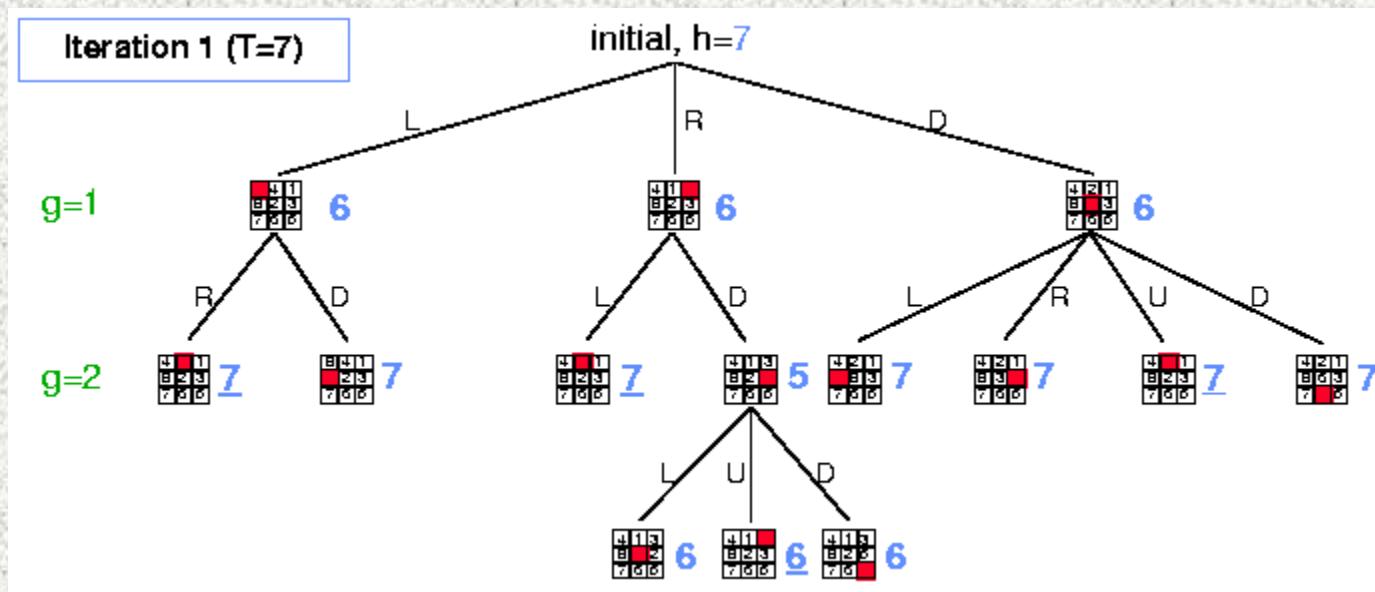


limit = $f(L) = 6$

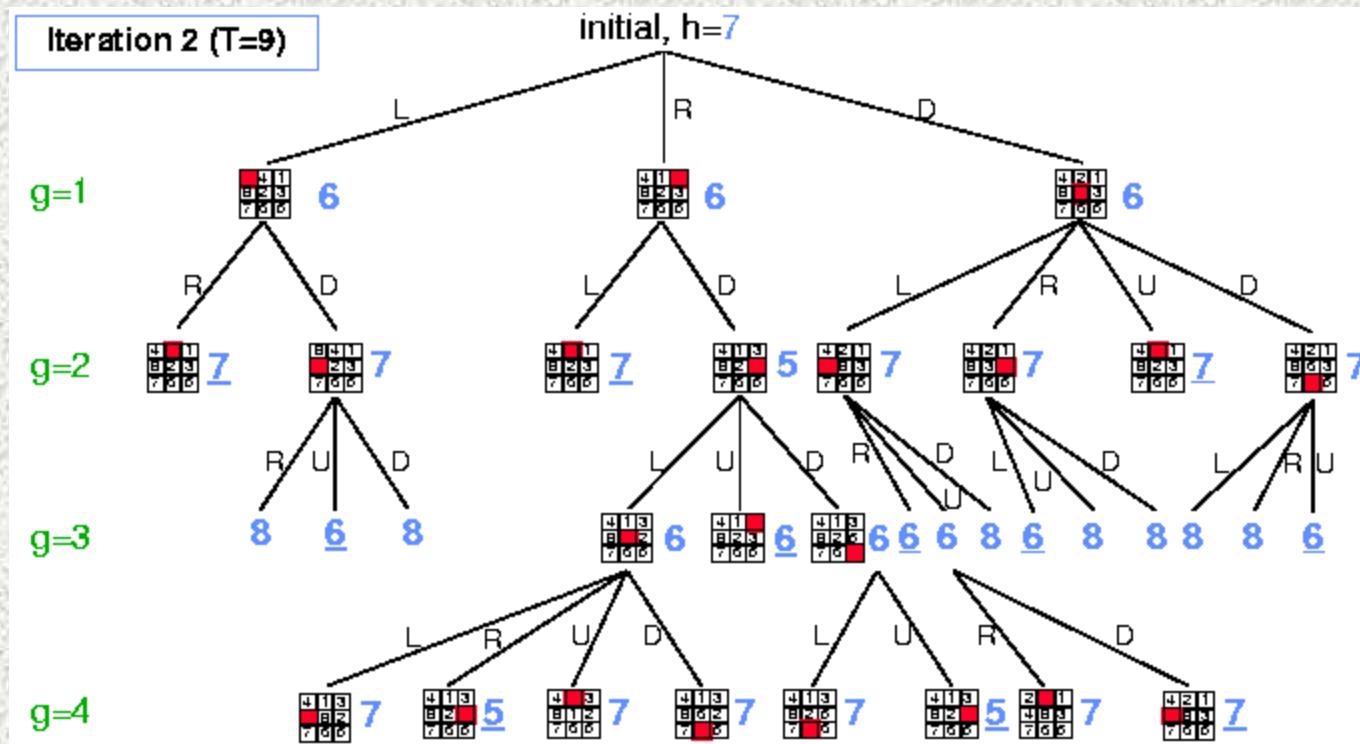
Example



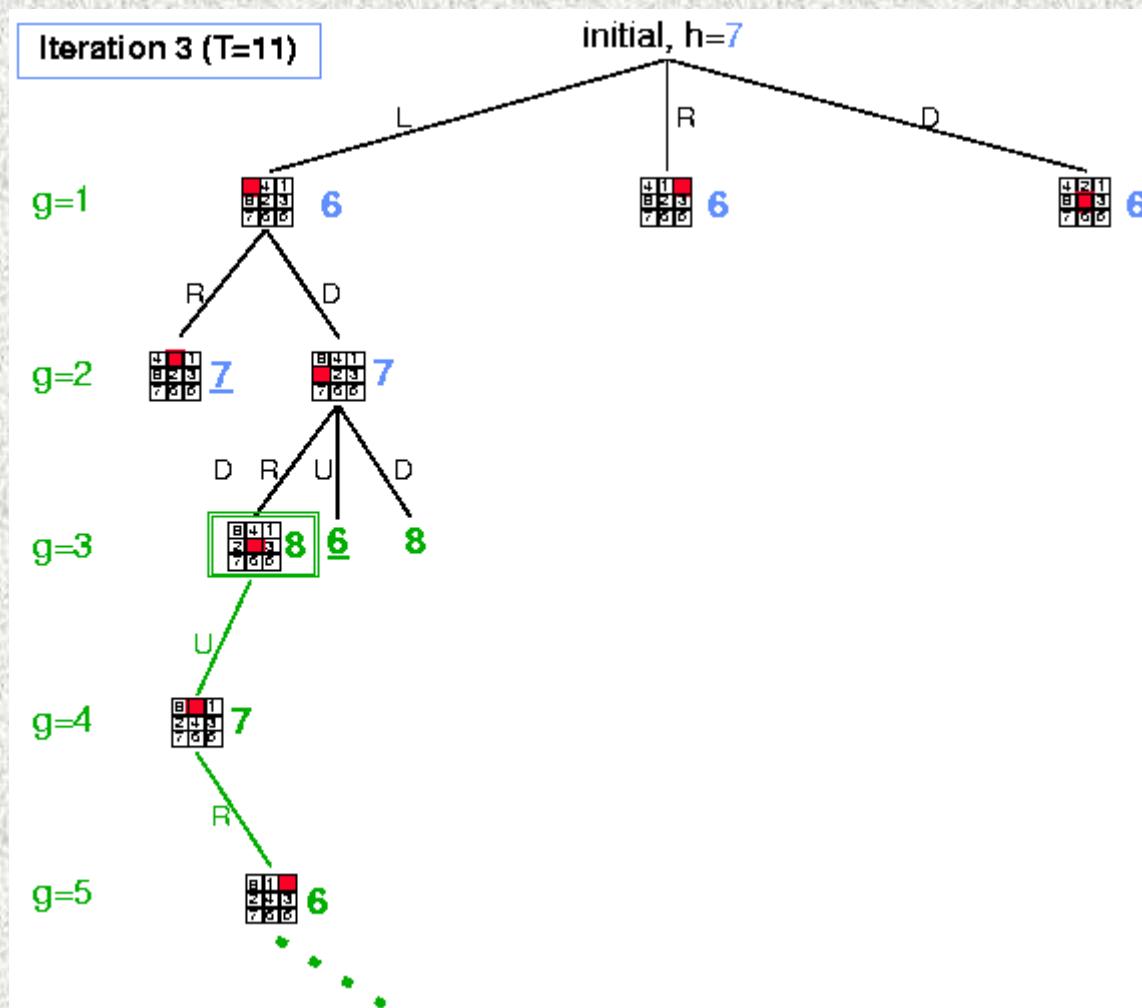
Example



Example



Example



Analysis

- Some redundant search
 - Small amount compared to work done on last iteration
- Dangerous if continuous-valued $h(n)$ values or if values very close
 - If threshold = 21.1 and value is 21.2, probably only include 1 new node each iteration
- Time complexity is $O(b^m)$
- Space complexity is $O(m)$

Comparison of Search Techniques

	DFS	BFS	UCS	IDS	Best	HC	Beam	A*	IDA*
Complete	N	Y	Y	Y	N	N	N	Y	Y
Optimal	N	N	Y	N	N	N	N	Y	Y
Heuristic	N	N	N	N	Y	Y	Y	Y	Y
Time	b^m	b^{d+1}	b^m	b^d	b^m	bm	nm	b^m	b^m
Space	bm	b^{d+1}	b^m	bd	b^m	b	bn	b^m	bm



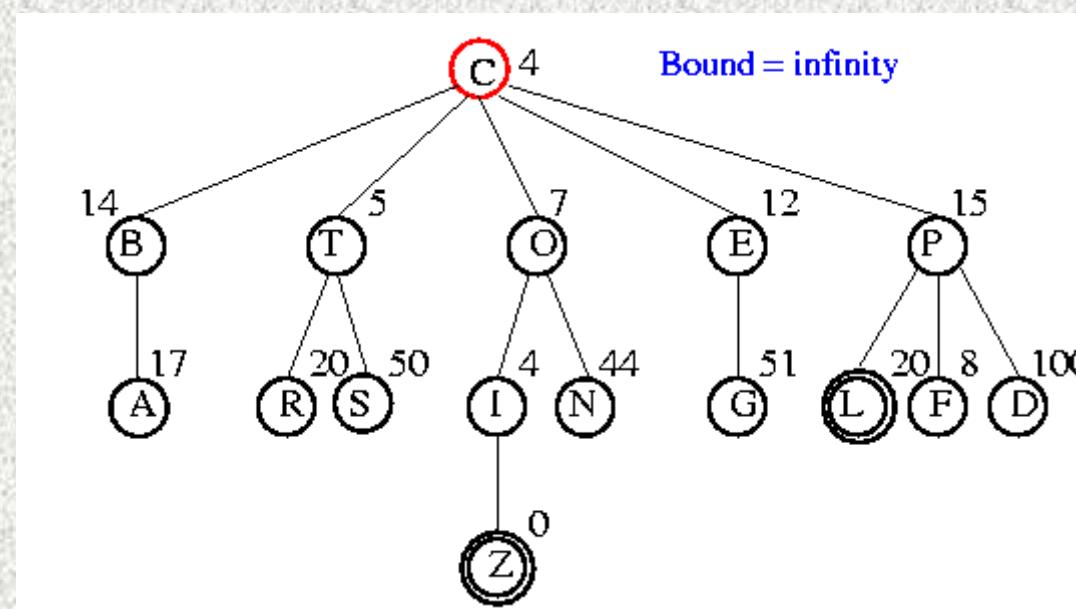
RBFS

- Recursive Best First Search
 - Linear space variant of A*
- Perform A* search but discard subtrees when perform recursion
- Keep track of alternative (next best) subtree
- Expand subtree until f value greater than bound
- Update f values before (from parent) and after (from descendant) recursive call

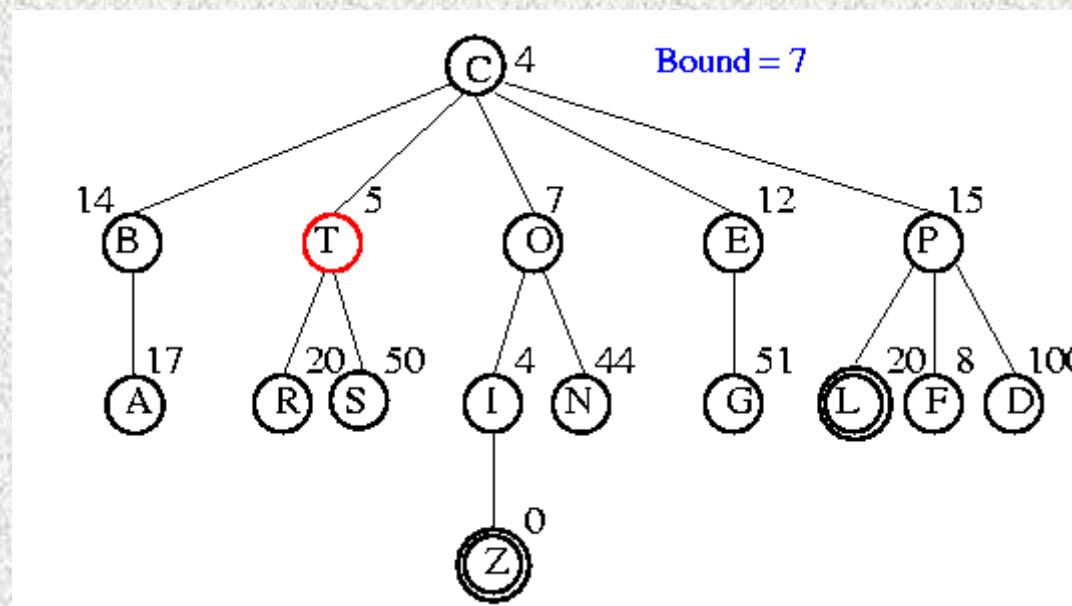
Algorithm

```
// Input is current node and f limit
// Returns goal node or failure, updated limit
RBFS(n, limit)
    if Goal(n)
        return n
    children = Expand(n)
    if children empty
        return failure, infinity
    for each c in children
        f[c] = max(g(c)+h(c), f[n])           // Update f[c] based on parent
    repeat
        best = child with smallest f value
        if f[best] > limit
            return failure, f[best]
        alternative = second-lowest f-value among children
        newlimit = min(limit, alternative)
        result, f[best] = RBFS(best, newlimit) // Update f[best] based on descendant
    if result not equal to failure
        return result
```

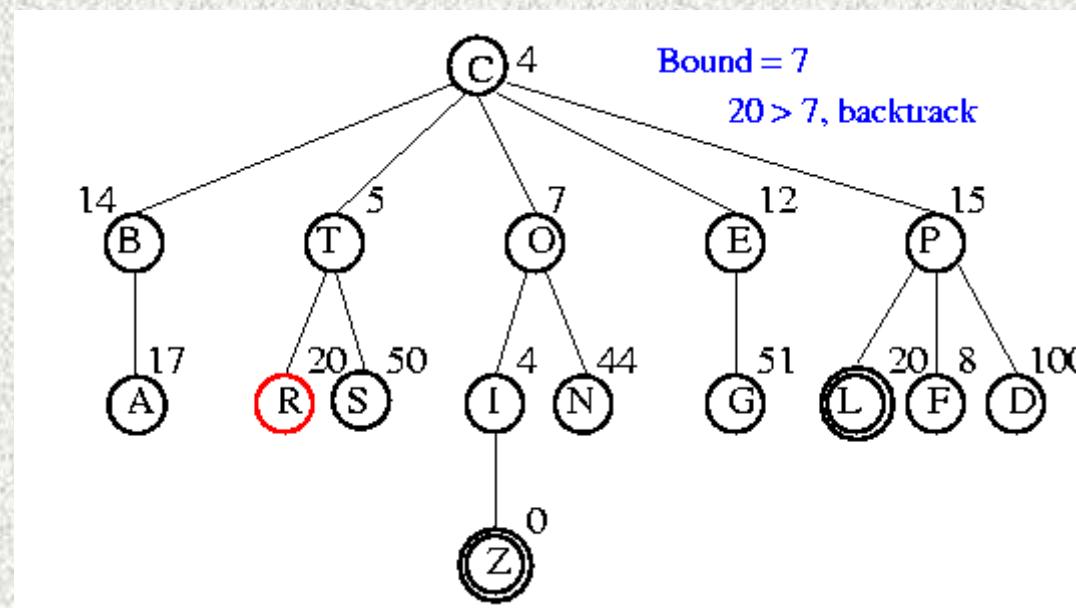
Example



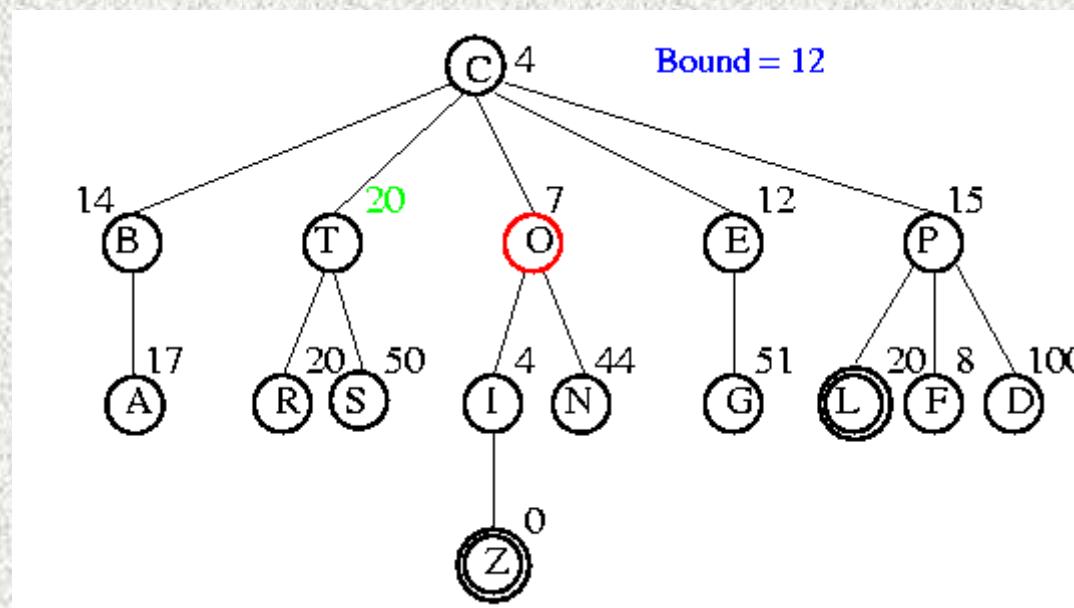
Example



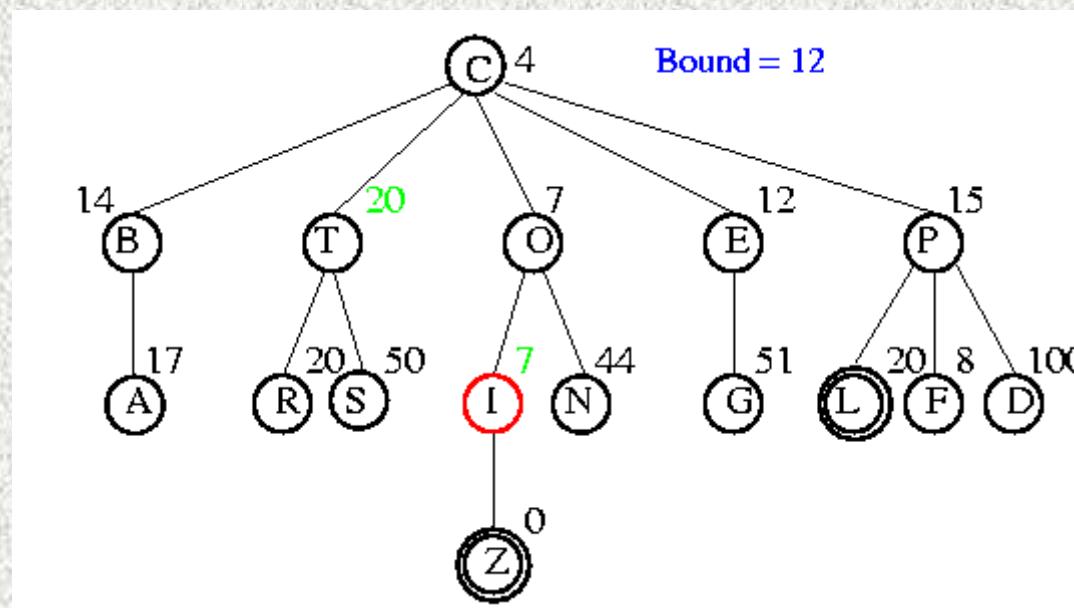
Example



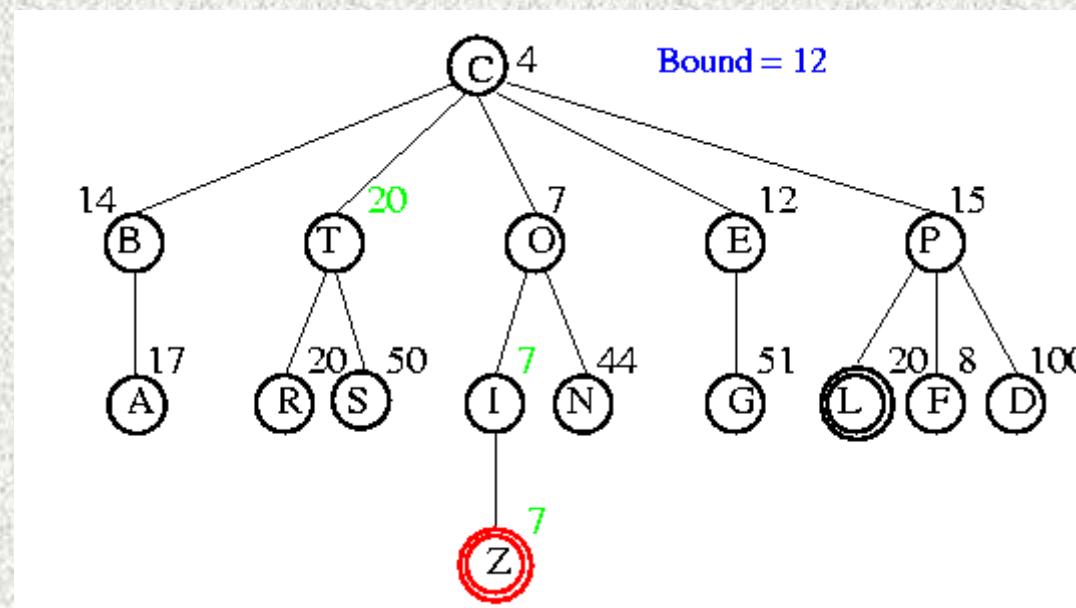
Example



Example



Example

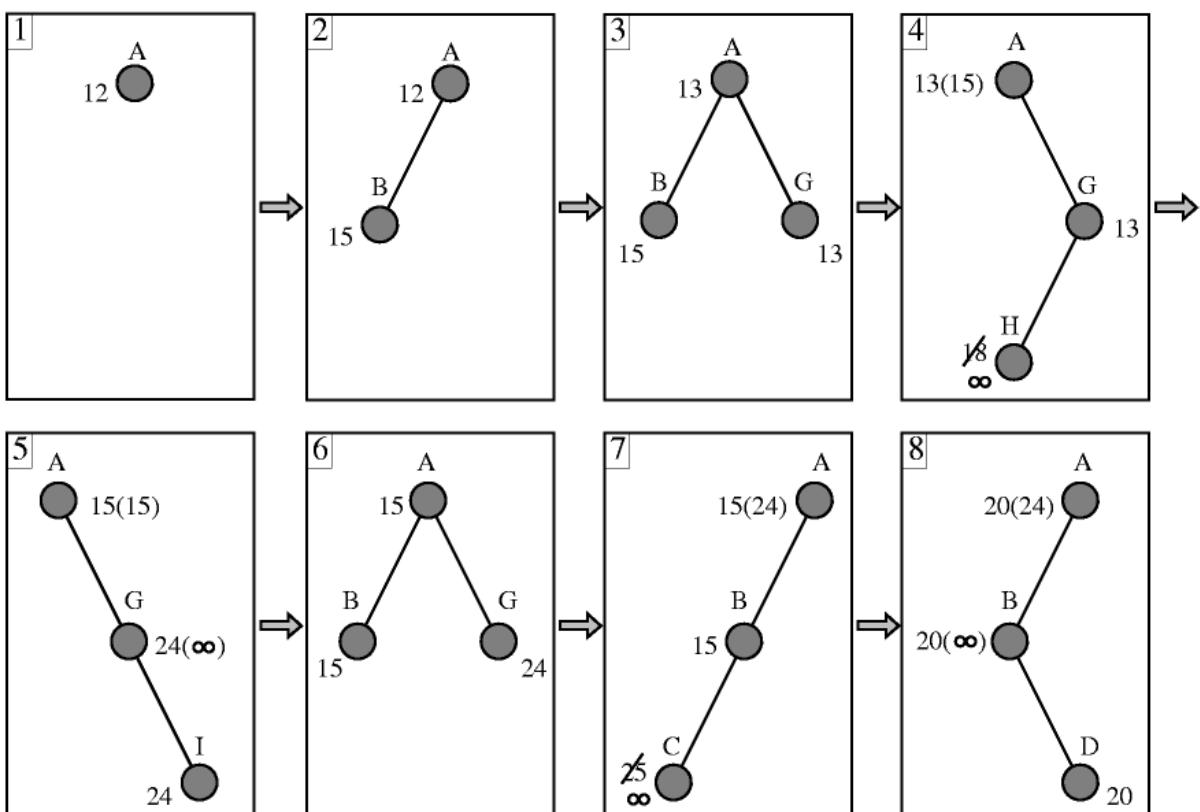
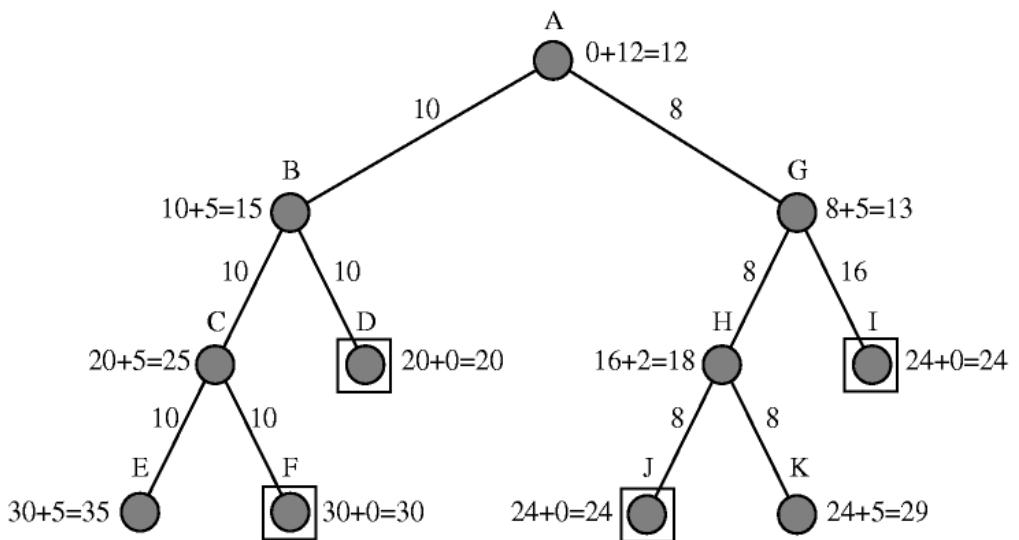


Analysis

- Optimal if $h(n)$ is admissible
- Space is $O(bm)$
- Features
 - Potentially exponential time in cost of solution
 - More efficient than IDA*
 - Keeps more information than IDA* but benefits from storing this information

SMA*

- Simplified Memory-Bounded A* Search
- Perform A* search
- When memory is full
 - Discard worst leaf (largest $f(n)$ value)
 - Back value of discarded node to parent
- Optimal if solution fits in memory



Example

- Let $\text{MaxNodes} = 3$
- Initially B&G added to open list, then hit max
- B is larger f value so discard but save $f(B)=15$ at parent A
 - Add H but $f(H)=18$. Not a goal and cannot go deeper, so set $f(h)=\infty$ and save at G.
- Generate next child I with $f(I)=24$, bigger child of A. We have seen all children of G, so reset $f(G)=24$.
- Regenerate B and child C. This is not goal so $f(c)$ reset to infinity
- Generate second child D with $f(D)=24$, backing up value to ancestors
- D is a goal node, so search terminates.

Heuristic Functions

- Q: Given that we will only use heuristic functions that do not overestimate, what type of heuristic functions (among these) perform best?
- A: Those that produce higher $h(n)$ values.

Reasons

- Higher h value means closer to actual distance
- Any node n on open list with
 - $f(n) < f^*(\text{goal})$
 - will be selected for expansion by A*
- This means if a lot of nodes have a low underestimate (lower than actual optimum cost)
 - All of them will be expanded
 - Results in increased search time and space

Informedness

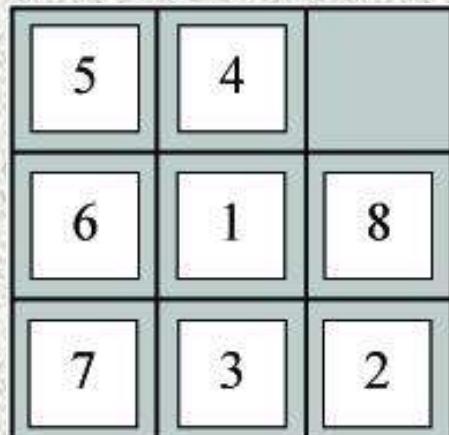
- If h_1 and h_2 are both admissible and
- For all x , $h_1(x) > h_2(x)$, then h_1 “dominates” h_2
 - Can also say h_1 is “more informed” than h_2
- Example
 - $h_1(x)$: $|x_{goal} - x|$
 - $h_2(x)$: Euclidean distance $\sqrt{(x_{goal} - x)^2 + (y_{goal} - y)^2}$
 - h_2 dominates h_1

Effect on Search Cost

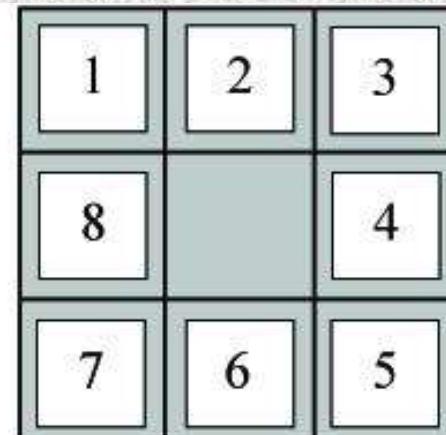
- If $h_2(n) \geq h_1(n)$ for all n (both are admissible)
 - then h_2 dominates h_1 and is better for search
- Typical search costs
 - $d=14$, IDS expands 3,473,941 nodes
 - A* with h_1 expands 539 nodes
 - A* with h_2 expands 113 nodes
 - $d=24$, IDS expands $\sim 54,000,000,000$ nodes
 - A* with h_1 expands 39,135 nodes
 - A* with h_2 expands 1,641 nodes

Which of these heuristics are admissible?

Which are more informed?



Start State



Goal State

- $h_1(n) = \# \text{tiles in wrong position}$
- $h_2(n) = \text{Sum of Manhattan distance between each tile and goal location for the tile}$
- $h_3(n) = 0$
- $h_4(n) = 1$
- $h_5(n) = \min(2, h^*[n])$
- $h_6(n) = \text{Manhattan distance for blank tile}$
- $h_7(n) = \max(2, h^*[n])$

Generating Heuristic Functions

- Generate heuristic for simpler (relaxed) problem
 - Relaxed problem has fewer restrictions
 - Eight puzzle where multiple tiles can be in the same spot
 - Cost of optimal solution to relaxed problem is an admissible heuristic for the original problem
- Learn heuristic from experience

Iterative Improvement Algorithms

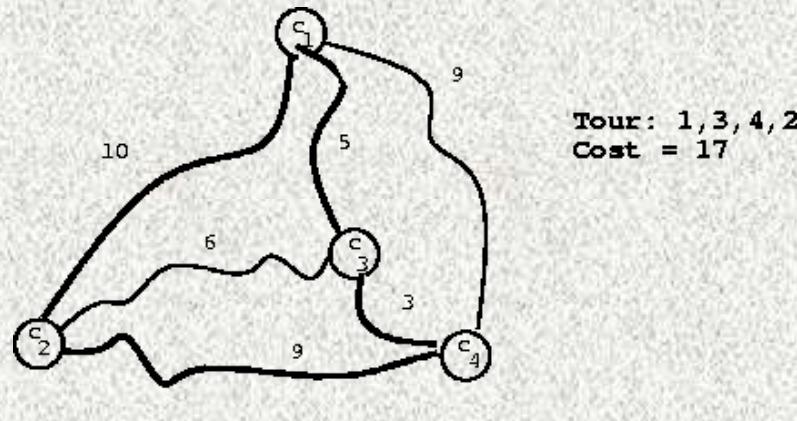
- Hill climbing
- Simulated annealing
- Genetic algorithms

Iterative Improvement Algorithms

- For many optimization problems, **solution path** is irrelevant
 - Just want to reach goal state
- State space / search space
 - Set of “complete” configurations
 - Want to find optimal configuration
(or at least one that satisfies goal constraints)
- For these cases, use iterative improvement algorithm
 - Keep a single current state
 - Try to improve it
- Constant memory

Example

- Traveling salesman

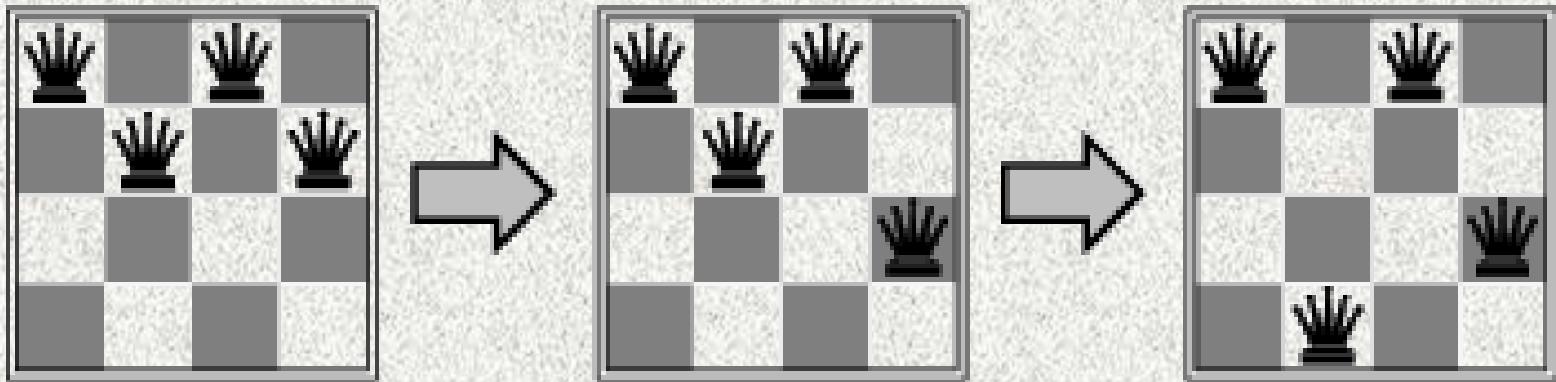


TRAVELING SALESMAN PROBLEM

- Start with any complete tour
- Operator: Perform pairwise exchanges

Example

- N-queens



- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- Operator: Move queen to reduce #conflicts

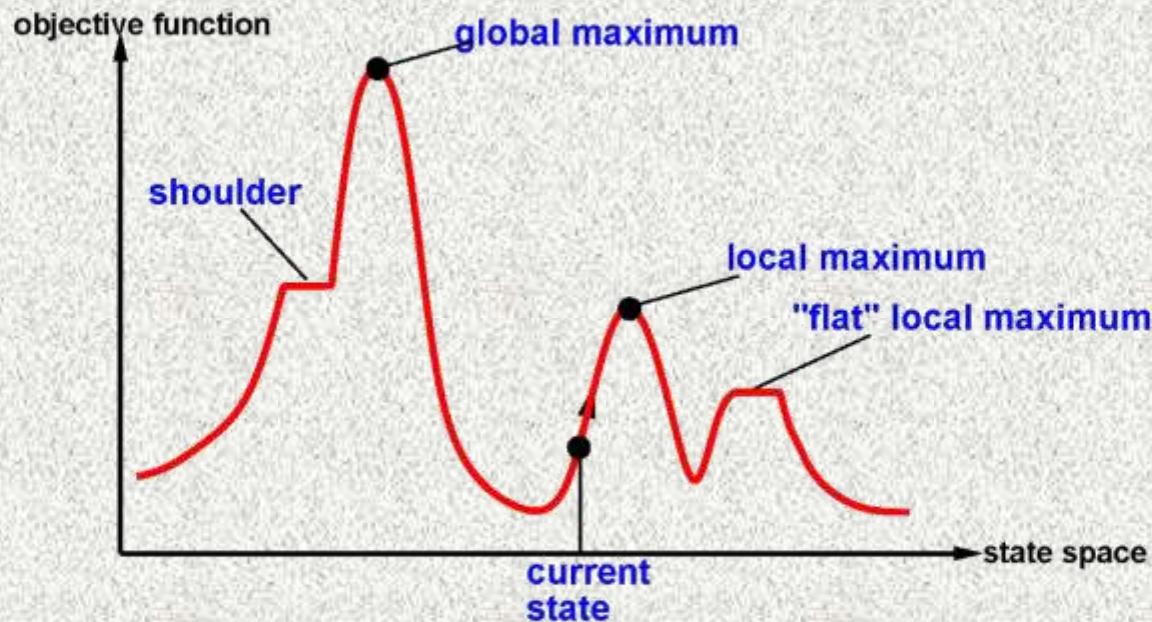
Hill Climbing (gradient ascent/descent)

- “Like climbing Mount Everest in thick fog with amnesia”

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if VALUE[neighbor]  $\leq$  VALUE[current] then return STATE[current]
    current  $\leftarrow$  neighbor
```

Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😊 escape from shoulders 😢 loop on flat maxima

Local Beam Search

- Keep k states instead of 1
- Choose top k of all successors
- Problem
 - Many times all k states end up on same local hill
 - Choose k successors RANDOMLY
 - Bias toward good ones
- Similar to natural selection

Simulated Annealing

- Pure hill climbing is not complete, but pure random search is inefficient.
- Simulated annealing offers a compromise.
- Inspired by **annealing** process of gradually cooling a liquid until it changes to a low-energy state.
- Very similar to hill climbing, except include a user-defined **temperature schedule**.
- When temperature is “high”, allow some random moves.
- When temperature “cools”, reduce probability of random move.
- If T is decreased slowly enough, guaranteed to reach best state.

Algorithm

```
function SimulatedAnnealing(problem, schedule) // returns solution state
    current = MakeNode(Initial-State(problem))
    for t = 1 to infinity
        T = schedule[t]
        if T = 0
            return current
        next = randomly-selected child of current
        ΔE = Value[next] - Value[current]
        if ΔE > 0
            current = next                                // if better than accept state
        else current = next with probability  $e^{\frac{-\Delta E}{T}}$ 
```

[Simulated annealing applet](#)

[Traveling salesman simulated annealing applet](#)

Genetic Algorithms

- What is a Genetic Algorithm (GA)?
 - An adaptation procedure based on the mechanics of natural genetics and natural selection
- Gas have 2 essential components
 - Survival of the fittest
 - Recombination
- Representation
 - Chromosome = string
 - Gene = single bit or single subsequence in string, represents 1 attribute



Humans

- DNA made up of 4 nucleic acids (4-bit code)
- 46 chromosomes in humans, each contain 3 billion DNA
- $4^{3 \text{ billion}}$ combinations of bits
- Can random search find humans?
 - Assume only 0.1% genome must be discovered, $3(10^6)$ nucleotides
 - Assume very short generation, 1 generation/second
 - $3.2(10^{10^7})$ individuals per year, but $10^{1.8(10^7)}$ alternatives
 - $10^{18}10^6$ years to generate human randomly
- Self reproduction, self repair, adaptability are the rule in natural systems, they hardly exist in the artificial world
- Finding and adopting nature's approach to computational design should unlock many doors in science and engineering

GAs Exhibit Search

- Each attempt a GA makes towards a solution is called a **chromosome**
 - A sequence of information that can be interpreted as a possible solution
- Typically, a chromosome is represented as sequence of binary digits
 - Each digit is a **gene**
- A GA maintains a collection or **population** of chromosomes
 - Each chromosome in the population represents a different guess at the solution

The GA Procedure

1. Initialize a population (of solution guesses)
2. Do (once for each generation)
 - a. Evaluate each chromosome in the population using a **fitness function**
 - b. Apply GA operators to population to create a new population
3. Finish when solution is reached or number of generations has reached an allowable maximum.

Common Operators

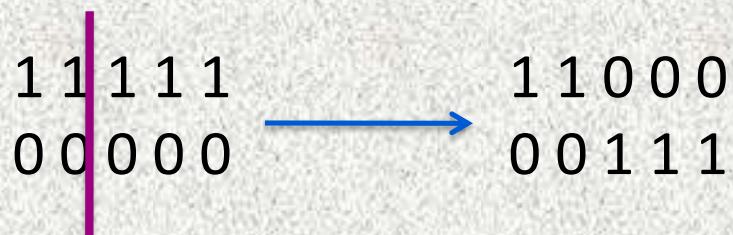
- Reproduction
- Crossover
- Mutation

Reproduction

- Select individuals x according to their fitness values $f(x)$
 - Like beam search
- Fittest individuals survive (and possibly mate) for next generation

Crossover

- Select two parents
- Select cross site
- Cut and splice pieces of one parent to those of the other



Mutation

- With small probability, randomly alter 1 bit
- Minor operator
- An insurance policy against lost bits
- Pushes out of local minima

Population:

1 1 0 0 0
1 0 1 0 0
1 0 0 1 0
0 1 0 0 0

Goal: 0 1 1 1 1

Mutation needed to find the goal

Example

- Solution = 0 0 1 0 1 0
 - Fitness(x) = #digits that match solution
- | | |
|----------------|----------|
| A) 0 1 0 1 0 1 | Score: 1 |
| B) 1 1 1 1 0 1 | Score: 1 |
| C) 0 1 1 0 1 1 | Score: 3 |
| D) 1 0 1 1 0 0 | Score: 3 |

Recombine top two twice.

Note: 64 possible combinations

Example

- Solution = 0 0 1 0 1 0

C) 0 1 1 0 1 1

D) 1 0 1 1 0 0

E) 0 | 0 1 1 0 0

F) 1 | 1 1 0 1 1

G) 0 1 1 0 1 | 0

H) 1 0 1 1 0 | 1

- Next generation:

E) 0 0 1 1 0 0

F) 0 1 1 0 1 0

G) 0 1 1 | 1 0 0 Score: 3

H) 0 0 1 | 0 1 0 Score: 6

I) 0 0 | 1 0 1 0 Score: 6

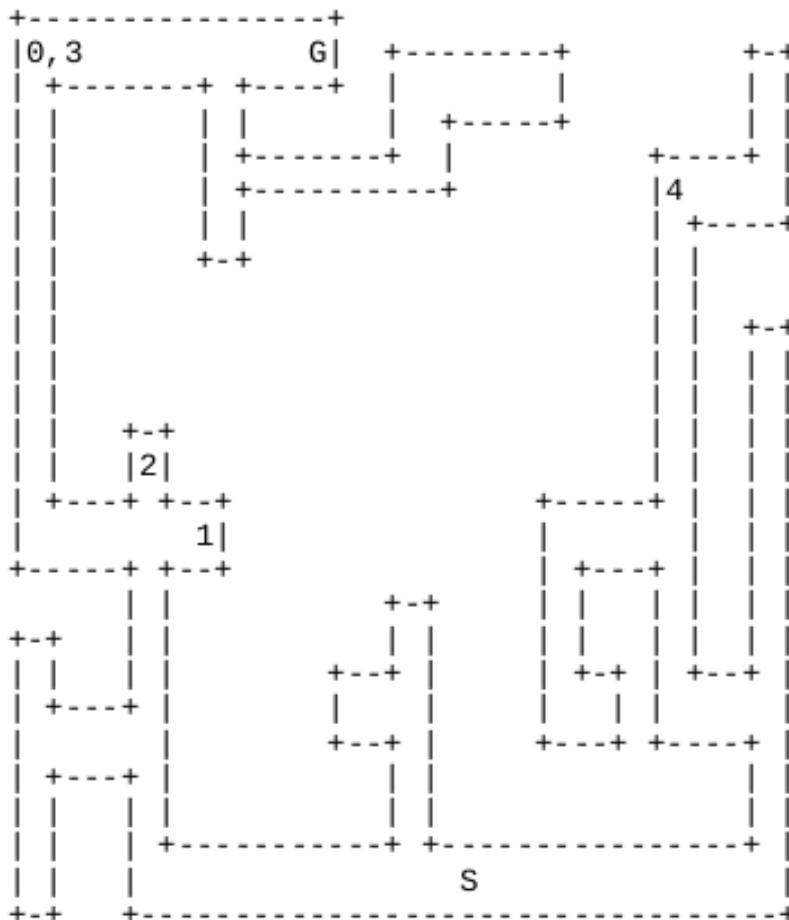
J) 0 1 | 1 1 0 0 Score: 3

DONE! Got it in 10 guesses.

Issues

- How select original population?
- How handle non-binary solution types?
- What should be the size of the population?
- What is the optimal mutation rate?
- How are mates picked for crossover?
- Can any chromosome appear more than once in a population?
- When should the GA halt?
- Local minima?
- Parallel algorithms?

GAs for Mazes



Representation:

Sequence of directions:
N^, E>, W<, Sv

Solution:

Gene that leads from S to G

At each decision point in maze,
look up next direction.

N E W S W E

Fitness Function =
distance at end from goal
(#steps)

Note: make gene long to give
slack

- 0) EWWNNEWWSWNWW
f=43 (1,1) 15 moves
- 1) WSWNEESSESWNENE
f=43
- 2) SNEESEWWNNNSEWN
f=43
- 3) EEWWSNNSENWNENW
f=43
- 4) WEEWSENWSWWSNEN
f=42

Takes 40 generations to solve

GAs for Optimization

- Traveling salesman problem
- Eaters
- Hierarchical GAs for game playing

GAs for Control

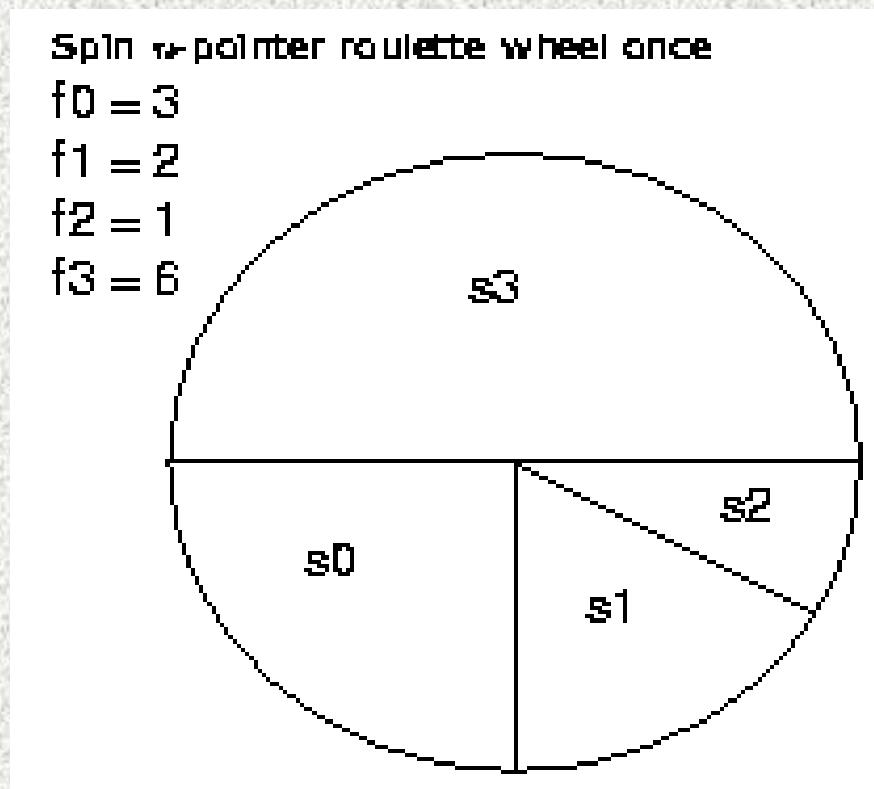
- Simulator

GAs for Graphic Animation

- Simulator
- Evolving Circles
- 3D Animation
- Scientific American Frontiers

Biased Roulette Wheel

- For each hypothesis, spin the roulette wheel to determine the guess



Inversion

- Invert selected subsequence
- 1 0 | 1 1 0 | 1 1 -> 1 0 0 1 1 1 1

Elitism

- Some of the best chromosomes from previous generation replace some of the worst chromosomes from current generation

K-point crossover

- Pick k random splice points to crossover parents
- Example
 - K = 3

1 1 1 1 1 1 1 1 1 1 1 1	->	1 1 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0		0 0 1 1 1 0 0 1 1 1 1 1

Diversity Measure

- Fitness ignores diversity
- As a result, populations tend to become uniform
- Rank-space method
 - Sort population by sum of fitness rank and diversity rank
 - Diversity rank is the result of sorting by the function $1/d^2$

Classifier Systems

- GAs and load balancing
- SAMUEL

CptS 440 / 540

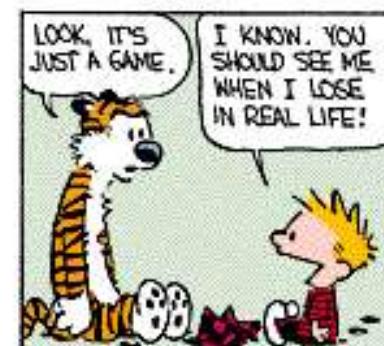
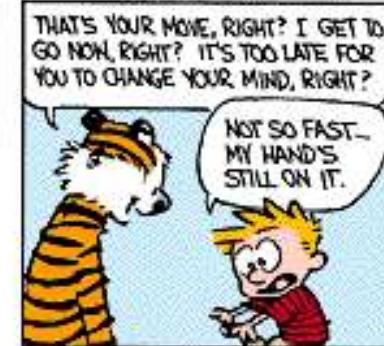
Artificial Intelligence

Adversarial Search

Game Playing

Calvin and Hobbes

by MCKEON



Why Study Game Playing?

- Games allow us to experiment with easier versions of real-world situations
- Hostile agents act against our goals
- Games have a finite set of moves
- Games are fairly easy to represent
- Good idea to decide about what to think
- Perfection is unrealistic, must settle for good
- One of the earliest areas of AI
 - Claude Shannon and Alan Turing wrote chess programs in 1950s
- The opponent introduces uncertainty
- The environment may contain uncertainty (backgammon)
- Search space too hard to consider exhaustively
 - Chess has about 10^{40} legal positions
 - Efficient and effective search strategies even more critical
- Games are fun to target!

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

Zero-Sum Games

- Focus primarily on “adversarial games”
- Two-player, zero-sum games

As Player 1 gains strength

Player 2 loses strength

and vice versa

The sum of the two strengths is always 0.

Search Applied to Adversarial Games

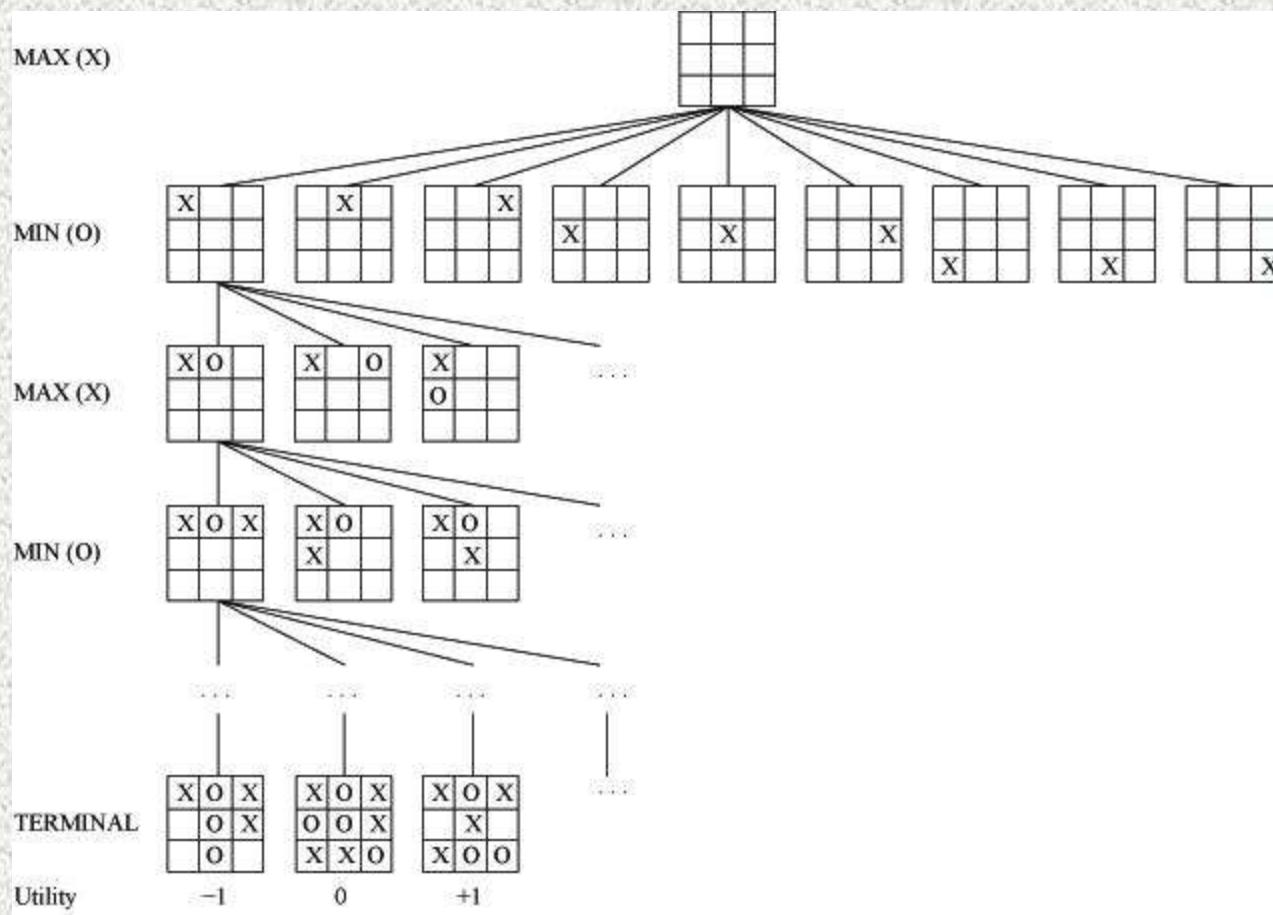
- Initial state
 - Current board position (description of current game state)
- Operators
 - Legal moves a player can make
- Terminal nodes
 - Leaf nodes in the tree
 - Indicate the game is over
- Utility function
 - Payoff function
 - Value of the outcome of a game
 - Example: tic tac toe, utility is -1, 0, or 1

Using Search

- Search could be used to find a perfect sequence of moves except the following problems arise:
 - There exists an adversary who is trying to minimize your chance of winning every other move
 - You cannot control his/her move
 - Search trees can be very large, but you have finite time to move
 - Chess has 10^{40} nodes in search space
 - With single-agent search, can afford to wait
 - Some two-player games have time limits
 - Solution?
 - Search to n levels in the tree (n **ply**)
 - Evaluate the nodes at the nth level
 - Head for the best looking node

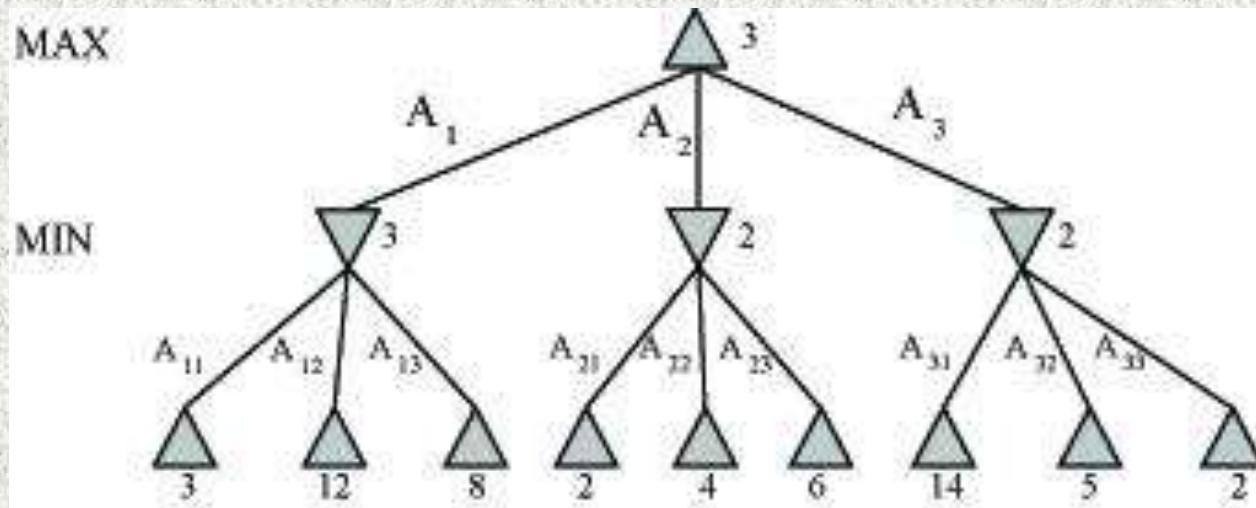
Game Trees

- [Tic tac toe](#)
- Two players, MAX and MIN
- Moves (and levels) alternate between two players



Minimax Algorithm

- Search the tree to the end
- Assign utility values to terminal nodes
- Find the best move for MAX (on MAX's turn), assuming:
 - MAX will make the move that maximizes MAX's utility
 - MIN will make the move that minimizes MAX's utility
- Here, MAX should make the leftmost move
- [Minimax applet](#)



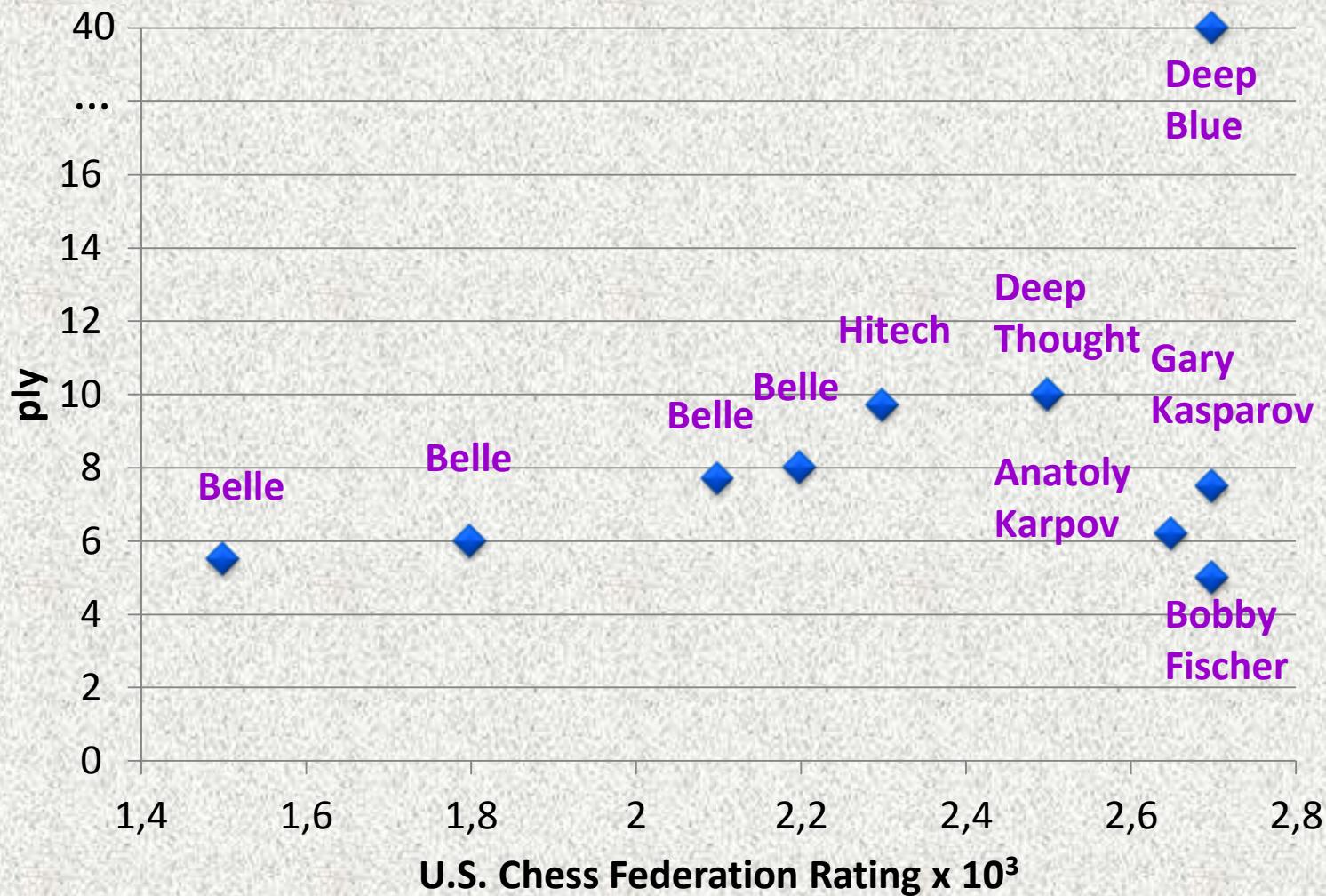
Minimax Properties

- Complete if tree is finite
- Optimal if play against opponent with same strategy (utility function)
- Time complexity is $O(b^m)$
- Space complexity is $O(bm)$ (depth-first exploration)
- If we have 100 seconds to make a move
 - Can explore 10^4 nodes/second
 - Can consider 10^6 nodes / move
- Standard approach is
 - Apply a cutoff test (depth limit, quiescence)
 - Evaluate nodes at cutoff (evaluation function estimates desirability of position)

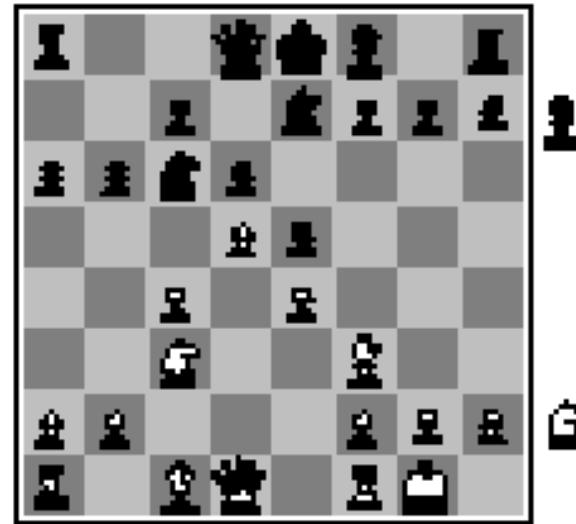
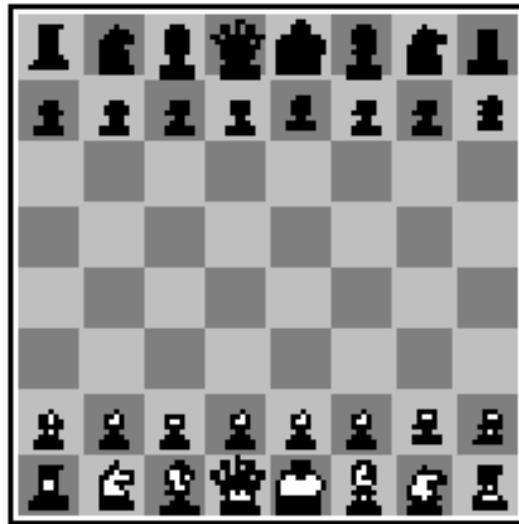
Static Board Evaluator

- We cannot look all the way to the end of the game
 - Look ahead ply moves
 - Evaluate nodes there using SBE
- Tic Tac Toe example
 - #unblocked lines with Xs - #unblocked lines with Os
- Tradeoff
 - Stupid, fast SBE: Massive search
 - These are “Type A” systems
 - Smart, slow SBE: Very little search
 - These are “Type B” systems
 - Humans are Type B systems
 - Computer chess systems have been more successful using Type A
 - They get better by searching more ply

Comparison

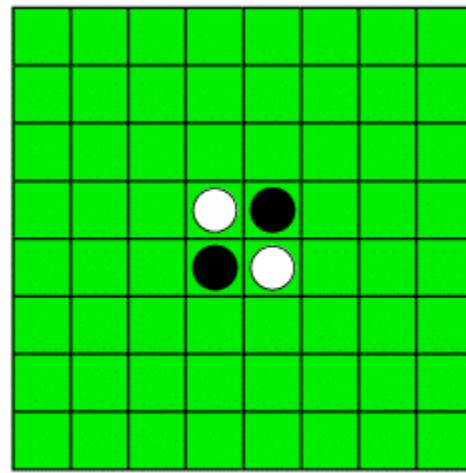


Example



- Chess, SBE is typically linear weighted sum of features
 - $SBE(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
 - E.g., $w_1 = 9$
 - $F_1(s) = \#white\ queens - \#black\ queens$
- For chess:
 - 4 ply is human novice
 - 8 ply is typical PC or human master
 - 12 ply is grand master

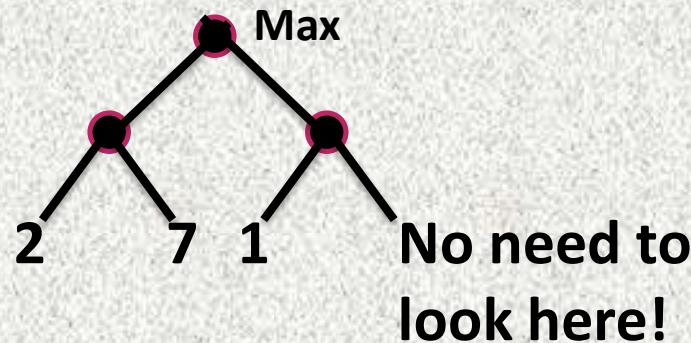
Example



- Othello
- SBE1: #white pieces - #black pieces
- SBE2: weighted squares

Alpha-Beta Pruning

- Typically can only look 3-4 ply in allowable chess time
- Alpha-beta pruning simplifies search space without eliminating optimality
 - By applying common sense
 - If one route allows queen to be captured and a better move is available
 - Then don't search further down bad path
 - If one route would be bad for opponent, ignore that route also

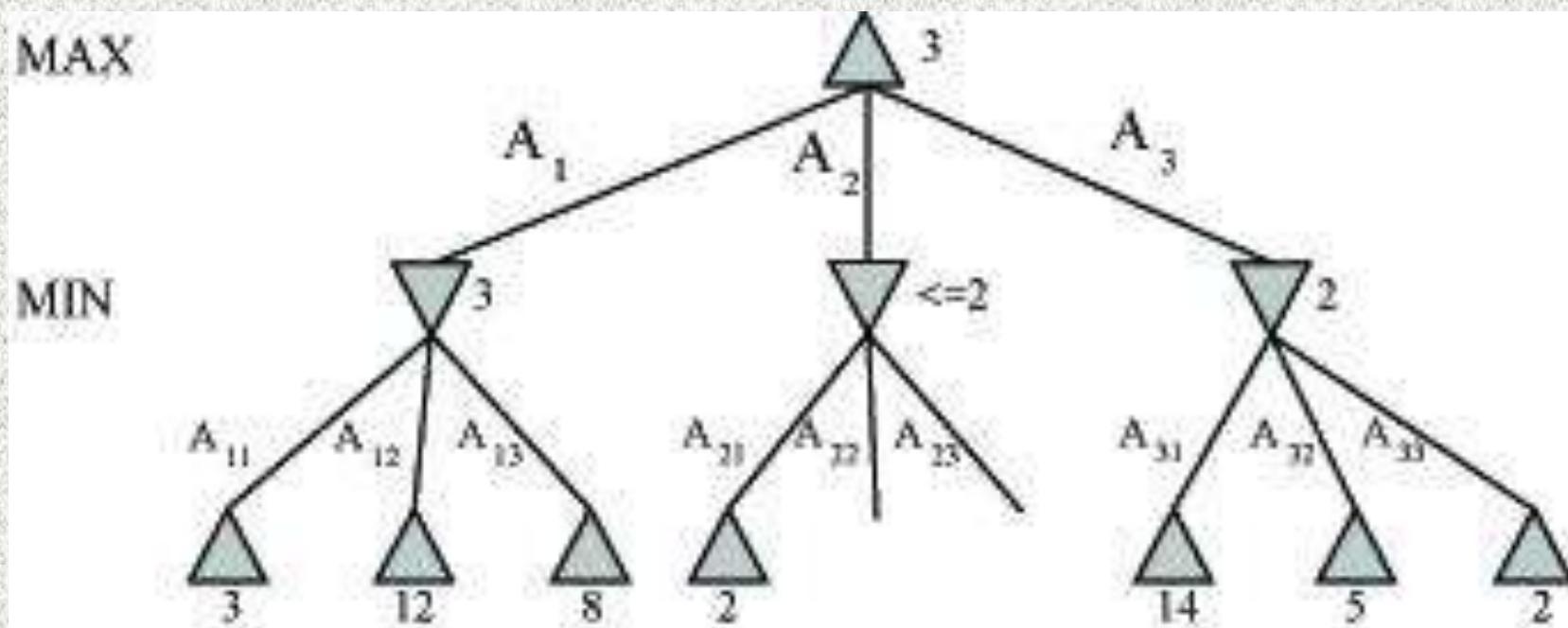


Maintain [alpha, beta] window at each node during depth-first search

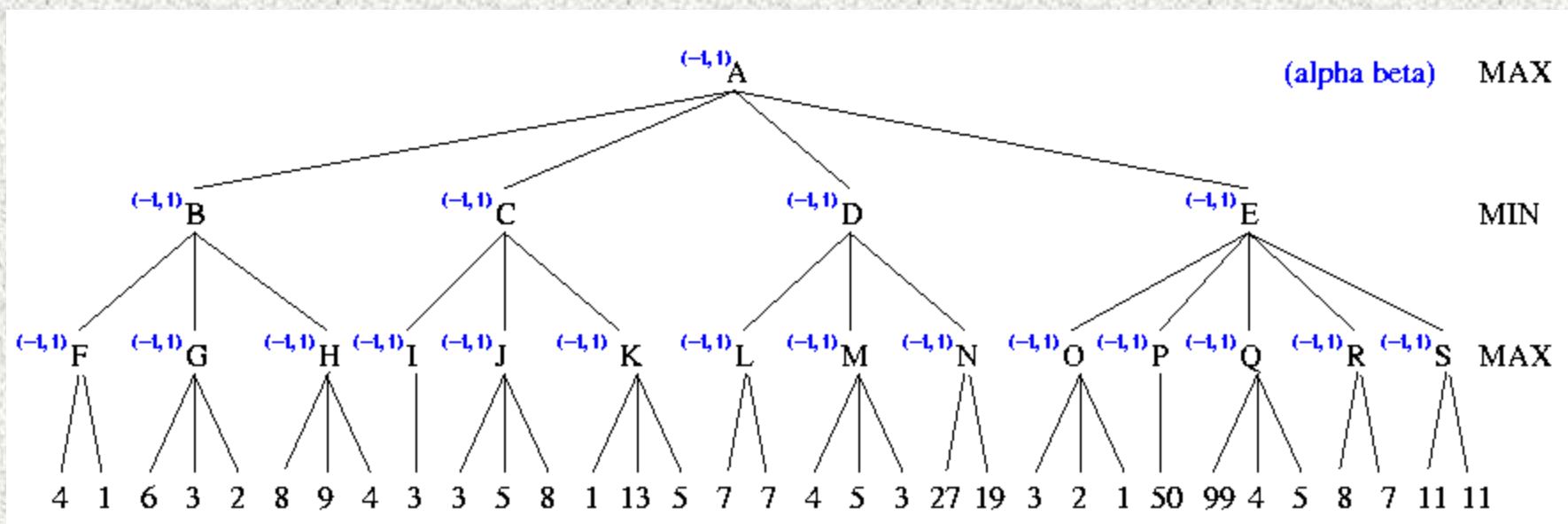
alpha = lower bound, change at max levels

beta = upper bound, change at min levels

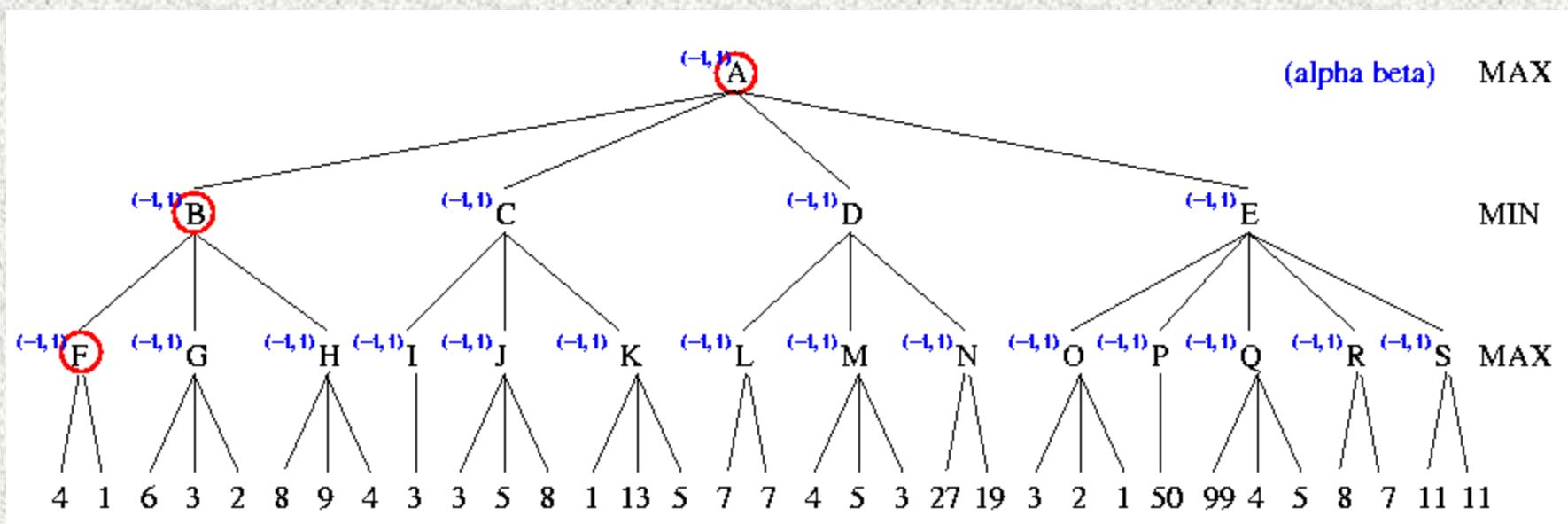
Example



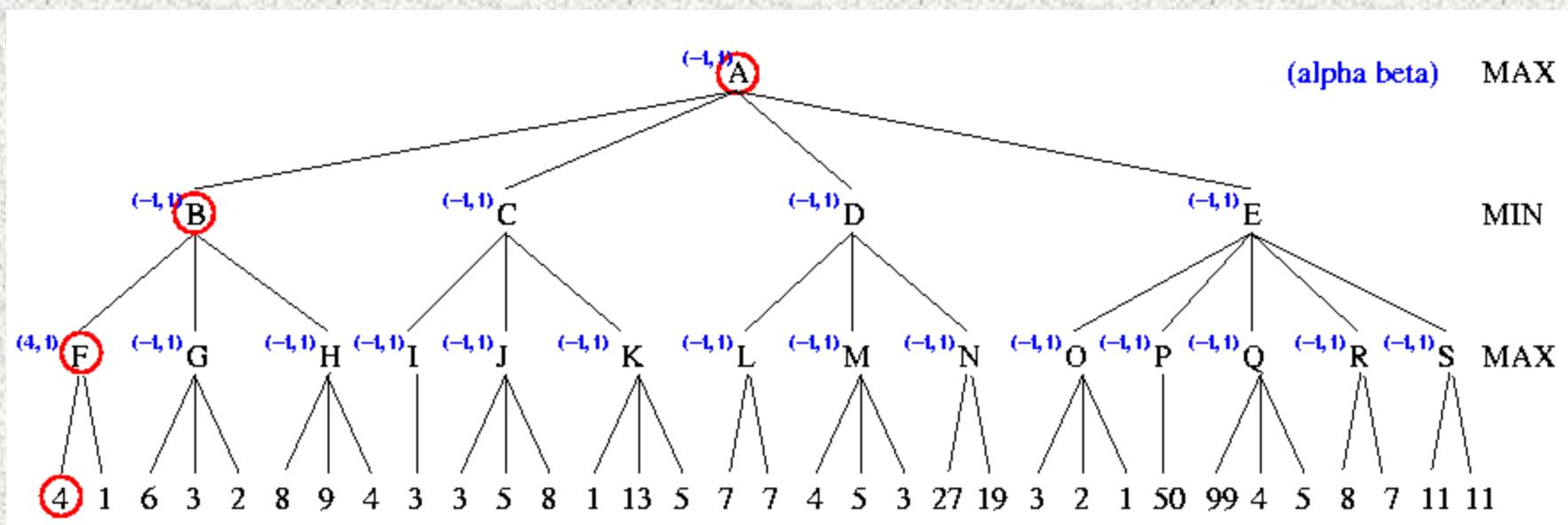
Example



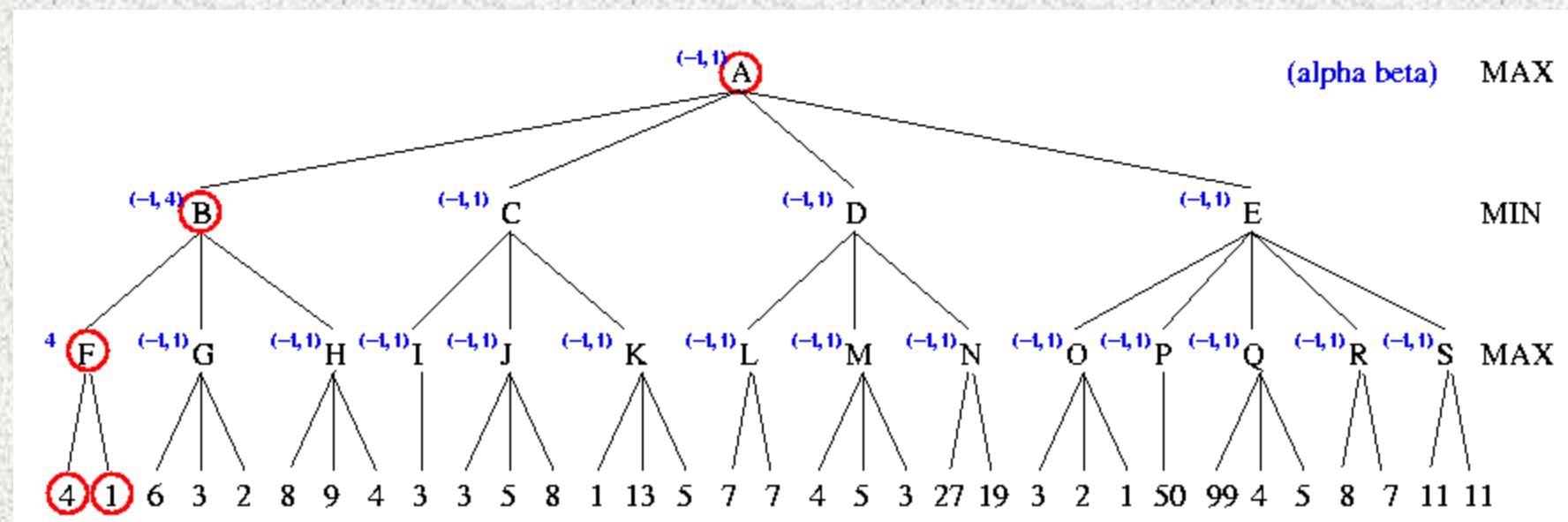
Example



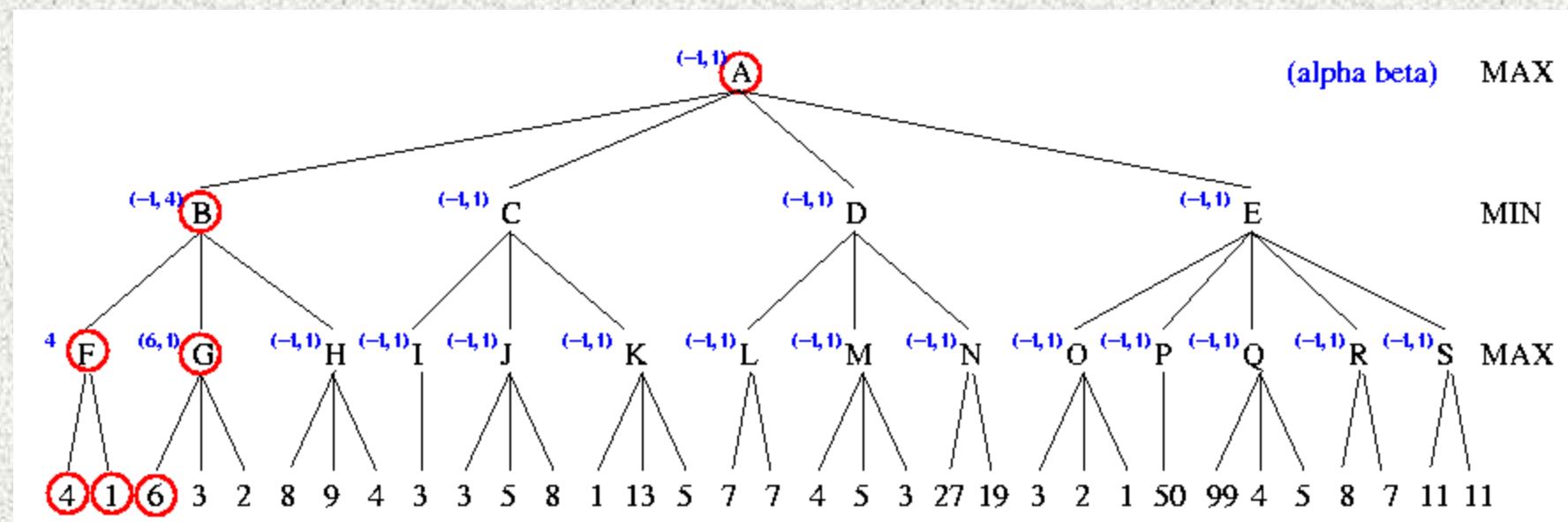
Example



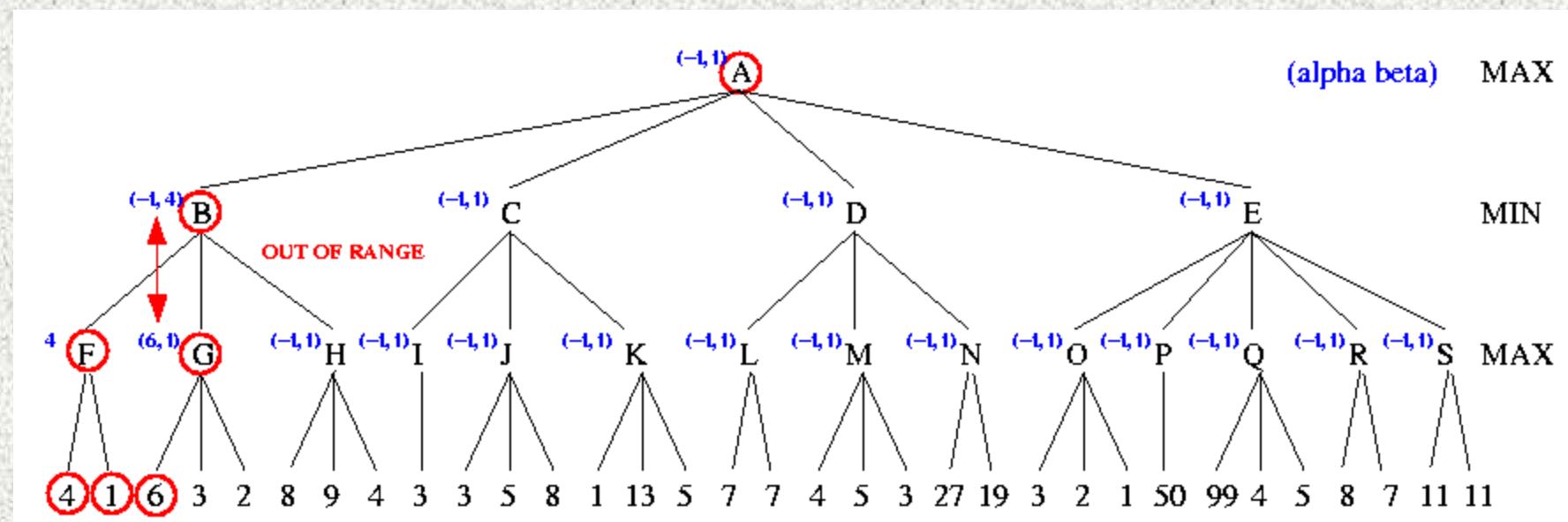
Example



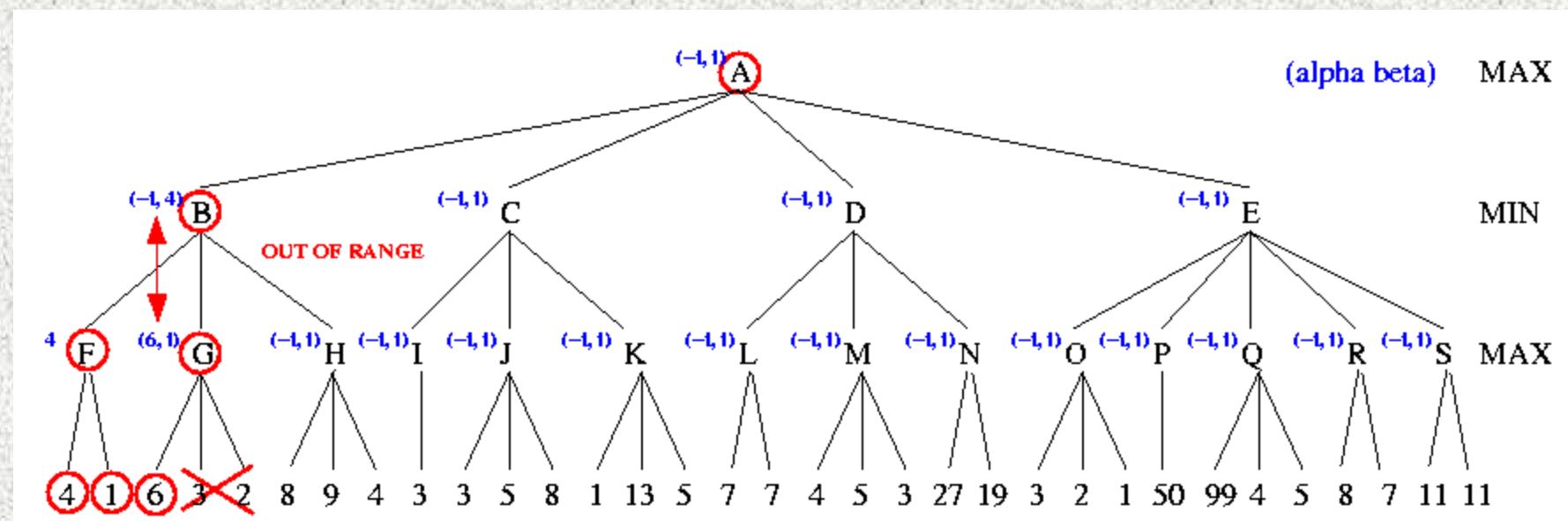
Example



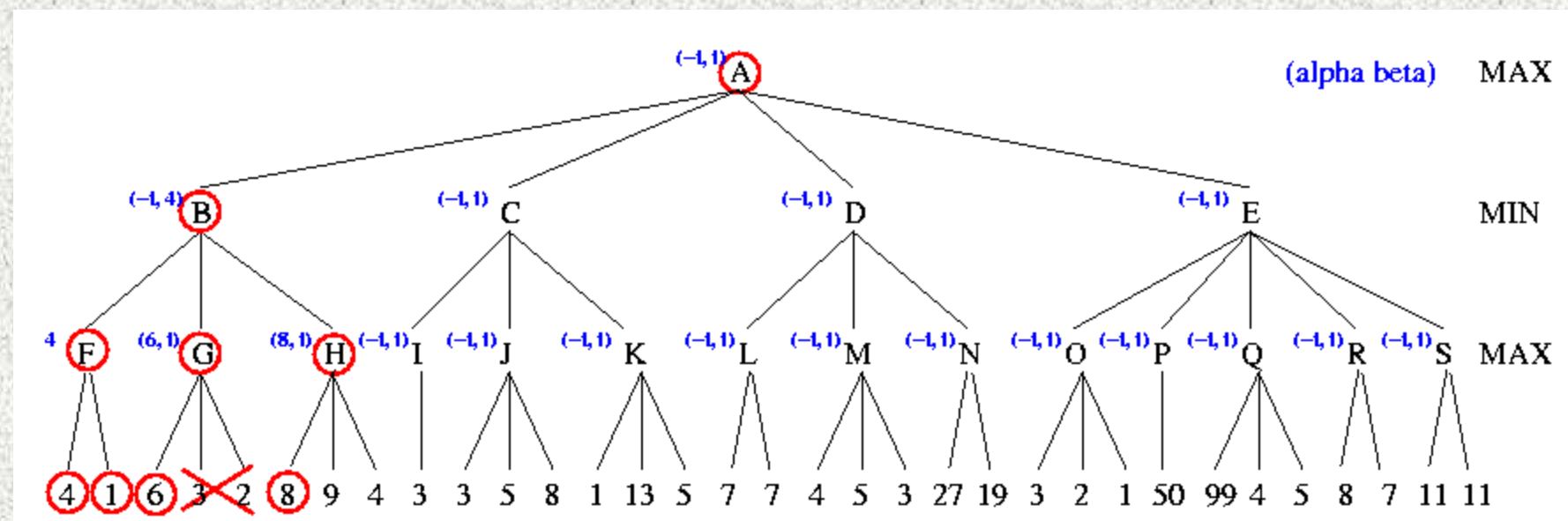
Example



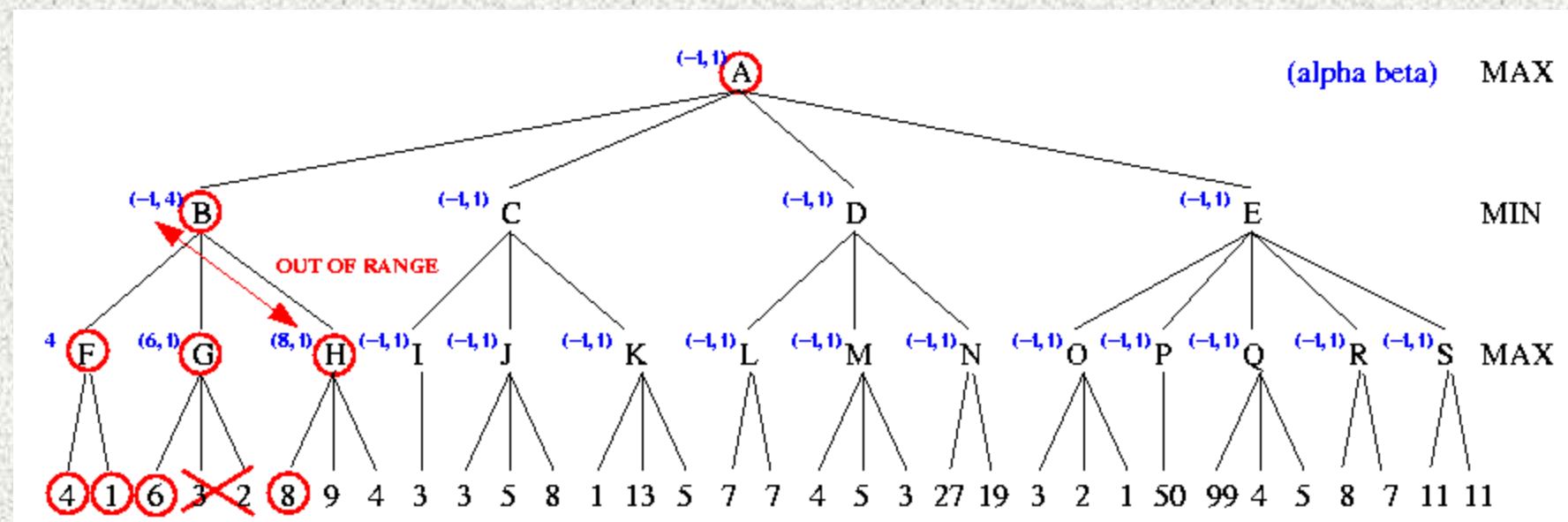
Example



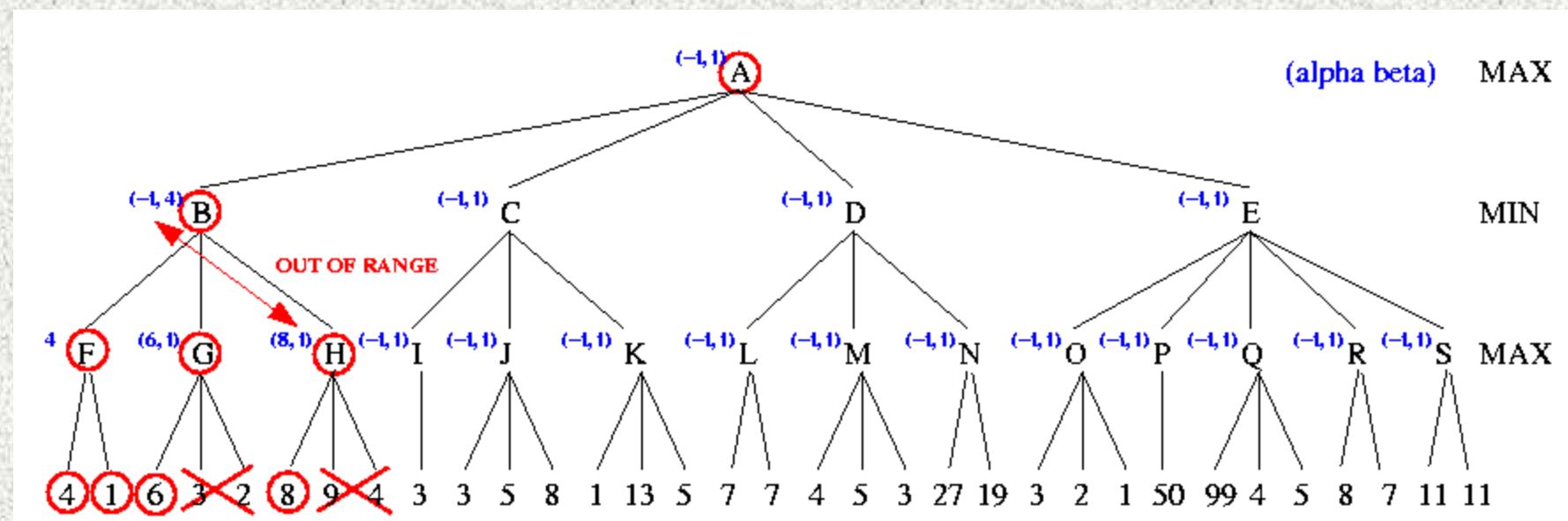
Example



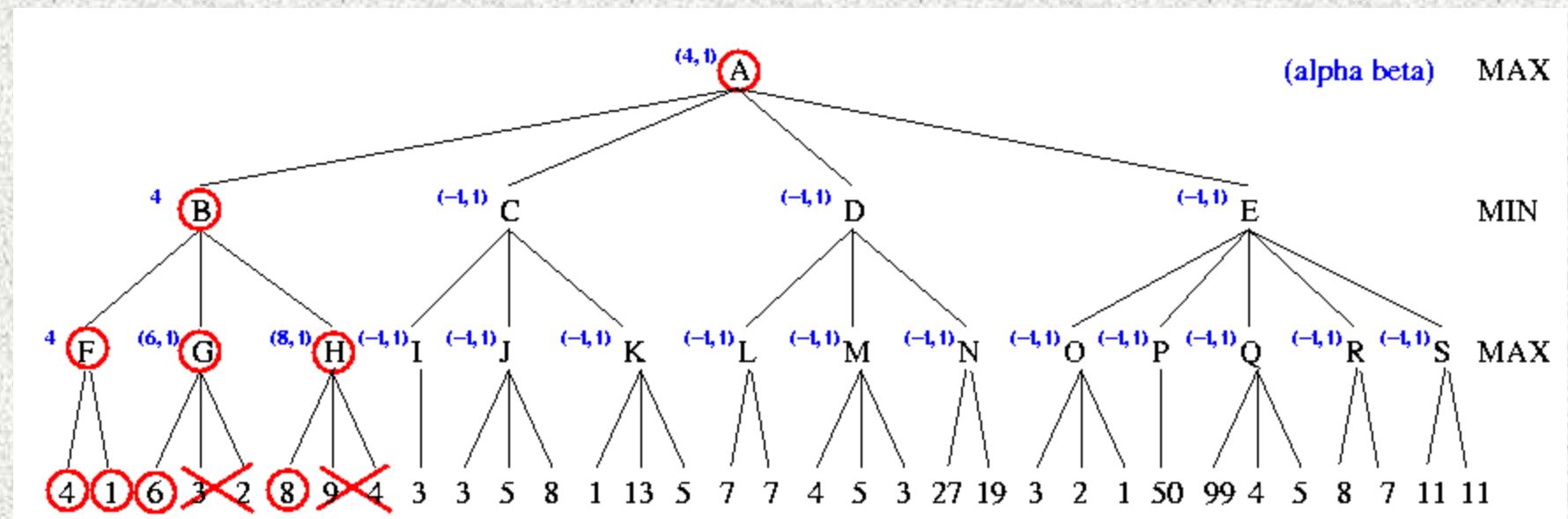
Example



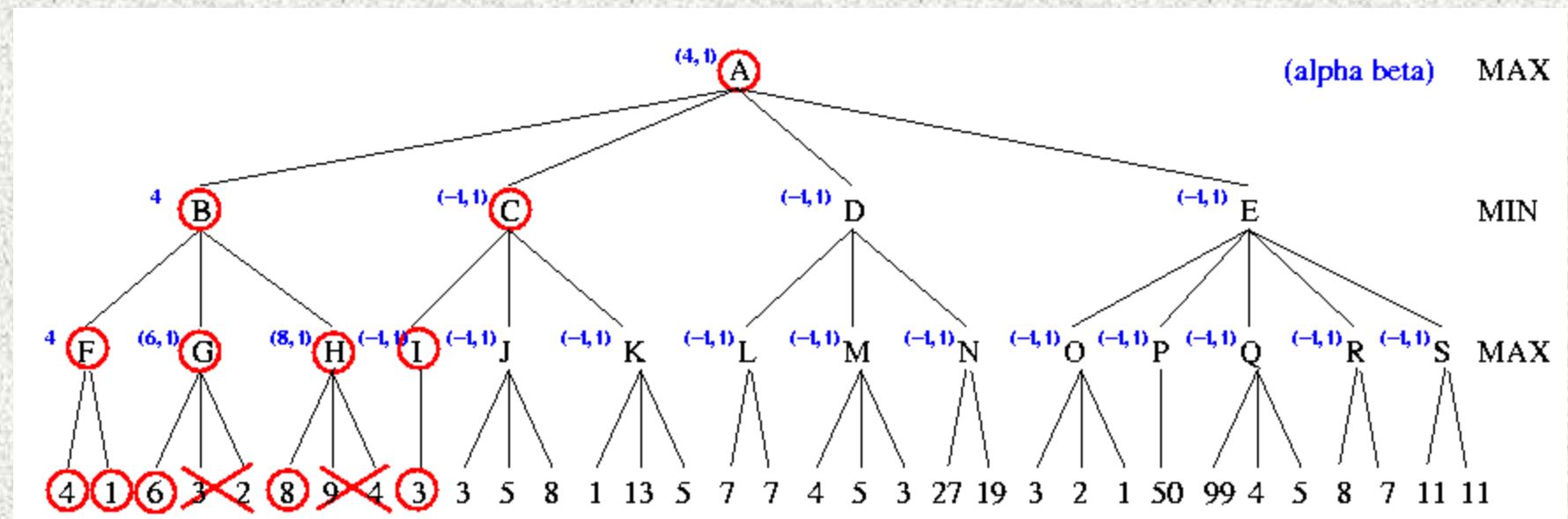
Example



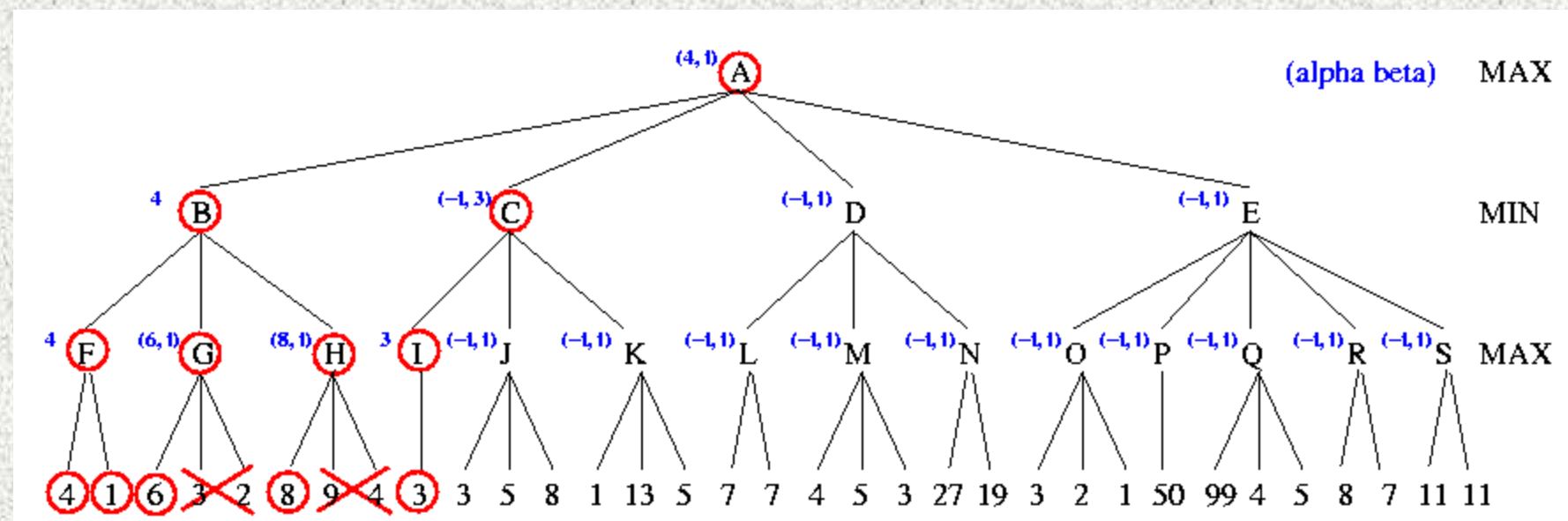
Example



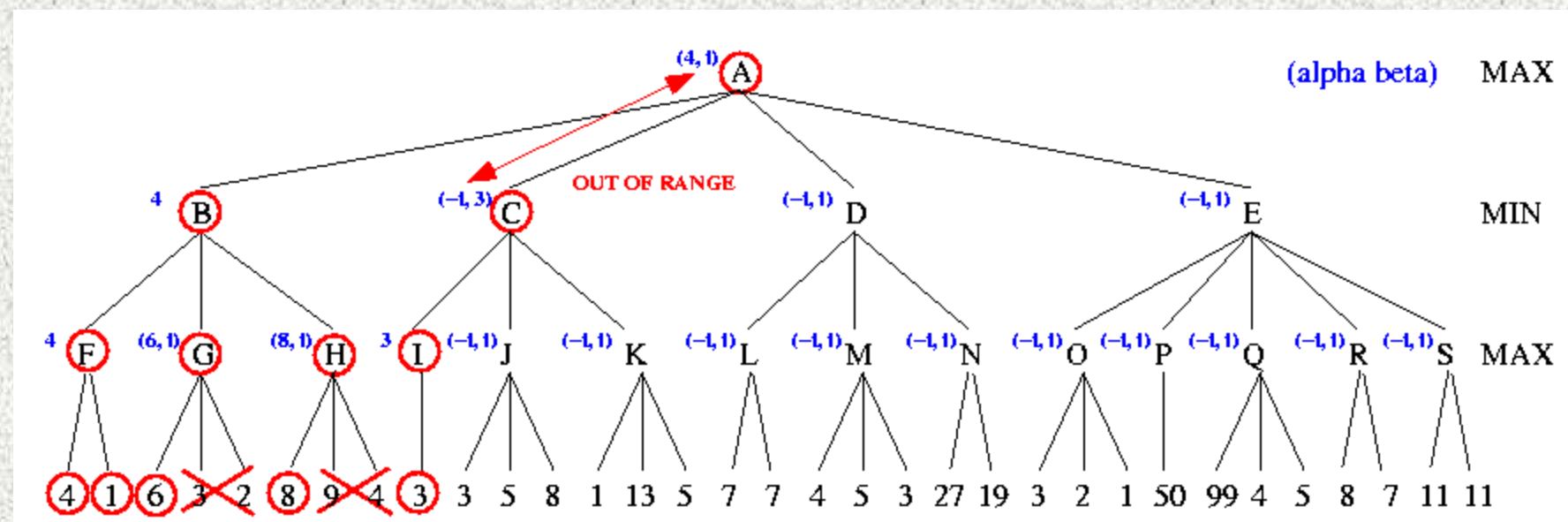
Example



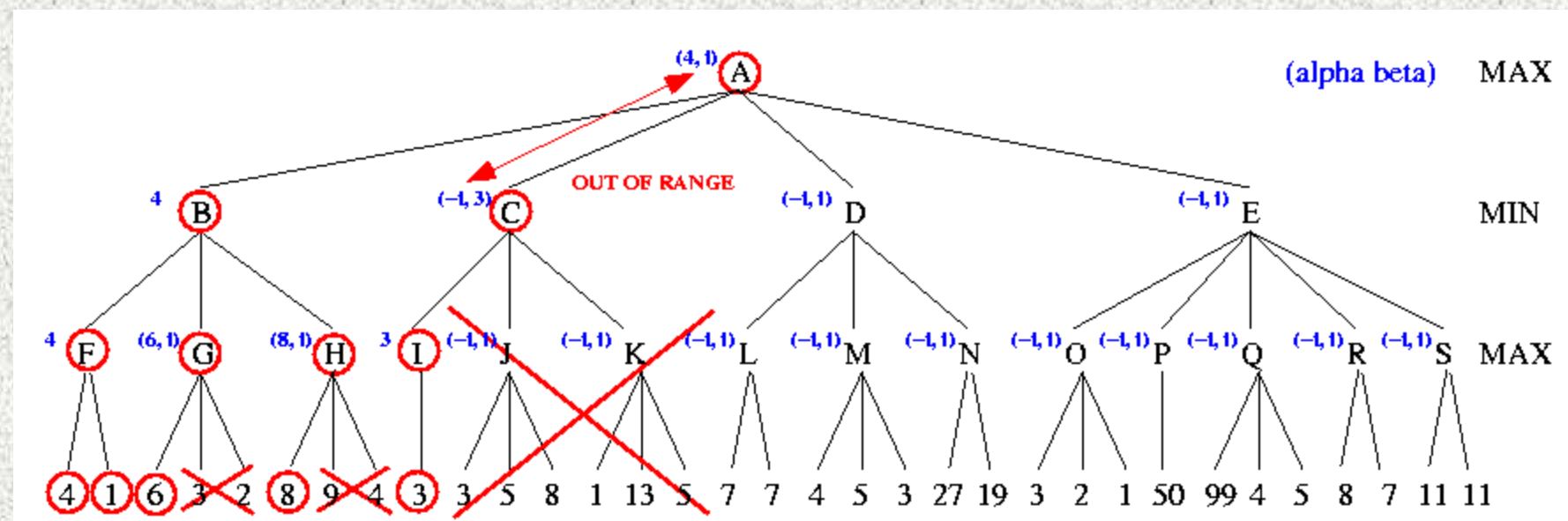
Example



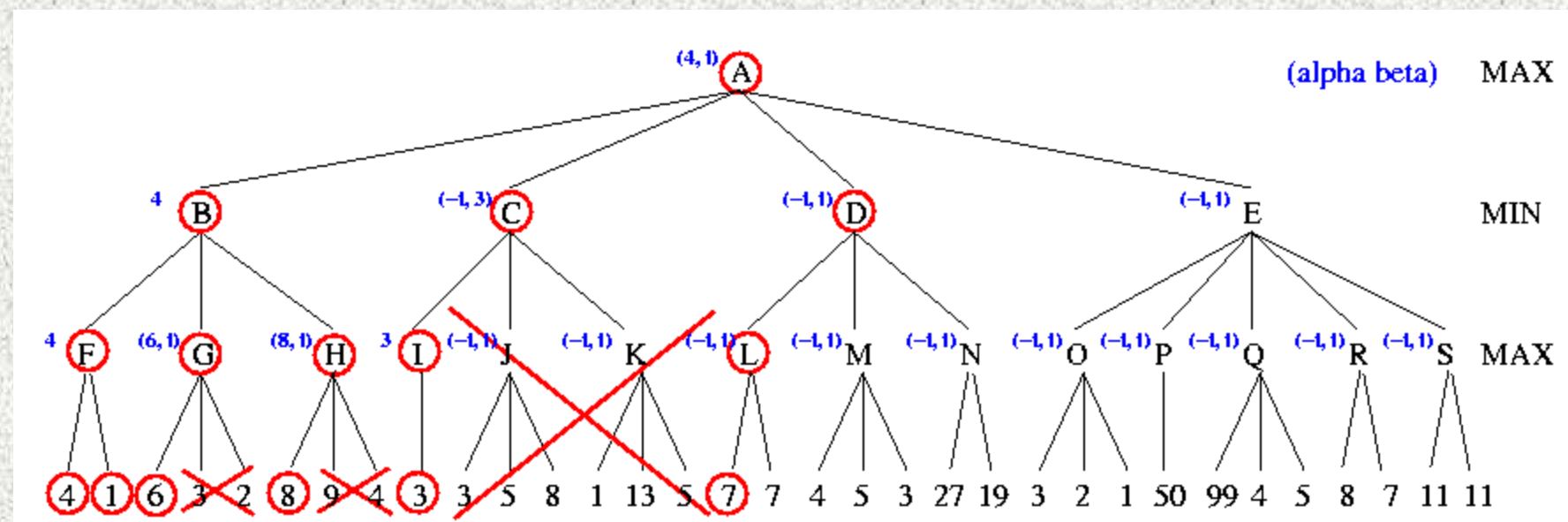
Example



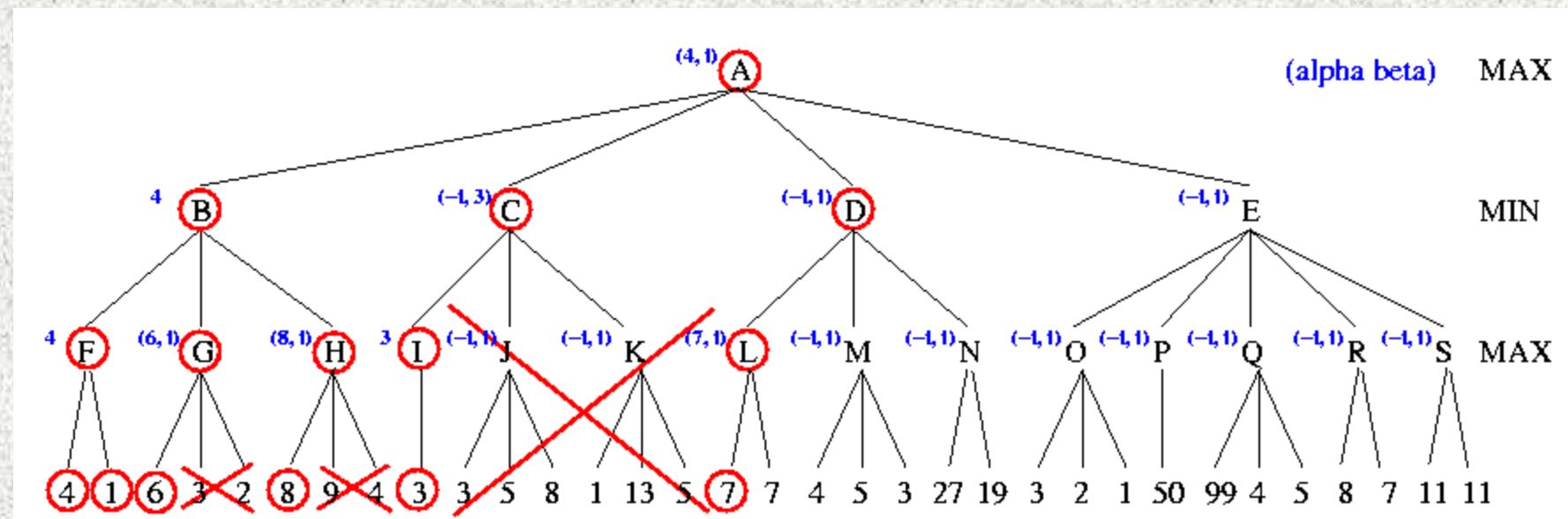
Example



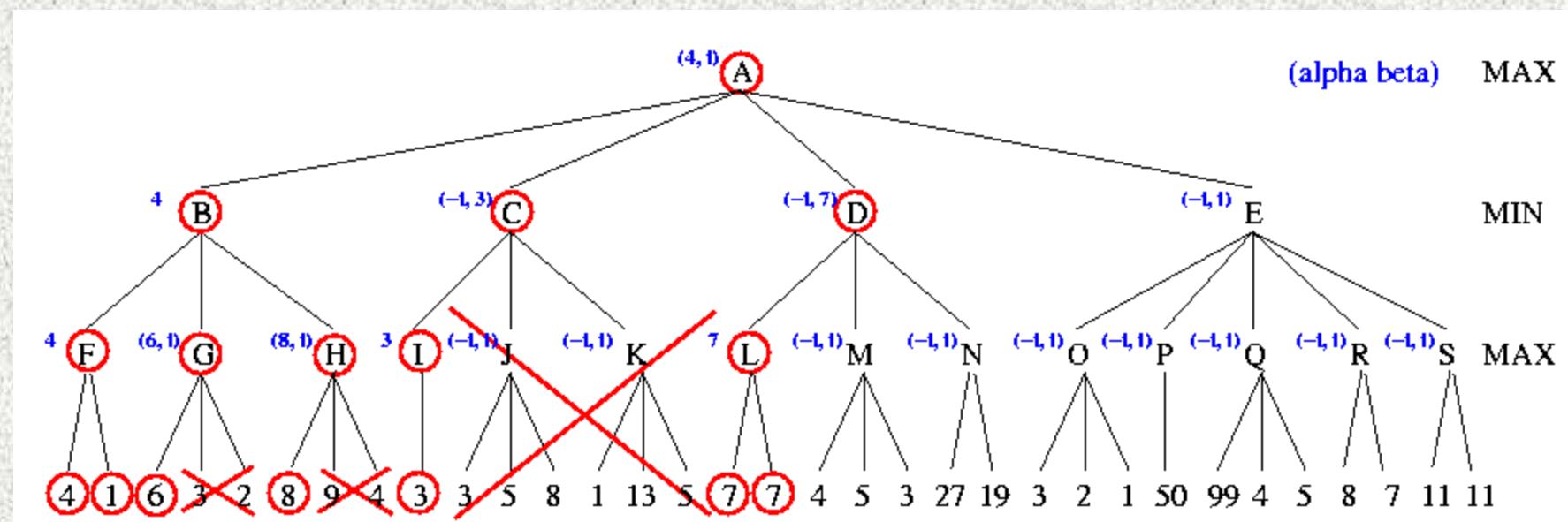
Example



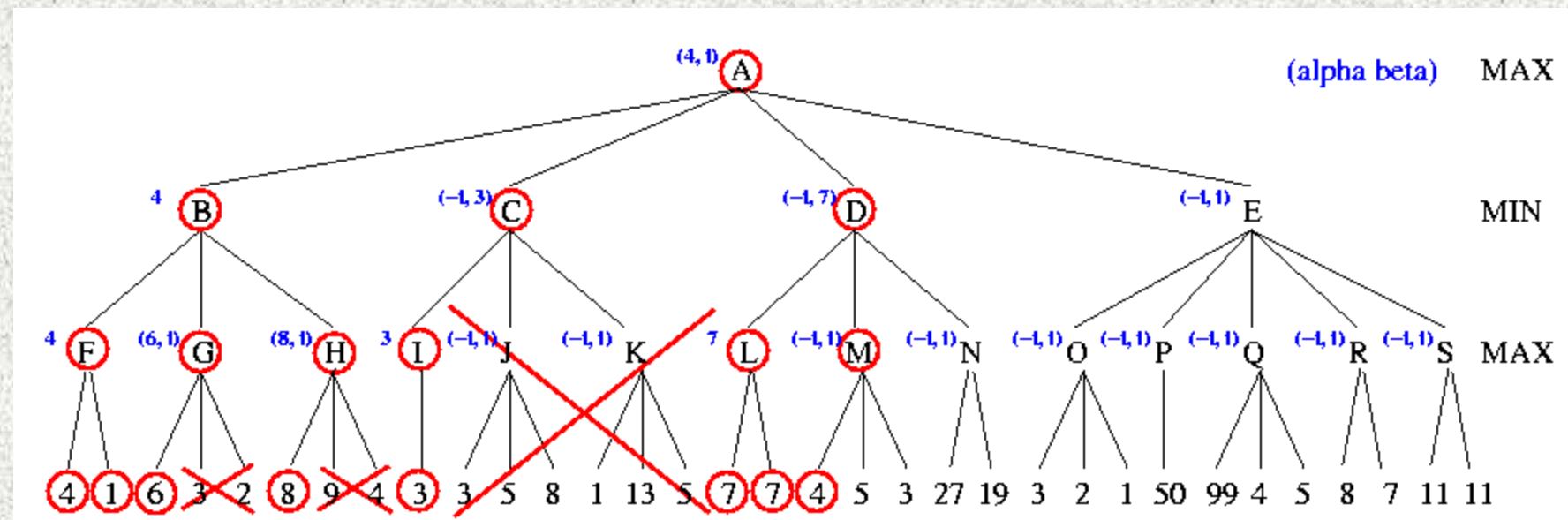
Example



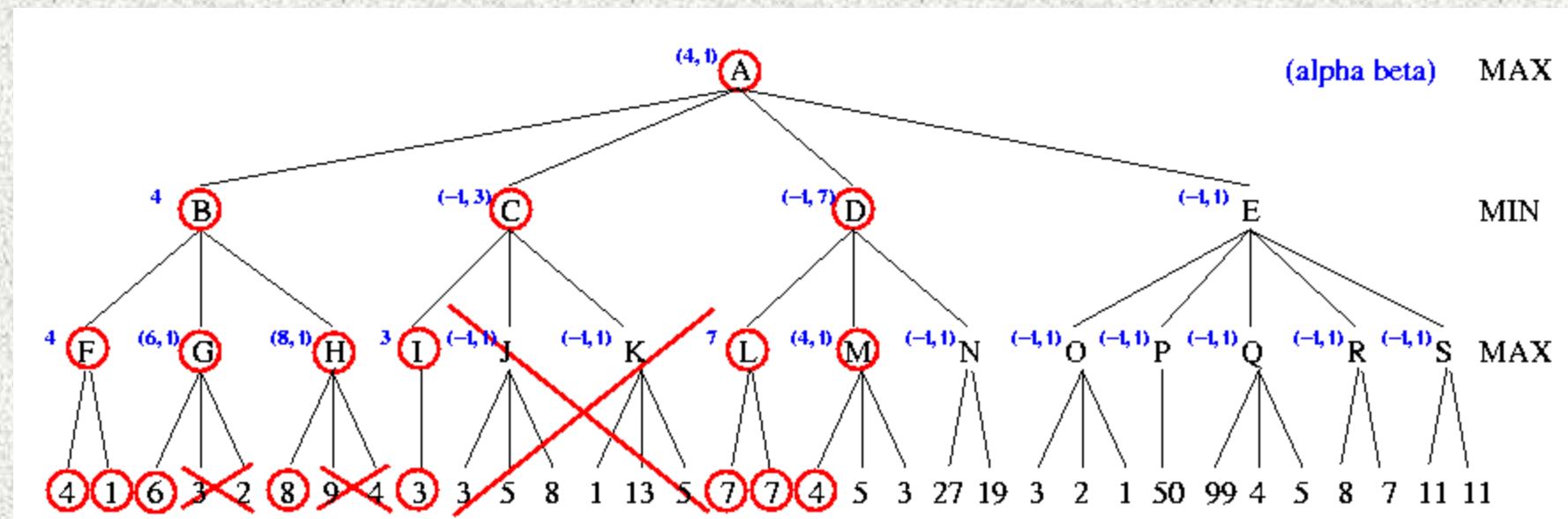
Example



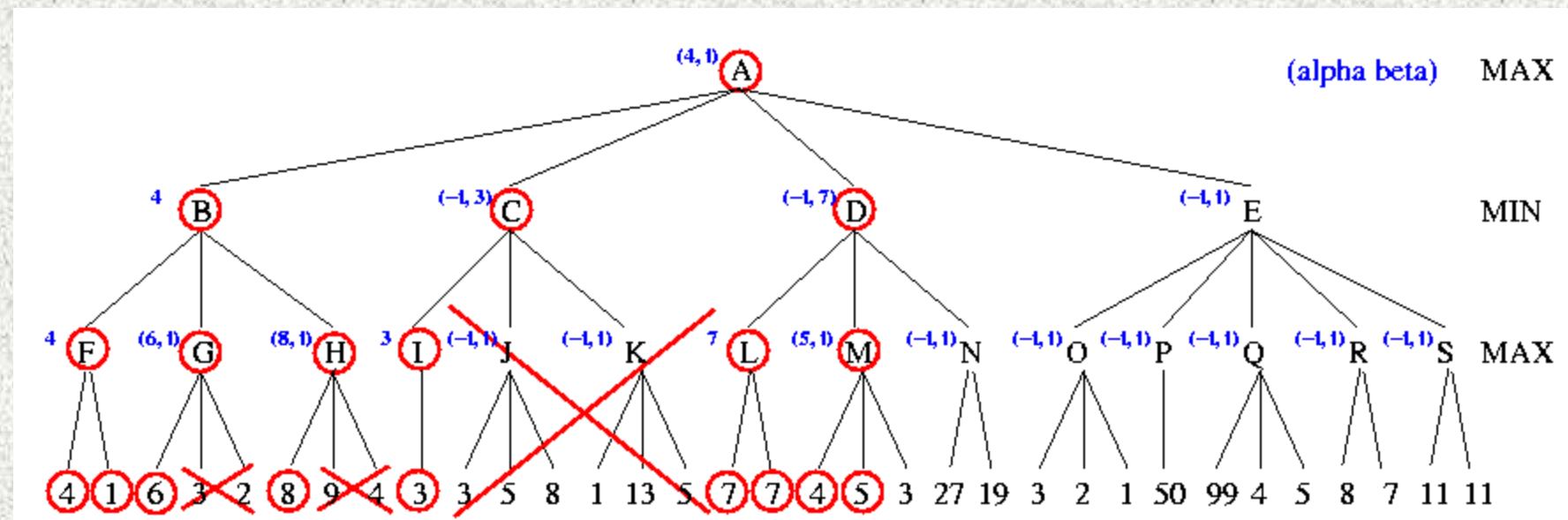
Example



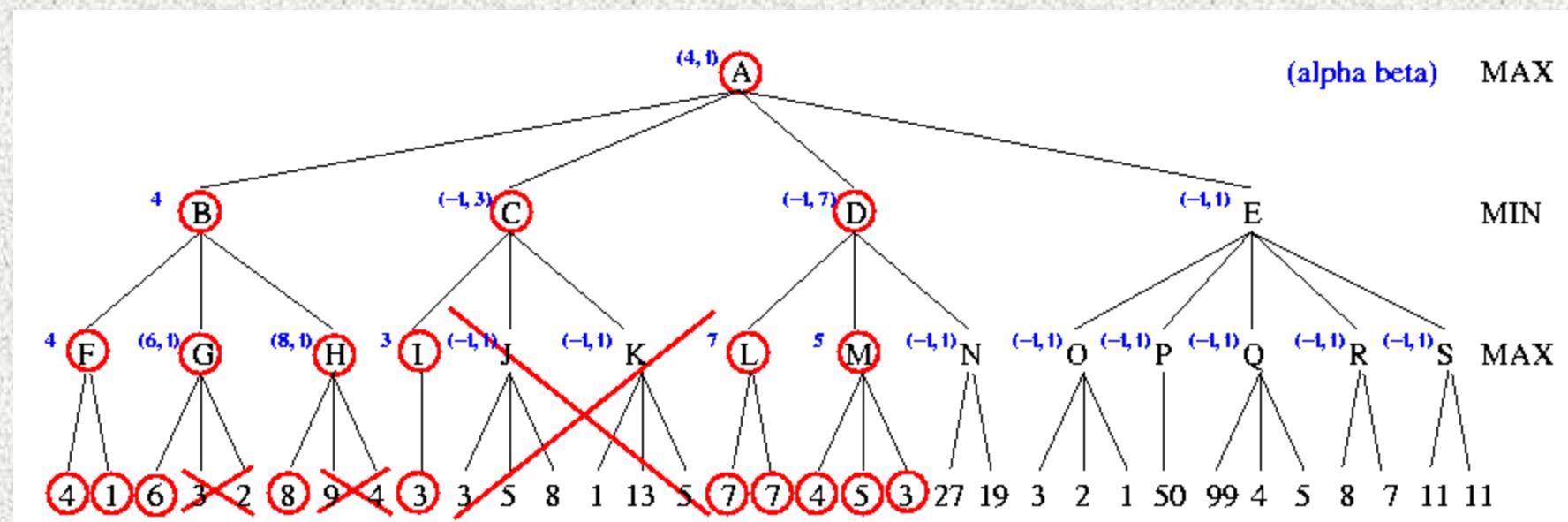
Example



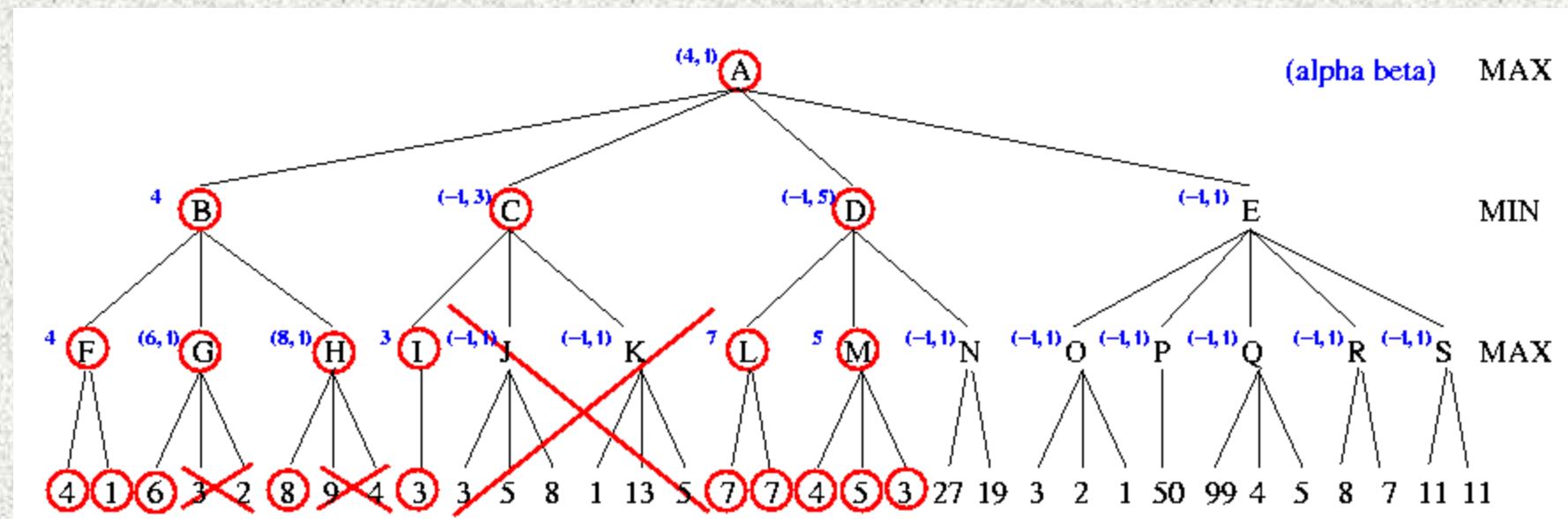
Example



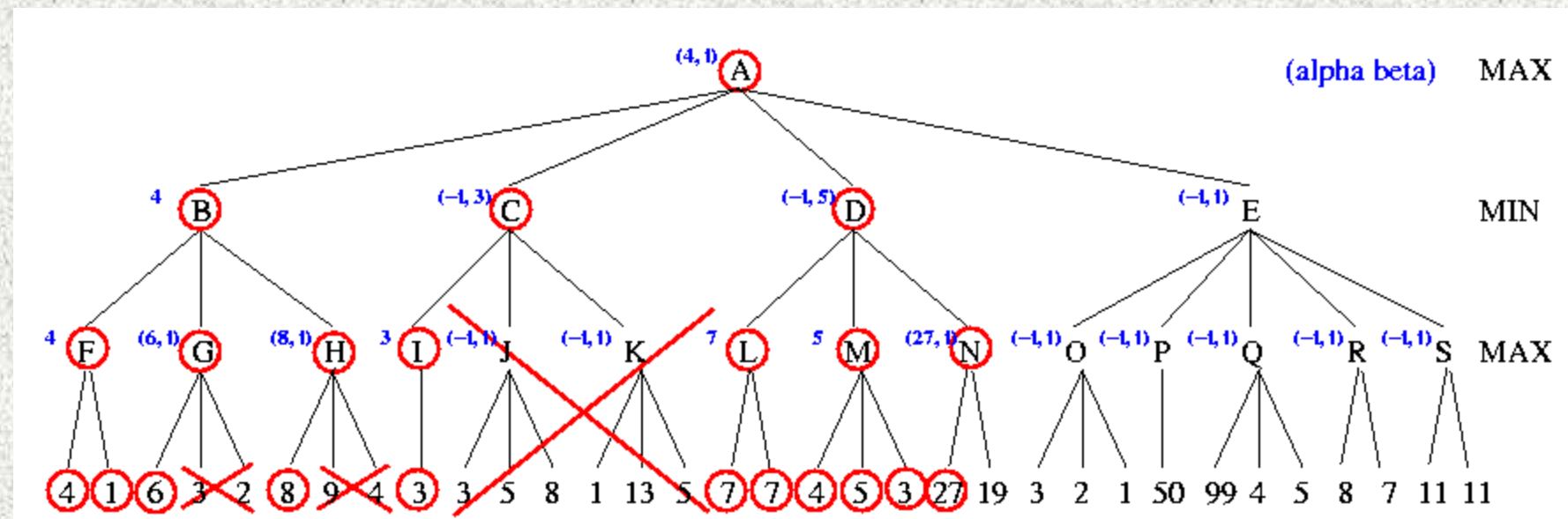
Example



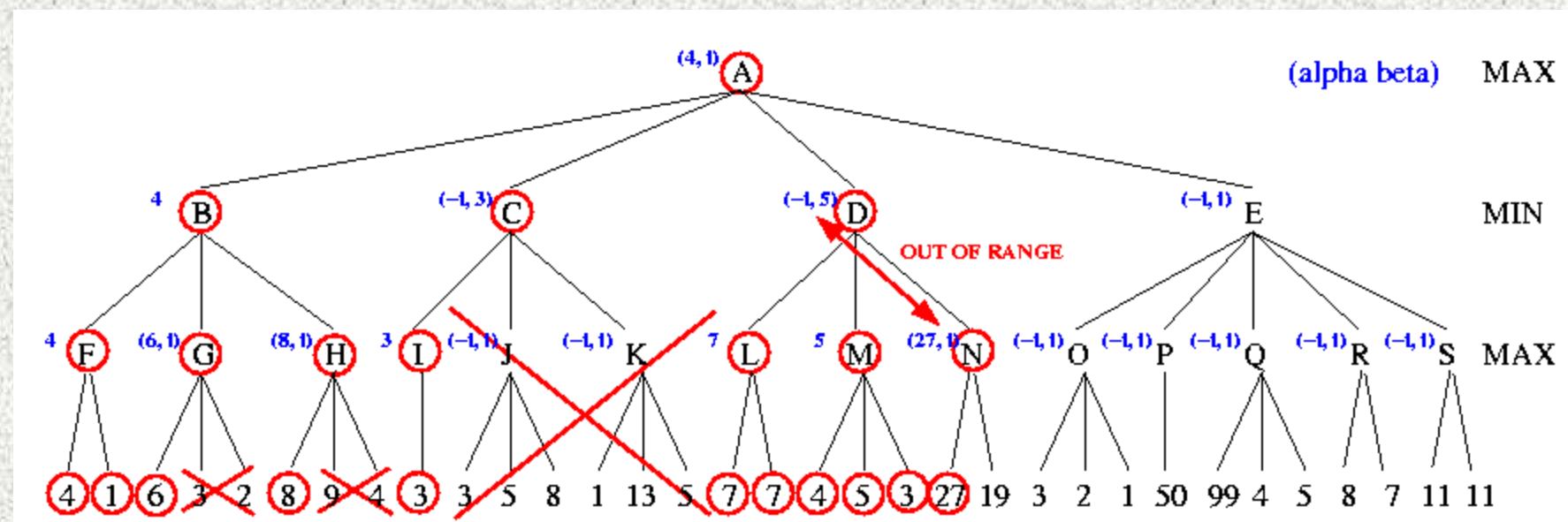
Example



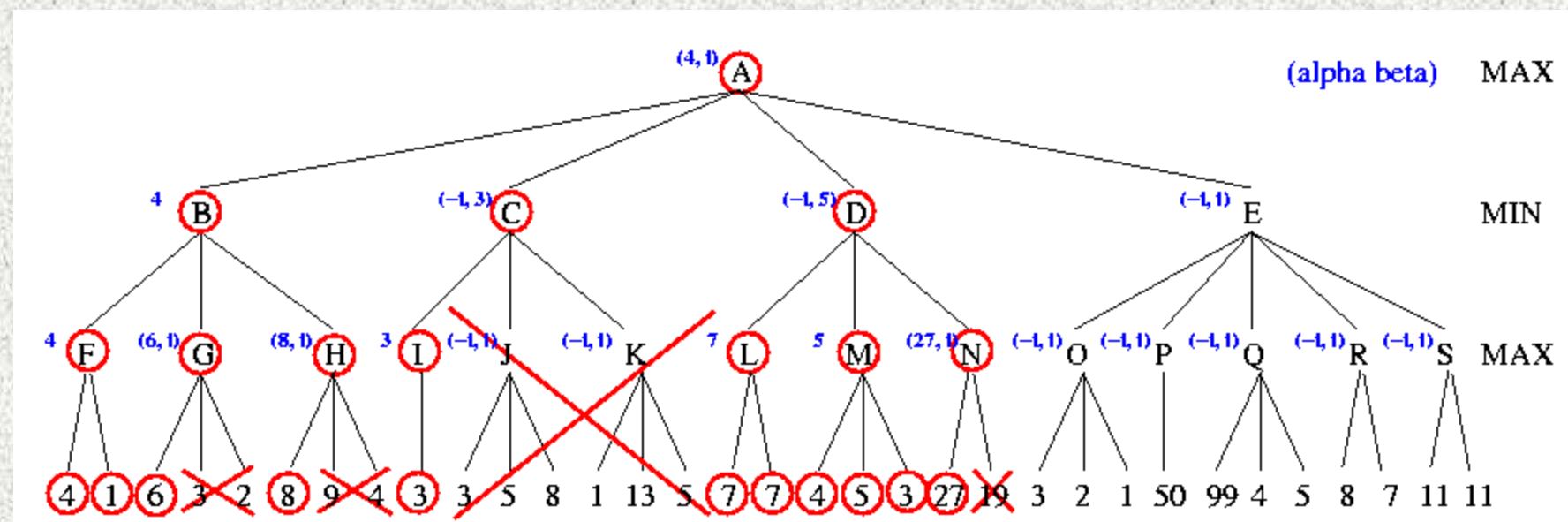
Example



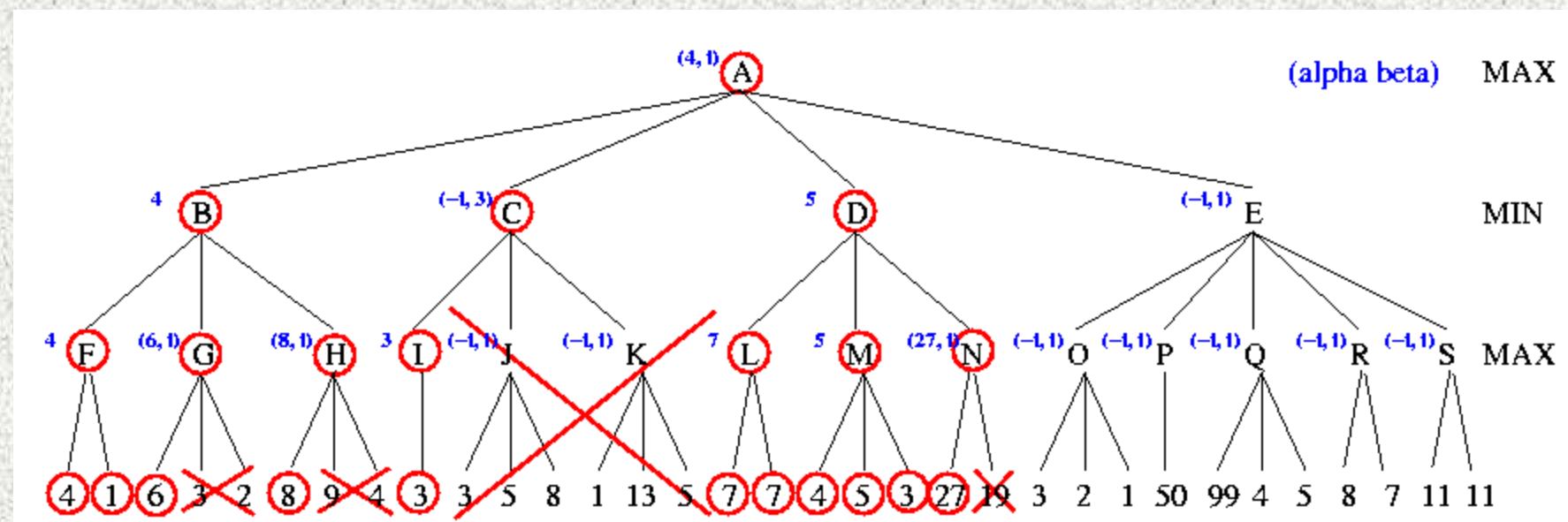
Example



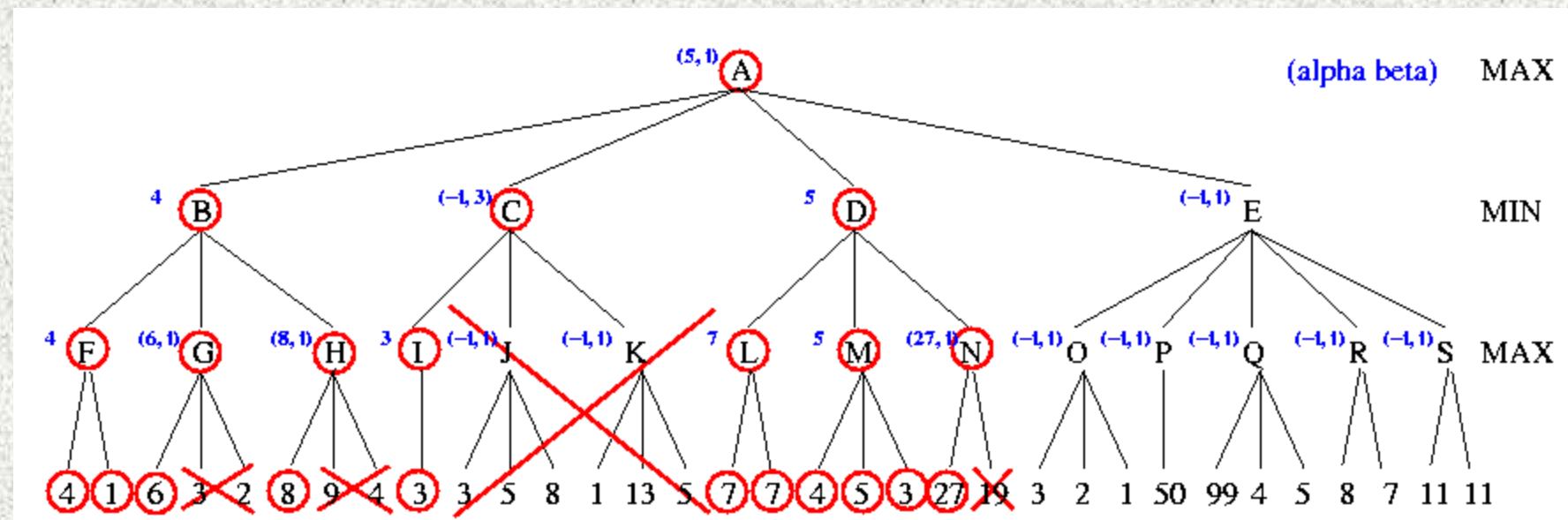
Example



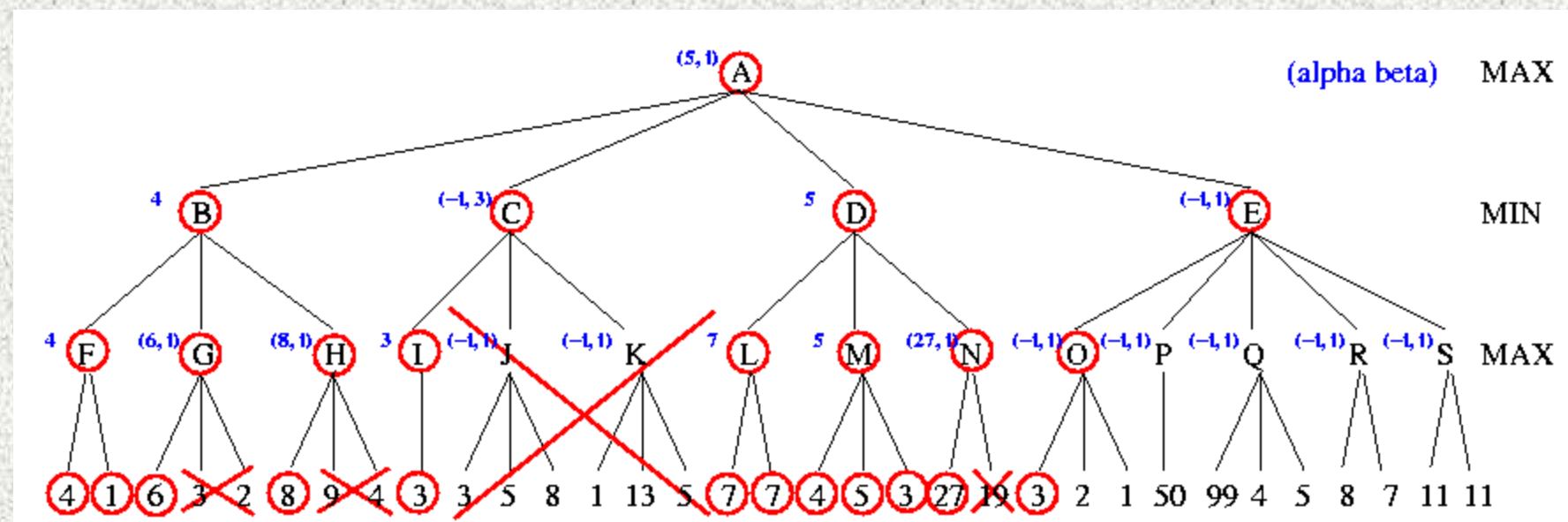
Example



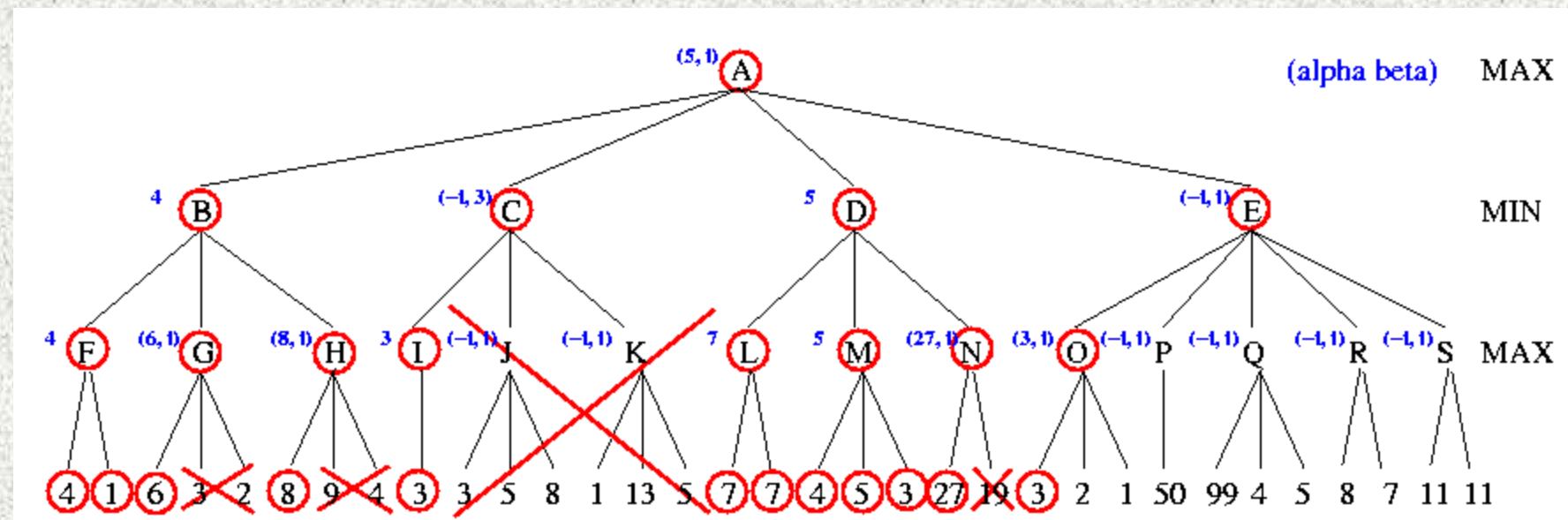
Example



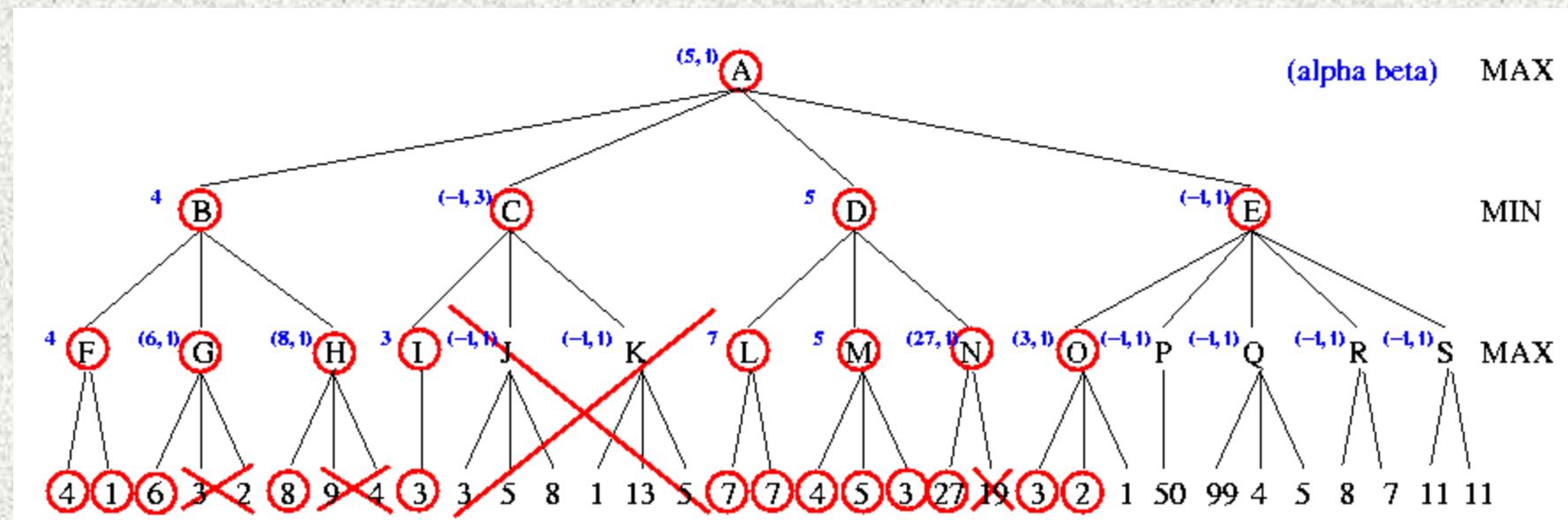
Example



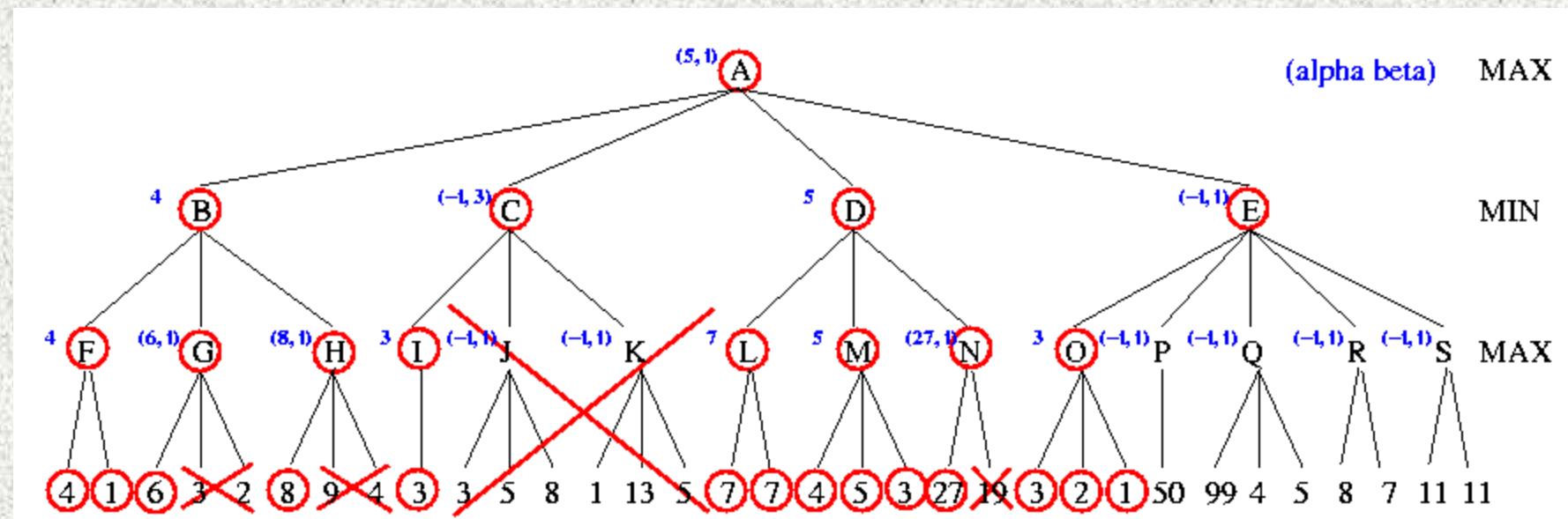
Example



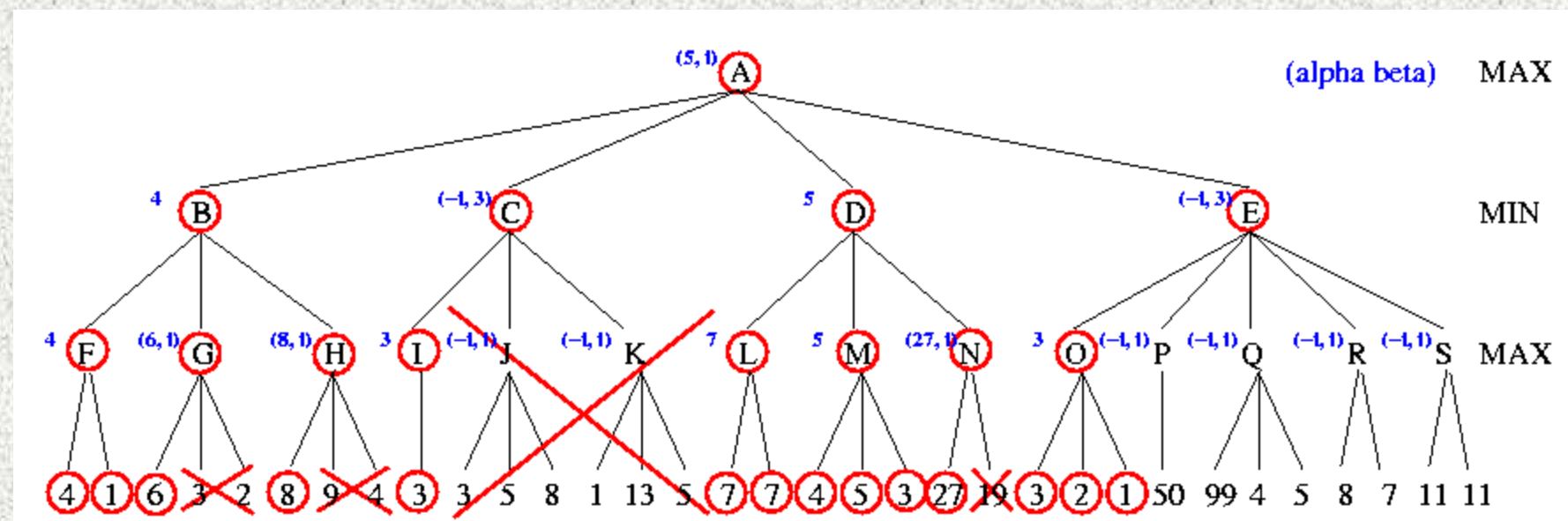
Example



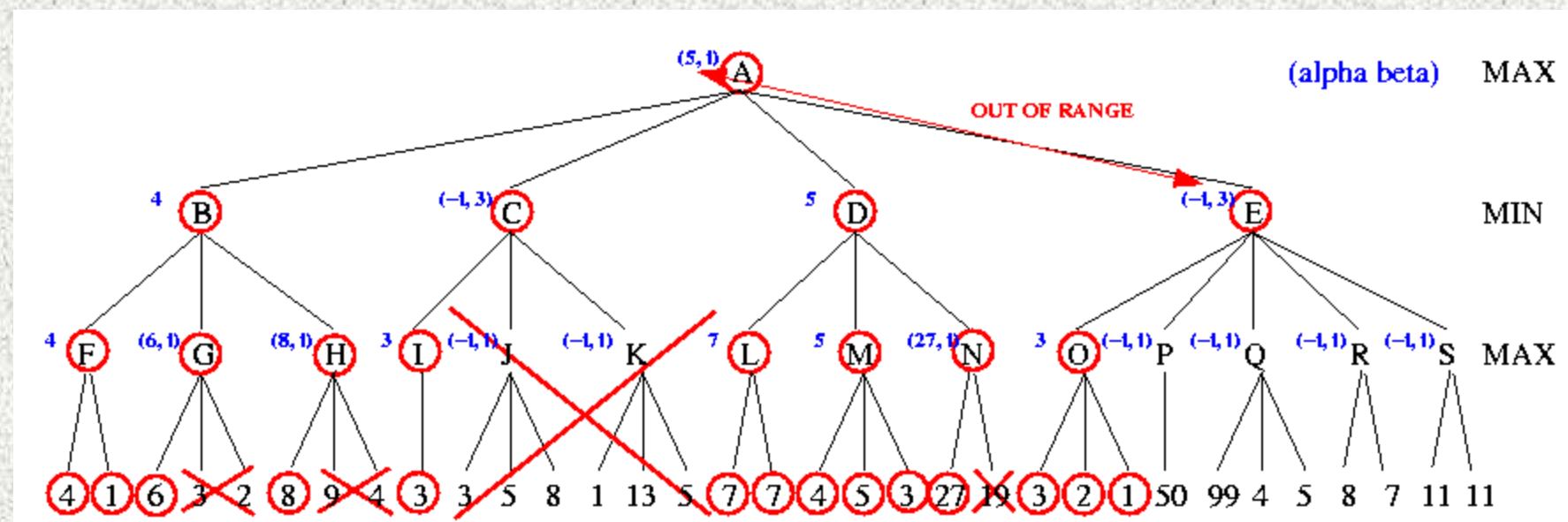
Example



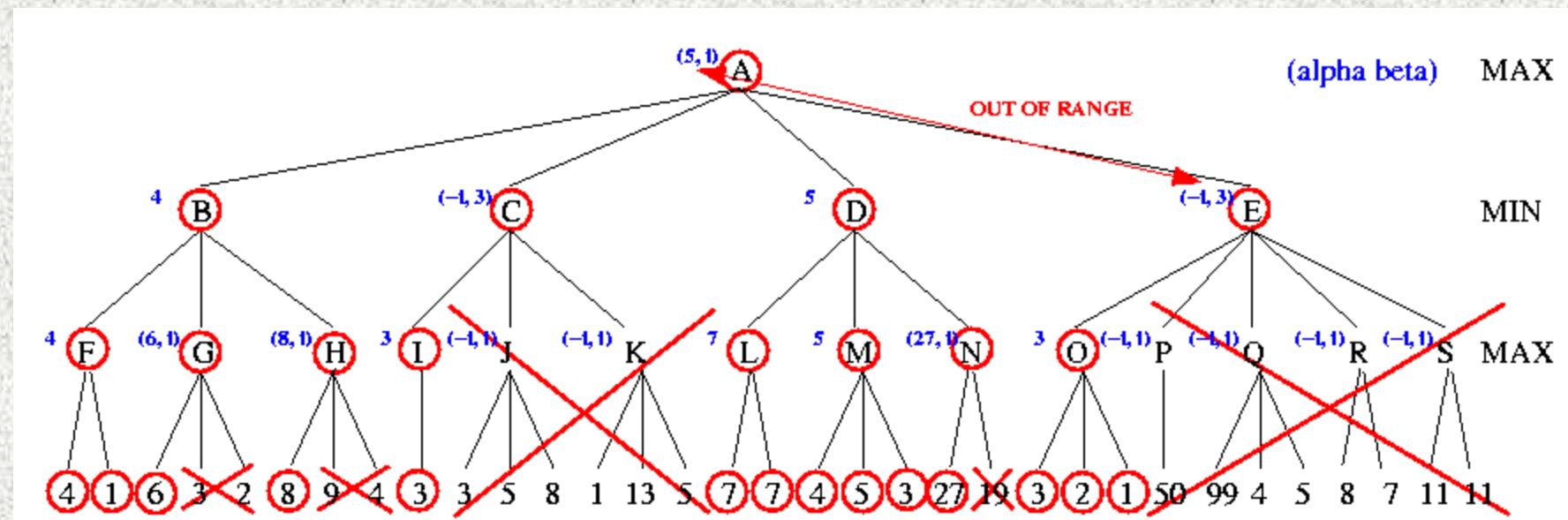
Example



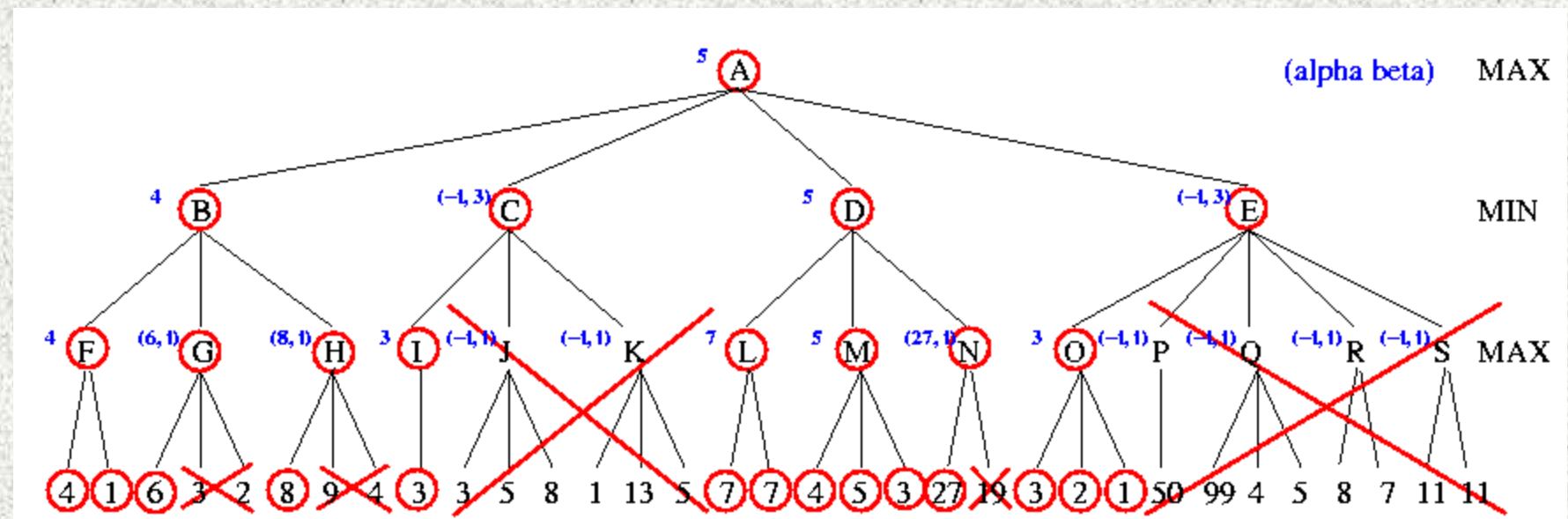
Example



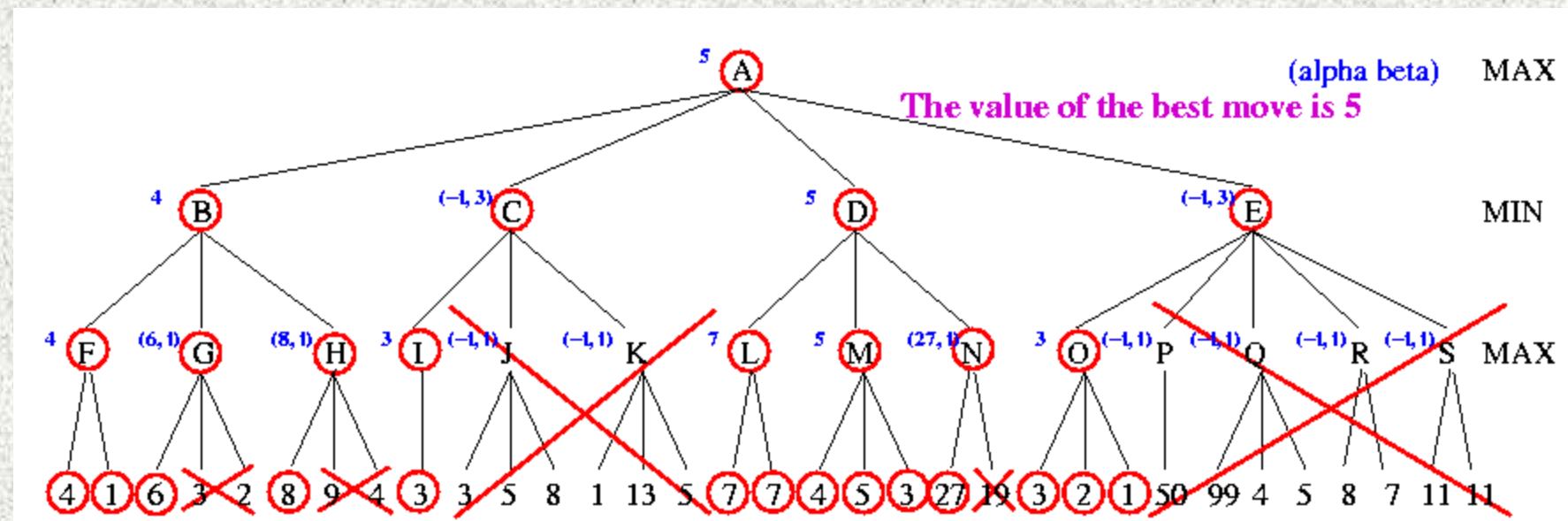
Example



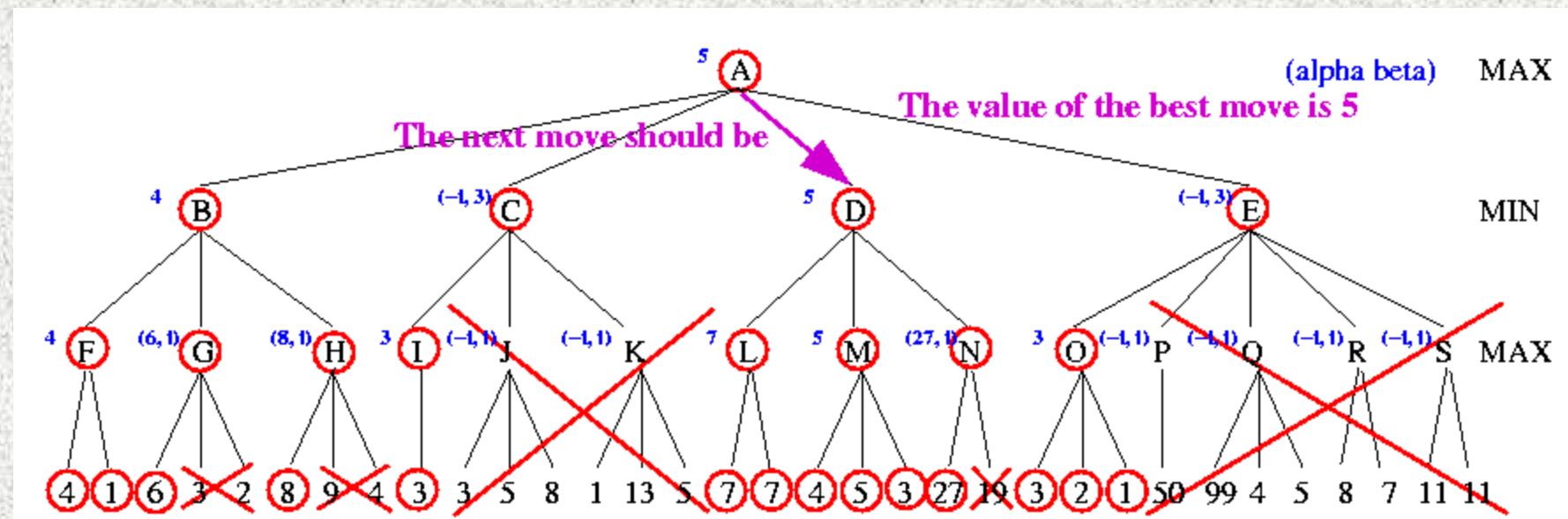
Example



Example

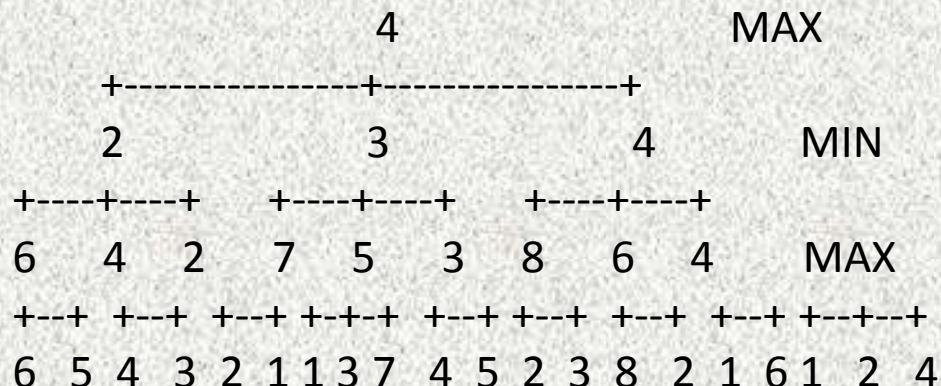


Example

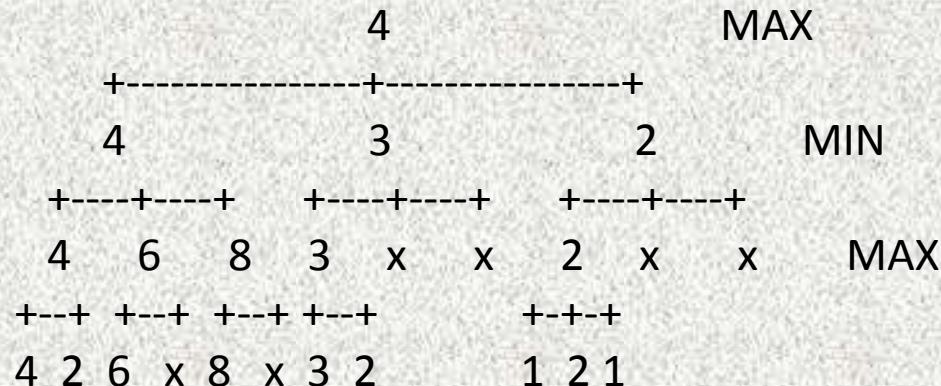


Bad and Good Cases for Alpha-Beta Pruning

- Bad: Worst moves encountered first



- Good: Good moves ordered first

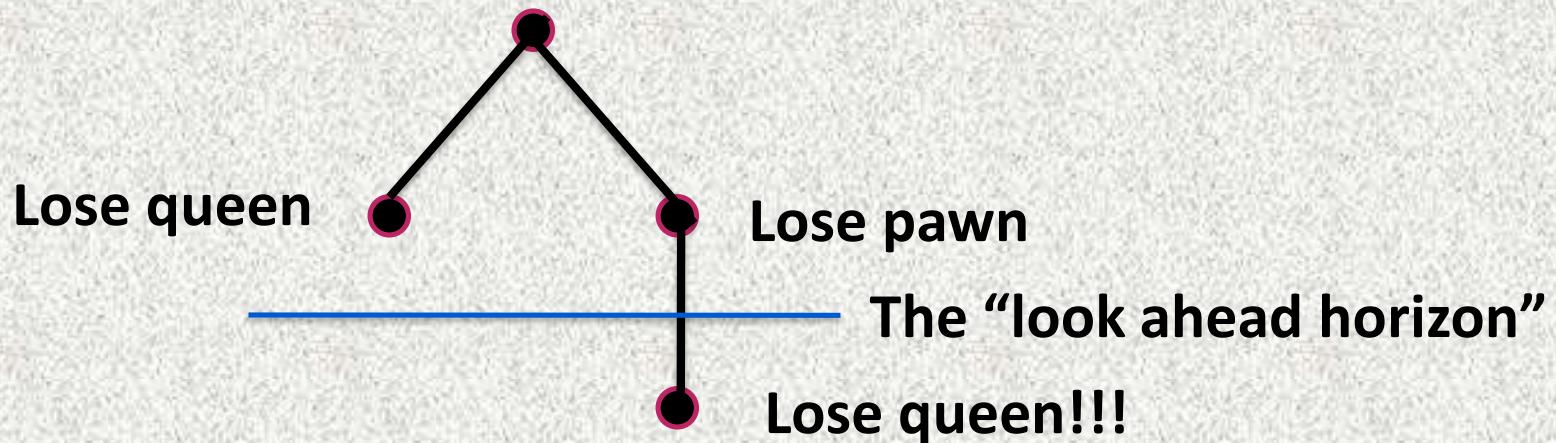


- If we can order moves, we can get more benefit from alpha-beta pruning

Alpha Beta Properties

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With perfect ordering, time complexity is $O(b^{m/2})$

Problems with a fixed ply: The Horizon Effect



- Inevitable losses are postponed
- Unachievable goals appear achievable
- Short-term gains mask unavoidable consequences (traps)

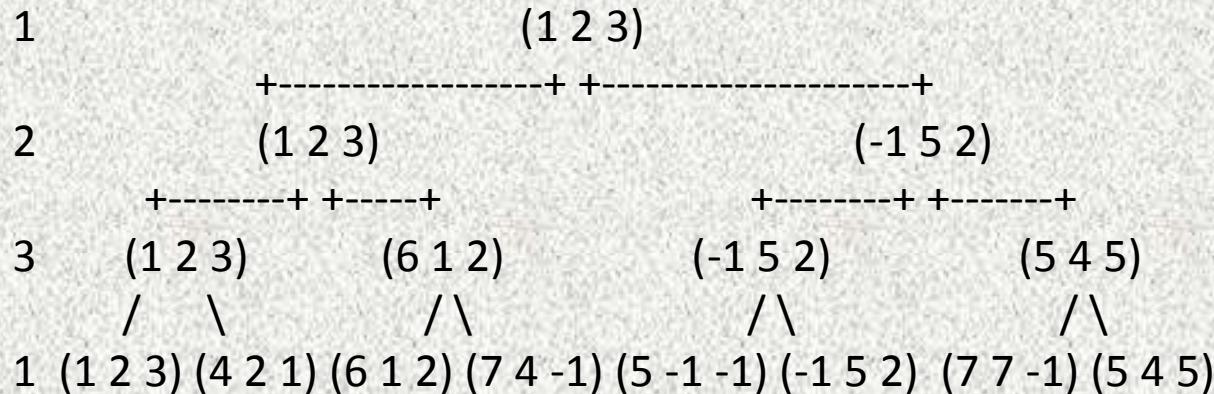
Solutions

- How to counter the horizon effect
 - Feedover
 - Do not cut off search at non-quiescent board positions (dynamic positions)
 - Example, king in danger
 - Keep searching down that path until reach quiescent (stable) nodes
 - Secondary Search
 - Search further down selected path to ensure this is the best move
 - Progressive Deepening
 - Search one ply, then two ply, etc., until run out of time
 - Similar to IDS

Variations on 2-Player Games

Multiplayer Games

to move



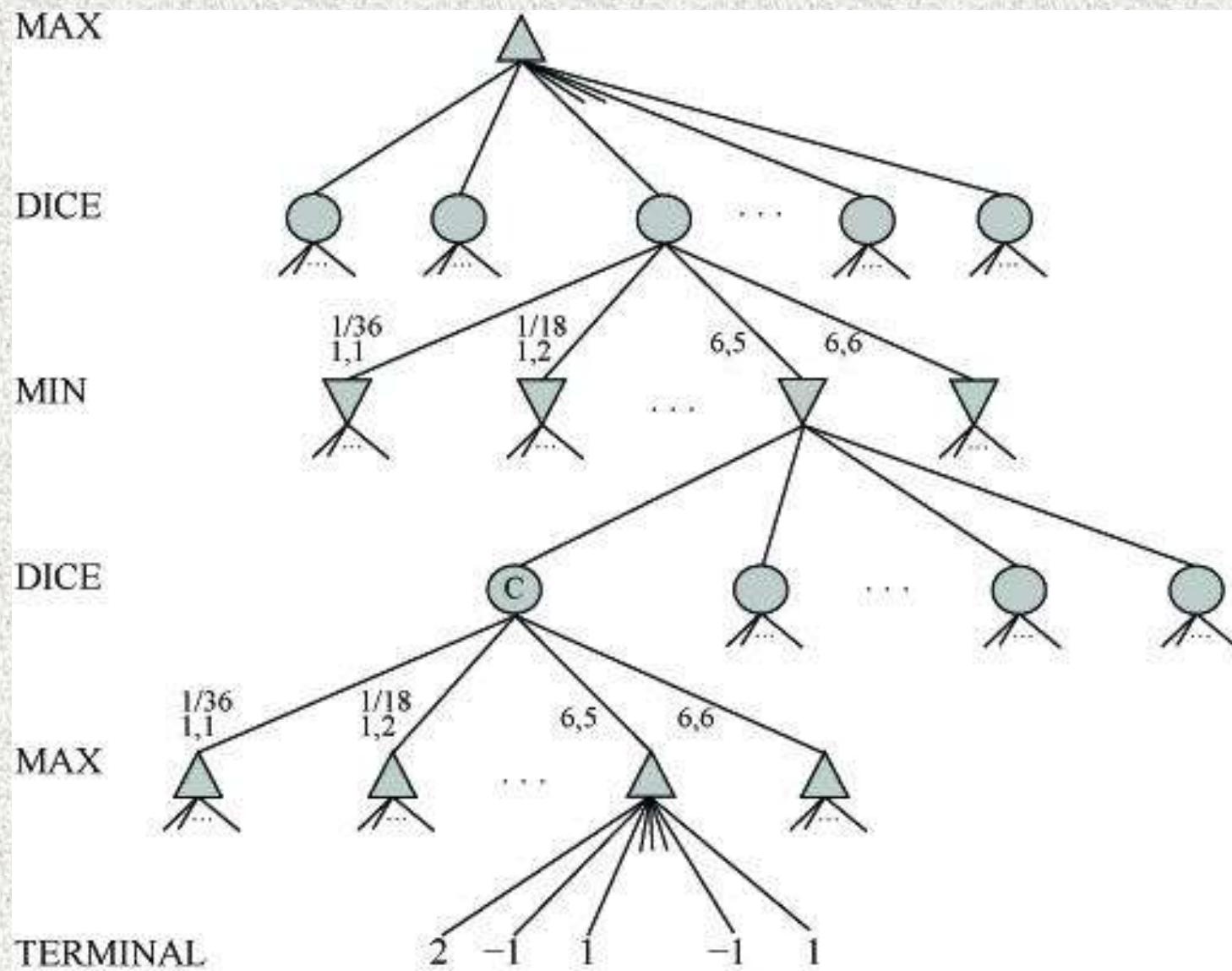
- Each player maximizes utility
- Each node stores a vector of utilities
- Entire vector is backed up the tree
- 3-player example: If in leftmost state, should player 3 choose first move because higher utility values?
- Result will be terminal state with utility values ($v_1=1, v_2=2, v_3=3$)
- This vector is backed up to the parent node
- Need to consider cooperation among players

Nondeterministic Games

- In backgammon, the dice rolls determine legal moves



Nondeterministic Games



Nondeterministic Game Algorithm

- Just like Minimax except also handle chance nodes
- Compute ExpectMinimaxValue of successors
 - If n is terminal node, then $\text{ExpectMinimaxValue}(n) = \text{Utility}(n)$
 - If n is a Max node, then
$$\text{ExpectMinimaxValue}(n) = \max_{s \in \text{Successors}(n)} \text{ExpectMinimaxValue}(s)$$
 - If n is a Min node, then
$$\text{ExpectMinimaxValue}(n) = \min_{s \in \text{Successors}(n)} \text{ExpectMinimaxValue}(s)$$
 - If n is a chance node, then
$$\text{ExpectMinimaxValue}(n) = \sum_{s \in \text{Successors}(n)} P(s) * \text{ExpectMinimaxValue}(s)$$

Status of AI Game Players

- Tic Tac Toe
 - Tied for best player in world
- Othello
 - [Computer](#) better than any human
 - Human champions now refuse to play computer
- Scrabble
 - Maven beat world champions Joel Sherman and Matt Graham
- Backgammon
 - 1992, [Tesauro](#) combines 3-ply search & neural networks (with 160 hidden units) yielding top-3 player
- Bridge
 - [Gib](#) ranked among top players in the world
- Poker
 - Pokie plays at strong intermediate level
- Checkers
 - 1994, [Chinook](#) ended 40-year reign of human champion Marion Tinsley
- Chess
 - 1997, [Deep Blue](#) beat human champion Gary Kasparov in six-game match
 - Deep Blue searches 200M positions/second, up to 40 ply
 - Now looking at other applications (molecular dynamics, drug synthesis)
- Go
 - 2008, MoGo running on 25 nodes (800 cores) beat Myungwan Kim
 - \$2M prize available for first computer program to defeat a top player

CptS 440 / 540

Artificial Intelligence

Knowledge Representation

Knowledge Representation



Knowledge Representation

- When we use search to solve a problem we must
 - Capture the knowledge needed to formalize the problem
 - Apply a search technique to solve problem
 - Execute the problem solution

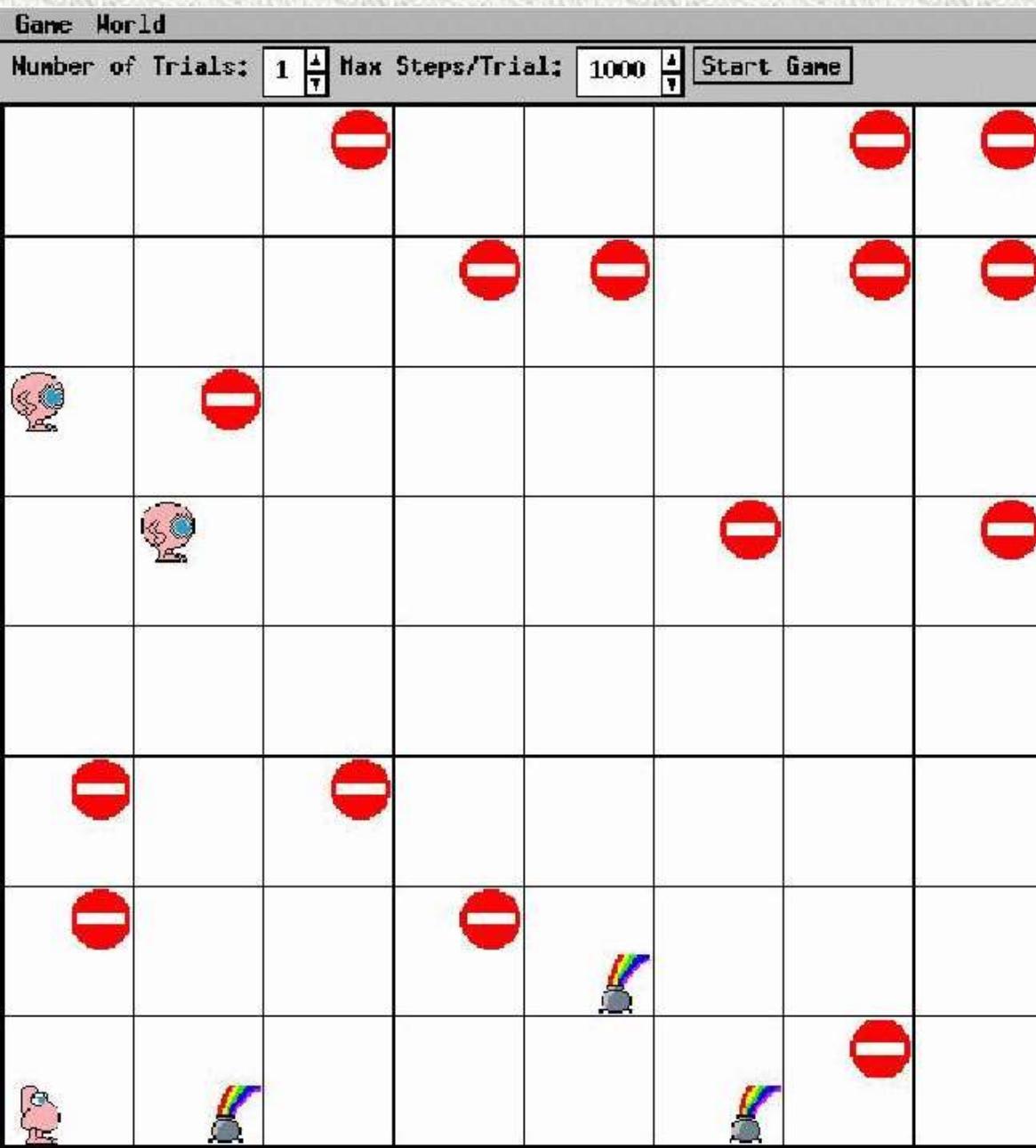
Role of KR

- The first step is the role of “knowledge representation” in AI.
- Formally,
 - The intended role of knowledge representation in artificial intelligence is to reduce problems of intelligent action to search problems.
- A good description, developed within the conventions of a good KR, is an open door to problem solving
- A bad description, using a bad representation, is a brick wall preventing problem solving

A Knowledge-Based Agent

- We previously talked about applications of search but not about methods of formalizing the problem.
- Now we look at extended capabilities to general logical reasoning.
- Here is one knowledge representation: logical expressions.
- A knowledge-based agent must be able to
 - Represent states, actions, etc.
 - Incorporate new percepts
 - Update internal representations of the world
 - Deduce hidden properties about the world
 - Deduce appropriate actions
- We will
 - Describe properties of languages to use for logical reasoning
 - Describe techniques for deducing new information from current information
 - Apply search to deduce (or learn) specifically needed information

The Wumpus World Environment



Player (Facing Down)	Player (Facing Up)
Wumpus (Facing Down)	Wumpus (Facing Up)
Player (Facing Left)	Player (Facing Right)
Wumpus (Facing Left)	Wumpus (Facing Right)
Player (Dead)	Pot of Gold
Wumpus (Dead)	Bottomless Pit

Percepts

Percepts & Status for AGENT: 0

PERCEP INFORMATION	
Stench:	false
Breeze:	true
Glitter:	false
Bump:	false
Scream:	false

NATURAL LANGUAGE HINT

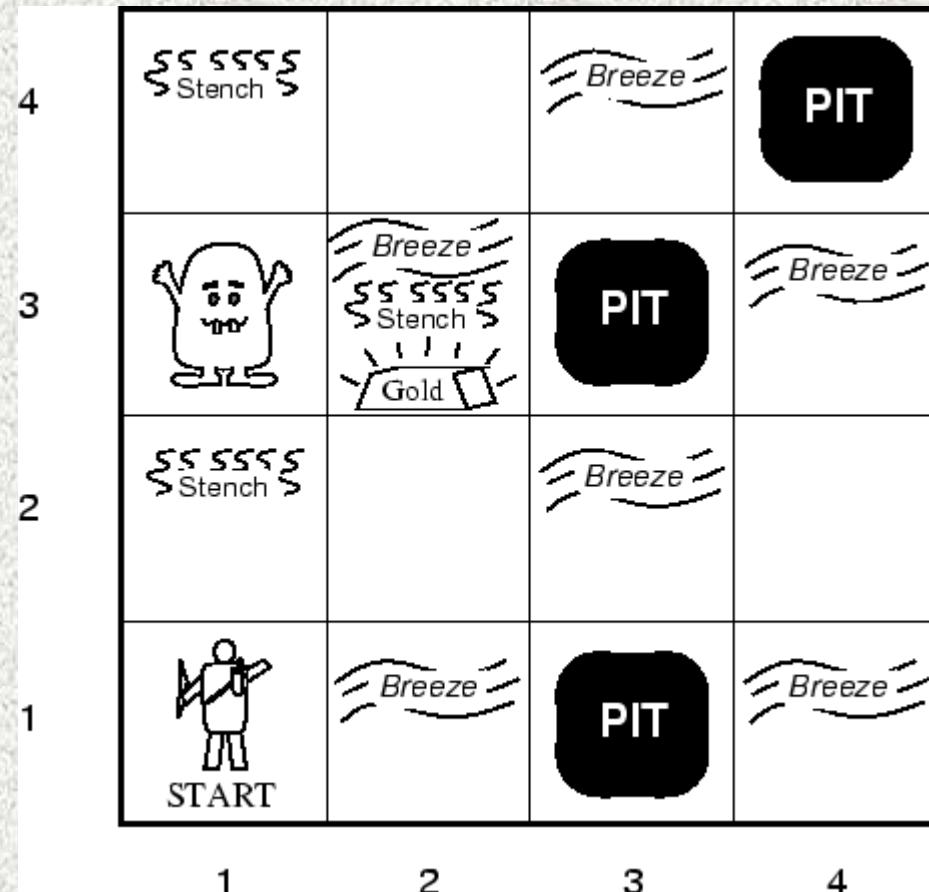
Hint:	a stench is in square (2, 1)
-------	------------------------------

STATUS INFORMATION

Last Action:	TURN_LEFT	Agent State:	1
Score:	-1	Arrows:	1
Gold Pieces:	0		

WW Agent Description

- Performance measure
 - gold +1000, death -1000
 - -1 per step, -10 for using arrow
- Environment
 - Squares adjacent to wumpus are smelly
 - Squares adjacent to pit are breezy
 - Glitter iff gold is in same square
 - Shooting kills wumpus if agent facing it
 - Shooting uses up only arrow
 - Grabbing picks up gold if in same square
 - Releasing drops gold in same square
- Actuators
 - Left turn, right turn, forward, grab, release, shoot
- Sensors
 - Breeze, glitter, smell, bump, scream



WW Environment Properties

- Observable?
 - Partial
- Deterministic?
 - Yes
- Episodic?
 - Sequential
- Static?
 - Yes (for now), wumpus and pits do not move
- Discrete?
 - Yes
- Single agent?
 - Multi (wumpus, eventually other agents)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2	3,2	4,2
1,1 (A) OK	2,1 OK	3,1	4,1

Percept: [None, None, None, None, None]

Deduce: Agent alive, so (1,1) OK
No breeze, so (1,2) and (2,1) OK

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2	3,2	4,2
1,1 (A) OK	2,1 OK	3,1	4,1

Percept: [None, None, None, None, None]

Deduce: Agent alive, so (1,1) OK
No breeze, so (1,2) and (2,1) OK

Action: Move East (turnright, goforward)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2 P?	3,2	4,2
1,1 OK	2,1 OK	3,1 P?	4,1

Percept: [None, Breeze, None, None, None]

Deduce: Pit in (2,2) or (3,1)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
1,2 OK	2,2 P?	3,2	4,2
1,1 OK	2,1 OK	3,1 P?	4,1

Percept: [None, Breeze, None, None, None]

Deduce: Pit in (2,2) or (3,1)

Action: Back to (1,1) then to (1,2)
(turnleft, turnleft, goforward,
turnright, goforward)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	2,4	2,5
W			
1,2 A OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, None, None, None, None]

Deduce: Wumpus in (1,3)
No pit in (2,2), pit in (3,1)

Sample Run

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
W			
1,2 (A) OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, None, None, None, None]

Deduce: Wumpus in (1,3)
No pit in (2,2), pit in (3,1)

Action: Move to (2,2) (turnright, goforward)
Ignore percept for now
Move to (2,3) (turnleft, goforward)

Sample Run

1,4	2,4 P?	3,4	4,4
1,3 W	2,3 OK Ⓐ	3,3 P?	4,3
1,2 OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, Breeze, Glitter, None, None]

Deduce: Pit in (2,4) or (3,3)
Gold in (2,3)

Action: Move to (2,2) (turnright, goforward)
Ignore percept for now
Move to (2,3) (turnleft, goforward)

Sample Run

1,4	2,4 P?	3,4	4,4
1,3 W	2,3 OK Ⓐ G	3,3 P?	4,3
1,2 OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, Breeze, Glitter, None, None]

Deduce: Pit in (2,4) or (3,3)
Gold in (2,3)

Action: Move to (1,1) through OK locations

Sample Run

1,4	2,4 P?	3,4	4,4
1,3 W	2,3 OK Ⓐ	3,3 P?	4,3
1,2 OK	2,2 OK	3,2	4,2
1,1 OK	2,1 OK	3,1 P	4,1

Percept: [Stench, Breeze, Glitter, None, None]

Deduce: Pit in (2,4) or (3,3)
Gold in (2,3)

Action: Move to (1,1) through OK locations

Now we look at

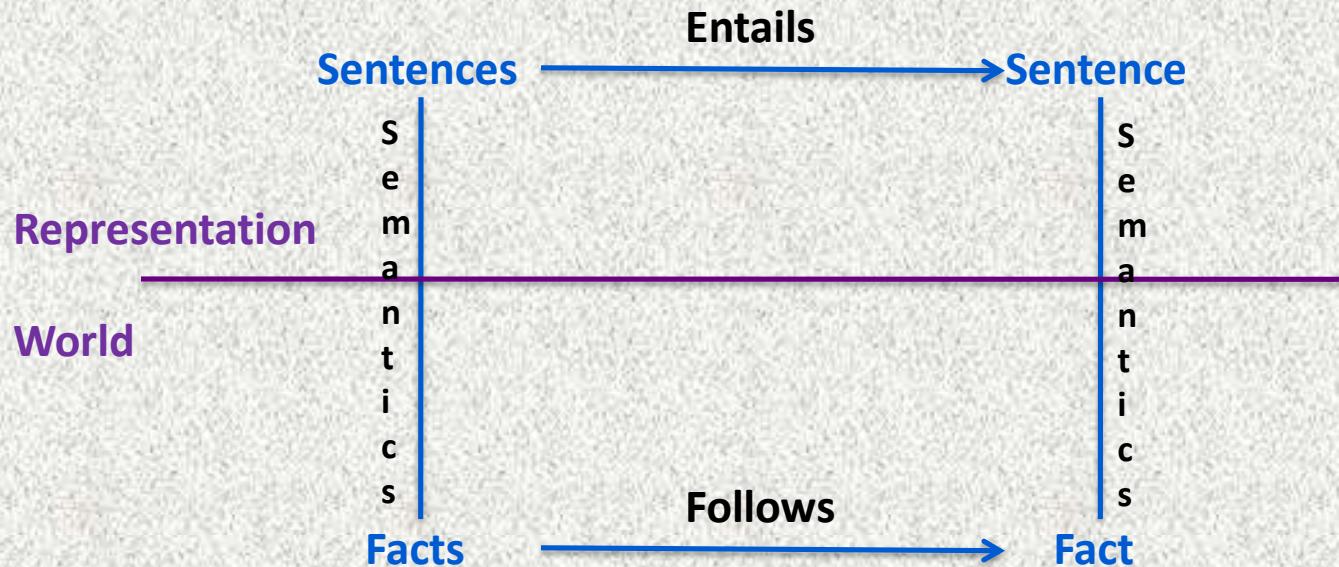
- How to represent facts / beliefs
 - ✓ “There is a pit in (2,2) or (3,1)”
- How to make inferences
 - ✓ “No breeze in (1,2), so pit in (3,1)”

Representation, Reasoning and Logic

- Sentence: Individual piece of knowledge
 - English sentence forms one piece of knowledge in English language
 - Statement in C forms one piece of knowledge in C programming language
- Syntax: Form used to represent sentences
 - Syntax of C indicates legal combinations of symbols
 - $a = 2 + 3;$ is legal
 - $a = + 2 3$ is not legal
 - Syntax alone does not indicate meaning
- Semantics: Mapping from sentences to facts in the world
 - They define the truth of a sentence in a “possible world”
 - Add the values of 2 and 3, store them in the memory location indicated by variable a
- In the language of arithmetic:
 - $x + 2 \geq y$ is a sentence
 - $x2 + y >$ is not a sentence
 - $x + 2 \geq y$ is true in all worlds where the number $x + 2$ is no less than the number y
 - $x + 2 \geq y$ is true in a world where $x = 7, y = 1$
 - $x + 2 \geq y$ is false in a world where $x = 0, y = 6$

Entailment

- There can exist a relationship between items in the language
 - Sentences “entail” sentences (representation level)
 - Facts “follow” from facts (real world)
- Entail / Follow mean the new item is true if the old items are true
- A collection of sentences, or knowledge base (KB), entail a sentence
 - KB |= sentence
 - KB entails the sentence iff the sentence is true in all worlds where the KB is true



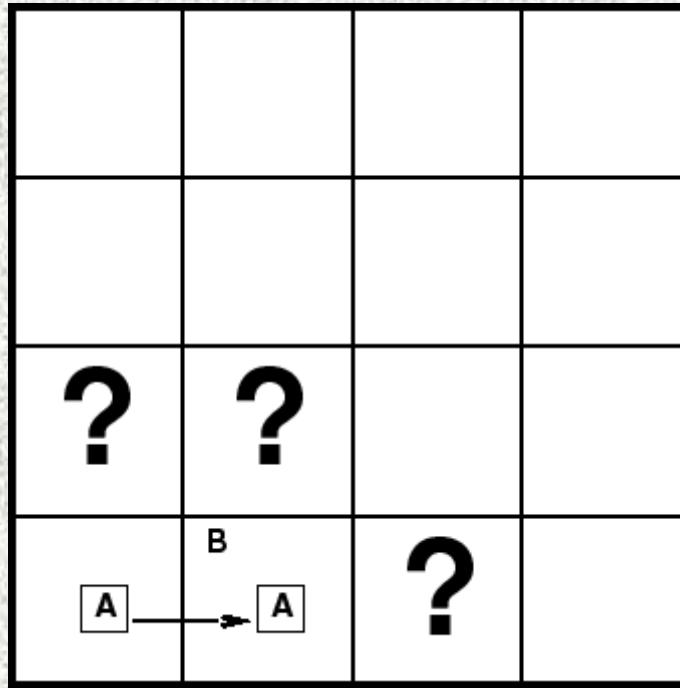
Entailment Examples

- KB
 - The Giants won
 - The Reds won
- Entails
 - Either the Giants won or the Reds won
- KB
 - To get a perfect score your program must be turned in today
 - I always get perfect scores
- Entails
 - I turned in my program today
- KB
 - CookLectures \rightarrow TodayIsTuesday \vee TodayIsThursday
 - \neg TodayIsThursday
 - TodayIsSaturday \rightarrow SleepLate
 - Rainy \rightarrow GrassIsWet
 - CookLectures \vee TodayIsSaturday
 - \neg SleepLate
- Which of these are correct entailments?
 - \neg SleepLate
 - GrassIsWet
 - \neg SleepLate \vee GrassIsWet
 - TodayIsTuesday
 - True

Models

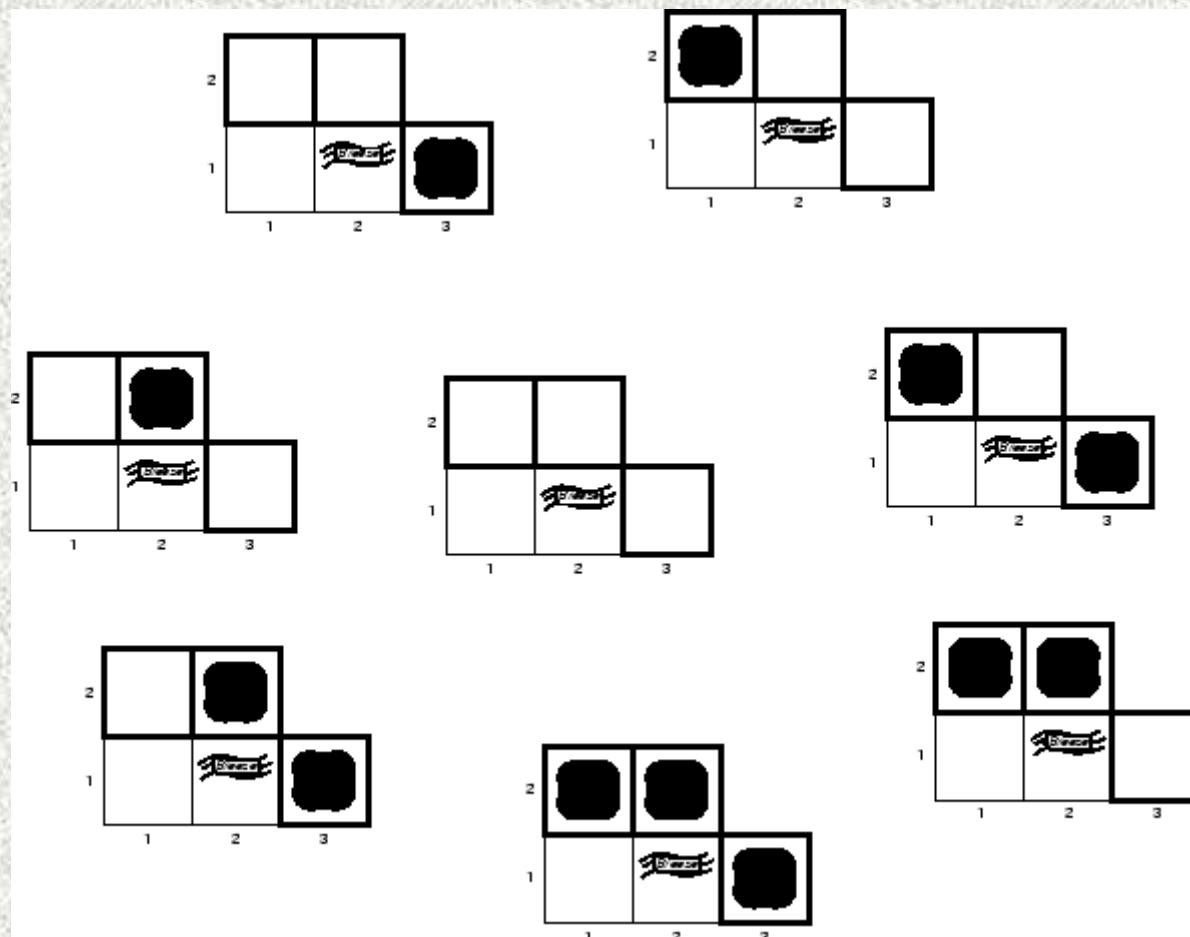
- Logicians frequently use **models**, which are formally structured worlds with respect to which truth can be evaluated. These are our “possible worlds”.
- M is a **model** of a sentence s if s is true in M .
- $M(s)$ is the set of all **models** of s .
- KB **entails** s ($KB \models s$) if and only if $M(KB)$ is a subset of $M(s)$
- For example, $KB = \text{Giants won and Reds won}$,
 $s = \text{Giants won}$

Entailment in the Wumpus World

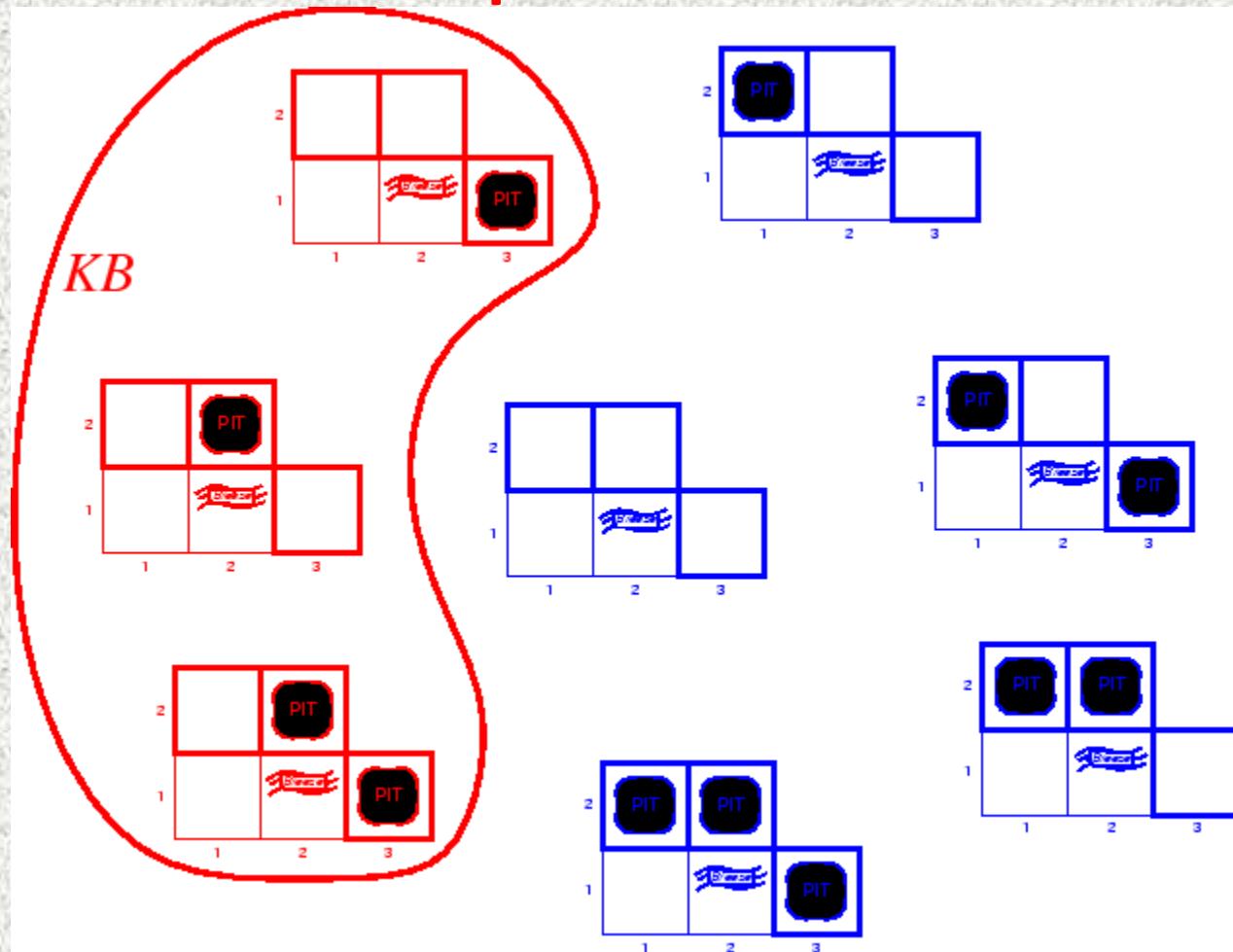


- Situation after detecting nothing in [1,1], moving right, breeze in [2,1]
- Consider possible models for the situation (the region around the visited squares) assuming only pits, no wumpi
- 3 Boolean choices, so there are 8 possible models

Wumpus Models

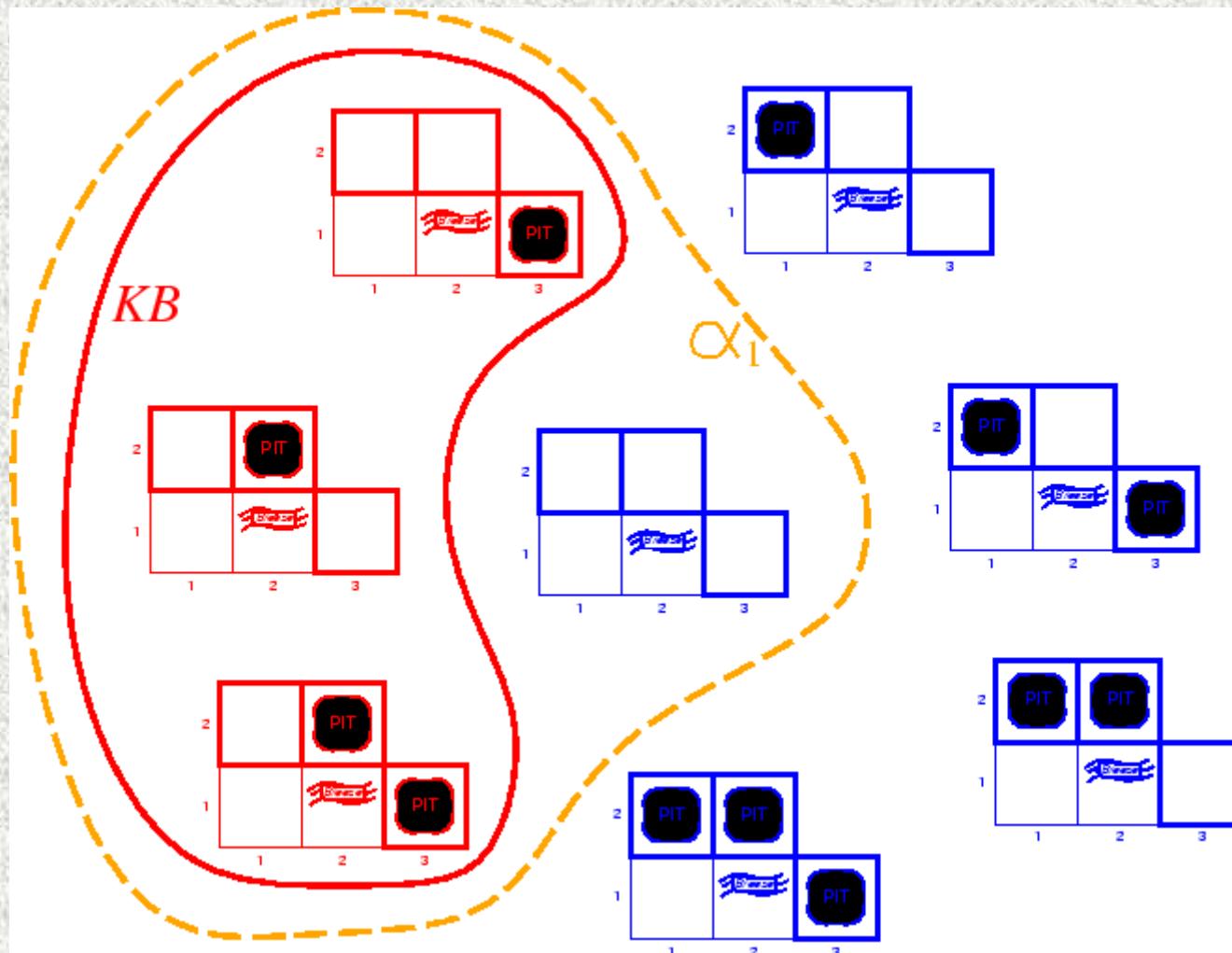


Wumpus Models



KB = wumpus world rules + observations

Wumpus Models

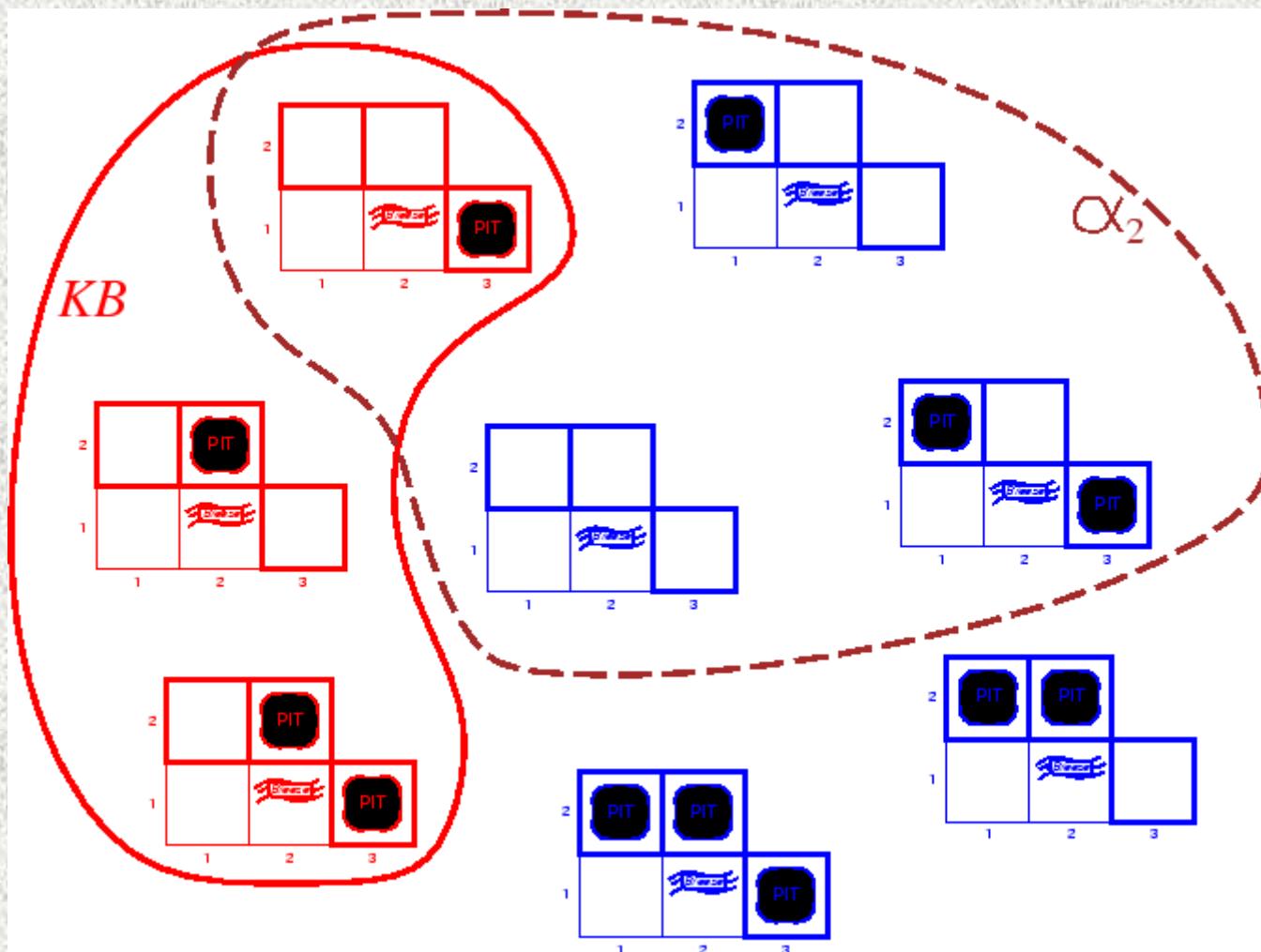


KB = wumpus world rules + observations

Possible conclusion: α_1 = "[1,2] is safe"

KB \models α_1 , proved by [model checking](#)

Wumpus Models



KB = wumpus world rules + observations

α_2 = “[2,2] is safe”, $\text{KB} \neq \alpha_2$

Inference

- Use two different ways:
 - Generate new sentences that are entailed by KB
 - Determine whether or not sentence is entailed by KB
- A **sound** inference procedure generates only entailed sentences
- Modus ponens is sound
$$\frac{A, A \rightarrow B}{B}$$
- Abduction is not sound
$$\frac{B, A \rightarrow B}{A}$$
- Logic gone bad

Definitions

- A **complete** inference procedure can generate all entailed sentences from the knowledge base.
- The meaning of a sentence is a mapping onto the world (a model).
- This mapping is an **interpretation** (interpretation of Lisp code).
- A sentence is **valid** (necessarily true, tautology) iff true under all possible interpretations.
 - $A \vee \neg A$
- A could be:
 - Stench at [1,1]
 - Today is Monday
 - $2+3=5$
- These statements are not valid.
 - $A \wedge \neg A$
 - $A \vee B$
- The last statement is **satisfiable**, meaning there exists at least one interpretation that makes the statement true. The previous statement is **unsatisfiable**.

Logics

- Logics are formal languages for representing information such that conclusions can be drawn
- Logics are characterized by their “primitives” commitments
 - Ontological commitment: What exists? Facts? Objects? Time? Beliefs?
 - Epistemological commitment: What are the states of knowledge?

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	value in [0, 1]
Fuzzy logic	degree of truth	known interval value

Examples

- Propositional logic
 - Simple logic
 - Symbols represent entire facts
 - Boolean connectives ($\&$, \vee , \rightarrow , \Leftrightarrow , \sim)
 - Propositions (symbols, facts) are either TRUE or FALSE
- First-order logic
 - Extend propositional logic to include variables, quantifiers, functions, objects

Propositional Logic

- Proposition symbols P, Q, etc., are sentences
- The true/false value of propositions and combinations of propositions can be calculated using a truth table
- If P and S are sentences, then so are $\neg P$, $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, $P \leftrightarrow Q$
- An interpretation I consists of an assignment of truth values to all proposition symbols I(S)
 - An interpretation is a logician's word for what is often called a “possible world”
 - Given 3 proposition symbols P, Q, and R, there are 8 interpretations
 - Given n proposition symbols, there are 2^n interpretations
- To determine the truth of a complex statement for I, we can
 - Substitute I's truth value for every symbol
 - Use truth tables to reduce the statement to a single truth value
 - End result is a single truth value, either True or False

Propositional Logic

- For propositional logic, a row in the truth table is one interpretation
- A logic is **monotonic** as long as entailed sentences are preserved as more knowledge is added

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>True</i>
<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>
<i>True</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>	<i>False</i>
<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>	<i>True</i>

Rules of Inference for Propositional Logic

- Modus ponens $\frac{A, A \rightarrow B}{B}$

All men are mortal (Man \rightarrow Mortal)

Socrates is a man (Man)

Socrates is mortal (Mortal)

- And introduction $\frac{A, B}{A \wedge B}$

- Or introduction $\frac{A}{AvBvCvDv\dots}$

- And elimination $\frac{A \wedge B \wedge C \wedge \dots \wedge Z}{A}$

- Double-negation elimination $\frac{\neg\neg A}{A}$

- Unit resolution $\frac{AvB, \neg B}{A}$

Today is Tuesday or Thursday

Today is not Thursday

Today is Tuesday

- Resolution

$$\frac{AvB, \neg B \vee C \quad \neg A \rightarrow B, B \neg \rightarrow C}{AvC} \quad \neg A \neg \rightarrow C$$

Today is Tuesday or Thursday

Today is not Thursday or tomorrow is Friday

Today is Tuesday or tomorrow is Friday

Normal Forms

- Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms
- Conjunctive Normal Form (**CNF**)
conjunction of disjunctions of literals (conjunction of clauses)
For example, $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- Disjunctive Normal Form (**DNF**)
disjunction of conjunctions of literals (disjunction of terms)
For example, $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)$
- **Horn Form (restricted)**
conjunction of Horn clauses (clauses with ≤ 1 positive literal)
For example, $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
Often written as a set of implications:
 $B \rightarrow A$ and $(C \wedge D) \rightarrow B$

Proof methods

- Model checking
 - Truth table enumeration (sound and complete for propositional logic)
 - Show that all interpretations in which the left hand side of the rule is true, the right hand side is also true
 - Application of inference rules
 - Sound generation of new sentences from old
Proof = a sequence of inference rule applications
Can use inference rules as operators in a standard search algorithm

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>
<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>

Wumpus World KB

- Vocabulary
 - Let $P_{i,j}$ be true if there is a pit in $[i,j]$
 - Let $B_{i,j}$ be true if there is a breeze in $[i,j]$
- Sentences
 - $\neg P_{1,1}$
 - $\neg B_{1,1}$
 - $B_{2,1}$
- “Pits cause breezes in adjacent squares”
 - $B_{1,1} \leftrightarrow P_{1,2} \vee P_{2,1}$
 - $B_{2,1} \leftrightarrow P_{1,1} \vee P_{2,2} \vee P_{3,1}$

An Agent for the Wumpus World

- Imagine we are at a stage in the game where we have had some experience
 - What is in our knowledge base?
 - What can we deduce about the world?
- Example: Finding the wumpus
- If we are in [1,1] and know
 - $\neg S_{11}$
 - S_{12}
 - S_{21}
 - $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$
 - $S_{12} \rightarrow W_{11} \vee W_{12} \vee W_{13} \vee W_{22}$
 - $S_{21} \rightarrow W_{11} \vee W_{21} \vee W_{31} \vee W_{22}$
- What can we conclude?

1,4	2,4	3,4	4,4
1,3 W	2,3	3,3	4,3
1,2 A S OK	2,2	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus

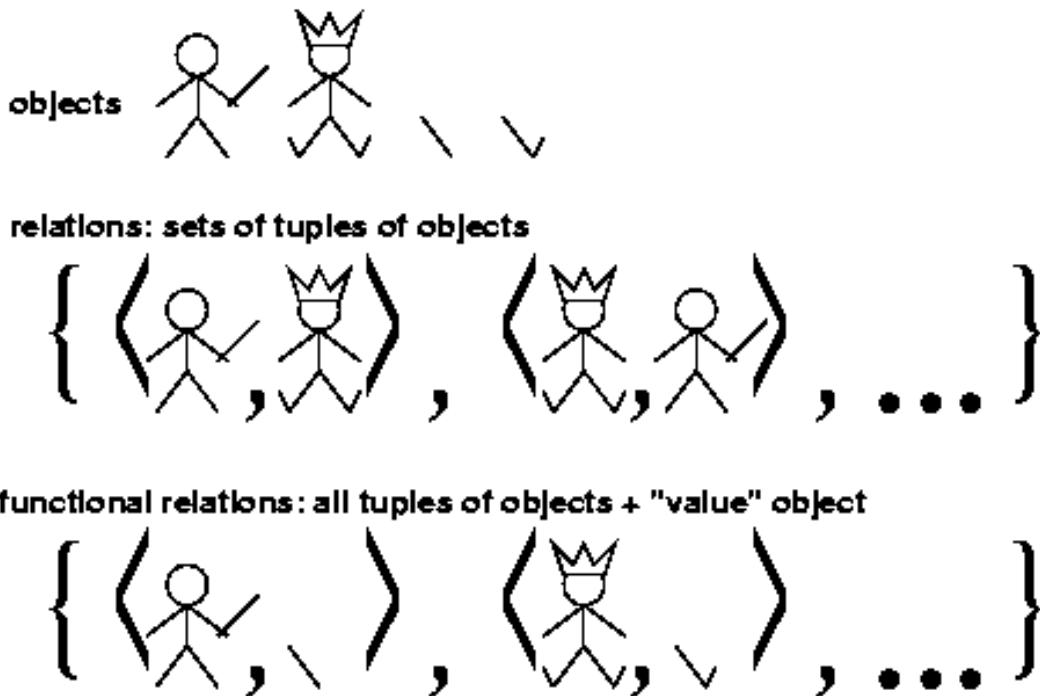
Limitations of Propositional Logic

- Propositional logic cannot express general-purpose knowledge succinctly
- We need 32 sentences to describe the relationship between wumpi and stenches
- We would need another 32 sentences for pits and breezes
- We would need at least 64 sentences to describe the effects of actions
- How would we express the fact that there is only one wumpus?
- Difficult to identify specific individuals (Mary, among 3)
- Generalizations, patterns, regularities difficult to represent (all triangles have 3 sides)

First-Order Predicate Calculus

- Propositional Logic uses only propositions (symbols representing facts), only possible values are True and False
- First-Order Logic includes:
 - Objects: peoples, numbers, places, ideas (atoms)
 - Relations: relationships between objects (predicates, T/F value)
 - Example: father(fred, mary)
 - Properties: properties of atoms (predicates, T/F value)
Example: red(ball)
 - Functions: father-of(mary), next(3), (any value in range)
 - Constant: function with no parameters, MARY

FOPC Models



Example

- Express “Socrates is a man” in
 - Propositional logic
 - MANSOCRATES - single proposition representing entire idea
 - First-Order Predicate Calculus
 - Man(SOCRATES) - predicate representing property of constant SOCRATES

FOPC Syntax

- **Constant** symbols (Capitalized, Functions with no arguments)
Interpretation must map to exactly one object in the world
- **Predicates** (can take arguments, True/False)
Interpretation maps to relationship or property T/F value
- **Function** (can take arguments)
Maps to exactly one object in the world

Definitions

- **Term**
Anything that identifies an object
Function(args)
Constant - function with 0 args
- **Atomic sentence**
Predicate with term arguments
Enemies(WilyCoyote, RoadRunner)
Married(FatherOf(Alex), MotherOf(Alex))
- **Literals**
atomic sentences and negated atomic sentences
- **Connectives**
 $(\&)$, (\vee) , (\rightarrow) , (\Leftrightarrow) , (\sim)
if connected by \wedge , conjunction (components are conjuncts)
if connected by \vee , disjunction (components are disjuncts)
- **Quantifiers**
Universal Quantifier \forall
Existential Quantifier \exists

Universal Quantifiers

- How do we express “All unicorns speak English” in Propositional Logic?
- We would need to specify a proposition for each unicorn
- \forall is used to express facts and relationships that we know to be true for all members of a group (objects in the world)
- A variable is used in the place of an object
 $\forall x \text{ Unicorn}(x) \rightarrow \text{SpeakEnglish}(x)$
The **domain** of x is the world
The **scope** of x is the statement following \forall (sometimes in [])
- Same as specifying
 - $\text{Unicorn(Uni1)} \rightarrow \text{SpeakEnglish(Uni1)} \ \&$
 - $\text{Unicorn(Uni2)} \rightarrow \text{SpeakEnglish(Uni2)} \ \&$
 - $\text{Unicorn(Uni3)} \rightarrow \text{SpeakEnglish(Uni3)} \ \&$
 - ...
 - $\text{Unicorn(Table1)} \rightarrow \text{Table(Table1)} \ \&$
 - ...
- One statement for each object in the world
- We will leave variables lower case (sometimes $?x$)
Notice that x ranges over all objects, not just unicorns.
- A term with no variables is a **ground term**

Existential Quantifier

- This makes a statement about some object (not named)
- $\exists x [Bunny(x) \wedge EatsCarrots(x)]$
- This means there exists some object in the world (at least one) for which the statement is true. Same as disjunction over all objects in the world.
 - $(Bunny(Bun1) \wedge EatsCarrots(Bun1)) \vee$
 - $(Bunny(Bun2) \wedge EatsCarrots(Bun2)) \vee$
 - $(Bunny(Bun3) \wedge EatsCarrots(Bun3)) \vee$
 - ...
 - $(Bunny(Table1) \wedge EatsCarrots(Table1)) \vee$
 - ...
- What about $\exists x Unicorn(x) \rightarrow SpeakEnglish(x)$?
- Means implication applies to at least one object in the universe

DeMorgan Rules

- $\forall x \neg P \leftrightarrow \neg \exists x P$
- $\forall x P \leftrightarrow \neg \exists x \neg P$
- $\neg \forall x \neg P \leftrightarrow \exists x P$
- $\neg \forall x P \leftrightarrow \exists x \neg P$
- Example:

$$\forall x LovesWatermelon(x) \leftrightarrow \neg \exists x \neg LovesWatermelon(x)$$

Other Properties

- $(X \rightarrow Y) \leftrightarrow \neg X \vee Y$
 - Can prove with truth table
- Not true:
 - $(X \rightarrow Y) \leftrightarrow (Y \rightarrow X)$
 - This is a type of inference that is not sound (abduction)

Examples

- All men are mortal

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous
 - $\forall x [(Purple(x) \wedge Mushroom(x)) \rightarrow Poisonous(x)]$

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous
 - $\forall x [(Purple(x) \wedge Mushroom(x)) \rightarrow Poisonous(x)]$
- A mushroom is poisonous only if it is purple

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous
 - $\forall x [(Purple(x) \wedge Mushroom(x)) \rightarrow Poisonous(x)]$
- A mushroom is poisonous only if it is purple

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous
 - $\forall x [(Purple(x) \wedge Mushroom(x)) \rightarrow Poisonous(x)]$
- A mushroom is poisonous only if it is purple
 - $\forall x [(Mushroom(x) \wedge Poisonous(x)) \rightarrow Purple(x)]$

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous
 - $\forall x [(Purple(x) \wedge Mushroom(x)) \rightarrow Poisonous(x)]$
- A mushroom is poisonous only if it is purple
 - $\forall x [(Mushroom(x) \wedge Poisonous(x)) \rightarrow Purple(x)]$
- No purple mushroom is poisonous

Examples

- All men are mortal
 - $\forall x [Man(x) \rightarrow Mortal(x)]$
- Socrates is a man
 - $Man(Socrates)$
- Socrates is mortal
 - $Mortal(Socrates)$
- All purple mushrooms are poisonous
 - $\forall x [(Purple(x) \wedge Mushroom(x)) \rightarrow Poisonous(x)]$
- A mushroom is poisonous only if it is purple
 - $\forall x [(Mushroom(x) \wedge Poisonous(x)) \rightarrow Purple(x)]$
- No purple mushroom is poisonous
 - $\neg (\exists x [Purple(x) \wedge Mushroom(x) \wedge Poisonous(x)])$

Examples

- There is exactly one mushroom

Examples

- There is exactly one mushroom

$$\exists x \text{Mushroom}(x) \wedge (\forall y (NEQ(x, y) \rightarrow \neg \text{Mushroom}(y)))]$$

– Because “exactly one” is difficult to express we can use $\exists!$ To denote exactly one of a type of object

- Every city has a dog catcher who has been bitten by every dog in town

Examples

- There is exactly one mushroom

$$\exists x \text{Mushroom}(x) \wedge (\forall y (NEQ(x, y) \rightarrow \neg \text{Mushroom}(y)))]$$

– Because “exactly one” is difficult to express we can use $\exists!$ To denote exactly one of a type of object

- Every city has a dog catcher who has been bitten by every dog in town

– Use City(c), DogCatcher(c), Bit(d,x), Lives(x,c)

$$\forall a, b [\text{City}(a) \rightarrow \exists c \text{DogCatcher}(c) \wedge (\text{Dog}(b) \wedge \text{Lives}(b, a) \rightarrow \text{Bit}(b, c))]$$

Examples

- No human enjoys golf

Examples

- No human enjoys golf
 $\forall x[Human(x) \rightarrow \neg Enjoys(x, Golf)]$
- Some professor that is not a historian writes programs

Examples

- No human enjoys golf

$$\forall x[Human(x) \rightarrow \neg Enjoys(x, Golf)]$$

- Some professor that is not a historian writes programs

$$\exists x[Professor(x) \wedge \neg Historian(x) \wedge Writes(x, Programs)]$$

- Every boy owns a dog

Examples

- No human enjoys golf

$$\forall x[Human(x) \rightarrow \neg Enjoys(x, Golf)]$$

- Some professor that is not a historian writes programs

$$\exists x[Professor(x) \wedge \neg Historian(x) \wedge Writes(x, Programs)]$$

- Every boy owns a dog

$$\forall x \exists y[Boy(x) \rightarrow Owns(x, y)]$$

$$\exists y \forall x[Boy(x) \rightarrow Owns(x, y)]$$

- Do these mean the same thing?
- Brothers are siblings
- “Sibling” is reflexive and symmetric
- One’s mother is one’s female parent
- A first cousin is a child of a parent’s sibling

Higher-Order Logic

- FOPC quantifies over objects in the universe.
- Higher-order logic quantifies over relations and functions as well as objects.
 - All functions with a single argument return a value of 1
 - $\forall x, y [Equal(x(y), 1)]$
 - Two objects are equal iff all properties applied to them are equivalent
 - $\forall x, y [(x=y) \leftrightarrow (\forall p [p(x) \leftrightarrow p(y)])]$
 - Note that we use “=” as a shorthand for equal, meaning they are in fact the same object

Additional Operators

- Existential Elimination
 - $\exists v [..v..]$
 - Substitute k for v anywhere in sentence, where k is a constant (term with no arguments) and does not already appear in the sentence (Skolemization)
- Existential Introduction
 - If $[..g..]$ true (where g is ground term)
 - then $\exists v [..v..]$ true (v is substituted for g)
- Universal Elimination
 - $\forall x [..x..]$
 - Substitute M for x throughout entire sentence, where M is a constant and does not already appear in the sentence

Example Proof

Known:

1. If x is a parent of y , then x is older than y
 - $\forall x,y [Parent(x,y) \rightarrow Older(x,y)]$
2. If x is the mother of y , then x is a parent of y
 - $\forall x,y [Mother(x,y) \rightarrow Parent(x,y)]$
3. Lulu is the mother of Fifi
 - $Mother(Lulu, Fifi)$

Prove: Lulu is older than Fifi
 $(Older(Lulu, Fifi))$

4. $Parent(Lulu, Fifi)$
 - 2,3, Universal Elimination, Modus Ponens
5. $Older(Lulu, Fifi)$
 - 1,4, Universal Elimination, Modus Ponens
 - We “bind” the variable to a constant

Point A is twice as far from point C as point B is from A. If the distance from point B to point C is 5 inches, how far is point A from point C?



Example Proof

The law says that it is a crime for an American to sell weapons to hostile nations.

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$

Example Proof

The law says that it is a crime for an American to sell weapons to hostile nations. The country **Nono**, an enemy of America, **has some missiles**, and all of its missiles were sold to it by Colonel West, who is an American.

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$
- 2) $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$

Example Proof

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is an American.

- 1) $\text{FAx,y,z}[(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x,z,y)) \rightarrow \text{Criminal}(x)]$
- 2) $\text{EX x } [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)]$
- 3) $\text{FA x } [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)) \rightarrow \text{Sells}(\text{West}, \text{Nono},x)]$
- 4) $\text{FA x } [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\text{FA x } [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$

Prove: West is a criminal.

Prove: West is a Criminal

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$
 - 2) $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$
 - 3) $\forall x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
 - 4) $\forall x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
 - 5) $\forall x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
 - 6) American(West)
 - 7) Nation(Nono)
 - 8) Enemy(Nono, America)
 - 9) Nation(America)
- 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$
— 2 & Existential Elimination

Prove: West is a Criminal

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$
- 2) $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$
- 3) $\forall x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\forall x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\forall x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$
- 11) $\text{Owns}(\text{Nono}, \text{M1})$
 - 10 & And Elimination

Prove: West is a Criminal

- 1) $\text{FA}_{x,y,z}[(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x,z,y)) \rightarrow \text{Criminal}(x)]$
- 2) $\text{EX } x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$
- 3) $\text{FA } x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\text{FA } x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\text{FA } x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$
- 11) $\text{Owns}(\text{Nono}, \text{M1})$
- 12) $\text{Missile}(\text{M1})$
 - 10 & And Elimination

Prove: West is a Criminal

- | | | |
|----|---|--|
| 1) | $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$ | 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$ |
| 2) | $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$ | 11) $\text{Owns}(\text{Nono}, \text{M1})$ |
| 3) | $\forall x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)]$ | 12) $\text{Missile}(\text{M1})$ |
| 4) | $\forall x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$ | 13) $\text{Missile}(\text{M1}) \rightarrow \text{Weapon}(\text{M1})$
– 4 & Universal Elimination |
| 5) | $\forall x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$ | Universal Elimination |
| 6) | $\text{American}(\text{West})$ | FORALL v [] |
| 7) | $\text{Nation}(\text{Nono})$ | If true for universal variable v,
then true for a ground term |
| 8) | $\text{Enemy}(\text{Nono}, \text{America})$ | (term with no variables) |
| 9) | $\text{Nation}(\text{America})$ | |

Prove: West is a Criminal

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$
- 2) $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$
- 3) $\forall x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)) \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)]$
- 4) $\forall x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\forall x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$
- 11) $\text{Owns}(\text{Nono}, \text{M1})$
- 12) $\text{Missile}(\text{M1})$
- 13) $\text{Missile}(\text{M1}) \rightarrow \text{Weapon}(\text{M1})$
- 14) $\text{Weapon}(\text{M1})$
 - 12, 13, Modus Ponens

Prove: West is a Criminal

- 1) $\text{FA}_{x,y,z}[(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x,z,y)) \rightarrow \text{Criminal}(x)]$
- 2) $\text{EX}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)]$
- 3) $\text{FA}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\text{FA}_x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\text{FA}_x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, M1) \& \text{Missile}(M1)$
- 11) $\text{Owns}(\text{Nono}, M1)$
- 12) $\text{Missile}(M1)$
- 13) $\text{Missile}(M1) \rightarrow \text{Weapon}(M1)$
- 14) $\text{Weapon}(M1)$
- 15) $\text{Owns}(\text{Nono}, M1) \& \text{Missile}(M1) \rightarrow \text{Sells}(\text{West}, \text{Nono}, M1)$
 - 3 & Universal Elimination

Prove: West is a Criminal

- | | | |
|----|---|---|
| 1) | FAx,y,z[(American(x) & Weapon(y) & Nation(z) & Hostile(z) & Sells(x,z,y)) -> Criminal(x)] | 10) Owns(Nono,M1) & Missile(M1) |
| 2) | EX x [Owns(Nono,x) & Missile(x)] | 11) Owns(Nono, M1) |
| 3) | FA x [Owns(Nono,x) & Missile(x)] -> Sells(West, Nono,x)] | 12) Missile(M1) |
| 4) | FA x [Missile(x) -> Weapon(x)] | 13) Missile(M1) -> Weapon(M1) |
| 5) | FA x [Enemy(x,America) -> Hostile(x)] | 14) Weapon(M1) |
| 6) | American(West) | 15) Owns(Nono,M1) & Missile(M1) -> Sells(West,Nono,M1) |
| 7) | Nation(Nono) | 16) Sells(West,Nono,M1)
— 10, 15, Modus Ponens |
| 8) | Enemy(Nono, America) | |
| 9) | Nation(America) | |

Prove: West is a Criminal

- 1) $\text{FA}_{x,y,z}[(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x,z,y)) \rightarrow \text{Criminal}(x)]$
- 2) $\text{EX}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)]$
- 3) $\text{FA}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\text{FA}_x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\text{FA}_x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$
- 11) $\text{Owns}(\text{Nono}, \text{M1})$
- 12) $\text{Missile}(\text{M1})$
- 13) $\text{Missile}(\text{M1}) \rightarrow \text{Weapon}(\text{M1})$
- 14) $\text{Weapon}(\text{M1})$
- 15) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1}) \rightarrow \text{Sells}(\text{West}, \text{Nono}, \text{M1})$
- 16) $\text{Sells}(\text{West}, \text{Nono}, \text{M1})$
- 17) $\text{American}(\text{West}) \& \text{Weapon}(\text{M1}) \& \text{Nation}(\text{Nono}) \& \text{Hostile}(\text{Nono}) \& \text{Sells}(\text{West}, \text{Nono}, \text{M1}) \rightarrow \text{Criminal}(\text{West})$
 - 1, Universal Elimination (x West) (y M1) (z Nono)

Prove: West is a Criminal

- 1) $\text{FA}_{x,y,z}[(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x,z,y)) \rightarrow \text{Criminal}(x)]$
- 2) $\text{EX}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)]$
- 3) $\text{FA}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\text{FA}_x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\text{FA}_x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, M1) \& \text{Missile}(M1)$
- 11) $\text{Owns}(\text{Nono}, M1)$
- 12) $\text{Missile}(M1)$
- 13) $\text{Missile}(M1) \rightarrow \text{Weapon}(M1)$
- 14) $\text{Weapon}(M1)$
- 15) $\text{Owns}(\text{Nono}, M1) \& \text{Missile}(M1) \rightarrow \text{Sells}(\text{West}, \text{Nono}, M1)$
- 16) $\text{Sells}(\text{West}, \text{Nono}, M1)$
- 17) $\text{American}(\text{West}) \& \text{Weapon}(M1) \& \text{Nation}(\text{Nono}) \& \text{Hostile}(\text{Nono}) \& \text{Sells}(\text{West}, \text{Nono}, M1) \rightarrow \text{Criminal}(\text{West})$
- 18) $\text{Enemy}(\text{Nono}, \text{America}) \rightarrow \text{Hostile}(\text{Nono})$
 - 5, Universal Elimination

Prove: West is a Criminal

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$
- 2) $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$
- 3) $\forall x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\forall x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\forall x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1})$
- 11) $\text{Owns}(\text{Nono}, \text{M1})$
- 12) $\text{Missile}(\text{M1})$
- 13) $\text{Missile}(\text{M1}) \rightarrow \text{Weapon}(\text{M1})$
- 14) $\text{Weapon}(\text{M1})$
- 15) $\text{Owns}(\text{Nono}, \text{M1}) \& \text{Missile}(\text{M1}) \rightarrow \text{Sells}(\text{West}, \text{Nono}, \text{M1})$
- 16) $\text{Sells}(\text{West}, \text{Nono}, \text{M1})$
- 17) $\text{American}(\text{West}) \& \text{Weapon}(\text{M1}) \& \text{Nation}(\text{Nono}) \& \text{Hostile}(\text{Nono}) \& \text{Sells}(\text{West}, \text{Nono}, \text{M1}) \rightarrow \text{Criminal}(\text{West})$
- 18) $\text{Enemy}(\text{Nono}, \text{America}) \rightarrow \text{Hostile}(\text{Nono})$
- 19) $\text{Hostile}(\text{Nono})$
 - 8, 18, Modus Ponens

Prove: West is a Criminal

- 1) $\text{FA}_{x,y,z}[(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x,z,y)) \rightarrow \text{Criminal}(x)]$
- 2) $\text{EX}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)]$
- 3) $\text{FA}_x [\text{Owns}(\text{Nono},x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\text{FA}_x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\text{FA}_x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) $\text{American}(\text{West})$
- 7) $\text{Nation}(\text{Nono})$
- 8) $\text{Enemy}(\text{Nono}, \text{America})$
- 9) $\text{Nation}(\text{America})$
- 10) $\text{Owns}(\text{Nono}, M1) \& \text{Missile}(M1)$
- 11) $\text{Owns}(\text{Nono}, M1)$
- 12) $\text{Missile}(M1)$
- 13) $\text{Missile}(M1) \rightarrow \text{Weapon}(M1)$
- 14) $\text{Weapon}(M1)$
- 15) $\text{Owns}(\text{Nono}, M1) \& \text{Missile}(M1) \rightarrow \text{Sells}(\text{West}, \text{Nono}, M1)$
- 16) $\text{Sells}(\text{West}, \text{Nono}, M1)$
- 17) $\text{American}(\text{West}) \& \text{Weapon}(M1) \& \text{Nation}(\text{Nono}) \& \text{Hostile}(\text{Nono}) \& \text{Sells}(\text{West}, \text{Nono}, M1) \rightarrow \text{Criminal}(\text{West})$
- 18) $\text{Enemy}(\text{Nono}, \text{America}) \rightarrow \text{Hostile}(\text{Nono})$
- 19) $\text{Hostile}(\text{Nono})$
- 20) $\text{American}(\text{West}) \& \text{Weapon}(M1) \& \text{Nation}(\text{Nono}) \& \text{Hostile}(\text{Nono}) \& \text{Sells}(\text{West}, \text{Nono}, M1)$
 - 6, 7, 14, 16, 19, And Introduction

Prove: West is a Criminal

- 1) $\forall x, y, z [(\text{American}(x) \& \text{Weapon}(y) \& \text{Nation}(z) \& \text{Hostile}(z) \& \text{Sells}(x, z, y)) \rightarrow \text{Criminal}(x)]$
- 2) $\exists x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)]$
- 3) $\forall x [\text{Owns}(\text{Nono}, x) \& \text{Missile}(x)] \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- 4) $\forall x [\text{Missile}(x) \rightarrow \text{Weapon}(x)]$
- 5) $\forall x [\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)]$
- 6) American(West)
- 7) Nation(Nono)
- 8) Enemy(Nono, America)
- 9) Nation(America)
- 10) Owns(Nono, M1) & Missile(M1)
- 11) Owns(Nono, M1)
- 12) Missile(M1)
- 13) Missile(M1) \rightarrow Weapon(M1)
- 14) Weapon(M1)
- 15) Owns(Nono, M1) & Missile(M1) \rightarrow Sells(West, Nono, M1)
- 16) Sells(West, Nono, M1)
- 17) American(West) & Weapon(M1) & Nation(Nono) & Hostile(Nono) & Sells(West, Nono, M1) \rightarrow Criminal(West)
- 18) Enemy(Nono, America) \rightarrow Hostile(Nono)
- 19) Hostile(Nono)
- 20) American(West) & Weapon(M1) & Nation(Nono) & Hostile(Nono) & Sells(West, Nono, M1)
- 21) **Criminal(West)**
 - 17, 20, Modus Ponens

FOPC and the Wumpus World

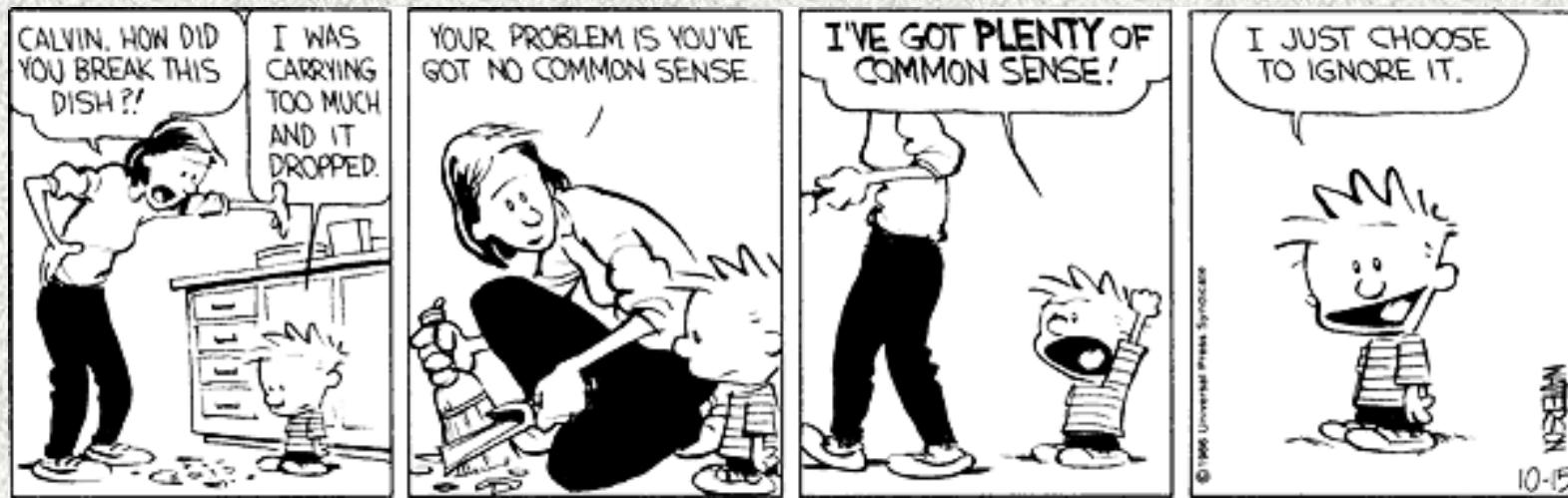
- Perception rules
 - $\forall b,g,t \text{ Percept}([\text{Smell},b,g],t) \rightarrow \text{Smelled}(t)$
 - Here we are indicating a Percept occurring at time t
 - $\forall s,b,t \text{ Percept}([s,b,\text{Glitter}],t) \rightarrow \text{AtGold}(t)$
- We can use FOPC to write rules for selecting actions:
 - Reflex agent: $\forall t \text{ AtGold}(t) \rightarrow \text{Action}(\text{Grab}, t)$
 - Reflex agent with internal state:
 $\forall t \text{ AtGold}(t) \& \neg \text{Holding}(\text{Gold},t) \rightarrow \text{Action}(\text{Grab}, t)$
 - $\text{Holding}(\text{Gold},t)$ cannot be observed, so keeping track of change is essential

Deducing Hidden Properties

- Properties of locations:
- Squares are breezy near a pit
 - Diagnostic rule: infer cause from effect
 - $\forall y \text{Breezy}(y) \rightarrow \exists x \text{ Pit}(x) \& \text{Adjacent}(x,y)$
 - Causal rule: infer effect from cause
 - $\forall x,y \text{ Pit}(x) \& \text{Adjacent}(x,y) \rightarrow \text{Breezy}(y)$
- Neither of these is complete
- For example, causal rule doesn't say whether squares far away from pits can be breezy
- Definition for Breezy predicate
 - $\forall \text{Breezy}(y) \leftrightarrow [\exists \text{ Pit}(x) \& \text{Adjacent}(x,y)]$

Inference As Search

- Operators are inference rules
- States are sets of sentences
- Goal test checks state to see if it contains query sentence
- AI, UE, MP a common inference pattern, but generate a huge branching factor
- We need a single, more powerful inference rule



Generalized Modus Ponens

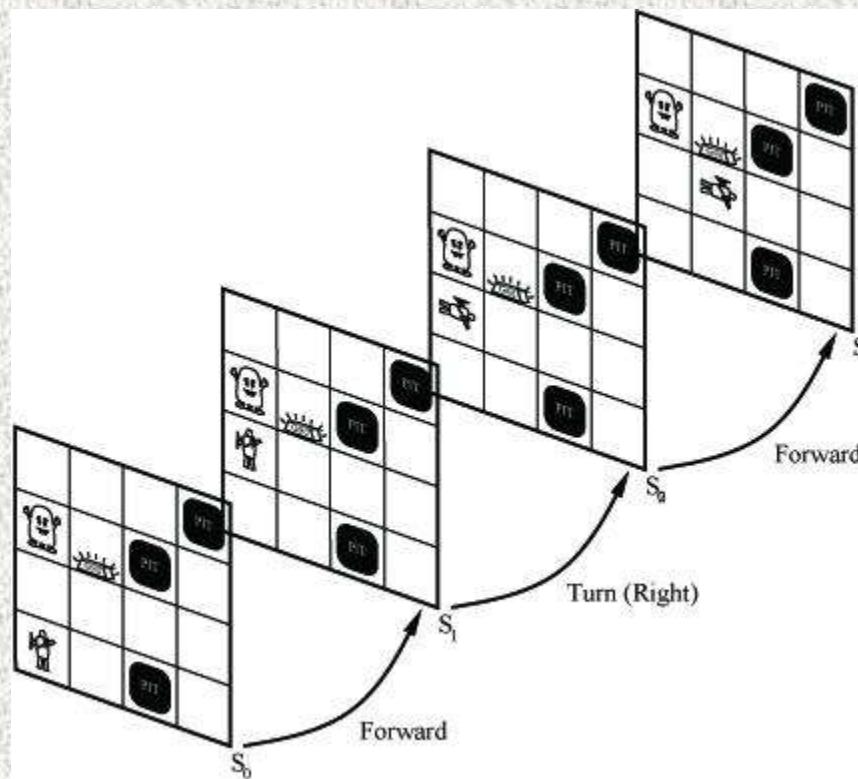
- If we have a rule
 - $p_1(x) \And p_2(x) \And p_3(x,y) \And p_4(y) \And p_5(x,y) \rightarrow q(x,y)$
- Each p involves universal / existential quantifiers
- Assume each antecedent appears in KB
 - $p_1(\text{WSU})$
 - $p_2(\text{WSU})$
 - $p_3(\text{WSU}, \text{Washington})$
 - $p_4(\text{Washington})$
 - $p_5(\text{WSU}, \text{Washington})$
- If we find a way to “match” the variables
- Then we can infer $q(\text{WSU}, \text{Washington})$

GMP Example

- Rule: $\text{Missile}(x) \& \text{Owns}(\text{Nono}, x) \rightarrow \text{Sells}(\text{West}, \text{Nono}, x)$
- KB contains
 - Missile(M1)
 - Owns(Nono,M1)
- To apply, GMP, make sure instantiations of x are the same
- Variable matching process is called **unification**

Keeping Track Of Change

- Facts hold in **situations**, rather than forever
 - Example, Holding(Gold,Now) rather than Holding(Gold)
- **Situation calculus** is one way to represent change in FOPC
 - Adds a situation argument to each time-dependent predicate
 - Example, Now in Holding(Gold,Now) denotes a situation
- Situations are connected by the Result function
 - Result(a, s) is the situation that results from applying action a in s



Describing Actions

- **Effect** axiom: describe changes due to action
 - $\forall s \text{ AtGold}(s) \rightarrow \text{Holding}(\text{Gold}, \text{Result}(\text{Grab}, s))$
- **Frame** axiom--describe **non-changes** due to action
 - $\forall s \text{ HaveArrow}(s) \rightarrow \text{HaveArrow}(\text{Result}(\text{Grab}, s))$
- **Frame problem**: find an elegant way to handle non-change
 - (a) Representation--avoid frame axioms
 - (b) Inference--avoid repeated ``copy-overs'' to keep track of state
- Qualification problem : true descriptions of real actions require endless caveats - what if gold is slippery or nailed down or ...
- Ramification problem : real actions have many secondary consequences - what about the dust on the gold, wear and tear on gloves, ...

Describing Actions

- Successor-state axioms solve the representational frame problem
- Each axiom is about a predicate (not an action per se)
 - P true afterwards \leftrightarrow
 - [an action made P true]
 - $\vee P$ true already and no action made P false]
- For holding the gold
 - $\forall a,s \text{ Holding(Gold, Result}(a,s)) \leftrightarrow ((a = \text{Grab} \ \& \ \text{AtGold}(s)) \vee (\text{Holding(gold},s) \ \& \ a \neq \text{Release}))$

Generating Plans

- Initial condition in KB
 - $\text{At}(\text{Agent}, [1,1], \text{S0})$
 - $\text{At}(\text{Gold}, [1,2], \text{S0})$
- Query
 - $\text{Ask}(\text{KB}, \exists s \text{ Holding}(\text{Gold}, s))$
 - In what situation will I be holding the gold?
- Answer: $\{s / \text{Result}(\text{Grab}, \text{Result}(\text{Forward}, \text{S0}))\}$
 - Go forward and then grab the gold
 - This assumes that the agent is interested in plans starting at S0 and that S0 is the only situation described in the KB

Generating Plans: A Better Way

- Represent plans as action sequences $[a_1, a_2, \dots, a_n]$
- $\text{PlanResult}(p, s)$ is the result of execute p (an action sequence) in s
- Then query $\text{Ask}(\text{KB}, \exists p \text{ Holding(Gold, PlanResult}(p, S_0))$
has solution $\{p/[Forward, Grab]\}$
- Definition of PlanResult in terms of Result :
 - $\forall s \text{ PlanResult}([], s) = s$
 - $\forall a, p, s \text{ PlanResult}([a | p], s) = \text{PlanResult}(p, \text{Result}(a, s))$
- Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

CptS 440 / 540

Artificial Intelligence

Logical Reasoning

Unification

- We sometimes want to “match” statements
 - $\neg \text{dog}(\text{x}) \vee \text{feathers}(\text{x})$
 - $\text{Feathers}(\text{Tweety})$
 - $\text{Dog}(\text{Rufus})$
 - $(\text{x } \text{Tweety})$
- The match needs to be consistent
- During the match we build a **binding list**
- Example
 - $\neg \text{hold}(\text{P1}, \text{?card}) \wedge \neg \text{hold}(\text{P2}, \text{?card}) \wedge \neg \text{hold}(\text{P3}, \text{?card}) \rightarrow \text{solution}(\text{?card})$
 - $\neg \text{hold}(\text{P1}, \text{Rope}) \wedge \neg \text{hold}(\text{P2}, \text{Rope}) \wedge \neg \text{hold}(\text{P3}, \text{Rope})$
 - If we substitute ?card with Rope everywhere $(\text{?card } \text{Rope})$ then statement is equivalent to the left-hand side of the rule
- Two expressions are **unifiable** iff there exists a substitution list (binding list) that, when applied to both expressions, makes them the same

Unification

- Three valid types of substitutions:
 1. variable -> constant
 2. variable1 -> variable2
 3. variable -> function, if function doesn't contain variable
 - Things to look out for:
 1. What if variable is already bound (substituted)?
 $(f ?x ?x)$ and $(f \text{ Peter Mary})$
 2. What if function contains a second variable, which is already bound to the first variable?
 $(f ?x)$ and $(f \text{ red(balloon)})$
 $(f ?x)$ and $(f ?y)$
 $(f ?x)$ and $(f \text{ red(?x)})$
- 1. Programmer(?x)
 - 2. Programmer(Cook)
 - $((?x \text{ Cook}))$
 - 1. Programmer(Cook)
 - 2. Programmer(Cook)
- 1. Father(?x, Father(john))
 - 2. Father(Grandfather(?y), Father(?y))
 - $((?x \text{ Grandfather(?y)}) \ (?y \text{ John}))$
 - 1. Father(Grandfather(John), Father(John))
 - 2. Father(Grandfather(John), Father(John))

Unification Code

- ```
// Determine whether two expressions can be unified. If yes, return bindings. If no, return FAIL
```
- Function unify(p1, p2, bindings)
    1. If (p1 = p2) return bindings
    2. If var(p1) try to add (p1 p2) to list of bindings
    3. If var(p2) try to add (p2 p1) to list of bindings
    4. If p1&p2 are length 1 return FAIL
    5. If (length(p1) != length(p2)) return FAIL
    6. Recursively unify each term pair in p1 and p2
    7. Return binding list

# Unification Code

// See if (var pat) can be added to bindings

- Function varunify(var, pat, bindings)
  1. If var is already in bindings unify new match with old match
  2. See if pattern already unifies with var using bindings
  3. If var occurs in pat return FAIL
  4. Else add (var pat) to bindings

# Unification Code

- Function unify(p1, p2, bindings)
  1. If  $(p1 = p2)$  return bindings
  2. If  $\text{var}(p1)$  try to add  $(p1 p2)$  to list of bindings
  3. If  $\text{var}(p2)$  try to add  $(p2 p1)$  to list of bindings
  4. If  $p1 \& p2$  are length 1 return FAIL
  5. If  $(\text{length}(p1) \neq \text{length}(p2))$  return FAIL
  6. Recursively unify each term pair in p1 and p2, return result
  7. Return binding list
- Function varunify(var, pat, bindings)
  1. If var is already in bindings unify new match with old match
  2. See if pattern already unifies with var using bindings
  3. If var occurs in pat return FAIL
  4. Else add  $(\text{var pat})$  to bindings

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - ?
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - ?
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - ?
- $f(?x, ?y, ?z)$  and  $f(?y, ?z,$   
 $\text{Fred})$ 
  - ?
- $p(?x, ?x)$  and  
 $p(\text{cook}, \text{henderson})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$ 
  - ?
- Try:  
 $(r, f(?y), ?y, ?x)$  and  
 $(r, ?x, f(A), f(?v)))$

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - OK  $((?x ?y) (?y ?z))$
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - ?
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - ?
- $f(?x, ?y, ?z)$  and  $f(?y, ?z,$   
 $\text{Fred})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, \text{henderson})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$ 
  - ?
- Try:  
 $(r, f(?y), ?y, ?x)$  and  
 $(r, ?x, f(A), f(?v)))$

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - OK  $((?x ?y) (?y ?z))$
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - ?
- $f(?x, ?y, ?z)$  and  $f(?y, ?z,$   
 $\text{Fred})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, \text{henderson})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$ 
  - ?
- Try:  
 $(r, f(?y), ?y, ?x)$  and  
 $(r, ?x, f(A), f(?v)))$

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - OK  $((?x ?y) (?y ?z))$
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, ?z, \text{Fred})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, \text{henderson})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$ 
  - ?
- Try:
  - $(r, f(?y), ?y, ?x)$  and
  - $(r, ?x, f(A), f(?v)))$

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - OK  $((?x ?y) (?y ?z))$
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, ?z, \text{Fred})$ 
  - OK  $((?x ?y) (?y ?z) (?z \text{Fred}))$
- $p(?x, ?x)$  and  $p(\text{cook}, \text{henderson})$ 
  - ?
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$ 
  - ?
- Try:
  - $(r, f(?y), ?y, ?x)$  and
  - $(r, ?x, f(A), f(?v)))$

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - OK  $((?x ?y) (?y ?z))$
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, ?z, \text{Fred})$ 
  - OK  $((?x ?y) (?y ?z) (?z \text{Fred}))$
- $p(?x, ?x)$  and  $p(\text{cook}, \text{henderson})$ 
  - NO
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$ 
  - ?
- Try:
  - $(r, f(?y), ?y, ?x)$  and
  - $(r, ?x, f(A), f(?v)))$

# Examples

- $f(?x, ?x)$  and  $f(?y, ?z)$ 
  - OK  $((?x ?y) (?y ?z))$
- $f(?x, ?x)$  and  $f(\text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, \text{John}, \text{Fred})$ 
  - NO
- $f(?x, ?y, ?z)$  and  $f(?y, ?z, \text{Fred})$ 
  - OK  $((?x ?y) (?y ?z) (?z \text{Fred}))$
- $p(?x, ?x)$  and  $p(\text{cook}, \text{henderson})$   
NO
- $p(?x, ?x)$  and  $p(\text{cook}, ?y)$   
OK  $((?x \text{ cook}) (?y \text{ cook}))$
- Try:  
 $(r, f(?y), ?y, ?x)$  and  
 $(r, ?x, f(A), f(?v)))$

# Unifiers

- Note that there can be more than one binding list that unifies two expressions.
  - $(f ?x) (f ?y)$
  - binding list =  $((?x ?y))$  or  $((?x \text{ foo}) (?y \text{ foo}))$
- In general, we prefer to not overly constrain the substitutions.
- If keep general, result can apply to greater number of future situations.
- The less constraints placed, the more general the substitution (the binding list, or unifier)
- Given expressions p and q, a **unifier** of p and q is any binding list b such that  $pb = qb$  ( $pb$  means binding list b is applied to expression p)

# Most General Unifier

- Unifier  $b_1$  is more general than unifier  $b_2$  if for every expression  $p$ ,  $pb_2$  is an instance of  $pb_1$  (or  $pb_1 b_3 = pb_2$ )
- $((?x ?y))$  is more general than  $((?x \text{ foo}) (?y \text{ foo}))$ 
  - because  $(f ?x) ((?x ?y)) ((?y \text{ foo})) = (f ?x) ((?x \text{ foo}) (?y \text{ foo}))$
- If two expressions are unifiable, then there exists an **mgu** (**most general unifier**)
- The code we designed returns an mgu.
  - $Q(F(?x), ?z, A)$  and  $Q(?a, ?z, ?y)$
  - Unifiable?
  - Yes! substitution list =  $((?a F(?x)) (?y A))$
  - $P(?x)$  and  $P(A)?$
  - $P(F(?x, G(A,?y)), G(A,?y))$  and  $P(F(?x,?z), ?z)?$
  - $Q(?x, ?y, A)$  and  $Q(B, ?y, ?z)?$
  - $R(?x)$  and  $R(F(?x))?$

# Unification and Eliza

- Very popular AI program developed in the 60s.
- Eliza simulates a Rogerian psychiatrist.
- Takes your statements, permutes them, rephrases them as a question.
- Can it pass the Turing test?
- Parry - simulates paranoiac
- Simulator of autistic child
  - hello
  - i would like to talk about artificial intelligence
  - i don't want to talk about you right now
  - what?
  - i want to know about ai
  - i like wsu but i have to read this book
  - help
  - goodbye
- [Try it out](#)

# Eliza

- Eliza uses unification to match input sentences against a set of rules.  
Looks at \*eliza-rules\*.
- Here variable can match a series of words.
- Rule 2:
  - (((?\* ?x) I want (?\* ?y))  
(What would it mean if you got ?y))
  - means
  - Look for the words “I want”. Match ?x to everything before the phrase, and match ?y to everything after the phrase.
  - Now respond “What would it mean if you got ?y”
  - Every now and then, I want a chocolate malt.
  - What would it mean if you got a chocolate malt

# Eliza Rules

```
(defparameter *eliza-rules*
'(((?* ?x) hello (?* ?y))
 (How do you do. Please state your problem.))
(((?* ?x) computer (?* ?y))
 (Do computers worry you?)
 (What do you think about machines?)
 (Why do you mention computers?)
 (What do you think machines have to do with
your problem?))
(((?* ?x) name (?* ?y))
 (I am not interested in names))
(((?* ?x) sorry (?* ?y))
 (Please don't apologize)
 (Apologies are not necessary)
 (What feelings do you have when you
apologize))
(((?* ?x) I remember (?* ?y))
 (Do you often think of ?y)
 (Does thinking of ?y bring anything else to
mind?))
 (What else do you remember)
 (Why do you recall ?y right now?)
 (What in the present situation reminds you of
?y)
 (What is the connection between me and ?y))
(((?* ?x) do you remember (?* ?y))
 (Did you think I would forget ?y ?)
 (Why do you think I should recall ?y now)
 (What about ?y)
 (You mentioned ?y))
(((?* ?x) if (?* ?y))
 (Do you really think its likely that ?y)
 (Do you wish that ?y)
 (What do you think about ?y)
 (Really-- if ?y))
```

# Eliza Rules

- ((?\* ?x) I dreamed about (?\* ?y))  
(Really-- ?y)  
(Have you ever fantasized ?y while you were awake?)  
(Have you dreamed about ?y before?)  
((?\* ?x) dream about (?\* ?y))  
(How do you feel about ?y in reality?)  
((?\* ?x) dream (?\* ?y))  
(What does this dream suggest to you?)  
(Do you dream often?)  
(What persons appear in your dreams?)  
(Don't you believe that dream has to do with your problem?)  
((?\* ?x) my mother (?\* ?y))  
(Who else in your family ?y)  
(Tell me more about your family))  
((?\* ?x) my father (?\* ?y))  
(Your father)  
(Does he influence you strongly?)  
(What else comes to mind when you think of your father?))  
((?\* ?x) I want (?\* ?y))  
(What would it mean if you got ?y)  
(Why do you want ?y)  
(Suppose you got ?y soon))  
((?\* ?x) if (?\* ?y))  
(Do you really think its likely that ?y)  
(Do you wish that ?y)  
(What do you think about ?y)  
(Really-- if ?y))
- ((?\* ?x) no (?\* ?y))  
(Why not?)  
(You are being a bit negative)  
(Are you saying "NO" just to be negative?))  
((?\* ?x) I was (?\* ?y))  
(Were you really?)  
(Perhaps I already knew you were ?y)  
(Why do you tell me you were ?y now?))  
((?\* ?x) I am glad (?\* ?y))  
(How have I helped you to be ?y)  
(What makes you happy just now)  
(Can you explain why you are suddenly ?y)) ((?\* ?x) I am sad (?\* ?y))  
(I am sorry to hear you are depressed)  
(I'm sure its not pleasant to be sad))  
((?\* ?x) are like (?\* ?y))  
(What resemblance do you see between ?x and ?y))  
((?\* ?x) is like (?\* ?y))  
(In what way is it that ?x is like ?y)  
(What resemblance do you see?)  
(Could there really be some connection?)  
(How?))  
(((?\* ?x) alike (?\* ?y))  
(In what way?)  
(What similarities are there?))  
(((?\* ?x) same (?\* ?y))  
(What other connections do you see?))

# Eliza Rules

|                                                     |                                                        |
|-----------------------------------------------------|--------------------------------------------------------|
| ((?* ?x) I was (?* ?y))                             |                                                        |
| (Were you really?)                                  | ...                                                    |
| (Perhaps I already knew you were ?y)                |                                                        |
| (Why do you tell me you were ?y now?)               |                                                        |
| ((?* ?x) was I (?* ?y))                             | ((?* ?x))                                              |
| (What if you were ?y ?)                             | (Very interesting)                                     |
| (Do you thin you were ?y)                           | (I am not sure I understand you fully)                 |
| (What would it mean if you were ?y))                | (What does that suggest to you?)                       |
| ((?* ?x) I am (?* ?y))                              | (Please continue)                                      |
| (In what way are you ?y)                            | (Go on)                                                |
| (Do you want to be ?y ?))                           | (Do you feel strongly about discussing such things?))) |
| ((?* ?x) am I (?* ?y))                              |                                                        |
| (Do you believe you are ?y)                         |                                                        |
| (Would you want to be ?y)                           |                                                        |
| (You wish I would tell you you are ?y)              |                                                        |
| (What would it mean if you were ?y))                |                                                        |
| ((?* ?x) am (?* ?y))                                |                                                        |
| (Why do you say "AM?")                              |                                                        |
| (I don't understand that))                          |                                                        |
| ((?* ?x) are you (?* ?y))                           |                                                        |
| (Why are you interested in whether I am ?y or not?) |                                                        |
| (Would you prefer if I weren't ?y)                  |                                                        |
| (Perhaps I am ?y in your fantasies))                |                                                        |
| ((?* ?x) you are (?* ?y))                           |                                                        |
| (What makes you think I am ?y ?))                   |                                                        |

# Sample Run

```
#<STRING-CHAR-FILE-STREAM #'out">
> (hello there eliza)
> (eliza)
```

```
ELIZA> (hello there eliza)(HOW DO YOU DO. PLEASE STATE YOUR PROBLEM.)
ELIZA> (computer are like students)(DO COMPUTERS WORRY YOU?)
ELIZA> (computers are like students)
 (WHAT RESEMBLANCE DO YOU SEE BETWEEN COMPUTERS AND STUDENTS)
ELIZA> (they both run if they see programs)(REALLY-- IF THEY SEE PROGRAMS)
ELIZA> (yes that is correct)(YOU ARE SURE)
ELIZA> (yes)(I UNDERSTAND)
ELIZA> (what computer is your mother)(DO COMPUTERS WORRY YOU?)
ELIZA> (only your mother)(PLEASE CONTINUE)
ELIZA> (my mother is not a computer)
 (WHAT DO YOU THINK MACHINES HAVE TO DO WITH YOUR PROBLEM?)
```

# Sample Run

ELIZA> (my mother is not a machine)(TELL ME MORE ABOUT YOUR FAMILY)  
ELIZA> (my father is not a machine either)(DOES HE INFLUENCE YOU STRONGLY?)  
ELIZA> (no)(YOU ARE BEING A BIT NEGATIVE)  
ELIZA> (yes)(YOU ARE SURE)  
ELIZA> (i want to stop this conversation)(WHY DO YOU WANT TO STOP THIS CONVERSATION)  
ELIZA> (do you remember what happened last time you talked this way)  
(WHAT ABOUT WHAT HAPPENED LAST TIME I TALKED THIS WAY)  
ELIZA> (enough)(PLEASE CONTINUE)  
ELIZA> (no)(ARE YOU SAYING "NO" JUST TO BE NEGATIVE?)  
ELIZA> (yes)(I UNDERSTAND)  
ELIZA> (bye)(DO YOU FEEL STRONGLY ABOUT DISCUSSING SUCH THINGS?)  
ELIZA> (quit)(WHAT DOES THAT SUGGEST TO YOU?)

# Eliza and Parry Sample Run

I don't understand your motives.

WHY DON'T YOU UNDERSTAND MY MOTIVES?

Shouldn't I be?

WHY DID YOU SAY THAT JUST NOW?

That's just the way it is.

SUPPOSE THAT'S JUST THE WAY IT WERE NOT.

So?

I'M NOT SURE I UNDERSTAND YOU FULLY.

You harp on the same questions too much.

WHAT DOES THAT SIMILARITY SUGGEST TO YOU?

I don't get you.

DO YOU WISH THAT YOU DID GET I?

# Example Proof With Unification

|                                  |    |                                                                            |
|----------------------------------|----|----------------------------------------------------------------------------|
| Bob is a buffalo                 | 1. | Buffalo(Bob)                                                               |
| Pat is a pig                     | 2. | Pig(Pat)                                                                   |
| Buffalo outrun pigs              | 3. | $\forall x, y$<br>Buffalo(x) $\wedge$ Pig(y) $\rightarrow$<br>Faster(x, y) |
| Prove: Bob is faster<br>than Pat |    |                                                                            |

# Example Proof With Unification

|                                  |    |                                                                            |
|----------------------------------|----|----------------------------------------------------------------------------|
| Bob is a buffalo                 | 1. | Buffalo(Bob)                                                               |
| Pat is a pig                     | 2. | Pig(Pat)                                                                   |
| Buffalo outrun pigs              | 3. | $\forall x, y$<br>Buffalo(x) $\wedge$ Pig(y) $\rightarrow$<br>Faster(x, y) |
| Prove: Bob is faster<br>than Pat |    |                                                                            |
| AI 1 & 2<br>(or GMP(1,2,3))      | 4. | Buffalo(Bob) $\wedge$ Pig(Pat)                                             |

# Example Proof With Unification

|                                  |    |                                                  |
|----------------------------------|----|--------------------------------------------------|
| Bob is a buffalo                 | 1. | Buffalo(Bob)                                     |
| Pat is a pig                     | 2. | Pig(Pat)                                         |
| Buffalo outrun pigs              | 3. | FA x,y<br>Buffalo(x) ^ Pig(y) -><br>Faster(x, y) |
| Prove: Bob is faster<br>than Pat |    |                                                  |
| AI 1 & 2                         | 4. | Buffalo(Bob) ^ Pig(Pat)                          |
| UE 3, {x/Bob, y/Pat}             | 5. | Buffalo(Bob) ^ Pig(Pat)<br>-> Faster(Bob, Pat)   |

# Example Proof With Unification

|                                  |    |                                                                            |
|----------------------------------|----|----------------------------------------------------------------------------|
| Bob is a buffalo                 | 1. | Buffalo(Bob)                                                               |
| Pat is a pig                     | 2. | Pig(Pat)                                                                   |
| Buffalo outrun pigs              | 3. | $\forall x, y$<br>Buffalo(x) $\wedge$ Pig(y) $\rightarrow$<br>Faster(x, y) |
| Prove: Bob is faster<br>than Pat |    |                                                                            |
| AI 1 & 2                         | 4. | Buffalo(Bob) $\wedge$ Pig(Pat)                                             |
| UE 3, {x/Bob, y/Pat}             | 5. | Buffalo(Bob) $\wedge$ Pig(Pat)<br>$\rightarrow$ Faster(Bob, Pat)           |
| MP 4 & 5                         | 6. | Faster(Bob, Pat)                                                           |

# Resolution

- A complete inference rule
  - $(a \vee b, \neg b) \rightarrow ? a$
  - $(p_1 \vee p_2 \vee \dots \vee p_m, \neg p_1 \vee p_n \vee \dots \vee p_q) \rightarrow (p_2 \vee p_m \vee p_n \vee \dots \vee p_q)$
- Unification with resolution
  - $P(?x) Q(?x), Q(\text{Mary})$
  - UNIFY the two clauses  $P(\text{Mary}) \vee Q(\text{Mary}), \neg Q(\text{Mary})$ ,  
binding list  $((?x \text{ Mary}))$
  - Result is  $P(\text{Mary})$

# Proof By Refutation Using Resolution

- To prove a statement using resolution:
  - Convert each statement in database to clausal form  
(Clausal form is a disjunction of literals)
  - Negate the goal statement and convert to clausal form
  - Repeatedly apply resolution to generate new statements
  - If generate empty statement [], proof is complete
- This is Proof by refutation
  - We know database statements are all true
  - Resolution is sound  
All statements generated are true if database is true
  - How can we come up with empty (false) clause?
  - Only possible false statement is negated goal
  - If negated goal is false, goal must be true ( $A \vee \neg A$  is a tautology)

# Example

- If Linus is sitting in the pumpkin patch, it must be Halloween. The Great Pumpkin appears on Halloween. Linus is sitting in the pumpkin patch. Prove the Great Pumpkin will appear today.
1.  $\text{SitPatch}(\text{Linus}) \rightarrow \text{Halloween}(\text{Today})$
  2.  $\text{Halloween}(\text{Today}) \rightarrow \text{GrPumpkin}(\text{Today})$
  3.  $\text{SitPatch}(\text{Linus})$

Rewrite:

1.  $\neg \text{SitPatch}(\text{Linus}) \vee \text{Halloween}(\text{Today})$
2.  $\neg \text{Halloween}(\text{Today}) \vee \text{GrPumpkin}(\text{Today})$
3.  $\text{SitPatch}(\text{Linus})$
4.  $\neg \text{GrPumpkin}(\text{Today}) \quad // \text{ Negated goal}$

Proof:

# Example

- If Linus is sitting in the pumpkin patch, it must be Halloween. The Great Pumpkin appears on Halloween. Linus is sitting in the pumpkin patch. Prove the Great Pumpkin will appear today.

1.  $\text{SitPatch}(\text{Linus}) \rightarrow \text{Halloween}(\text{Today})$
2.  $\text{Halloween}(\text{Today}) \rightarrow \text{GrPumpkin}(\text{Today})$
3.  $\text{SitPatch}(\text{Linus})$

Rewrite:

1.  $\neg \text{SitPatch}(\text{Linus}) \vee \text{Halloween}(\text{Today})$
2.  $\neg \text{Halloween}(\text{Today}) \vee \text{GrPumpkin}(\text{Today})$
3.  $\text{SitPatch}(\text{Linus})$
4.  $\neg \text{GrPumpkin}(\text{Today})$  // Negated goal

Proof:

5. [2,4]  $\neg \text{Halloween}(\text{Today})$
6. [1,5]  $\neg \text{SitPatch}(\text{Linus})$
7. [3,6] NULL

# Example

- If the maid stole the jewelry, then the butler wasn't guilty. Either the maid stole the jewelry or she milked the cows. If the maid milked the cows, then the butler got the cream.
- Goal: Therefore, if the butler was guilty, then he got the cream.

$$G(M) \rightarrow \neg G(B)$$

$$G(M) \vee \text{Cows}(M)$$

$$\neg \text{Cows}(M) \vee \text{Cream}(B)$$

$$\neg(G(B) \rightarrow \text{Cream}(B))$$

Rewrite:

1.  $\neg G(M) \vee \neg G(B)$
2.  $G(M) \vee \text{Cows}(M)$
3.  $\neg \text{Cows}(M) \vee \text{Cream}(B)$
4.  $G(B)$
5.  $\neg \text{Cream}(B)$

# Example

- If the maid stole the jewelry, then the butler wasn't guilty. Either the maid stole the jewelry or she milked the cows. If the maid milked the cows, then the butler got the cream.
- Goal: Therefore, if the butler was guilty, then he got the cream.

$G(M) \rightarrow \neg G(B)$

$G(M) \vee \text{Cows}(M)$

$\neg \text{Cows}(M) \vee \text{Cream}(B)$

$\neg(G(B) \rightarrow \text{Cream}(B))$

Rewrite:

1.  $\neg G(M) \vee \neg G(B)$

2.  $G(M) \vee \text{Cows}(M)$

3.  $\neg \text{Cows}(M) \vee \text{Cream}(B)$

4.  $G(B)$

5.  $\neg \text{Cream}(B)$

Proof:

6. [3, 5]  $\neg \text{Cows}(M)$

7. [2, 6]  $G(M)$

8. [1, 7]  $\neg G(B)$

9. [4, 8] NULL

Q.E.D.

# Unification and Resolution

Using the database below, prove that Colonel Mustard is a suspect.

1. crime(Kitchen)
2. in(ProfPlum, Library)
3. talking(MissScarlet, ProfPlum)
4. -talking(?x4, ?y4) v with(?x4, ?y4)
5. -with(?x5, ?y5) v -in(?y5, ?z5) v in(?x5, ?z5)
6. -in(?x6, ?y6) v -crime(?y6) v suspect(?x6)
7. talking(ColonelMustard, MisterGreen)
8. -hungry(?x8) v in(?x8, Kitchen)
9. hungry(MisterGreen)

# Example

1. crime(Kitchen)
  2. in(ProfPlum, Library)
  3. talking(MissScarlet, ProfPlum)
  4. -talking(?x4, ?y4) v with(?x4, ?y4)
  5. -with(?x5, ?y5) v -in(?y5, ?z5) v in(?x5, ?z5)
  6. -in(?x6, ?y6) v -crime(?y6) v suspect(?x6)
  7. talking(ColonelMustard, MisterGreen)
  8. -hungry(?x8) v in(?x8, Kitchen)
  9. hungry(MisterGreen)
  10. -suspect(CM)
  11. (6&10) -in(CM, ?y11) v -crime(?y11)
  12. (1&11) -in(CM, Kitchen)
  13. (8&9) in(MG, Kitchen)
  14. (4&7) with(CM, MG)
  15. (5&14) -in(MG, ?z15) v in(CM, ?z15)
  16. (12&15) in(CM, Kitchen)
  17. (12&16) []
- Q.E.D.
- What clause would you add to the database instead of the negated goal if you were trying to find out who is a suspect?

# Example

1. Mother(Lulu, Fifi)
2. Alive(Lulu)
3. FA x, y [Mother(x,y) -> Parent(x,y)  
-Mother(x,y) v Parent(x,y)]
4. FA x, y [Parent(x,y) ^ Alive(x)) -> Older(x,y)  
-Parent(x,y) v -Alive(x) v Older(x,y)]
5. (Negated Goal) -Older(Lulu, Fifi)
6. [1,3] Parent(Lulu, Fifi)
7. [4,6] -Alive(Lulu) v Older(Lulu, Fifi)
8. [2,7] Older(Lulu, Fifi)
9. [5,8] NULL

# Example

- What if the desired conclusion was “Something is older than Fifi”?
  - EXISTS x Older(x, Fifi)
  - (Negated Goal) ~EXISTS x Older(x, Fifi)
  - ~Older(x, Fifi)
- The last step of the proof would be
  - [5,8] Older(Lulu, Fifi) resolved with ~Older(x, Fifi)  
NULL (with x instantiated as Lulu)
- Do not make the mistake of first forming clause from conclusion and then denying it
  - Goal: EXISTS x Older(x, Fifi)
  - Clausal Form: Older(C, Fifi)
  - (Negated Goal): ~Older(C, Fifi)
- Cannot unify this statement with Older(Lulu, Fifi)

# Converting To Clausal Form

- Two benefits of seeing this process:
  1. Learn sound inference rules
  2. Convert FOPC to clausal form for use in resolution proofs



# Examples

- $P(A) \rightarrow Q(B,C)$   
 $\sim P(A) \vee Q(B,C)$
- $\sim(P(A) \rightarrow Q(B,C))$   
 $P(A), \sim Q(B,C)$
- $P(A) \wedge (Q(B,C) \vee R(D))$   
 $P(A), Q(B,C) \vee R(D)$
- $P(A) \vee (Q(B,C) \wedge R(D))$   
 $P(A) \vee Q(B,C)$   
 $P(A) \vee R(D)$
- $\text{FORALL } x \ P(x)$   
 $P(x)$
- $\text{FORALL } x \ P(x) \rightarrow Q(x,A)$   
 $\sim P(x) \vee Q(x,A)$
- $\text{EXISTS } x \ P(x)$   
 $P(E)$ , where E is a new constant
- $P(A) \rightarrow \text{EXISTS } x \ Q(x)$   
 $\sim P(A) \vee Q(F)$
- $\text{FORALL } x \ P(x)$   
 $\sim P(G)$

# Example

- First, a brick is on something that is not a pyramid; second, there is nothing that a brick is on and that is on the brick as well; and third, there is nothing that is not a brick and also is the same thing as a brick.
- $\forall x [Brick(x) \rightarrow (\exists y [On(x,y) \wedge \neg Pyramid(y)] \wedge \neg \exists y [On(x,y) \wedge On(y,x)]) \wedge \forall y [\neg Brick(y) \rightarrow \neg Equal(x,y)]]$

# Example

- $\text{FORALL } x \text{ [Brick}(x) \rightarrow (\text{EXISTS } y \text{ [On}(x,y) \& \sim\text{Pyramid}(y)] \& \sim\text{EXISTS } y \text{ [On}(x,y) \& \text{On}(y,x)] \& \text{FORALL } y [\sim\text{Brick}(y) \rightarrow \sim\text{Equal}(x,y)])]$

## 1) Eliminate implications

Note that  $p \rightarrow q \leftrightarrow \sim p \vee q$

$\text{FORALL } x [\sim\text{Brick}(x) \vee$   
 $(\text{EXISTS } y \text{ [On}(x,y) \& \sim\text{Pyramid}(y)] \&$   
 $\sim\text{EXISTS } y \text{ [On}(x,y) \& \text{On}(y,x)] \&$   
 $\text{FORALL } y [\sim\sim\text{Brick}(y) \vee \sim\text{Equal}(x,y)])]$

# Example

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
  ~EXISTS y [On(x,y) & On(y,x)] &  
  FORALL y [~~Brick(y) v ~Equal(x,y)])]

## 2) Move $\sim$ to literals

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
  ~EXISTS y [On(x,y) & On(y,x)] &  
  FORALL y [~~Brick(y) v ~Equal(x,y)])]

Note that  $\sim(p \vee q) \leftrightarrow \sim p \wedge \sim q$   
 $\sim(p \wedge q) \leftrightarrow \sim p \vee \sim q$   
 $\sim\text{FORALL } x [p] \leftrightarrow \text{EXISTS } x [\sim p]$   
 $\sim\text{EXISTS } x [p] \leftrightarrow \text{FORALL } x [\sim p]$   
 $\sim\sim p \leftrightarrow p$

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
  FORALL y ~[On(x,y) & On(y,x)] &  
  FORALL y [Brick(y) v ~Equal(x,y)])]

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
  FORALL y [~On(x,y) v ~On(y,x)] &  
  FORALL y [Brick(y) v ~Equal(x,y)])]

# Example

$$\begin{aligned} \text{FORALL } x & [ \sim \text{Brick}(x) \vee (\text{EXISTS } y [ \text{On}(x,y) \wedge \sim \text{Pyramid}(y) ] \wedge \\ & \quad \text{FORALL } y [ \sim \text{On}(x,y) \vee \sim \text{On}(y,x) ] \wedge \\ & \quad \text{FORALL } y [ \text{Brick}(y) \vee \sim \text{Equal}(x,y) ]) ] \end{aligned}$$

## 3) Standardize variables (rename variables if duplicates)

$$\begin{aligned} \text{FORALL } x & [ \sim \text{Brick}(x) \vee (\text{EXISTS } y [ \text{On}(x,y) \wedge \sim \text{Pyramid}(y) ] \wedge \\ & \quad \text{FORALL } a [ \sim \text{On}(x,a) \vee \sim \text{On}(a,x) ] \wedge \\ & \quad \text{FORALL } b [ \text{Brick}(b) \vee \sim \text{Equal}(x,b) ]) ] \end{aligned}$$

# Example

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
FORALL a [~On(x,a) v ~On(a,x)] &  
FORALL b [Brick(b) v ~Equal(x,b)])]

## 4) Skolemization

Remove existential quantifiers using Existential Elimination.  
Make sure new constant is not used anywhere else.  
Dangerous because could be inside universal quantifiers

# Example

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
FORALL a [~On(x,a) v ~On(a,x)] &  
FORALL b [Brick(b) v ~Equal(x,b)])]

## 4) Skolemization

FORALL x Person(x) -> EXISTS y Heart(y) & Has(y,x)

If we just replace y with H, then

FORALL x Person(x) -> Heart(H) & Has(x,H)

which says that everyone has the same heart.

- Remember that because y is in the scope of x, y is dependent on the choice of x. In fact, we can represent y as a Skolemized function of x.
- FORALL x Person(x) -> Heart(F(x)) & Has(x,F(x))

# Example

FORALL x [~Brick(x) v (EXISTS y [On(x,y) & ~Pyramid(y)] &  
FORALL a [~On(x,a) v ~On(a,x)] &  
FORALL b [Brick(b) v ~Equal(x,b)])]

## 4) Skolemization

FORALL x [~Brick(x) v (On(x,F(x)) & ~Pyramid(F(x)) &  
FORALL a [~On(x,a) v ~On(a,x)] &  
FORALL b [Brick(b) v ~Equal(x,b)])]

# Example

FORALL x [~Brick(x) v (On(x,F(x)) & ~Pyramid(F(x)) &  
FORALL a [~On(x,a) v ~On(a,x)] &  
FORALL b [Brick(b) v ~Equal(x,b))]]

## 5) Remove universal quantifiers

~Brick(x) v (On(x,F(x)) & ~Pyramid(F(x)) &  
(~On(x,a) v ~On(a,x)) &  
(Brick(b) v ~Equal(x,b)))

## 6) Distribute & over v

(~Brick(x) v On(x,F(x)) &  
(~Brick(x) v ~Pyramid(F(x))) &  
(~Brick(x) v ~On(x,a) v ~On(a,x)) &  
(~Brick(x) v Brick(b) v ~Equal(x,b))

Note that

$$a \& (b \vee c \vee d)$$

$\leftrightarrow$

$$(a \& b) \vee (a \& c) \vee (a \& d)$$

and

$$a \vee (b \& c \& d)$$

$\leftrightarrow$

$$(a \vee b) \& (a \vee c) \& (a \vee d)$$

# Example

$(\sim \text{Brick}(x) \vee \text{On}(x, F(x))) \ \&$   
 $(\sim \text{Brick}(x) \vee \sim \text{Pyramid}(F(x))) \ \&$   
 $(\sim \text{Brick}(x) \vee \sim \text{On}(x, a) \vee \sim \text{On}(a, x)) \ \&$   
 $(\sim \text{Brick}(x) \vee \text{Brick}(b) \vee \sim \text{Equal}(x, b))$

## 7) Separate conjuncts into unique sentences

1.  $\sim \text{Brick}(x) \vee \text{On}(x, F(x))$
2.  $\sim \text{Brick}(x) \vee \sim \text{Pyramid}(F(x))$
3.  $\sim \text{Brick}(x) \vee \sim \text{On}(x, a) \vee \sim \text{On}(a, x)$
4.  $\sim \text{Brick}(x) \vee \text{Brick}(b) \vee \sim \text{Equal}(x, b)$

## 8) Give each sentence unique variables (**uniquify**)

1.  $\sim \text{Brick}(x_1) \vee \text{On}(x_1, F(x_1))$
2.  $\sim \text{Brick}(x_2) \vee \sim \text{Pyramid}(F(x_2))$
3.  $\sim \text{Brick}(x_3) \vee \sim \text{On}(x_3, a_3) \vee \sim \text{On}(a_3, x_3)$
4.  $\sim \text{Brick}(x_4) \vee \text{Brick}(b_4) \vee \sim \text{Equal}(x_4, b_4)$

# Resolution

- Given:
  - Whoever can read is literate  
 $\forall x [Read(x) \rightarrow Literate(x)]$
  - Dolphins are not literate  
 $\forall x [Dolphin(x) \rightarrow \neg Literate(x)]$
  - Some dolphins are intelligent  
 $\exists x [Dolphin(x) \wedge Intelligent(x)]$
- Prove: Some who are intelligent cannot read  
 $\exists x [Intelligent(x) \wedge \neg Read(x)]$

# Example

1.  $\neg \text{Read}(x) \vee \text{Literate}(x)$
2.  $\neg \text{Dolphin}(y) \vee \neg \text{Literate}(y)$
3.  $\text{Dolphin(F1)}$
4.  $\text{Intelligent(F1)}$
5. (Negated Goal)  $\neg \text{Intelligent}(z) \vee \text{Read}(z)$

Proof:

6. [4, 5]  $\text{Read(F1)}$
7. [1, 6]  $\text{Literate(F1)}$
8. [2, 7]  $\neg \text{Dolphin(F1)}$
9. [3, 8] NULL

# Example

- Jack owns a dog  
 $\text{Dog}(D)$   
 $\text{Owns}(J, D)$
- Tuna is a cat  
 $\text{Cat}(T)$
- Every dog owner is an animal lover  
 $\forall x, y [\text{Dog}(x) \wedge \text{Owns}(y, x) \rightarrow \text{AnimalLover}(y)]$
- No animal lover kills an animal  
 $\forall x, z [\neg \text{AnimalLover}(z) \wedge \text{Animal}(w) \wedge \text{Kill}(z, w)]$
- Either Jack or Curiosity killed the cat who is called Tuna  
 $\text{Kill}(J, T) \vee \text{Kill}(C, T)$
- Cats are animals  
 $\forall u [\text{Cat}(u) \rightarrow \text{Animal}(u)]$
- Prove: Curiosity killed the cat  
 $\text{Kill}(C, T)$

# Example

1. Dog(D)
2. Owns(J, D)
3. Cat(T)
4.  $\neg \text{Dog}(x) \vee \neg \text{Owns}(y, x) \vee \text{AnimalLover}(y)$
5.  $\neg \text{AnimalLover}(z) \vee \neg \text{Animal}(w) \vee \neg \text{Kill}(z, w)$
6. Kill(J, T)  $\vee$  Kill(C, T)
7.  $\neg \text{Cat}(u) \vee \text{Animal}(u)$
8. (Negated Goal)  $\neg \text{Kill}(C, T)$
9. [6, 8] Kill(J, T)
10. [5, 9]  $\neg \text{AnimalLover}(J) \vee \neg \text{Animal}(T)$
11. [3, 7] Animal(T)
12. [10, 11]  $\neg \text{AnimalLover}(J)$
13. [1, 2, 4] AnimalLover(J)
14. [12, 13] NULL

# Example

1. Dog(D)
2. Owns(J, D)
3. Cat(T)
4.  $\neg \text{Dog}(x) \vee \neg \text{Owns}(y, x) \vee \text{AnimalLover}(y)$
5.  $\neg \text{AnimalLover}(z) \vee \neg \text{Animal}(w) \vee \neg \text{Kill}(z, w)$
6. Kill(J, T)  $\vee$  Kill(C, T)
7.  $\neg \text{Cat}(u) \vee \text{Animal}(u)$
8. (Negated Goal)  $\neg \text{Kill}(C, T)$
9. [6, 8] Kill(J, T)
10. [5, 9]  $\neg \text{AnimalLover}(J) \vee \neg \text{Animal}(T)$
11. [3, 7] Animal(T)
12. [10, 11]  $\neg \text{AnimalLover}(J)$
13. [1, 2, 4] AnimalLover(J)
14. [12, 13] NULL

# Resolution Strategies

- Marcus was a man.
    - Man(Marcus)
  - Marcus was a Pompeian.
    - Pompeian(Marcus)
  - All Pompeians were Romans.
    - FORALL x [Pompeian(x) -> Roman(x)]
  - Caesar was a ruler.
    - Ruler(Caesar)
  - All Romans were either loyal to Caesar or hated him.
    - FORALL x [Roman(x) -> Loyalto(x, Caesar) v Hated(x, Caesar)]
  - Everyone is loyal to someone.
    - FORALL x EXISTS y [Loyalto(x, y)]
  - Men only try to assassinate rulers they are not loyal to.
    - FORALL x, y [Man(x) & Ruler(y) & Tryassassinate(x,y) -> ~Loyalto(x,y)]
  - Marcus tried to assassinate Caesar.
    - Tryassassinate(Marcus, Caesar)
  - Goal: Marcus hated Caesar.
- 
- 1. Man(Marcus)
  - 2. Pompeian(Marcus)
  - 3. -Pompeian(x1) v Roman(x1)
  - 4. Ruler(Caesar)
  - 5. -Roman(x2) v Loyalto(x2, Caesar) v Hate(x2, Caesar)
  - 6. Loyalto(x3, f1(x3))
  - 7. -Man(x4) v -Ruler(y1) v - Tryassassinate(x4, y1) v - Loyalto(x4, y1)
  - 8. Tryassassinate(Marcus, Caesar)
- 
- Negated Goal:
- 9. -Hate(Marcus, Caesar)

# Resolution Strategies

- Marcus was a man.
    - Man(Marcus)
  - Marcus was a Pompeian.
    - Pompeian(Marcus)
  - All Pompeians were Romans.
    - FORALL x [Pompeian(x) -> Roman(x)]
  - Caesar was a ruler.
    - Ruler(Caesar)
  - All Romans were either loyal to Caesar or hated him.
    - FORALL x [Roman(x) -> Loyalto(x, Caesar) v Hated(x, Caesar)]
  - Everyone is loyal to someone.
    - FORALL x EXISTS y [Loyalto(x, y)]
  - Men only try to assassinate rulers they are not loyal to.
    - FORALL x, y [Man(x) & Ruler(y) & Tryassassinate(x,y) -> ~Loyalto(x,y)]
  - Marcus tried to assassinate Caesar.
    - Tryassassinate(Marcus, Caesar)
  - Goal: Marcus hated Caesar.
- 
- 1. Man(Marcus)
  - 2. Pompeian(Marcus)
  - 3. -Pompeian(x1) v Roman(x1)
  - 4. Ruler(Caesar)
  - 5. -Roman(x2) v Loyalto(x2, Caesar) v Hate(x2, Caesar)
  - 6. Loyalto(x3, f1(x3))
  - 7. -Man(x4) v -Ruler(y1) v - Tryassassinate(x4, y1) v - Loyalto(x4, y1)
  - 8. Tryassassinate(Marcus, Caesar)
- 
- Negated Goal:
- 9. -Hate(Marcus, Caesar)

# Resolution Strategies

1. Man(Marcus)
  2. Pompeian(Marcus)
  3. -Pompeian(x1) v Roman(x1)
  4. Ruler(Caesar)
  5. -Roman(x2) v Loyalto(x2, Caesar) v Hate(x2, Caesar)
  6. Loyalto(x3, f1(x3))
  7. -Man(x4) v -Ruler(y1) v -Tryassassinate(x4, y1) v -Loyalto(x4, y1)
  8. Tryassassinate(Marcus, Caesar)
  - Negated Goal:
  - Hate(Marcus, Caesar)
- Proof:
    10. [5&9] -Roman(Marcus) v Loyalto(Marcus, Caesar) ((x2 Marcus))
    11. [3&10] -Pompeian(Marcus) v Loyalto(Marcus, Caesar) ((x1 Marcus))
    12. [2&11] Loyalto(Marcus, Caesar)
    13. [7&12] -Man(Marcus) v -Ruler(Caesar) v -Tryassassinate(Marcus, Caesar) ((x4 Marcus) (y1 Caesar))
    14. [8&13] -Man(Marcus) v -Ruler(Caesar)
    15. [1&14] -Ruler(Caesar)
    16. [4&15] []
- Q.E.D.

# Resolution Properties

- Resolution is **complete** in the following sense:  
if a formula  $g$  follows from a set of formulas  $S$ ,  
then there is a sequence of resolutions of  
clauses in  $S \cup \neg g$  that terminates in the empty  
clause.

# Set Of Support (Depth First)

- **Rationale:** Conclusion should always play major role in proof
- **Method:** Give priority to resolvents derived from set of support (clauses which are part of  $-G$  or are resolvents with a parent in the set of support)
- **Example:** Resolve clauses (5&9), then (3&10), etc., the same way we did last time

# Resolution Strategies

1. Man(Marcus)
2. Pompeian(Marcus)
3. -Pompeian(x1) v Roman(x1)
4. Ruler(Caesar)
5. -Roman(x2) v Loyalto(x2, Caesar) v Hate(x2, Caesar)
6. Loyalto(x3, f1(x3))
7. -Man(x4) v -Ruler(y1) v -Tryassassinate(x4, y1) v -Loyalto(x4, y1)
8. Tryassassinate(Marcus, Caesar)

Negated Goal:

9. -Hate(Marcus, Caesar)

# Linear Format

- **Rationale:** Gives some direction to the search [Anderson and Bledsoe] prove that any provable theorem in predicate calculus can be proven with this strategy
- **Method:** Use most recent resolvent as a parent (the question of which two are resolved first is open)
- **Example:** Resolve clauses (1&7), then (4&NEW), then (8&NEW), then (5&NEW), etc.

# Unit Resolution

- **Rationale:**
  - Want to derive [] (0 literals), therefore, we want to make smaller and smaller resolvents.
  - Suppose  $c_1$  has 4 literals,  $c_2$  has 7 literals,  $R$  will have 9 literals!
- **Method:**
  - Use unit clause as a parent
  - $R = \# \text{literals}(c_1) + \# \text{literals}(c_2) - 2$
  - If  $c_1$  is a unit,  $R = \# \text{literals}(c_2) - 1$  (getting smaller)
- **Variation:**
  - Unit Preference
  - Use unit if available, otherwise look for next smaller clause size
- **Example:**
  - Resolve clauses (1&7), then (4&10), then (8&11), then (5&9), then (2&3), etc.

# Subsumption

- Keep knowledge base small
- If one sentence subsumed by another, remove it (the subsumed sentence).
- $P(A)$  subsumed by  $P(x)$ 
  - can remove  $P(A)$
- $P(A) \vee Q(B)$  subsumed by  $P(A)$ 
  - Can remove  $P(A) \vee Q(B)$

# Green's Trick

- We have shown how resolution can prove a specific theorem.
  - Named after Cordell Green, who used logic for software engineering applications.
  - We can also use resolution to extract answers (perform problem solving).
- Procedure:
  - The question to be answered, or the goal, must be an existentially quantified statement. It is this variable that we will derive.
  - Converted negated goal to clausal form.
  - Form a disjunction between the positive goal clause and the negated goal clause, and add this disjunction to the database.
  - Apply resolution as usual. When the positive goal clause is resolved, the variable will be instantiated with the answer

# Example

- Sally is studying with Morton. Morton is in the student union information office. If any person is studying with another person who is at a particular place, the first person is also at that place. If someone is at a particular place, then he or she can be reached on the telephone at the number for that place.
  - What is the number where Sally can be reached?
  - $Sw(x, y)$ :  $x$  is studying with  $y$
  - $A(x, y)$ :  $x$  is at place  $y$
  - $R(x, y)$ :  $x$  can be reached (by telephone) at number  $y$
  - $Ph(x)$ : the telephone number for place  $x$
- We want to find a sequence that will provide an answer to the problem. We therefore represent the question as a statement that the solution EXISTS.
- The question to be answered, or the goal, must be an existentially quantified statement. It is this variable that we will derive.
- The negated form will be eliminated (theorem proving), and the positive form will contain the answer.
- **FOPC statements**
  - $Sw(\text{Sally}, \text{Morton})$
  - $A(\text{Morton}, \text{Union})$
  - Forall  $x, y, z [Sw(x, y) \wedge A(y, z) \rightarrow A(x, z)]$
  - Forall  $x, y [A(x, y) \rightarrow R(x, Ph(y))]$
  - GOAL: exists  $x R(\text{Sally}, x)$

# Example

- Who is Lulu older than?
- Prove that there is an  $x$  such that Lulu is older than  $x$
- FOPC: EXISTS  $x$  Older(Lulu,  $x$ )  
(Negated Goal):  $\sim$ Older(Lulu,  $x$ )
- A successful proof will yield  $x$ /Fifi

# Example

1. Mother(Lulu, Fifi)
2. Alive(Lulu)
3.  $\neg\text{Mother}(x,y) \vee \text{Parent}(x,y)$
4.  $\neg\text{Parent}(x,y) \vee \neg\text{Alive}(x) \vee \text{Older}(x,y)$
5. Query: Who is Lulu older than?
  - There exists someone that Lulu is older than
  - We want to find out who
  - Exists  $x \text{ Older}(\text{Lulu},x)$
  - Negate goal and create disjunction with “opposite”
  - $\text{Older}(\text{Lulu},x) \vee \neg\text{Older}(\text{Lulu},x)$

# Example

1. Mother(Lulu, Fifi)
2. Alive(Lulu)
3.  $\neg\text{Mother}(x_3,y_3) \vee \text{Parent}(x_3,y_3)$
4.  $\neg\text{Parent}(x_4,y_4) \vee \neg\text{Alive}(x_4) \vee \text{Older}(x_4,y_4)$
5. Older(Lulu,x)  $\vee \neg\text{Older}(Lulu,x)$
6. [4,5]  $\neg\text{Parent}(Lulu,y_6) \vee \neg\text{Alive}(Lulu) \vee \text{Older}(Lulu,y_6)$
7. [2,6]  $\neg\text{Parent}(Lulu,y_7) \vee \text{Older}(Lulu,y_7)$
8. [3,7]  $\neg\text{Mother}(Lulu,y_8) \vee \text{Older}(Lulu,y_8)$
9. [1,8] Older(Lulu,Fifi)

# Example

- What is older than what?
- FOPC: EXISTS x,y Older(x, y)  
(Negated Goal)  $\sim$ Older(x,y)
- A successful proof will yield x/Lulu, y/Fifi

# Example

1. Mother(Lulu, Fifi)
2. Alive(Lulu)
3.  $\neg\text{Mother}(x_3,y_3) \vee \text{Parent}(x_3,y_3)$
4.  $\neg\text{Parent}(x_4,y_4) \vee \neg\text{Alive}(x_4) \vee \text{Older}(x_4,y_4)$
5. Older(x,y)  $\vee \neg\text{Older}(x,y)$
6. [4,5]  $\neg\text{Parent}(x_6,y_6) \vee \neg\text{Alive}(x_6) \vee \text{Older}(x_6,y_6)$
7. [2,6]  $\neg\text{Parent}(\text{Lulu},y_7) \vee \text{Older}(\text{Lulu},y_7)$
8. [3,7]  $\neg\text{Mother}(\text{Lulu},y_8) \vee \text{Older}(\text{Lulu},y_8)$
9. [1,8] Older(Lulu,Fifi)

# OTTER

- `Organized Techniques for Theorem proving and Effective Research"
- OTTER was developed at Argonne, written in C, described in book ``Automated Reasoning"
- [Otter Home Page](#)
- Employs binary resolution, hyperresolution, binary paramodulation (use equality substitutions), set of support resolution, and unit-resulting resolution
- Takes FOPC statements, converts to clausal form, applies resolution

# OTTER Features

- Can use autonomous mode, where OTTER selects resolution strategies
- Answer literals
  - Can extract answer to query like we will in class
  - Our example has an answer literal
  - If no answer literal, end of proof yields “.” clause
- Forward and backward subsumption
  - If clause  $P(x)$  exists in database and clause  $P(x) \vee Q(x)$  exists in database, we know  $P(x)$  SUBSUMES  $P(x) \vee Q(x)$ , so delete  $P(x) \vee Q(x)$  from database.
- Factoring
  - Unifying 2 literals of same sign within a clause
- Weighting
  - Weight terms to affect ordering in the proof

# OTTER Format

- Set parameters of the system
- Give general formulas
- Give set of support (facts and formulas used in every step of proof)
- ``%" is used at the beginning of a comment line.
- OPTIONS
  - set(ur\_res) % or other inference mechanisms, if not auto
  - assign(max\_mem, 1500) % use at most 1.5 megabytes
  - assign(max\_seconds, 1800) % time limit of 30 CPU minutes
- FORMULA LIST
  - formula\_list(?). % "?" is either "sos" or "usable".
  - ... - facts and rules
  - end\_of\_list.
- SYNTAX
  - All clauses and other statements end with ":"
  - & = and
  - | = or
  - (all x all y ())
  - (exists x exists y ())
  - - = not

# OTTER Format

- OTTER is not case sensitive. If a symbol is quantified, it is a variable. Otherwise, it is a predicate, function or constant.
- Put the negated goal in the sos list.
- OTTER will keep going until sos is empty
  - Use clause in sos to make inferences (with clauses in usable list)
  - Move used clause to usable
- For answer extraction use \$answer(literal).
- OTTER output contains
  - List of clauses from original database
  - List of new clauses with parent and resolution strategy
  - *ur, 4, 1 above(B, A).*
  - If contradiction found

----- PROOF -----

- list of options used
- stats
- If no contradiction found and no clauses left to resolve “sos empty”

# OTTER Example

Input:

```
set(ur_res).
set(hyper_res).
```

```
formula_list(usable).
all x all y all z (sw(x,y) & a(y,z) -> a(x,z)).
end_of_list.
```

```
formula_list(sos).
all x all y (a(x,y) -> r(x, ph(y))).
sw(Sally, Morton).
a(Morton, StudentUnion).
-(exists x (r(Sally, x))).
end_of_list.
```

Output:

----- Otter 3.0.4, August 1995 -----

The job was started by ...

```
set(ur_res).
set(hyper_res).
```

-----> usable clausifies to:

```
list(usable).
1 [] -sw(x,y) | -a(y,z) | a(x,z).
end_of_list.
```

-----> sos clausifies to:

```
list(sos).
2 [] -a(x,y) | r(x,ph(y)).
3 [] sw(Sally,Morton).
4 [] a(Morton,StudentUnion).
5 [] -r(Sally,x).
end_of_list.
```

===== end of input processing =====

# OTTER Example

===== start of search =====

given clause #1: (wt=3) 3 [] sw(Sally,Morton).

given clause #2: (wt=3) 4 [] a(Morton,StudentUnion).

\*\* KEPT (pick-wt=3): 6 [hyper,4,1,3] a(Sally,StudentUnion).

given clause #3: (wt=3) 5 [] -r(Sally,x).

given clause #4: (wt=3) 6 [hyper,4,1,3] a(Sally,StudentUnion).

given clause #5: (wt=7) 2 [] -a(x,y) | r(x,ph(y)).

\*\* KEPT (pick-wt=4): 7 [hyper,2,6] r(Sally,ph(StudentUnion)).

---> UNIT CONFLICT at 0.08 sec ---> 8 [binary,7.1,5.1] \$F.

Length of proof is 2. Level of proof is 2.

----- PROOF -----

1 [] -sw(x,y) | -a(y,z) | a(x,z).

2 [] -a(x,y) | r(x,ph(y)).

3 [] sw(Sally,Morton).

4 [] a(Morton,StudentUnion).

5 [] -r(Sally,x).

6 [hyper,4,1,3] a(Sally,StudentUnion).

7 [hyper,2,6] r(Sally,ph(StudentUnion)).

8 [binary,7.1,5.1] \$F.

----- end of proof -----

Search stopped by max\_proofs option.

# OTTER Input Using \$Answer

Input:

```
set(ur_res).
set(hyper_res).
```

```
formula_list(usable).
all x all y all z (sw(x,y) & a(y,z) -> a(x,z)).
end_of_list.
```

```
formula_list(sos).
all x all y (a(x,y) -> r(x, ph(y))).
sw(Sally, Morton).
a(Morton, StudentUnion).
-(exists x (-$answer(r(Sally,x)) & r(Sally, x))).
end_of_list.
```

Output:

----- Otter 3.0.4, August 1995 -----  
The job was started by ...

-----> usable clausifies to:

```
list(usable).
1 [] -sw(x,y) | -a(y,z) | a(x,z).
end_of_list.
```

-----> sos clausifies to:

```
list(sos).
2 [] -a(x,y) | r(x,ph(y)).
3 [] sw(Sally,Morton).
4 [] a(Morton,StudentUnion).
5 [] $answer(r(Sally,x)) | -r(Sally,x).
end_of_list.
```

===== end of input processing =====

# OTTER Output Using \$Answer

===== start of search =====

given clause #1: (wt=3) 3 [] sw(Sally,Morton).

given clause #2: (wt=3) 4 [] a(Morton,StudentUnion).

\*\* KEPT (pick-wt=3): 6 [hyper,4,1,3] a(Sally,StudentUnion).

given clause #3: (wt=3) 5 [] \$answer(r(Sally,x)) | -r(Sally,x).

given clause #4: (wt=3) 6 [hyper,4,1,3] a(Sally,StudentUnion).

given clause #5: (wt=7) 2 [] -a(x,y) | r(x,ph(y)).

\*\* KEPT (pick-wt=4): 7 [hyper,2,6] r(Sally,ph(StudentUnion)).

----> UNIT CONFLICT at 0.08 sec ---->

8 [binary,7.1,5.1] \$answer(r(Sally,ph(StudentUnion))). ----- PROOF -----

Length of proof is 2. Level of proof is 2.

1 [] -sw(x,y) | -a(y,z) | a(x,z).

2 [] -a(x,y) | r(x,ph(y)).

3 [] sw(Sally,Morton).

4 [] a(Morton,StudentUnion).

5 [] -r(Sally,x).

6 [hyper,4,1,3] a(Sally,StudentUnion).

7 [hyper,2,6] r(Sally,ph(StudentUnion)).

8 [binary,7.1,5.1]

\$answer(r(Sally,ph(StudentUnion))).

----- end of proof -----

Search stopped by max\_proofs option.

# CptS 440 / 540

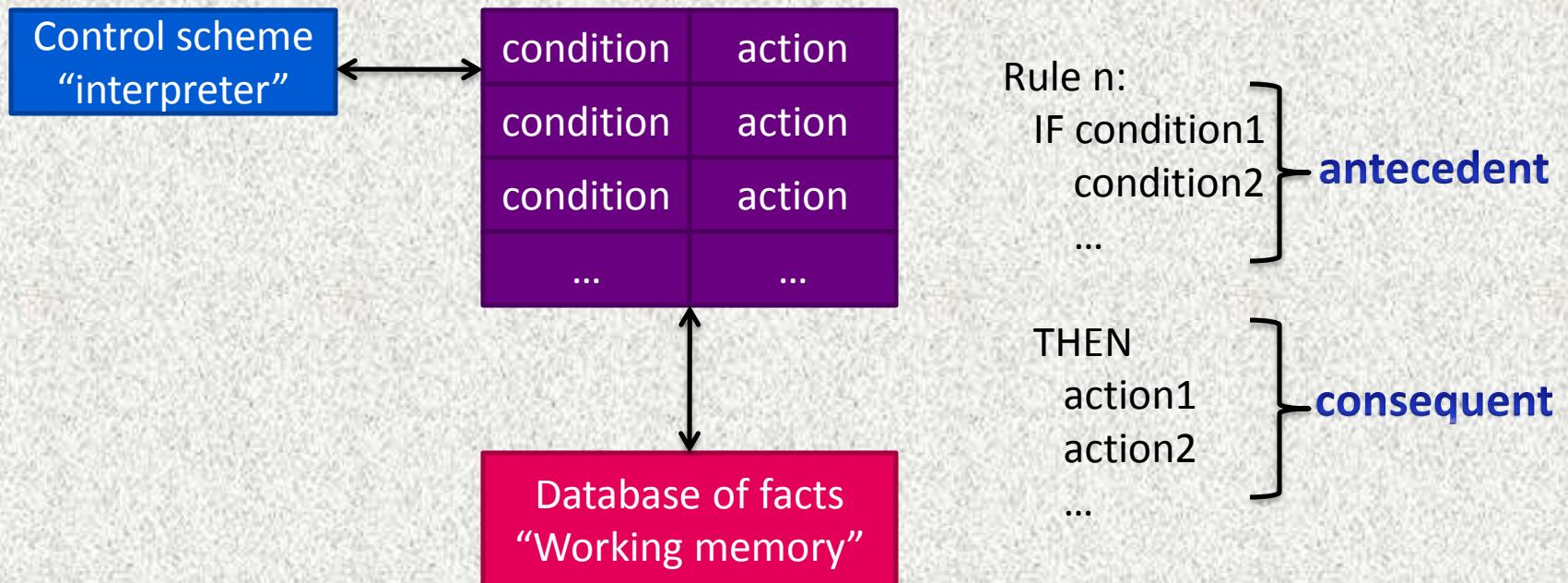
# Artificial Intelligence

Expert Systems and Planning

# Expert Systems

- ES perform tasks for specific domains that require human expertise
  - Medical diagnosis, fault diagnosis, status monitoring, data interpretation, computer configuration, etc.
- ES solve problems using domain-specific knowledge
- Domain knowledge is acquired by interviewing human experts
- ES cannot operate in situations requiring common sense

# Rule-Based Systems



- When one part of the IF portion matches a fact in working memory, the antecedent is **SATISFIED**. When all antecedents are satisfied, the rule is **TRIGGERED**. When the consequent of a rule is performed, the rule is **FIRED**.

# Three Phases

1. Match phase
  - Match left side of rules (antecedents) with facts in working memory
  - Unification
2. Conflict resolution
  - Of all the rules that are triggered, decide which rule to fire.
  - Some strategies include:
    - Do not duplicate rule with same instantiated arguments twice
    - Prefer rules that refer to recently created WM elements
    - Prefer rules that are more specific (antecedents are more constraining)
    - Prefer  $\text{Mammal}(x) \& \text{Human}(x) \rightarrow \text{add Legs}(x,2)$  over  $\text{Mammal}(x) \rightarrow \text{add Legs}(x,4)$
  - If rules are ranked, fire according to ranking
3. Act phase
  - Add or delete facts to WM as specified by rule consequent

# Expert Systems at a Glance

- ES are one of AI's early showpieces
- ES use rule-based systems along with additional capabilities
- Components of an ES include:
  - Evidence gathering
  - Control of inference
  - Uncertainty reasoning
  - Explanation generation and tutoring
  - User interface
  - Validation
  - Social issues

# Some Famous Early Expert Systems

- Dendral (Stanford, Ed Feigenbaum, 1965) - organic chemical analysis
- Macsyma (MIT, 1965) - symbolic math problems
- Mycin (Stanford, 1972) - diagnose blood diseases
- Prospector (SRI, 1972) - mineral exploration
- Caduceus (1975) - internal medicine
- Xcon and R1 (1980, 1982) - computer system configuration (for DEC)
- Harpy - document retrieval
- Hearsay - Speech understanding
- Max (Nynex, 1992) - telephone network troubleshooting  
bacterial infection diagnosis

# Examples

- [Expert System Applet](#)
- [Aaron, Artist Expert System](#)
- Some expert system shells
  - [CLIPS](#)
  - KEE
  - [JESS](#)

# Mycin Example

## Rules

PREMISE (\$AND (SAME CNTXT GRAM GRAMNEG)  
          (SAME CNTXT MORH ROD)  
          (SAME CNTXT AIR AEROBIC))

ACTION: (CONCLUDE CNTXT CLASS ENTEROBACTERIACEAE .8)

If the stain of the organism is gramneg, and  
the morphology of the organism is rod, and  
the aerobiticity of the organism is aerobic

Then there is strongly suggestive evidence (.8) that  
the class of organism is enterobacteriaceae

## Data

ORGANISM-1:

GRAM = (GRAMNEG 1.0)

MORP = (ROD .8) (COCCUS .2)

AIR = (AEROBIC .6) (FACUL .4)

# Forward Chaining and Backward Chaining

There are two main types of control schemes that are applied to rule-based systems.

Z1 If ?x has hair

    Then ?x is a mammal

Z2 If ?x gives milk

    Then ?x is a mammal

Z3 If ?x has feathers

    Then ?x is a bird

Z6 If ?x is a mammal

    ?x has pointed teeth

    ?x has claws

    ?x has forward-pointing eyes

    Then ?x is a carnivore

Z8 If ?x is a mammal

    ?x chews cud

    Then ?x is an ungulate

Z11 If ?x is an ungulate

    ?x has long legs

    ?x has long neck

    ?x has tawny color

    ?x has dark spots

    Then ?x is a giraffe

Database

---

F1) Stretch has hair

F2) Stretch chews cud

F3) Stretch has long legs

F4) Stretch has a long neck

F5) Stretch has tawny color

F6) Stretch has dark spots

# Forward Chaining

- Reason FORWARD from facts/rules to (hopefully) a needed goal
- Use modus ponens to generate new facts
- Rule antecedents are compared with facts from database
- If match, add consequents to database
- Repeat as long as needed
- Forward chaining is “data driven”

Match Z1 & F1

Add: Stretch is a mammal

Match Z8/1 & F7 (?x Stretch)

Match Z8/2 & F2

Add: Stretch is an ungulate

Match Z11/1 & F8 (?x Stretch)

Match Z11/2 & F3

Match Z11/3 & F4

Match Z11/4 & F5

Match Z11/5 & F6

Add: Stretch is a giraffe

Note: Z5/1, Z6/1, Z7/1 would be  
matched before Z8/1

# Backward Chaining

- Reasons BACKWARD from goal through rules to facts
- Use modus ponens
  - Start at goals
  - Match goals to consequents or facts
  - If match consequents, antecedents become new subgoals
- Repeat until
  - All subgoals are proven or
  - At least one subgoal cannot be proven
- Backward chaining is “goal driven”

# Backward Chaining Example

Goal 1: Stretch is a giraffe

Match: Goal 1 and Z11/C (does not match with any facts)

Subgoal 2: Stretch is an ungulate

Subgoal 3: Stretch has long legs

Subgoal 4: Stretch has long neck

Subgoal 5: Stretch has tawny color

Subgoal 6: Stretch has dark spots

Match: Subgoal 2 and Z8/C (does not match with any facts)

Subgoal 7: Stretch is a mammal

Subgoal 8: Stretch chews cud

Match: Subgoal 7 and Z1/C (does not match with any facts)

Subgoal 9: Stretch has hair

Match: Subgoal 9 and F1

Subgoals 9, 7, met

Match: Subgoal 8 and F2

Subgoals 8, 2 met

Match: Subgoal 3 and F3

Subgoal 3 met

Match: Subgoal 4 and F4

Subgoal 4 met

Match: Subgoal 5 and F5

Subgoal 5 met

Match: Subgoal 6 and F6

Subgoal 6 met, Goal 1 met

# Forward Chaining vs. Backward Chaining

- High fan out - use backward chaining

1) Human(Albert)  
2) Human(Alfred)  
3) Human(Barry)  
4) Human(Charlie)  
...  
50) Human(Highlander)  
...  
100) Human(Shaun)  
...  
500) Human(Zelda)  
501) Human(x) -> Mortal(x)  
502) Mortal(x) -> CanDie(x)  
Can we kill Shaun Connery?

- FC  
503) Mortal(Albert)  
504) Moral(Alfred)  
...  
1003) CanDie(Albert)  
1004) CanDie(Alfred)  
...  
1100) CanDie(Shaun)

- BC
  - Prove: CanDie(Shaun)
    - Match: Goal and 502/C
    - Prove: Mortal(Shaun)
      - Match: Mortal(Shaun) and 501/C
      - Prove: Human(Shaun)
        - » Match: Human(Shaun) and 100
    - Done!

# Forward Chaining vs. Backward Chaining

- Forward chaining may generate a lot of useless facts

If ?x has feathers  
Then ?x can be used as a pen

If ?x has feathers  
Than ?x can fly

If ?x has feathers  
Then ?x is a bird

...

1 condition – many actions

Use BC!

# Forward Chaining vs. Backward Chaining

- High fan in – use forward chaining
- Potential problem with BC
  - If many subgoals, each must be examined

If  $?x$  has feathers

$?x$  is brown

$?x$  sings @#!@#! Patterns

$?x$  sleeps on one foot

$?x$  makes good pot pie

$?x$  lives in tree

Then  $?x$  is bird

- Many conditions – 1 action
- If time is crucial and you only have 1 goal to prove, use BC
- If you have extra time and want to be prepared for future questions, use FC to generate all possible facts
- Or use bi-directional search

# Limitation of ES

- A lot of matches!
- May perform many matches for Rule 1, then have to perform same matches for Rule 2 (should share partial match information)
- May match first few antecedents, but fail on last If last antecedent added later, have to start again at beginning of rule (should save partial match information)
- One solution: RETE net (RETE stands for ``network'')
  - Contains:
    - alpha nodes (one for each antecedent)
    - beta nodes (combination of matches)
    - terminal nodes (one for each rule)
  - Explicitly store shared and partial match information
  - Machine rules and facts are numerous, matching is laborious
  - Much redundant work in match phase
  - RETE representation eliminates some redundant work

# RETE Network

Rule 1

if ?x is small  
?y is larger than ?x  
?y is a carnivore  
then ?y eats ?x (ADD)

Rule 2

if ?x is small  
?x lives on land  
?y is strong  
?y is a carnivore  
then ?y eats ?x (ADD)

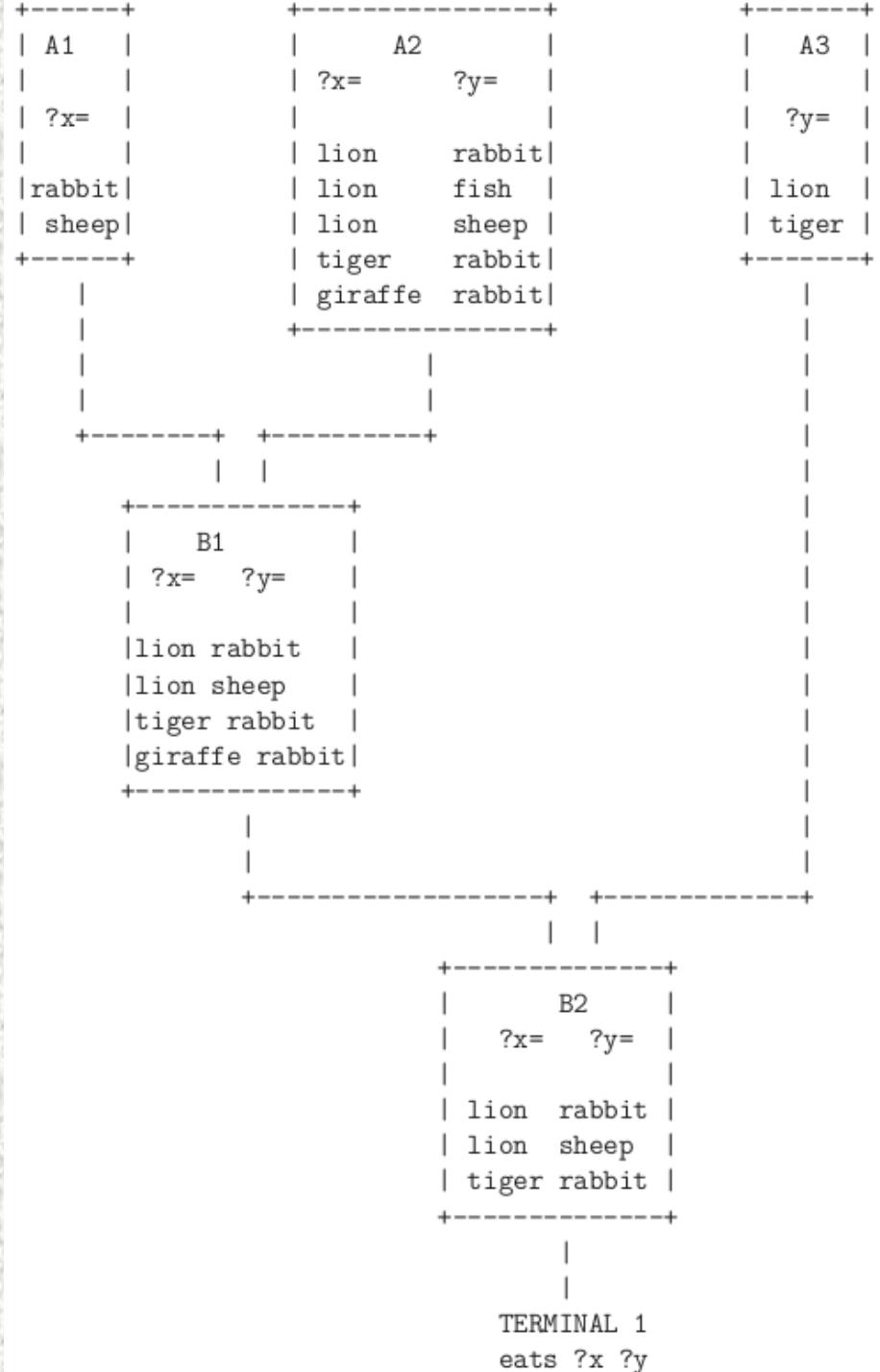
Working Memory:

rabbit is small  
sheep is small  
bird is small  
mouse is small  
lion is a carnivore  
tiger is a carnivore  
lion is larger than rabbit  
lion is larger than fish  
lion is larger than sheep  
tiger is larger than rabbit  
giraffe is larger than rabbit

# RETE Network

- For each rule
  - For each antecedent
    - Create alpha node
    - Store antecedent matches in this node
  - For each alpha node except the first
    - Create beta node
    - If this is the first beta node in the network
      - Then parents are alpha1 and alpha2
      - Else parents are previous beta and next unmatched alpha
    - Store partial matches in this node
  - Create terminal node for each rule
    - Parent is last beta for the rule
- To run
  - Distribute assertions to top of network
  - Match assertions with alpha nodes - pass bindings to beta nodes
  - For each beta node, JOIN by checking for consistent bindings
  - When an assertion (binding list) hits terminal node, change WM
  - For each WM change, add information to alpha node and propagate down
  - Do this for additions and deletions

# RETE Network



# Planning

- What is planning?
- Strategies required to achieve a goal or to solve a particular problem



# Need for Planning

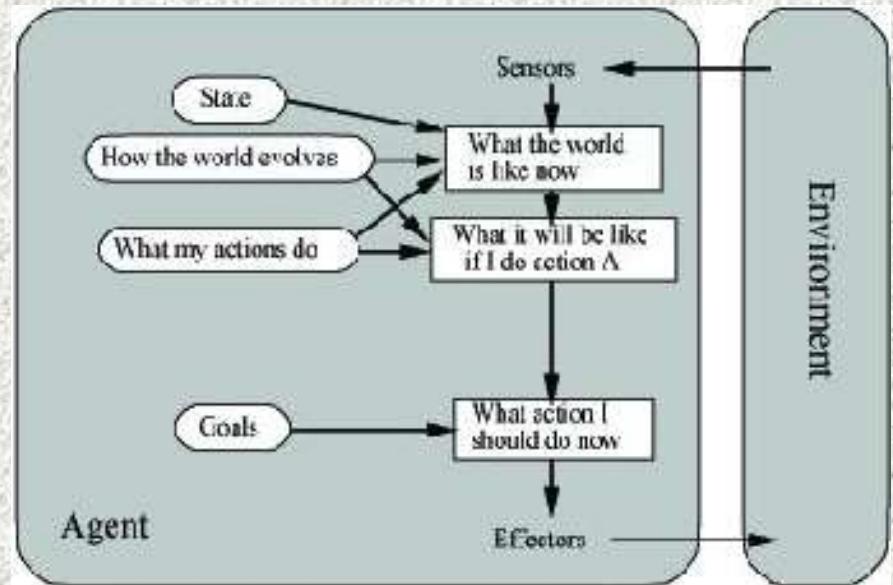
- If our goal is to create an autonomous, intelligent entity, we need to study planning.
- Logic, natural language, learning, vision, are all useful, and they would make nice human-driven tools.
- But what is the entity going to DO???
- It has to decide WHAT it wants to do, and HOW to do it.

# Environment Properties

- Fully observable vs. partially observable
- Deterministic vs. stochastic / strategic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Single agent vs. multiagent

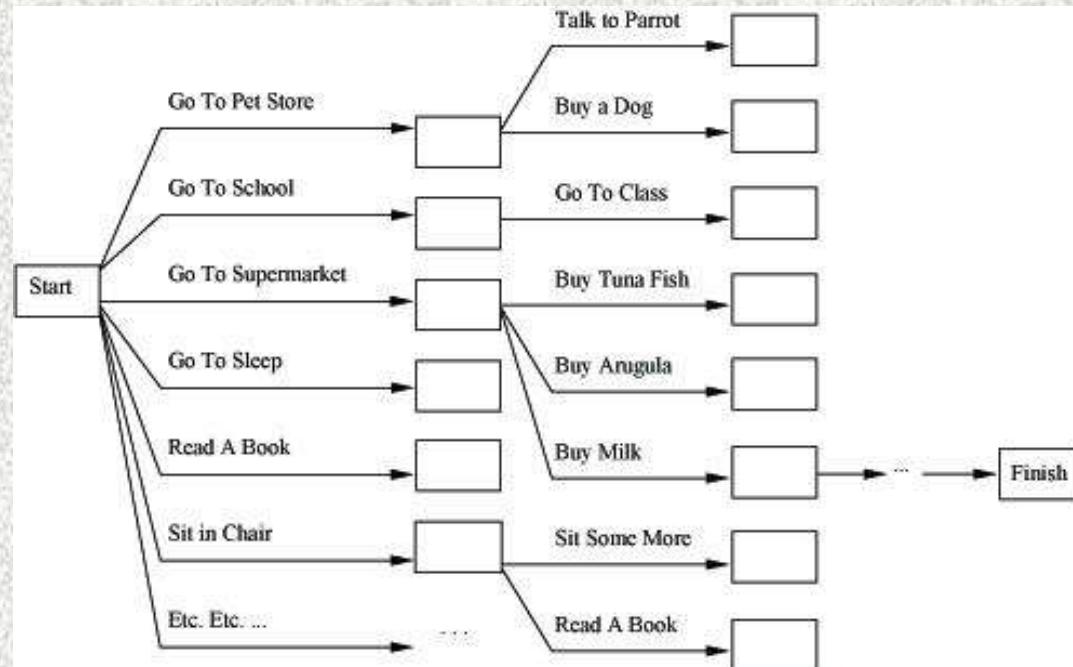
# Goal-Based Agents

- Goal reflects desires of agents
- May project actions to see if consistent with goals
- Takes time, world may change during reasoning



# Planning Search Space

- Suppose problem is “Get a quart of milk and two eggs (to make pancakes) and a variable-speed cordless drill (to fix house when done”).
  - Search through a space of states
  - Try each possible action
  - Thousands of actions, millions of states
  - The heuristic function directs search, but doesn't prune states



# Planning Key 1

- In planning we will modify our representation of the search space (states, goals, and actions) to allow more efficient reasoning.
- Instead of search through states from initial state to goal state, we can look for actions that seem relevant and are accessible.
- If goal includes  $\text{Have}(\text{Milk})$
- and  $\text{Buy}(x)$  achieves  $\text{Have}(x)$
- then agent should consider plan that includes  $\text{Buy}(\text{Milk})$ .
- Do not consider  $\text{Buy}(\text{WhippingCream})$  or  $\text{GoToSleep}$ .

# Planning Key 2

- The planner can add actions to the plan wherever they are needed, instead of always at the end (or beginning) of the plan.
- Agent may want to Buy(Milk), but doesn't know how to get there, how to buy it, what to do after that.
- Make “obvious” or “important” decisions first.

# Planning Key 3

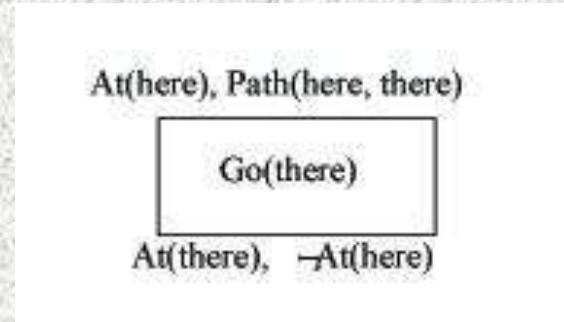
- Parts of the world are independent from other parts.
- We can solve one part of a conjunctive goal with one plan, another part of the goal with another plan.

# Planning Representation

- States are represented as a conjunction of instantiated literals (no functions)
  - $\text{At}(\text{Home}) \wedge \text{Have}(\text{Milk}) \wedge \text{Have}(\text{Bananas}) \wedge \text{Have}(\text{Drill}) \wedge \dots$
  - States can be incomplete - if don't specify, then negation is assumed.
  - If  $\text{Have}(\text{Unicorn})$  is not included, assume  $\neg \text{Have}(\text{Unicorn})$ . This is the **Closed-World Assumption**.
- Goals are described as conjunction of literals (possibly with variables).
  - $\text{At}(\text{Home}) \wedge \text{Have}(\text{Milk}) \wedge \text{Have}(\text{Bananas}) \wedge \text{Have}(\text{Drill})$
  - $\text{At}(x) \wedge \text{Sells}(x, \text{Milk})$  if goal is to be at store that sells milk.
- Variables must be existential (like theorem prover goal).
  - Here, we are not seeing if database entails goal.
  - We are trying to transform state to one that includes goal.

# Planning Representation

- STRIPS Operators
  - Precondition
    - Conjunction of atoms (positive literals) that must be true to apply operator
  - Effect
    - Conjunction of literals that describe how the situation changes when operator is applied
- Example
  - OP Go(?there)
  - Precondition: At(?here) ^ Path(?here, ?there)
  - Effect: At(?there) ^ -At(?here)



# State Space Planner

- Operators can have variables - then we **unify** goals with facts.
- We call this type of planner a **situation space** planner, or **state space** planner, because nodes in search space represent states or situations.
- This type of planner completely solves one goal, then tacks on plan for next goal.
- “Progression planner” if search forward (A\*) This prunes options if high fan-in
- “Regression planner” if search backward (GPS) This prunes options if high fan-out

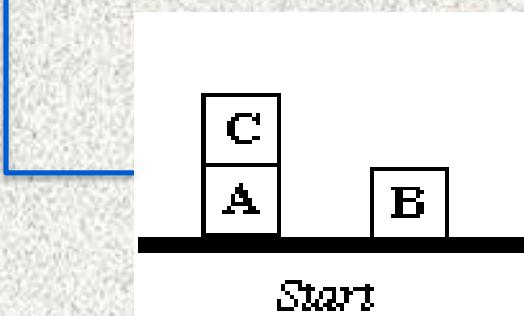
# Example State Space Planner

- General Problem Solver, [GPS](#)
- Uses Means-Ends Analysis

# Limitations

- Note what happens when we apply STRIPS planning to this problem.

On(C,A)  
On(A,Table)  
On(B,Table)  
Clear(C)  
Clear(B)



On(A,B)  
On(B,C)



|          |                       |
|----------|-----------------------|
| Operator | PutOn(X,Y)            |
| Prec:    | Clear(X)              |
|          | Y=Table v Clear(Y)    |
|          | On(X,Y)               |
| Add:     | On(X,Y)               |
|          | Clear(Z) (if On(X,Z)) |
| Delete:  | On(X,~)               |
|          | Clear(Y)              |

## Approach 1

- ON(A,B)
  - Try PutOn(C,Table) to clear A
  - PutOn(A,B) to achieve first goal
- ON(B,C)

To achieve this goal, B will be re-cleared, undoing first goal!

## Approach 2

- ON(B,C)
  - Try PutOn(B,C)
  - ON(A,C)

We are farther from this goal now than we were in the initial state!

A

C

B

# Sussman's Anomaly

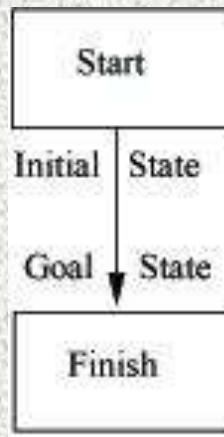
- This problem can be solved, but it cannot be attacked by first applying all the operators to achieve one goal, and then applying operators to achieve another goal.
- The problem is that we have forced an ORDERING on the operators. Sometimes steps for multiple goals need to be interleaved.
- **Partial-order planning** is a type of plan generation in which ordering is imposed on operators ONLY when it has to be imposed in order to achieve the goals.

# Plan Space Planning

- Node (state space): world state
- Node (plan space): partial plan
- Operator (state space): next step in plan sequence
- Operator (plan space)
  - Add a link from existing action to open condition (precondition that is not yet fulfilled)
  - Add a plan step to fulfill open condition
  - Order a step in the sequence
- Gradually move from incomplete/vague plans to complete plans.  
We call planners that use this type of approach **partial-order planners**
- Example: Have shoes on both feet.

# Partially-Ordered Plans

- The initial plan has only two steps (Start and Finish), 1 ordering (Start before Finish).
- A sock must be put on before a shoe.



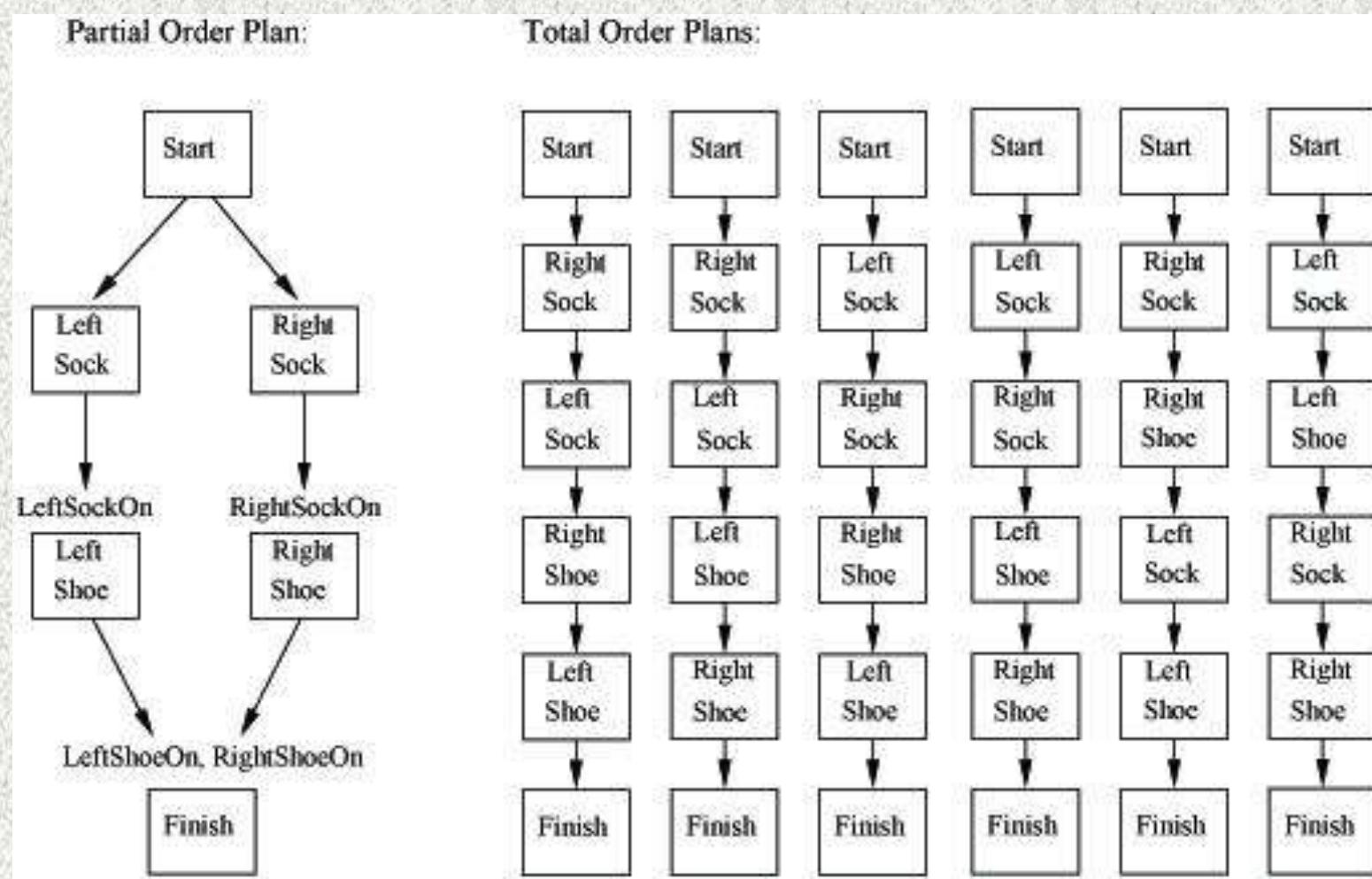
(a)



(b)

# Partially-Ordered Plans

- One partial-order plan can have many linearizations (total orderings), as seen in this example.

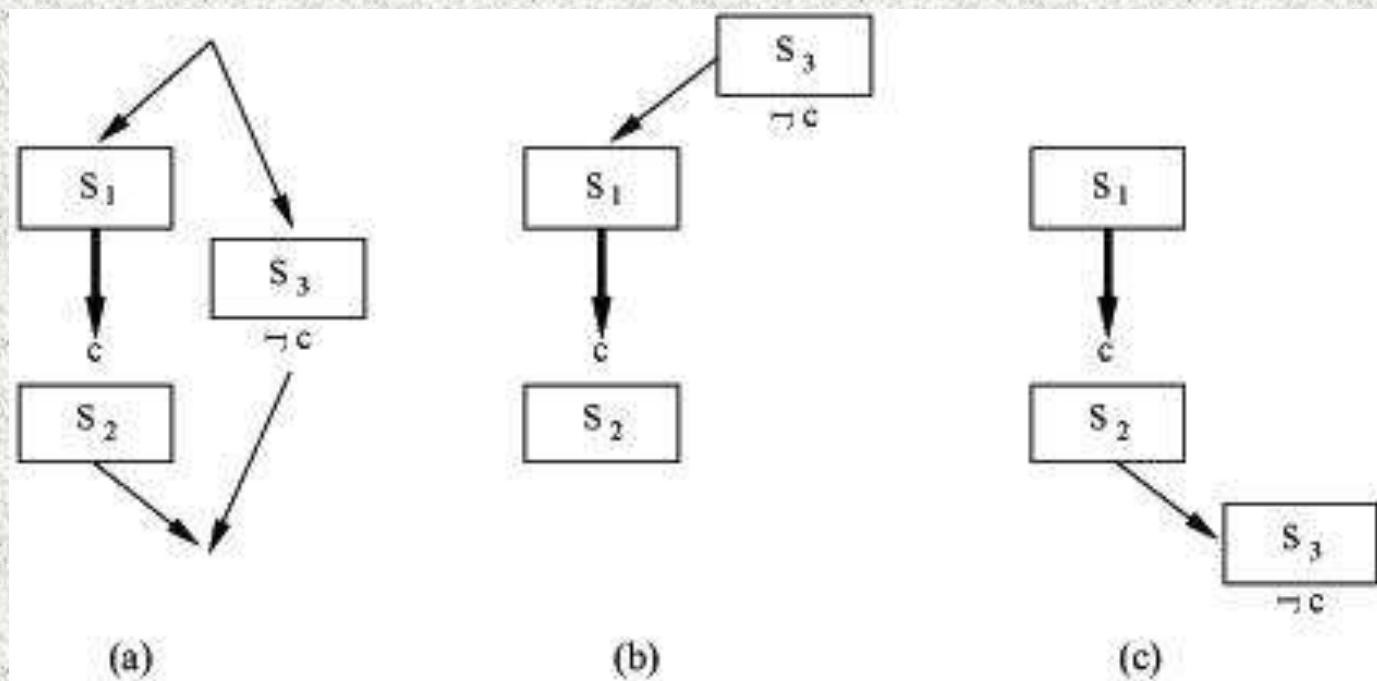


# Partially-Ordered Plans

- A plan in this scenario consists of the following components:
  1. Set of plan steps
  2. Set of ordering constraints (Step I must occur before Step j)
  3. Variable binding constraints ( $v = x$ )
  4. Causal links ( $S_i \xrightarrow{c} S_j$ ) which reads “Step i achieves condition c for Step j”
    - This is a **protection interval**. If Step i achieves condition c for Step j, make sure nothing removes c during this interval

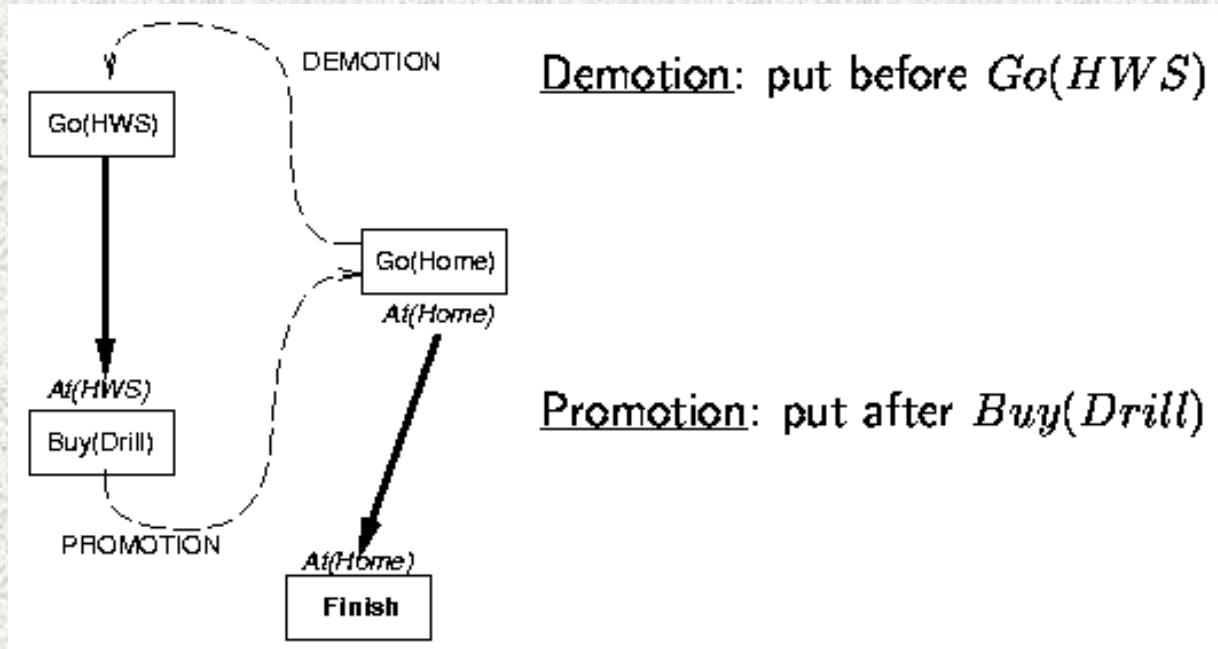
# Clobbering and Promotion / Demotion

- A clobberer is a potentially intervening step that destroys the condition achieved by a causal link
- We can prevent clobbering by ordering the clobberer before (**demotion**) the start of the link or after (**promotion**) the end of the link



# Example

- In this example, Go(Home) clobbers At(HWS)



# Example

- Here is the shopping problem initial incomplete plan



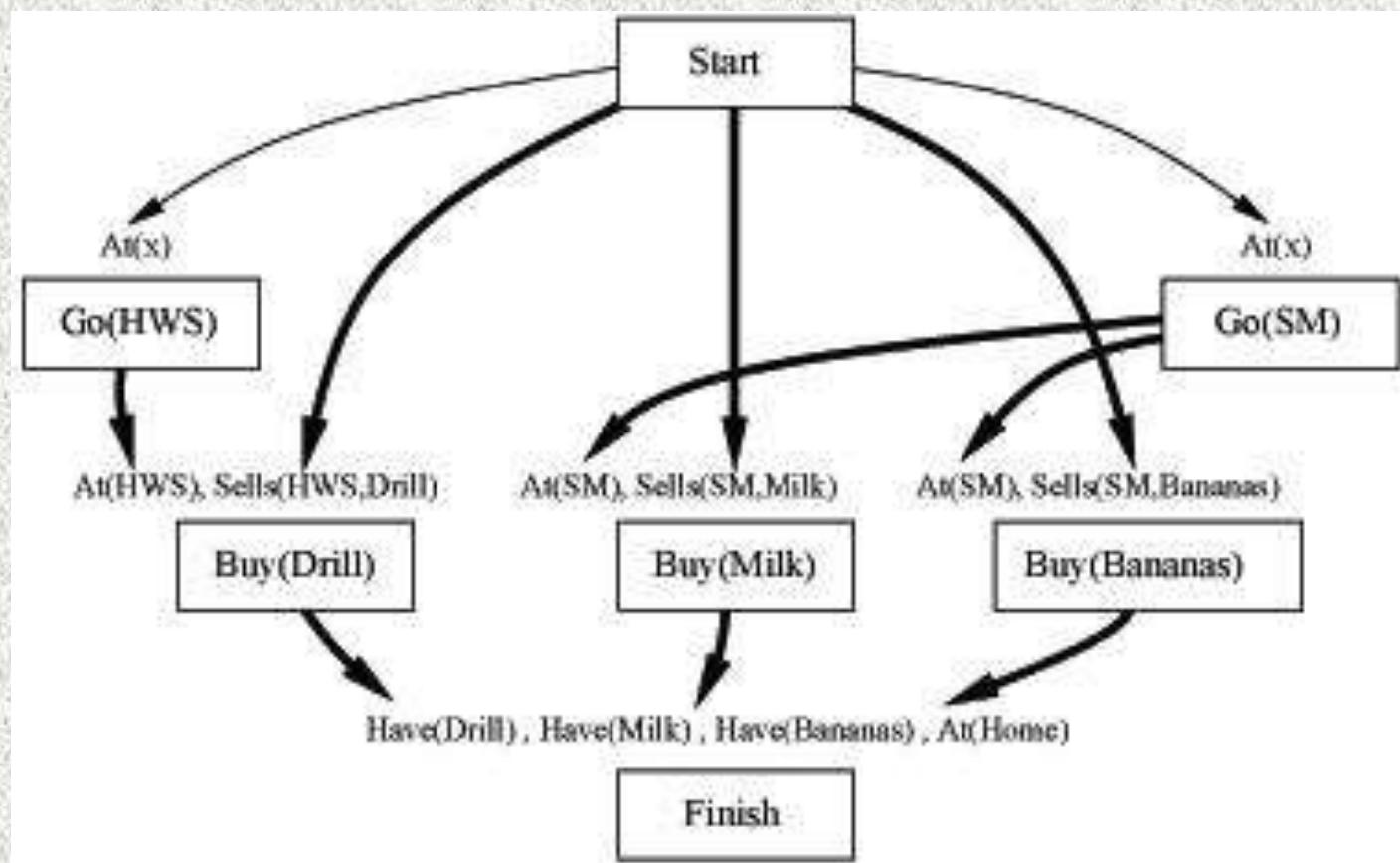
# Example

- The top plan achieves three of the four Finish preconditions, the heavy arrows show causal links. The bottom plan refines the top plan by adding causal links to achieve the Sells preconditions of the Buy steps.



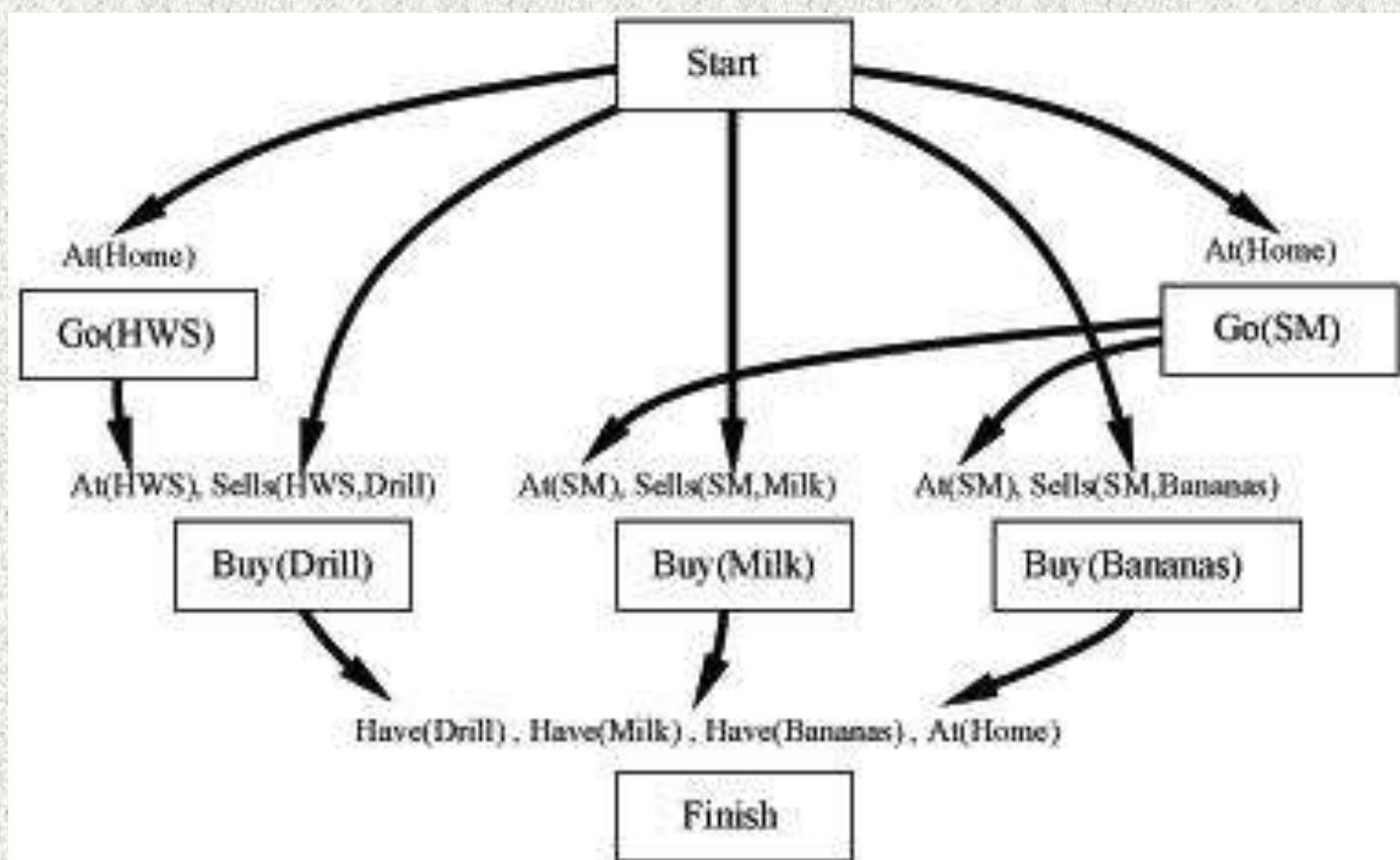
# Example

- This partial plan achieves the At preconditions of the three Buy actions.



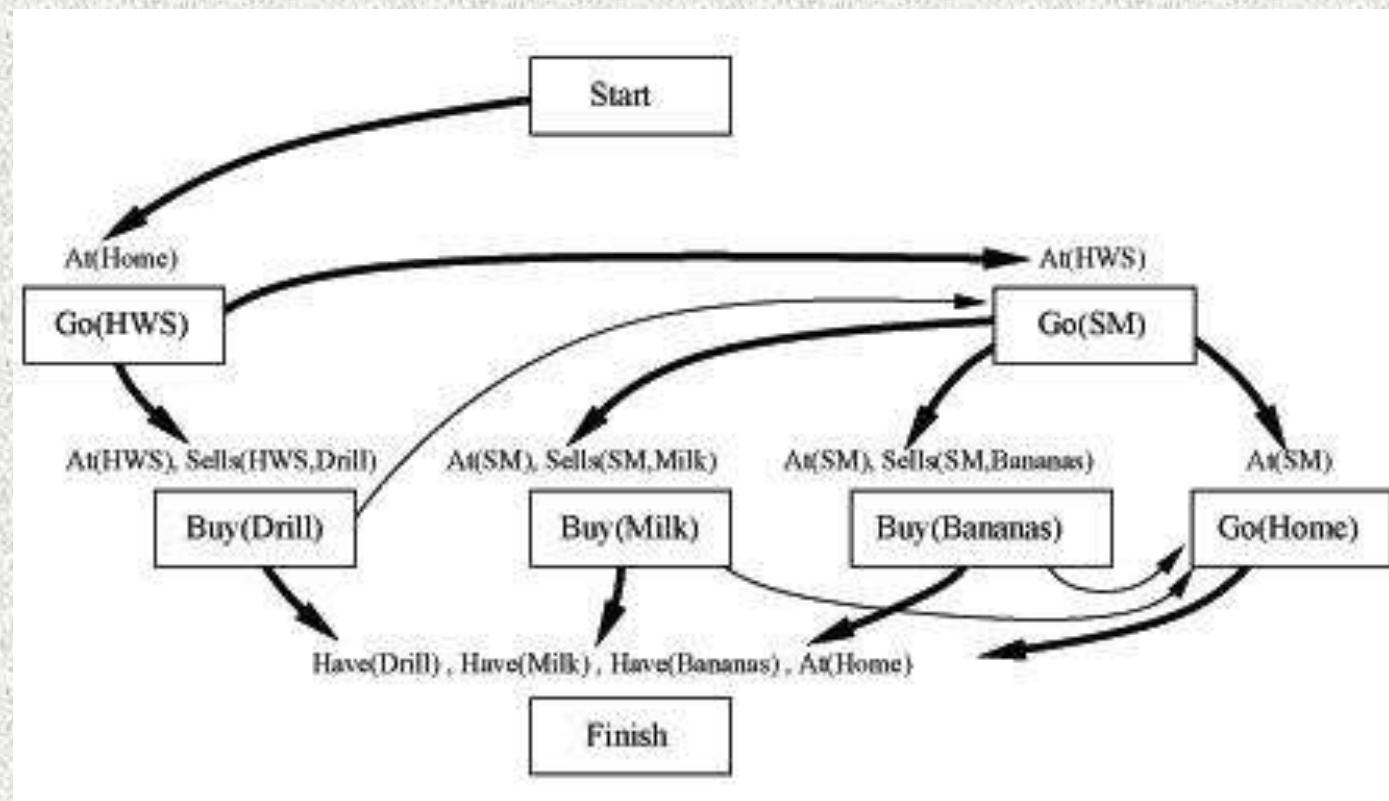
# Example

- This is a flawed plan for directing the agent to the hardware store and the supermarket.
- There is no way to resolve the threat that each Go step poses to the other.
- We now backtrack in the search process.



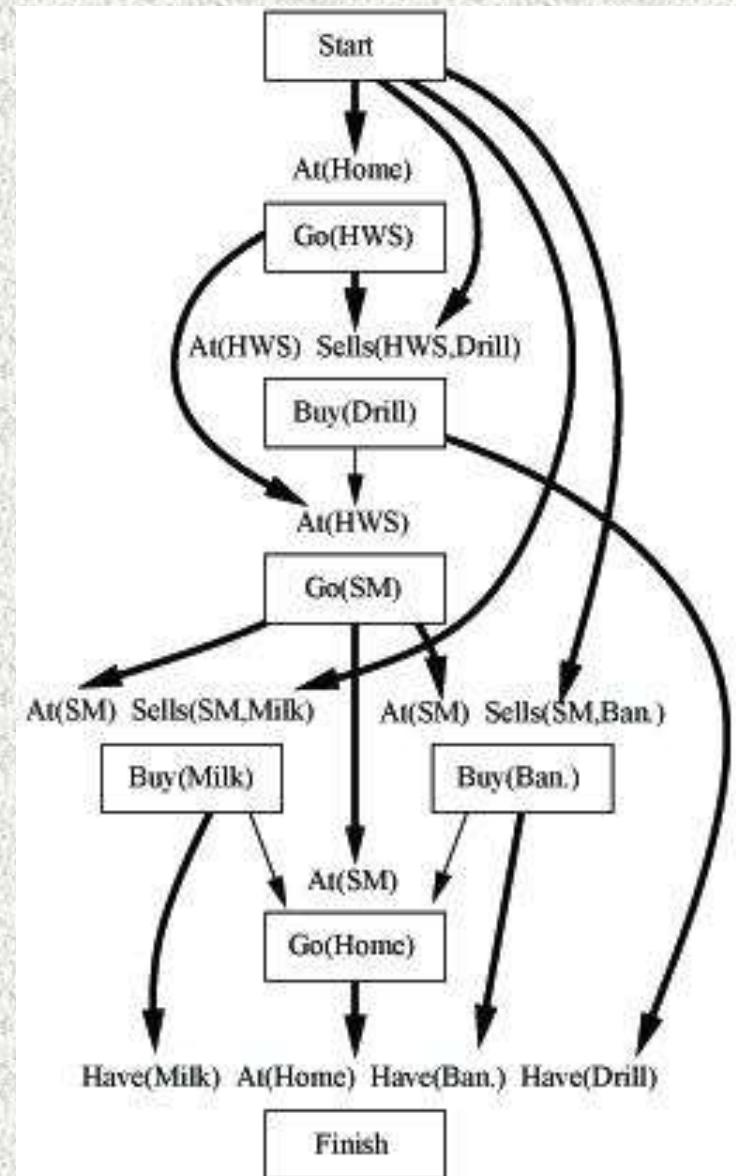
# Example

- The next choice is to achieve the At(x) precondition of the Go(SM) step by adding a causal link from Go(HWS) to Go(SM).
- The Go(SM) step now threatens the At(HWS) precondition of the Buy(Drill) step, and is resolved by promotion.

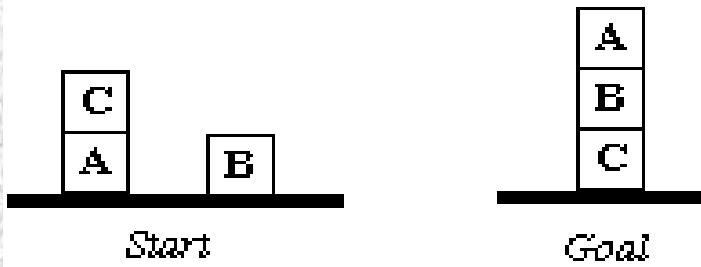


# Example

- The At(Home) precondition of the Finish step is still unachieved.
  - If the planner links At(Home) in the initial state to Finish, there will be no way to resolve the threats raised by Go(HWS) and Go(SM).
  - If the planner tries to link At(x) to Go(HWS), there will be no way to resolve the threat posed by Go(SM), which is ordered after Go(HWS).
  - The planner links Go(SM) to At(x), so x is bound to SM and Go(Home) deletes the At(SM) condition, resulting in threats to the At(SM) precondition for Buy(Milk) and Buy(Bananas). These threats are resolved by promoting Go(Home).
- Here is the final plan.



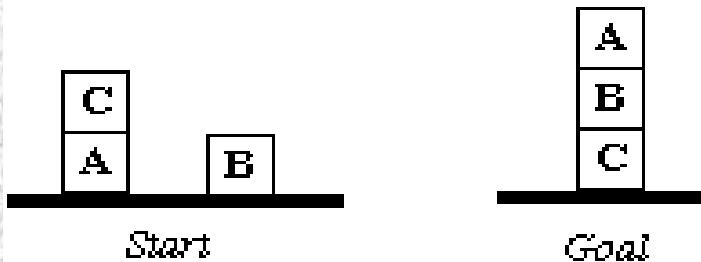
# Partial Plan



- Plan 0
  - Steps: Start(0), Finish(1, pre=On(A,B), On(B,C))
  - Order: 0 before 1 (ignore bindings for now)
  - Links: none
  - Issues: Prec of 1 are not met. Try adding step.

0 → 1

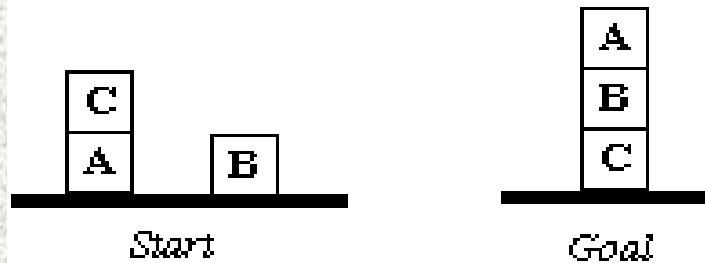
# Partial Plan



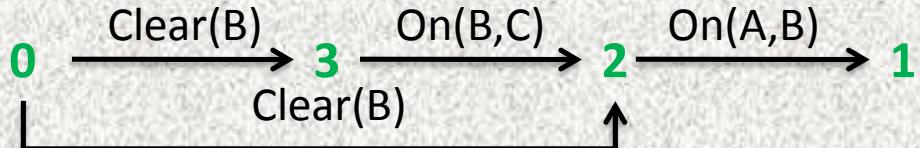
- Plan 1 ( sibling plan could be to add PutOn(B,C))
  - Steps: Start(0), Finish(1, pre=On(A,B), On(B,C))  
PutOn(A,B)(2, pre=Clear(A), Clear(B))
  - Order: 0 before 1, 2 before 1
  - Links:
    - 2 establishes On(A,B) for 1, order 2 before 1
      - No step removes On(A,B) so okay for now
    - 0 establishes Clear(B) for 2, order 0 before 2
  - Issues: The other precondition of step 1 is not met.

$$0 \xrightarrow{\text{Clear}(B)} 2 \xrightarrow{\text{On}(A,B)} 1$$

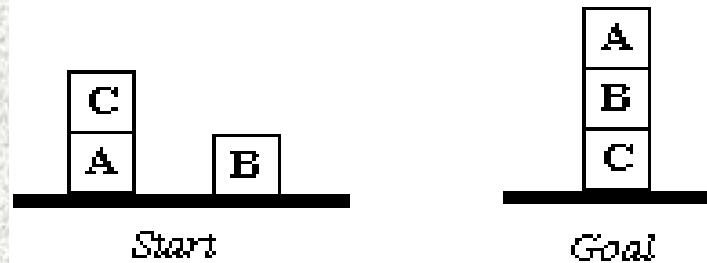
# Partial Plan



- Plan 2 (sibling plans would address pre of step 2)
  - Steps: Start(0), Finish(1, pre=On(A,B), On(B,C))  
PutOn(A,B)(2, pre=Clear(A), Clear(B))  
**PutOn(B,C)(3, pre=Clear(B), Clear(C))**
  - Order: 0 before 1, 2 before 1, 0 before 2, **0 before 3, 3 before 2, 3 before 1**
  - Links:
    - 2 establishes On(A,B) for 1, order 2 before 1
    - 0 establishes Clear(B) for 2, order 0 before 2
    - **0 establishes Clear(B) and Clear(C) for 3, order 0 before 3**
    - **2 clobbers Clear(B), order 3 before 2**
    - 3 establishes On(B,C) for 1, order 3 before 1
    - **No step removes On(B,C), okay for now**
  - Issues: One precondition of step 2 is not met.

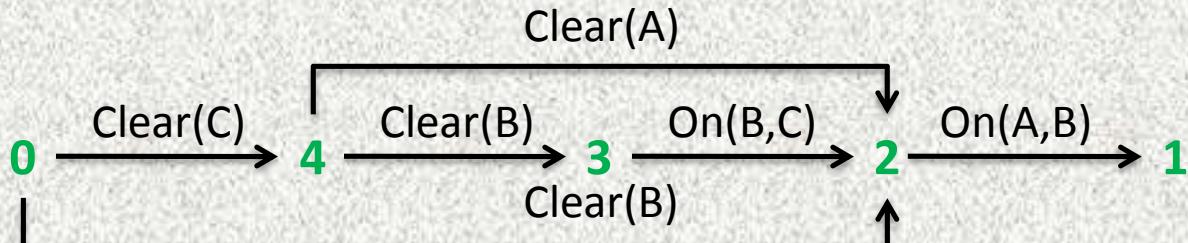


# Partial Plan



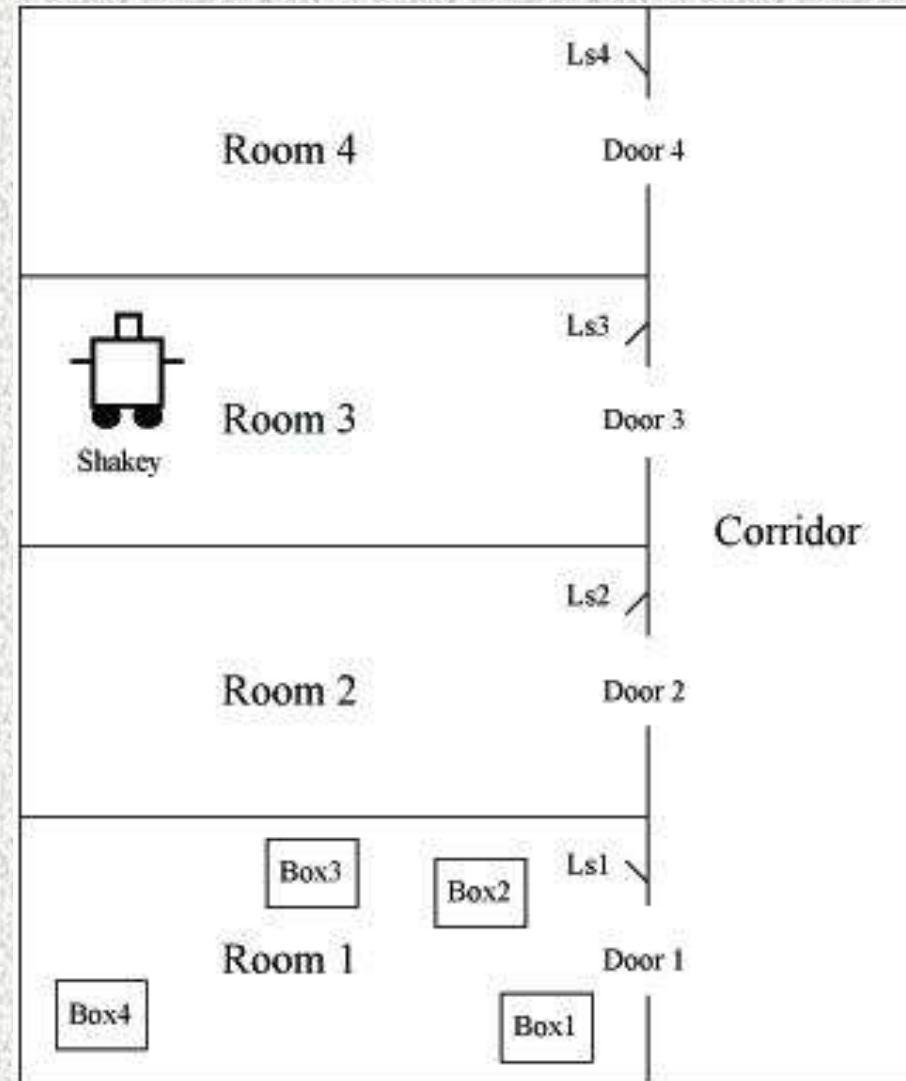
- Plan 3

- Steps: Start(0), Finish(1, pre=On(A,B), On(B,C))  
PutOn(A,B)(2, pre=Clear(A), Clear(B))  
PutOn(B,C)(3, pre=Clear(B), Clear(C))  
**PutOn(C,Table)(4, pre=Clear(C))**
- Order: 0 before 1, 2 before 1, 0 before 2, 0 before 3, 3 before 2, 3 before 1,  
**0 before 4, 4 before 2, 4 before 3**
- Links:
  - 2 establishes On(A,B) for 1, order 2 before 1; 0 establishes Clear(B) for 2, order 0 before 2
  - 0 establishes Clear(B) for 3, order 0 before 3; 2 **clobbers** Clear(B), order 3 before 2
  - 3 establishes On(B,C) for 1, order 3 before 1; No step removes On(B,C), okay for now
  - **4 establishes Clear(A) for 2, order 4 before 2; No step removes Clear(A), okay**
  - **3 clobbers Clear(C), order 4 before 3**
- Issues: **NONE**



# Shakey

- STRIPS was originally designed to control SRI's Shakey robot, using the operators `Go(x)`, `Push(b,x,y)`, `Climb(b)`, `Down(b)`, `TurnOnLight(s)`, and `TurnOffLight(s)`.
- The next generation - [Flakey](#)



# Planning Graphs

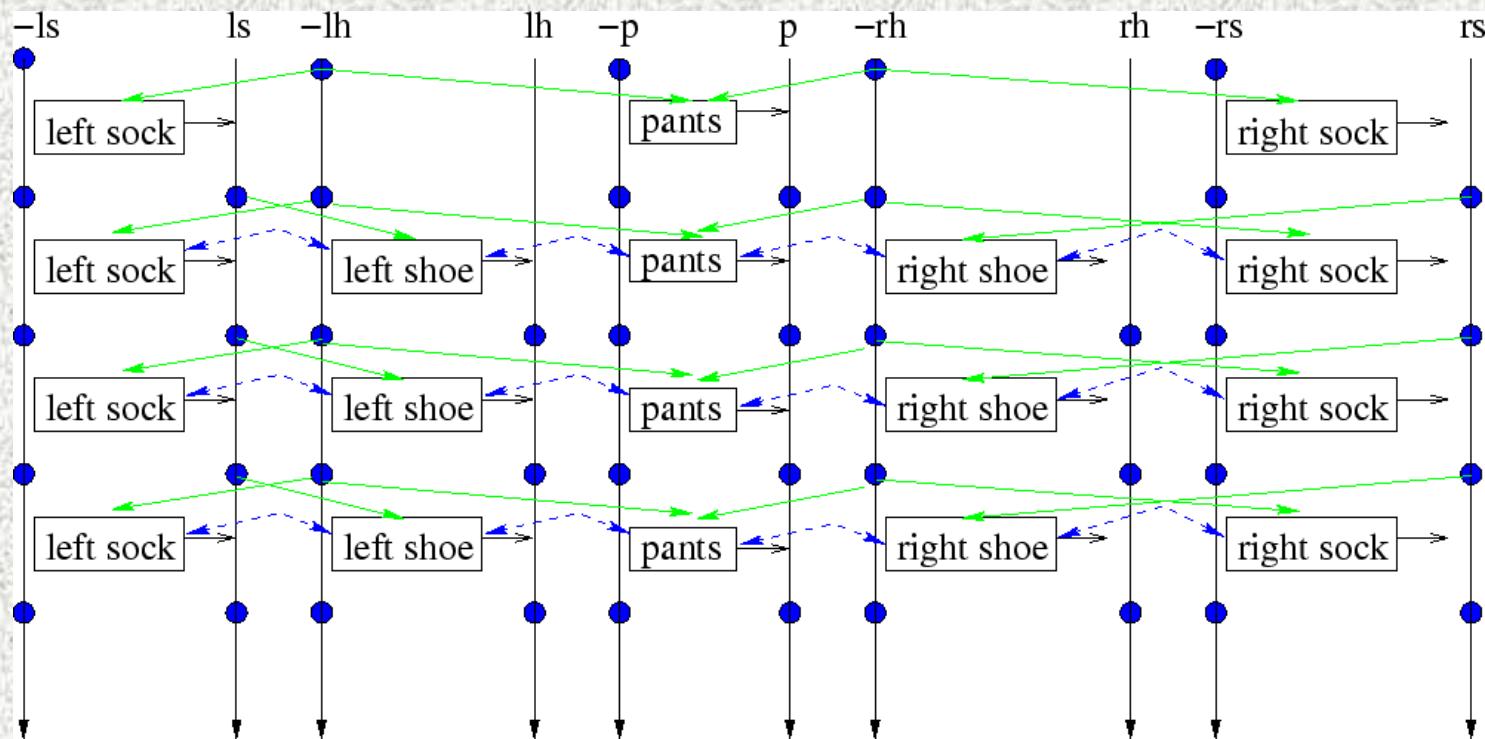
- More efficient and accurate planning technique.
- The idea is to build a graph with  $n$  layers (the maximum plan length is thus  $n$ ). Every level in the graph corresponds to a plan time step.
- The initial state is level 0

# Graph

- Each level contains a set of propositions and a set of actions
- Propositions at level  $i$  could be true in situation  $S_i$ , depending on actions selected earlier
- Actions at level  $i$  could be executed (preconditions satisfied), depending on which propositions are true in  $S_i$
- Every level  $i$  contains all literals that result from any possible choice of actions in the previous level
- Constraints (negative interactions among actions) are noted

# Example

- Propositions
  - ls, rs (wearing left/right sock)
  - lh, rh (wearing left/right shoe)
  - p (wearing pants)
- Actions
  - left/right sock
    - preconditions =  $\neg ls/\neg rs$ ,  $\neg lh/\neg rh$
    - delete =  $\neg ls/\neg rs$
    - add = ls/rs
  - left/right shoe
    - preconditions =  $\neg lh/\neg rh$ , ls/rs
    - delete =  $\neg lh/\neg rh$
    - add = lh/rh
  - pants
    - preconditions =  $\neg p$ ,  $\neg lh/\neg rh$
    - delete =  $\neg p$
    - add = p



# Generating Planning Graph

- First level is initial state
- List possible actions, generate next level as resulting propositions
- Persistence actions are used to represent propositions that remain true from one step to the next
- Mutual exclusion (mutex) links connect actions that cannot co-occur
  - Mutex two actions if one action negatives an effect of the other (inconsistent effects)
  - Mutex two actions if one action effect is the negation of a precondition of the other action (interference)
  - Mutex two actions if the preconditions of the actions are contradictory (competing needs)
- Mutex links also connect literals that cannot appear together
  - One literal is the negation of the other
  - The actions that generate the two literals are mutex (inconsistent support)
- Continue generating graph until two consecutive levels are identical (graph has leveled off)
- Alternatively, continue generating graph until a level contains the goal

# Search Through Graph

- One graph is generated, search backward from goal to initial state for plan.
- Goals at time t
  - ls, rs, lh, rh, p
- Select set of non-mutex actions at time t
  - pants, left sock, right sock
  - left shoe, right shoe
- Goals at time t-1
  - -p, -ls, -rs, -lh, -rh, lh, rh
    - These goals are contradictory, so cannot use
  - -lh, -rh, ls, rs, p

# Search Through Graph

- Select set of non-mutex actions at time t-1
  - left sock, right sock, pants
- Goals at time t-2
  - -p, -ls, -rs, -lh, -rh
- These are satisfied by initial conditions.
- The final plan is
  1. left sock, right sock, pants
  2. left shoe, right shoe

# Analysis

- Efficient
- Graphplan is Complete. If Graphplan does not generate a solution, then no solution exists.

# Additional Planning Considerations

- Resource Constraints
  - Conditional Planning
  - Monitoring / Replanning
  - Scalability
- GraphPlan

# Planning in Practice

- NASA New Millenium Program
  - Programs on board spacecraft perform science planning and scheduling
  - Execute plans without human intervention
- Hitachi's O-PLAN
  - Process planning and scheduling of 30-day assembly schedules
- Desert Storm Planning
  - Plan for transportation of troops, supplies, weapons

# CptS 440 / 540

# Artificial Intelligence

Introduction to Prolog

# Prolog – PROgramming in LOGic

- Declarative and logical programming language
- Built-in inference mechanism based on resolution

# SWI-Prolog

- Download from [www.swi-prolog.org](http://www.swi-prolog.org)
  - Log in to Its1
    - You can get to this machine from EECS or VPNd in
    - If off campus, ssh to ssh-server.eecs.wsu.edu, then connect to Its1
  - Type `pl` to start.
  - Type `halt.` to exit.

pl

Welcome to SWI-Prolog (Multi-threaded, Version 5.2.13)

Copyright (c) 1990-2003 University of Amsterdam.

SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,  
and you are welcome to redistribute it under certain conditions.

Please visit <http://www.swi-prolog.org> for details.

For help, use `?- help(Topic).` or `?- apropos(Word).`

`?-`

Type `help(help)`. for online manual.

# Structure of Prolog Programs

- Every entry into the interpreter is a goal that Prolog tries to satisfy.
- Every goal in Prolog ends with .
- A Prolog program consists of a database of facts, rules, and queries. This program is, in essence, a knowledge base.
  - Fact: head but no body  
man(socrates).  
man(plato).
  - Rules: head and body  
mortal(X) :- man(X).
  - Questions: body but no head  
mortal(X).  
Use ``;'' to get next possible answer, Return to end.  
Yes means true with no variables, no means not consistent with database.

# Test Sample Program (intro.pl)

Load intro.pl into Prolog by typing

[intro].

The ``yes'' at the end indicates that Prolog checked the code and found no errors.

Check all of what Prolog has recorded by asking for a listing.

?- [listing](#).

```
% Foreign: rl_add_history/1
% Foreign: rl_read_init_file/1
likes(mary, food).
likes(mary, wine).
likes(john, food).
likes(john, wine).
```

Yes

# Pose a Query

?- likes(mary,food).

Yes

?- likes(X,food).

X = mary /\* press ; to get next answer, press return to finish \*/

Yes

# Interactive Definitions (soc.pl)

To enter facts and/or rules directly, type:

```
| ?- consult(user).
```

```
|
```

You get a new prompt as above (|). At this point you would enter your facts and rules. For example,

```
| man(socrates).
```

```
| mortal(X) :- man(X).
```

```
| end_of_file.
```

```
| ?-
```

When done entering facts and rules, just type 'end\_of\_file.', and you will return to the original prompt.

Now we can ask queries of our facts and rules. For example,

```
| ?- man(socrates).
```

```
yes
```

```
| ?- mortal(X).
```

X = socrates      <press return at this point>

```
| ?- man(plato).
```

no

```
| ?-
```

# Logical Operators

a :- b. /\* a if b \*/

a :- b,c. /\* a if b and c. \*/

a :- b;c. /\* a if b or c. \*/

a :- not b. /\* a if b fails \*/

a :- b -> c;d. /\* a if (if b then c else d) \*/

# Compound Query (dog.pl)

likes(X,Y), food(Y).

There is an implied “and” between the query terms.

# Disjunction

- ;
- Multiple rules
- $P :- Q.$   $P :- R,$  S. Equivalent to  
➤  $P :- Q; R,$  S.
- $\text{ancestor}(X,Z) :- \text{parent}(X,Z); \text{parent}(X,Y), \text{ancestor}(Y,Z).$

# Example (family.pl)

Let family.pl consist of the KB

```
% male(P) is true when P is male
male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).
```

```
% female(P) is true when P is female
female(catherine).
female(elizabeth).
female(sophia).
```

```
% parent(C,P) is true when C has a parent P
parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
parent(james2, charles1).
parent(sophia, elizabeth).
parent(george1, sophia).
```

# Queries

- Was George I the parent of Charles I?

parent(charles1, george1).

# Queries

- Who was the parent of Charles I?

parent(charles1, Parent).

# Queries

- Who were the children of Charles I?

parent(Child, charles1).

# Create Rules (rules.pl, neg1.pl, neg2.pl)

- M is the mother of P if she is a parent of P and is female.
- F is the father of P if he is a parent of P and is male
- X is a sibling of Y if they both have the same parent.
- sister, brother, aunt, uncle, grandparent, cousin

# Prolog Data Objects

- Atoms
  - String of letters and digits
  - Start with lower case letter
  - yes, parent, female, tom, tom\_jones
  - String of characters enclosed in single quotes ('Tom Jones')
- Numbers [-][0-9]\*[.][0-9]\*
  - 2.71, -100.5, 0.5 (but not .5)
- Variables
  - String of letters and digits
  - Start with upper case letter (or \_, anonymous variable)
  - Universally quantified
- Structures
  - functor(arg1, arg2, )
  - date(2, october, 1964)
  - Arguments can be constants, variables, or other structures
  - Defined by Name and Arity (date/3 different from date/2)

# Arithmetic Operators

Operators +, -, \*, /, sqrt, exp, cos, and so forth are available. However, such expressions will not “match” a variable.

prime(2).

prime(3).

prime(5).

...

prime(1+1) will fail, because cannot be unified with any entry in the KB.

# Comparison Operators

X = Y % X and Y are unifiable

T1 == T2 % True if T1 and T2 are identical (names of variables are the same)

T1 \== T2 % True if T1 and T2 are not identical

E1 =:= E2 % True if values of expressions E1 and E2 are equal

E1 =\= E2 % True if values of expressions E1 and E2 are not equal

E1 < E2 % True if numeric value of E1 is < numeric value of E2

Similar for operators =<, >, and >=

positive(N) :- N>0.

non\_zero(N) :- N<0 ; N>0.

# Assignment

The predicate “N is E” will succeed whenever N is an unbound variable and E is an arithmetic expression.

Try:

X is 1+1, prime(X).

# Prolog Arithmetic Queries

?- N is 1+1.

?- N is 1+1, P is N\*2, Q is P+P.

?- N is X+1.

?- I is I+1.

?- I is 6, I is I+1. /\* Creates an error \*/

?- I is 6, J is I+1.

# Example Program (factorial.pl)

```
?- [factorial]. /* Could also type pl -s factorial.pl at start */
```

```
% factorial.pl compiled 0.00 sec, 628 bytes
```

Yes

```
?- listing(factorial/2).
```

```
factorial(0, 1).
```

```
factorial(A, B) :-
```

```
 A>0,
```

```
 C is A-1,
```

```
 factorial(C, D),
```

```
 B is A*D.
```

Yes

```
?- factorial(10,What).
```

```
What = 3628800 /* Hit return to get other possible solutions */
```

Yes

```
?- halt.
```

# Arithmetic Examples (arithmetic.pl)

- The result of adding 1 to a number
- The function signum( $x$ ) which is  $x-1$  if  $x > 0$ , and 0 otherwise.
- The maximum of two numbers
- The maximum of three numbers
- The absolute value of a number

# Negation

How can Prolog handle verification  
that a goal is *not* entailed by the  
knowledge base?

The **not** predicate.

Consider the program:

```
bachelor(P) :- male(P),
 not(married(P)).
```

```
male(henry).
```

```
male(tom).
```

```
married(tom).
```

Then

```
?- bachelor(henry).
```

yes

```
?- bachelor(tom).
```

no

```
?- bachelor(Who).
```

Who= henry ;

no

```
?- not(married(Who)).
```

no.

# Negation Examples

$p(X) :- q(X), \text{not}(r(X)).$

$r(X) :- w(X), \text{not}(s(X)).$

$q(a).$   $q(b).$   $q(c).$

$s(a).$   $s(c).$

$w(a).$   $w(b).$

Test for  $p(a).$

# I/O (read\_line.pl)

- Input
  - Predicate `read(X)` reads a term from a file or the keyboard.
- Output
  - Predicate `write(X)` writes X to a file or the screen, `writeln(X)` adds newline, `nl` outputs a newline on the screen, `tab(X)` tabs X number of spaces.

# Example (io.pl)

```
cube(C,N) :- C is N * N * N.
```

For interactive input:

```
cube :- read(X), calc(X).
```

```
calc(stop) :- !.
```

```
calc(X) :- C is X * X * X, write(C), cube.
```

Now will read until see “stop”.

# Files

```
browse(File) :-
 seeing(Old), /* save for later */
 see(File), /* open this file */
 repeat,
 read(Data), /* read from File */
 process(Data),
 seen, /* close File */
 see(Old), /* previous read source */
 !. /* stop now */
```

```
process(end-of-file) :- !.
process(Data) :- write(Data), nl, fail.
```

# Interactive Version

```
browse(File) :-
 seeing(Old), /* save for later */
 see(user),
 write('Enter name of file to browse: '),
 read(File),
 see(File), /* open this file */
 repeat,
 read(Data), /* read from File */
 process(Data),
 seen, /* close File */
 see(Old), /* previous read source */
 !. /* stop now */
```

# Green's Trick Revisited

Sally is studying with Morton

1.  $\text{studyingwith}(\text{Sally}, \text{Morton})$

Morton is at the CUB.

2.  $\text{at}(\text{Morton}, \text{Cub})$

If any person is studying with another person who is at a particular place, the first person is also at that place.

3.  $\text{Forall } x, y \text{ studyingwith}(x, y) \& \text{at}(y, z) \rightarrow \text{at}(x, z)$

If someone is at a particular place, then that person can be reached using the telephone number for that place.

4.  $\text{Forall } x, y \text{ at}(x, y) \rightarrow \text{reach}(x, \text{phone}(y))$

Where can Sally be reached?

# Green's Trick Revisited

1. studyingwith(Sally,Morton)
2. at(Morton,Cub)
3. -studyingwith(x3,y3)  $\vee$  -at(y3,z3)  $\vee$  at(x3,z3)
4. -at(x4,y4)  $\vee$  reach(x4,phone(y4))

Where can Sally be reached?

Exists x reach(Sally, x)

Negate and clausefy: -reach(Sally,x5)

Form disjunct with opposite and add to DB:

5. -reach(Sally, x5)  $\vee$  reach(Sally, x5)

# Green's Trick Revisited

1. studyingwith(Sally,Morton)
2. at(Morton,CUB)
3. -studyingwith(x3,y3)  $\vee$  -at(y3,z3)  $\vee$  at(x3,z3)
4. -at(x4,y4)  $\vee$  reach(x4,phone(y4))
5. –reach(Sally, x5)  $\vee$  reach(Sally, x5)
6. [4,5 x4/Sally x5/phone(y4)]  
  –at(Sally, y6)  $\vee$  reach(Sally, phone(y6))
7. [3,6 x3/Sally z3/y6]  
  –studyingwith(Sally, y7)  $\vee$  –at(y7,a7)  $\vee$  reach(Sally, phone(a7))
8. [1,7 y7/Morton] –at(Morton,a8)  $\vee$  reach(Sally, phone(a8))
9. [2,8 a8/CUB] **reach(Sally, phone(CUB))**

# Green's Trick in Prolog (s.pl)

studyingwith(sally,morton).

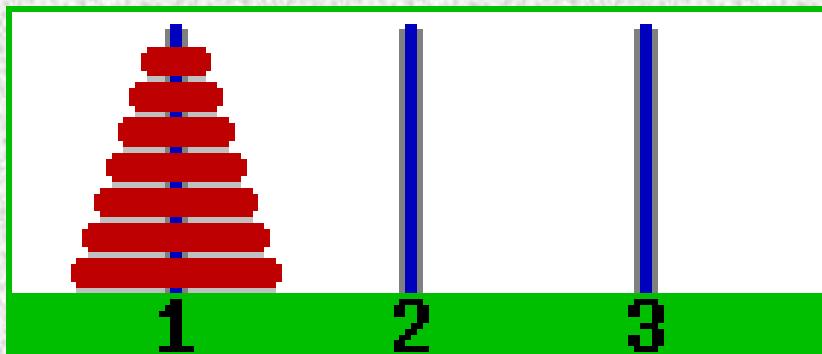
at(morton,cub).

at(X,Z) :- studyingwith(X,Y), at(Y,Z).

reach(X,phone(Y)) :- at(X,Y).

Query: reach(sally, X).

# Example Problems – Towers of Hanoi



**States:** combinations of poles and disks

**Operators:** move disk x from pole y to pole z  
subject to constraints

- cannot move disk on top of smaller disk
- cannot move disk if other disks on top

**Goal test:** disks from largest (at bottom) to smallest on goal pole

**Path cost:** 1 per move

[Towers of Hanoi applet](#)

# Towers of Hanoi (hanoi.pl)

```
move(1,X,Y,_) :-
 write('Move top disk from '),
 write(X),
 write(' to '),
 write(Y),
 nl.
```

```
move(N,X,Y,Z) :- /* First argument is #disks, others are the 3 poles */
 N>1,
 M is N-1,
 move(M,X,Z,Y),
 move(1,X,Y,_),
 move(M,Z,Y,X).
```

Try `move(3, left, right, center).`

# Cut (cutexample.pl)

The cut predicate, ``!'', eliminates choices in a Prolog derivation tree (stops the backtracking along a particular path).

***Useful if you only want one solution for part or all of a rule.***

Consider the program

```
/* program P clause # */
p(a). /* #1 */
p(X) :- q(X), r(X). /* #2 */
p(X) :- u(X). /* #3 */
q(X) :- s(X). /* #4 */
r(a). /* #5 */
r(b). /* #6 */
s(a). /* #7 */
s(b). /* #8 */
s(c). /* #9 */
u(d). /* #10 */
```

Consider goal

p(X),!.

Result is

X=a ;  
no

Cut succeeds when it is the current goal and backtracking up to the cut is pruned.

In this case, the second and third derivations are eliminated, and hence the entire subtrees below these two edges are also cut off.

Try

r(X),s(Y).  
r(X),!,s(Y).  
r(X), s(Y), !.

# Cut Example

part(a). part(b). part(c).

red(a). black(b).

color(P,red) :- red(P),!.

color(P,black) :- black(P),!.

color(P,unknown).

This finds a stored color or concludes the color is unknown. Derivations for color are prevented. Thus unknown is not reported for a or b.

# Expert Systems in Prolog (animal.pl)

- **assert(Term)**
  - Add fact or clause to database.
- **retract(Term)**
  - Remove all facts or clauses in the database that unify with Term.
- Can see results in listing.

# Lists in Prolog

- Elements of lists are any valid Prolog data object
- Elements are terms separated by commas, do not have to be same type
- Vertical bar separates head (first) from tail (rest)
- The head of [john, mary, pat] is john
- The tail of [john, mary, pat] is [mary, pat]
- The representation of the combination is Head | Tail (in our example, [john | [mary, pat]] or [john | [mary | [pat | []]]])
- Member function
  - member(X, L) :- L = [X | \_].
  - member(X, L) :- L = [A | B], member(X, B).

# Prolog list manipulation [sum.pl, temp.pl]

first([X|Y],X).

rest([X|Y],Y).

addfirst(X,R,[X|R]).

means

The head (first) of [X|Y] is X.

The tail (rest) of [X|Y] is Y.

Putting X at the head and Y as the tail constructs  
(addfirst) the list [X|R].

# CptS 440 / 540

# Artificial Intelligence

Uncertainty Reasoning

# Non-monotonic Logic

- Traditional logic is **monotonic**
  - The set of legal conclusions grows **monotonically** with the set of facts appearing in our initial database
- When humans reason, we use **defeasible** logic
  - Almost every conclusion we draw is subject to reversal
  - If we find contradicting information later, we'll want to **retract** earlier inferences
- **Nonmonotonic logic**, or **defeasible reasoning**, allows a statement to be retracted
- Solution: **Truth Maintenance**
  - Keep explicit information about which facts/inferences support other inferences
  - If the foundation disappears, so must the conclusion

# Uncertainty

- On the other hand, the problem might not be in the fact that T/F values can change over time but rather that we are not **certain** of the T/F value
- Agents almost never have access to the whole truth about their environment
- Agents must act in the presence of **uncertainty**
  - Some information ascertained from facts
  - Some information inferred from facts and knowledge about environment
  - Some information based on assumptions made from experience

# Environment Properties

- Fully observable vs. partially observable
- Deterministic vs. stochastic / strategic
- Episodic vs. sequential
- Static vs. dynamic
- Discrete vs. continuous
- Single agent vs. multiagent

# Uncertainty Arises Because of Several Factors

- Incompleteness
  - Many rules are incomplete because too many conditions to be explicitly enumerated
  - Many rules incomplete because some conditions are unknown
- Incorrectness

# Where Do Probabilities Come From?

- Frequency
- Subjective judgment
- Consider the probability that the sun will still exist tomorrow.
- There are several ways to compute this
- Choice of experiment is known as the **reference class problem**

# Acting Under Uncertainty

- Agents must still act even if world not certain
- If not sure which of two squares have a pit and must enter one of them to reach the gold, the agent will take a chance
- If can only act with certainty, most of the time will not act. Consider example that agent wants to drive someone to the airport to catch a flight, and is considering plan A90 that involves leaving home 60 minutes before the flight departs and driving at a reasonable speed. Even though the Pullman airport is only 5 miles away, the agent will not be able to reach a definite conclusion - it will be more like “[Plan A90 will get us to the airport in time, as long as my car doesn't break down or run out of gas, and I don't get into an accident, and there are no accidents on the Moscow-Pullman highway, and the plane doesn't leave early, and there's no thunderstorms in the area, ...](#)”
- We may still use this plan if it will improve our situation, given known information
- The performance measure here includes getting to the airport in time, not wasting time at the airport, and/or not getting a speeding ticket.

# Limitation of Deterministic Logic

- Pure logic fails for three main reasons:
- **Laziness**
  - Too much work to list complete set of antecedents or consequents needed to ensure an exceptionless rule, too hard to use the enormous rules that result
- **Theoretical ignorance**
  - Science has no complete theory for the domain
- **Practical ignorance**
  - Even if we know all the rules, we may be uncertain about a particular patient because all the necessary tests have not or cannot be run

# Probability

- Probabilities are numeric values between 0 and 1 (inclusive) that represent ideal certainties (not beliefs) of statements, given assumptions about the circumstances in which the statements apply.
- These values can be verified by testing, unlike certainty values. They apply in highly controlled situations.

$$\text{Probability(event)} = P(\text{event}) = \frac{\#\text{instances of the event}}{\text{total } \#\text{instances}}$$

# Example

- For example, if we roll two dice, each showing one of six possible numbers, the number of total unique rolls is  $6*6 = 36$ . We distinguish the dice in some way (a first and second or left and right die). Here is a listing of the joint possibilities for the dice:

(1,1) (1,2) (1,3) (1,4) (1,5) (1,6)  
(2,1) (2,2) (2,3) (2,4) (2,5) (2,6)  
(3,1) (3,2) (3,3) (3,4) (3,5) (3,6)  
(4,1) (4,2) (4,3) (4,4) (4,5) (4,6)  
(5,1) (5,2) (5,3) (5,4) (5,5) (5,6)  
(6,1) (6,2) (6,3) (6,4) (6,5) (6,6)

- The number of rolls which add up to 4 is 3 ((1,3), (2,2), (3,1)), so the probability of rolling a total of 4 is  $3/36 = 1/12$ .
- This does not mean 8.3% true, but 8.3% chance of it being true.

# Probability Explanation

- $P(\text{event})$  is the probability in the absence of any additional information
- Probability depends on evidence.
- Before looking at dice:  $P(\text{sum of } 4) = 1/12$
- After looking at dice:  $P(\text{sum of } 4) = 0 \text{ or } 1$ , depending on what we see
- All probability statements must indicate the evidence with respect to which the probability is being assessed.
- As new evidence is collected, probability calculations are updated.
- Before specific evidence is obtained, we refer to the **prior** or **unconditional** probability of the event with respect to the evidence. After the evidence is obtained, we refer to the **posterior** or **conditional** probability.

# Probability Distributions

- If we want to know the probability of a variable that can take on multiple values, we may define a **probability distribution**, or a set of probabilities for each possible variable value.
- TemperatureToday =  
    {Below50, 50s, 60s, 70s, 80s, 90sAndAbove}
- $P(\text{TemperatureToday}) =$   
    {0.1, 0.1, 0.5, 0.2, 0.05, 0.05}
- Note that the sum of the probabilities for possible values of any given variable must always sum to 1.

# Joint Probability Distribution

- Because events are rarely isolated from other events, we may want to define a joint probability distribution, or  $P(X_1, X_2, \dots, X_n)$ .
- Each  $X_i$  is a vector of probabilities for values of variable  $X_i$ .
- The joint probability distribution is an n-dimensional array of combinations of probabilities.

|             | Wet | $\sim$ Wet |
|-------------|-----|------------|
| Rain        | 0.6 | 0.4        |
| $\sim$ Rain | 0.4 | 0.6        |

# Inference by Enumeration

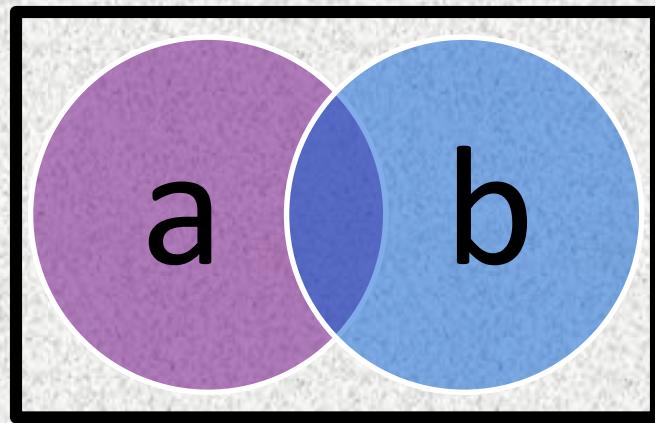
- To determine the probability of one variable (e.g., toothache), sum the events in the joint probability distribution where it is true:

|         | toothache |        | ~toothache |        |
|---------|-----------|--------|------------|--------|
|         | catch     | ~catch | catch      | ~catch |
| cavity  | .108      | .012   | .072       | .008   |
| ~cavity | .016      | .064   | .144       | .576   |

$$P(\text{toothache}) = .108 + .012 + .016 + .064 = 0.2$$

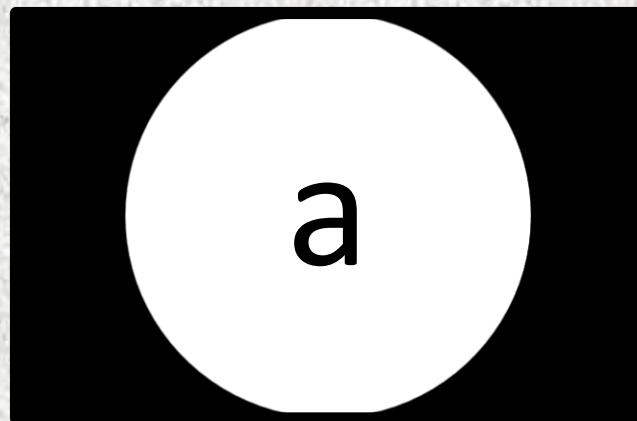
# Axioms of Probability

- $0 \leq P(\text{Event}) \leq 1$
- Disjunction,  $a \vee b$ ,  $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$



# Axioms of Probability

- Negation,  $P(\sim a) = 1 - P(a)$

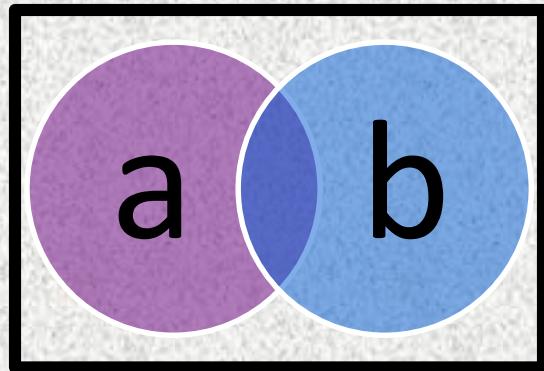


# Axioms of Probability

- Conditional probability
  - Once evidence is obtained, the agent can use conditional probabilities,  $P(a|b)$
  - $P(a|b)$  = probability of a being true given that we know b is true
  - The equation  $P(a|b) = \frac{P(a \wedge b)}{P(b)}$  holds whenever  $P(b)>0$
- An agent who bets according to probabilities that violate these axioms can be forced to bet so as to lose money regardless of outcome [deFinetti, 1931]

# Axioms of Probability

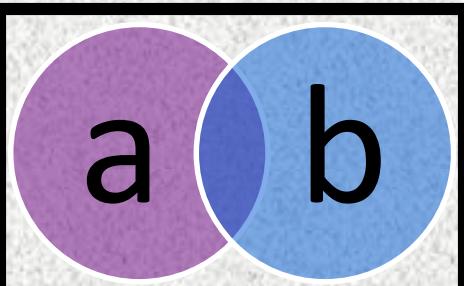
- Conjunction
  - Product rule
  - $P(a \wedge b) = P(a) * P(b | a)$
  - $P(a \wedge b) = P(b) * P(a | b)$



- In other words, the only way a and b can both be true is if a is true and we know b is true given a is true (thus b is also true)

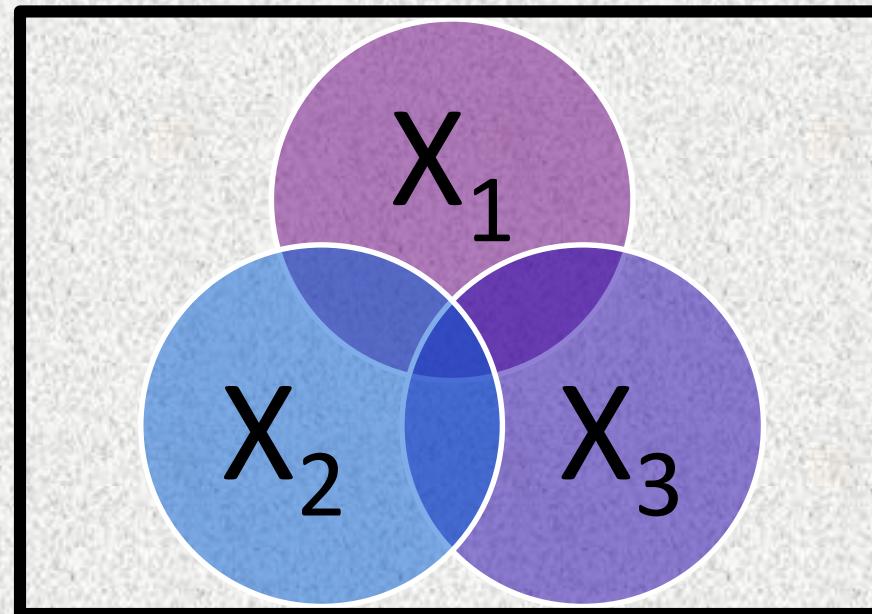
# Axioms of Probability

- If a and b are independent events (the truth of a has no effect on the truth of b), then  $P(a \wedge b) = P(a) * P(b)$ .
- “Wet” and “Raining” are not independent events.
- “Wet” and “Joe made a joke” are pretty close to independent events.



# More Than 2 Variables

- The **chain rule** is derived by successive application of the product rule:
- $$\begin{aligned} P(X_1, \dots, X_n) &= P(X_1, \dots, X_{n-1})P(X_n | X_1, \dots, X_{n-1}) \\ &= P(X_1, \dots, X_{n-2})P(X_{n-1} | X_1, \dots, X_{n-2})P(X_n | X_1, \dots, X_{n-1}) \\ &= \dots \\ &= \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1}) \end{aligned}$$



# Law of Alternatives

- If we know that exactly one of A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub> are true, then we know

$$P(B) = P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + \dots + P(B|A_n)P(A_n)$$

and

$$P(B|X) = P(B|A_1, X) + \dots + P(B|A_n, X)P(A_n, X)$$

- Example

- $P(\text{Sunday}) = P(\text{Monday}) = \dots = P(\text{Saturday}) = 1/7$

- $P(\text{Football Today}) =$

$$P(\text{Football Today} | \text{Sunday})P(\text{Sunday}) +$$

$$P(\text{Football Today} | \text{Monday})P(\text{Monday}) +$$

$$\dots +$$

$$P(\text{Football Today} | \text{Saturday})P(\text{Saturday})$$

$$= 0 + 0 + 0 + 0 + 0 + 0 + 1/7 * 1 = 1/7$$

# Lunar Lander Example

- A lunar lander crashes somewhere in your town (one of the cells at random in the grid). The crash point is uniformly random (the probability is uniformly distributed, meaning each location has an equal probability of being the crash point).
- D is the event that it crashes downtown.
- R is the event that it crashes in the river.

|   |   |   |   |   |    |    |    |   |  |
|---|---|---|---|---|----|----|----|---|--|
|   |   |   |   |   |    |    |    |   |  |
|   |   |   |   |   |    |    |    |   |  |
|   |   |   |   |   | D  | D  | D  |   |  |
| R | R | R | R | R | DR | DR | DR | R |  |
| R | R | R | R | R | DR | DR | DR | R |  |
|   |   |   |   |   | D  | D  | D  |   |  |

What is  $P(R)$ ? 18/54

What is  $P(D)$ ? 12/54

What is  $P(D \cap R)$ ? 6/54

What is  $P(D|R)$ ? 6/18

What is  $P(R|D)$ ? 6/12

What is  $P(R \cap D)/P(D)$ ? 6/12

# Axioms of Probability

- Bayes' Rule
  - Given a hypothesis ( $H$ ) and evidence ( $E$ ), and given that  $P(E) = 0$ , what is  $P(H|E)$ ?
- Many times rules and information are uncertain, yet we still want to say something about the consequent; namely, the degree to which it can be believed. A British cleric and mathematician, Thomas Bayes, suggested an approach.
- Recall the two forms of the product rule:
  - $P(ab) = P(a) * P(b|a)$
  - $P(ab) = P(b) * P(a|b)$
- If we equate the two right-hand sides and divide by  $P(a)$ , we get 
$$P(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

# Example

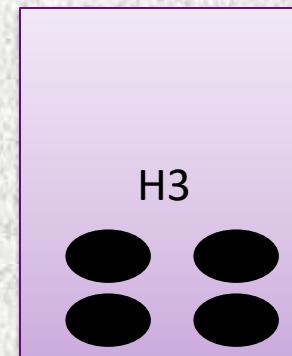
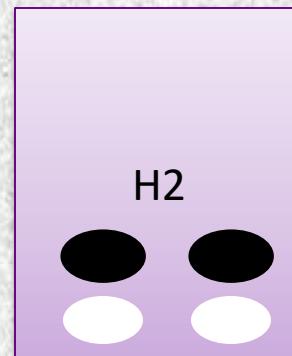
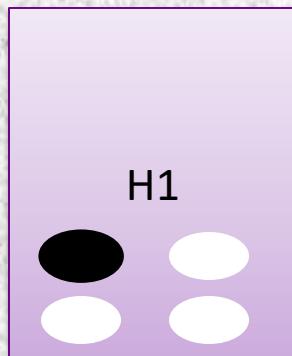
- Bayes' rule is useful when we have three of the four parts of the equation.
- In this example, a doctor knows that meningitis causes a stiff neck in 50% of such cases. The prior probability of having meningitis is 1/50,000 and the prior probability of any patient having a stiff neck is 1/20.
- What is the probability that a patient has meningitis if they have a stiff neck?
- $H = \text{"Patient has meningitis"}$
- $E = \text{"Patient has stiff neck"}$

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

$$P(H|E) = (0.5 * .00002) / .05 = .0002$$

# Example

- I have three identical boxes labeled H1, H2, and H3
  - I place 1 black bead and 3 white beads into H1
  - I place 2 black beads and 2 white beads into H2
  - I place 4 black beads and no white beads into H3
- I draw a box at random, and randomly remove a bead from that box. Given the color of the bead, what can I deduce as to which box I drew?
- If I replace the bead, then redraw another bead at random from the same box, how well can I predict its color before drawing it?

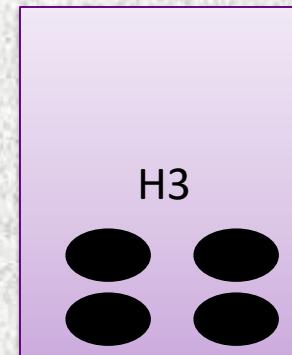
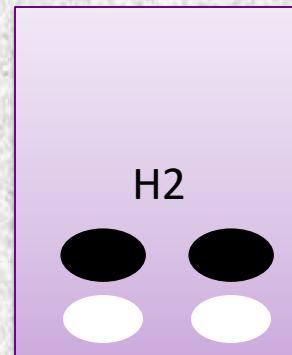
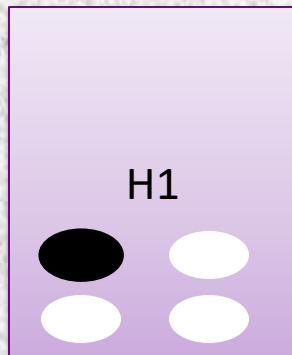


# Answer

- Observation: I draw a white bead.
- $P(H1|W) = P(H1)P(W|H1) / P(W)$   
 $= (1/3 * 3/4) / 5/12 = 3/12 * 12/5 = 36/60 = 3/5$
- $P(H2|W) = P(H2)P(W|H2) / P(W)$   
 $= (1/3 * 1/2) / 5/12 = 1/6 * 12/5 = 12/30 = 2/5$
- $P(H3|W) = P(H3)P(W|H3) / P(W)$   
 $= (1/3 * 0) / 5/12 = 0 * 12/5 = 0$

# Example

- If I replace the bead, then redraw another bead at random from the same box, how well can I predict its color before drawing it?
- $P(H1)=3/5$ ,  $P(H2) = 2/5$ ,  $P(H3) = 0$
- $P(W) = P(W|H1)P(H1) + P(W|H2)P(H2) + P(W|H3)P(H3)$   
 $= 3/4 * 3/5 + 1/2 * 2/5 + 0 * 0 = 9/20 + 4/20 = 13/20$



# Monty Hall Problem

- [Monty Hall Applet](#)
- [Another Monty Hall Applet](#)

# Example

- We wish to know probability that John has malaria, given that he has a slightly unusual symptom: a high fever.
- We have 4 kinds of information
  - a) probability that a person has malaria regardless of symptoms (0.0001)
  - b) probability that a person has the symptom of fever given that he has malaria (0.75)
  - c) probability that a person has symptom of fever, given that he does NOT have malaria (0.14)
  - d) John has high fever
- $H = \text{John has malaria}$
- $E = \text{John has a high fever}$

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

Suppose  $P(H) = 0.0001$ ,  $P(E|H) = 0.75$ ,  $P(E|\sim H) = 0.14$

# Example

- We wish to know probability that John has malaria, given that he has a slightly unusual symptom: a high fever.
- We have 4 kinds of information
  - a) probability that a person has malaria regardless of symptoms
  - b) probability that a person has the symptom of fever given that he has malaria
  - c) probability that a person has symptom of fever, given that he does NOT have malaria
  - d) John has high fever
- $H = \text{John has malaria}$
- $E = \text{John has a high fever}$

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

Suppose  $P(H) = 0.0001$ ,  $P(E|H) = 0.75$ ,  $P(E|\sim H) = 0.14$

Then  $P(E) = 0.75 * 0.0001 + 0.14 * 0.9999 = 0.14006$

and  $P(H|E) = (0.75 * 0.0001) / 0.14006 = 0.0005354$

On the other hand, if John did not have a fever, his probability of having malaria would be

$$P(H|\sim E) = \frac{P(\sim E|H) * P(H)}{P(\sim E)} = \frac{(1-0.75)(0.0001)}{(1-0.14006)} = 0.000029$$

Which is much smaller.

# Making Decision Under Uncertainty

- Consider the following plans for getting to the airport:
  - $P(A_{25} \text{ gets me there on time} \mid \dots) = 0.04$
  - $P(A_{90} \text{ gets me there on time} \mid \dots) = 0.70$
  - $P(A_{120} \text{ gets me there on time} \mid \dots) = 0.95$
  - $P(A_{1440} \text{ gets me there on time} \mid \dots) = 0.9999$
- Which action should I choose?
- Depends on my **preferences** for missing the flight vs. time spent waiting, etc.
  - **Utility theory** is used to represent and infer preferences
  - **Decision theory** is a combination of probability theory and utility theory

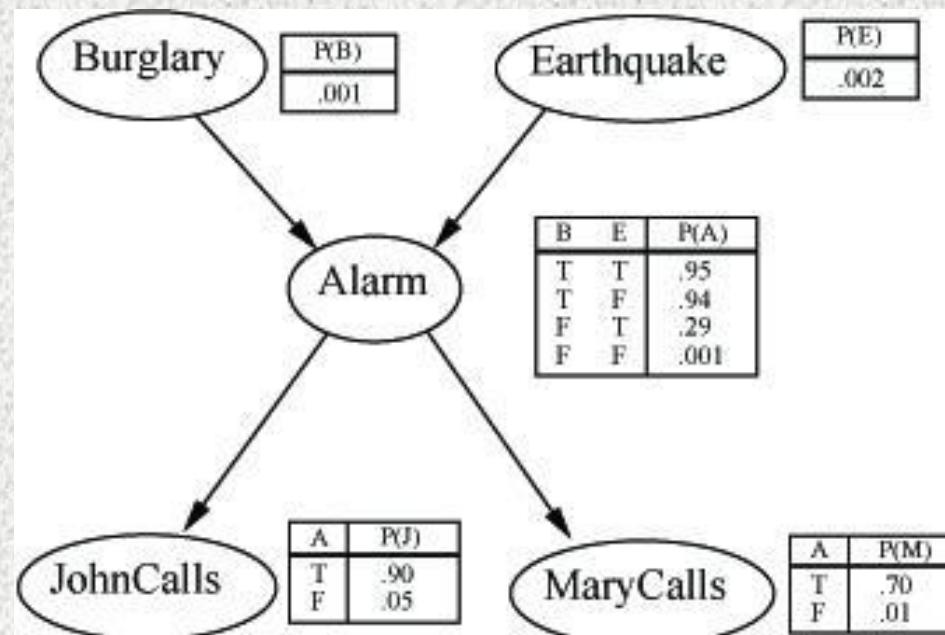
# Belief Networks

- A belief network (Bayes net) represents the dependence between variables.
- Components of a belief network graph:
- Nodes
  - These represent variables
- Links
  - X points to Y if X has a direct influence on Y
- Conditional probability tables
  - Each node has a CPT that quantifies the effects the parents have on the node
- The graph has no directed cycles

# Example

- I'm at work, neighbor John calls to say my alarm is ringing, but neighbor Mary doesn't call. Sometimes it's set off by minor earthquakes. Is there a burglar?
- Variables: **Burglar**, **Earthquake**, **Alarm**, **JohnCalls**, **MaryCalls**

Network topology reflects “causal” knowledge:

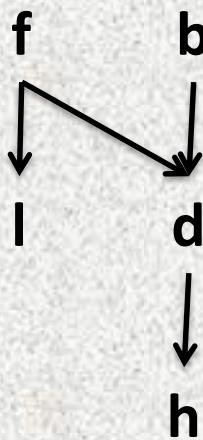


# Example

- Suppose you are going home, and you want to know the probability that the lights are on given the dog is barking and the dog does not have a bowel problem. If the family is out, often the lights are on. The dog is usually in the yard when the family is out and when it has bowel troubles. If the dog is in the yard, it probably barks.
- Use the variables:  
 $f$  = family out  
 $l$  = light on  
 $b$  = bowel problem  
 $d$  = dog out  
 $h$  = hear bark
- There should be a graph with five nodes.

# Example

- We know
  - l is directly influenced by f and is independent of b,d,h given f  
Add link from f to l
  - d is directly influenced by f and b, independent of l and h  
Add link from f to d and b to d
  - h is directly influenced by d, independent of f, l, b, and d  
Add link from d to h



Once we specify the topology (or learn it from data), we need to specify the conditional probability table for each node

$$p(f) = 0.15, 0.85$$

$$p(l|f) = 0.60, 0.40$$

$$p(d|f,b) = 0.99, 0.01$$

$$p(d|-f,b) = 0.97, 0.03$$

$$p(h|d) = 0.70, 0.30$$

$$p(b) = 0.01, 0.99$$

$$p(l|-f) = 0.05, 0.95$$

$$p(d|f,-b) = 0.90, 0.10$$

$$p(d|-f,-b) = 0.30, 0.70$$

$$p(h|-d) = 0.01, 0.99$$

# Example

- Smart Home Example
- [JavaBayes](#)
- [Other Free Bayes Network Software Packages](#)

# The Bad (and Challenging) News

- General querying of Bayes nets is NP-Complete
- The best known algorithm is exponential in the number of variables
- Pathfinder system
  - Heckerman, 1991
  - Diagnostic system for lymph-node diseases
  - 60 diseases, 100 symptoms and test rules
  - 14,000 probabilities
  - 8 hours to determine variables, 35 hours for topology, 40 hours for CPTs
  - Outperforms world experts in diagnosis
  - Being extended to several dozen other medical domains
- [LA Times article on belief networks](#)

# Netica

- Nature nodes, decision nodes, utility nodes
- Links
- Learn values from observations
- Probabilities (percentages) must sum to 100.0
- Compile
- Make observation
- Calculate posterior probabilities
- Netica Smart Home example

# Utility Node

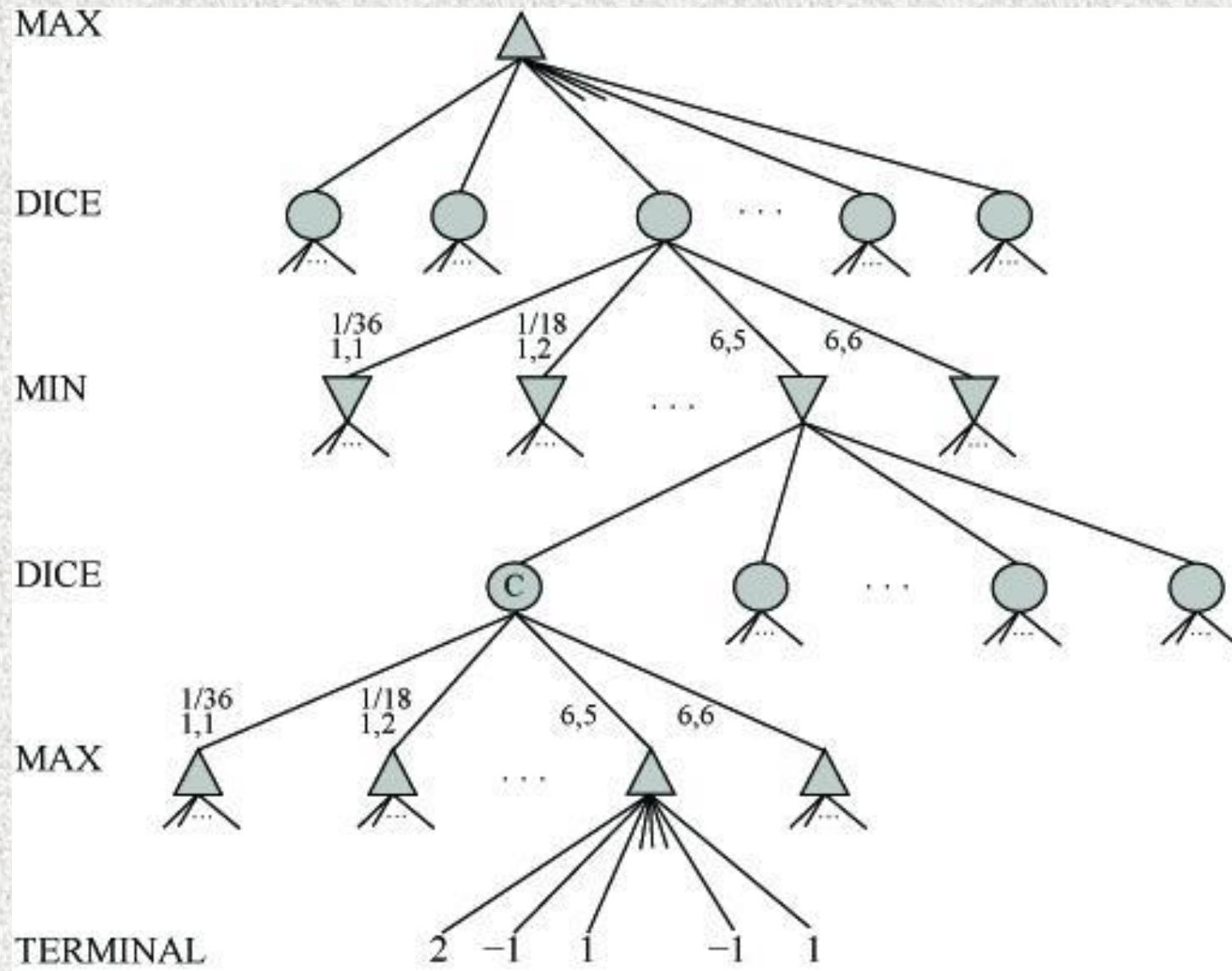
- Expected value of a variable is the sum of the products of the variable values and their probabilities
- $E(\text{Dice roll}) = 1/6*1 + 1/6*2 + 1/6*3 + 1/6*4 + 1/6*5 + 1/6*6 = 3.5$
- Utility of an action is a numeric value indicating the goodness of the outcome of the action (utility can also apply to state)
- If actions have probabilistic outcomes, then expected utility is probability of outcome \* utility of outcome, summed over all possible outcomes

# Nondeterministic Games

- In backgammon, the dice rolls determine legal moves



# Nondeterministic Games

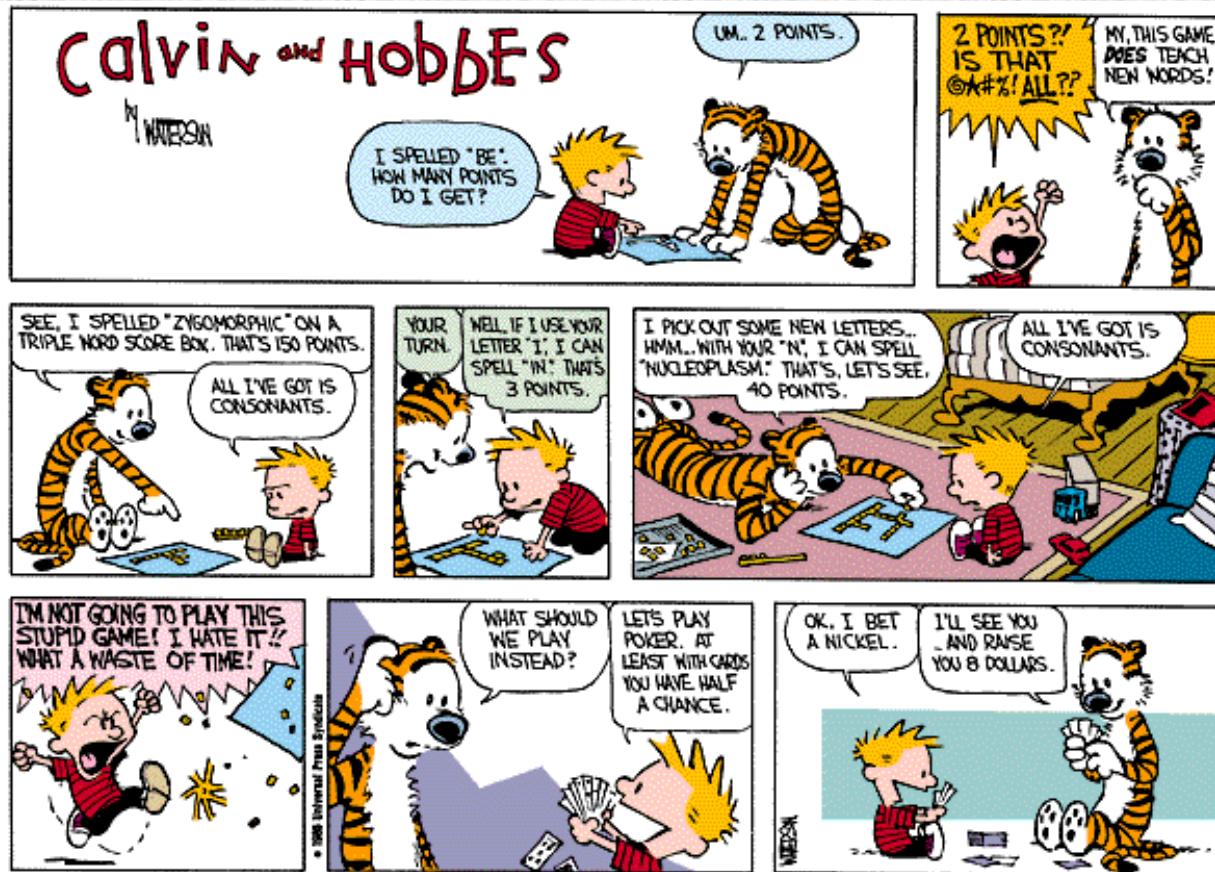


# Nondeterministic Game Algorithm

- Just like Minimax except also handle chance nodes
- Compute ExpectMinimaxValue of successors
  - If n is terminal node, then  $\text{ExpectMinimaxValue}(n) = \text{Utility}(n)$
  - If n is a Max node, then  
$$\text{ExpectMinimaxValue}(n) = \max_{s \in \text{Successors}(n)} \text{ExpectMinimaxValue}(s)$$
  - If n is a Min node, then  
$$\text{ExpectMinimaxValue}(n) = \min_{s \in \text{Successors}(n)} \text{ExpectMinimaxValue}(s)$$
  - If n is a chance node, then  
$$\text{ExpectMinimaxValue}(n) = \sum_{s \in \text{Successors}(n)} P(s) * \text{ExpectMinimaxValue}(s)$$

# Game Theory

- Decision problems in which utility of an action depends on environment AND on actions of other agents
- Assume agents make decisions simultaneously without knowledge of decisions of other agents
- Trading Agent Competition



# Prisoner's Dilemma

- Problem drawn from political science and game theory
- Two players, each with a choice of cooperating with the other or defecting
- Each receives payoff according to payoff matrix for their decision
- When both cooperate, both rewarded equal, intermediate payoff (**reward, R**)
- When one player defects, he/she receives highest payoff (**temptation, T**) and other gets poor payoff (**sucker, S**)
- When both player defect they receive intermediate penalty **P**
- Make problem more interesting by repeating with same players, use history to guide future decisions (iterated prisoner's dilemma)
- Some strategies:
- **Tit For Tat:**
  - Cooperate on first move then do whatever opponent did on previous move, performed best in tournament
- **Golden Rule:**
  - Always cooperate
- **Iron Rule:**
  - Always defect

# Examples

- In the first example, the other player chooses randomly
- [Prisoner's Dilemma Applet](#)
- [Visualize Prisoner's Dilemma](#)

# Fuzzy Logic

- “Precision carries a cost”
  - Boolean logic relies on sharp distinctions
  - 6’ is tall, 5’ 11  $\frac{1}{2}$ ” is not tall
- The tolerance for imprecision feeds human capabilities
  - Example, drive in city traffic
- Fuzzy logic is NOT logic that is fuzzy
  - Logic that is used to describe fuzziness

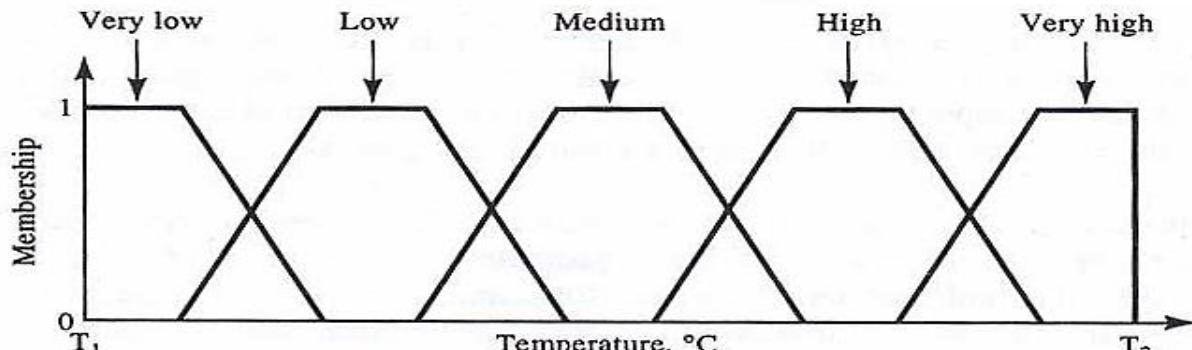
# Fuzzy Logic

- Fuzzy Logic is a multivalued logic that allows intermediate values to be defined between conventional evaluations like yes/no, true/false, black/white, etc.
- Fuzzy Logic was initiated in 1965 by Lotfi A. Zadeh, professor of computer science at the University of California in Berkeley.
- The concept of fuzzy sets is associated with the term ``graded membership".
- This has been used as a model for inexact, vague statements about the elements of an ordinary set.
- Fuzzy logic prevalent in products:
  - Washing machines
  - Video cameras
  - Razors
  - Dishwasher
  - Subway systems

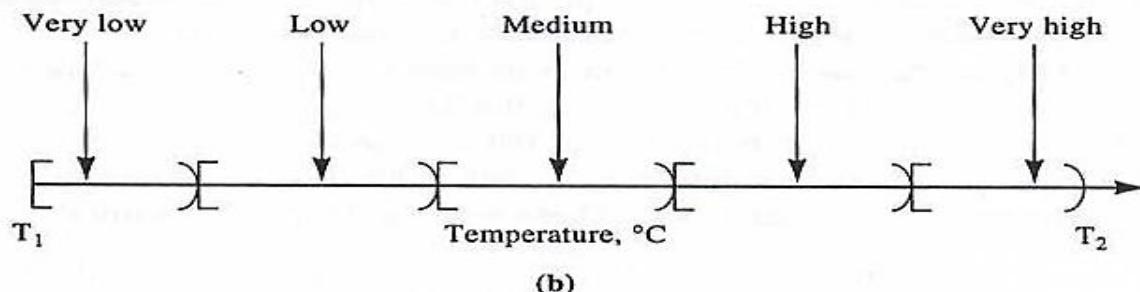
# Fuzzy Sets

- In a fuzzy set the elements have a DEGREE of existence.
- Some typically fuzzy sets are large numbers, tall men, young children, approximately equal to 10, mountains, etc.

# Fuzzy Sets



(a)



(b)

Temperature in the range  $[T_1, T_2]$  conceived as: (a) a fuzzy variable; (b) a traditional (crisp) variable.

# Ordinary Sets

$$f_A(x) = \begin{cases} 1 & \text{If } x \text{ in } A \\ 0 & \text{If } x \text{ not in } A \end{cases}$$

# A Fuzzy Set has Fuzzy Boundaries

- A fuzzy set  $A$  of universe  $X$  is defined by function  $f_A(x)$  called the membership function of set  $A$

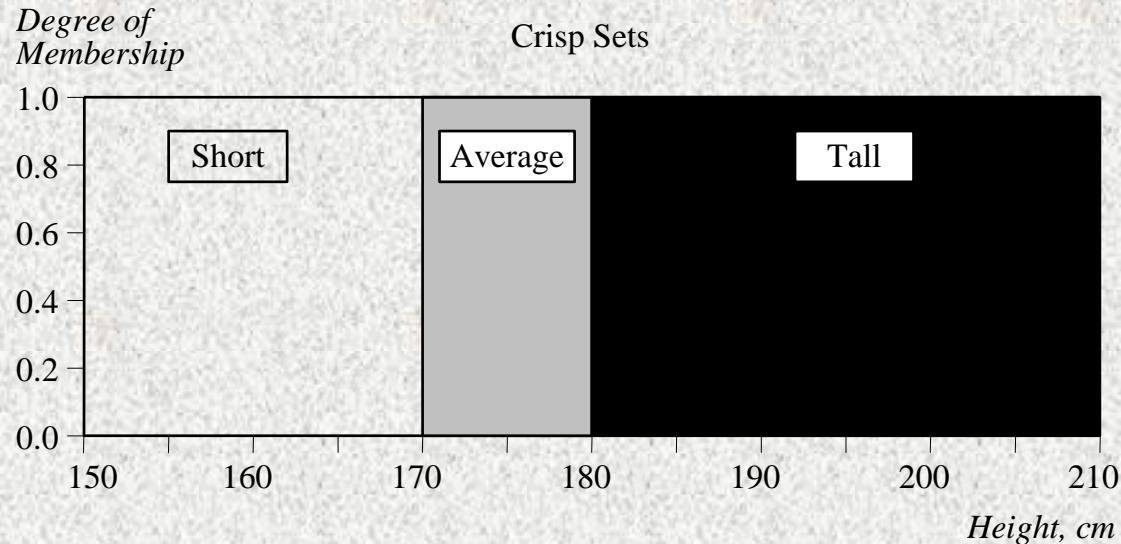
$f_A(x) = \{0, 1\}$ , where       $f_A(x) = 1$  if  $x$  is totally in  $A$ ;  
                                                 $f_A(x) = 0$  if  $x$  is not in  $A$ ;  
                                                 $0 < f_A(x) < 1$  if  $x$  is partly in  $A$ .

- $f_A(x) = i$ , where  $0 \leq i \leq 1$
- If  $f_A(x) > f_A(y)$ , then  $x$  is “more in” the set than  $y$
- If  $f_A(x) = 1$ , then  $x$  in  $A$
- If  $f_A(x) = 0$ , then  $x$  in  $A$
- If  $f_A(x) = \lambda$ , where  $0 < \lambda < 1$ , then  $x \in_{\lambda} A$
- Degree of membership sometimes determined as a function (degree of tall calculated as a function of height)

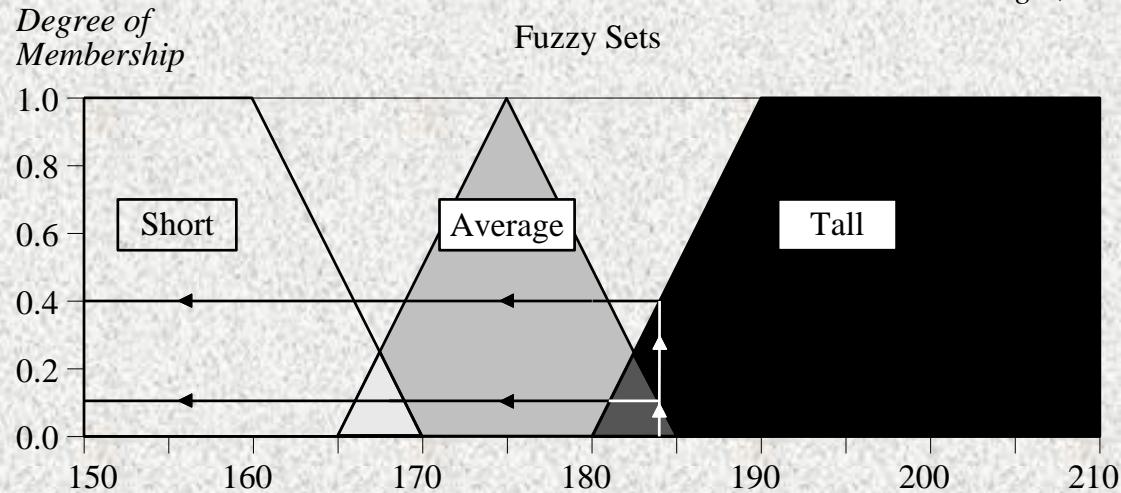
# Fuzzy Sets

| Name   | Height, cm | Degree of Membership |              |
|--------|------------|----------------------|--------------|
|        |            | <i>Crisp</i>         | <i>Fuzzy</i> |
| Chris  | 208        | 1                    | 1.00         |
| Mark   | 205        | 1                    | 1.00         |
| John   | 198        | 1                    | 0.98         |
| Tom    | 181        | 1                    | 0.82         |
| David  | 179        | 0                    | 0.78         |
| Mike   | 172        | 0                    | 0.24         |
| Bob    | 167        | 0                    | 0.15         |
| Steven | 158        | 0                    | 0.06         |
| Bill   | 155        | 0                    | 0.01         |
| Peter  | 152        | 0                    | 0.00         |

# Fuzzy Set Representation



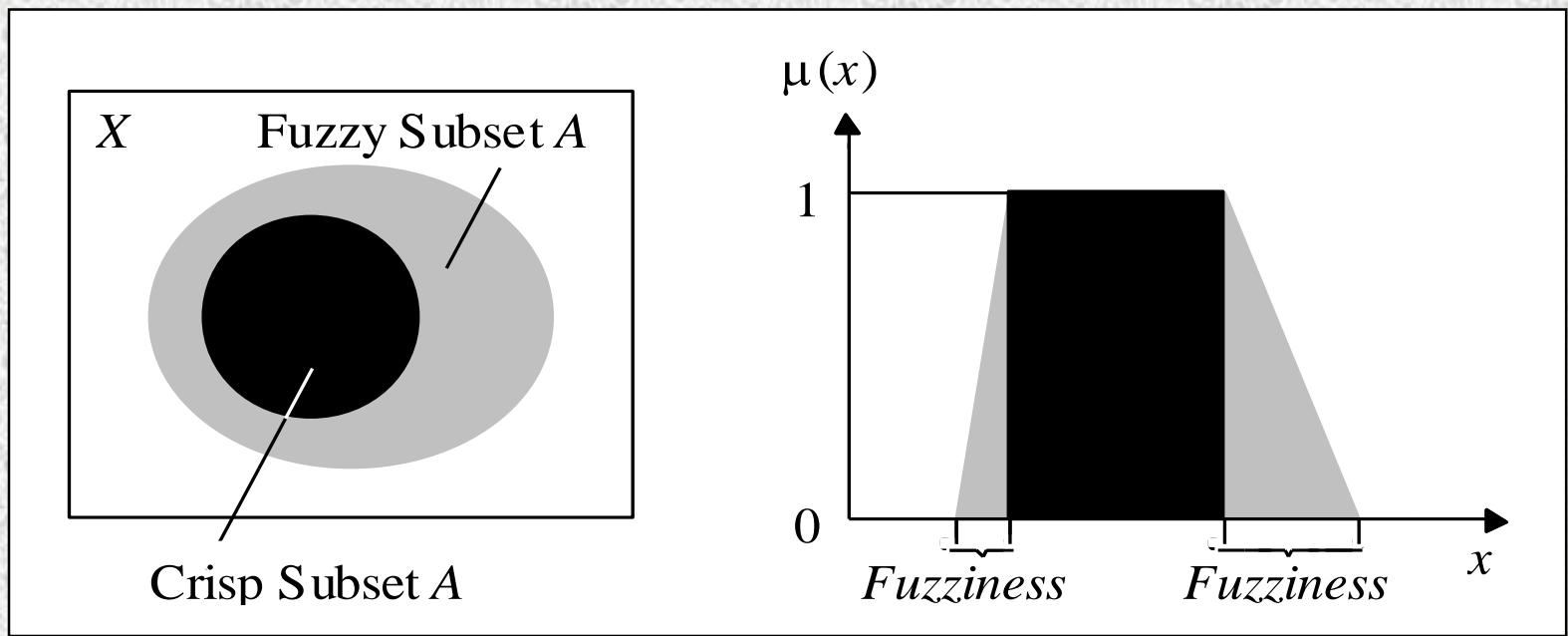
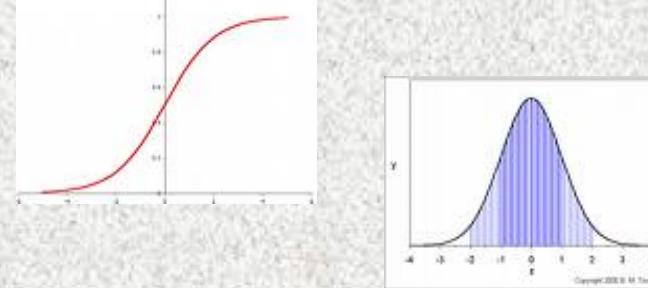
A man who is 184 cm tall is a member of the *average* men set with a degree of membership of 0.1



At the same time, he is also a member of the *tall* men set with a degree of 0.4.

# Fuzzy Set Representation

- Typical functions that can be used to represent a fuzzy set are
  - Sigmoid
  - Gaussian
  - Linear fit (preferred because low computation cost)



# Linguistic Variables and Hedges

- In fuzzy expert systems, linguistic variables are used in fuzzy rules. For example:

IF            wind                        is **strong**  
THEN        sailing                      is **good**

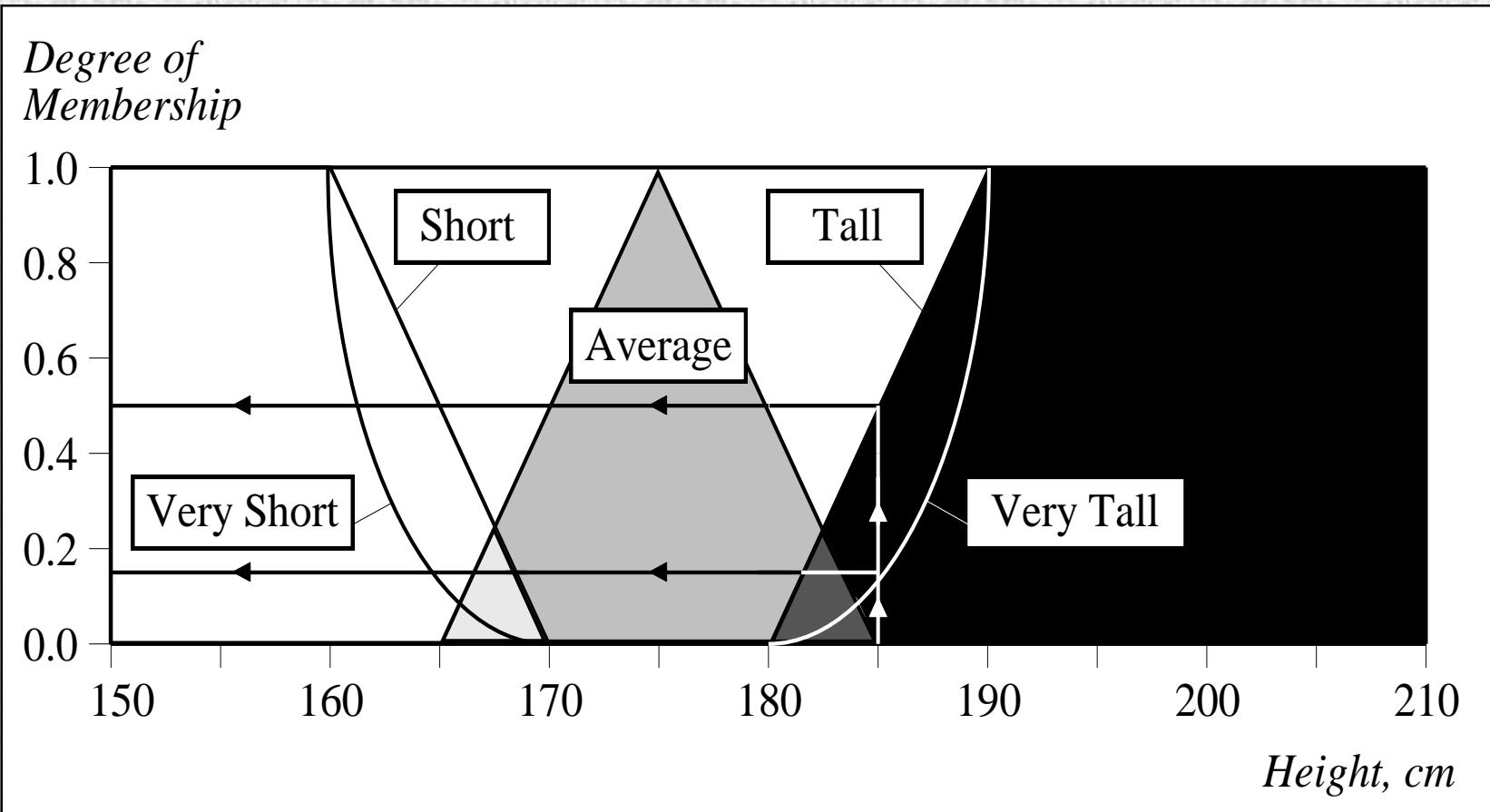
IF            project\_duration    is **long**  
THEN        completion\_risk      is **high**

IF            speed                        is **slow**  
THEN        stopping\_distance    is **short**

# Linguistic Variables and Hedges

- The range of possible values of a linguistic variable represents the universe of discourse of that variable.
  - Example, speed
  - University of discourse might have range 0 .. 220 mph
  - Fuzzy subsets might be *very slow*, *slow*, *medium*, *fast*, and *very fast*.
- Hedges
  - Modify the shape of fuzzy sets
  - Adverbs such as *very*, *somewhat*, *quite*, *more or less* and *slightly*.

# Linguistic Variables and Hedges



# Fuzzy Set Relations

- One set A is a **subset** of set B if for every  $x$ ,  $f_A(x) \leq f_B(x)$   
Sets A and B are equal if for every element  $x$ ,  $f_A(x) = f_B(x)$ .
- OR / Union
  - $A \cup B$  is the smallest fuzzy subset of X containing both A and B, and is defined by  $f_{A \cup B} = \max(f_A(x), f_B(x))$
- AND / Intersection
  - The intersection  $A \cap B$  is the largest fuzzy subset of X contained in both A and B, and is defined by  $f_{A \cap B}(x) = \min(f_A(x), f_B(x))$
- NOT:  $\text{truth}(\sim x) = 1.0 - \text{truth}(x)$
- IMPLICATION:  $A \rightarrow B = \sim A \vee B$ , so  
 $\text{truth}(A \rightarrow B) = \max(1.0 - f_A(x), f_B(x))$

# Examples

- Fuzzy Logic Washing Machine
- Fuzzy Logic Rice Cooker
- Fuzzy Logic Barcode Scanner
- Fuzzy Logic Blender
- Fuzzy Logic Shampoo
- Fuzzy Logic Monitor

# CptS 440 / 540

# Artificial Intelligence

Machine Learning

# Machine Learning

- What is learning?

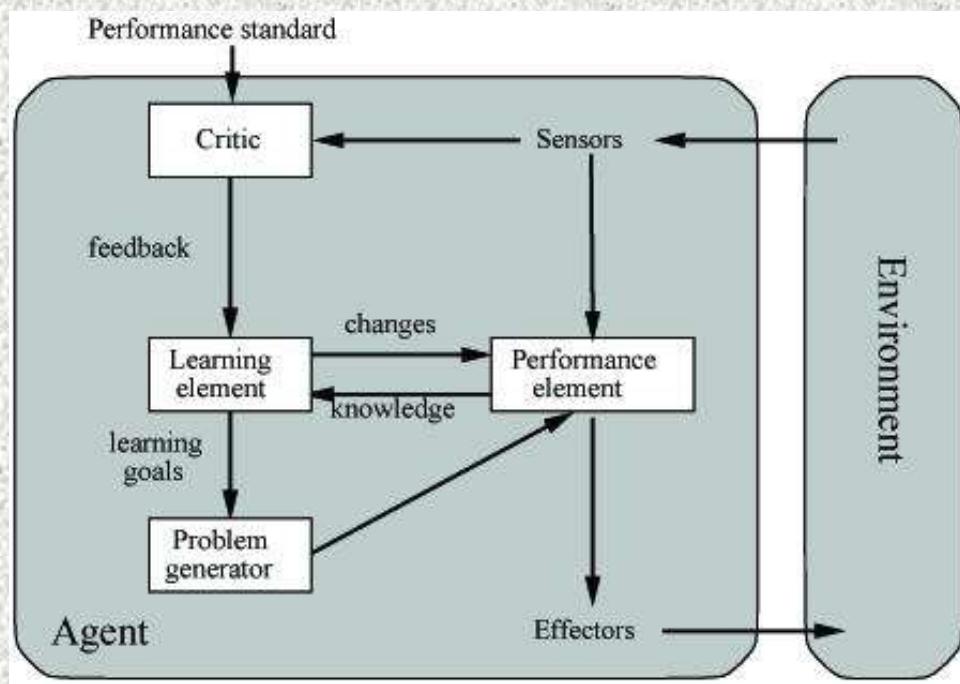
# Definitions

- **Webster**
  - To gain knowledge or understanding of or skill in by study, instruction or experience; memorize; to acquire knowledge or skill in a behavioral tendency; discovery, to obtain knowledge of for the first time
- **Simon**
  - Any process by which a system improves its performance
- So far we have programmed knowledge into the agent (expert rules, probabilities, search space representations), but an autonomous agent should acquire this knowledge on its own.
- Machine Learning will make this possible



# A General Model of Learning Agents

- **Learning Element**
  - Adds knowledge, makes improvement to system
- **Performance Element**
  - Performs task, selects external actions
- **Critic**
  - Monitors results of performance, provides feedback to learning element
- **Problem Generator**
  - Actively suggests experiments, generates examples to test
- **Performance Standard**
  - Method / standard of measuring performance



# The Learning Problem

- Learning = Improving with experience at some task
  - Improve over task T
  - With respect to performance measure P
  - Based on experience E
- Example: Learn to play checkers (Chinook)
  - T: Play checkers
  - P: % of games won in world tournament
  - E: opportunity to play against self
- Example: Learn to Diagnose Patients
  - T: Diagnose patients
  - P: Percent of patients correctly diagnosed
  - E: Pre-diagnosed medical histories of patients

# Categories of Learning

- Learning by being told
  - Learning by examples / Supervised learning
  - Learning by discovery / Unsupervised learning
  - Learning by experimentation / Reinforcement learning
- [Syskill and Webert Perform Web Page Rating](#)
  - [Example of supervised learning](#)
  - [Example of supervised learning](#)
  - [Example of unsupervised learning](#)



# Learning From Examples

- Learn general concepts or categories from examples
- Learn a task (drive a vehicle, win a game of backgammon)
- Examples of objects or tasks are gathered and stored in a database
- Each example is described by a set of **attributes** or **features**
- Each example used for training is classified with its correct label (chair, not chair, horse, not horse, 1 vs. 2 vs. 3, good move, bad move, etc.)
- The machine learning program learns general concept description from these specific examples
- The ML program should be applied to classify or perform tasks *never before seen* from learned concept

# Learning From Examples

- First algorithm: naïve Bayes classifier
- D is training data
  - Each data point is described by attributes  $a_1..a_n$
- Learn mapping from data point to a class value
  - Class values  $v_1..v_j$
- We are searching through the space of possible concepts
  - Functions that map data to class value

# Supervised Learning Algorithm – Naïve Bayes

1. For each hypothesis  $h \in H$ , calculate the posterior probability

$$P(h | D) = \frac{P(D | h)P(h)}{P(D)}$$

2. Output the hypothesis  $h_{MAP}$  (maximum a posteriori hypothesis) with the highest posterior probability

$$h_{MAP} = \arg \max_{h \in H} P(h | D)$$

# NBC Definition

- Assume target function is  $f:D \rightarrow V$ , where each instance  $d$  is described by attributes  $\langle a_1, a_2, \dots, a_n \rangle$ . The most probable value of  $f(d)$  is

$$v_{MAP} = \arg \max_{v_j \in V} P(v_j | a_1, a_2, \dots, a_n)$$

$$v_{MAP} = \arg \max_{v_j \in V} \frac{P(a_1, a_2, \dots, a_n) P(v_j)}{P(a_1, a_2, \dots, a_n)}$$

$$= \arg \max_{v_j \in V} P(a_1, a_2, \dots, a_n | v_j) P(v_j)$$

# NB Assumption

- Assume that the attributes are independent with respect to the class. The result is

$$P(a_1, a_2, \dots, a_n \mid v_j) = \prod_i P(a_i \mid v_j)$$

- which yields the NBC

$$v_{NB} = \arg \max_{v_j \in V} P(v_j) \prod_i P(a_i \mid v_j)$$

# Using NBC

- Training
  - For each target value (class value)  $v_j$  estimate  $P(v_j)$
- For each attribute value  $a_i$  of each attribute  $a$  estimate  $P(a_i | v_j)$
- Classify new instance

$$v_{NB} = \arg \max_{v_j \in V} P(v_j) \prod_{a_i \in d} P(a_i | v_j)$$

# PlayTennis Training Examples

| Day | Outlook  | Temperature | Humidity | Wind   | PlayTennis |
|-----|----------|-------------|----------|--------|------------|
| D1  | Sunny    | Hot         | High     | Weak   | No         |
| D2  | Sunny    | Hot         | High     | Strong | No         |
| D3  | Overcast | Hot         | High     | Weak   | Yes        |
| D4  | Rain     | Mild        | High     | Weak   | Yes        |
| D5  | Rain     | Cool        | Normal   | Weak   | Yes        |
| D6  | Rain     | Cool        | Normal   | Strong | No         |
| D7  | Overcast | Cool        | Normal   | Strong | Yes        |
| D8  | Sunny    | Mild        | High     | Weak   | No         |
| D9  | Sunny    | Cool        | Normal   | Weak   | Yes        |
| D10 | Rain     | Mild        | Normal   | Weak   | Yes        |
| D11 | Sunny    | Mild        | Normal   | Strong | Yes        |
| D12 | Overcast | Mild        | High     | Strong | Yes        |
| D13 | Overcast | Hot         | Normal   | Weak   | Yes        |
| D14 | Rain     | Mild        | High     | Strong | No         |

# PlayTennis Example

- Target values are Yes and No
- Assume examples 1-12 comprise training data
- $P(\text{Yes}) = 8/12$ ,  $P(\text{No}) = 4/12$
- Now we want to determine class for example 13  
(Outlook=Overcast, Temperature=Hot, Humidity=Normal,  
Wind=Weak)
- Determine which value is larger
- $$P(\text{Yes}) * P(\text{Overcast} | \text{Yes}) * P(\text{Hot} | \text{Yes}) * P(\text{Normal} | \text{Yes}) * P(\text{Weak} | \text{Yes}) \\ = 8/12 * 2/8 * 1/8 * 5/8 * 5/8 = 0.00814$$
- $$P(\text{No}) * P(\text{Overcast} | \text{No}) * P(\text{Hot} | \text{No}) * P(\text{Normal} | \text{No}) * P(\text{Weak} | \text{No}) \\ = 4/12 * 0 * 2/4 * 1/4 * 2/4 = 0.0$$
- Answer is Yes

# NBC Subtleties

- Conditional independence is often violated...
  - ...but it works surprisingly well anyway. We do not need the actual probability, just the class that yields the largest value.
- Some attribute values may not appear in any training examples.
  - Use small non-zero value

# NBC For Text Classification

- Uses
  - Filter spam
  - Classify web pages by topic
  - Categorize email
- NBC is very effective for this application.
- What attributes shall we use?
- Do we care about word position?
- Joachims performed experiment with 20 newsgroups, 1000 articles per class, 2/3 training 1/3 testing, achieved 89% accuracy

# Prediction Problems

- Customer purchase behavior

Customer103; (time=0)

Sex: M  
Age: 53  
Income: \$50k  
Own House: Yes  
MS Products: Word  
Computer: 386 PC  
Purchase Excel?: ?

Customer103; (time=1)

Sex: M  
Age: 53  
Income: \$50k  
Own House: Yes  
MS Products: Word  
Computer: Pentium  
Purchase Excel?: ?

Customer103; (time=t<sub>n</sub>)

Sex: M  
Age: 53  
Income: \$50k  
Own House: Yes  
MS Products: Word  
Computer: Pentium  
Purchase Excel?: Yes

# Prediction Problems

- Customer retention

Customer103; (time=0)

Sex: M  
Age: 53  
Income: \$50k  
Own House: Yes  
MS Products: Word  
Computer: 386 PC  
Purchase Excel?: ?

Customer103; (time=1)

Sex: M  
Age: 53  
Income: \$50k  
Own House: Yes  
MS Products: Word  
Computer: Pentium  
Purchase Excel?: ?

Customer103; (time=t<sub>n</sub>)

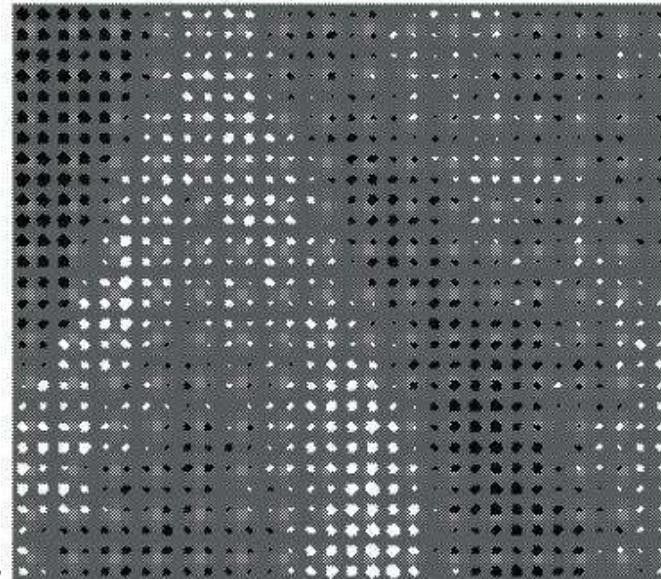
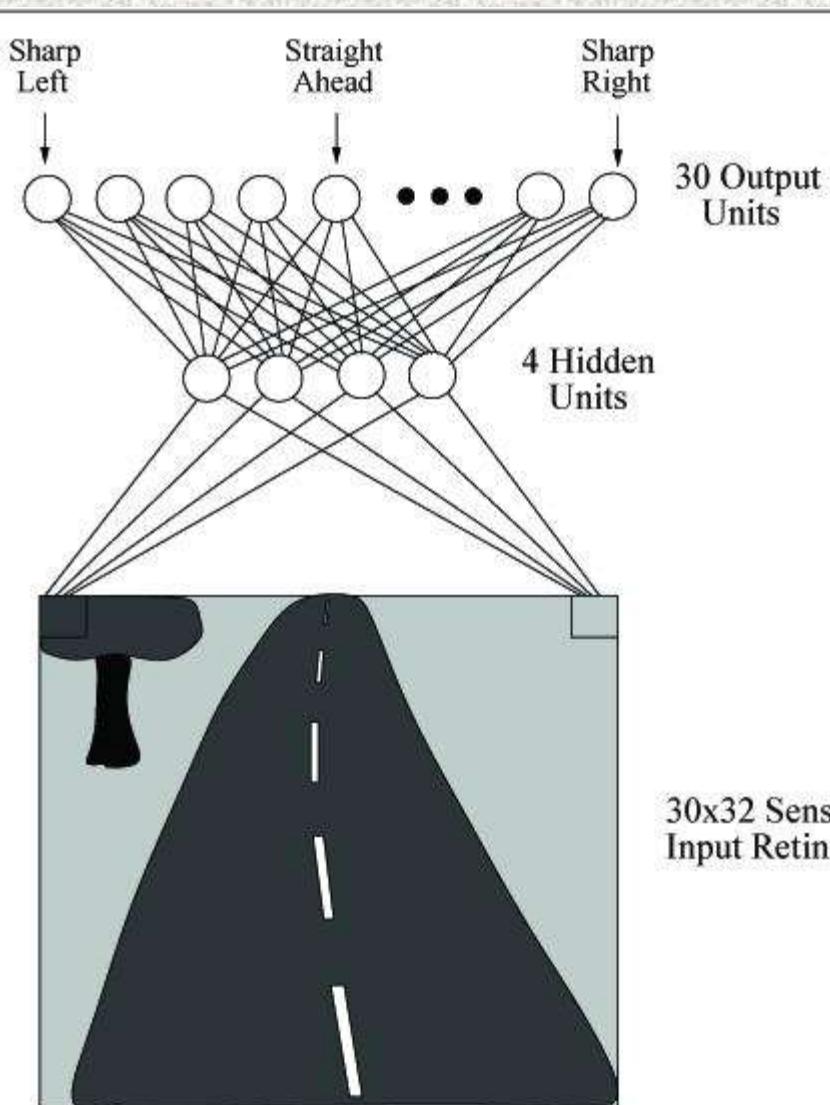
Sex: M  
Age: 53  
Income: \$50k  
Own House: Yes  
MS Products: Word  
Computer: Pentium  
Purchase Excel?: Yes

# Prediction Problems

| Product72: (time=t0)     | Product72: (time=t1)     | Product72: (time=tn)      |
|--------------------------|--------------------------|---------------------------|
| Stage: mix               | Stage: cook              | Stage: cool               |
| Mixing-speed: 60rpm      | Temperature: 325         | Fan-speed: medium         |
| Viscosity: 1.3           | Viscosity: 3.2           | Viscosity: 1.3            |
| Fat content: 15%         | Fat content: 12%         | Fat content: 12%          |
| Density: 2.8             | Density: 1.1             | Density: 1.2              |
| Spectral peak: 2800      | Spectral peak: 3200      | Spectral peak: 3100       |
| Product underweight?: ?? | Product underweight?: ?? | Product underweight?: Yes |
| ...                      | ...                      | ...                       |

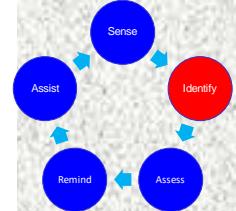
# Prediction Problems

- Problems too difficult to program by hand



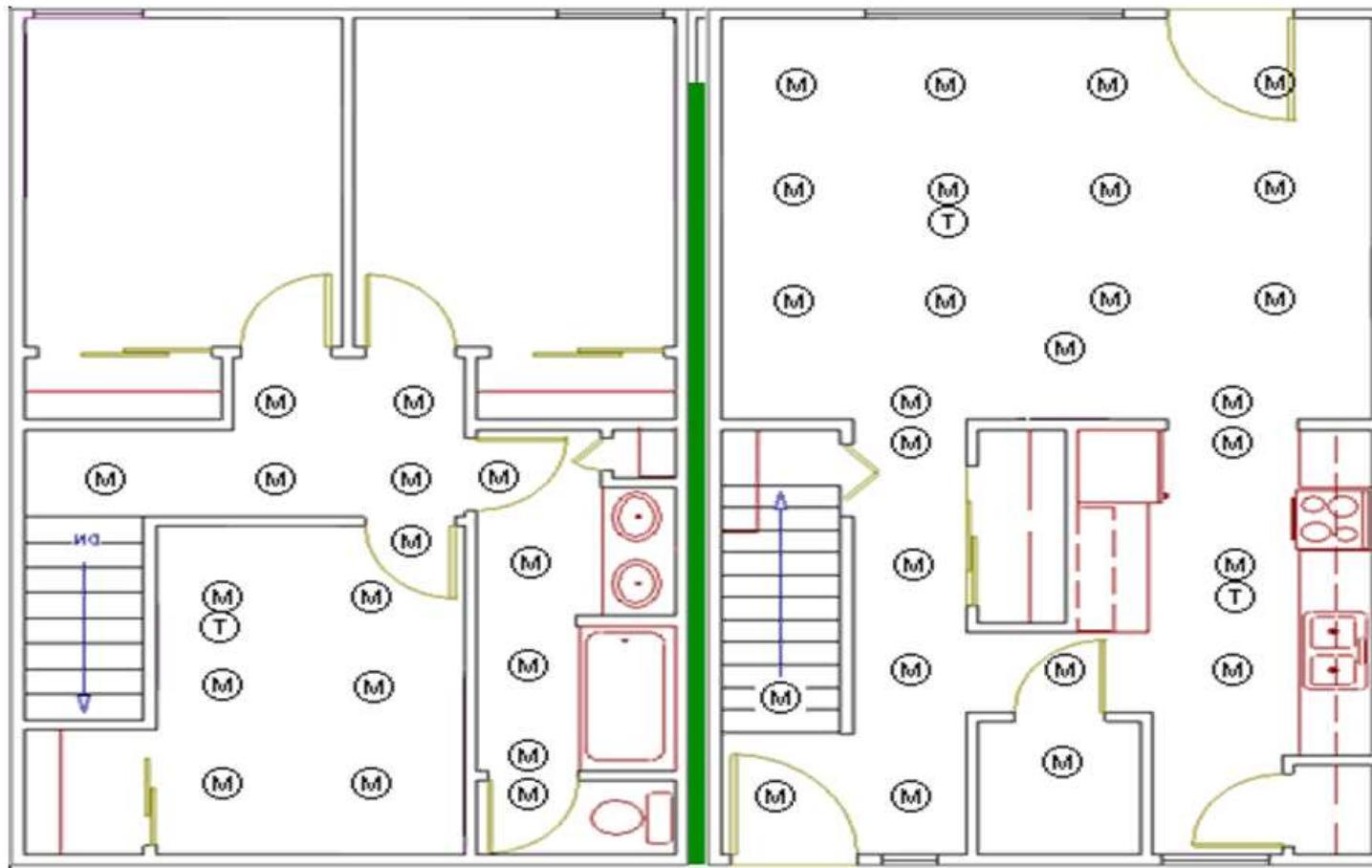


# Identify



sensor ID |  
date / time | reading

---



# Prediction Problems

- Software that customizes to user

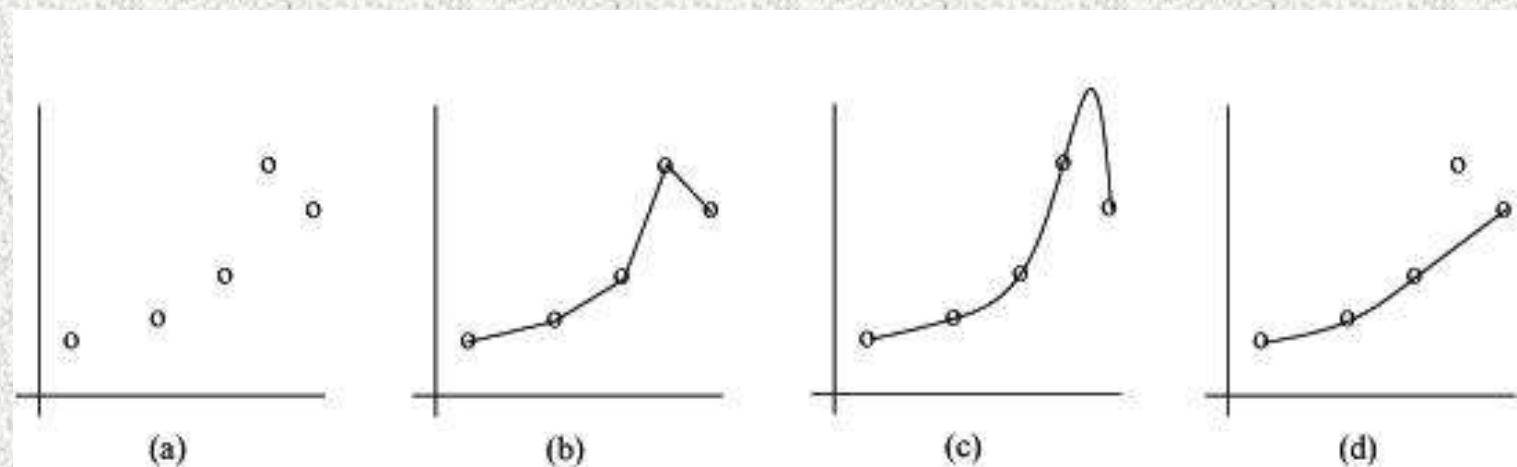


# Inductive Learning Hypothesis

- Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples

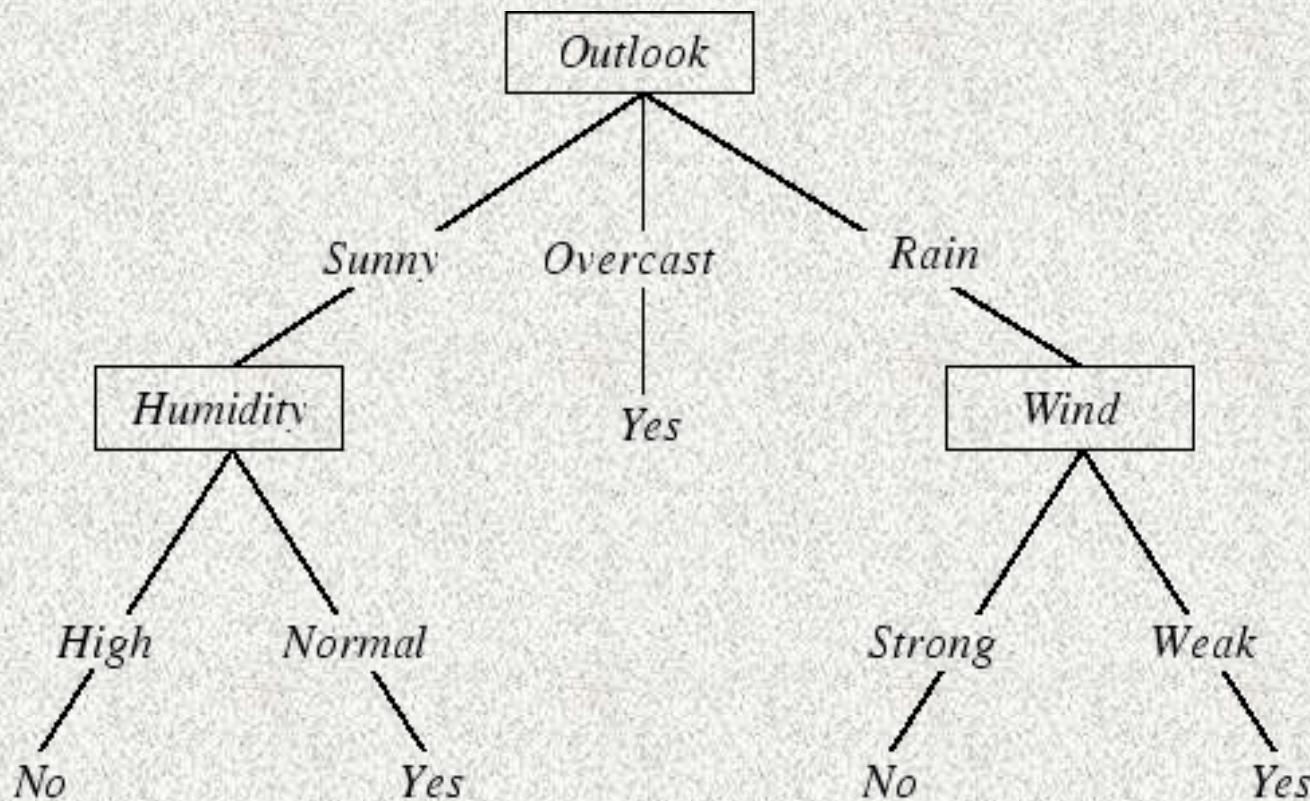
# Inductive Bias

- There can be a number of hypotheses consistent with training data
- Each learning algorithm has an **inductive bias** that imposes a preference on the space of all possible hypotheses



# Decision Trees

- A decision tree takes a description of an object or situation as input, and outputs a yes/no "decision".
- Can also be used to output greater variety of answers.
- Here is a decision tree for the concept *PlayTennis*



# Decision Tree Representation

- Each internal node tests an attribute
- Each branch corresponds to attribute value
- Each leaf node assigns a classification

# When to Consider Decision Trees

- Instances describable by attribute-value pairs
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possibly noisy training data

# Inducing Decision Trees

- Each **example** is described by the values of the attributes and the value of the goal predicate (Yes/No)
- The value of the goal predicate is called the **classification** of the example
- If the classification is true (Yes), this is a **positive** example, otherwise this is a **negative** example
- The complete set of examples is called the **training set**

# Decision Tree Learning

- Any concept that can be expressed as a propositional statement can be expressed using a decision tree
- No type of representation is efficient for all kinds of functions  
How represent of m of n?
- Once we know how to use a decision tree, the next question is, how do we automatically construct a decision tree?
- One possibility: search through the space of all possible decision trees
  - All possible n features at root
  - For each root, n-1 possible features at each child
  - ...
  - Keep the hypotheses that are consistent with training examples
  - Among these, keep one that satisfies bias
- Too slow!
- Another possibility: construct one path for each positive example
  - Not very general
- Another possibility: find smallest decision tree consistent with all examples
  - Inductive Bias: Ockham's Razor

# Top-Down Induction of Decision Trees

1. At each point, decide which attribute to use as next test in the tree
2. Attribute splits data based on answer to question
  - Each answer forms a separate node in decision tree
  - Each node is the root of an entire sub-decision tree problem, possibly with fewer examples and one fewer attribute than its parent

# Four Cases To Consider

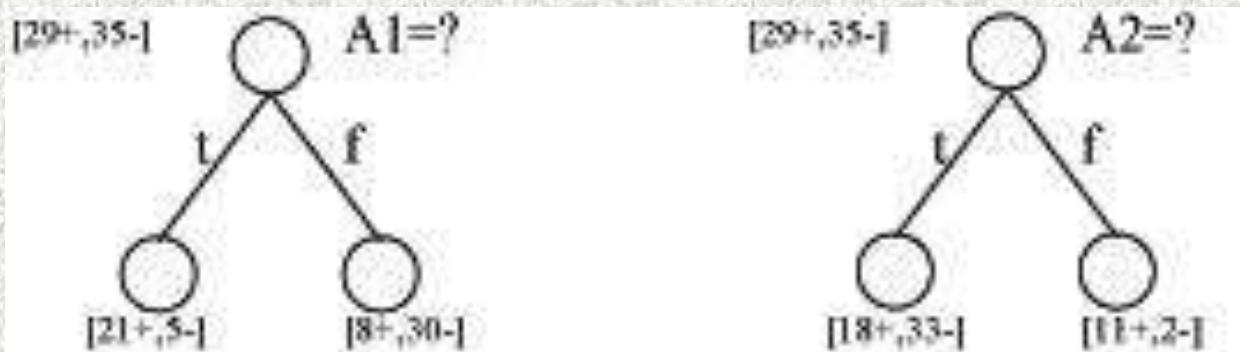
1. If both + and -, choose best attribute to split
2. If all + (or -), then we are done
3. If no examples, no examples fit this category, return default value (calculate using majority classification from parent)
4. If no attributes left, then there are inconsistencies, called **noise**

We can use a majority vote to label the node.



# Which Attribute Is Best?

- Pick one that provides the highest expected amount of **information**
- **Information Theory** measures information content in bits
- One bit of information is enough to answer a yes/no question about which one has no idea

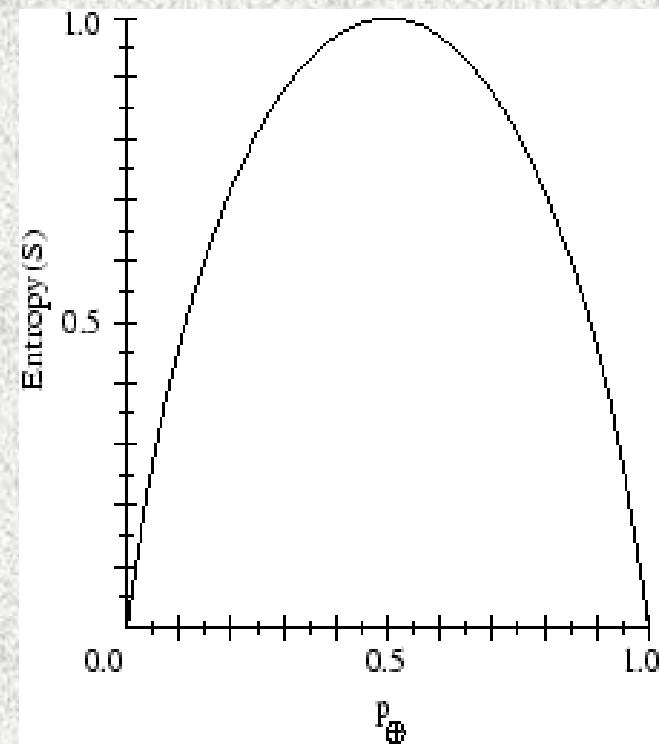


# Entropy

- $S$  is a sample of training examples
- $p_+$  is the proportion of positive examples in  $S$
- $p_-$  is the proportion of negative examples in  $S$
- Entropy measure the impurity of  $S$
- $\text{Entropy}(S) = \text{expected \#bits needed to encode class (+ or -) of randomly drawn element of } S \text{ (using optimal, shortest-length code)}$

# Entropy

- Information theory: optimal length code assigns  $-\log_2 p$  bits to message of probability  $p$
- Expected number of bits to encode + or - of random element of  $S$ 
$$p_+(-\log_2 p_+) + p_-(-\log_2 p_-)$$
- $\text{Entropy}(S) = -p_+\log_2 p_+ - p_-\log_2 p_-$



# Information Content

- Entropy is also called the **information content I** of an actual answer
- Suppose you are sending messages to someone, could send several possible messages. How many bits are needed to distinguish which message is being sent?
- If  $P(\text{message}) = 1.0$ , don't need any bits (pure node, all one class).
- For other probability distributions, use shorter encodings for higher-probability classes.  $P = 0.8, 0.2$  requires fewer bits on average than  $P = 0.5, 0.5$ .
- Log of a fraction is always negative, so term is multiplied by -1.
- If possible answers  $v_i$  have probabilities  $P(v_i)$  then the information content I of actual answer is given by

$$I((P(v_1), \dots, P(v_n))) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$

$$I(1/2, 1/2) = -(1/2 \log_2 1/2) - (1/2 \log_2 1/2) = 1 \text{ bit}$$

$$I(0.01, 0.99) = 0.08 \text{ bits}$$

# Information Theory and Decision Trees

- What is the correct classification?
- Before splitting, estimate of probabilities of possible answers calculated as proportions of positive and negative examples
- If training set has  $p$  positive examples and  $n$  negative examples, then the information contained in a **correct** answer is

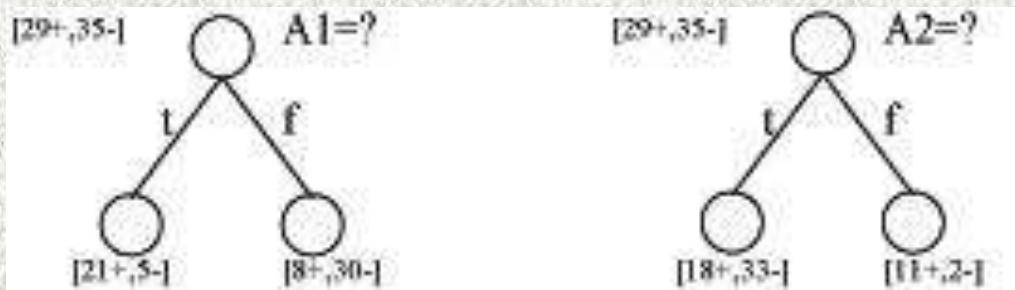
$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right)$$

- Splitting on a single attribute does not usually answer entire question, but it gets us closer
- How much closer?

# Information Gain

- $\text{Gain}(S, A) = \text{expected reduction in entropy due to sorting on } A$
- Look at decrease in information of correct answer after split

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

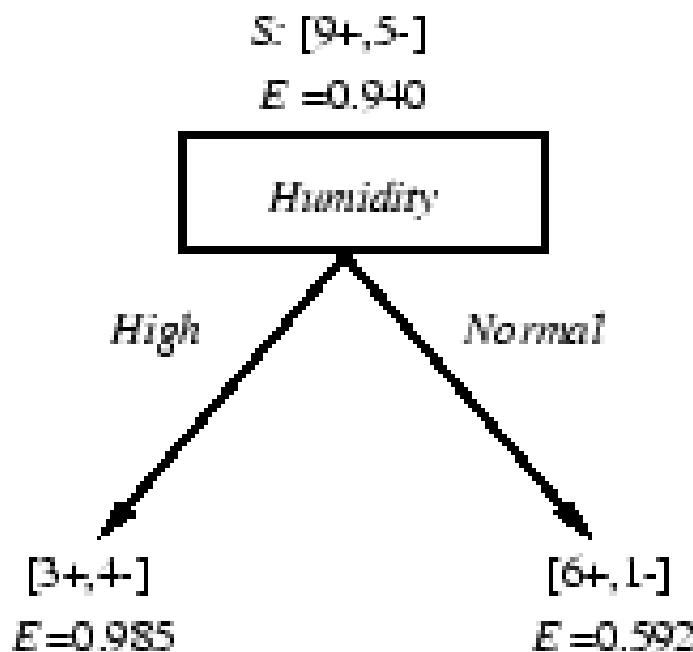


# Training Examples

| Day | Outlook  | Temperature | Humidity | Wind   | PlayTennis |
|-----|----------|-------------|----------|--------|------------|
| D1  | Sunny    | Hot         | High     | Weak   | No         |
| D2  | Sunny    | Hot         | High     | Strong | No         |
| D3  | Overcast | Hot         | High     | Weak   | Yes        |
| D4  | Rain     | Mild        | High     | Weak   | Yes        |
| D5  | Rain     | Cool        | Normal   | Weak   | Yes        |
| D6  | Rain     | Cool        | Normal   | Strong | No         |
| D7  | Overcast | Cool        | Normal   | Strong | Yes        |
| D8  | Sunny    | Mild        | High     | Weak   | No         |
| D9  | Sunny    | Cool        | Normal   | Weak   | Yes        |
| D10 | Rain     | Mild        | Normal   | Weak   | Yes        |
| D11 | Sunny    | Mild        | Normal   | Strong | Yes        |
| D12 | Overcast | Mild        | High     | Strong | Yes        |
| D13 | Overcast | Hot         | Normal   | Weak   | Yes        |
| D14 | Rain     | Mild        | High     | Strong | No         |

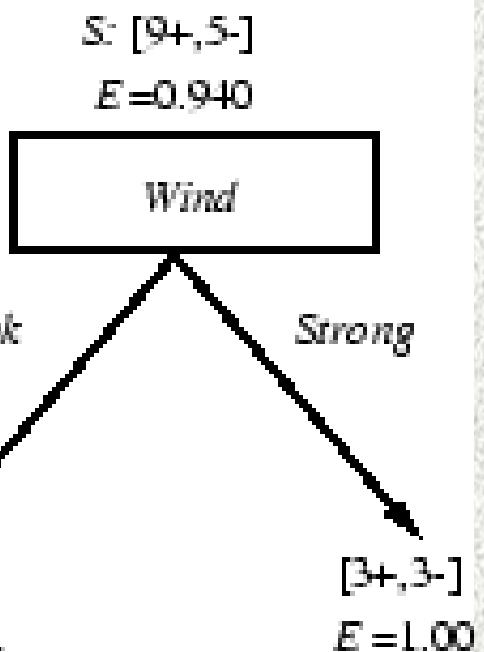
# Selecting the Next Attribute

Which attribute is the best classifier?



*Gain (S, Humidity)*

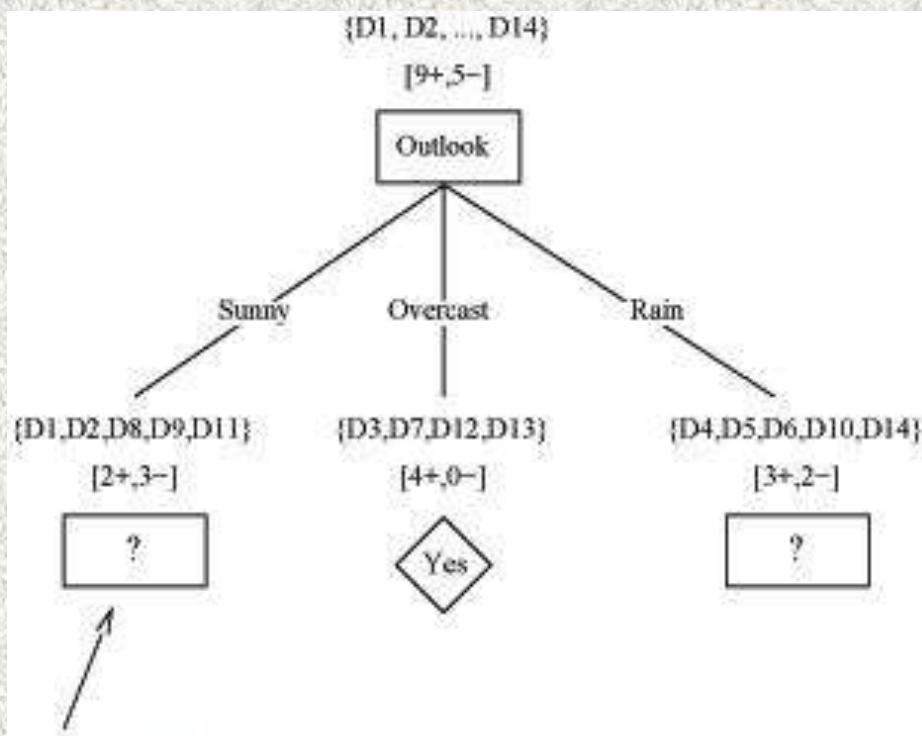
$$= .940 - (7/14).985 - (7/14).592 \\ = .151$$



*Gain (S, Wind)*

$$= .940 - (3/14).811 - (6/14)1.0 \\ = .048$$

# Partially Learned Tree



Which attribute should be tested here?

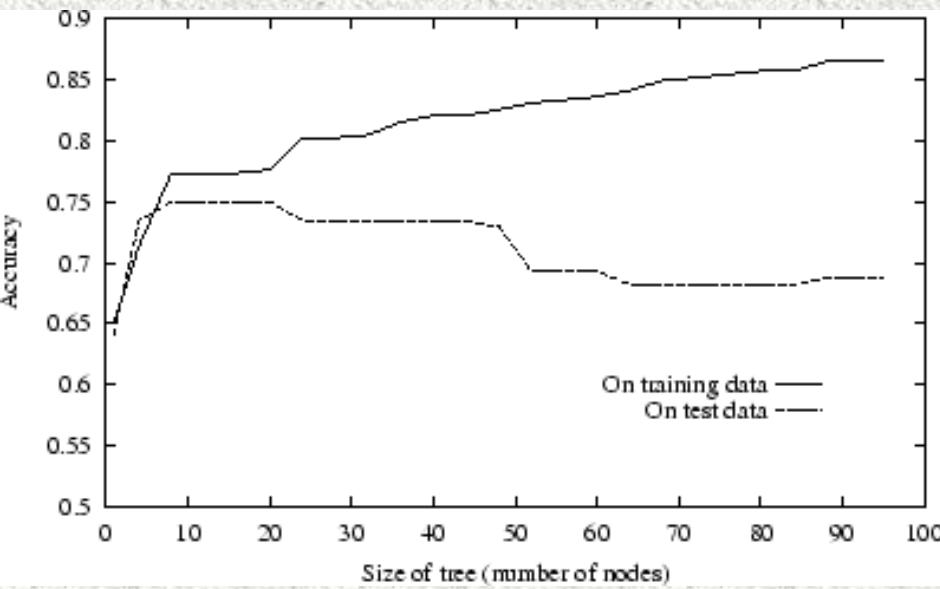
$$S_{\text{sunny}} = \{D1, D2, D8, D9, D11\}$$

$$\text{Gain}(S_{\text{sunny}}, \text{Humidity}) = .970 - (3/5)0.0 - (2/5)0.0 = .970$$

$$\text{Gain}(S_{\text{sunny}}, \text{Temperature}) = .970 - (2/5)0.0 - (2/5)1.0 - (1/5)0.0 = .570$$

$$\text{Gain}(S_{\text{sunny}}, \text{Wind}) = .970 - (2/5)1.0 - (3/5).918 = .019$$

# Danger: Overfit



One solution:  
prune decision tree

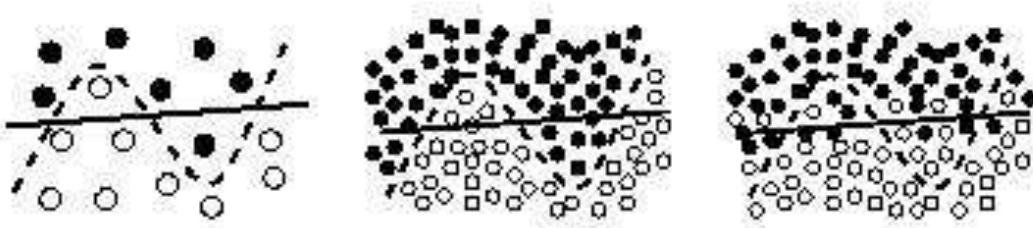


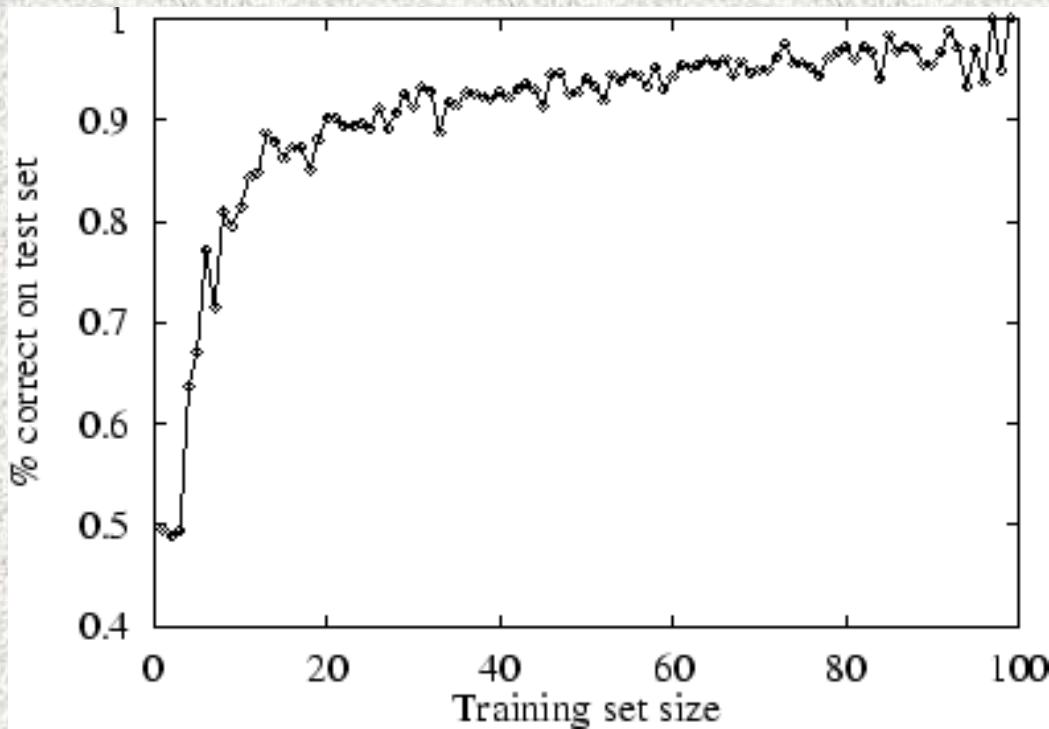
Fig. 1. Illustration of the overfitting dilemma: Given only a small sample (left) either, the solid or the dashed hypothesis might be true, the dashed one being more complex, but also having a smaller training error. Only with a large sample we are able to see which decision reflects the true distribution more closely. If the dashed hypothesis is correct the solid would underfit (middle); if the solid were correct the dashed hypothesis would overfit (right).

# How Do We Prune a Decision Tree?

- Delete a decision node
- This causes entire subtree rooted at node to be removed
- Replace by leaf node, and assign it by majority vote
- Reduced error pruning: remove nodes as long as performance improves on validation set

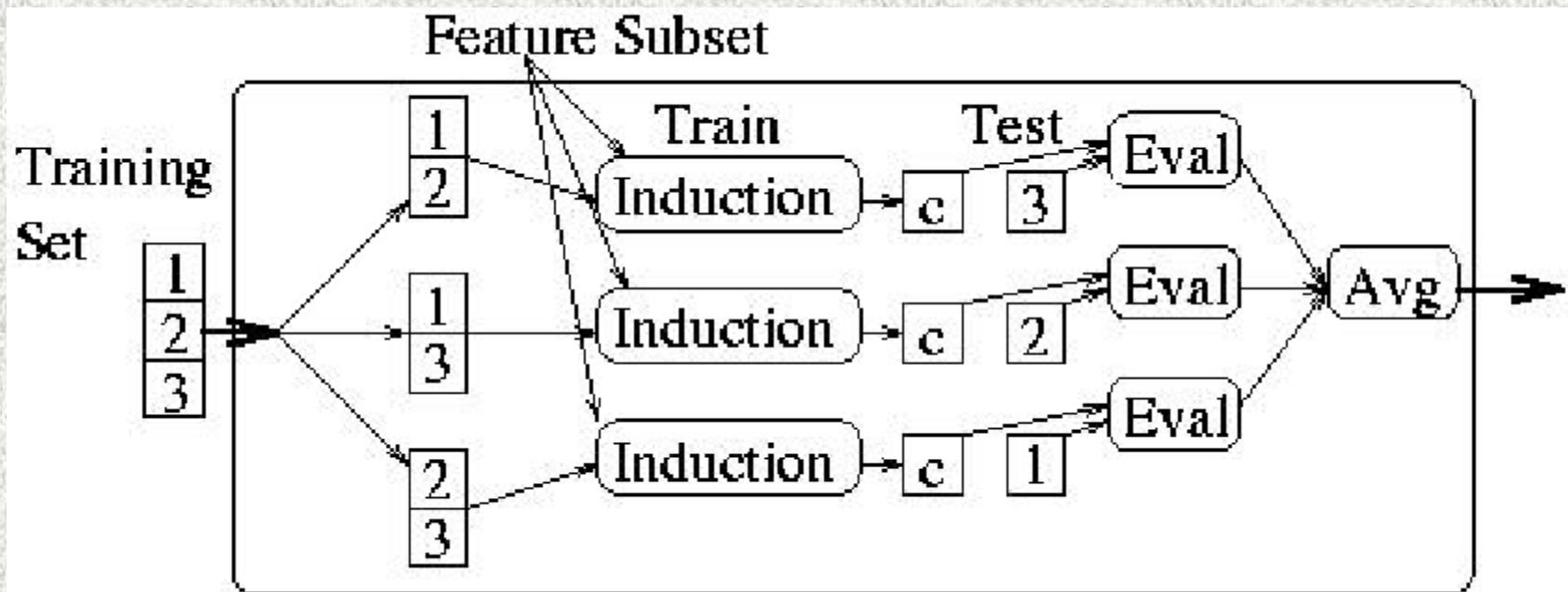
# Measure Performance of a Learning Algorithm

- Collect large set of examples (as large and diverse as possible)
- Divide into 2 disjoint sets (training set and test set)
- Learn concept based on training set, generating hypothesis H
- Classify test set examples using H, measure percentage correctly classified
- Should demonstrate improved performance as training set size increases (learning curve)
- How quickly does it learn?



# Measure Performance of a Learning Algorithm

- Use statistics tests to determine significance of improvement
- Cross-validation



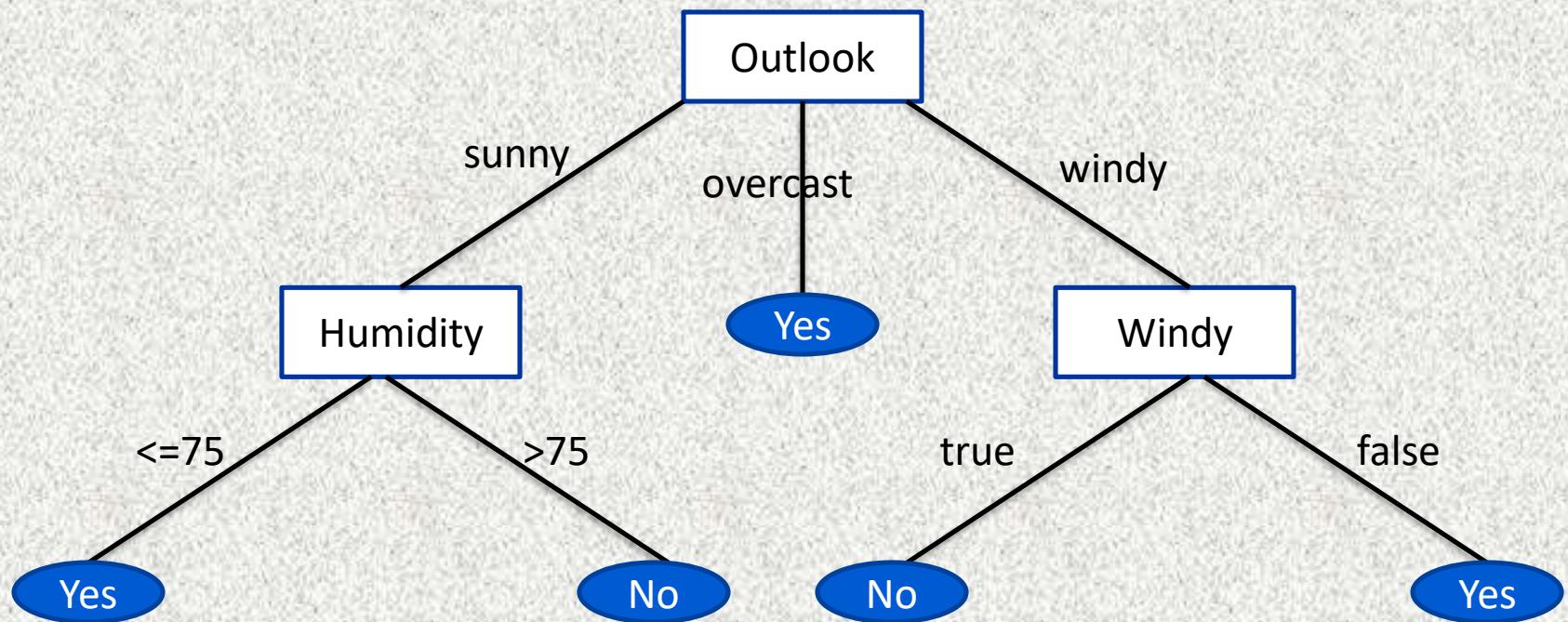
# Challenges for Decision Trees

- Numeric attributes
- Missing attribute values
- Incremental updating

# Training Examples

| Day | Outlook  | Temperature | Humidity | Wind   | PlayTennis |
|-----|----------|-------------|----------|--------|------------|
| D1  | Sunny    | 85          | 85       | Weak   | No         |
| D2  | Sunny    | 80          | 90       | Strong | No         |
| D3  | Overcast | 83          | 86       | Weak   | Yes        |
| D4  | Rain     | 70          | 96       | Weak   | Yes        |
| D5  | Rain     | 68          | 80       | Weak   | Yes        |
| D6  | Rain     | 65          | 70       | Strong | No         |
| D7  | Overcast | 64          | 65       | Strong | Yes        |
| D8  | Sunny    | 72          | 95       | Weak   | No         |
| D9  | Sunny    | 69          | 70       | Weak   | Yes        |
| D10 | Rain     | 75          | 80       | Weak   | Yes        |
| D11 | Sunny    | 75          | 70       | Strong | Yes        |
| D12 | Overcast | 72          | 90       | Strong | Yes        |
| D13 | Overcast | 81          | 75       | Weak   | Yes        |
| D14 | Rain     | 71          | 91       | Strong | No         |

# Decision Tree



# Performance Measures

- Percentage correctly classified, averaged over folds
- Confusion matrix

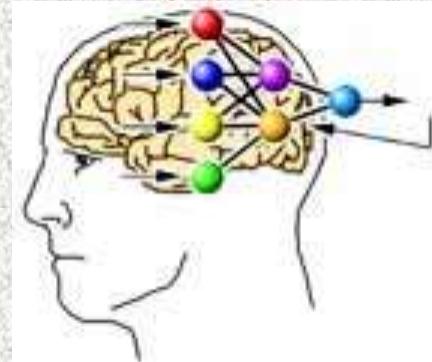
|                    | Predicted<br>Negative | Predicted<br>Positive |
|--------------------|-----------------------|-----------------------|
| Actual<br>Negative | TN                    | FP                    |
| Actual<br>Positive | FN                    | TP                    |

- Accuracy =  $(TP+TN) / (TP+FP+TN+FN)$
- Error =  $1 - \text{Accuracy}$
- Precision, Recall, ROC curves

# Examples

- Bet on a basketball team
- Build decision tree
- Build decision trees

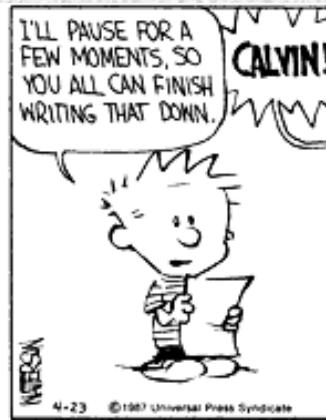
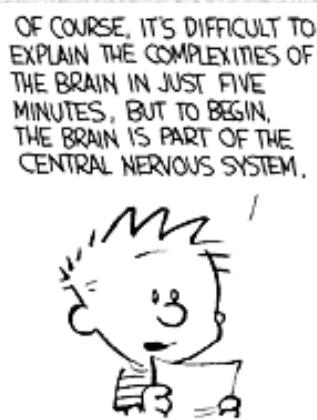
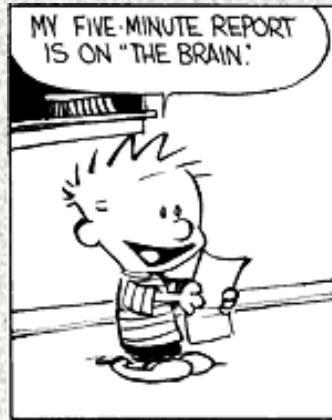
# Neural Networks



- Instead of traditional von Neumann machines, researchers wanted to build machines based on the human brain
- An architecture as well as a learning technique  
Connection Machine
- The data structure is a network of units that act as "neurons"
- Tested as a computing device originally by researchers such as Jon Hopfield (Cal Tech), Hebb (1949), Minsky (1951), and Rosenblatt (1957)
- Also models human performance  
Voice recognition, handwriting recognition, face recognition (traditionally computers bad at these tasks, humans great)

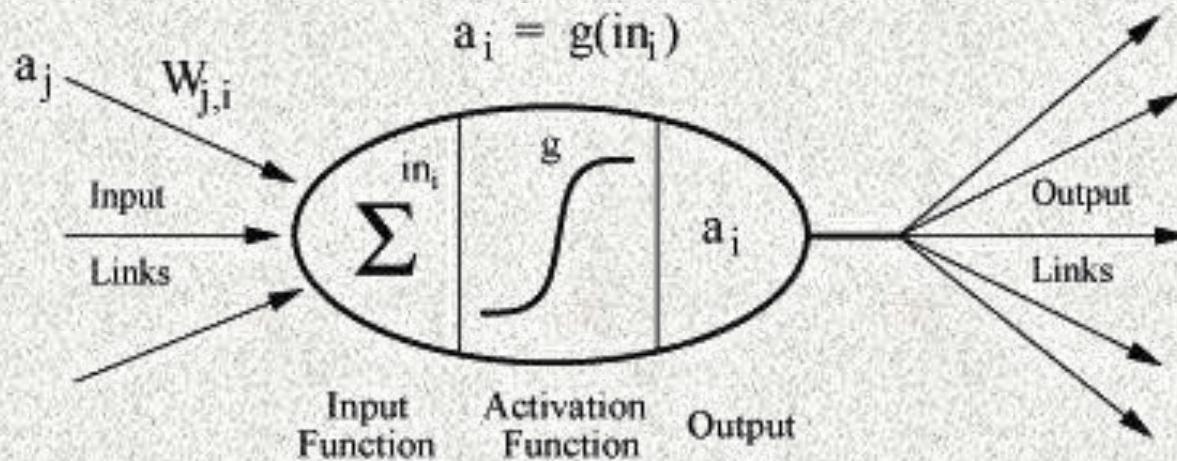
# Power In Numbers

- Each neuron is not extremely powerful by itself
- Neuron switching time is ~ second
- Each message 100,000 times slower than a computer switch
- 10 billion - 1 trillion neurons
- Each neuron has 1,000 - 100,000 connections
- Computational neural networks are inspired by biology, but do not exactly imitate biology

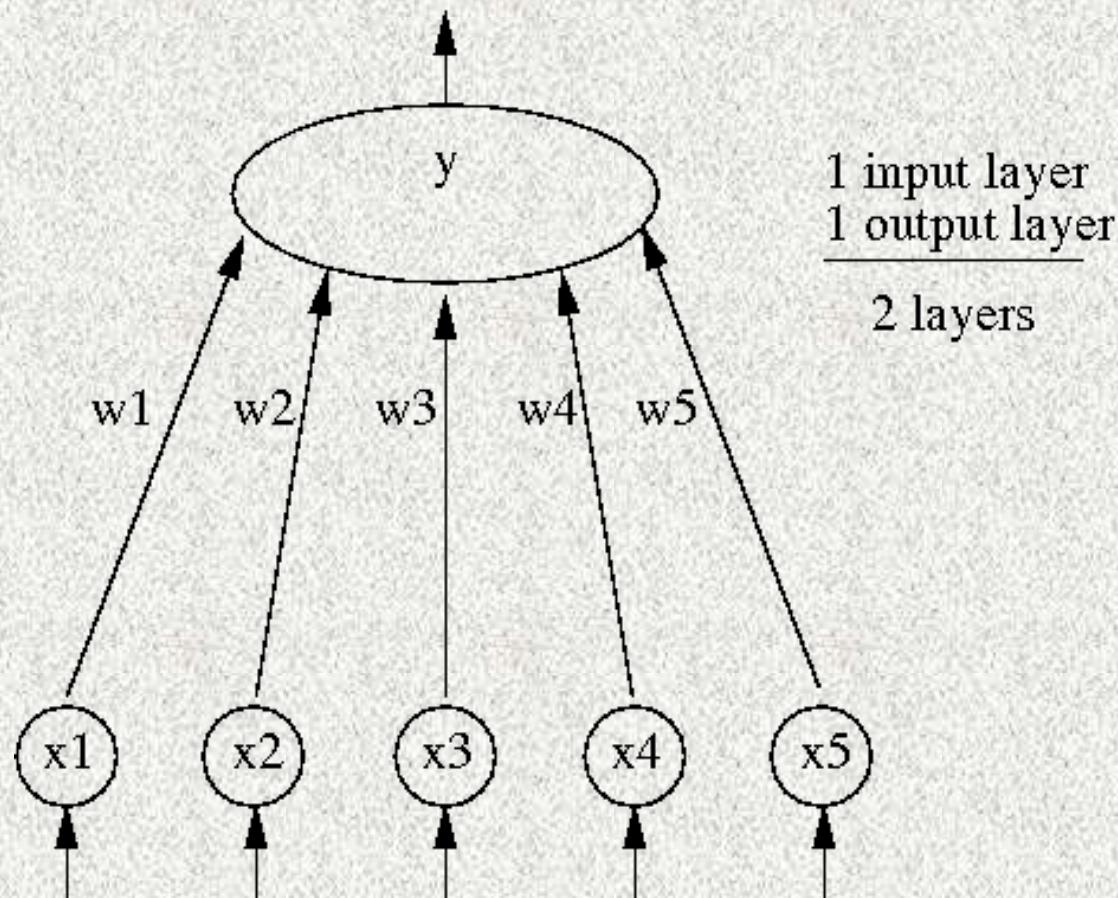


# Neuron

A neural network is made up of neurons, or processing elements



# A Simple Neural Network – The Perceptron



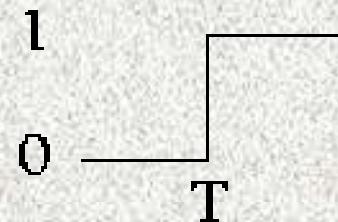
Input Units – One PE (neuron) per input feature

# Neuron

- Each activation value  $x_i$  is weighted by  $w_i$
- The output  $y$  is determined by the NN transfer function

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i > \text{Threshold} \\ 0 & \text{otherwise} \end{cases}$$

- This is a step transfer function



# Neural Networks Learn a Function

- $y = \text{function}(x_1, x_2, \dots, x_n)$
- Perceptrons use one input neuron for each input parameter  $x_1..x_n$
- $y$  is computed using the transfer function applied to values coming in to the output node
- Suppose we are trying to learn the concept “all binary strings of length five with a 1 in the first and last positions”
- $10101 \rightarrow 1$   
 $10100 \rightarrow 0$
- 5 input units, 1 output unit  
Input units are assigned value 0 or 1  
In this case, output is 0 or 1
- Input units are always assigned a value  
If we need symbolic inputs, can map to numeric inputs (convert to binary)
- $f(2 \text{ legs, brown, flat, } 3' \text{ high}) = \text{chair}$

# Applications

- Handwriting recognition
- Control problems
- Autonomous navigation
- Stock market prediction
- Image recognition
  - Alvinn drives 70 mph on highways
  - [Alvinn in action](#)

# When To Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

# Two Computation Phases

1. Training phase
2. Testing / use phase
  - During training, run perceptron on examples and compare network output ( $y$ ) to desired output ( $y^d$ )
  - Weights are adjusted after each training step using function

$$w^{new} = w^{old} + (y^d - y)x$$

- Optionally, the threshold can be adjusted as well using function

$$t^{new} = t^{old} - (y^d - y)$$

- Notice that  $x$  is the value of the input feature, thus weights are changed ONLY for nodes that are activated (used in the computation)

# Parameters That Can Affect Performance

- Initial weights (can be initialized to 0, usually better if randomly set)
- Initial threshold,  $y = 1$  if  $\sum w_i * x_i > Threshold$
- Transfer function
- Learning rate,  $w^{new} = w^{old} + \eta(y^d - y)x$
- Threshold update function
- Number of epochs

# Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Initially let  $w_1=0$ ,  $w_2=0$ ,  $T=0$ ,  $\eta=1$
- Epoch **1**

| x1 | x2 | y <sup>d</sup> |
|----|----|----------------|
| 0  | 0  | 0              |
| 0  | 1  | 0              |
| 1  | 0  | 0              |
| 1  | 1  | 1              |

| x1 | x2 | w1 <sup>old</sup> | w2 <sup>old</sup> | T <sup>old</sup> | y | y <sup>d</sup> | w1 <sup>new</sup> | w2 <sup>new</sup> | T <sup>new</sup> |
|----|----|-------------------|-------------------|------------------|---|----------------|-------------------|-------------------|------------------|
| 0  | 0  | 0                 | 0                 | 0                | 0 | 0              | 0                 | 0                 | 0                |
| 0  | 1  | 0                 | 0                 | 0                | 0 | 0              | 0                 | 0                 | 0                |
| 1  | 0  | 0                 | 0                 | 0                | 0 | 0              | 0                 | 0                 | 0                |
| 1  | 1  | 0                 | 0                 | 0                | 0 | 1              | 1                 | 1                 | -1               |

# Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch 2

| x1 | x2 | y <sup>d</sup> |
|----|----|----------------|
| 0  | 0  | 0              |
| 0  | 1  | 0              |
| 1  | 0  | 0              |
| 1  | 1  | 1              |

| x1 | x2 | w1 <sup>old</sup> | w2 <sup>old</sup> | T <sup>old</sup> | y | y <sup>d</sup> | w1 <sup>new</sup> | w2 <sup>new</sup> | T <sup>new</sup> |
|----|----|-------------------|-------------------|------------------|---|----------------|-------------------|-------------------|------------------|
| 0  | 0  | 1                 | 1                 | -1               | 1 | 0              | 1                 | 1                 | 0                |
| 0  | 1  | 1                 | 1                 | 0                | 1 | 0              | 1                 | 0                 | 1                |
| 1  | 0  | 1                 | 0                 | 1                | 0 | 0              | 1                 | 0                 | 1                |
| 1  | 1  | 1                 | 0                 | 1                | 0 | 1              | 2                 | 1                 | 0                |

# Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch 3

| x1 | x2 | y <sup>d</sup> |
|----|----|----------------|
| 0  | 0  | 0              |
| 0  | 1  | 0              |
| 1  | 0  | 0              |
| 1  | 1  | 1              |

| x1 | x2 | w1 <sup>old</sup> | w2 <sup>old</sup> | T <sup>old</sup> | y | y <sup>d</sup> | w1 <sup>new</sup> | w2 <sup>new</sup> | T <sup>new</sup> |
|----|----|-------------------|-------------------|------------------|---|----------------|-------------------|-------------------|------------------|
| 0  | 0  | 1                 | 1                 | 0                | 0 | 0              | 2                 | 1                 | 0                |
| 0  | 1  | 1                 | 1                 | 0                | 1 | 0              | 2                 | 0                 | 1                |
| 1  | 0  | 2                 | 0                 | 1                | 1 | 0              | 1                 | 0                 | 2                |
| 1  | 1  | 1                 | 0                 | 2                | 0 | 1              | 2                 | 1                 | 1                |

# Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch 4

| x1 | x2 | y <sup>d</sup> |
|----|----|----------------|
| 0  | 0  | 0              |
| 0  | 1  | 0              |
| 1  | 0  | 0              |
| 1  | 1  | 1              |

| x1 | x2 | w1 <sup>old</sup> | w2 <sup>old</sup> | T <sup>old</sup> | y | y <sup>d</sup> | w1 <sup>new</sup> | w2 <sup>new</sup> | T <sup>new</sup> |
|----|----|-------------------|-------------------|------------------|---|----------------|-------------------|-------------------|------------------|
| 0  | 0  | 2                 | 1                 | 1                | 0 | 0              | 2                 | 1                 | 1                |
| 0  | 1  | 2                 | 1                 | 1                | 0 | 0              | 2                 | 1                 | 1                |
| 1  | 0  | 2                 | 1                 | 1                | 1 | 0              | 1                 | 1                 | 2                |
| 1  | 1  | 1                 | 1                 | 2                | 0 | 1              | 2                 | 2                 | 1                |

# Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch 5

| x1 | x2 | y <sup>d</sup> |
|----|----|----------------|
| 0  | 0  | 0              |
| 0  | 1  | 0              |
| 1  | 0  | 0              |
| 1  | 1  | 1              |

| x1 | x2 | w1 <sup>old</sup> | w2 <sup>old</sup> | T <sup>old</sup> | y | y <sup>d</sup> | w1 <sup>new</sup> | w2 <sup>new</sup> | T <sup>new</sup> |
|----|----|-------------------|-------------------|------------------|---|----------------|-------------------|-------------------|------------------|
| 0  | 0  | 2                 | 2                 | 1                | 0 | 0              | 2                 | 1                 | 1                |
| 0  | 1  | 2                 | 2                 | 1                | 1 | 0              | 2                 | 1                 | 2                |
| 1  | 0  | 2                 | 1                 | 2                | 0 | 0              | 2                 | 1                 | 2                |
| 1  | 1  | 2                 | 1                 | 2                | 1 | 1              | 2                 | 1                 | 2                |

# Learn Logical AND of $x_1$ and $x_2$

$$\sum w_i * x_i > Threshold$$

- Epoch 6

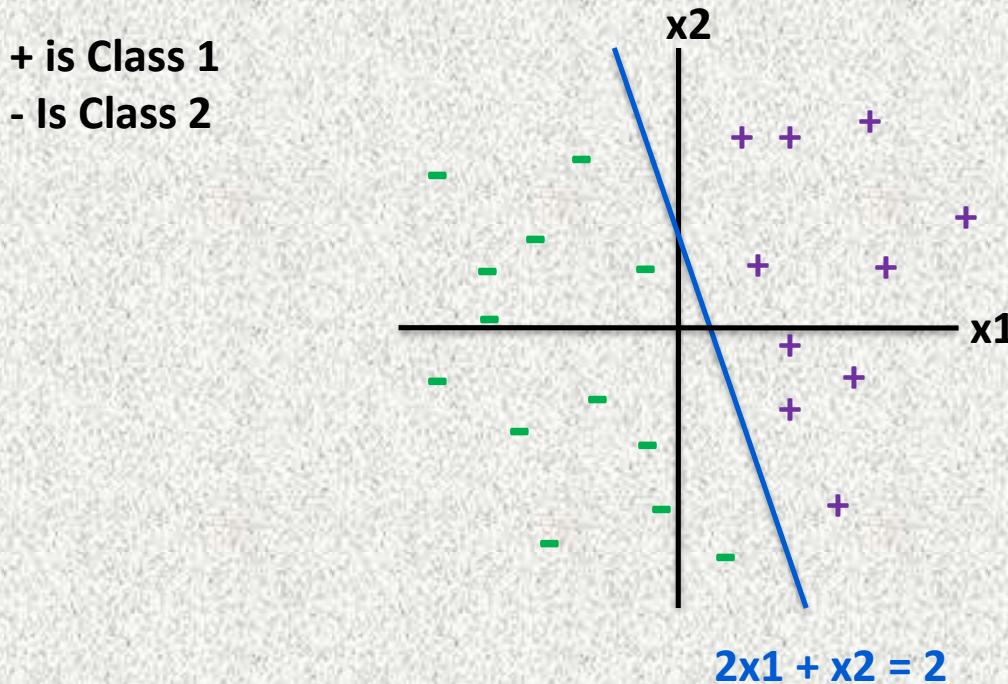
| x1 | x2 | y <sup>d</sup> |
|----|----|----------------|
| 0  | 0  | 0              |
| 0  | 1  | 0              |
| 1  | 0  | 0              |
| 1  | 1  | 1              |

| x1 | x2 | w1 <sup>old</sup> | w2 <sup>old</sup> | T <sup>old</sup> | y | y <sup>d</sup> | w1 <sup>new</sup> | w2 <sup>new</sup> | T <sup>new</sup> |
|----|----|-------------------|-------------------|------------------|---|----------------|-------------------|-------------------|------------------|
| 0  | 0  | 2                 | 1                 | 2                | 0 | 0              | 2                 | 1                 | 2                |
| 0  | 1  | 2                 | 1                 | 2                | 0 | 0              | 2                 | 1                 | 2                |
| 1  | 0  | 2                 | 1                 | 2                | 0 | 0              | 2                 | 1                 | 2                |
| 1  | 1  | 2                 | 1                 | 2                | 1 | 1              | 2                 | 1                 | 2                |

CONVERGENCE!

# The AND Function

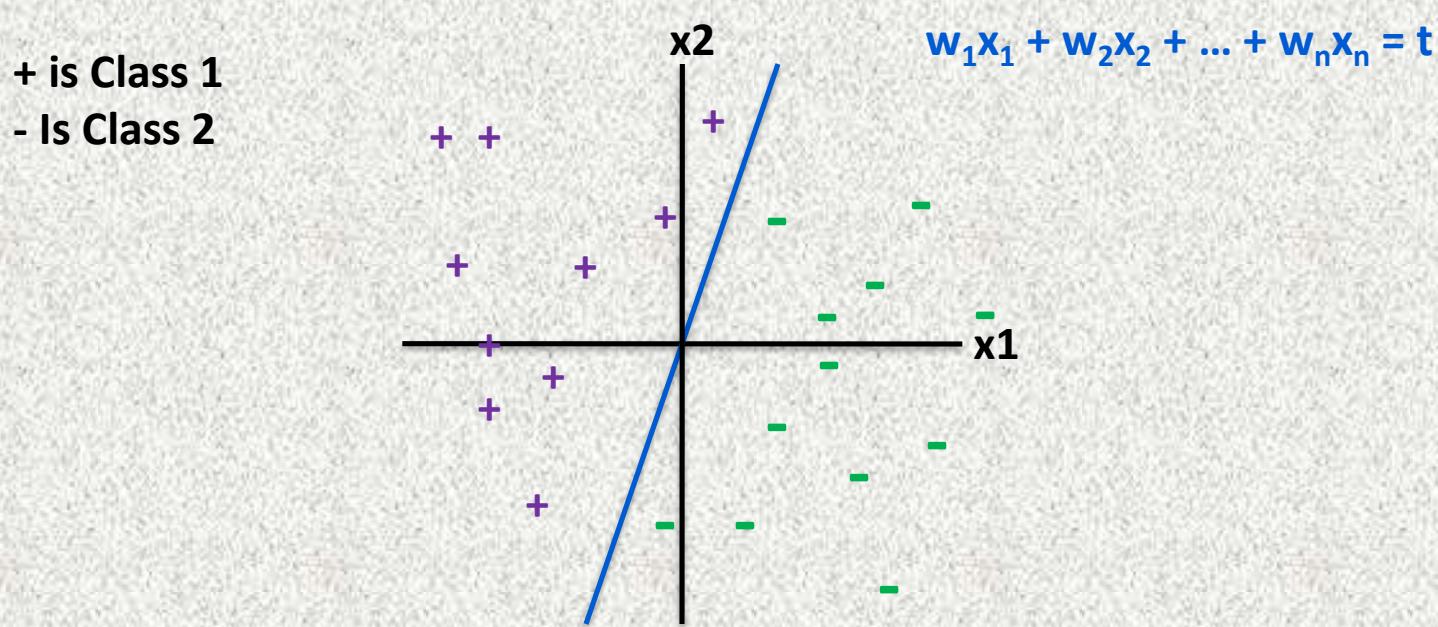
- Notice that the classes can be separated by a line (**hyperplane**)



- Learn XOR of two inputs,  $x_1$  and  $x_2$ 
  - $w_1=2$ ,  $w_2=1$ , Threshold=2
- Why does the perceptron run forever and never converge?

# Function Learned By Perceptron

- The final network can be expressed as an equation of the parameters  $x_1$  through  $x_n$
- $w_1 x_1 + w_2 x_2 + \dots + w_n x_n = \text{Threshold}$
- If learned, the network can be represented as a hyperplane

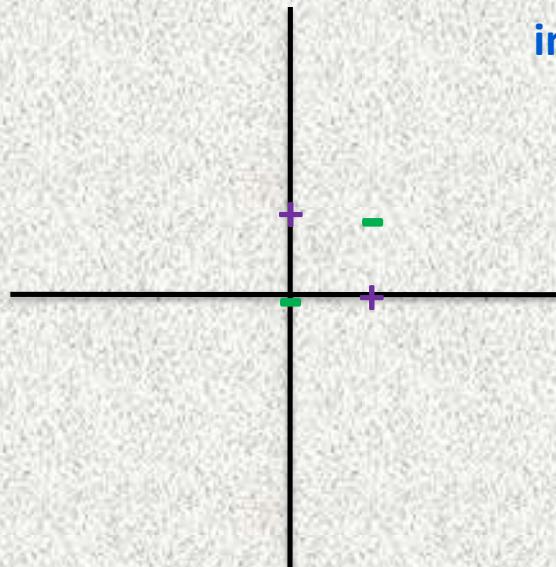


# Examples

- Perceptron Example
- Perceptron Example

# Linearly Separable

- If the classes can be separated by a hyperplane, then they are linearly separable.
- Linearly Separable Learnable by a Perceptron
- Here is the **XOR** space:



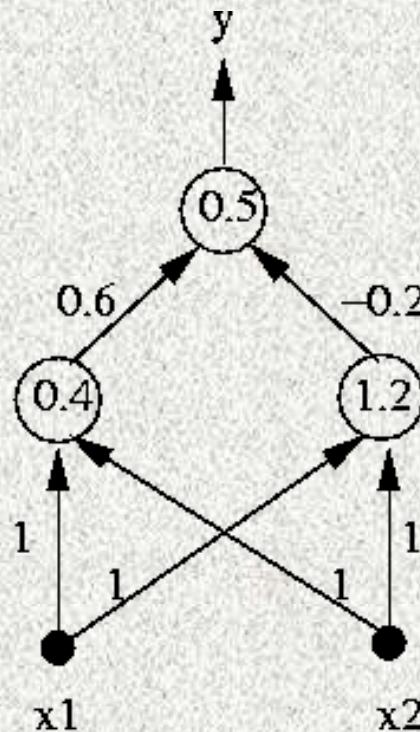
No line can separate these data points  
into two classes – need two lines

# How Can We Learn These Functions?

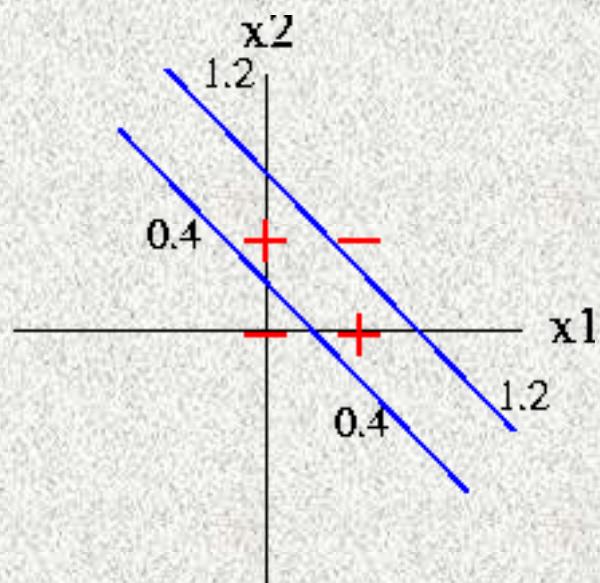
- Add more layers with more neurons!
- Features of perceptrons:
  - Only 1 neuron in output layer
  - Inputs only 0 or 1
  - Transfer function compares weighted sum to threshold
  - No hidden units
  - Output is only 0 or 1

# Multilayer Neural Networks

- 1 input layer (2 units), 1 hidden layer (2 units), 1 output layer (1 unit)
- Like before, output is a function of the weighted input to the node



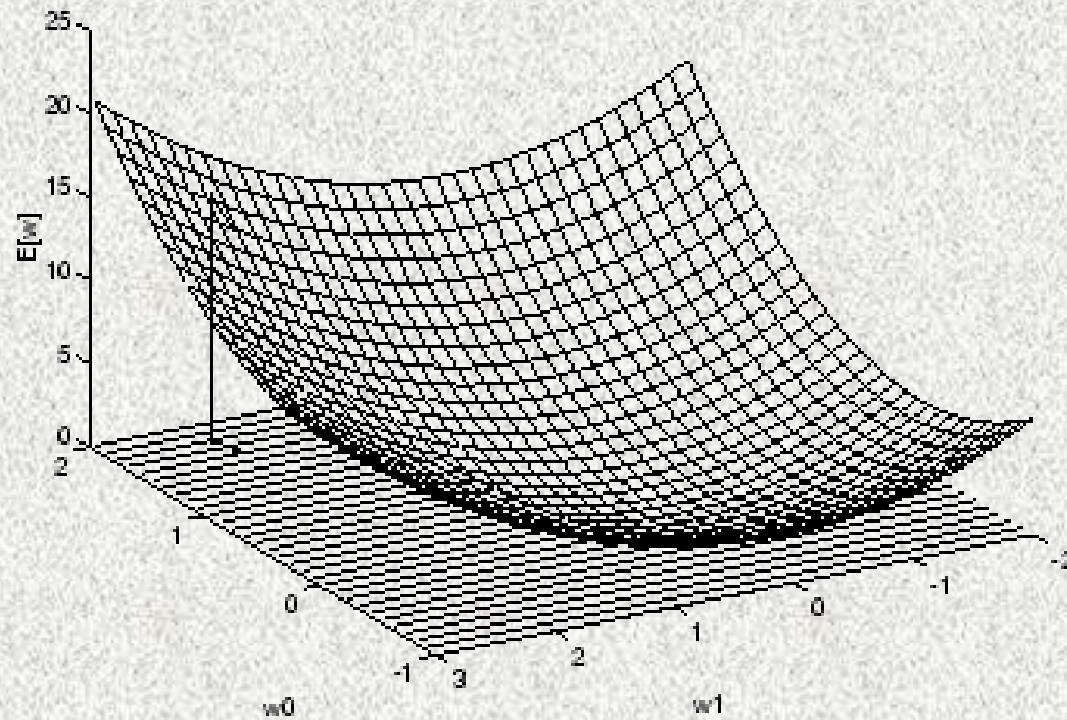
# Function Learned by MNN



# Learning in a Multilayer Neural Network

- How should we change (adapt) the weights in a multilayer neural network?
- A perceptron is easy - direct mapping between weights and output.
- Here, weights can contribute to intermediary functions and only indirectly affect output. These weights can indirectly affect multiple output nodes.
- If output is 12 and we want a 10, change weights to output node so that output next time would be (closer to) desired value 10.
- How do we change weights to hidden units?
- Assign portion of error to each hidden node, change weights to lessen that error next time.

# Weight Learning Using Gradient Descent



# Deriving an Update Function

Our goal is to reduce error (often *sum of squared errors*), which is

$$E = \frac{1}{2} Err^2 = \frac{1}{2} (t - o)^2$$

Since the gradient specifies direction of steepest increase of error, the training rule for gradient descent is to update each weight by the derivative of the error with respect to each weight, or

$$\frac{\partial E}{\partial W_j} = Err * \frac{\partial Err}{\partial W_j}$$

The derivative of a particular weight is  $Err \times \frac{\partial}{\partial W_j} (t - g(\sum_{j=0}^n W_j a_j)) = -Err \times g'(in) \times a_j$ ,

where  $g'(in)$  is the derivative of the transfer function &  $a_j$  is the activation value at source node j.

We want to eliminate the error when we adjust the weights, so we multiply the formula by -1. We want to constrain the adjustment, so we multiply the formula again by the learning rate  $\eta$ .

The result is the **Delta Rule**.

# Delta Rule

- Weight update for hidden-to-output links:

$$w_{ji} = w_{ji} + (\eta * a_j * Err_i * g'(in_i))$$

**w<sub>ji</sub>** = Weight of link from node j to node i

**η** = Learning rate, eta

**a<sub>j</sub>** = Activation of node j (output of hidden node j or input for input node j)

**Err<sub>i</sub>** = Error at this node (target minus actual output,  
total weight change needed to node i)

**g'(in<sub>i</sub>)** = Derivative of the transfer function g

Same general idea as before. If error is positive, then network output is too small so weights are increased for positive inputs and decreased for negative inputs.

The opposite happens when the error is negative.

# Hidden-to-output Weights

$$w_{ji} = w_{ji} + (\eta * a_j * Err_i * g'(in_i))$$

Let  $\Delta_i = Err_i * g'(in_i)$  represent the error term

$$\Delta_i = (t_i - o_i) * g'(in_i)$$

$t_i$  = true / target output for node i

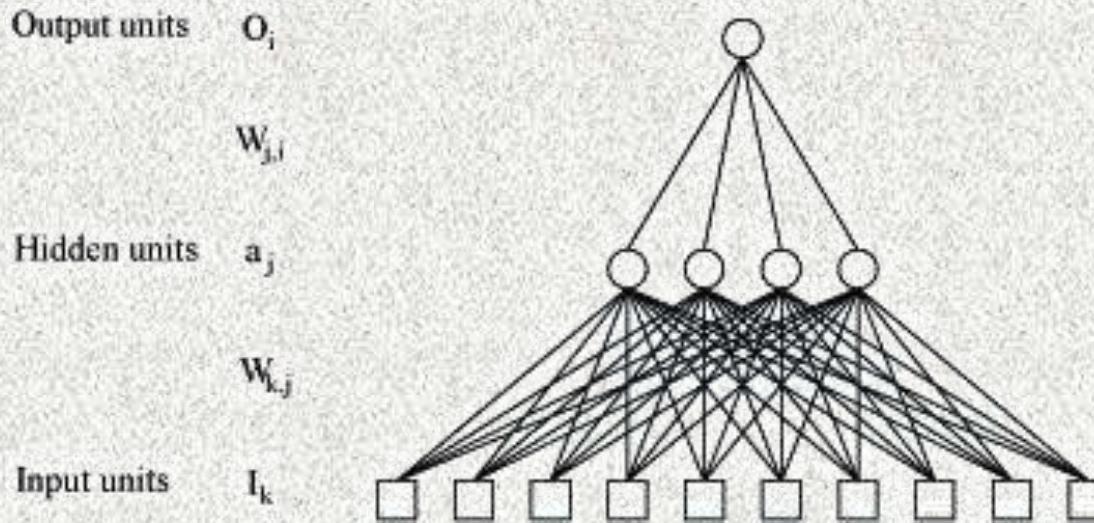
$o_i$  = actual / calculated output for node i

$in_i$  = sum of inputs

$g'$  = derivative of the transfer function

# Next Layer

- For input-to-hidden weights, we need to define a value analogous to the error term for output nodes. Here we perform error backpropagation. Each hidden node is assigned a portion of the error corresponding to its contribution to the output node.
- The formula  $\Delta_j = g'(in_j) \sum_i w_{ji} \Delta_i$ 
  - Assigns a portion of the responsibility to node j
  - The proportion is determined by the weight from j to all output nodes. Now we can give the weight update rule for links from input to hidden nodes.
- For each input node k to hidden node j use  $w_{kj} = w_{kj} + (\eta * I_k * \Delta_j)$



# Update Function

- Why use the derivative of the transfer function in the calculation of delta?
- Note the visualization of the weight space using gradient descent.
- We want to move the weights in the direction of steepest descent in this space.
- To do this, compute derivative of the error with respect to each weight in the equation. This results in the delta terms showed earlier.
- Note that the transfer function must be differentiable everywhere.

# Step Function

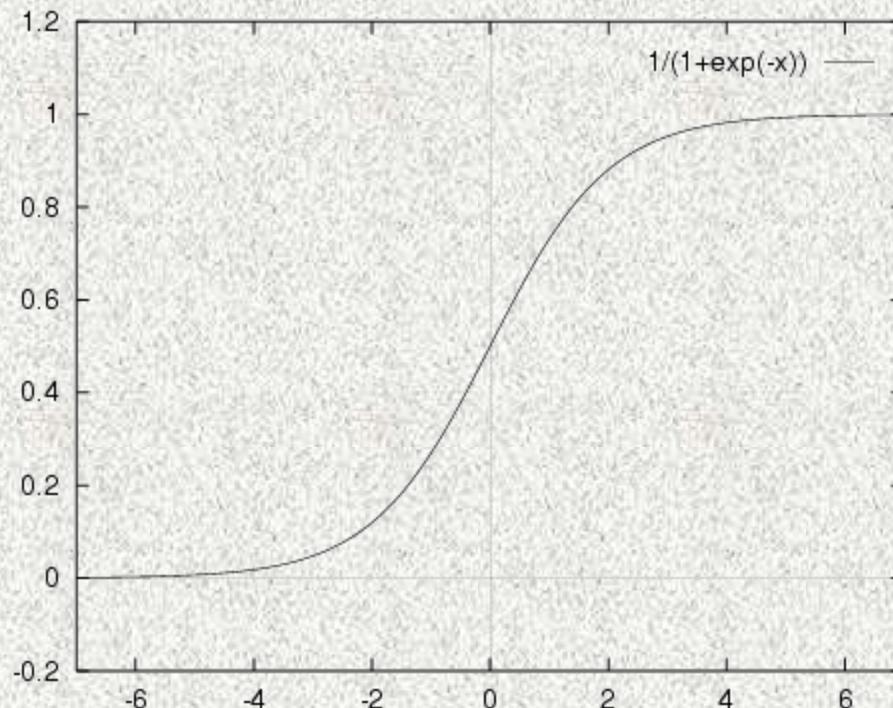
- Our perceptron transfer function will not work



derivative = infinity here

# Sigmoid Function

$$g(x) = \frac{1}{1-e^{-x}} \quad \text{where } x \text{ is weighted sum of inputs}$$



The sigmoid function is handy to use for backpropagation because its derivative is easily computed as  $g(x)*(1-g(x))$ .

# Sigmoid Update formula

- To calculate output, for each node in network
  - Calculate weighted sum of inputs,  $in_i = \sum_j w_{ji} a_j$
  - Compute output or activation of node,  $a_i = sigmoid(in_i)$
- For each node i in output layer
$$\Delta_i = sigmoid(in_i) * (1 - sigmoid(in_i)) * (t_i - a_i)$$
- For each node j in lower layers
$$\Delta_j = sigmoid(in_j) * (1 - sigmoid(in_j)) * \sum_i w_{ji} \Delta_i$$
- Update weight  $w_{ji}$  by  $w_{ji} = w_{ji} + \eta * a_j * \Delta_i$

# NN Applications - NETtalk

- Sejnowski and Rosenburg, 1985
- Written English text to English speech
- Based on DECTalk expert system
- Look at a window of 7 characters:

THIS\_IS  
↑  
(A-Z, " ", ".", "?")

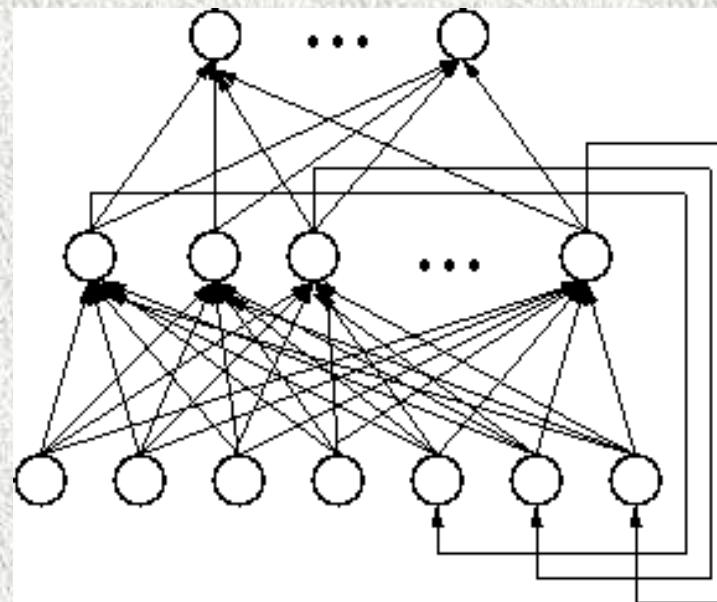
- Decide how to utter middle character
- 1 network
  - 1 hidden layer
  - 203 input units (7 character window \* 29 possible characters)
  - 80 hidden units
  - Approximately 30 output units
- Demo

# Examples

- [Handwriting Recognition](#)
- [Balancing Ball](#)
- [Learn 3D Map](#)

# Networks That Deal With Time

- In the feedforward networks we have been studying, transfer functions capture the network state (not usually spatiotemporal in nature).
- Recurrent neural networks feed signal back from output to network (output to hidden, hidden to hidden, hidden to input, others).
- In this way they can learn a function of time.
- $y(t) = w_1 x_1(t) + w_2 x_2(t) + \dots + w_{n+1} x_1(t-1) + w_{n+1} x_2(t-1) + \dots$

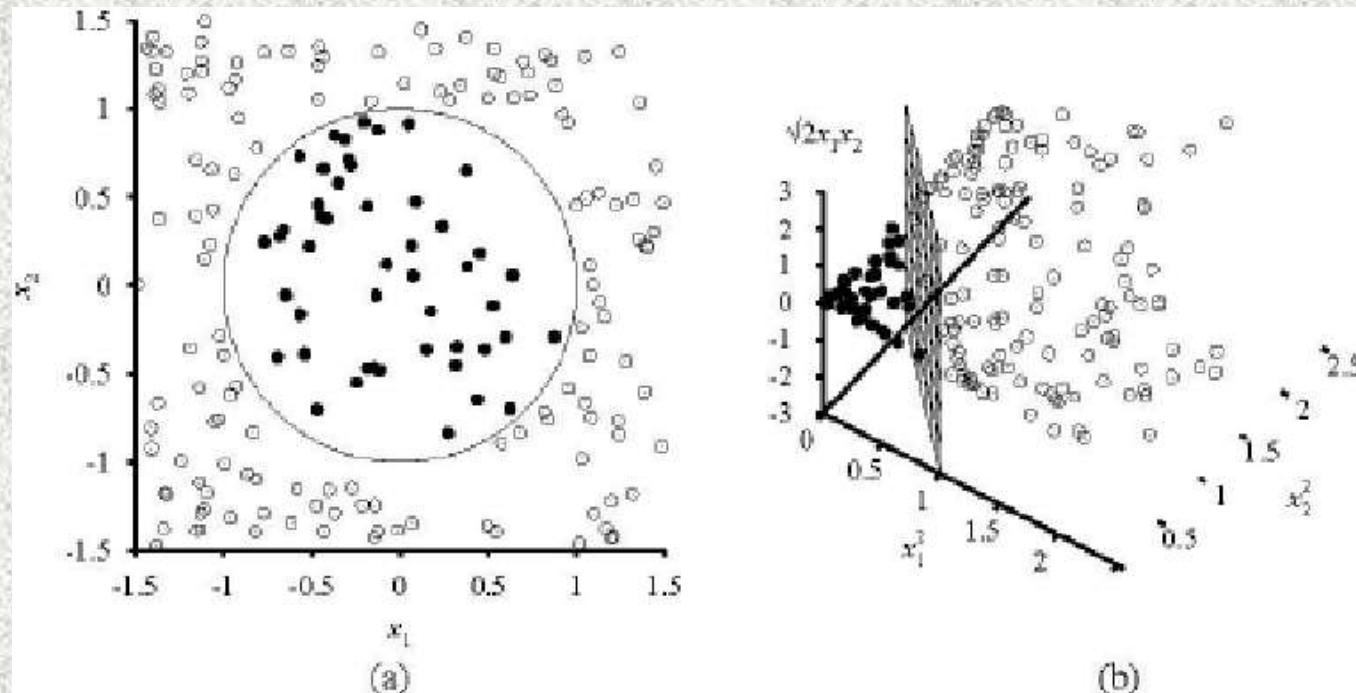


# Neural Network Issues

- Usefulness
- Generalizability
- Understandability (cannot explain results)
- Self-structuring neural networks
  - Add/delete nodes and links until error is minimized
- Networks and expert systems
  - Can we learn rules corresponding to network?
- Input background knowledge
  - Predefined structure, weights
- Computational complexity
  - Blum and Rivest in 1992 proved that training even a three-node network is NP Complete
- Inductive bias is smooth interpolation between data points

# Extending the Idea of Linear Separability

- No plane separates examples on the left



- We can, however, map the figures onto three *new* features

$$f_1 = x_1^2, f_2 = x_2^2, f_3 = \sqrt{2}x_1x_2$$

- and the data in the new space is separable as shown on the right.
- We can search for such mappings that leave maximal margins between classes. This is learning using support vector machines .

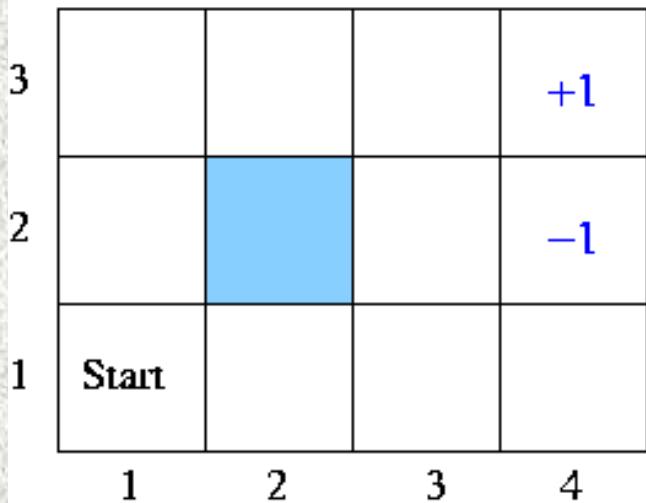
# Reinforcement Learning

- Learn action selection for probabilistic applications
  - Robot learning to dock on battery charger
  - Learning to choose actions to optimize factory output
  - Learning to play Backgammon
- Note several problem characteristics:
  - Delayed reward
  - Opportunity for active exploration
  - Possibility that state only partially observable
  - Possible need to learn multiple tasks with same sensors/effectors

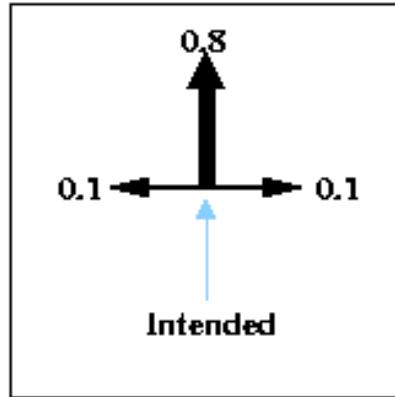
# Reinforcement Learning

- Learning an optimal strategy for maximizing future reward
- Agent has little prior knowledge and no immediate feedback
- Action credit assignment difficult when only future reward
- Two basic agent designs
  - Agent learns utility function  $U(s)$  on states
    - Used to select actions maximizing expected utility
    - Requires a model of action outcomes ( $T(s,a,s')$ )
  - Agent learns action-value function
    - Gives expected utility  $Q(s,a)$  of action  $a$  in state  $s$
    - Q-learning  $Q(s,a)$
    - No action outcome model, but cannot look ahead
- Can handle deterministic or probabilistic state transitions

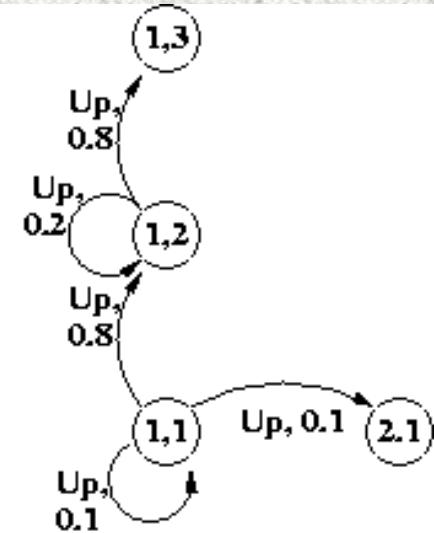
# Passive Learning in a Known Environment



Environment



Probabilistic Action



MDP

- $P(\text{intended move}) = 0.8$ ,  $P(\text{right angles to intended move}) = 0.1$
- Rewards at terminal states are +1 and -1, other states have reward of -0.04
- From our start position the recommended sequence is [Up, Up, Right, Right, Right]. This reaches the goal with probability  $0.8^5 = 0.32768$ .
- Transitions between states are probabilistic, and are represented as a Markov Decision Process.

# Markov Decision Processes

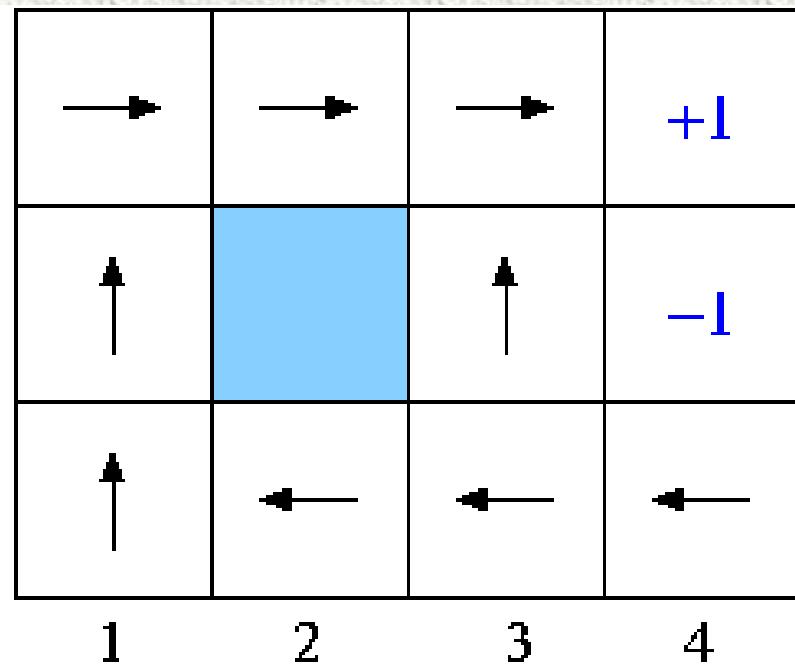
- Assume
    - Finite set of states  $S$  (with initial state  $S_0$ )
  - Set of actions  $A$
  - Transition model  $T(s,a,s')$ , which can be probabilistic
  - At each discrete time agent observes state  $s$  in  $S$  and chooses action  $a$  in  $A$
  - then receives immediate reward  $R(s)$
  - and state changes to  $a'$
- 
- Markov assumption: Resulting state  $s'$  depends only on current state  $s$  (or a finite history of previous states) and action  $a$

# Problem Solution

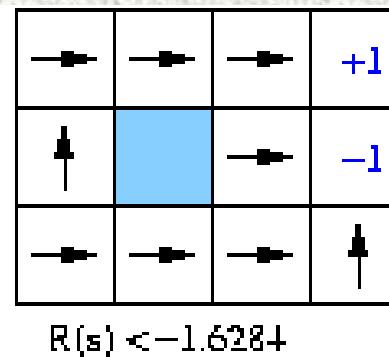
- Desirability of moving to a given state  $s$  is expressed by a utility value
- Utility is not the reward, but an estimate of the award that can be accrued from that state
- The policy  $\pi$  specifies what the agent should do for every reachable state.
- An **optimal policy**,  $\pi^*$ , is a policy that selects moves with the highest expected utility .

# Example Policy

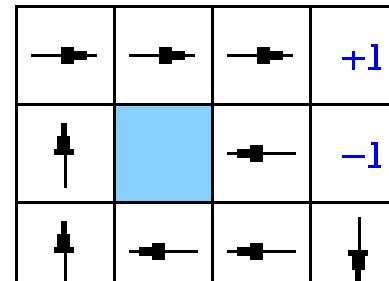
- Here is the optimal policy for our example environment.



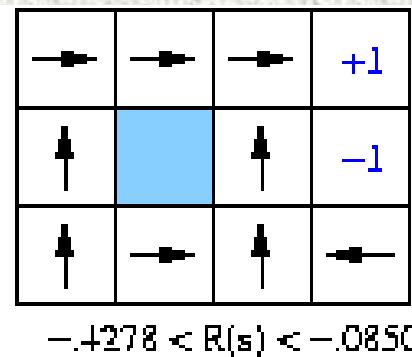
$R = -0.04$  for non-terminal states



$R(s) < -1.6284$



$-0.0221 < R(s) < 0$



$R(s) > 0$

# Define Utility Values

- Learn utilities  $U$  of each state, pick action that maximizes expected utility of resulting state
- Assume **infinite horizon**
  - Agent can move an infinite number of moves in the future
  - With fixed horizon of 3, agent would need to head from (3,1) directly to +1 terminal state
  - Given a fixed horizon of  $N$ ,  $U([s_0, s_1, \dots, s_{N+k}]) = U([s_0, s_1, \dots, s_N])$
- Reward is accumulated over entire sequence of states
  - Additive rewards
    - $U_h[s_0, s_1, s_2, \dots] = R(s_0) + R(s_1) + R(s_2) + \dots$
    - This could present a problem with infinite horizon problems
  - Discounted rewards:  $U_h[s_0, s_1, s_2, \dots] = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$
  - $\gamma$  is a discount factor
  - If  $R$  is bounded, even for an infinite horizon a discounted reward is finite
  - In the limit,  $R$  will approach  $\frac{R_{\max}}{1 - \gamma}$

# Calculate Utility Values

- We will define the utility of a state in terms of the utility of the state sequences that start from the state.
- The utility of state using policy is the expected discounted reward for sequence with t steps starting in s  $U^\pi(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s]$

|   |       |       |       |       |
|---|-------|-------|-------|-------|
| 3 | 0.812 | 0.868 | 0.918 | +1    |
| 2 | 0.762 |       | 0.660 | -1    |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |

- Here are utilities for our navigation problem with  $\gamma=1$  and  $R(s) = -.04$  for nonterminal states.

# Calculate Utility Values

- When selecting an action, the agent choose an action that maximizes the **Expected Utility** of the resulting state

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s') U(s')$$

- Because we define utility in terms of immediate reward and (discounted) expected utility of sequences from a state, we can define goodness now as

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$$

- This is the **Bellman equation**

# Value Iteration

- Calculate Bellman equation values incrementally

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s')$$

- Iterate until minimal changes
- Guaranteed to reach an equilibrium

# Example

|   |   |   |   |
|---|---|---|---|
|   |   |   |   |
| 3 | 0 | 0 | 0 |
| 2 | 0 | - | 0 |
| 1 | 0 | 0 | 0 |
|   | 1 | 2 | 3 |

# Example

|   |      |      |      |
|---|------|------|------|
|   |      |      |      |
| 3 | -.04 | -.04 | .36  |
| 2 | -.04 |      | -.14 |
| 1 | -.04 | -.04 | -.04 |
|   | 1    | 2    | 3    |

# Example

|   |      |      |       |       |
|---|------|------|-------|-------|
| 3 | -.04 | .1   | .371  | +1    |
| 2 | -.04 |      | .047  | -1    |
| 1 | -.04 | -.04 | -.065 | -.113 |
|   | 1    | 2    | 3     | 4     |

# Example

|   |       |       |       |       |
|---|-------|-------|-------|-------|
| 3 | 0.812 | 0.868 | 0.918 | +1    |
| 2 | 0.762 |       | 0.660 | -1    |
| 1 | 0.705 | 0.655 | 0.611 | 0.388 |

[RN] shows convergence after 30 iterations

# Where Does Learning Fit In?

- Learn the transition values
- Learn the utility values

# Adaptive Dynamic Programming

- Learn the transition model
- When a new state is encountered
  - Initialize utility to perceived reward for the state
  - Keep track of  $N_{sa}[s,a]$  (number of times action a was executed from state s and  $N_{sas'}[s,a,s']$  (number of times action a was executed from state s resulting in state s')
  - $T[s,a,s'] = N_{sas'}[s,a,s']/N_{sa}[s,a]$
  - Update utility as before
- Solve n equations in n unknowns,  $n = |\text{states}|$
- Converges slowly

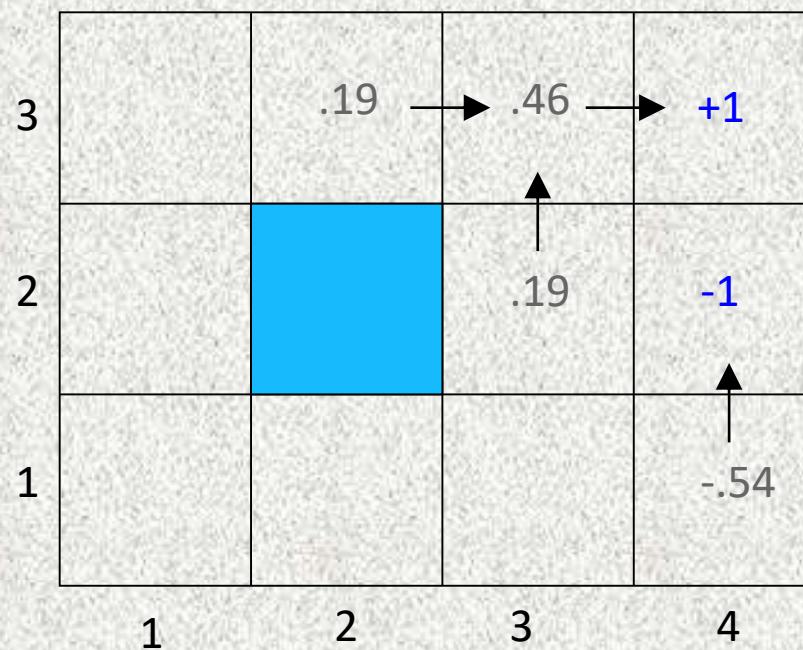
# Temporal Difference Learning

- Instead of solving equations for all states, incrementally update values only for states that are visited **after** they are visited
- TD reduces discrepancies between utilities of the current and past states
- If previous state has utility -100 and current state has utility +100, increase previous state utility to lessen discrepancy
- No more need to store explicit model, just need states, visited counts, rewards, and utility values

# TD Learning

- Temporal difference (TD)
  - When observe transition from state  $s$  to state  $s'$ 
    - $U(s) = U(s) + \alpha(R(s) + \gamma U(s') - U(s))$
    - Set  $U(s')$  to  $R(s')$  the first time  $s'$  is visited
    - $\alpha$  is learning rate
    - $\alpha$  can decrease as number of visits to  $s$  increases
    - $\alpha(N[s]) \sim 1/N[s]$
- Slower convergence than ADP, but much simpler with less computation

# TD Example



# Learning an Action-Value Function: Q-Learning

- Temporal difference removed the need for a model of transitions
- Q learning removes the need for a model of action selection
- Assigns value to action/state pairs, not just states
- These values are called Q-values  $Q(a,s)$ 
  - $Q(a,s) = \text{value of performing action } a \text{ in state } s$
  - Learn directly from observed transitions and rewards
  - Model-free approach
- $U(s)$  can be calculated as  $\max_a Q(s,a)$
- TD-based Q learning
  - When transition from state  $s$  to state  $s'$
  - $$Q(s, a) = Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

# Example

- Q Learning on Grid World
- Numbers show Q-value for each action, arrows show optimal action
- Rewards of +10 at (9,8), +3 at (8,3), -5 at (4,5), -10 at (4,8)
- Probability 0.7 of moving in desired direction, 0.1 of moving in any of other three directions
- Bumping into a wall incurs a penalty of 1 (reward of -1) and agent does not move

# Examples

- TD-Gammon [Tesauro, 1995]
  - Learn to play Backgammon
  - Immediate reward
    - +100 if win
    - -100 if lose
    - 0 for all other states
  - Trained by playing 1.5 million games against itself
  - Now approximately equal to best human player
- [More about TD-Gammon](#)
- [Q-Learning applet](#)
- [TD Learning applied to Tic Tac Toe](#)
- [Move graphic robot across space](#)
- [RL applied to channel allocation for cell phones](#)
- [RL and robot soccer](#)

# Subtleties and Ongoing Research

- Exploration vs. exploitation
- Scalability
- Generalize utilities from visited states to other states (inductive learning)
- Design optimal exploration strategies
- Extend to continuous actions, states

# CptS 440 / 540

# Artificial Intelligence

Review and Philosophical Questions

# Review and Philosophical Questions



# How Many AI Researchers Does It Take to Change a Lightbulb?

- [Rich and Knight, 1991]

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 1. The Search Group

- One to define the goal state
- One to define the operators
- One to decide on the least-cost, nearest-optimality search algorithm
- One to decide on a heuristic that indicates how close we are to a changed lightbulb
- One to indicate about how it is a model of human lightbulb-changing behavior
- One to call the Lisp hackers

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 2. The Lisp Hackers

- One to bring up the network
- One to order the Chinese food
- Four to hack on the Lisp debugger, compiler, window system, and microcode
- One to write the lightbulb-changing program

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 3. The Logic Group

- One to figure out how to describe lightbulb changing in predicate logic
- One to show the adequacy of predicate logic One to show the inadequacy of predicate logic
- One to show that lightbulb logic is nonmonotonic
- One to show that it isn't nonmonotonic
- One to incorporate nonmonotonicity into predicate logic
- One to determine the bindings for the variables
- One to show the completeness of the solution
- One to show the consistency of the solution
- One to hack a theorem prover for lightbulb resolution
- One to indicate how it is a description of human lightbulb changing behavior
- One to call the electrician

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 4. The Fuzzy Logic Group

- One to point out that, in the real world, a lightbulb is never "on" or "off", but usually somewhere in between

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 5. The Robotics Group

- One to build a vision system to recognize the dead bulb
- One to build a vision system to locate a new bulb
- One to figure out how to grasp the lightbulb without breaking it
- One to figure out the arm solutions that will get the arm to the socket
- One to organize the construction teams
- One to hack the planning system
- One to indicate how the robot mimics human motor behavior in lightbulb changing

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 6. The Game-Playing Group

- One to design a two-player game tree with the robot as one player and the lightbulb as the other
- One to write a minimax search algorithm that assumes optimal play on the part of the lightbulb
- One to build special-purpose hardware to enable 24-ply search
- One to enter the robot in a human lightbulb-changing tournament
- One to state categorically that lightbulb changing is “no longer considered AI”

# How Many AI Researchers Does It Take to Change a Lightbulb?

## 7. The Learning Group

- One to collect thirty lightbulbs
- One to collect thirty “near misses”
- One to write a concept-learning program that learns to identify lightbulbs
- One to show that the program found a local maximum in the space of lightbulb descriptions

# How Many AI Researchers Does It Take to Change a Lightbulb?

8. The Neural Network Group
  - One to claim that lightbulb changing can only be achieved through massive parallelism
  - One to build a backpropagation network to direct the robot arm
  - One to assign initial random weights to the connections in the network
  - One to train the network by showing it how to change a lightbulb (training shall consist of 500,000 repeated epochs)
  - One to tell the media that the network learns “just like a human”
  - One to compare the performance of the resulting system with that of symbolic learning approaches (optional)

# Philosophical Questions Regarding AI

- We have been concentrating on techniques to get AI to work
- Philosophers have been attacking the big questions
  - How CAN minds work?
  - How DO human minds work?
  - Can nonhumans have minds?

# Weak AI Position

- Machines can be made to act *as if* they were intelligent
  - There are things that computers cannot do, no matter how we program them
  - Certain ways of designing intelligent programs are bound to fail in the long run
  - The task of constructing the appropriate programs is infeasible

# Strong AI Position

- Machines that act intelligently can have real, conscious minds
- Weak AI doubts can be refuted
  - Locate a task thought impossible, design a program to accomplish task
  - Helps identify and remove AI researcher assumptions
- Strong AI doubts are difficult to refute
  - Hard to define
  - Hard to prove or disprove

# Possibility of Achieving Intelligent Behavior

- Can machines "think"?  
What does "think" mean?
- Can a machine reason about intentionality and consciousness?  
A machine needs to be aware of its own mental state and actions
- An intelligent agent needs emotions
- [Kismet, the sociable robot](#)
- An intelligent agent needs humor
- [AI needs to understand satire](#)
- An intelligent agent must be creative
- [Aaron, Computer Artist](#)
- [Musical Intelligence](#)



# Brain Prosthesis Experiment

- Gradually replace human neurons with electronic devices
- Electronic counterparts have identical I/O behavior
- Will consciousness of subject change? If so, when?

# AI: Present and Future

- Have we succeeded yet?
- Many tasks present, a few fundamental limitations



# Other AI Research

- There are other areas of study in AI that are also needed and are being pursued
  - Natural language processing
  - Researchers predicted this would be the first task achieved
  - Turns out to be one of the most difficult
  - [Language learning](#)
- Speech recognition
  - [Example using speech and lip reading](#)
- Additional machine learning techniques
  - Reinforcement learning, Bayes classification, Nearest neighbor, Inductive logic programming, graph-based learning
- Multiagent systems
- Robotics
  - The "action" component of an intelligent agent
- [Robot planetary explorer](#)
- [Robot fire-fighting competition](#)
- [Honda humanoid robot](#)
- [Cog MIT robot](#)
- [Example of multi-robot systems](#)
- [Another example of multi-robot systems](#)
- [Robot flocking](#)
- [Robot sheepdog](#)
- [Additional examples](#)

# What If We Succeed?

- Legal responsibility
- Should intelligent agents have rights?
- Job security
- Affirm uniqueness of humanity
- Privacy issues
- ``The success of AI might mean the end of the human race."
- How some perceive the future of AI
- [Panel discussions](#)
- [HAL](#)
- [Another example](#)
- What do you think?
- How should we use this technology?



# AI Grand Challenges

- Translating telephone
- Accident avoiding car
- Read college text and answer questions at the end
- Pass standardized tests (SAT)
- Identify all genes and therapeutic targets for specific types of cancer
- Fraud detection based on company financial statements before it is caught
- Discover new law publishable in journal for the related discipline
- Sponsored grand challenges:
  - [DARPA Grand Challenge](#)
  - [RoboCup](#)
  - [Netflix Prize](#)
  - [Loebner prize](#)