

Se3000

Week 1 notes

Agent
↳ perceives through sensor
↳ acts through effectors

Perceived action: action that maximizes expected value
↳ objective performance measure

Rationally depend on
↳ performance measure
↳ everything that the agent has perceived
↳ built-in knowledge about environment
↳ actions that can be performed

(Goal/Preference)
Perceives
Environment
Action

Types of Environment

Accessible (vs inaccessible)	Agent's sensory apparatus gives it access to the complete state of the environment
Deterministic (vs nondeterministic)	The next state of the environment is completely determined by the current state and the actions selected by the agent
Episodic (vs sequential)	Each episode is not affected by the previous taken actions
Static (vs dynamic)	Environment does not change while an agent is performing

Example: Chess

Accessible?	Yes. All positions in chessboard can be observed
Deterministic?	Yes. The outcome of each movement can be determined
Episodic?	No. The action depends on previous movements
Static?	Yes. When there is no clock, when are you considering the next step, the opponent can't move. Semi: When there is a clock, and time is up, you will give up the movement
Discrete?	Yes. All positions and movements are in discrete domains

Example: Driverless Taxi

Accessible?	No. Some traffic information on road is missing
Deterministic?	No. Some cars in front may turn right suddenly
Episodic?	No. The current action is based on previous driving actions
Static?	No. When the taxi moves, Other cars are moving as well
Discrete?	No. Speed, Distance, Fuel consumption are in real domains

More Examples

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Background	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Pat-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Design of Problem-Solving Agent

Steps

1. Goal formulation
2. Problem formulation (states, actions, goal test)
3. Solution (a sequence of legal action execution)
4. Search (algorithm of how to find a good solution)
 - No knowledge uninformed search
 - Knowledge informed search

Well-Defined Formulation

Definition of a problem	Re-state the goal with more specification (rules, resource, see below)
Formulation	<ul style="list-style-type: none"> • Initial state • Action set, i.e., available actions (successor functions) • State space, i.e., states reachable from the initial state <ul style="list-style-type: none"> • Solution path: sequence of actions from one state to another • Goal test predicate <ul style="list-style-type: none"> • Single state, enumerated list of states, abstract properties • Cost function <ul style="list-style-type: none"> • Path cost $g(n)$, sum of all (action) step costs along the path <p>A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test</p>
Solution (Legal play)	

Measuring Problem-Solving Performance

Search Cost

- What does it cost to find the solution?
e.g. How long (time)? How many resources used (memory)?

Total cost of problem-solving

- **Search cost ("offline") + Execution cost ("online")**

• Trade-offs often required

Search a very long time for the optimal solution, or

Search a shorter time for a "good enough" solution

More Examples

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's history	Questionnaire, tests, treatments	Healthy patient, expensive costs	Patient, hospital
Satellite image analysis system	Photos of varying intensity, color	Print a combination of photos	Image segmentation	Images from orbiting satellite
Pat-picking robot	Photos of varying intensity, color	Put up cards with insect pictures	Get rid of pests	Insects, plants, insect traps
Refinery controller	Temperature, pressure readings	Open/close valves, adjust valves	Maximize purity, yield, safety	Refinery
Interactive English tutor	Transcripts	Print exercises, magnify, summarize	Interactive predicted's Set of students score we test	Students

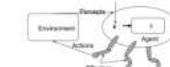
Example: Google X2: Driverless Taxi

- Percepts: video, speed, acceleration, engine status, GPS, radar, ...
- Actions: steer, accelerate, brake, horn, display, ...
- Goals (Measure): safety, reach destination, maximise profits, obey laws, passenger comfort,...
- Environment: Singapore urban streets, highways, traffic, pedestrians, weather, customers, ...

Agent

An agent is an entity that:

- **Perceives** through sensors (e.g. eyes, ears, cameras, infrared range sensors)
- **Acts** through effectors (e.g. hands, legs, motors)

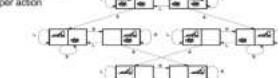


Rational Agents

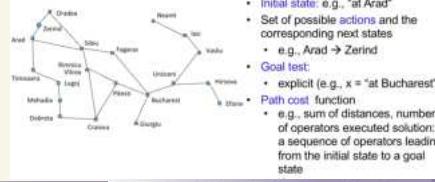
- A rational agent is one that does the **right** thing
- **Rational action**: action that maximises the expected value of an objective **performance** measure given the percept sequence to date
- Rationality depends on subject to:
 - performance measure
 - everything that the agent has perceived so far
 - built-in knowledge about the environment
 - actions that can be performed

Example: Vacuum World (Single-state Version)

- Initial state: one of the eight states shown previously
- Actions: left, right, suck
- Goal test: no dirt in any square
- Path cost: 1 per action

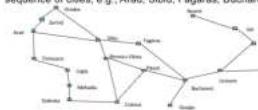


Single-State Problem Example



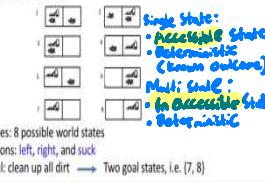
Example: Romania

- Goal: be in Bucharest
- Problem:
 - states: various cities
 - actions: drive between cities
 - goal test: city == Bucharest?
- Solution:
 - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest



Example: A Simple Vacuum World

- Two locations, each location may or may not contain dirt, and the agent may be in one location or the other



Multiple-State Problem Formulation

- Initial state set
- Set of possible actions and the corresponding sets of next states
 - e.g., Arad → Zerind
- Goal test
- Path cost function
- Solution:
 - a path (a connecting sets of states) that leads to a set of states all of which are goal states

Example: Vacuum World (Multiple-state Version)

- States: subset of the eight states
- Operators: left, right, suck
- Goal test: all states in state set have no dirt
- Path cost: 1 per operator



Week 2 mod 2

Search algo
Search strategy → efficiency

Completeness Does it always find a solution if one exists?

Time Complexity How long does it take to find a solution: the number of nodes generated

Space Complexity Maximum number of nodes in memory

Optimality Does it always find the best (least-cost) solution?

A* vs UCS

Uninformed vs Informed

Uninformed search strategies

- Use only the information available in the problem definition
- Breadth-first search is usually more efficient
- ✓ Uniform-cost search
- ✗ Depth-first search
- ✗ Depth-limited search
- ✗ Iterative deepening search

Depth-First Search

Detailed
- no maximum depth of the tree search
- uniformly expand states with the deepest f-values first
- required states characterized by their f-values
- If all states are closed, may need to backtrace

Cost per level $O(n)$

leaf $O(n^2)$

Contractive w/o

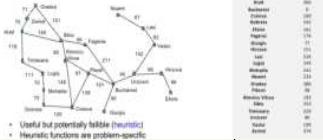
Delete last of b_n

Summary (we make assumptions for optimality)

Criterion	Breadth-first	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bi-directional (if applicable)
Time	b^d	b^d	b^m	b^l	b^l	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal	Yes	Yes	No	No	Yes	Yes

Example: Route-finding from Arad to Bucharest

$A(n) = \text{straight-line distance from } n \text{ to Bucharest}$



$A^* \text{ Search} \quad f(n) = g(n) + h(n)$

weighted A^* $f(n) = g(n) + w * h(n)$

Breadth-First Search

Expand shallowest unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue



Denote

- b: maximum branching factor of the search tree
- d: depth of the least-cost solution
- Complete: Yes
- Optimal: Yes when all step costs equal

Complexity of BFS

- Hypothetical state-space, where every node can be expanded into b new nodes, solution of path-length d
- Time: $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$
- Space: (keeps every node in memory) $O(b^d)$ are equal

Depth	Nodes	Time	Memory
0	1	1	1
1	2	2	2
2	4	4	4
3	8	8	8
4	16	16	16
5	32	32	32
6	64	64	64
7	128	128	128
8	256	256	256
9	512	512	512
10	1024	1024	1024
11	2048	2048	2048
12	4096	4096	4096
13	8192	8192	8192
14	16384	16384	16384

Uniform Cost Search (find f(n) with lowest cumulative cost) (weighted tree & graph)

Uniform-Cost Search

To consider edge costs, expand unexpanded nodes with the least path cost g

- Modification of breath-first search
- Instead of First-In-First-Out (FIFO) queue, using a priority queue with path cost $g(n)$ to order the elements
- BFS & UCS with $g(n) = \text{Depth}(n)$

Uniform-Cost Search

Time

of nodes with path cost $g \leq \text{cost of optimal solution}$
(e.g. # of nodes pop out from the priority queue)

Space

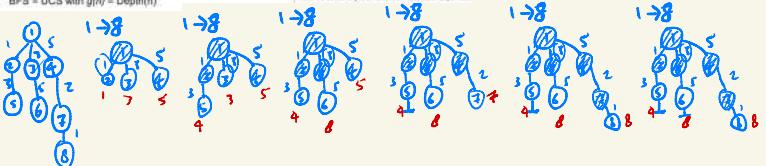
of nodes with path cost $g \leq \text{cost of optimal solution}$

Yes

No

Yes

No



! backtracking → always complete

Depth-Limited Search

To avoid infinite searching, Depth-first search with a cutoff on the max depth / of a path

Complete: Yes, if $L \geq d$

Time: $O(b^d)$

Space: $O(dL)$

Optimal: No

Iteratively estimate the max depth / of DLS one-by-one

Limit = 0



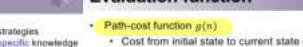
Limit = 1



Iteratively estimate the max depth / of DLS one-by-one



Limit = 3



function ITERATIVE-DEPTH-FIRST-SEARCH(problem, depth) returns a solution sequence inputs: problem, a problem

for depth = 1 to d

if DEPTH-LIMITED-SEARCH(problem, depth) succeeds then return its result

else return failure

Complete: Yes

Time: $O(d^d)$

Space: $O(dL)$

Optimal: Yes

General Search

Uninformed search strategies

- Systematic generation of new states (\rightarrow Grid-Test)
- Inefficient (exponential space and time complexity)

Greedy Search

Expands the node that appears to be closest to goal

- Evaluation function $h(n)$: estimate of cost from n to goal
- Function GREEDY-SEARCH(problem) returns solution
 - Return Best-First-Search(problem, h) if $h(goal) = 0$

Question: How to estimation the cost from n to goal?

Answer: Recall that we want to use problem-specific knowledge

A* Search

Uniform-cost search

- g(n): cost to reach n (Past Experience)
- optimal and complete, but can be very inefficient

Greedy search

- h(n): cost from n to goal (Future Prediction)
- neither optimal nor complete, but cuts search space considerably

A* Search

Idea: Combine Greedy search with Uniform-Cost search

Evaluation function: $f(n) = g(n) + h(n)$

• f(n): estimated total cost of path from n to goal (Whole Life)

• If $f = 0 \rightarrow$ greedy search. If $f = \infty$ → uniform-cost search

• Function A* (SEARCHproblem) returns solution

• Return Best-First-Search(problem, f)

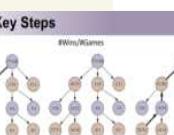
• Denote: $f(n) = g(n) + h(n)$

(d) After expanding Rimnicu Vilcea

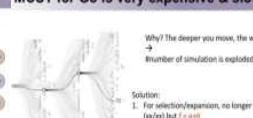
(e) After expanding Fagaras

(f) After expanding Pitesti

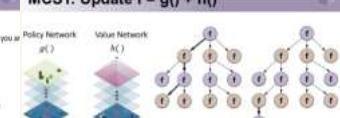
MCTS Key Steps



MCTS for Go is very expensive & slow



MCTS: Update $f = g() + h()$



WEEK 3

Constraint Satisfaction Problem (CSP): goal : discover state that satisfies a given set of constraints

Examples: Real-world CSPs

- Assignment problems
 - e.g. who teaches what class
- Timetabling problems
 - e.g. which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor-planning

A state of the problem is defined by an assignment of values to variables or all variables assigned.

An assignment that does not violate any constraints is called a **consistent** or **legal assignment**.

A solution to a CSP is an assignment with every variable having a value (complete) and the assignment satisfies all the constraints.

Applying Standard Search

- State:** defined by the values assigned so far
- Initial state:** all variables unassigned
- Actions:** assign a value to an unassigned variable
- Goal test:** all variables assigned, no constraints violated

Backtracking Search

Backtracking search: Do not waste time searching when constraints have already been violated

Initial State
S1
Go back to the last decision point
S2
Something goes wrong.
S3

- Before generating successors, check for constraint violations
- If yes, backtrack to try something else

Example: Map Colouring

WA NT Q NSW V SA T

Example: Cryptarithm

Question: How to represent constraints?
Answer: Example (e.g., 4-E)

Example: Row the 14 queen occupies $V_i \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ (similarity for V_j)
Non-attack constraint for V_i and V_j :
 $|i - j| > 1$, $i + 1 < 8 - j$, $|i - j| > 2$, $|i - j| > 3$, ...

Implicitly: use a function to test for constraint satisfaction

without Constraint Propagation

Example: 4-Queens

Search Tree of 4-Queens with Constraint Propagation

Heuristics for CSP

Plain backtracking is an uninformed algorithm!
More intelligent search that takes into consideration:

- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
- Forward checking and constraint propagation

Constraint propagation: propagating the implications of a constraint on one variable onto other variables.

Example (Map Colouring)

WA NT Q NSW V SA T

Example (Map Colouring)

WA NT Q NSW V SA T

Example (Map Colouring)

WA NT Q NSW V SA T

Example (Map Colouring)

WA NT Q NSW V SA T

Most Constraining Variable

Example: map colouring

To reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on unassigned variables.

Least Constraining Value

Allow 1 value for SA
Allow 0 values for SA

Choose the value that leaves maximum flexibility for subsequent variable assignments

Min-Conflicts Heuristic (8-queens)

- A local heuristic search method for solving CSPs
- Given an initial assignment, selects a variable in the scope of a violated constraint and assigns it to the value that minimises the number of violated constraints

DFS, IDS, BFS, A*, Greedy, UCS

Most suitable search algorithm:

~~XBFS~~ ~~DFS, IDS, A*~~ ~~DFS~~ ~~IDS~~ ~~XDFS~~ ~~Greedy~~ ~~A*~~ ~~optimal BFS, UCS, A*, IDS~~ ~~minimum no. of states~~

a) "very large search space": DFS, IDS, or A* - not BFS
"large branching factor" : DFS or IDS
"possibly infinite paths" : not DFS
"no heuristic function" : not A* et al
"minimum no. of states" : optimal, so BFS or IDS

→ Iterative Deepening Search (IDS) is best

b) "lots of cycles" : not DFS
"varying costs" : UCS or A* - not BFS, DFS
"no heuristic function" : not A* et al
"shortest path" : optimal, so UCS or A*

→ Uniform Cost Search (UCS / Dijkstra)

c) "fixed depth tree" : DFS, others
"goals at the bottom" : DFS - not BFS or IDS
"heuristic function" : Greedy Best First, not DFS
"find any goal quickly" : not opt., DFS or greedy

Best First

→ Greedy Best First Search is best

note: local search algorithms such as Hill-climbing may fail to find a goal (no backtracking)

Explain which search algorithm is most appropriate in the following situations:

- We have a very large search space with a large branching factor:
DFS (V), DLS (V), IDS (V), BFS (X), UCS (X), Informed Search (V)
- and with possibly infinite paths:
DFS (X), DLS (V), IDS (V), BFS (X), UCS (X), Informed Search (V)
- We have no heuristic function:
DFS (X), DLS (V), IDS (V), BFS (X), UCS (X), Informed Search (X)
- We want to find a path to the goal with minimum number of states (i.e. implying optimality):
DFS (X), DLS (X), IDS (V), BFS (X), UCS (X), Informed Search (X)

Iterative Deepening Search is the most suitable

Week 4

Games as Search Problems

- Absorption
 - total representation of real world problems
 - e.g. board games, chess, go, etc. as an abstraction of war games
 - Perfect information, i.e. fully observable
 - Accurate formulation: state space representation
- Uncertainty
 - Account for the existence of hostile agents (players)
 - Other agents acting so as to diminish the agent's well-being
 - Uncertainty (about other agents' actions):
 - not due to the effect of non-deterministic actions
 - not due to randomness
 - Contingency problem

Games as Search Problems...

Complexity

- Games are abstract but not simple
 - e.g. chess: average branching factor = 35, game length > 5!
 - complexity = 35^{10} (only 10¹⁰ for legal moves)
- Games are usually time limited
 - Complete search (for the optimal solution) not possible
 - Uncertainty on action desirability
 - Search efficiency is crucial

Types of Games

Deterministic

- Chess, Checkers, Go, Othello

Chance

- Backgammon, Monopoly

Imperfect Information

- Bridge, Poker, Scrabble, Nuclear war

In this course, we only focus on Perfect information

each player has complete information about his opponent and about the choices available to him

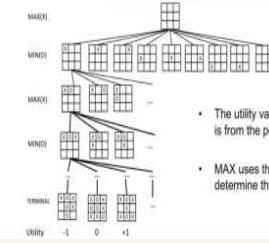
Game as a Search Problem

- Initial state: initial board configuration and indication of who makes the first move
- Operator: legal moves
- Terminal test: determines when the game is over
- State: state where the game has ended; terminal states
- Utility function (payoff function): returns a numeric score to quantify the outcome of a game

Example: Chess

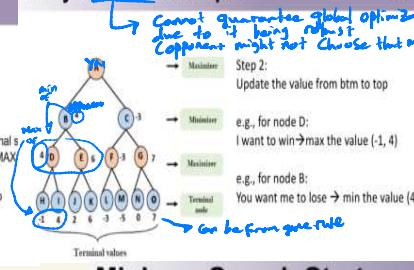
Win (+1), lost (-1) or draw (0)

Game Tree for Tic-Tac-Toe



- The utility value of the terminal state is from the point of view of MAX
- MAX uses the search tree to determine the best move

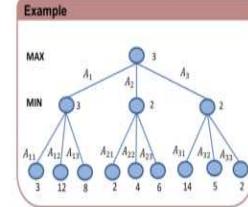
A Toy Min-Max Example



Perfect Decisions by Minimax Algorithm

Perfect decisions: no time limit is imposed

Example



Two players: MAX and MIN

- Choose move with best achievable payoff against best play
- MAX tries to max the utility, assuming that MIN will try to min it

Minimax Search Strategy

Search strategy

- Find a sequence of moves that leads to a terminal state (goal)

Minimax search strategy

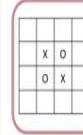
- Maximise one's own utility and minimise the opponent's
 - Assumption is that the opponent does the same

Minimax Search Strategy

3-step process

- Generate the entire game tree down to terminal states
- Calculate utility
 - Assess the utility of each terminal state
 - Determine the best utility of the parents of the terminal state
 - Repeat the process for their parents until the root is reached
- Select the best move (i.e. the move with the highest utility value)

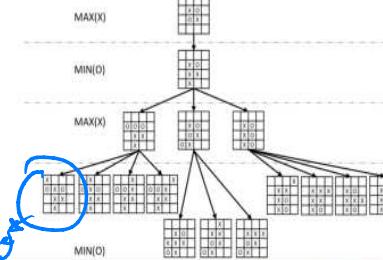
Othello 4



- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them
- After placing the new piece, the pieces from the opponent player will be captured and become the pieces from the same Player
- The player with the most pieces on the board wins

Need evaluation function (for huge game tree)

Game Tree Othello 4



Evaluation Functions for Othello 4

- A corner of the board is one of the most important positions. A piece at the corner can help capture other pieces from the opponent player.
- A square at the border is also more important than any position in the middle of the board

Heuristics for 'X' is proposed as follows:

- For any non-terminal game state, the evaluation function is computed as

$$3(X_c - O_c) + 2(X_b - O_b) + (X_m - O_m)$$

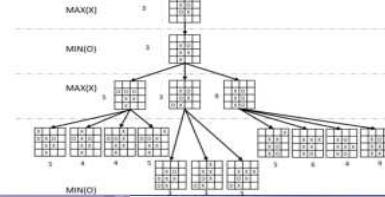
where X_c is the number of 'X's at the corners.

X_b is the number of 'X's at the border (excluding corners).

X_m is the number of 'X's in the middle of the board.

O_c , O_b and O_m are the number of 'O's at the corners, the border and in the middle of the board.

Minimax Search for Othello 4



Imperfect Decisions

For chess, branching factor = 35, each player typically makes 50 moves \rightarrow for the complete game tree, need to examine 35^{50} positions

Time/space requirements \rightarrow complete game tree search is intractable \rightarrow impractical to make perfect decisions

Modification: to minimax algorithm

- replace utility function by an estimated desirability of the position

Evaluation function

- partial tree search

E.g., depth limit

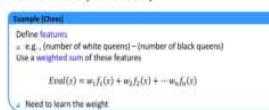
- Replace terminal test by a *cut-off* test

Evaluation Functions...

Requirements

- Computation is efficient
- Agrees with utility function on terminal states
- Accuracy reflects the chances of winning

Trade off between accuracy and efficiency



Common use of DNN

Quiescent Search

The evaluation function should only be applied to quiescent positions

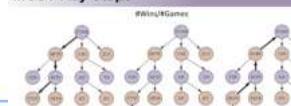
- positions that are not likely to have large variations in evaluation in the near future

Example (Chess)

Positions in which favorable captures can be made



MCST Key Steps

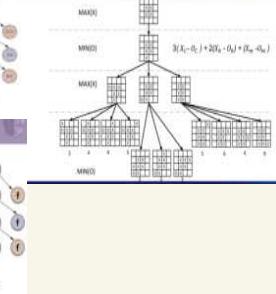


MCST: Update f = g() + h()

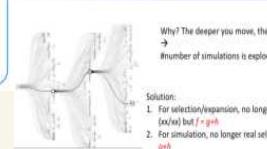
Policy Network



Evaluation Functions for Othello 4 ...



MCST for Go is very expensive & slow



Solution:

- For selection/expansion, no longer stats $(x|y)$ but $f(x|y)$
- For simulation, no longer real self-play but $f(x|y)$

Conclusion:

- For selection/expansion, no longer stats $(x|y)$ but $f(x|y)$
- For simulation, no longer real self-play but $f(x|y)$

WLS 5 NY Reinforcement Learning

(It is no function model fitting)

- Motivation
 - In last lecture, we compute the value function and find the optimal policy
 - But without the transition function $P(s'|s, a)$
 - We can learn the value function and find the optimal policy without transition
 - From experience



What is Monte Carlo

- Idea behind MC:
 - Just use randomness to solve a problem
- Simple definition:
 - Solve a problem by generating suitable random numbers and observing the fraction of numbers obeying some properties
- An example for calculating π (not problem in RL):
 - $\pi_{MC} = \frac{1}{N} \sum_{i=1}^N \delta_{S_i, \pi}$, $\delta_{S_i, \pi} = 1$
 - putting dots in the square randomly for $N = 3000$ times
 - $\pi = \pi \times \frac{\text{dots}}{N}$, N_{dot} is the number of dots in the circle

Sampling Monte Carlo

Bootstrapping S-learning

DFA for Function Approx.

DFA

Lesson outline

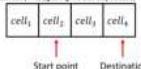
- RL algo - Monte-Carlo
- Temporal Diff
- Q-learning
- Deep Q-learning

Monte Carlo in RL: Prediction

- Basic Idea: we run in the world randomly and gain experience to learn what? Experience? Many trajectories!
 - $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \dots$
 - What we learn? Value function!
 - Recall that the return is the total discounted rewards:
- $$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^n r_{t+n} + \dots = \sum_i \gamma^i r_{t+i}$$
- Recall that the value function is the expected return from π :
- $$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$
- How we learn?
 - Use experience to learn an empirical state value function $\hat{V}_t(s) = \frac{1}{N} \sum_{i=1}^N G_i$

An Example

- One-dimensional grid world
 - A robot is in a 1x4 world
 - State: current cell $s \in [cell_1, cell_2, cell_3, cell_4]$
 - Action: left or right
 - Reward:
 - Move one step (+1)
 - Reach the destination cell (+10) (ignoring the one-step reward)



First Visit Monte Carlo Policy Evaluation

- Average returns only for the first time s is visited in an episode
- Algorithm
 - Initialize:
 - π — policy to be evaluated
 - V — an arbitrary state-value function
 - $Return(s)$ — an empty list, for all states s
 - Repeat many times:
 - Generate an episode using π
 - For each state s appearing in the episode:
 - $R \leftarrow$ return following the first occurrence of s
 - Append to $Return(s)$
 - $V(s) \leftarrow \text{average}(Return(s))$

One-dimensional Grid World

- Trajectory or episode:
 - The sequence of states from the starting state to the terminal state
 - Robot starts in $cell_1$, ends in $cell_4$
- The representation of the three episodes



Compute Value Function

- Idea: Average return observed after visits to (s, a)
- First-visit MC: average returns only for first time (s, a) is visited in an episode
- Return in one episode (trajectory):

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^n r_{t+n} + \dots = \sum_i \gamma^i r_{t+i}$$
- We calculate the return for $cell_1$ of first episode with $\gamma = 0.9$



$$G_1 = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$

- Similarly the return for $cell_2$ of second episode with $\gamma = 0.9$



$$G_2 = -1 \times 0.9^0 - 1 \times 0.9^1 - 10 \times 0.9^2 = -10 \times 0.9^3 = -8.38$$

- Similarly the return for $cell_3$ of third episode with $\gamma = 0.9$



$$G_3 = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 - 10 \times 0.9^3 = -10 \times 0.9^4 = -5.72$$

- The empirical value function for $cell_3$ is $\frac{-8.38 + -5.72}{2} = 5.72$
- Given these three episodes, we compute the value function for all non-terminal state



- We can get more accurate value function with more episodes



Computing Q-value

- MC for estimating Q:
 - A slight difference from estimating the value function
 - Average returns for state-action pair (s, a) is visited in an episode
- We calculate the return for $(cell_1, \text{right})$ of first episode with $\gamma = 0.9$



$$G_1 = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$

- Similarly the return for $(cell_1, \text{right})$ of second episode with $\gamma = 0.9$



$$G_2 = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 + 10 \times 0.9^3 = -10 \times 0.9^4 = -8.38$$

- Similarly the return for $(cell_1, \text{right})$ of third episode with $\gamma = 0.9$



$$G_3 = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 - 10 \times 0.9^3 = -10 \times 0.9^4 = -5.72$$

- The empirical Q-value function for $(cell_1, \text{right})$ is $\frac{-8.38 + -5.72}{2} = 5.72$



Deep Q-Network

- Previously, we represent the Q-value as a table
- However tabular representation is insufficient
 - Many real-world problems have enormous state and/or action spaces
 - Backgammon: 10³⁰ states
 - Computer Go: 10¹⁷⁰ states
 - Robots: continuous state space
- We use a neural network as a black box to replace the table



$$q(s, a) = \text{argmax}_a q^*(s, a)$$

- At $cell_1, cell_2$ and $cell_3$, we choose right



Q-Learning

Q-Learning

- Previously, we need the whole trajectory
- In Q-Learning, we only need one-step trajectory (s, a, r, s')
- The difference is the Q-value computing
 - Previously:

$$\hat{Q}_s(a) = \frac{1}{N} \sum_{i=1}^N G_i$$

- Now, updating rule:

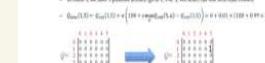
$$Q_{\text{old}}(s, a) \leftarrow Q_{\text{old}}(s, a) + \alpha(r_{t+1} + \gamma \max_{a'} Q_{\text{old}}(s_{t+1}, a')) - Q_{\text{old}}(s, a))$$
 - new estimation
 - learning rate
 - new sample
 - old estimation



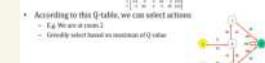
Q-Learning Step by Step

- Initialize matrix Q as a zero matrix
- $a = \text{argmax}_a Q(s, a)$
- Loop for each episode until converge
 - Initial state: we are in room 1 (1st floor)
 - Loop for each step of episode (until reach room 5)
 - ... (one step)
- Loop for each step of episode (until room 5)
 - By random selection, we get $a = 1$
 - We move to room 2
 - Update $Q_{\text{old}}(s_1, a_1) \leftarrow Q_{\text{old}}(s_1, a_1) + \alpha(r_{t+1} + \gamma \max_{a'} Q_{\text{old}}(s_{t+1}, a')) - Q_{\text{old}}(s_1, a_1))$
 - At room 2, we have 3 possible actions: go 1, 2 or 3. We select the one with the highest Q-value.
 - $= Q_{\text{old}}(s_1, 1) + \max_{a'} Q_{\text{old}}(s_1, a') = (10 + \gamma \max_{a'} Q_{\text{old}}(s_1, a')) - Q_{\text{old}}(s_1, 1)$

- When we loop many episodes, we can get



- According to this Q-table, we can select actions
 - E.g. We are in room 2
 - Greedily select the maximum of Q-value



QNN in Atari

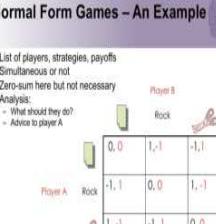
- Input state s is stack of raw pixels from last 4 frames
- Output is $q(s, a)$ for 18 button
- Reward is change in score for that step
- Pong's video: <https://www.youtube.com/watch?v=tPSQGK6k07vN>
- Beat human on many games



Game Theory

What is Game Theory?

- Game theory studies settings where multiple parties (agents) each have:
 - different preferences/utility functions
 - different actions that they can take
- Each agent's utility (potentially) depends on all agents' actions
- What is optimal for one agent depends on what other agents do
- Game theory studies how agents can rationally form beliefs over what other agents will do, and (hence) how agents should act
- Useful for predicting as well as predicting behavior of others



Nash Equilibrium

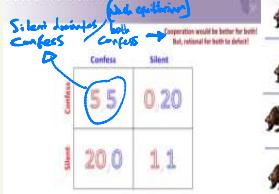
- Each agent is selfish
- Each agent makes decision based on what he thinks others would do
- No one can do better by changing strategy solely
- Any game with finite actions must have a Nash Equilibrium.



Prisoner's Dilemma

		Confess		remain silent	
		Confess	remain silent	Confess	remain silent
		Confess	5 years	0 years	20 years
		remain silent	0 years	5 years	20 years
		Confess	5 years	0 years	20 years
		remain silent	0 years	5 years	20 years

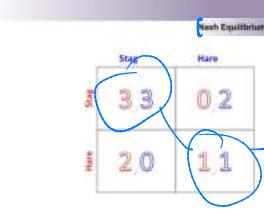
Game Abstraction



Nash Equilibrium



Game Abstraction



Nash Equilibrium: Pure Strategy (Set)

		Confess	Silent	Stag	Hare
		Confess	Silent	Stag	Hare
		5, 5	0, 20	3, 3	0, 2
		20, 0	1, 1	2, 0	1, 1

Nash Equilibrium: Mixed Strategy (Set)



Look at Red when Blue "Paper"

		Paper	Rock	Scissors	
		Paper	1/3	1/3	1/3
		Paper	0, 0	1, -1	-1, 1
		Rock	-1, 1	0, 0	1, -1
		Scissors	1, -1	-1, 1	0, 0

Nash Equilibrium: Mixed Strategy (Set)

		Paper	Rock	Scissors	
		Paper	1/3	1/3	1/3
		Paper	0, 0	1, -1	-1, 1
		Rock	-1, 1	0, 0	1, -1
		Scissors	1, -1	-1, 1	0, 0

NE is a "natural law"

Go	Stop
-5, -5	10
0, 1	-1, -1

Games: Complete Information v.s. Incomplete Information

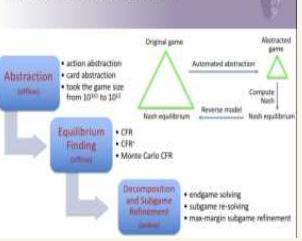


Texas Hold'em Poker



- Pre flop: two private cards are dealt to each player, followed by a betting round; players can either check, bet or fold
- Flop: three public cards are dealt, followed by a betting round
- Turn: a fourth public card is dealt, followed by a betting round
- River: a last public card is dealt, followed by a betting round
- The game ends when:
 - Only one player is left, all the other players fold
 - A Showdown: a hand with the best 5 cards using both the two private cards and the five public cards wins

Architecture of Libratus



Global Challenges for Security



Key challenges: Limited resources, surveillance

Stackelberg Games

Randomization: Increase Cost and Uncertainty to Attackers



PROTECT: Randomized Patrol Scheduling



Game Theory for Security: Applications



IRIS: Federal Air Marshals Service [2009] Scale Up Number of Defender Strategies



- 1000 Flights, 20 air marshals: 10^{11} combinations
- ARMOR: 8 Terminals Actions: "100"
- Not enumerate all combinations combinations
 - Branch and price
 - Branch and bound
 - Column generation

PAWS: Protection Assistant for Wildlife Security Trials in Uganda and Malaysia [2014]



PAWS Deployed in 2015 in Southeast Asia (with Panthera and WWF)

PAWS Version 2: Features



Week 11a – Default Logic Learning Goals

Unicorns

Default Logic

Understanding the:

- Basics of Reiter's Default Logic (RDL)
- Limitations of RDL

Default Logic

- Standard logic can only express that something is true or that something is false.
- Default logic is proposed to formalize reasoning with default assumptions.
- It can express facts like “by default, something is true”.

Given:

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

Question:

- Based on Modus Ponens, can we show that the unicorn is mythical, magical and horned?

KB:

$$\text{Mythical} \Rightarrow \neg \text{Mortal}$$

$$\neg \text{Mythical} \Rightarrow \text{Mortal}$$

$$\neg \text{Mythical} \Rightarrow \text{Mammal}$$

$$\neg \text{Mortal} \Rightarrow \text{Horned}$$

$$\neg \text{Mammal} \Rightarrow \text{Horned}$$

$$\neg \text{Horned} \Rightarrow \text{Magical}$$

$$\neg \text{Mythical} \Rightarrow \neg \text{Horned}$$

$$\neg \text{Horned} \Rightarrow \neg \text{Magical}$$

Assumptions

I can believe	Mythical	or	$\neg \text{Mythical}$
	Mythical		$\neg \text{Mythical}$
	$\neg \text{Mortal}$		$\neg \text{Mythical}$
	Horned		Mammal
	Magical		$\neg \text{Horned}$

where $x = x_1, \dots, x_m$, and $\alpha(x), \beta_1(x), \dots, \beta_n(x), \gamma(x)$ are formulae whose free variables are among x_1, \dots, x_m .

Prerequisite : Justification₁, ..., Justification_n
Conclusion

According to this default:

- If we believe that Prerequisite is true;
- AND each of Justifications is consistent with our current beliefs;
- THEN, we are led to believe that the Conclusion is true.

Default Logic Syntax

$\alpha(x) : \beta_1(x), \dots, \beta_n(x) \quad | \quad \gamma(x)$

where $x = x_1, \dots, x_m$, and $\alpha(x), \beta_1(x), \dots, \beta_n(x), \gamma(x)$ are formulae whose free variables are among x_1, \dots, x_m .

The default is applied by substituting **c** (the ground instance) into α and β to infer γ :

- Trigger: $\alpha(c)$ belongs to our set of beliefs.
- Justification: the set of our beliefs is consistent with each $\beta(c)$.

Our KB does not entail $\neg\beta(c)$

Default Theory

Example – the default rule that “birds typically fly”:

- $\Delta = \{\text{bird}(x) \rightarrow \text{fly}(x)\}$
- This rule means that, “if x is a bird, and it can be assumed that x is, then we can conclude that x flies.”
- $\Phi = \{\text{bird}(\text{Fever}), \text{cut}(\text{Swimmer})\}$

A common default assumption:

- What is not known to be true is believed to be false;
- This is known as the Closed-World Assumption.
- It is formalized in default logic using a default as follows for every fact F :

$$\neg F \quad | \quad \neg F$$

Types of Defaults

Normal Defaults: $\frac{\alpha(x) : \beta(x)}{\gamma(x)}$

Semi-Normal Defaults: $\frac{\alpha(x) : \beta(x)}{\gamma(x)}$, where $\beta(x) \simeq \gamma(x)$

Open Defaults (Default Schemas) have unbounded variables, e.g., x

Closed (Grounded) Defaults use ground terms, e.g., $\neg c$

Nixon Diamond

The default rules may be applied in different orders, and this may lead to different extensions. E.g.:

Let Theory $T = (\Delta, \Phi)$

$$\begin{cases} \Delta = \\ \quad \text{quaker}(x) : \text{pacifist}(x), \text{republican}(x) \dashv \neg \text{pacifist}(x) \\ \quad \text{pacifist}(x) \\ \Phi = \text{quaker}(\text{Nixon}), \text{republican}(\text{Nixon}) \end{cases}$$

Given (Δ, Φ)

- $\Delta =$

 - $\text{quaker}(x) : \text{pacifist}(x)$, $\text{republican}(x) \dashv \neg \text{pacifist}(x)$
 - $\text{pacifist}(x)$

- $\Phi = \text{quaker}(\text{Nixon}), \text{republican}(\text{Nixon})$

There are **two extensions**:

- One that contains: $\text{pacifist}(\text{Nixon})$
- ... and the one that contains: $\neg \text{pacifist}(\text{Nixon})$

Reiter Inference with Default Theory

Given a default theory (Δ, Φ) , Ξ is an extension if and only if

$$\Xi = \text{Co}\left(\bigcup_{x \in \Xi} \Xi_x\right)$$

where:

- $\Xi_0 = \Phi$
- $\Xi_{n+1} = \Xi_n \cup \{y(x) \mid \exists x \in \Xi_n \forall x \in \Xi_n \alpha(x) \rightarrow \beta(x) \in \Delta, \Xi_n \vdash \alpha(x), \neg\beta(x), \dots, \neg\beta(x) \in \Xi\}$

Reiter Extension: Compact Description

Given a default theory (Δ, Φ) , Ξ is an extension if and only if

$$\Xi = \text{Co}\left(\bigcup_{x \in \Xi} \Xi_x\right)$$

where:

- $\Xi_0 = \Phi$
- $\Xi_{n+1} = \Xi_n \cup \{y(x) \mid \exists x \in \Xi_n \forall x \in \Xi_n \alpha(x) \rightarrow \beta(x) \in \Delta, \Xi_n \vdash \alpha(x), \neg\beta(x), \dots, \neg\beta(x) \in \Xi\}$

Example

Given Theory: $T = \langle \Delta = \{\text{bird}(x) \rightarrow \text{fly}(x)\}, \Phi = \{\text{bird}(\text{Fever}), \text{cut}(\text{Swimmer})\} \rangle$

- Guesses the extension $\Xi = \{\text{Cn}(\{\text{bird}(x) \rightarrow \text{fly}(x)\})\}$
- Initialise beliefs $\Xi^* = \emptyset$
- Is there an α -ground instance of an (unused) default $\alpha(x) : \beta(x)$?
- Check two conditions:
 - Triggered?: $\Xi^* \vdash \alpha(x)$
 - Justified?: $\beta(x)$ is consistent with Ξ
- If yes: update beliefs $\Xi^* \leftarrow \Xi^* \cup \{\beta(x)\}$
- (end loop)
- If $\Xi^* = \Xi$ then extension found/confirmed

Entailment

The Nixon diamond example theory has two extensions:

- one in which Nixon is a pacifist, and
- one in which Nixon is not a pacifist.

Thus, we have:

- Neither $\text{pacifist}(\text{Nixon})$ nor $\neg \text{pacifist}(\text{Nixon})$ are skeptical
- Both $\text{pacifist}(\text{Nixon})$ and $\neg \text{pacifist}(\text{Nixon})$ are credulously entailed.

The credulous extensions of a default theory can be inconsistent with each other.

Makinson Approach

Given a default theory $T = (\Delta, \Phi)$, let $\Pi = (\delta_0, \delta_1, \dots)$ be a (finite or infinite) sequence of (closed) defaults from Δ without multiple occurrences.

Extension

Given a default theory $T = (\Delta, \Phi)$, let $\Pi[k]$ denotes the initial segment of sequence Π with length k .

Each sequence Π is associated with two sets:

- $\text{In}(\Pi) = \text{Cn}(\Phi \cup \{\text{consequence}(\delta) \mid \delta \text{ occurs in } \Pi\})$
- $\text{Out}(\Pi) = \{\neg\phi \mid \phi \in \text{justifications}(\delta) \text{ for some } \delta \text{ in } \Pi\}$

Everything I know denotes the existence of at least one of my assumptions (\neg the negation of all the justifications).

Process Tree

Process Tree: Properties

Operational Semantics

Example

Given a default theory $T = (\Delta, \Phi)$, let $\Pi = (\delta_0, \delta_1, \dots)$ be a (finite or infinite) sequence of (closed) defaults from Δ without multiple occurrences.

$\Pi[k]$ denotes the initial segment of sequence Π with length k .

Each sequence Π is associated with two sets:

- $\text{In}(\Pi) = \text{Cn}(\Phi \cup \{\text{consequence}(\delta) \mid \delta \text{ occurs in } \Pi\})$
- $\text{Out}(\Pi) = \{\neg\phi \mid \phi \in \text{justifications}(\delta) \text{ for some } \delta \text{ in } \Pi\}$

Given a default theory $T = (\Delta, \Phi)$ let Π be a process tree. Π is a tree $G = (V, E)$ with the set of nodes V & E are labelled with two sets of formulae:

- $\text{In}(\Pi)$ and $\text{Out}(\Pi)$
- $\text{In}(v)$ and $\text{Out}(v)$

The root of Π is labelled with $\{\text{Cn}(\Phi)\}$ as the in-set and Π as the out-set. If v is a node, then v denotes a default application and is labelled by it.

Process Tree

Process Tree: Properties

Example

Given a process tree Π and a default theory $T = (\Delta, \Phi)$, let $\text{In}(\Pi)$ and $\text{Out}(\Pi)$ be the sets of formulae in Π respectively.

- $\text{In}(\Pi)$ is a path in G starting from the root.
- A node $v \in \Pi$ is expanded if $\text{In}(\Pi(v)) \cap \text{Out}(\Pi) = \emptyset$. Otherwise, it is “tainted” off the tree.
- Expanded = If a child node, for every $\delta \in \text{In}(\Pi(v))$
 - δ does not appear in the path from the root to v .
 - Otherwise, it is “tainted” off the tree.
- $\text{Out}(\Pi)$ is the set of formulae in Π that do not appear in any node of Π .
- $\text{Out}(\Pi)$ is the set of formulae in Π that do not appear in any node of Π .

Process Tree Example

Given $T = \langle \Delta = \{\text{bird}(x) \rightarrow \text{fly}(x)\}, \Phi = \{\text{bird}(\text{Fever}), \text{cut}(\text{Swimmer})\} \rangle$

Let Π be a process tree. Π is an AND-parallel process tree with the following structure:

```

graph TD
    Root["{Cn(Φ)}"] -- "δ₀: {bird(x) → fly(x)}" --> N1["{bird(Fever)}"]
    Root -- "δ₀: {bird(x) → fly(x)}" --> N2["{bird(Swimmer)}"]
    N1 -- "δ₁: {bird(x) → fly(x)}" --> N3["{bird(Fever)}"]
    N1 -- "δ₁: {bird(x) → fly(x)}" --> N4["{bird(Swimmer)}"]
    N3 -- "δ₂: {bird(x) → fly(x)}" --> N5["{bird(Fly)}"]
    N4 -- "δ₂: {bird(x) → fly(x)}" --> N6["{bird(Fly)}"]
  
```

$\text{In}(\Pi) = \{\text{bird}(x) \rightarrow \text{fly}(x)\}$

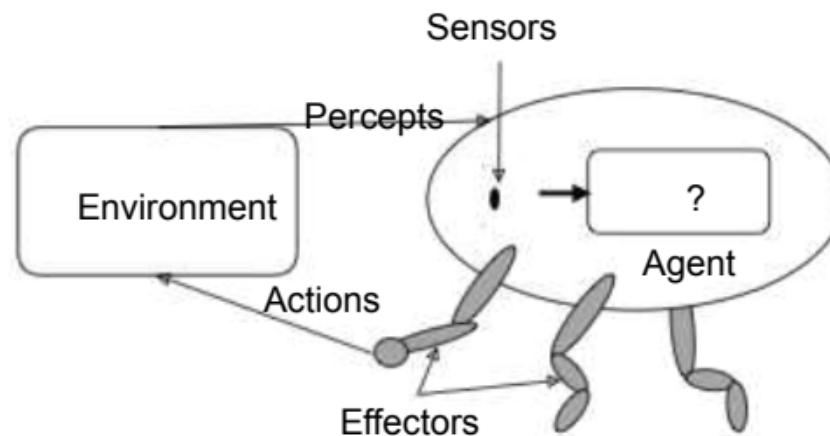
$\text{Out}(\Pi) = \{\neg\text{bird}(\text{Fever}), \neg\text{bird}(\text{Swimmer}), \neg\text{bird}(\text{Fly})\}$



Agent

An **agent** is an entity that

- **Perceives** through sensors (e.g. eyes, ears, cameras, infrared range sensors)
- **Acts** through effectors (e.g. hands, legs, motors)





Rational Agents

- A rational agent is one that does the **right** thing
- **Rational action**: action that maximises the expected value of an objective **performance** measure given the percept sequence to date
- Rationality depends on/subject to:
 - performance measure
 - everything that the agent has perceived so far
 - built-in knowledge about the environment
 - actions that can be performed





Example: Google X2: Driverless Taxi

- **Percepts:** video, speed, acceleration, engine status, GPS, radar, ...
- **Actions:** steer, accelerate, brake, horn, display, ...
- **Goals (Measure):** safety, reach destination, maximise profits, obey laws, passenger comfort, ...
- **Environment:** Singapore urban streets, highways, traffic, pedestrians, weather, customers, ...

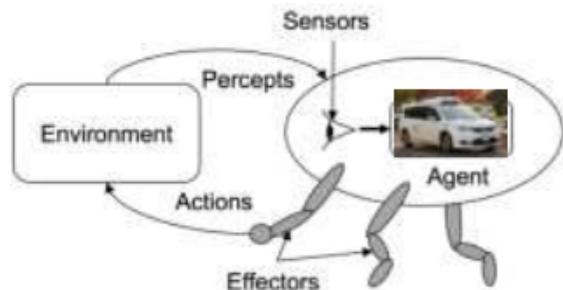


Image source: https://en.wikipedia.org/wiki/Waymo#/media/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg





More Examples

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sorts into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery





More Examples

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes



Design of Problem-Solving Agent

Idea

- Systematically considers the **expected outcomes** of different possible sequences of actions that lead to states of known value
- Choose the best one
 - shortest journey from A to B?
 - most cost effective journey from A to B?





Design of Problem-Solving Agent

Steps

1. Goal formulation
2. Problem formulation (states, actions, goal test)
3. Solution (a sequence of legal action execution)
4. Search (algorithm of how to find a good solution)
 - No knowledge uninformed search
 - Knowledge informed search





Example: Romania

- Goal: be in Bucharest
- Problem:
 - states: various cities
 - actions: drive between cities
 - goal test: city == Bucharest?
- Solution:
 - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest



Example: Vacuum Cleaner Agent

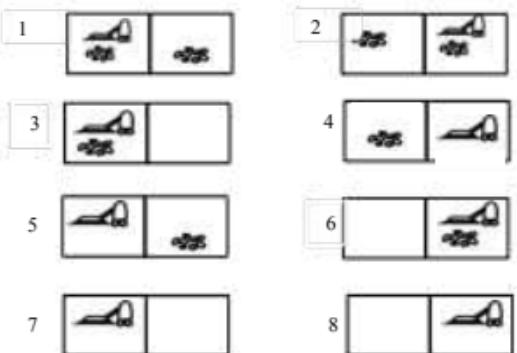
- Robotic vacuum cleaners move autonomously
- Some can come back to a docking station to charge their batteries
- A few are able to empty their dust containers into the dock as well





Example: A Simple Vacuum World

- Two locations, each location may or may not contain dirt, and the agent may be in one location or the other



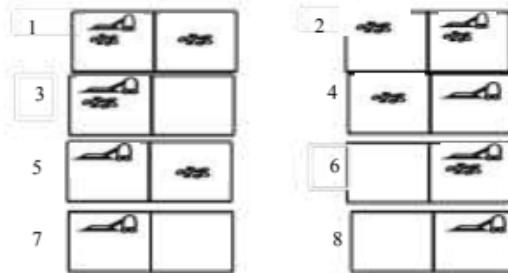
- States: 8 possible world states
- Actions: **left**, **right**, and **suck**
- Goal: clean up all dirt → Two goal states, i.e. {7, 8}





Single-State Problem

- Accessible world state (sensory information is available)
- Known outcome of action (deterministic)



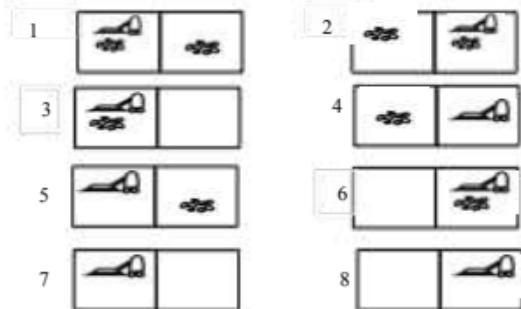
- e.g.: start in #5
- Solution: **right, suck**





Multiple-State Problem

- Inaccessible world state (with limited sensory information):
 - agent only knows which sets of states it is in
- Known outcome of action (deterministic)



- e.g.: start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
- Action right goes to $\{2, 4, 6, 8\}$





Well-Defined Formulation

Definition of a problem	Re-state the goal with more specification (rules, resource, see below)
Formulation	<ul style="list-style-type: none">Initial stateAction set, i.e. available actions (successor functions)State space, i.e. states reachable from the initial stateSolution path: sequence of actions from one state to anotherGoal test predicateSingle state, enumerated list of states, abstract propertiesCost functionPath cost $g(n)$, sum of all (action) step costs along the path
Solution	A path (a sequence of operators leading) from the Initial-State to a



Measuring Problem-Solving Performance

Search Cost

- What does it cost to find the solution?
- e.g. How long (time)? How many resources used (memory)?

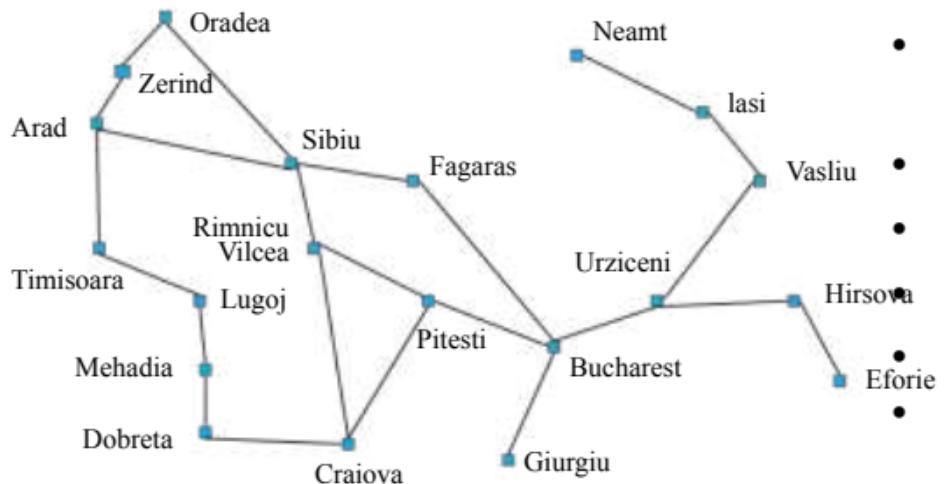
Total cost of problem-solving

- Search cost ("offline") + Execution cost ("online")
- Trade-offs often required
 - Search a very long time for the optimal solution, or
 - Search a shorter time for a "good enough" solution





Single-State Problem Example

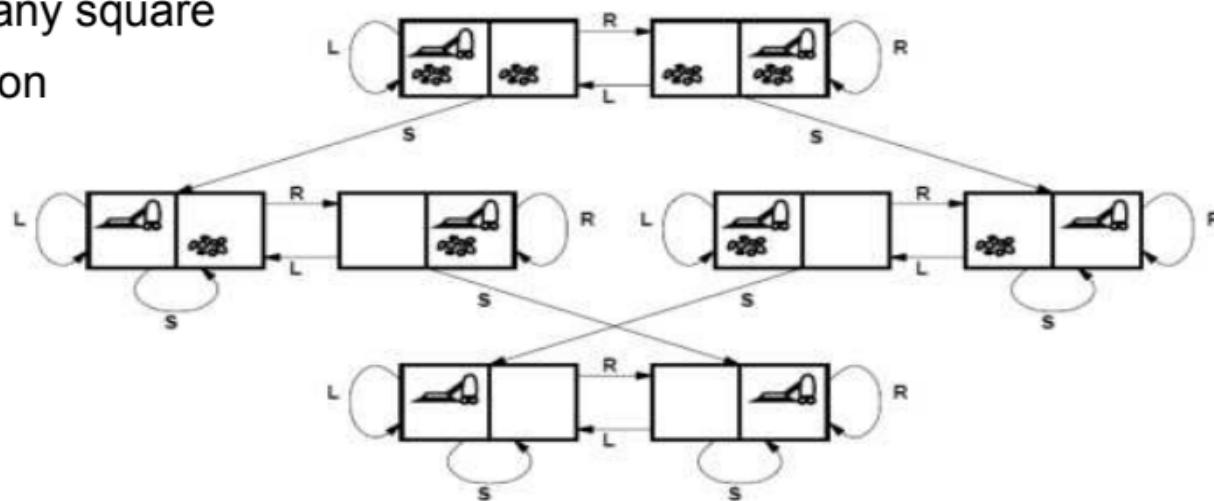


- **Initial state:** e.g., “at Arad”
- Set of possible **actions** and the corresponding next states
 - e.g., Arad Zerind
 -
 -
 -
 -
 -
 -
- **Goal test:**
 - explicit (e.g., $x = \text{"at Bucharest"}$)
- **Path cost** function
 - e.g., sum of distances, number of operators executed
 - solution: a sequence of operators leading from the initial state to a goal state

Example: Vacuum World (Single-state Version)



- Initial state: one of the eight states shown previously
- Actions: left, right, suck
- Goal test: no dirt in any square
- Path cost: 1 per action





Multiple-State Problem Formulation

- Initial state **set**
 - Set of possible actions and the corresponding **sets of** next states
 - Goal test
 - Path cost function
-
- **Solution:**
 - a path (connecting sets of states) that leads to a set of states all of which are goal states





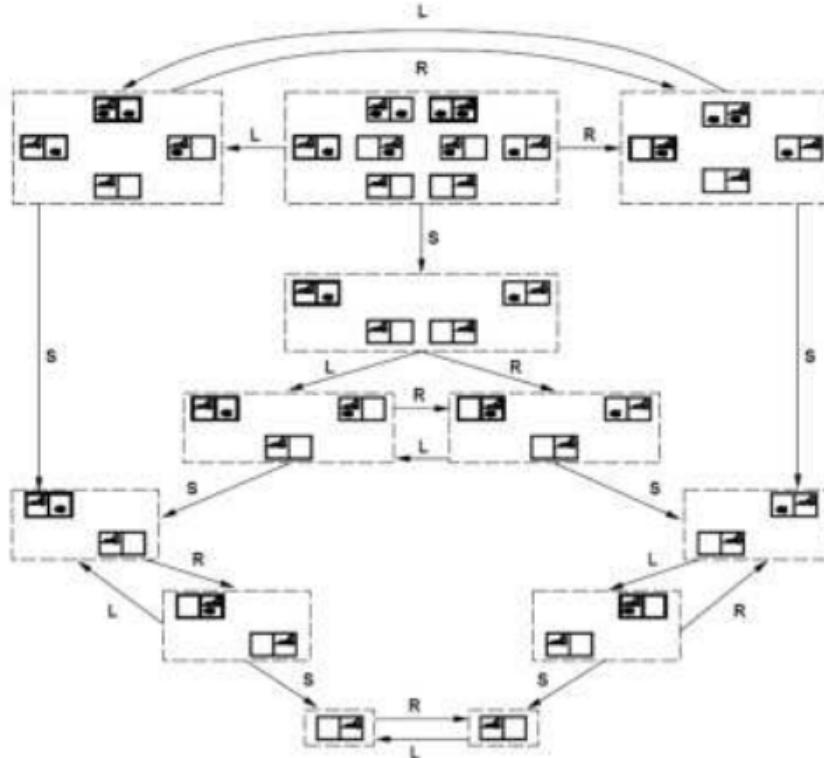
Example: Vacuum World (Multiple-state Version)

- States: subset of the eight states
- Operators: left, right, suck
- Goal test: all states in state set have no dirt
- Path cost: 1 per operator





Example: Vacuum World (Multiple-state Version)





Example: 8-puzzle

- **States:** integer locations of tiles
number of states = 9!
- **Actions:** move blank left, right, up, down
- **Goal test:** = goal state (given)
- **Path cost:** 1 per move

Start state

5	4	
6	1	8
7	3	2

Goal state

1	2	3
8		4
7	6	5





Real-World Problems

Route finding problems:

- Routing in computer networks
- Robot navigation
- Automated travel advisory
- Airline travel planning



Touring problems:

- Traveling Salesperson problem
- “Shortest tour”: visit every city exactly once

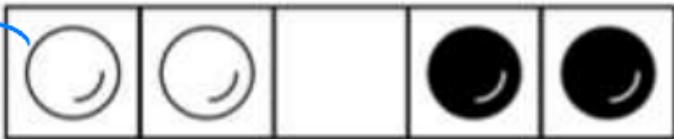




Question to Think About

The stone puzzle is characterised as follows. Two white stones and two black stones are initially positioned on a board, as illustrated below. The board is composed of a single row of five squares, each of which can be either empty or hold one stone. The white stones can only move to the right and the black stones only to the left, either into an immediately adjacent empty square or by jumping over an adjacent stone of opposite colour into an empty square.

State ↗



goal test 01 - 00

The problem is to exchange the respective positions of the white and black stones, moving them one at a time according the rules above. Answer the following to solve this puzzle using a problem-solving approach:

- Give a *well-defined formulation* of the problem in terms of states, operators, goal test predicate, and path cost.

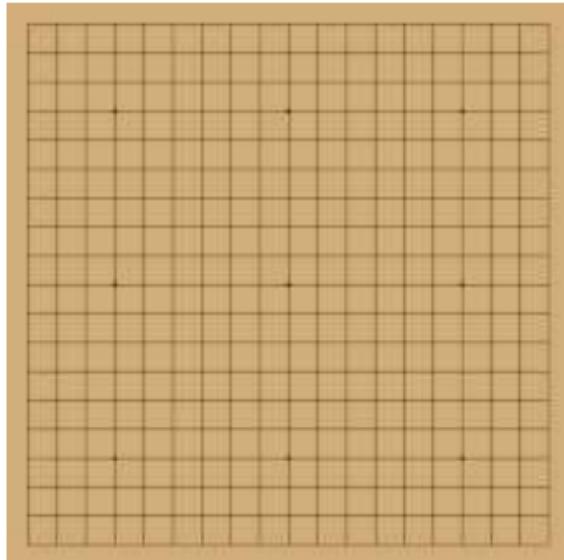




AlphaGo

<https://forms.gle/dmFLBxgx7kBktFPd>

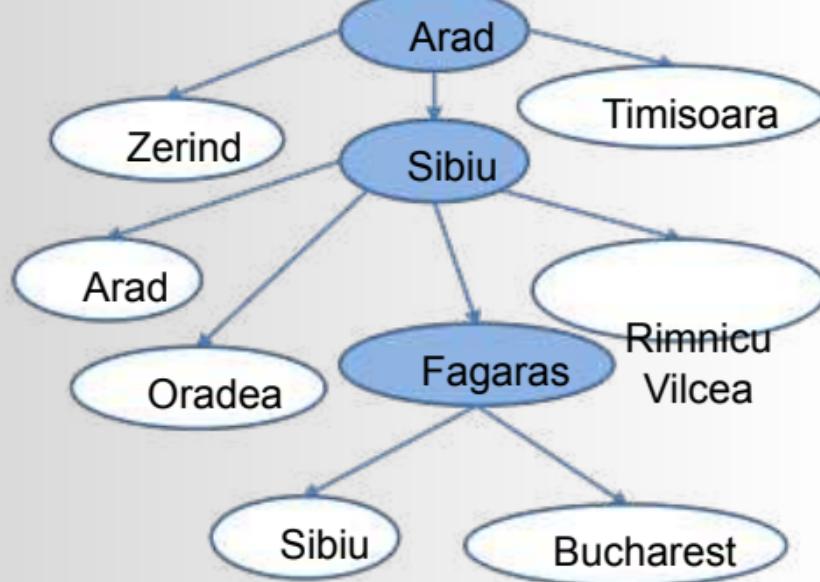
A





Search Algorithms

- Exploration of state space by generating successors of already-explored states
- Frontier: candidate nodes for expansion
- Explored set





Search Strategies

- A **strategy** is defined by picking the order of node expansion.
- Strategies are evaluated along the following dimensions:

Completeness	Does it always find a solution if one exists?
Time Complexity	How long does it take to find a solution: the number of nodes generated
Space Complexity	Maximum number of nodes in memory
Optimality	Does it always find the best (least-cost) solution?





Uninformed vs Informed

Uninformed search strategies

- Use **only** the information available in the problem definition

1. Breadth-first search
2. Uniform-cost search
3. Depth-first search
4. Depth-limited search
5. Iterative deepening search

Informed search strategies

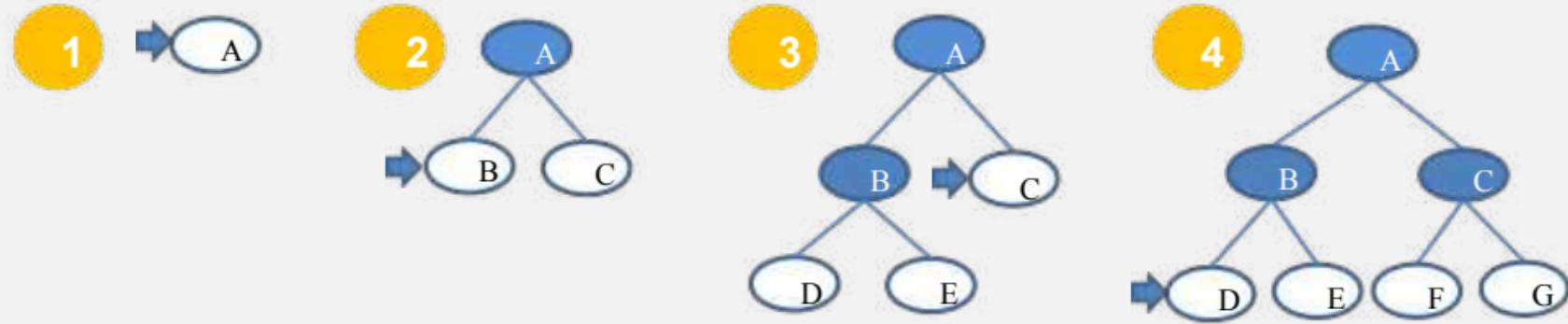
- Use **problem-specific knowledge** to guide the search
- Usually more efficient





Breadth-First Search

Expand **shallowest** unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue



- Denote
- b: maximum branching factor of the search tree
 - d: depth of the least-cost solution
 - Complete: Yes
 - Optimal: Yes when **all** step costs equally





Complexity of BFS

-



Depth	Nodes	Time	Memory
0	1	millisecond	100 bytes
2	0.1	seconds	1 kilobyte
4	11	seconds	11 kilobytes
6	18	minutes	111 megabytes
8	31	hours	11 gigabytes

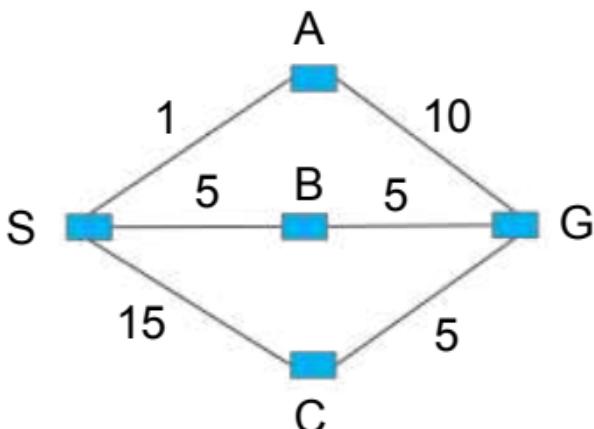




Uniform-Cost Search

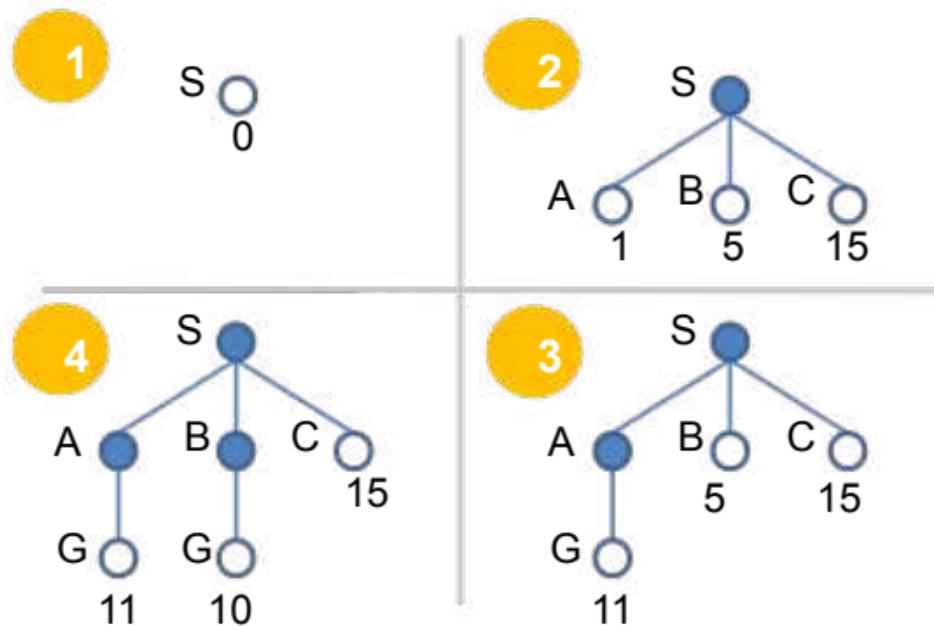
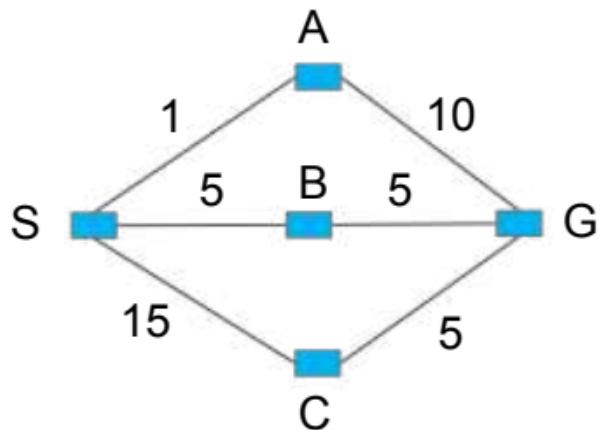
To consider **edge costs**, expand unexpanded node with the **least** path cost g

- Modification of breath-first search
- Instead of First-In-First-Out (FIFO) queue, using a priority queue with path cost $g(n)$ to order the elements
- BFS = UCS with $g(n) = \text{Depth}(n)$





Uniform-Cost Search



Here we do not expand nodes that have been expanded.



Uniform-Cost Search

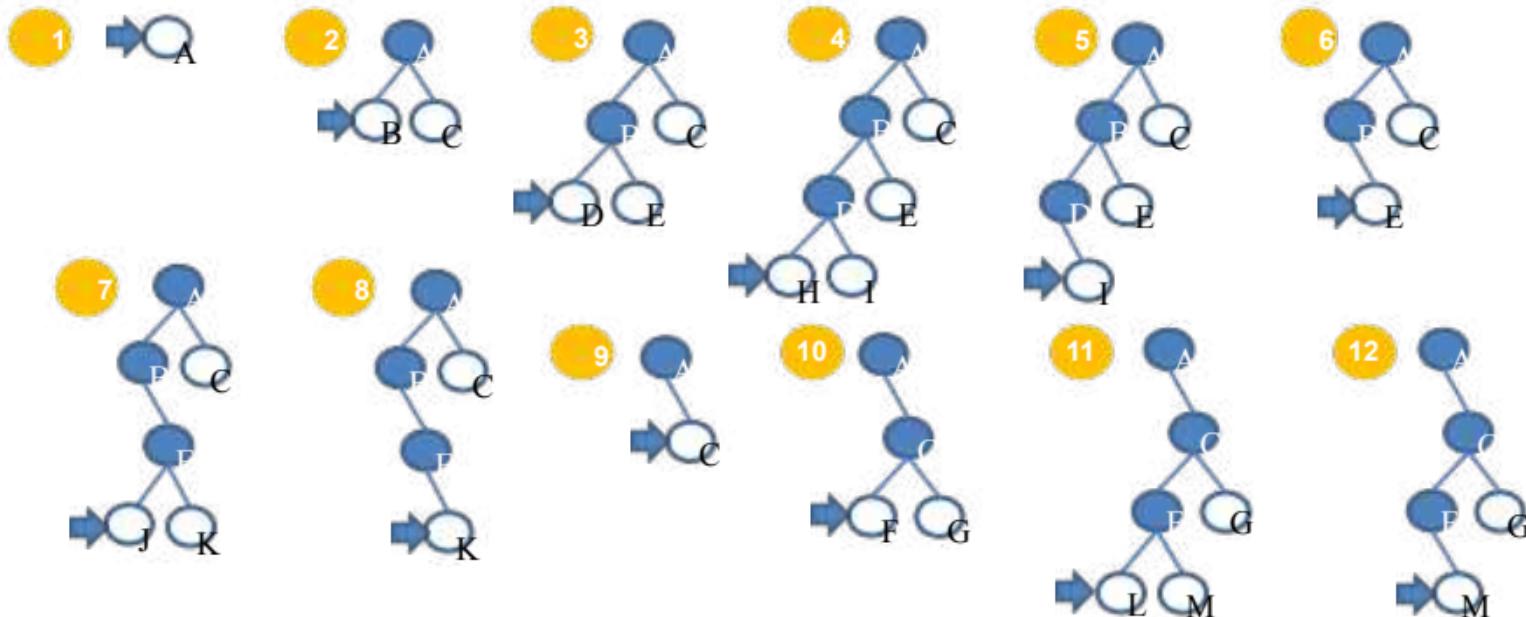
Complete	Yes
Time	# of nodes with path cost $g \leq$ cost of optimal solution (eqv. # of nodes pop out from the priority queue)
Space	# of nodes with path cost $g \leq$ cost of optimal solution
Optimal	Yes





Depth-First Search

Expand deepest unexpanded node which can be implemented by a Last-In-First-Out (LIFO) stack, Backtrack only when no more expansion





Depth-First Search

Denote

- m : maximum depth of the state space

Complete	<ul style="list-style-type: none">• infinite-depth spaces: No• finite-depth spaces with loops: No• with repeated-state checking: Yes• finite-depth spaces without loops: Yes
Time	
Space	
Optimal	No





Depth-Limited Search

To avoid infinite searching, Depth-first search with a [cutoff](#) on the max depth / of a path

Complete

Time

Space

Optimal

No





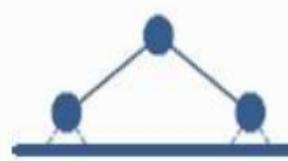
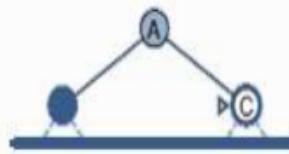
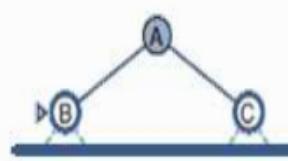
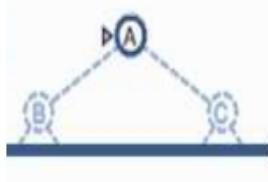
Iterative Deepening Search

Iteratively estimate the max depth / of DLS one-by-one

Limit = 0



Limit = 1

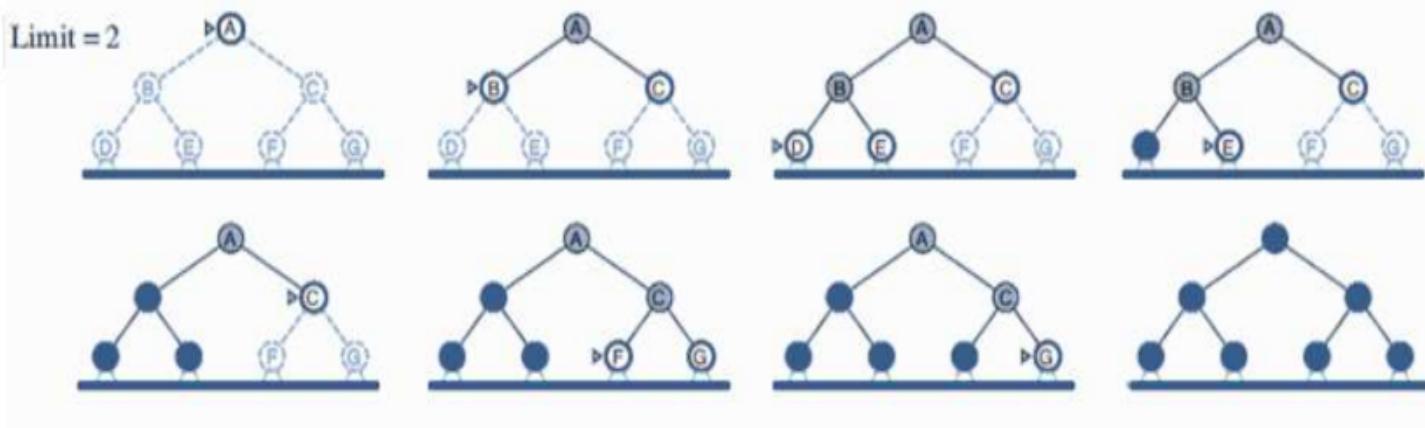




Iterative Deepening Search

Iteratively estimate the max depth / of DLS one-by-one

Limit = 2

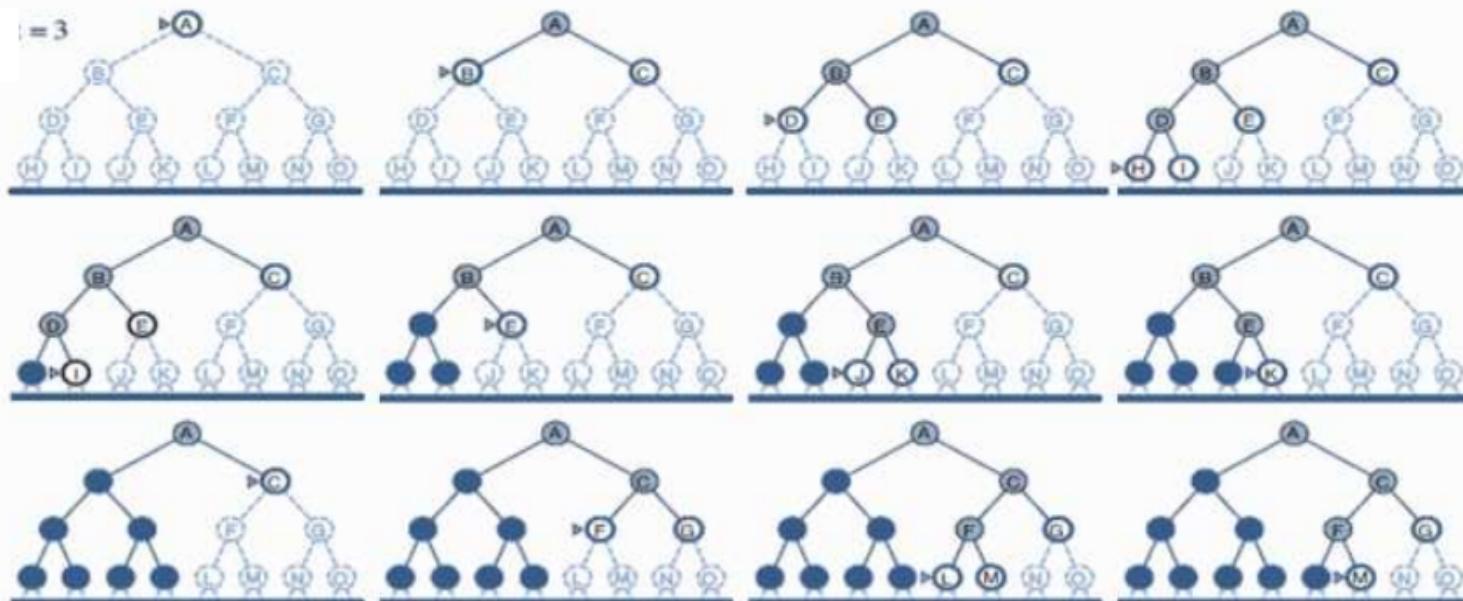




Iterative Deepening Search

Iteratively estimate the max depth / of DLS one-by-one

Limit = 3





Iterative Deepening Search...

Complete

Yes

Time

Space

Optimal

Yes





Summary (we make assumptions for optimality)

Criterion	Breadth-first	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time						
Space						
Optimal	Yes	Yes	No	No	Yes	Yes
Complete	Yes	Yes	No		Yes	Yes





General Search

Uninformed search strategies

- Systematic generation of new states (Goal Test)
- Inefficient (exponential space and time complexity)

Informed search strategies

- Use problem-specific knowledge
- To decide the order of node expansion



Best First Search: expand the most desirable unexpanded node

Use an evaluation function to estimate the “desirability” of each node





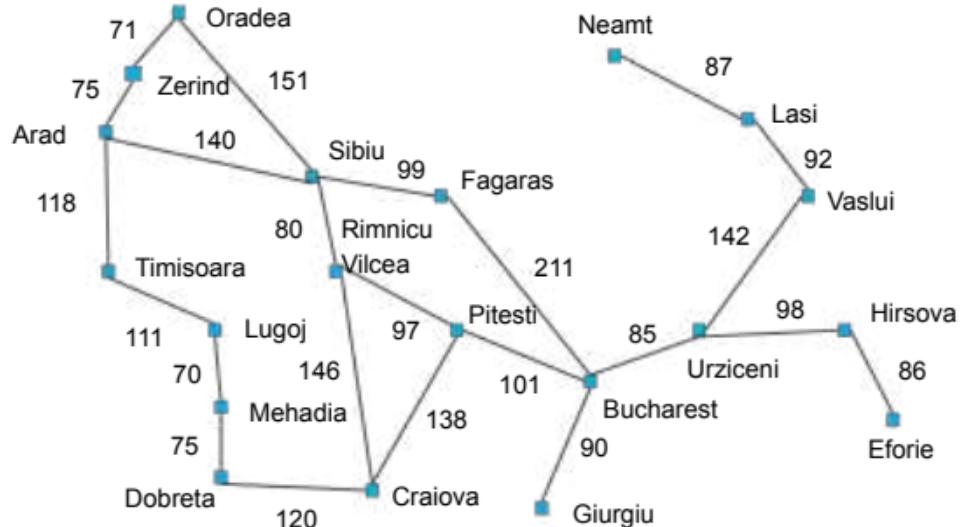
Evaluation function

- Path-cost function $g(n)$
 - Cost from initial state to current node n
 - Information on the cost toward goal
- “Heuristic” function $h(n)$





Example: Route-finding from Arad to Bucharest



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- Useful but potentially fallible ([heuristic](#))
- Heuristic functions are problem-specific





Example

a)

The initial state



366

Straight-line distance to Bucharest

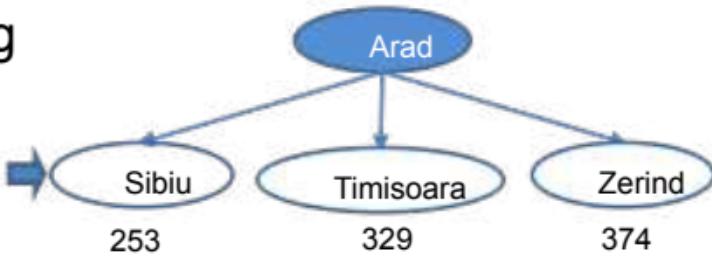
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





Example

a) After
expanding
Arad



Straight-line distance to Bucharest

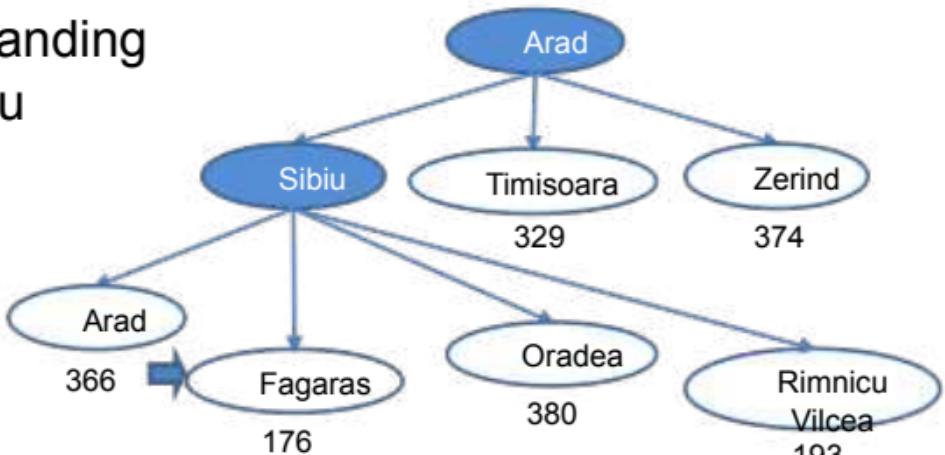
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





Example

a) After
expanding
Sibiu



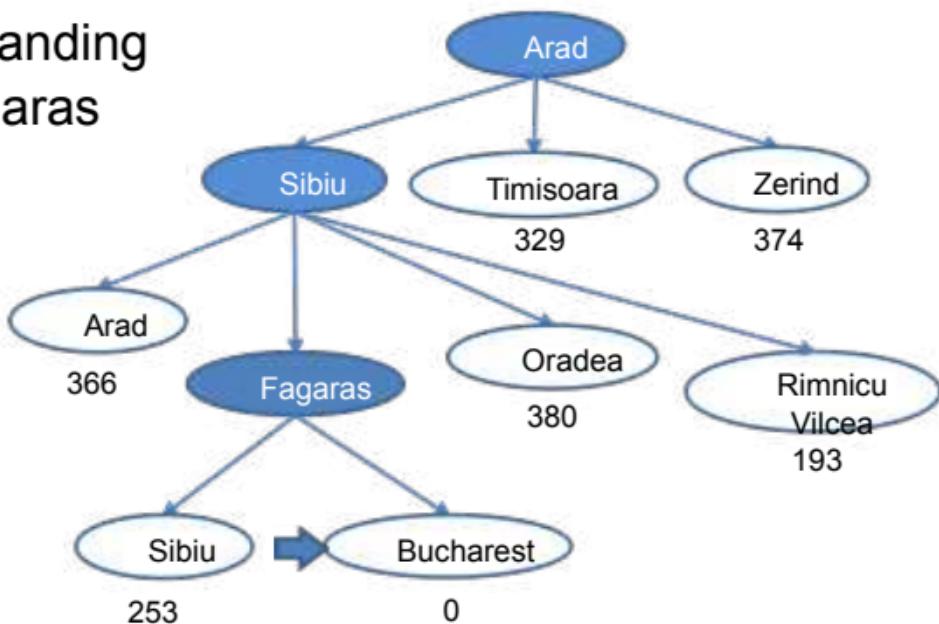
Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Example

a) After
expanding
Fagaras



Straight-line distance to Bucharest

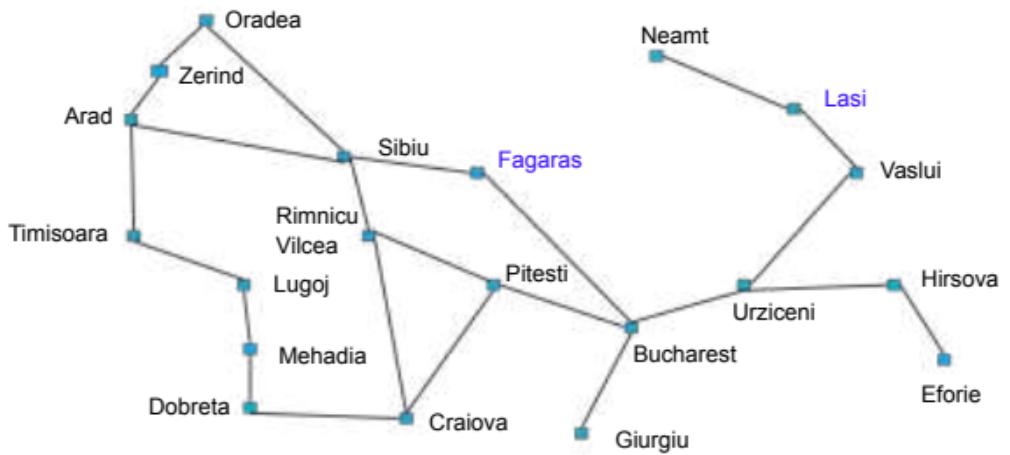
Arad	366
Bucharest	0
Craiova	166
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





Complete?

Question: Is this approach complete?



Answer: No



Greedy Search...

- m: maximum depth of the search space

Complete	No
Time	
Space	
Optimal	No





A * Search

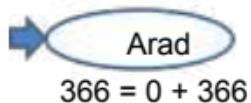
- Uniform-cost search
 - $g(n)$: cost to reach n (**Past Experience**)
 - optimal and complete, but can be very inefficient
- Greedy search
 - $h(n)$: cost from n to goal (**Future Prediction**)
 - neither optimal nor complete, but cuts search space considerably



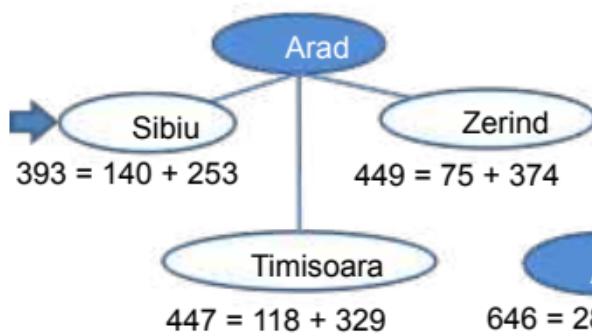


Example: Route-finding from Arad to Bucharest

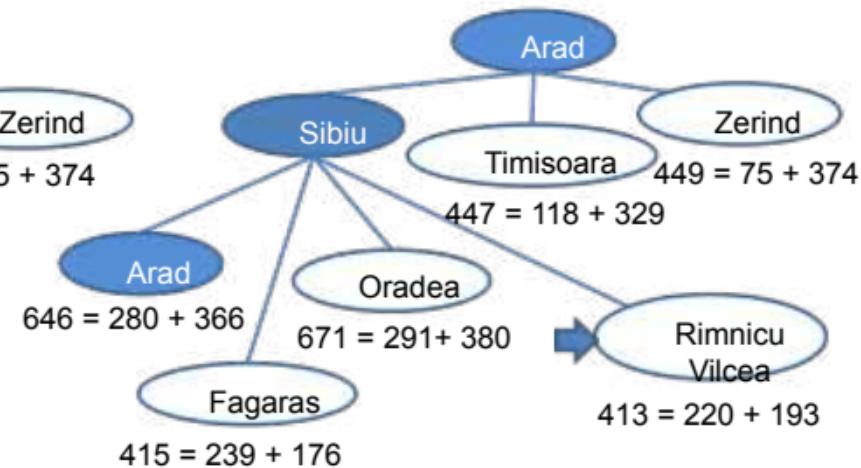
(a) The initial state



(b) After expanding Arad



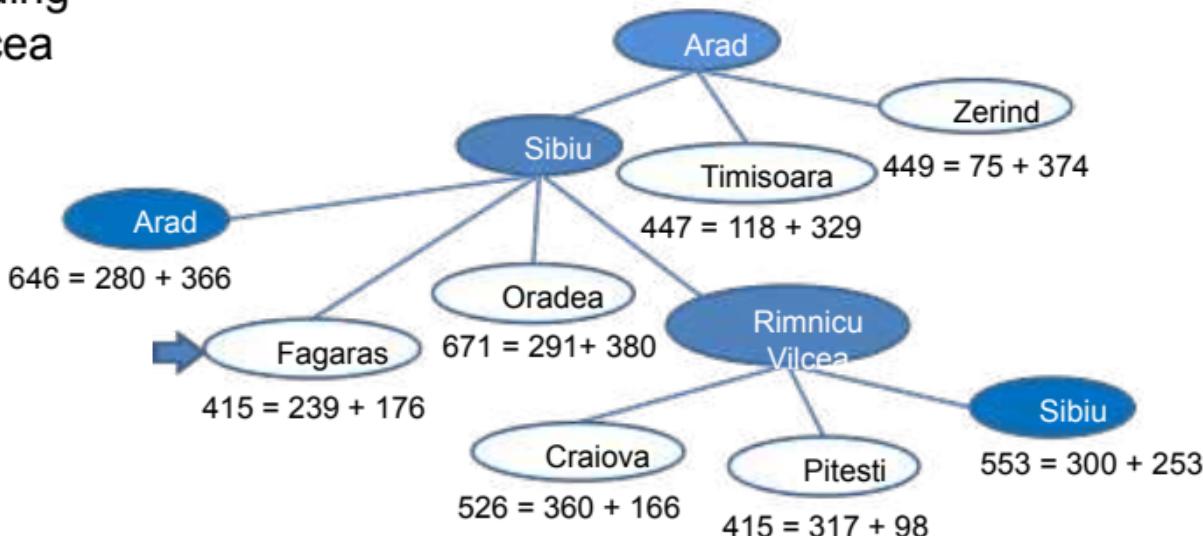
(c) After expanding Sibiu





Example: Route-finding from Arad to Bucharest

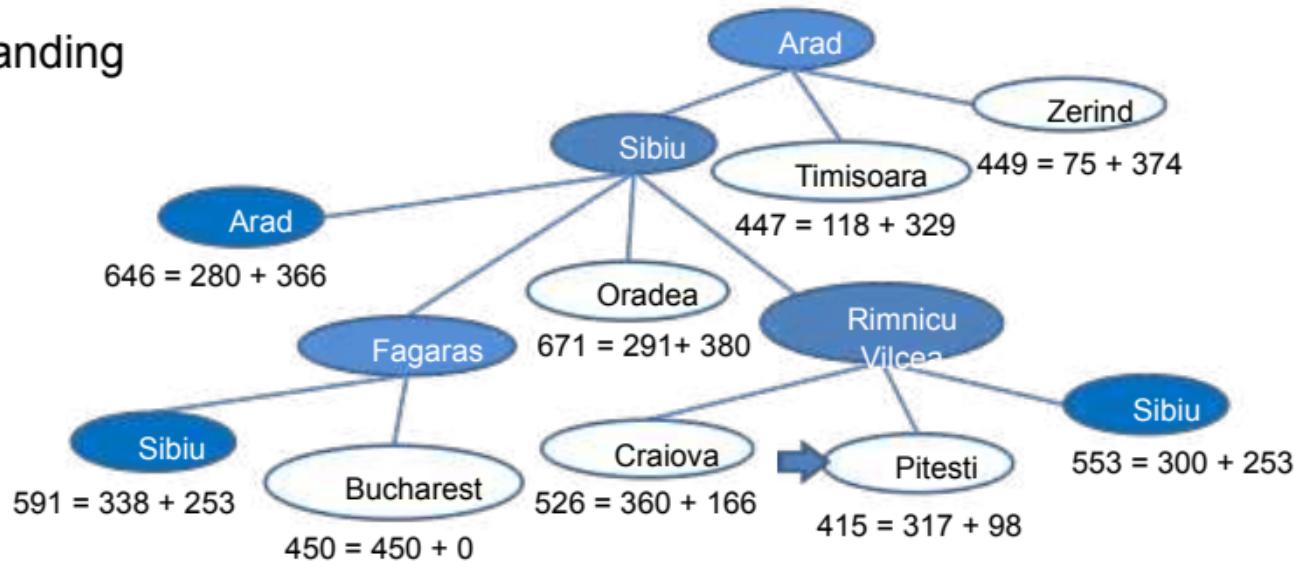
(d) After expanding
Rimnicu Vilcea





Example: Route-finding from Arad to Bucharest

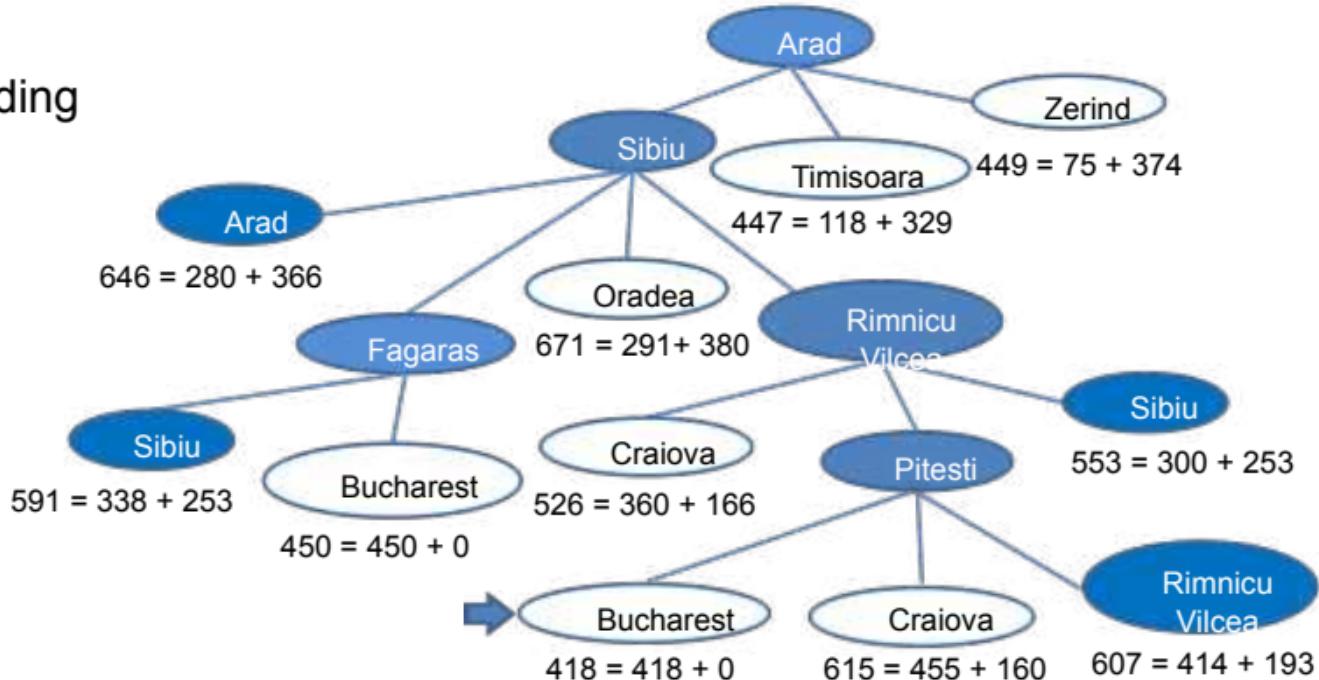
(e) After expanding
Fagaras





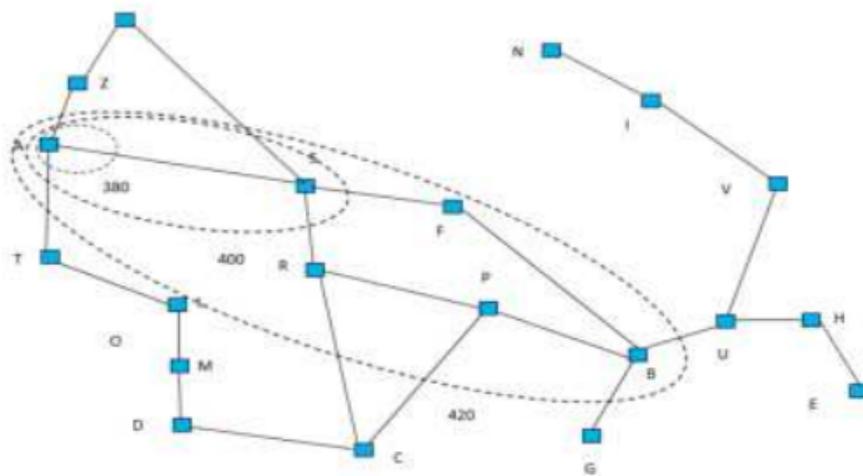
Example: Route-finding from Arad to Bucharest

(f) After expanding
Pitesti





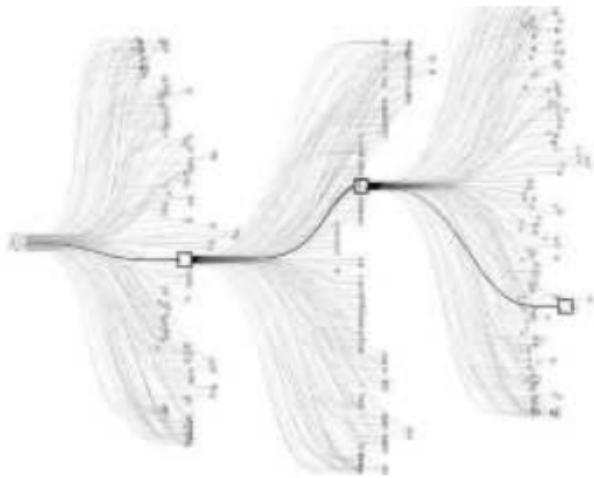
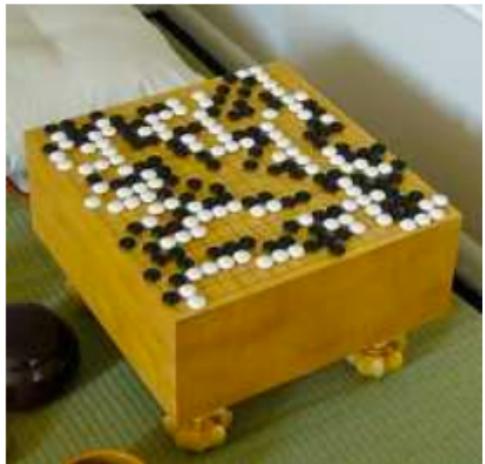
Complexity of A*



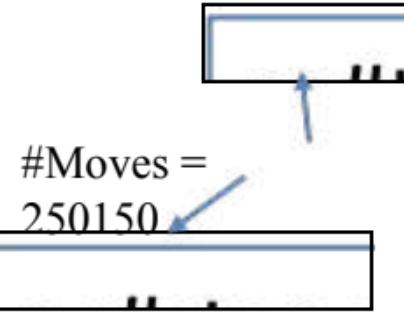
With a good heuristic, significant savings are still possible compared to uninformed search methods



Another Example: Alpha Go



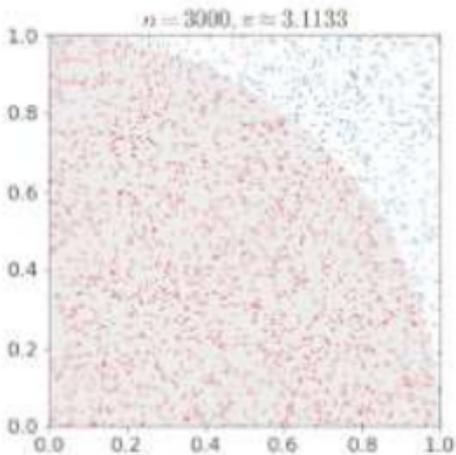
#Moves =
250150





Another Example: Monte Carlo Search Tree (MCST)

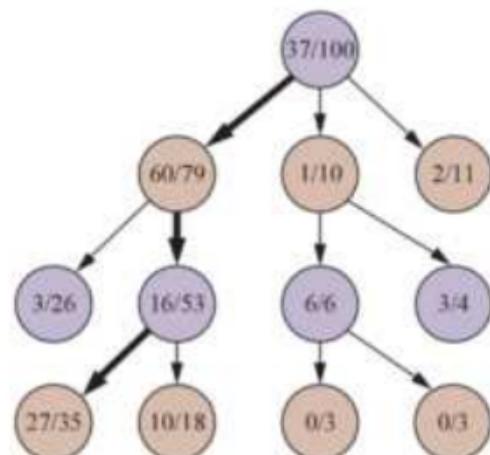
Monte Carlo
= random



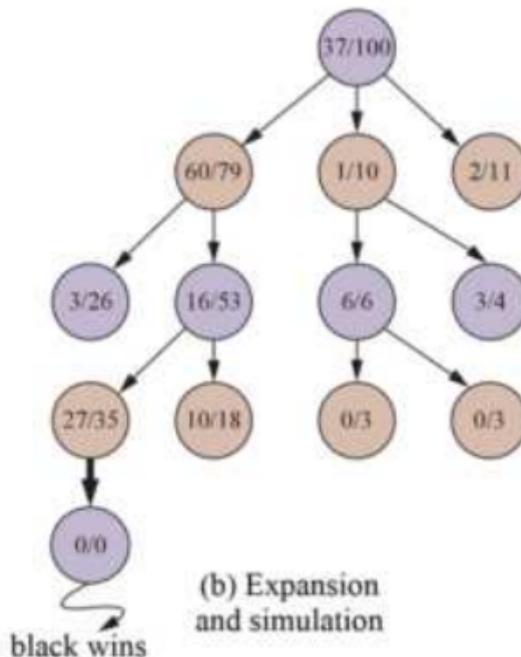


MCST Key Steps

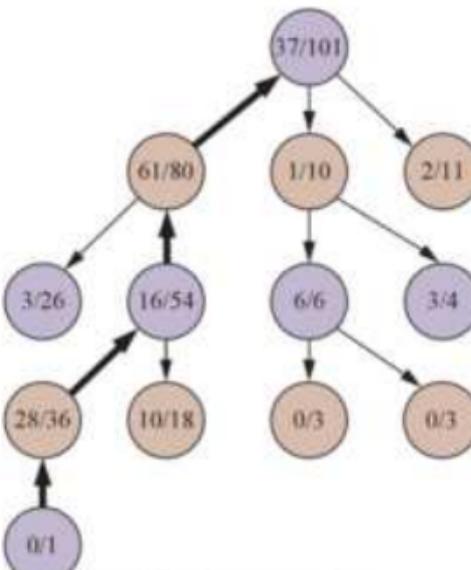
#Wins/#Games



(a) Selection



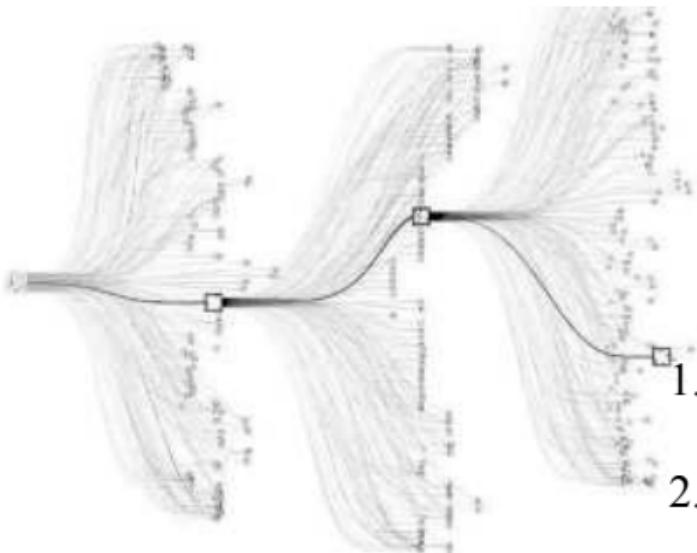
(b) Expansion
and simulation



(c) Backpropagation



MCST for Go is very expensive & slow



Why? The deeper you move, the wider you are
#number of simulation is exploded

Solution:

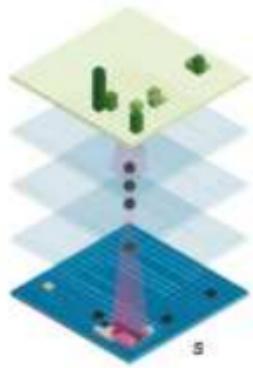
1. For selection/expansion, no longer stats (xx/xx) but $f = g+h$
2. For simulation, no longer real self-play but $f = g+h$



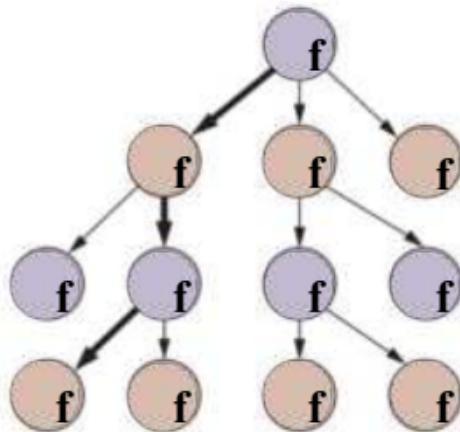
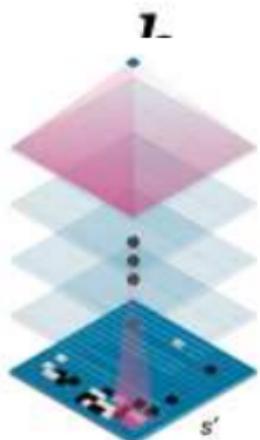


MCST: Update $f = g() + h()$

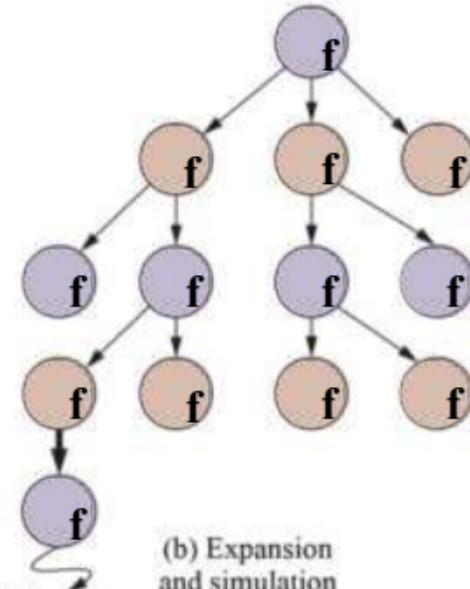
Policy Network



Value Network



(a) Selection



(b) Expansion
and simulation

No real play,
just an estimation



CZ3005 Artificial Intelligence

Constraint satisfaction and adversarial search

Asst/P Hanwang Zhang

<https://personal.ntu.edu.sg/hanwangzhang/>

Email: hanwangzhang@ntu.edu.sg



Lesson Outline

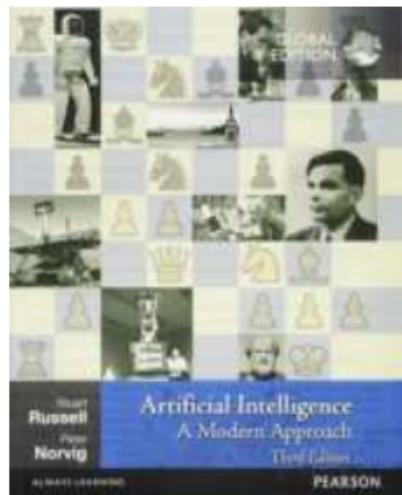
Use chapter



- Constraint Satisfaction
- Adversarial search (Game Playing)

A* is complete if it has
goal tracking n optimal if

heuristic
is accurate
to its actual
cost

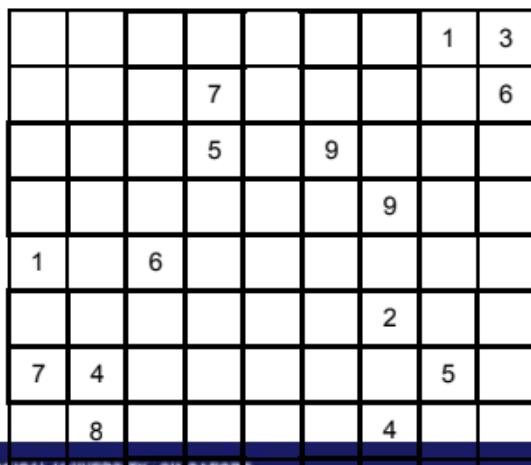




Constraint Satisfaction Problem (CSP)

Goal: Discover some state that satisfies a given set of constraints

Example: Sudoku



Example: Minesweeper





Examples: Real-world CSPs

- Assignment problems
 - e.g. who teaches what class
- Timetabling problems
 - e.g. which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor-planning





CSP

- State

Example: 8-queens

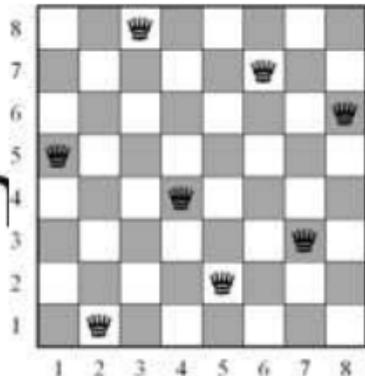
- Variables: Locations of each of the eight queens
- Values: squares on the board

Goal test

- a set of constraints specifying allowable combinations of values for subsets of variables

Example: 8-queens

- Goal test: No two queens in the same row, column or diagonal





Example: Cryptarithmetic Puzzle

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

- Variables: D, E, M, N, O, R, S, Y
- Value Domain: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
 - $Y = D + E$ or $Y = D + E - 10$, etc.
 - $D \neq E$, $D \neq M$, $D \neq N$, etc.
 - $M \neq 0$, $S \neq 0$ (unary constraints: concern the value of a single variable)

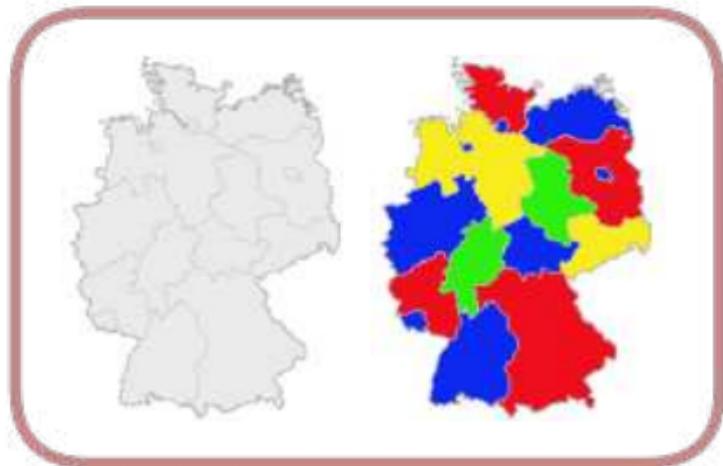
M cannot be > 2
 $m \neq 0$
then $m=1$
M+S need carry
S must be 9





Example: Map Colouring

Colour a map so that no adjacent parts have the same colour



- Variables: Countries C_i
- Domains: {Red, Blue, Green}
- Constraints: $C_1 \neq C_2, C_1 \neq C_5$, etc.
- **binary** constraints





Some Definitions

- A state of the problem is defined by an **assignment** of values to some or all of the variables.
- An assignment that does not violate any constraints is called a **consistent** or **legal** assignment.
- A **solution** to a CSP is an assignment with every variable given a value (**complete**) and the assignment satisfies all the constraints. *(Solution not be optimal)*





Applying Standard Search

- **States**: defined by the values assigned so far
- **Initial state**: all variables unassigned
- **Actions**: assign a value to an unassigned variable
- **Goal test**: all variables assigned, no constraints violated





Applying Standard Search

Question: How to represent constraints?

Answer: Explicitly (e.g., $D \neq E$)

, explicit
can be
implicit ?

, implicit cannot be
explicit ?

Example

- Row the 1st queen occupies: V_1
(similarly, for V_2)

Implicitly: use a function to test for constraint satisfaction

'implicit write
Explicit into'
heuristic code ?



in implicit, Y-give opposite of
a reward function if constraint is
Violated
in explicit, Violation not allowed



Explicit or Implicit?

Implicit



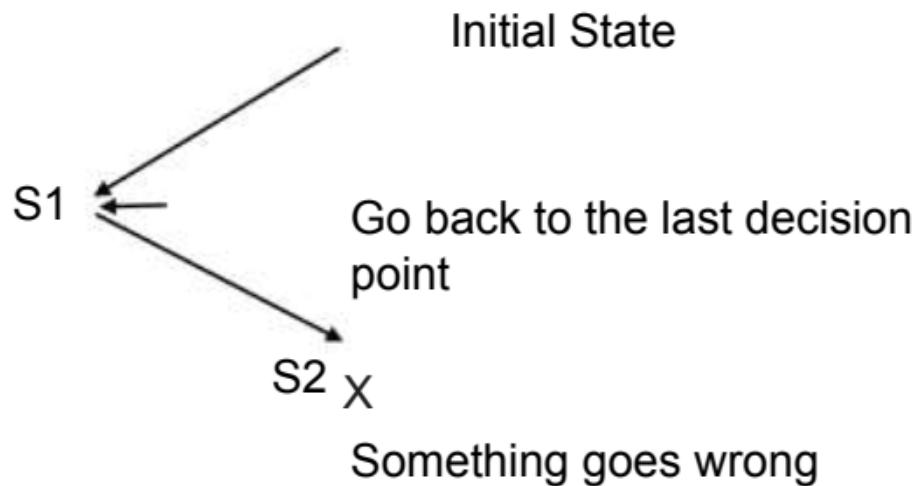
Explicit





Backtracking Search

Backtracking search: Do not waste time searching when constraints have already been violated

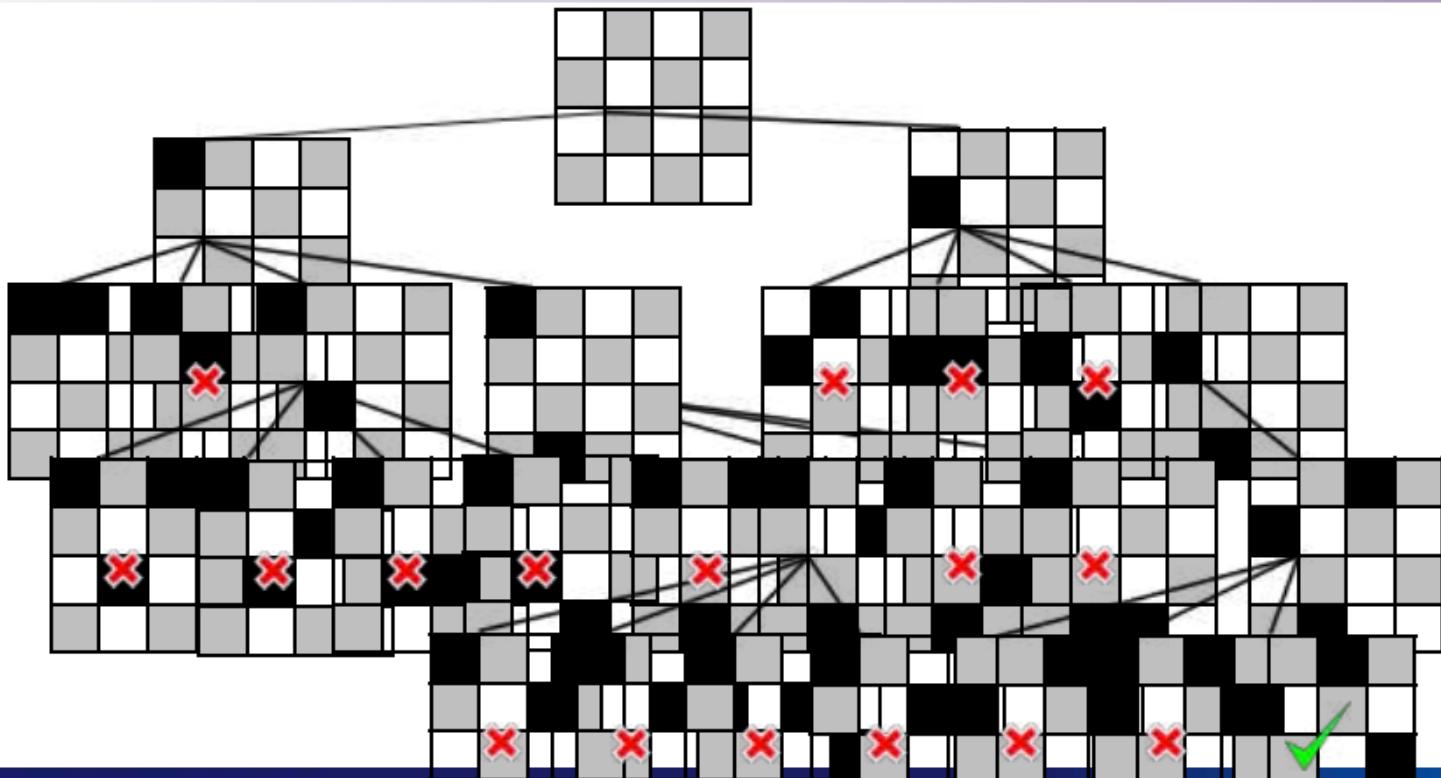


- Before generating successors, check for constraint violations
- If yes, backtrack to try something else





Example (4-Queens)



Heuristics for CSPs

P informed
Search



Plain backtracking is an uninformed algorithm!!

More intelligent search that takes into consideration

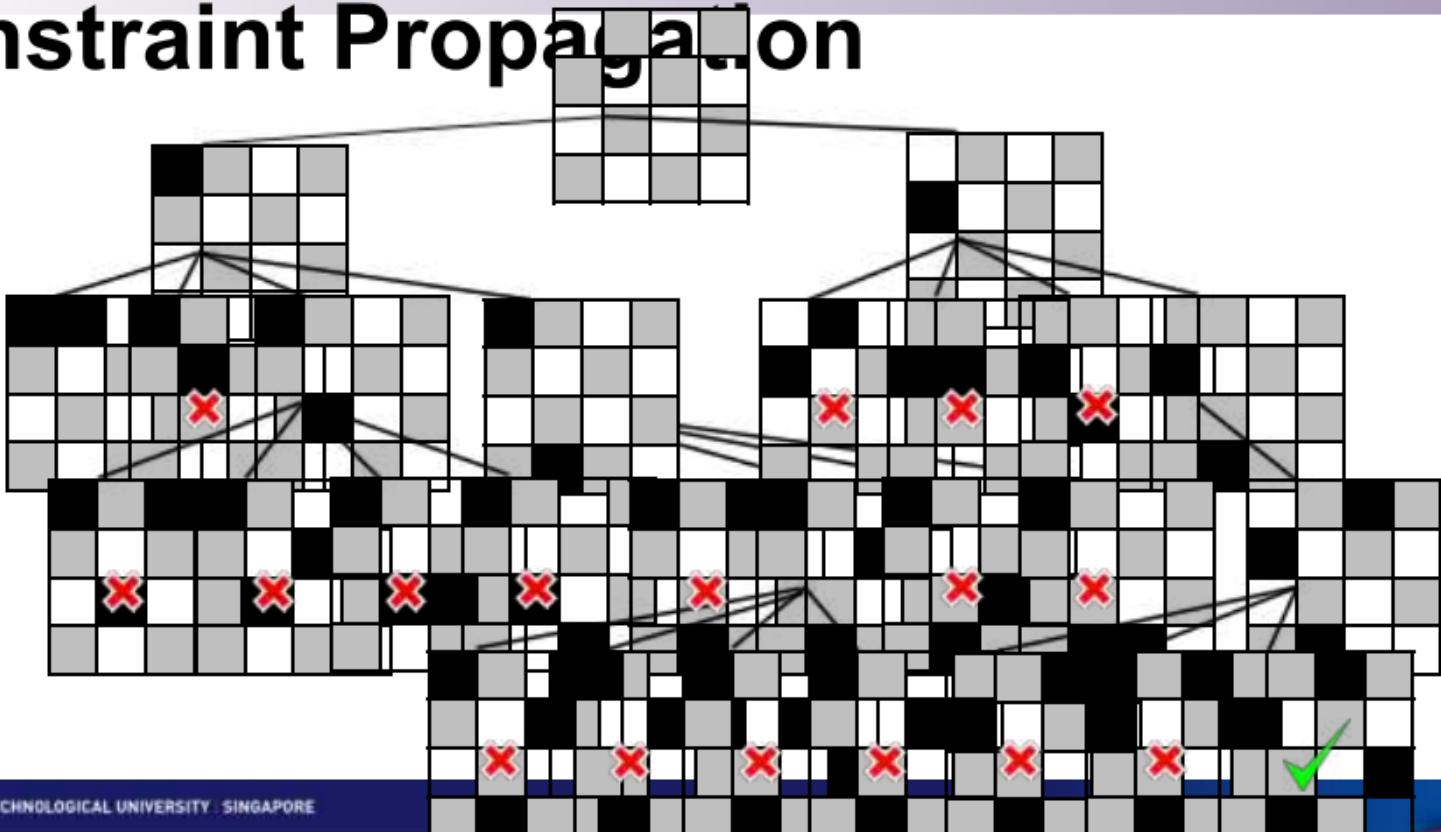
- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
 - forward checking and **constraint propagation**

Constraint propagation: propagating the implications of a constraint on one variable onto other variables

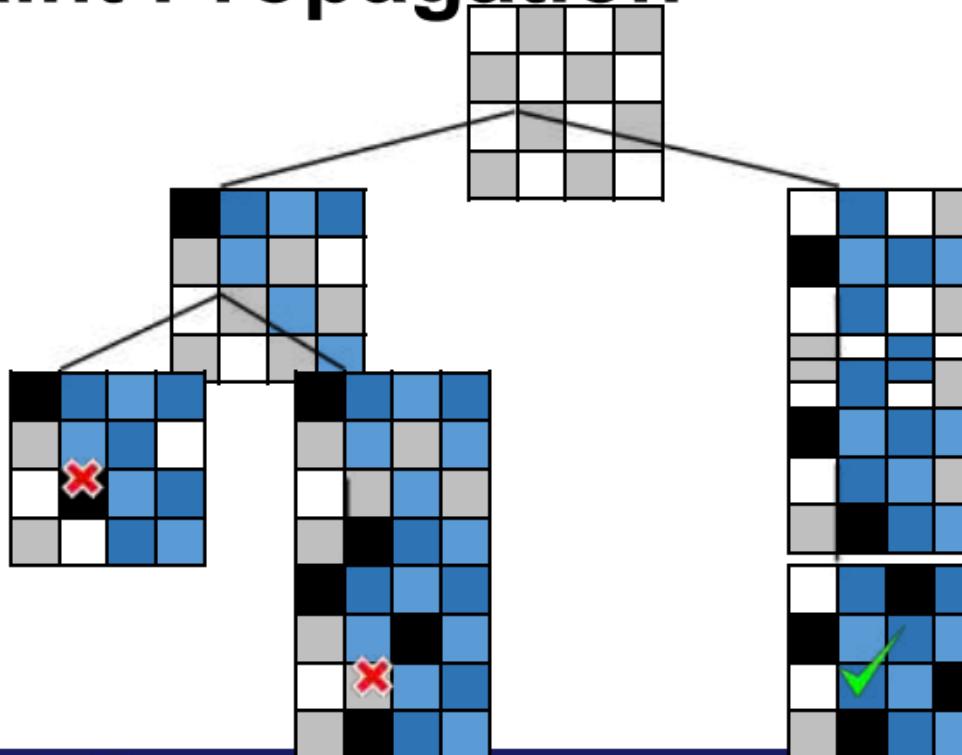




Example (4-Queens) without Constraint Propagation

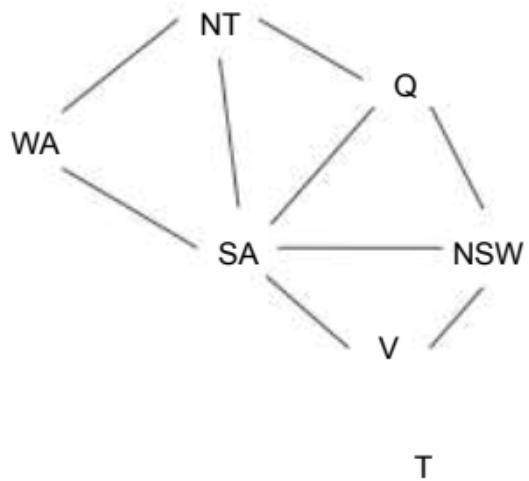


Search Tree of 4-Queens with Constraint Propagation



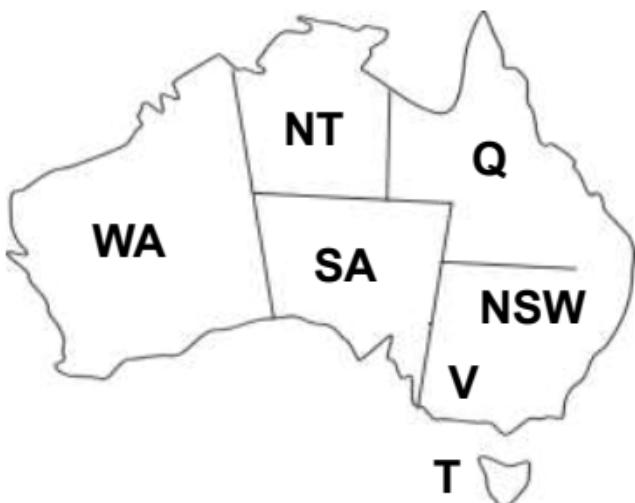


Example (Map Colouring)





Example (Map Colouring)...



WA

N

Q

NSW

V

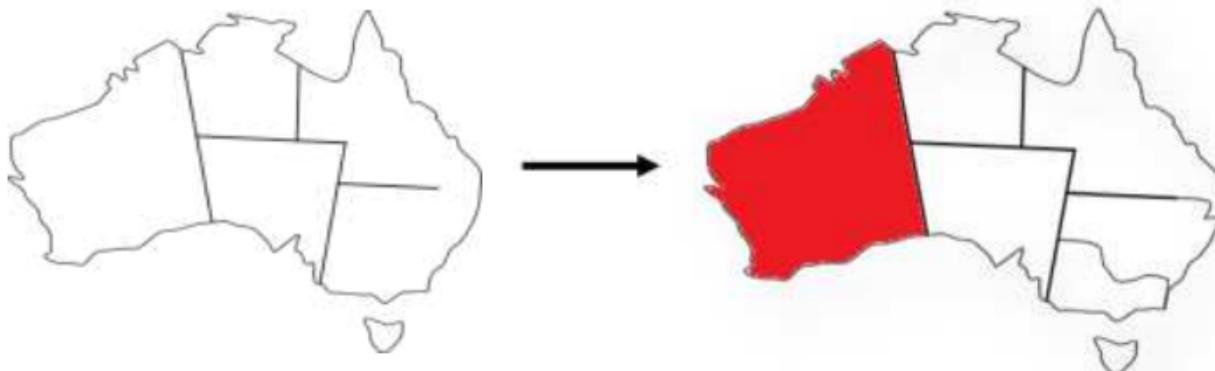
SA

T





Example (Map Colouring)...



WA	N	Q	NSW	V	SA	T
■ Red ■ Green ■ Blue						
■ Red	■ Green ■ Blue	■ Red ■ Green ■ Blue	■ Red ■ Green ■ Blue	■ Red ■ Green ■ Blue	■ Green ■ Blue	■ Red ■ Green ■ Blue





Example (Map Colouring)...



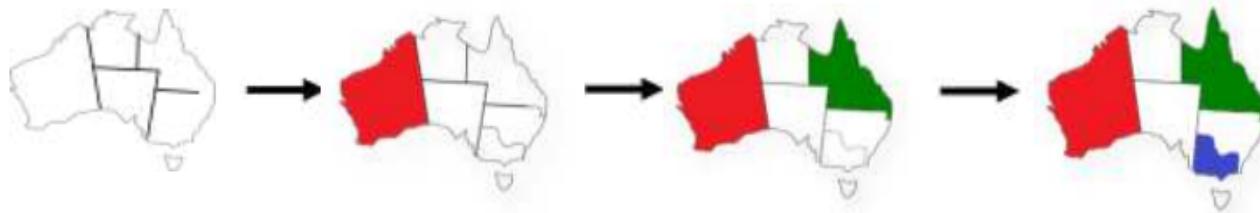
WA N Q NSW V SA T

Red	Green	Blue	Red	Green	Blue	Red	Green	Blue
Red	Green	Blue	Red	Green	Blue	Red	Green	Blue
Red	Green	Blue	Red	Green	Blue	Red	Green	Blue





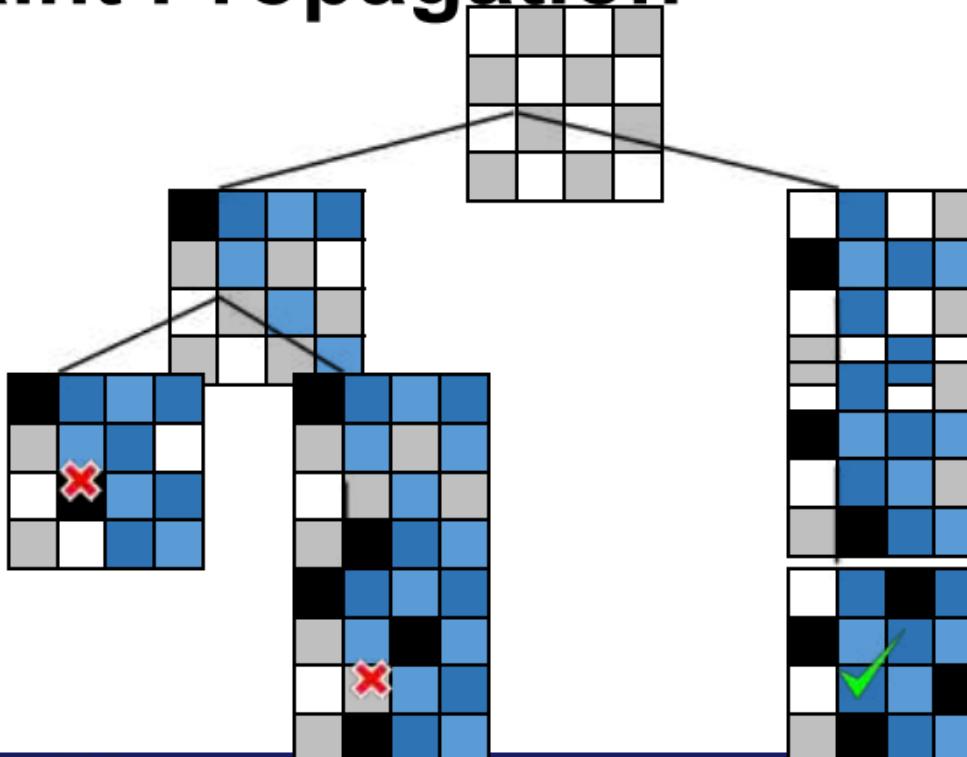
Example (Map Colouring)...



WA	N	Q	NSW	V	SA	T
Red	Green	Blue	Red	Green	Blue	Red
Red		Green	Blue	Red	Green	Blue
Red		Blue	Green	Red	Green	Blue
Red		Blue		Blue		Red



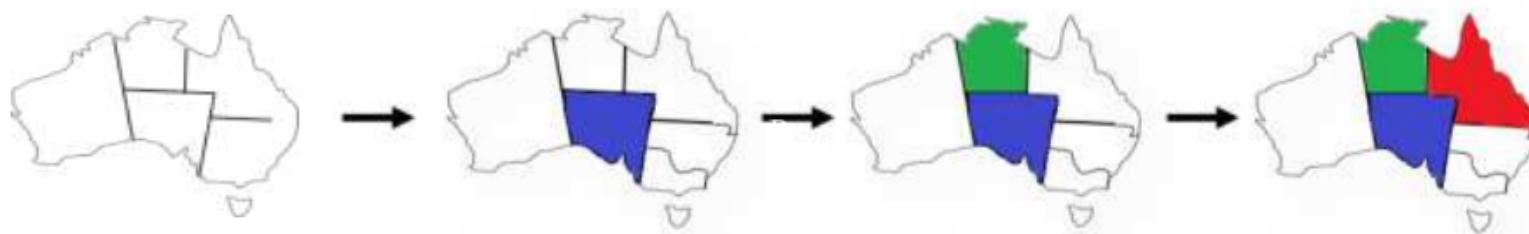
Search Tree of 4-Queens with Constraint Propagation





Most Constraining Variable

Example: map colouring

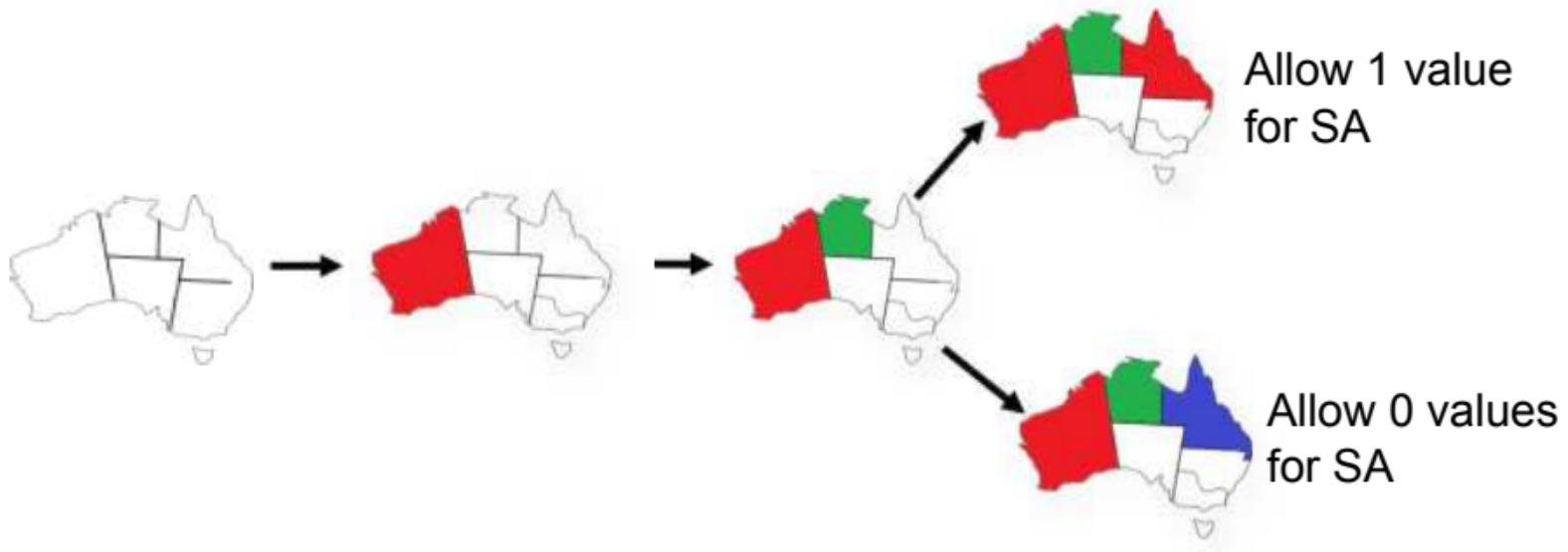


To reduce the branching factor on future choices by selecting the variable that is involved in the **largest number of constraints** on unassigned variables.





Least Constraining Value



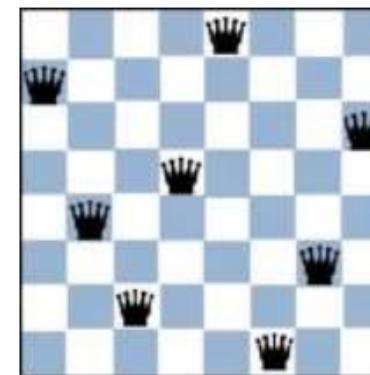
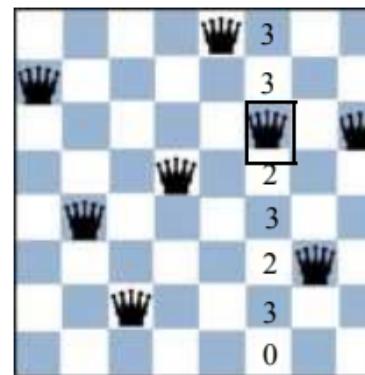
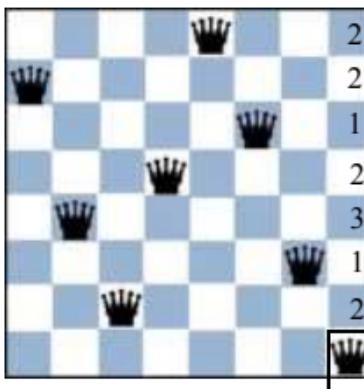
Choose the value that leaves maximum flexibility for subsequent variable assignments

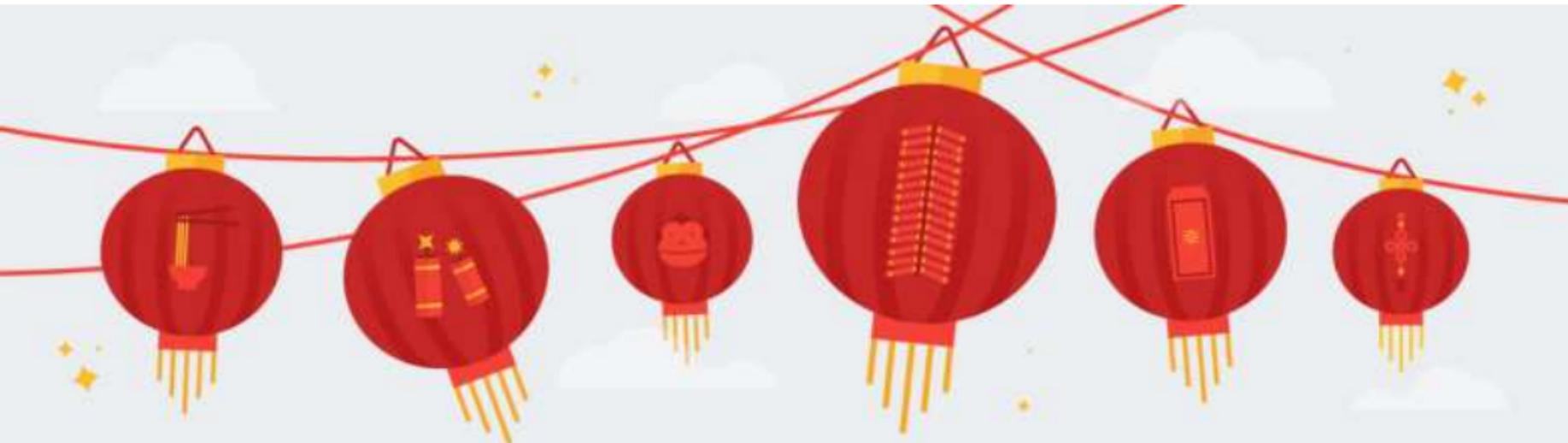
Not in Exam

Min-Conflicts Heuristic (8-queens)



- A local heuristic search method for solving CSPs
- Given an initial assignment, selects a variable in the scope of a violated constraint and assigns it to the value that minimises the number of violated constraints





Adversarial Search





Games as Search Problems

Abstraction

- Ideal representation of real world problems
 - e.g. board games, chess, go, etc. as an abstraction of war games
 - Perfect information, i.e. fully observable
- Accurate formulation: state space representation

Uncertainty

- Account for the existence of **hostile** agents (players)
 - Other agents acting so as to diminish the agent's well-being
 - Uncertainty (about other agents' actions):
 - not due to the effect of non-deterministic actions
 - not due to randomness
- Contingency problem



Games as Search Problems...

- Complexity
 - Games are abstract but not simple
 - e.g. chess: average branching factor → complexity = 35^{50} (only 10^{40} for legal moves)

Games are usually time limited





Types of Games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information		Bridge, Poker, Scrabble, Nuclear war

- In this course, we only focus on Perfect information
 - each player has complete information about his opponent's position and about the choices available to him





Game as a Search Problem

- Initial state: initial board configuration and indication of who makes the first move
- Operators: legal moves
- Terminal test: determines when the game is over
 - states where the game has ended: **terminal states**
- Utility function (payoff function): returns a numeric score to **quantify** the outcome of a game

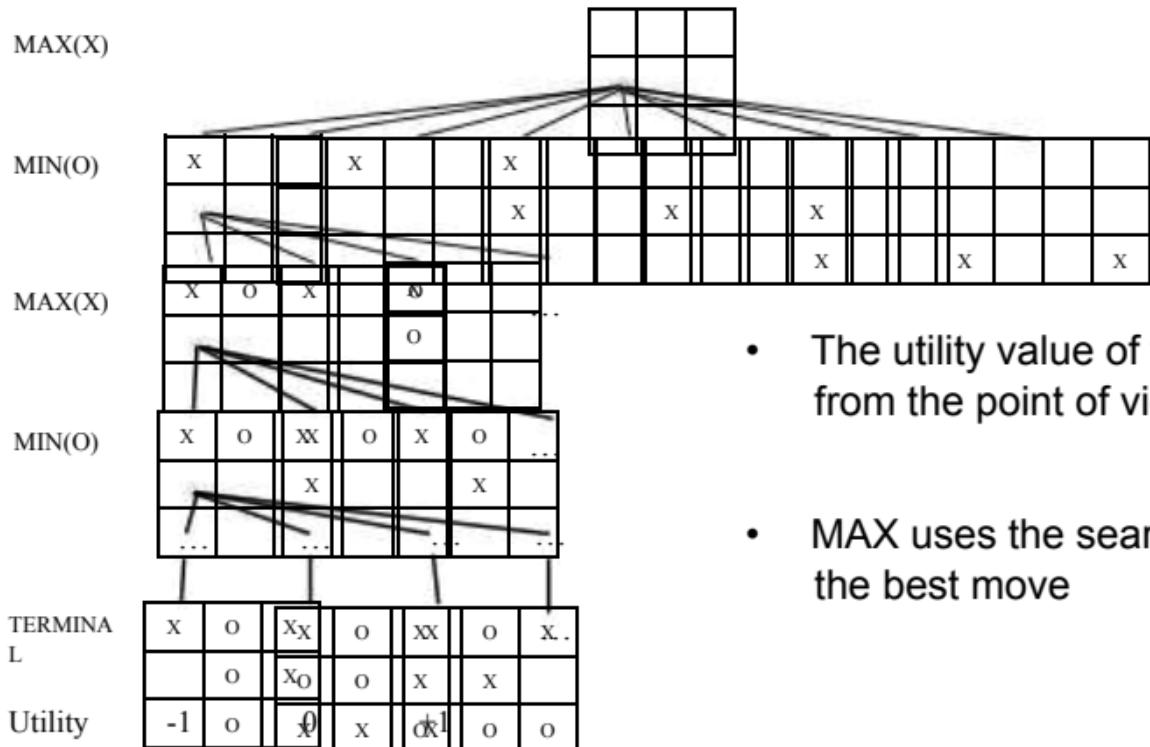
Example: Chess

Win (+1), loss(-1) or draw (0)





Game Tree for Tic-Tac-Toe

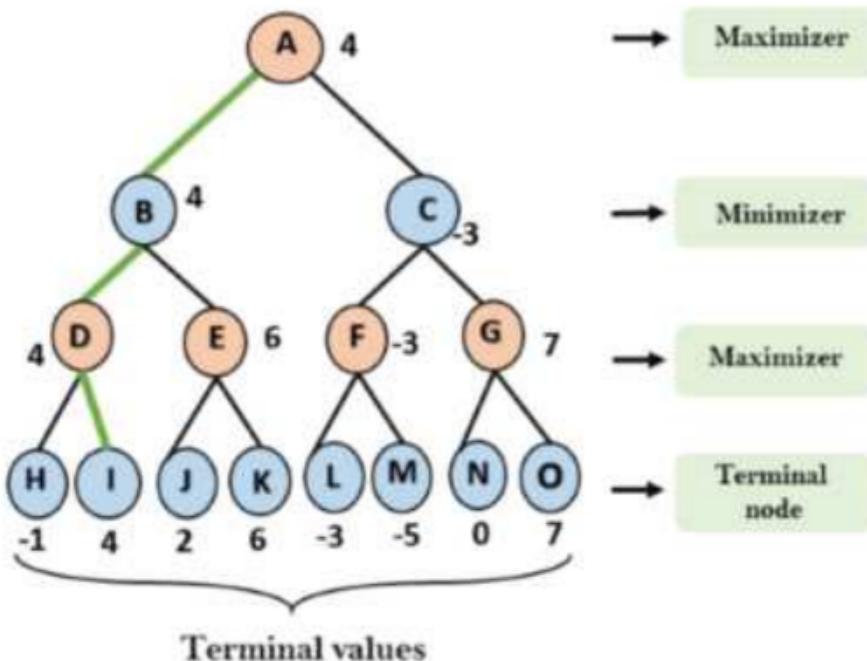


- The utility value of the terminal state is from the point of view of MAX
- MAX uses the search tree to determine the best move



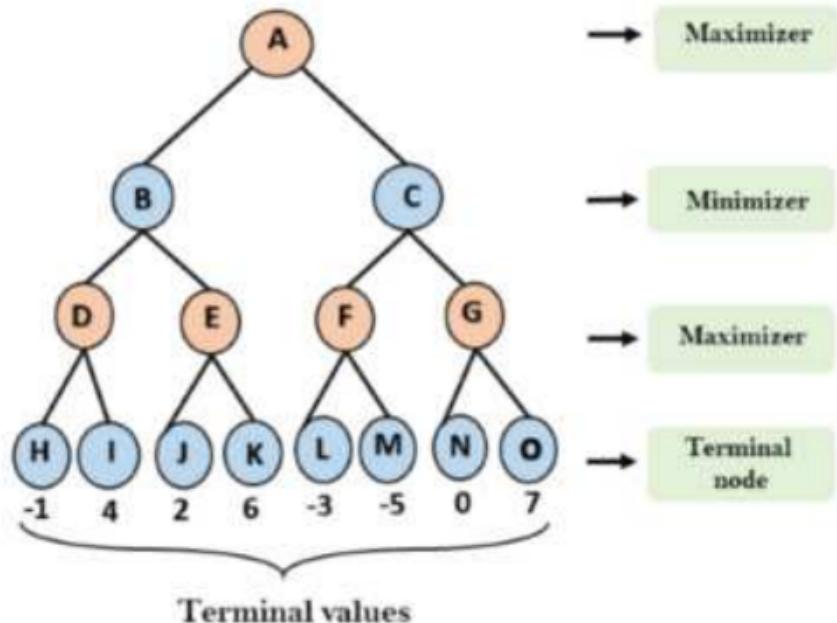


A Toy Min-Max Example





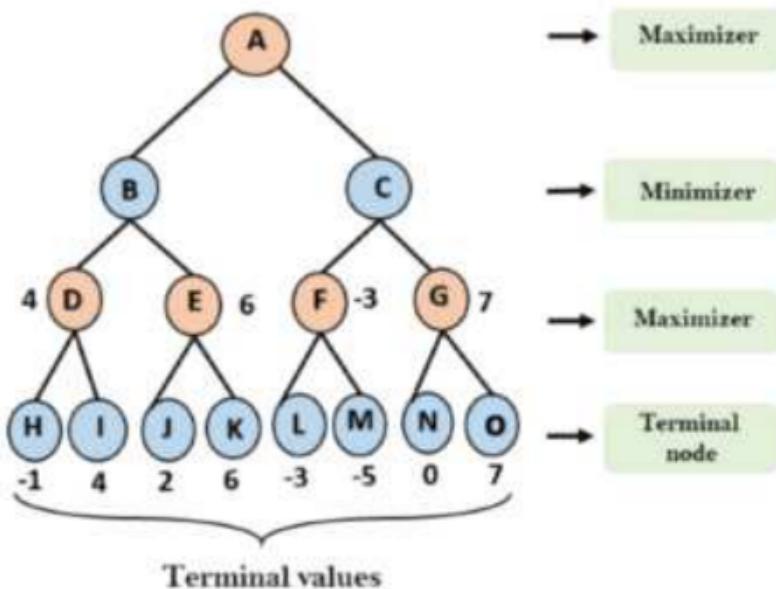
A Toy Min-Max Example



Step 1:
Expand the whole game tree.
Initialize each node to $-\infty$ for max
 $+\infty$ for min



A Toy Min-Max Example

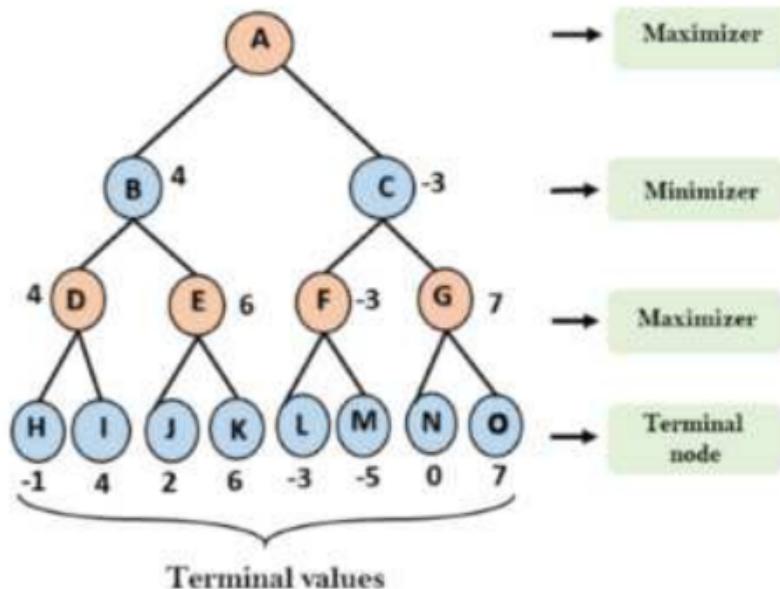


Step 2:
Update the value from btm to top

e.g., for node D:
We want to win max the value (-1, 4)



A Toy Min-Max Example



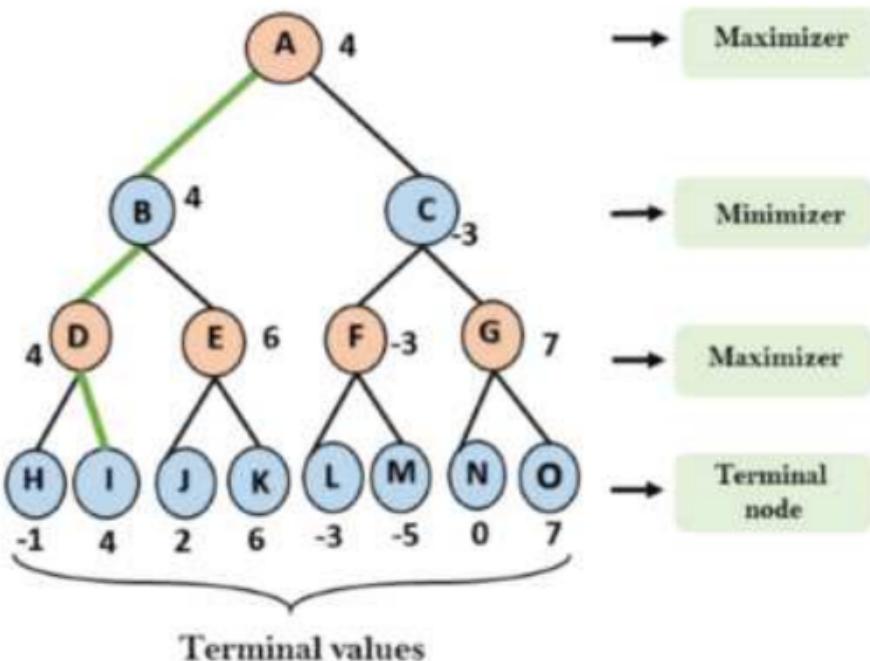
Step 2:
Update the value from btm to top

e.g., for node D:
I want to win max the value (-1, 4)

e.g., for node B:
You want me to lose min the value (4, 6)



A Toy Min-Max Example



Minimax Search Strategy



Search strategy

- Find a sequence of moves that leads to a terminal state (goal)

Minimax search strategy

- Maximise one's own utility and minimise the opponent's
 - Assumption is that the opponent does the same





Minimax Search Strategy

3-step process

1. Generate the entire game tree down to terminal states

2. Calculate utility
 - a) Assess the utility of each terminal state
 - b) Determine the best utility of the parents of the terminal state
 - c) Repeat the process for their parents until the root is reached

3. Select the best move (i.e. the move with the highest utility value)





Perfect Decisions by Minimax Algorithm

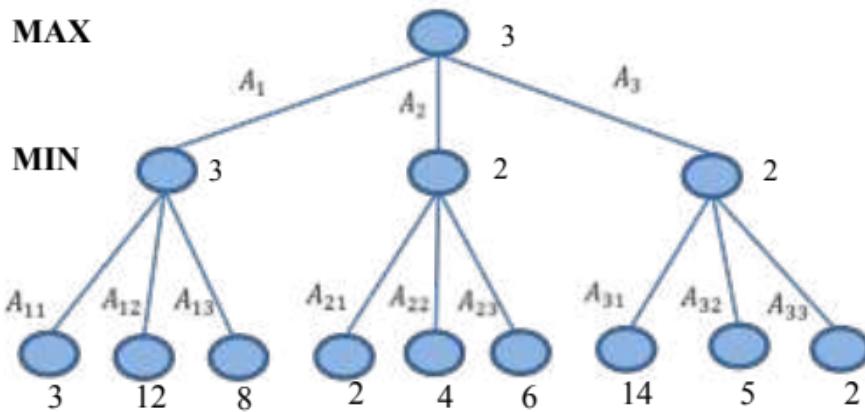
Perfect decisions. **no** time limit is imposed

- generate the **complete** search tree

Two players: **MAX** and **MIN**

- Choose move with best achievable payoff against best play
- **MAX** tries to **max** the utility, assuming that **MIN** will try to **min** it

Example





Othello 4

X	O		
O	X		

- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them

After placing the new piece, the pieces from the opponent player will be captured and become the pieces from the same Player

- The player with the most pieces on the board wins





'X' plays first

X considers the game now

O considers the game now

X

X

X

X

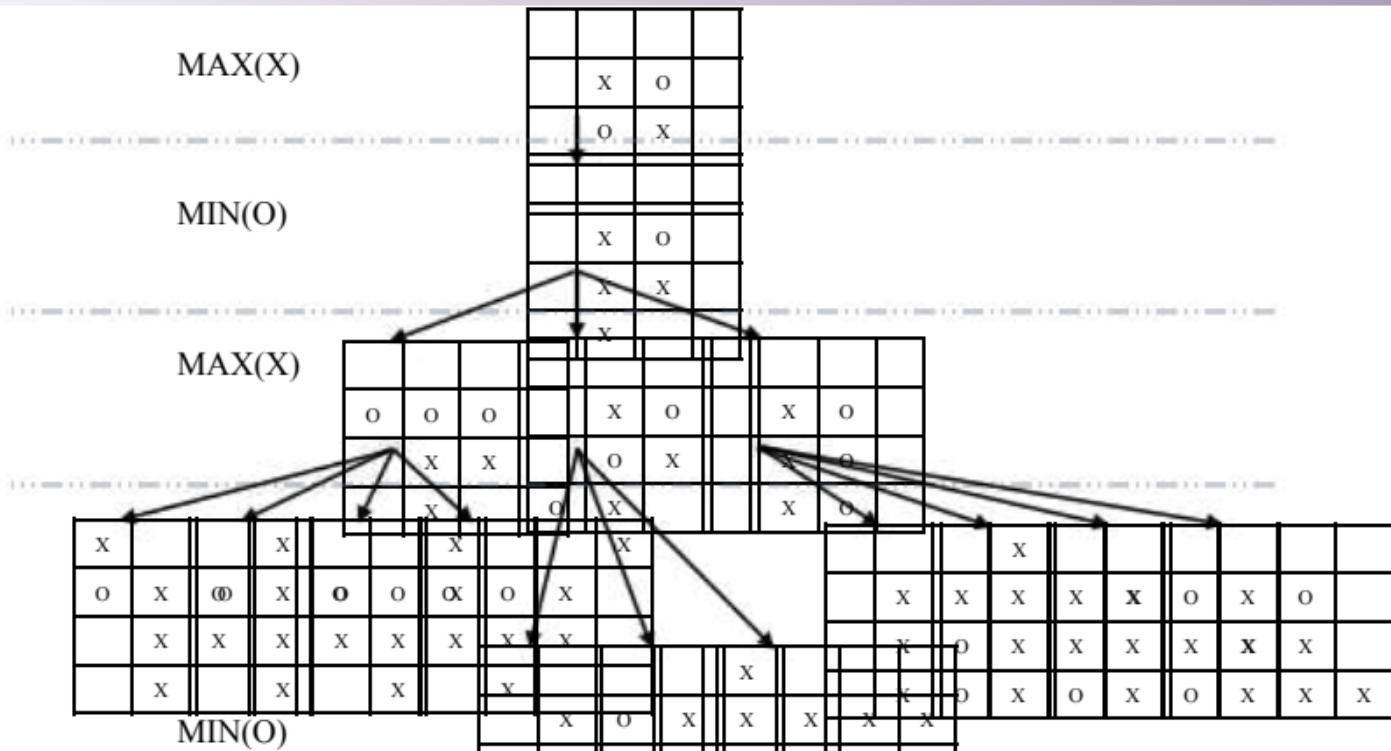
X

X considers the game now





Game Tree Othello 4





Imperfect Decisions

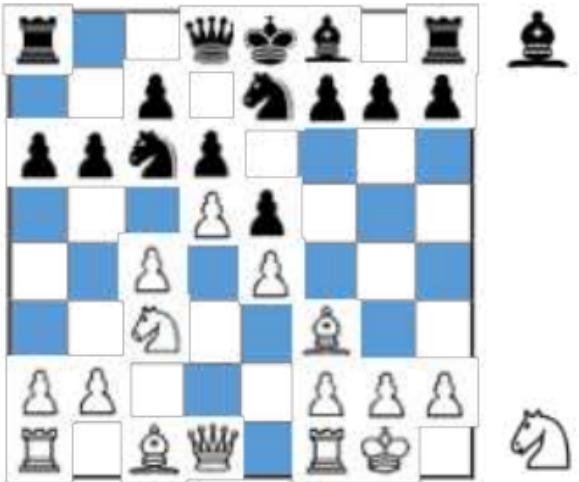
- For chess, branching factor ≈ 35 , each complete game tree, need to examine:
Time/space requirements \rightarrow complete **impractical** to make perfect decisions
Modifications to minimax algorithm





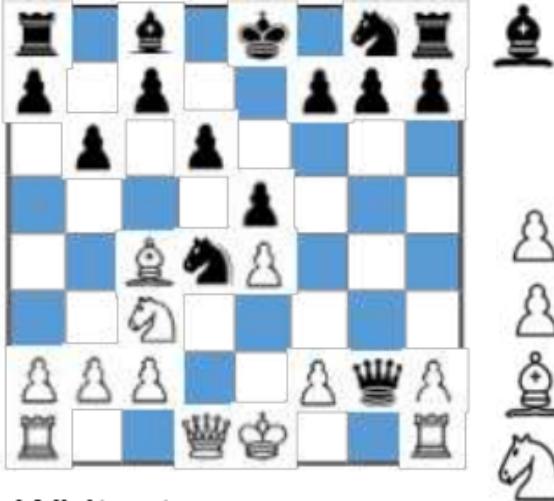
Evaluation Functions

Returns an **estimate** of the expected utility of the game from a given position



Black: to move

White: slightly better



White: to move

Black: winning





Evaluation Functions...

Requirements

- Computation is **efficient**
- Agrees with utility function on **terminal states**
- **Accurately** reflects the chances of winning

Trade off between accuracy and efficiency

Example (Chess)

Define **features**

- e.g., (number of white queens) – (number of black queens)

Use a **weighted sum** of these features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$$

- Need to learn the weight



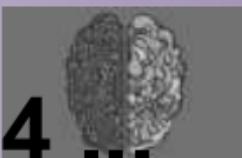


Evaluation Functions for Othello 4

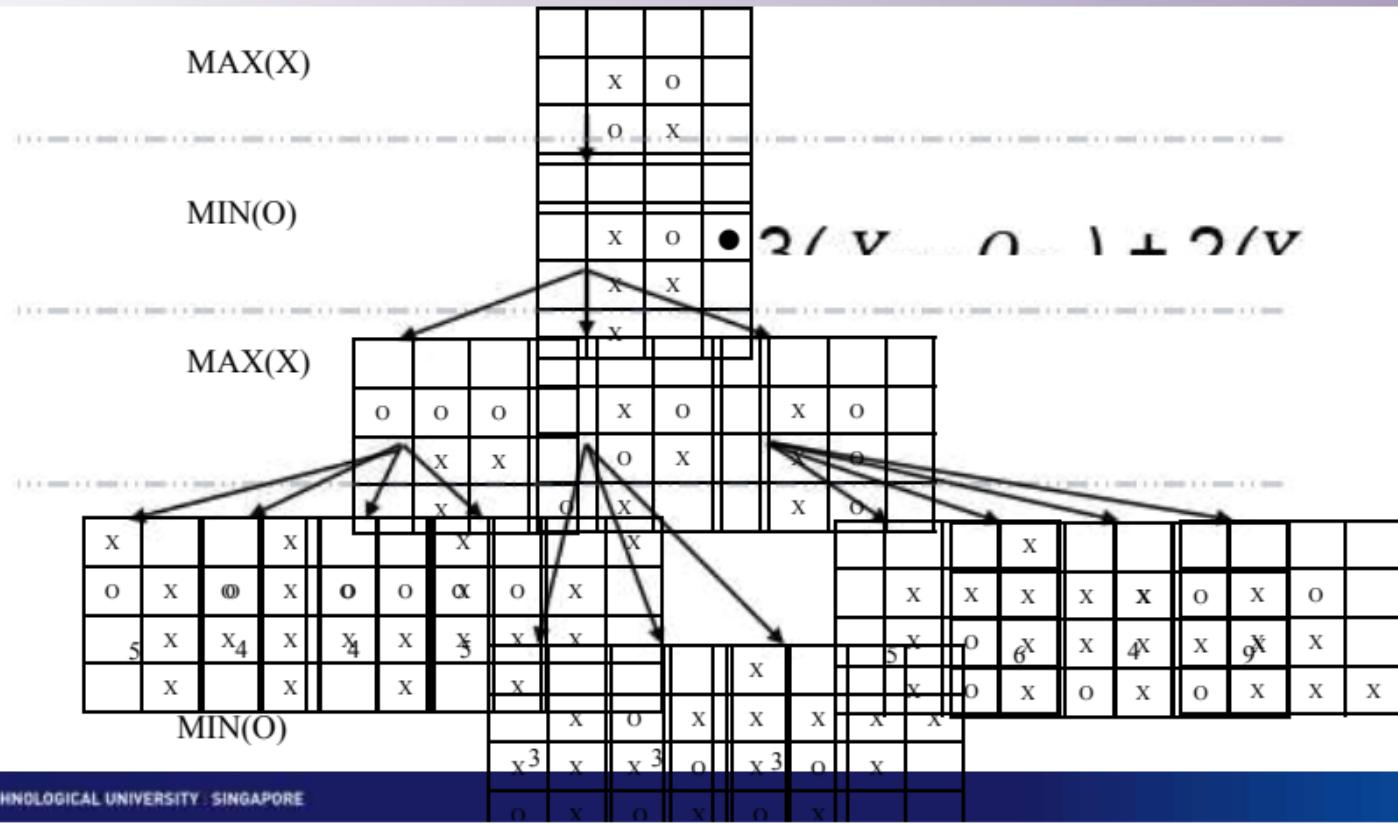
- - A corner of the board is one of piece at the corner can help capture opponent player
 - A square at the border is also the middle of the board

Heuristics for 'X' is proposed as follows:



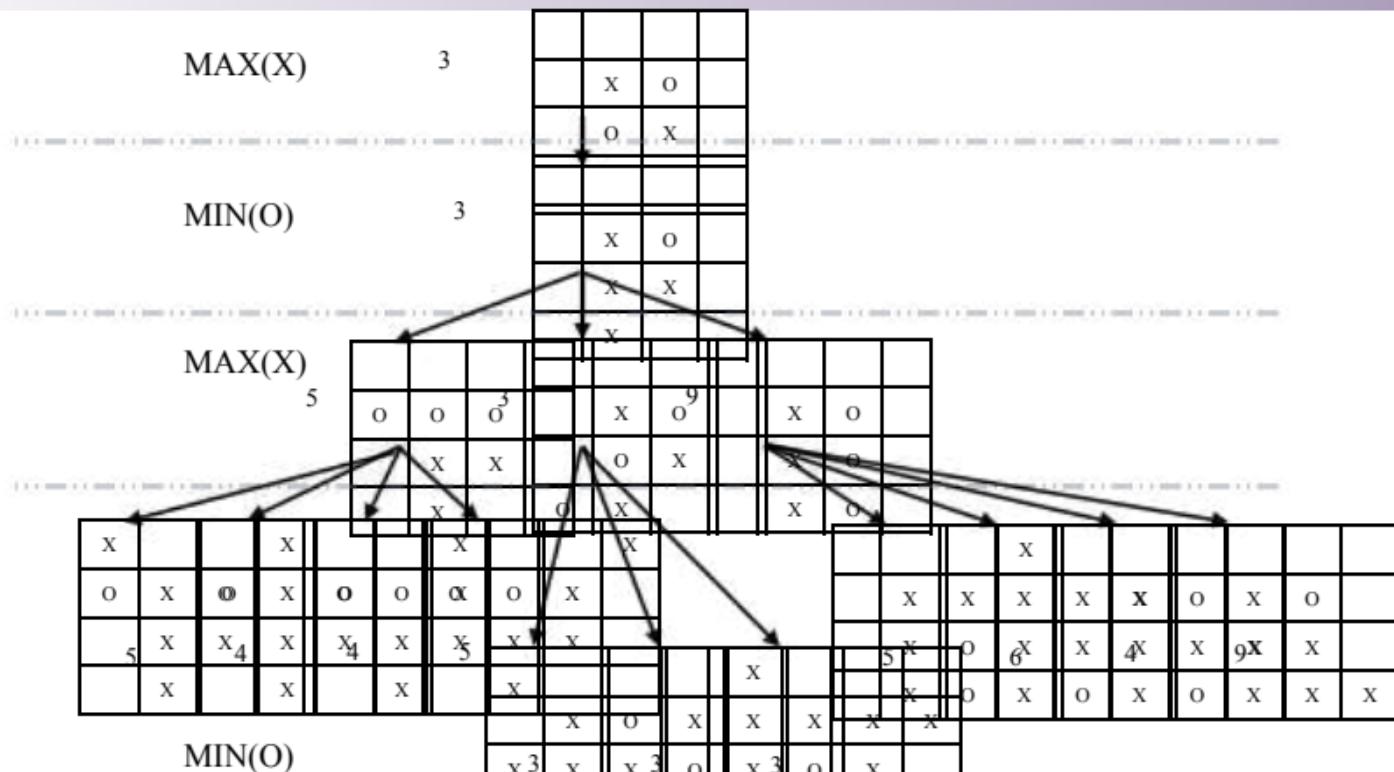


Evaluation Functions for Othello 4 ...





Minimax Search for Othello 4





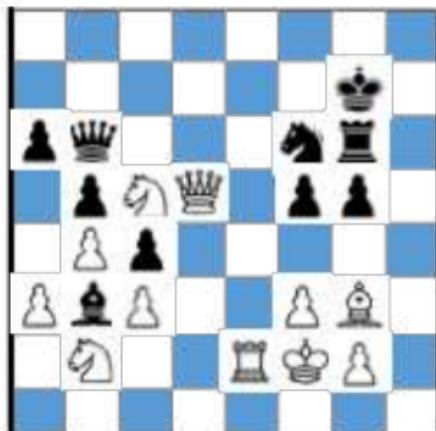
Quiescent Search

The evaluation function should only be applied to **quiescent** positions

- positions that are **not** likely to have large variations in evaluation in the near future

Example (Chess)

Positions in which favorable captures can be made



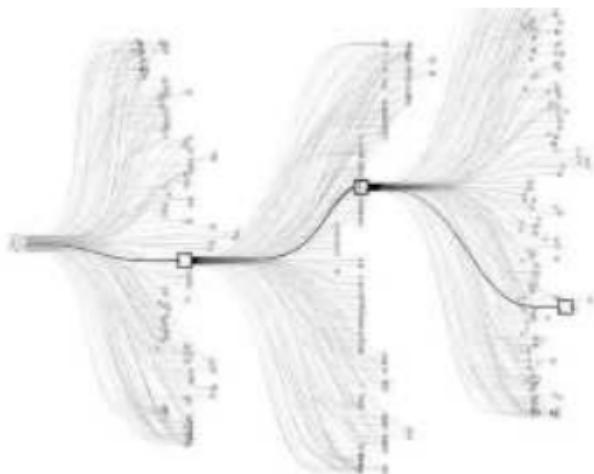
Black: to move
White: about to lose

Expansion of non-quiescent positions until quiescent positions are reached

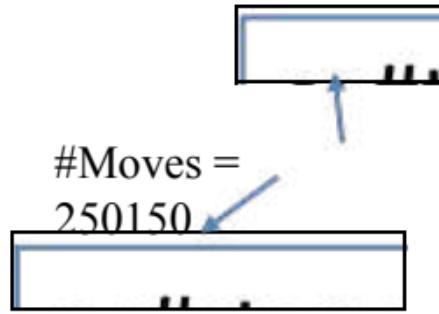




Revisit: Alpha Go

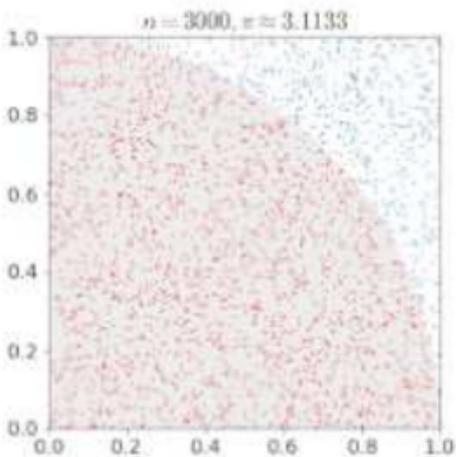


#Moves =
250150





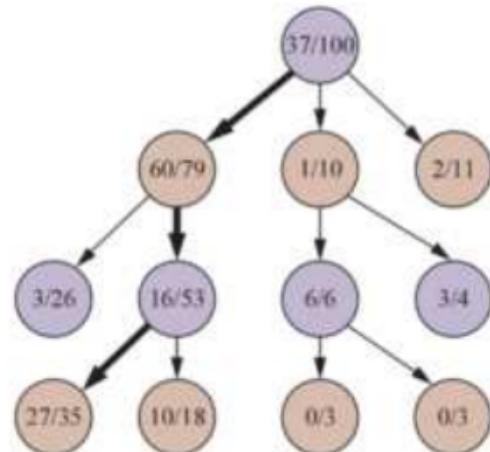
Another Example: Monte Carlo Search Tree (MCST)



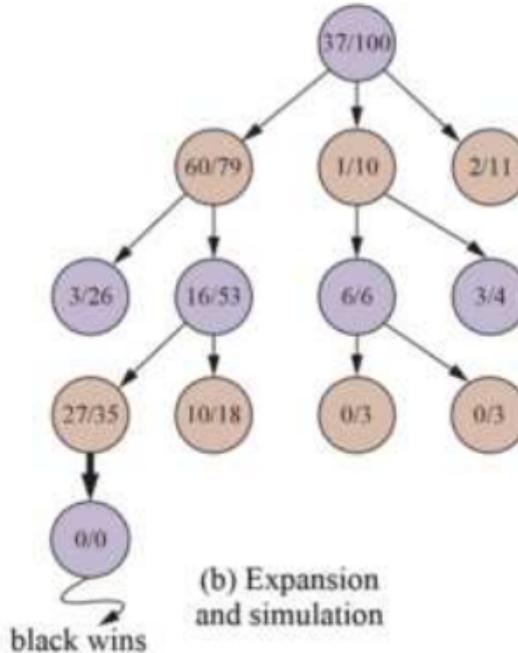


MCST Key Steps

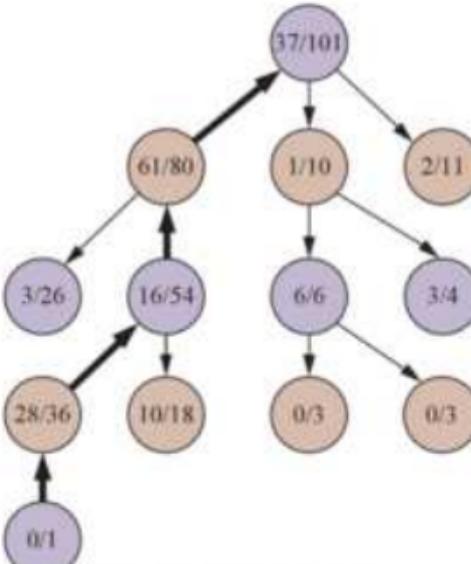
#Wins/#Games



(a) Selection



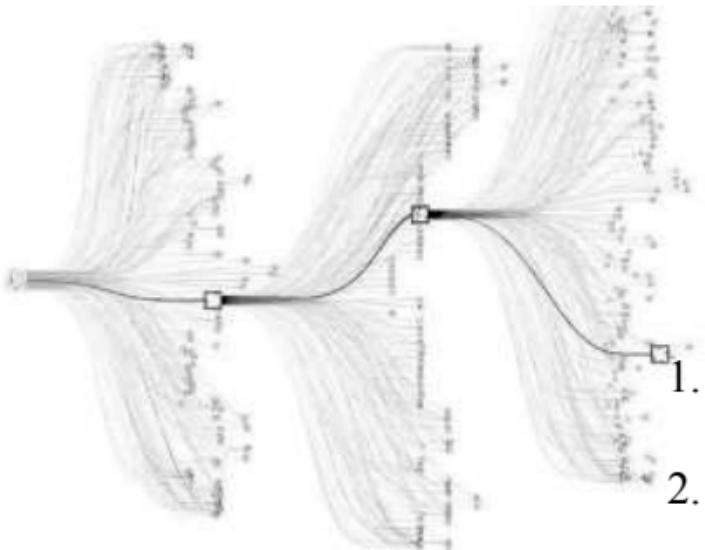
(b) Expansion
and simulation



(c) Backpropagation



MCST for Go is very expensive & slow



Why? The deeper you move, the wider you are
#number of simulations is exploded

Solution:

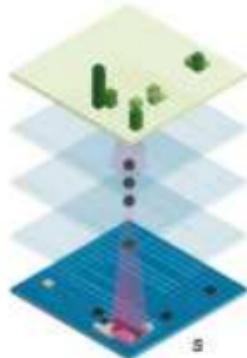
1. For selection/expansion, no longer stats
 (xx/xx) but $f = g + h$
2. For simulation, no longer real self-play but $f = g + h$



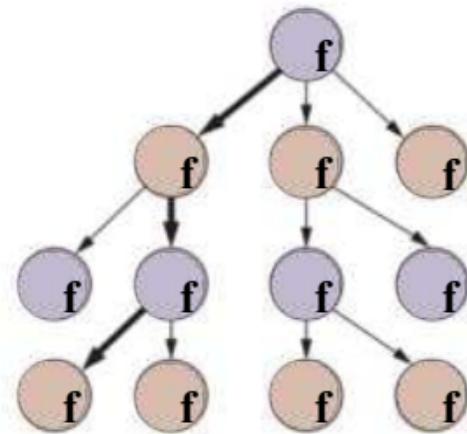
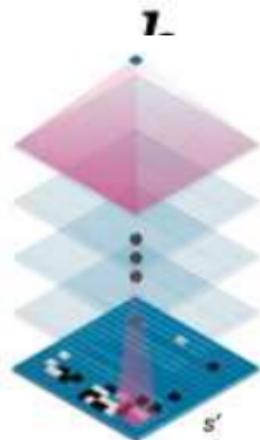


MCST: Update $f = g() + h()$

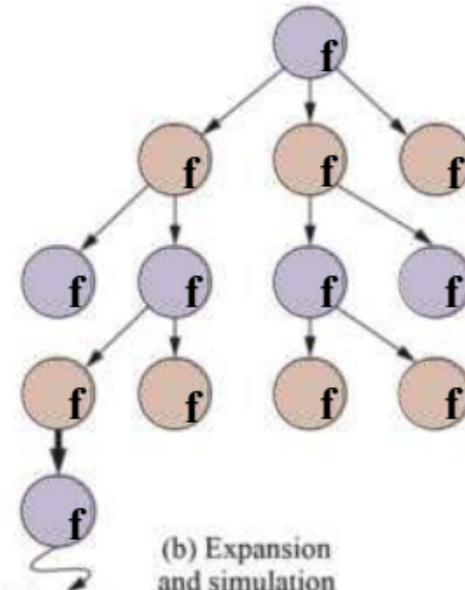
Policy Network



Value Network



(a) Selection



(b) Expansion
and simulation

No real play,
just an estimation





CZ3005

Artificial Intelligence

Markov Decision Process

Asst/P Hanwang Zhang

<https://personal.ntu.edu.sg/hanwangzhang/>

Email: hanwangzhang@ntu.edu.sg





Lesson Outline

- Introduction
- Markov Decision Process
- Two methods for solving MDP
 - Value iteration
 - Policy iteration





Introduction

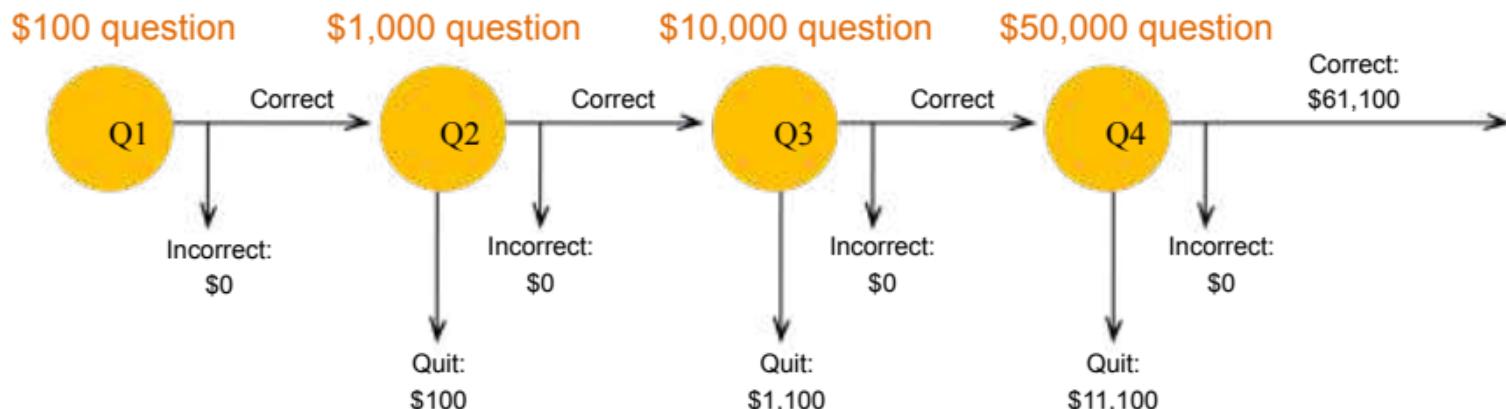
- We consider a framework for decision making under uncertainty
- Markov decision processes (MDPs) and their extensions provide an extremely general way to think about how we can act optimally under uncertainty
- For many medium-sized problems, we can use the techniques from this lecture to compute an optimal decision policy
- For large-scale problems, approximate techniques are often needed (more on these in later lectures), but the paradigm often forms the basis for these approximate methods





Game Show

- A series of questions with increasing level of difficulty and increasing payoff
- Decision: at each step, take your earnings and quit, or go for the next question
 - If you answer wrong, you lose everything





Recursive Mechanism

Week 4 (Q4-Q5)

4. For each of the following algorithms, find a recurrence equation for the number of multiplications as a function of N , and find the asymptotic growth rate of the solution to equation. To simplify the analysis, you may assume that N is a power of two; that is $N = 2^k$ for some integer K . These functions compute X to the power N , where N is a positive integer.

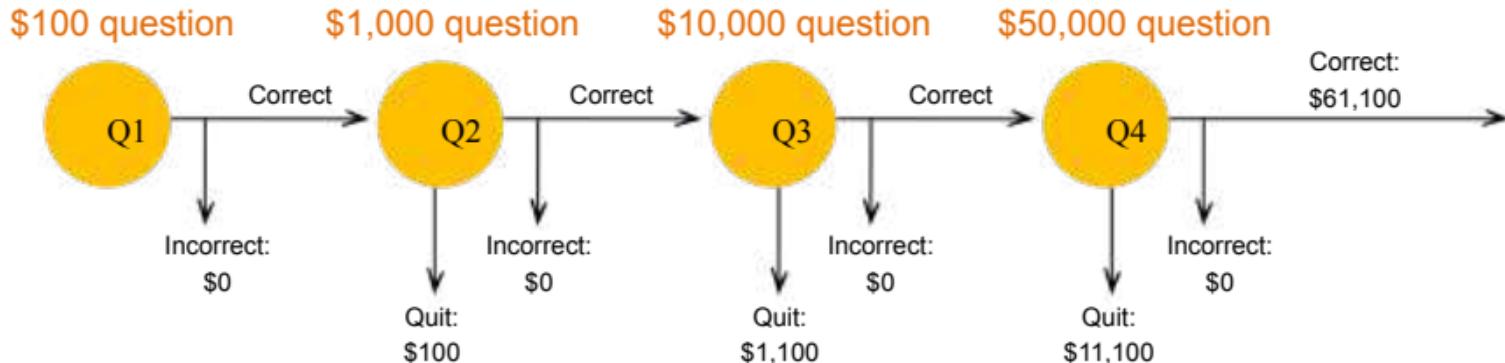
```
int power1(int X,int N)
{
    if (N == 1) return(X);
    else return X * power1(X, N-1);
}
```





Game Show

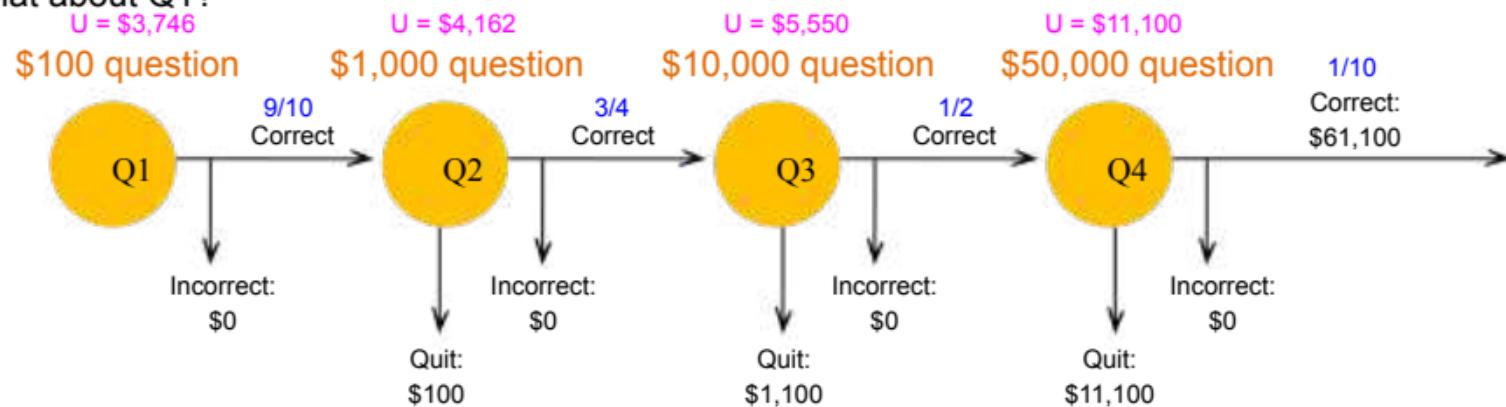
- Consider \$50,000 question
 - Probability of guessing correctly: 1/10
 - Quit or go for the question?
- What is the expected payoff for continuing?
 $0.1 * 61,100 + 0.9 * 0 = 6,110$
- What is the optimal decision?





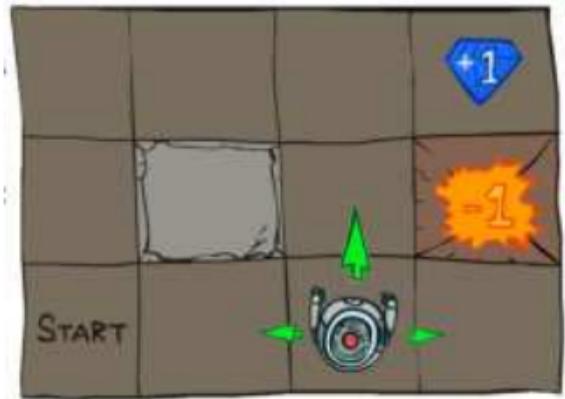
Game Show

- What should we do in Q3?
 - Payoff for quitting: \$1,100
 - Payoff for continuing: $0.5 * \$11,100 = \$5,550$
- What about Q2?
 - \$100 for quitting vs. \$4,162 for continuing
- What about Q1?



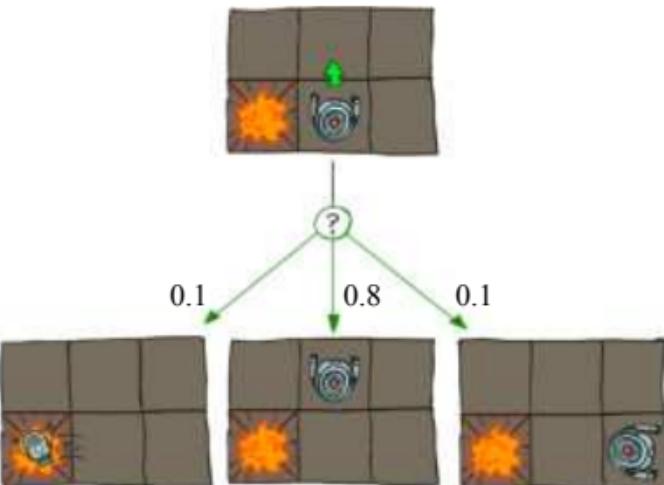


Grid World

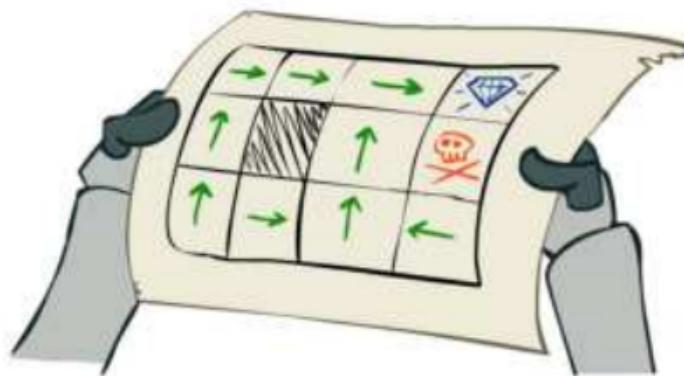
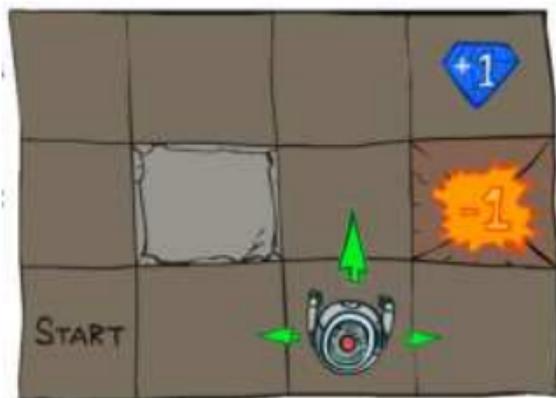


$R(s) = -0.04$ for every non-terminal state

Transition model:

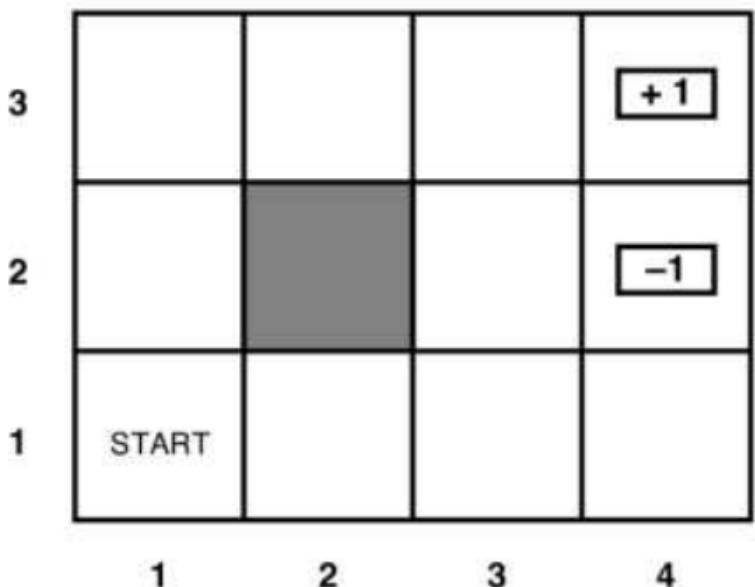


Goal: Policy

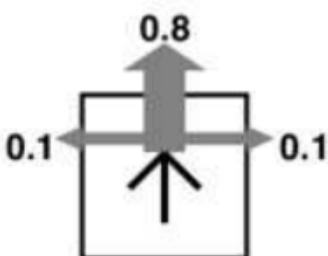




Grid World



Transition model:

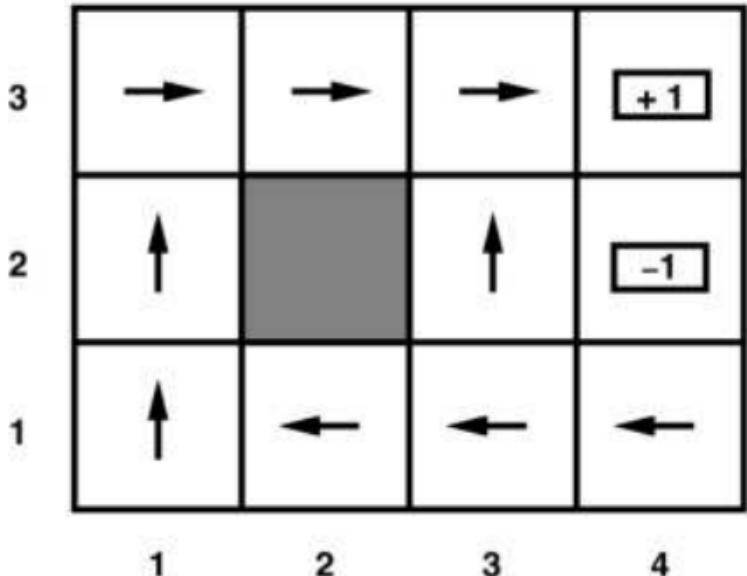


$R(s) = -0.04$ for every non-terminal state





Grid World



Optimal policy when
 $R(s) = -0.04$ for every
non-terminal state



Atari Video Games



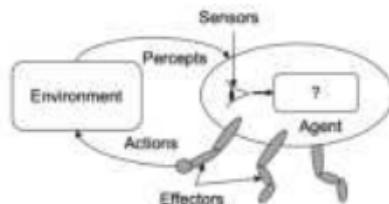
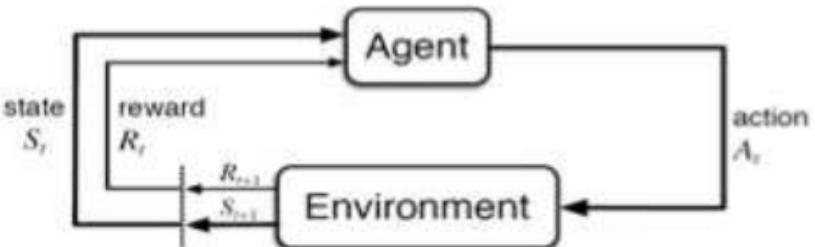
Execute actions
maximum
accumulated rewards



to get the

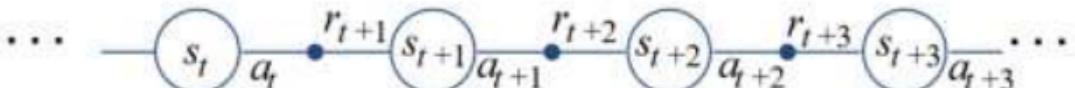


The Agent-Environment Interface



Agent and environment interact at discrete time steps.

Agent:





Making Complex Decisions

- Make a sequence of decisions
 - Agent's utility depends on a sequence of decisions
 - Sequential Decision Making
- Markovian Property
 - Transition properties depend only on the current state, not on previous history (how that state was reached)
 - Markov Decision Processes





Markov Decision Processes

- Formulate the agent-environment interaction
- Components:
 - **Markov States** s , beginning with s_0
 - **Actions** a
 - Each state s has actions A_s



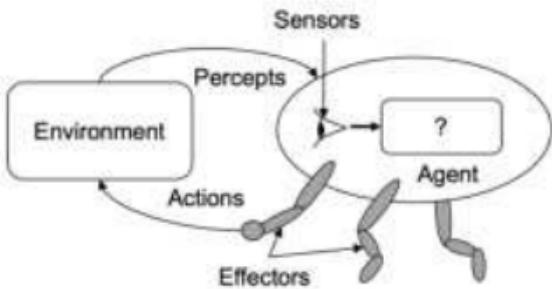
RAP-Star: PI





Solving MDPs

- MDP components
 - States s
 - Actions a
 - Transition model $P(s'$
 - Reward function $R(s)$





Maximizing Accumulated Rewards

- The optimal policy should maximise trajectories like $\tau = \langle S_1, A_1, R_1, \dots, S_n \rangle$





Accumulated Rewards

- Normally, we would define the *accumulated rewards* of trajectories as the discounted sum of the rewards
- **Problem:** infinite time horizon
- **Solution:** *discount* the individual rewards by a factor between 0 and 1:

$$G_t = R_t + \gamma R_{t+1}$$

- Sooner rewards count more than later rewards
- Makes sure the total *accumulated rewards* stays bounded
- Helps algorithms converge





Value Function

- The “true” value of a state, denoted v^* , is the total rewards if the agent executes an optimal policy π^* starting from state s .

$$v^*(s) = \mathbb{E}_{\pi^*} [r]$$

- Similarly, we define the action-value function Q^* :





Recursive Mechanism

Week 4 (Q4-Q5)

4. For each of the following algorithms, find a recurrence equation for the number of multiplications as a function of N , and find the asymptotic growth rate of the solution to equation. To simplify the analysis, you may assume that N is a power of two; that is $N = 2^k$ for some integer K . These functions compute X to the power N , where N is a positive integer.

```
int power1(int X,int N)
{
    if (N == 1) return(X);
    else return X * power1(X, N-1);
}
```





Method 1: Value Iteration

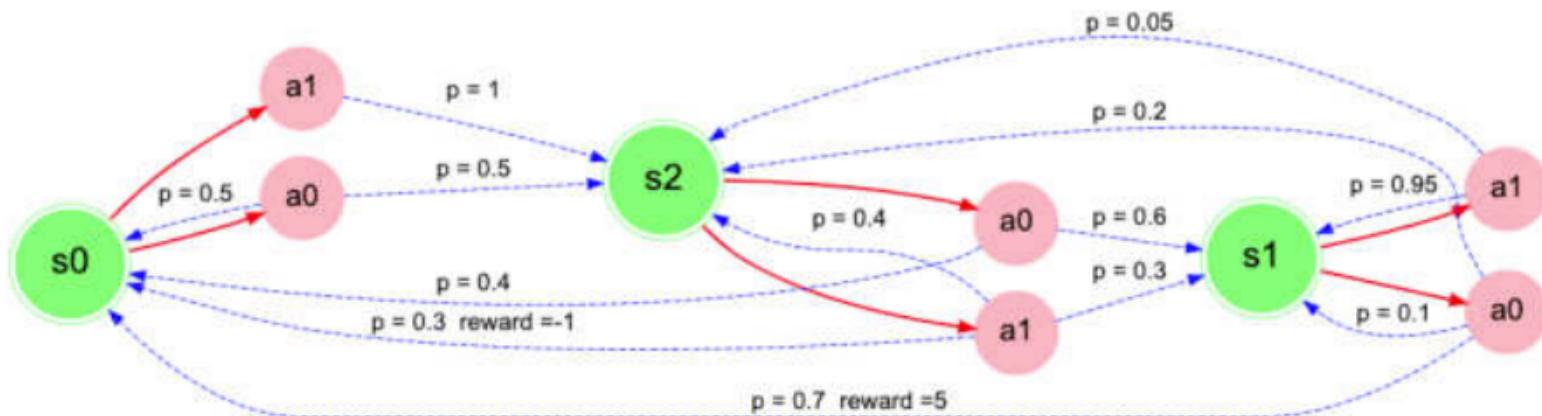
- Start out with every $V(s) = 0$
- Iterate until convergence
 - During the i th iteration, update rule:





Value Iteration: Example

- A simple example to show how VI works
 - State, action, reward (non-zero) and transition probability are shown in the figure
 - We use this example as an MDP and solve it using VI and PI



These structures can be easily implemented by **dict** of Python or **HashMap** of Java



Value Iteration: Example (cont'd)

- Given the states and actions a represent the value function V
- 1. Initialize $V_0(s)$, for all s
- Pseudo-code of VI





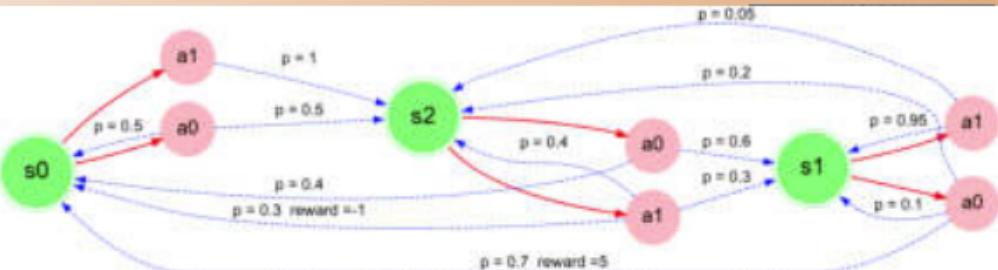
Value Iteration: Example (cont'd)

- How VI works in an iteration?

For all state find

Finally

Value Iteration: Ex



- We use the previous example and
- Construct a Q table, $Q(s, a)$

			-
	0	0	0
	0	0	0
	0	0	0

The initial Q and V table

3 rows (3 states) and 2 columns (2 actions)
 $V_0^0 = [0, 0, 0]$ # $\#3 \text{ states} = P(s_0 | s_1, a_1)$

Repeat this process until it converges

$$Q(s_0, a_1) = 0.7 \times [5 + 0] + 0$$



Value Iteration: Example (cont'd)

- Iterating the process (code available on later slides)

iter	0		$V(s_0) = 0.000$	$V(s_1) = 0.000$	$V(s_2) = 0.000$
------	---	--	------------------	------------------	------------------

iter	1		$V(s_0) = 0.000$	$V(s_1) = 3.500$	$V(s_2) = 0.000$
------	---	--	------------------	------------------	------------------

iter	2		$V(s_0) = 0.000$	$V(s_1) = 3.815$	$V(s_2) = 1.890$
------	---	--	------------------	------------------	------------------

iter	3		$V(s_0) = 1.701$	$V(s_1) = 4.184$	$V(s_2) = 2.060$
------	---	--	------------------	------------------	------------------

... ...

iter	63		$V(s_0) = 8.020$	$V(s_1) = 11.160$	$V(s_2) = 8.912$
------	----	--	------------------	-------------------	------------------

iter	64		$V(s_0) = 8.021$	$V(s_1) = 11.161$	$V(s_2) = 8.913$
------	----	--	------------------	-------------------	------------------

iter	65		$V(s_0) = 8.022$	$V(s_1) = 11.162$	$V(s_2) = 8.915$
------	----	--	------------------	-------------------	------------------

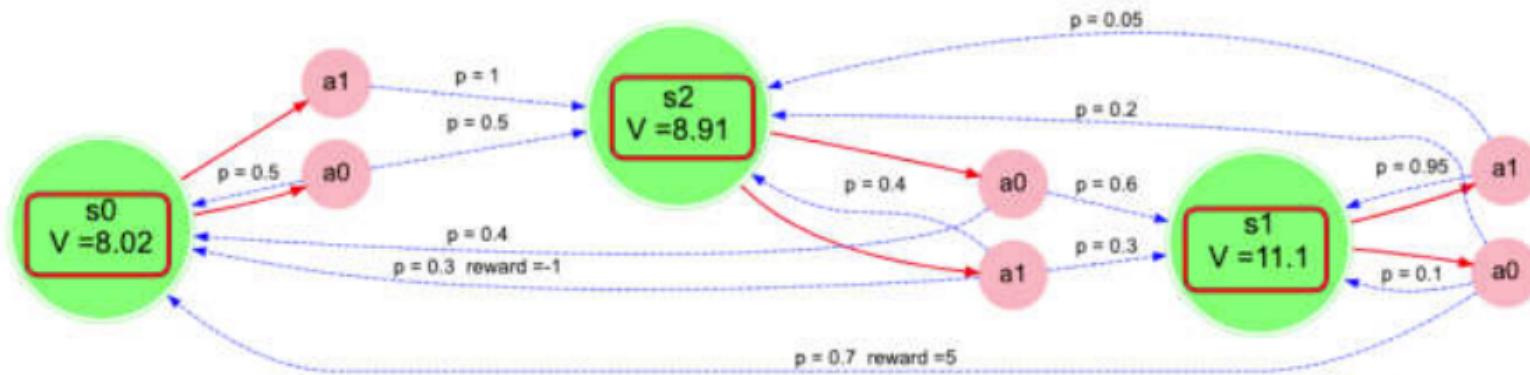
Very good, the
values converge
now





Value Iteration: Example (cont'd)

- Put the optimal values on the graph





Value Iteration: Example (cont'd)

- Use V^* to find optimal policy
 - aka optimal actions in each state



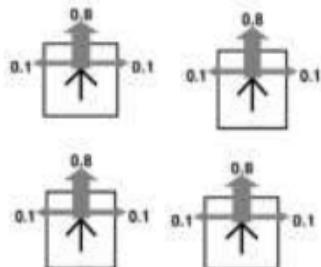
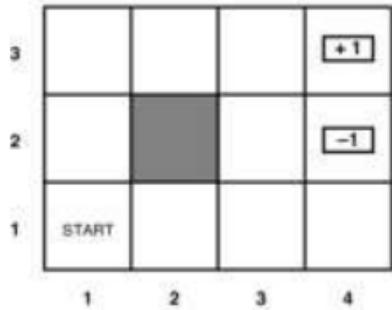
- Done! We get the optimal policy of the example

Search *Markov Decision Process (MDP) Toolbox for Python*





Grid World



$$Q_1(s_0, a_0) = P(s_0 | s_0, a_0)$$

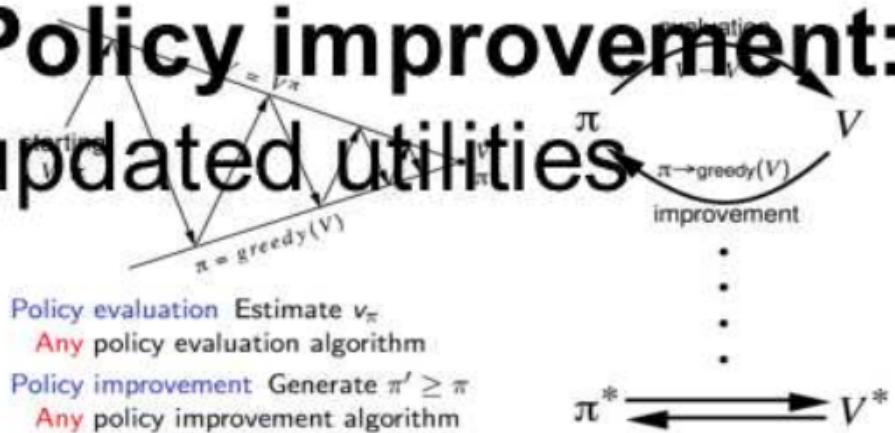
$P(s_1 | s_0, a_0)$

$$Q_1((1,1), Up) = 0.1 * [-0.4 + 0] \cdot$$



Method 2: Policy Iteration

- Start with some initial policy π_0 and
 - **Policy evaluation:** calculate the updated utilities
 - **Policy improvement:** calculate the updated utilities





Policy Iteration: Main Steps

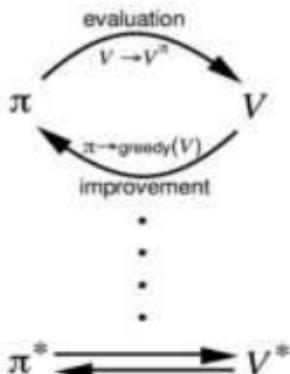
- Unlike VI, policy iteration has to map states - and estimate V^{π_i} based on
- Iterate this process until converge
- Steps of PI
 - 1. Initialization
 - 2. Policy Evaluation (calculating the V)





Policy Iteration: Detailed Steps

- 1. Initialization
 - a. Initialize $V(s)$ and $\pi(s)$ for all state s
Use current Policy to stabilize Value
- 2. Policy Evaluation (calcul)
 - a. Repeat
 - b. $\Delta \leftarrow 0$
 - c. For each state s :
 - d. $v \leftarrow V(s)$
 - e. $V(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) [R(s, h(s)) + \gamma v]$
Use current Value to stabilize Policy

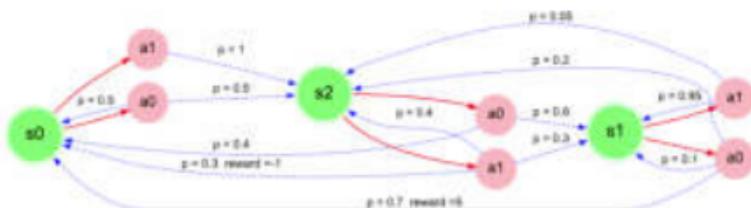




Policy Iteration: Policy Evaluation

- Use the example used in VI
- Start iteration $i = 0$

$$V(s_0) \leftarrow P(s_0, \pi(s_0), s_0)[$$



$$V(s_1) \leftarrow P(s_1, \pi(s_1), s_0)[R($$

$$V(s_2) \leftarrow P(s_2, \pi(s_2), s_0)[R($$



Policy Iteration: Policy Evaluation (cont'd)

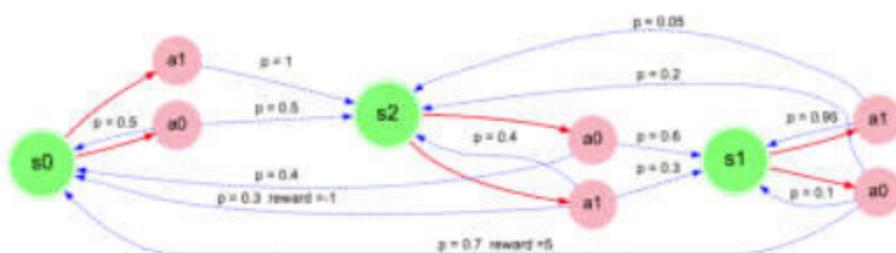
- Use the example used in VI
- Start iteration $i = 0$, $\gamma = 0.99$, initialize values are calculated asynchronously
The converged values are used for policy improvement



$$V(s_0) = 0.5$$

$$V(s_0) = 0.3 + (-1) + 0$$

... loop this process as instructed in step 2

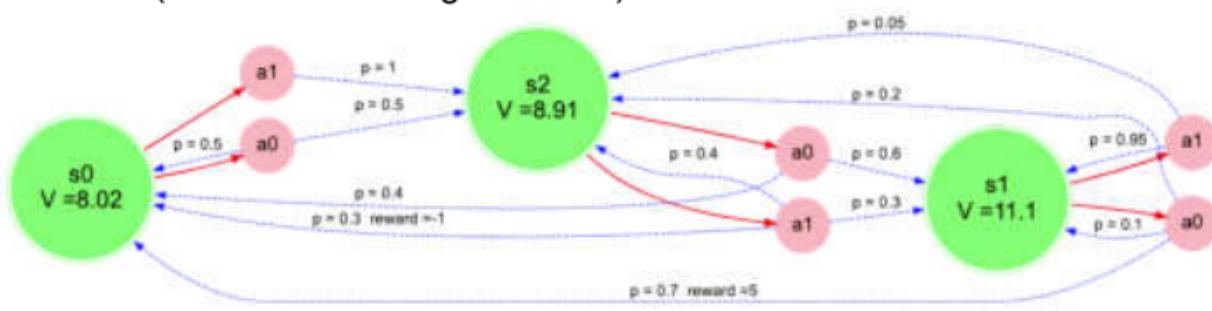




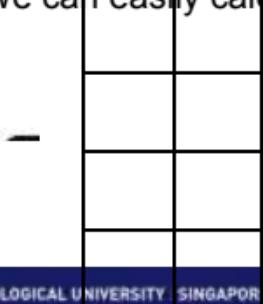
Policy Iteration: Example (cont'd)

- The optimal Q values are
 - Nearly the same as that of VI (as shown in the figure below)

	8.03
	11.2
	8.9



- We can easily calculate the optimal policy. Can you try it?





CZ3005 Artificial Intelligence

Reinforcement Learning

Assoc Prof Bo AN

Research area: artificial intelligence, computational game theory, reinforcement learning, optimization

www.ntu.edu.sg/home/boan

Email: boan@ntu.edu.sg

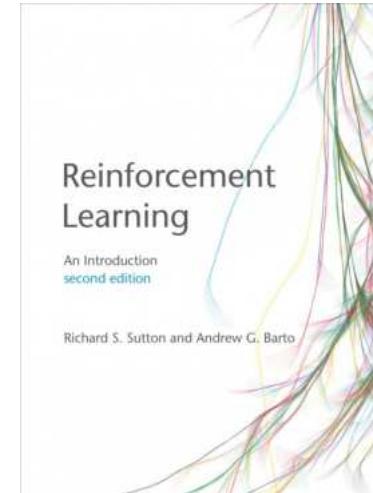
Office: N4-02b-55





Lesson Outline

- Some RL algorithms:
 - Monte-Carlo
 - Q-learning
 - Deep Q-Network





Reinforcement Learning

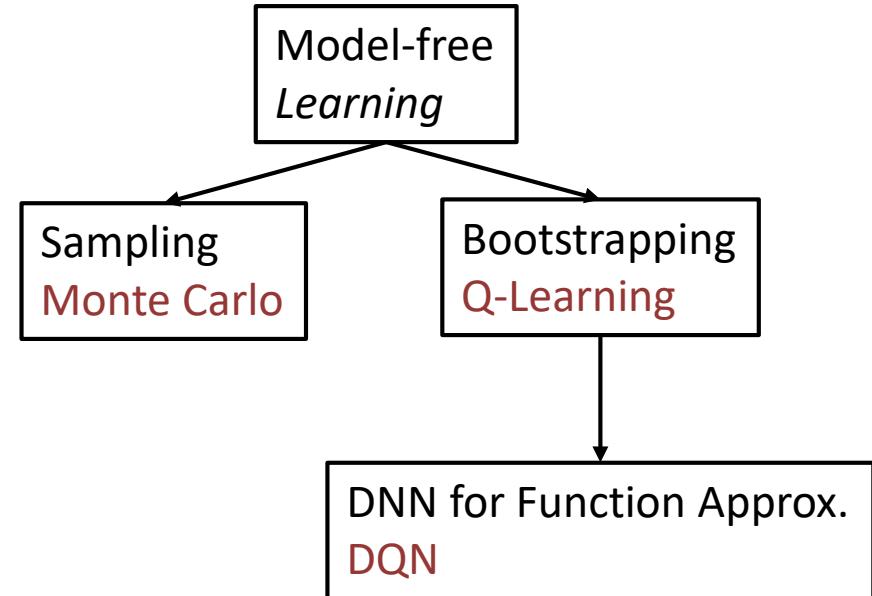
- Motivation
 - In last lecture, we compute **the value function** and find **the optimal policy**
 - But if without the transition function $P(s'|s, a)$?
 - We can learn **the value function** and find **the optimal policy** without transition
 - From experience

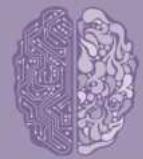




RL algorithms

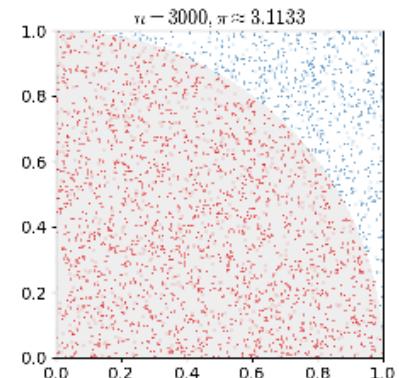
- Types
 - Monte Carlo
 - Q-Learning
 - DQN
 - ...





What is Monte Carlo

- Idea behind MC:
 - Just use randomness to solve a problem
- Simple definition:
 - Solve a problem by generating suitable random numbers and observing the fraction of numbers obeying some properties
- An example for calculating π (not policy in RL):
 - $S_{red} = \frac{1}{4}\pi r^2, S_{square} = r^2$
 - putting dots on the square randomly for $n = 3000$ times
 - $\pi \approx 4 \times \frac{N_{red}}{n}$, N_{red} is the number of dots in the circle

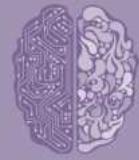




Monte Carlo in RL: Prediction

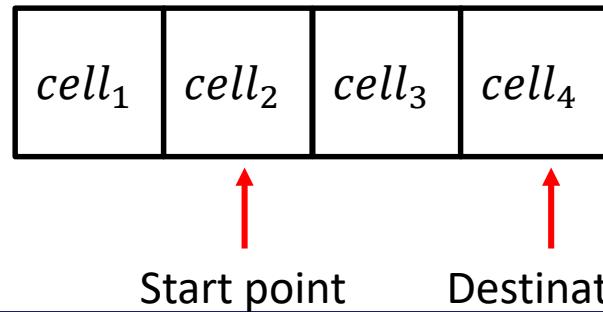
- Basic Idea: we run in the world randomly and gain experience to learn
- What experience? Many trajectories!
 - $(s_1, a_1, r_2, s_2, a_2, r_3, \dots, s_T), \dots$
- What we learn? Value function!
 - Recall that the return is the total discounted rewards:
$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^n r_{t+n} + \dots = \sum_i \gamma^i r_{t+i}$$
 - Recall that the value function is the expected return from s
$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$
- How we learn?
 - Use experience to learn an empirical state value function $\tilde{V}_\pi(s) = \frac{1}{N} \sum_{i=1}^N G_{i,s}$





An Example

- One-dimensional grid world
 - A robot is in a 1×4 world
 - State: current cell $s \in [cell_1, cell_2, cell_3, cell_4]$
 - Action: left or right
 - Reward:
 - Move one step (-1)
 - Reach the destination cell (+10) (ignoring the one-step reward)



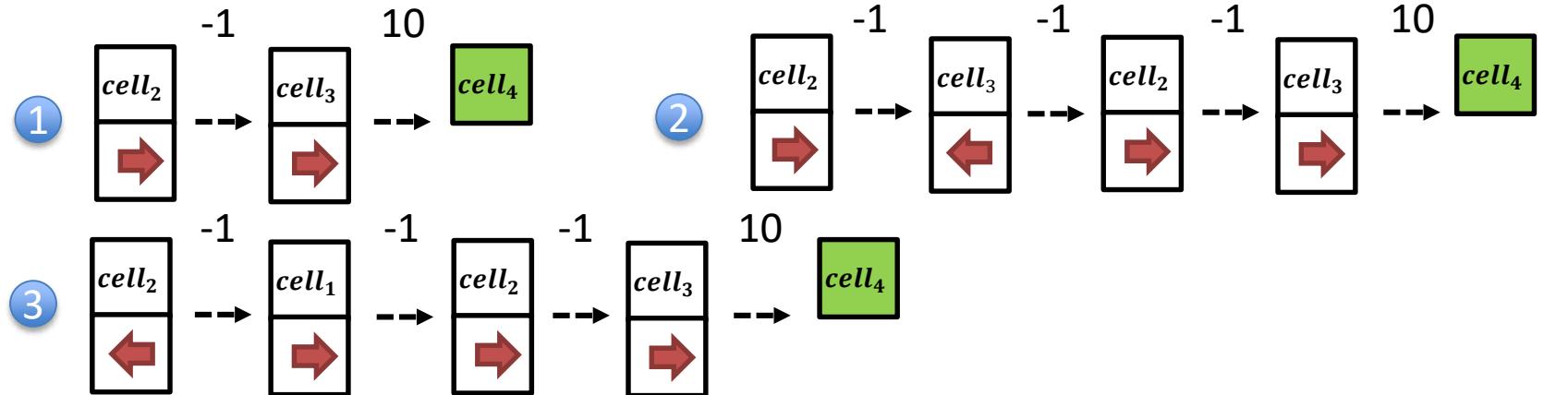


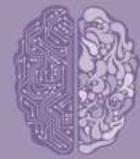
One-dimensional Grid World

- Trajectory or episode:

- The sequence of states from the starting state to the terminal state
 - Robot starts in $cell_2$, ends in $cell_4$

- The representation of the three episodes



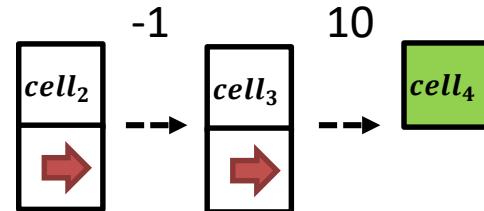


Compute Value Function

- Idea: Average return observed after visits to (s, a)
- First-visit MC: average returns only for **first** time (s, a) is visited in an episode
- Return in one episode (trajectory):

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^n r_{t+n} + \dots = \sum_i \gamma^i r_{t+i}$$

- We calculate the return for $cell_2$ of first episode with $\gamma = 0.9$



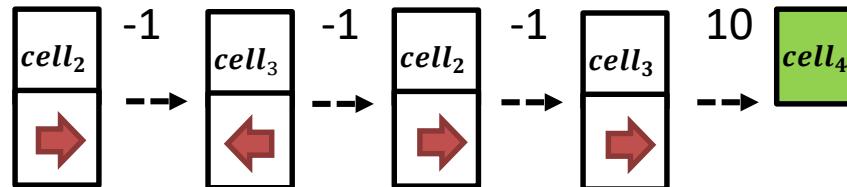
$$G_t = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$





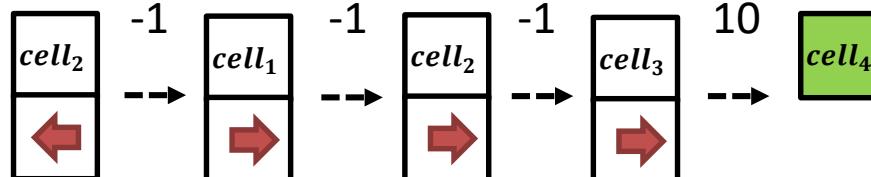
Compute Value Function (cont'd)

- Similarly the return for $cell_2$ of second episode with $\gamma = 0.9$



$$G_t = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 + 10 \times 0.9^3 = 4.58$$

- Similarly the return for $cell_2$ of third episode with $\gamma = 0.9$



$$G_t = -1 \times 0.9^0 - 1 \times 0.9^1 - 1 \times 0.9^2 + 10 \times 0.9^3 = 4.58$$

- The empirical value function for $cell_2$ is $\frac{8 + 4.58 + 4.58}{3} = 5.72$



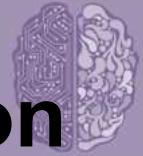
Compute Value Function (cont'd)

- Given these three episodes, we compute the value function for all non-terminal state

6.2 <i>cell</i> ₁	5.72 <i>cell</i> ₂	8.73 <i>cell</i> ₃
---------------------------------	----------------------------------	----------------------------------

- We can get more accurate value function with more episodes





First Visit Monte Carlo Policy Evaluation

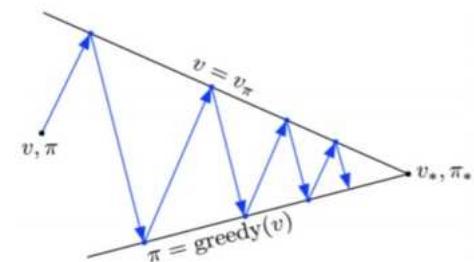
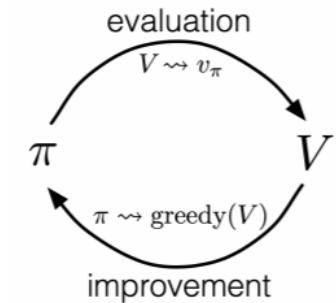
- Average returns only for the first time s is visited in an episode
- Algorithm
 - Initialize:
 - $\pi \leftarrow$ policy to be evaluated
 - $V \leftarrow$ an arbitrary state-value function
 - $Returns(s) \leftarrow$ an empty list, for all state s
 - Repeat many times:
 - Generate an episode using π
 - For each state s appearing in the episode:
 - $R \leftarrow$ return following **the first occurrence** of s
 - Append R to $Returns(s)$
 - $V(s) \leftarrow average(Returns(s))$

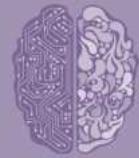




Monte Carlo in RL: Control

- Now, we have the value function of all states given a policy
- We need to improve policy to be better
- Policy Iteration
 - Policy evaluation
 - Policy improvement
- However, we need to know how good an action is



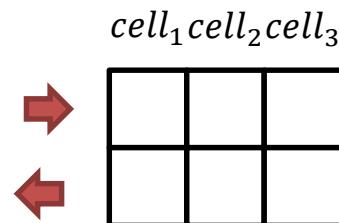


Q-value

- Estimate how good an action is when staying in a state
- Defined as the expected return starting from s , taking the action a and thereafter following policy π

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

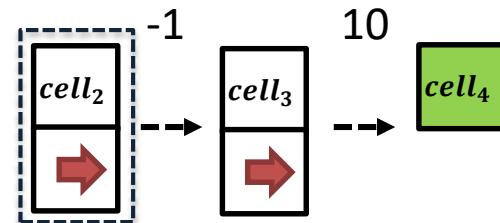
- Representation: A table
 - Filled with the Q-value given a state and an action





Computing Q-value

- MC for estimating Q:
 - A slight difference from estimating the value function
 - Average returns for state-action pair (s, a) is visited in an episode
- We calculate the return for $(cell_2, \text{right})$ of first episode with $\gamma = 0.9$



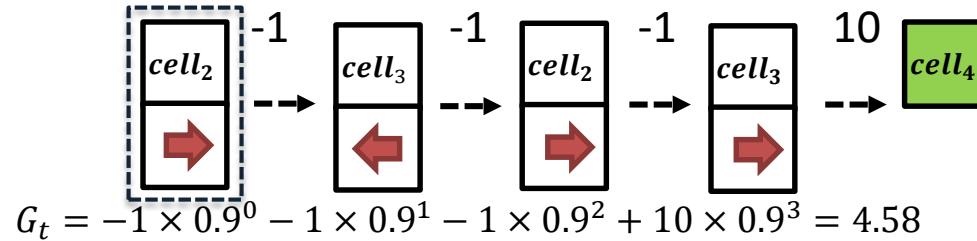
$$G_t = -1 \times 0.9^0 + 10 \times 0.9^1 = 8$$



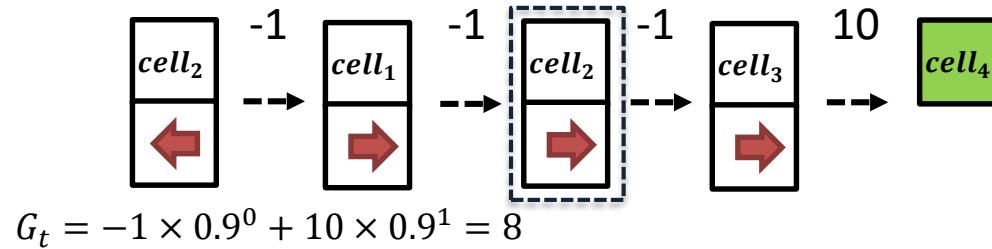


Compute Q-Value (cont'd)

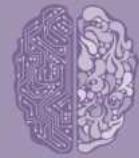
- Similarly the return for $(cell_2, \text{right})$ of second episode with $\gamma = 0.9$



- Similarly the return for $(cell_2, \text{right})$ of third episode with $\gamma = 0.9$



- The empirical Q-value function for $(cell_2, \text{right})$ is $\frac{8 + 4.58 + 8}{3} = 6.86$



Q-Value for Control

- Filling the Q-table
 - By going through all state-action pairs, we get a complete Q-table with all the entries filled
 - A possible Q-table example

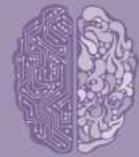
	$cell_1$	$cell_2$	$cell_3$
$cell_1$	8.1	9.3	9.9
$cell_2$	4.5	5.6	7.5
$cell_3$	✖	✖	✖

- Selecting action

$$\pi'(s) = \operatorname{argmax}_{a \in A} Q^\pi(s, a)$$

At $cell_1$, $cell_2$ and $cell_3$, we choose right





MC control algorithm

Policy evaluation

Policy improvement

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s, a) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

$\pi \leftarrow$ an arbitrary ε -soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$R \leftarrow$ return following the first occurrence of s, a

Append R to $Returns(s, a)$

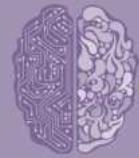
$Q(s, a) \leftarrow \text{average}(Returns(s, a))$

(c) For each s in the episode:

$a^* \leftarrow \arg \max_a Q(s, a)$

For all $a \in \mathcal{A}(s)$:

$$\pi(s, a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon / |\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon / |\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

Q-Learning

- Previously, we need the whole trajectory
- In Q-Learning, we only need one-step trajectory: (s, a, r, s')
- The difference is the Q-value computing
 - Previously:

$$\tilde{Q}_\pi(s, a) = \frac{1}{N} \sum_{i=1}^N G_{i,s}$$

- Now, updating rule:

$$Q_{new}(S_t, A_t) \leftarrow Q_{old}(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q_{old}(S_{t+1}, a) - Q_{old}(S_t, A_t))$$





Q-Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

 Initialize S

 Loop for each step of episode:

 Choose A from S using policy derived from Q (e.g., ε -greedy)

 Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

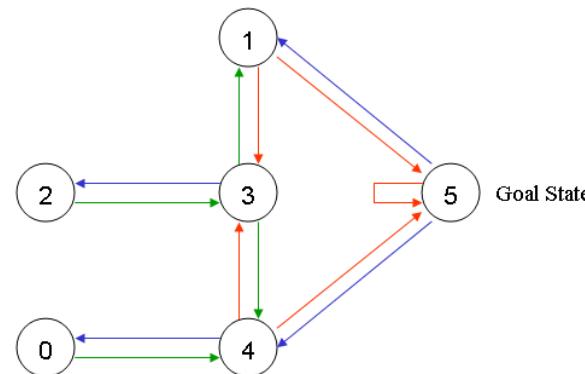
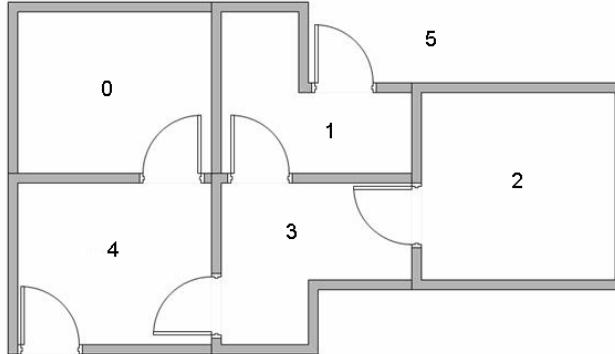
$$S \leftarrow S'$$

 until S is terminal



A Step-by-step Example

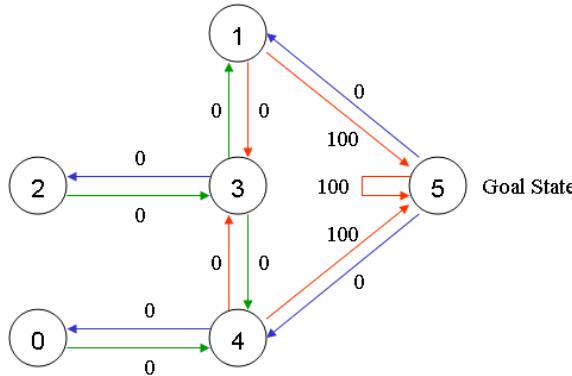
- 5-room environment as MDP
 - We'll number each room 0 through 4
 - The outside of the building can be thought of as one big room 5
 - End at room 5
 - Notice that doors at rooms 1 and 4 lead into the building from room 5 (outside)





A Step-by-step Example (cont'd)

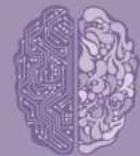
- Goal
 - Put an agent in any room, and from that room, go outside (or room 5)
- Reward
 - The doors that lead immediately to the goal have an instant reward of 100
 - Other doors not directly connected to the target room have zero reward



$$R = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{matrix} \right] \end{matrix}$$

state

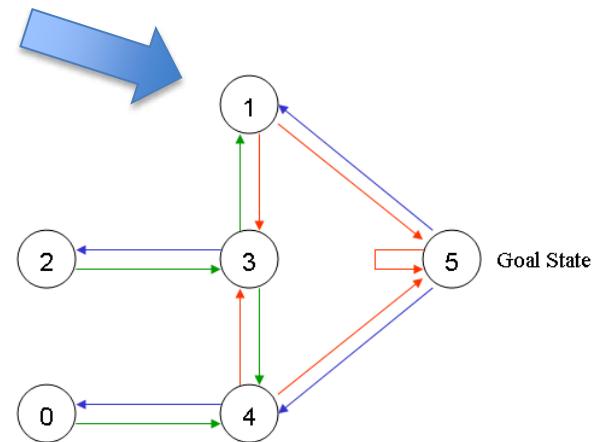




Q-Learning Step by Step

- Initialize matrix Q as a zero matrix
- $\alpha = 0.01, \gamma = 0.99$
- Loop for each episode until converge
 - Initial state: current we are in room 1 (1st outer loop)
 - Loop for each step of episode (until reach room 5)
 - ... (Next slide)

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix} \right] \end{matrix}$$





Q-Learning Step by Step (cont'd)

- ... (last slide)
 - Loop for each step of episode (until room 5)
 - By random selection, we go to 5
 - We get 100 reward
 - Update Q: $Q_{new}(S_t, A_t) \leftarrow Q_{old}(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q_{old}(S_{t+1}, a) - Q_{old}(S_t, A_t))$
 - At room 5, we have 3 possible actions: go to 1, 4 or 5; We select the one with max reward
 - $Q_{new}(1,5) \leftarrow Q_{old}(1,5) + \alpha \left(100 + \gamma \max_a Q_{old}(5, a) - Q_{old}(1,5) \right) = 0 + 0.01 \times (100 + 0.99 \times 0 - 0) = 1$

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$



$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$



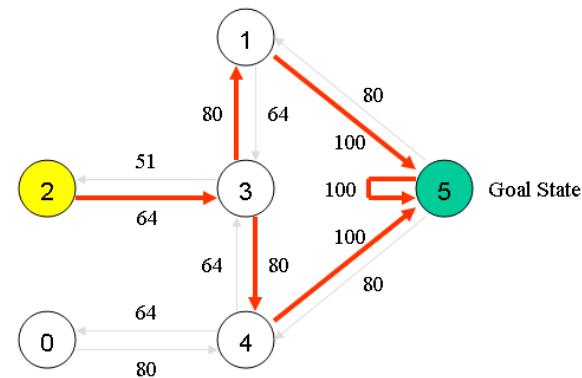


Q-Learning Step by Step (cont'd)

- When we loop many episodes, we can get

$$Q = \begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \left[\begin{matrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{matrix} \right] \end{matrix}$$

- According to this Q-table, we can select actions
 - E.g. We are at room 2
 - Greedily select based on maximum of Q value





An Example of Iteration Process

- A complex grid world example
- https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html





Deep Q-Network

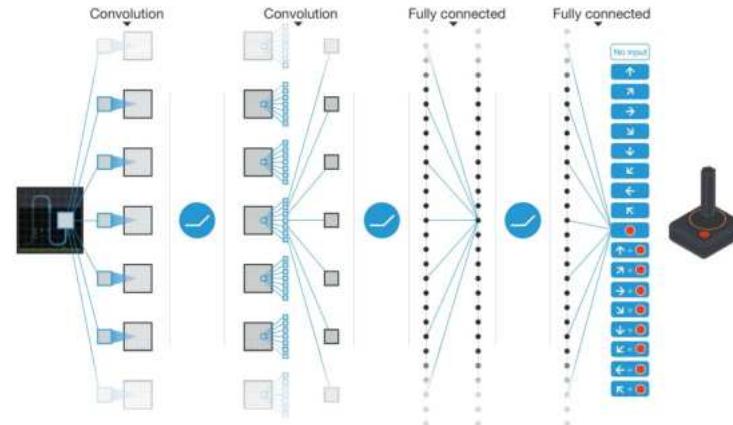
- Previously, we represent the Q-value as a table
- However, tabular representation is insufficient
 - Many real world problems have enormous state and/or action spaces
 - Backgammon: 10^{20} states
 - Computer Go: 10^{170} states
 - Robots: continuous state space
- We use a neural network as a black box to replace the table
 - Input a state and an action, output the Q-value





DQN in Atari

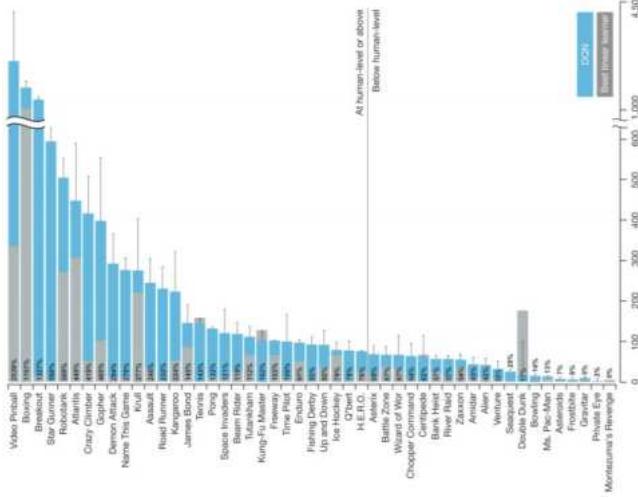
- Input state s is stack of raw pixels from last 4 frames
- Output is $q(s,a)$ for 18 button
- Reward is change in score for that step





DQN in Atari (cont'd)

- Pong's video
- <https://www.youtube.com/watch?v=PSQt5KGv7Vk>
- Beat human on many games



CZ3005

Artificial Intelligence

Game Theory

Asst/P Hanwang Zhang

<https://personal.ntu.edu.sg/hanwangzhang/>

Email: hanwangzhang@ntu.edu.sg

Office: N4-02c-87



What is Game Theory?



- Game theory studies settings where multiple parties (**agents**) each have
 - different preferences (utility functions),
 - different actions that they can take
- Each agent's utility (potentially) depends on all agents' actions
 - What is optimal for one agent depends on what other agents do
 - Very circular!
- Game theory studies how agents can rationally form beliefs over what other agents will do, and (hence) how agents should act
 - Useful for acting as well as predicting behavior of others
- John von Neumann



What is Game Theory?



Economics



Politics



Games



Biology

Normal Form Games – An Example



- List of players, strategies, payoffs
- Simultaneous or not
- Zero-sum here but not necessary
- Analysis:
 - What should they do?
 - Advice to player A

Player A

Rock



Player B

Rock



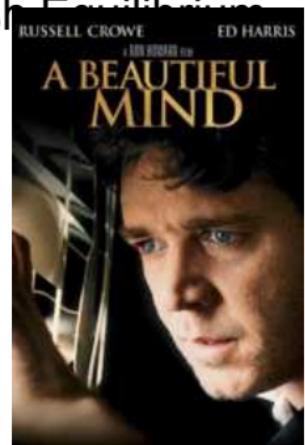
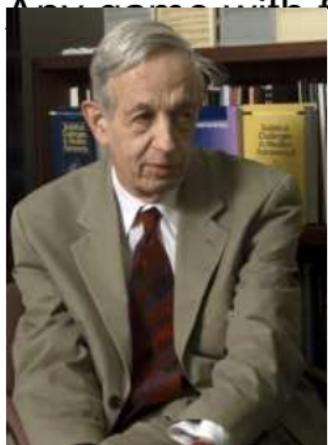
0, 0	1, -1	-1, 1
-1, 1	0, 0	1, -1
1, -1	-1, 1	0, 0



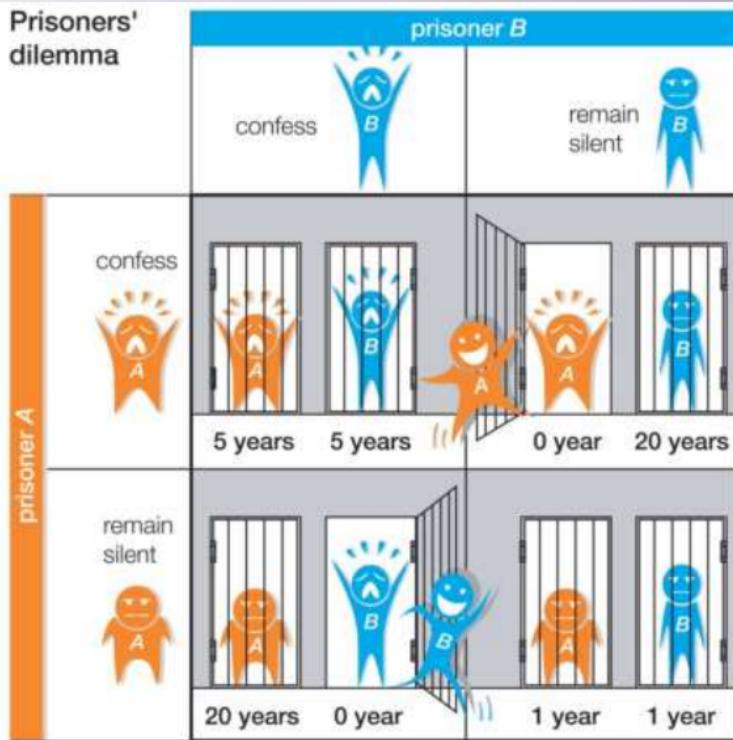
Nash Equilibrium



- Each agent is selfish
- Each agent makes decision based on what he thinks others would do
- No one can do better by changing strategy solely
- Any game with finite actions must have a Nash Equilibrium



Nash Equilibrium



Game Abstraction



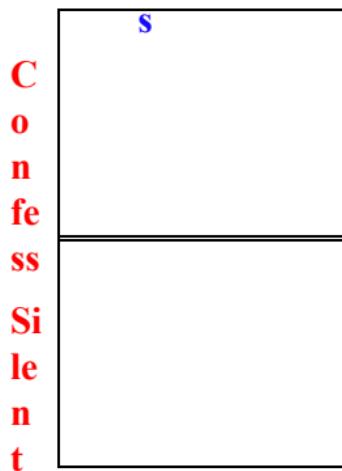
		Confes	Silen
		s	t
C o n fe ss	Confe		
	Sile		



Let's look at Red when Blue “confess”



Confes



Let's look at Red when Blue “Silent”



C
o
n
fe
ss
Si
le
n
t

Silen

t

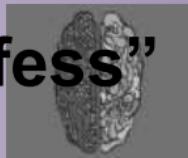


Let's look at Red: Silent dominates Confess



		Confes	Silen
		s	t
C o n fe	s		
	ss		
Si le n t	Si		
	nt		

Let's look at Blue when Red "confess"



	Confes	Silen
s		t
C o n fe ss		

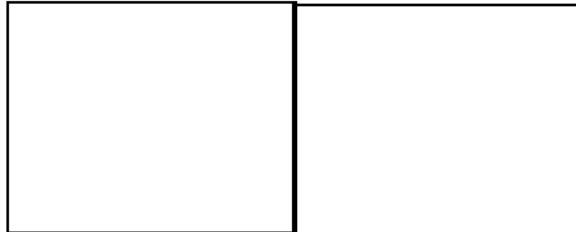
Let's look at Blue when Red “silent”



Confes
s

Silen
t

Si
le
n
t



Nash Equilibrium: both confess



		Confes	Silen
		s	t
C o n fe ss	Confe		
	Silen		

Cooperation would be better for both!

But, rational for both to defect!

Nash Equilibrium



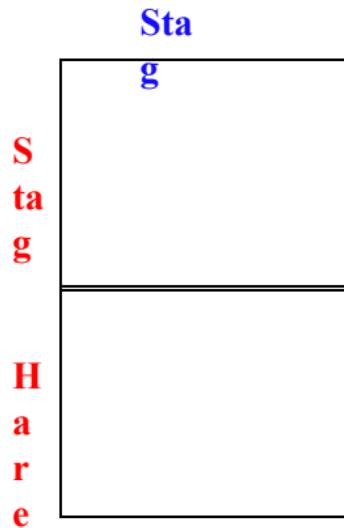
Game Abstraction



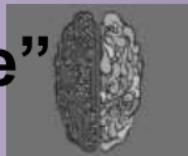
	Sta	Har
Stag	g	e
Hare		



Let's look at Red when Blue “Stag”



Let's look at Red when Blue “Hare”



S
ta
g

H
a
r
e

Har

e



Let's look at Red: Two Strategies



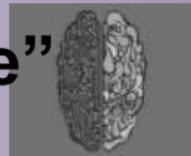
	Sta	Har
Stag	g	e
Hare		

Let's look at Blue when Red “Stag”



	Sta	Har
S	g	e
ta		
g		

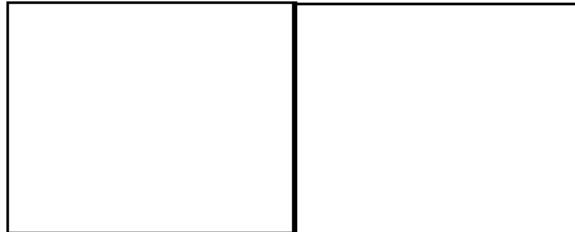
Let's look at Blue when Red “Hare”



Sta
g

Har
e

H
a
r
e



Let's look at Blue: Two Strategies



	Sta	Har
Stag	g	e
Hare		



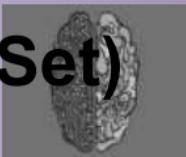
Nash Equilibrium: A Strategy Set



	Sta	Har
Strag	g	e
Hare		



Nash Equilibrium: Pure Strategy (Set)

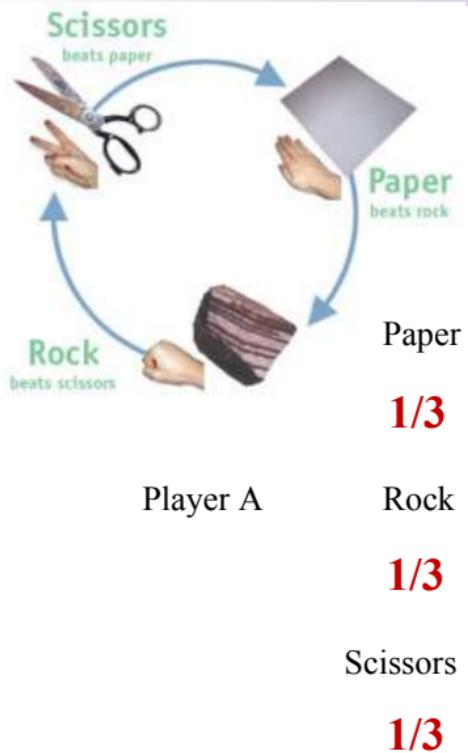


		Confes	Silen
C o n fe ss	s		t
Si le n t			

		Sta	Har
S ta g	g		e
H a r e			



Nash Equilibrium: Mixed Strategy (Set)



Player B

	Paper	Rock	Scissors
Paper	1/3	1/3	1/3
Rock	1/3	1/3	1/3
Scissors	1/3	1/3	1/3

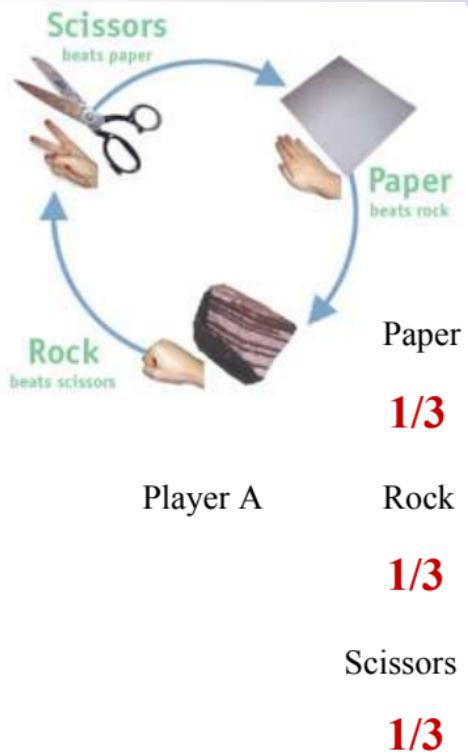
	0, 0	1, -1	-1, 1
Paper	0, 0	1, -1	-1, 1
Rock	-1, 1	0, 0	1, -1
Scissors	1, -1	-1, 1	0, 0

Look at Red when Blue “Paper”



	Paper	Roc k	Scissors
Paper	0, 0	1,-1	-1,1
Roc k	-1, 1	0, 0	1, -1
Scissors	1, -1	-1, 1	0, 0

Nash Equilibrium: Mixed Strategy (Set)



Player B

Paper	1/3	Rock	1/3	Scissors	1/3
-------	------------	------	------------	----------	------------

0, 0	1, -1	-1, 1
-1, 1	0, 0	1, -1
1, -1	-1, 1	0, 0

NE is a “natural law”



G	o	Sto	p
G o			
S to p			

Games: Complete Information v.s. Incomplete Information



Games with
complete information

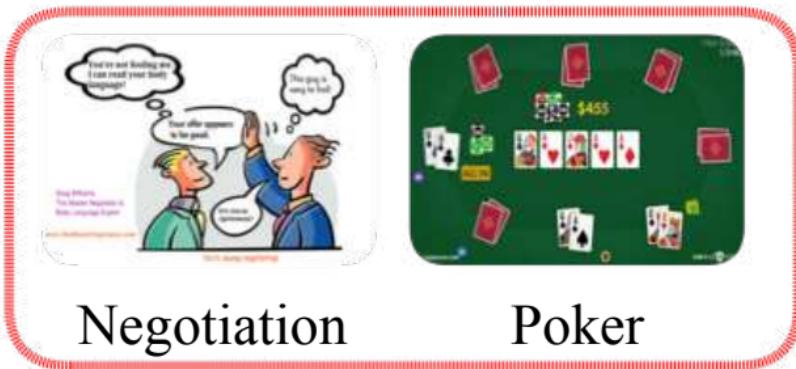


Chess



Go

Games with
incomplete information



Negotiation



Poker

Texas Hold'em Poker



- **Preflop:** two private cards are dealt to each player, followed by a **betting** round; players can either **check**, **bet** or **fold**
- **Flop:** three public cards are dealt, followed by a betting round
- **Turn:** a fourth public card is dealt, followed by a betting round
- **River:** a last public card is dealt, followed by a betting round
- The game ends when
 - Only one player is left, all the other players fold
 - A show hand: a hand with the best 5 cards using both the two private

Architecture of Libratus



Abstraction (offline)

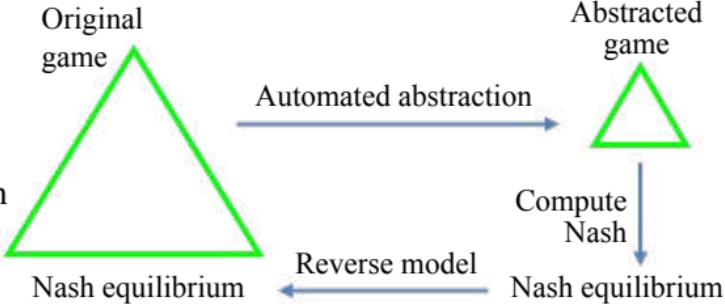
- action abstraction
- card abstraction
- took the game size from 10161 to 1012

Equilibrium Finding (offline)

- CFR
- CFR+
- Monte Carlo CFR

Decomposition and Subgame Refinement (online)

- endgame solving
- subgame re-solving
- max-margin subgame refinement



Global Challenges for Security



Key challenges: Limited resources, surveillance



Stackelberg Games

Randomization: Increase Cost and Uncertainty to

Attackers

Security allocation

- Target weights
- Opponent reaction
- Stackelberg: Security forces commit first
- Optimal allocation: Weighted random
 - Strong Stackelberg Equilibrium



Attacker
↓



Defender



	Target #1	Target #2
Target #1	4, -3	-1, 1
Target #2	-5, 5	2, -1





Game Theory for Security: Applications

Game Theory + Optimization + Uncertainty + Learning + ...

Infrastructure Security Games



Coast Guard



Coast Guard: Ferry



LAX



TSA



LA Sheriff



USC



Argentina Airport



Chile Border

Green Security Games



Coast Guard



Panthera/WWF



India

Opportunistic Crime Games



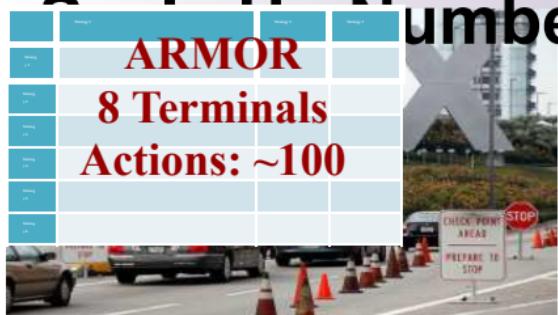
Cyber Security Games





IRIS: Federal Air Marshals Service [2009]

Challenge: Number of Definitions



- 1000 Flights, 20 air marshals: 1041 combinations
 - ARMOR out of memory
- Not enumerate all combinations
 - Branch and price
 - o Branch & bound + column generation



PROTECT: Randomized Patrol Scheduling [2011]



Coordination (Scale-up) and Ferries (Continuous Space/time)

The collage illustrates the PROTECT project's focus on coordination and ferries. The map shows the geographical context of the project, while the video frame shows a naval officer presenting. The diagram represents the temporal and spatial grid used for scheduling patrols. The bridge image represents the continuous space-time nature of the problem.

Port Liberte

St. George

STATEN ISLAND

Bayonne

Atlantic Highlands

Hillsides

RECORDED LIVE

RDML Butt

01:12:42 / 02:18:00

Infrastructure USHR12 Transport

t 3

t 4

t 5



PAWS: Protection Assistant for Wildlife Security Trials in Uganda and Malaysia [2014]

- Important lesson: Geography!



Uganda



Andrew Lemieux



Malaysia



Panthera



Malaysia



PAWS Deployed in 2015 in Southeast Asia (with Panthera and WWF)



PAWS Version 2: Features

- Street map
 - Ridgelines, riversstreams
- Species Distribution Models (SDMs)
 - From data points to distribution map



Indonesia



Malaysia





NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 8a – Logical Agent

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Second Half Lecturer



Nanyang Assistant Professor Yu Han

- Email: han.yu@ntu.edu.sg
- Office: N4-02c-109
- Website: <http://hanyu.sg/>

Course Overview (2nd Half)

Week	Date	Online (Zoom) https://ntu-sq.zoom.us/j/82750496869 Meeting ID: 827 5049 6869 Passcode: 864874	Topic	Tutorial	Lab
8	06/03/2023	1	Logical Agent (a)	Tutorial 3	
8	08/03/2023	2	Logical Agent (b)		
9	13/03/2023	3	Propositional Logic (a)		
9	15/03/2023	4	Propositional Logic (b)		
10	20/03/2023	5	First-Order Logic (a)		One Lab session in Week 10/11 (Lab Attendance is <u>NOT</u> Compulsory)
10	22/03/2023	6	First-Order Logic (b)		
11	27/03/2023	7	Default Logic (a)	Tutorial 4	
11	29/03/2023	8	Default Logic (b)		
12	03/04/2023	9	Fuzzy Logic (a)		
12	05/04/2023	10	Fuzzy Logic (b)		
13				Tutorial 5	
13					

Why Logic?

- The earliest way people imagined how to build AI system
- Still a relevant tool to guide today's AI with human knowledge
- Good interpretability (which deep learning-based AI lacks)

An Example Logical Agent

https://www.youtube.com/watch?v=MxI_tzOs6ME



Why Logic?

HA

are you built on computer logic?



Yes, as an AI language model, I am built on computer logic. AI models like me use mathematical algorithms and logic to analyze large amounts of data and learn patterns and relationships in that data. The underlying technology behind my operation is based on computer logic, which enables me to understand and generate human-like responses to questions and queries.



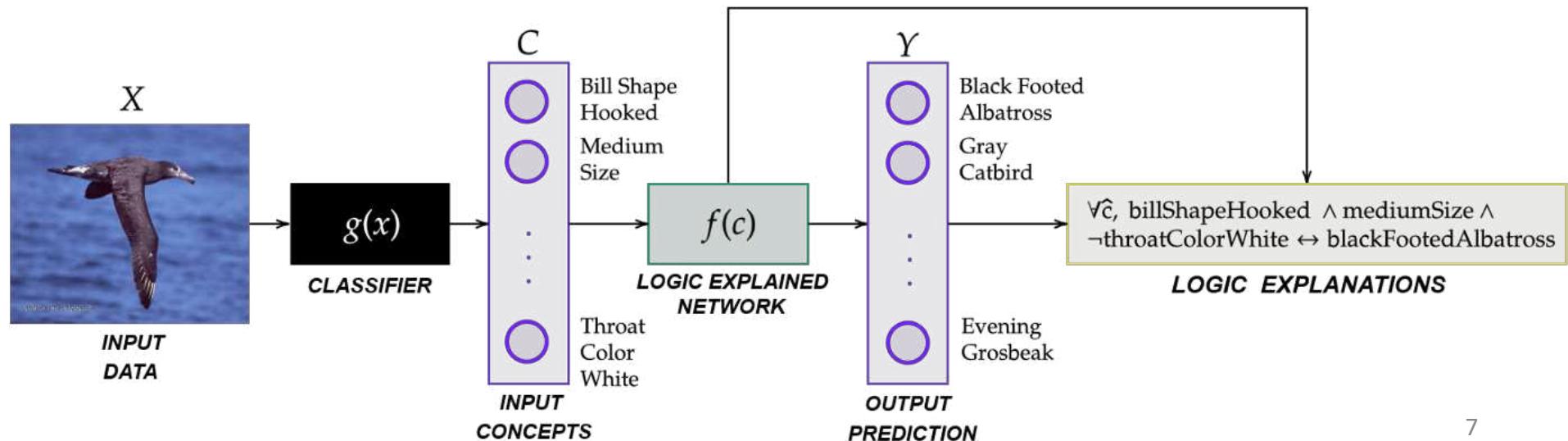
My architecture and design are based on sophisticated algorithms, including deep learning and natural language processing (NLP), which are built on top of the principles of logic and mathematics. These algorithms allow me to analyze and understand natural language, and generate responses that are contextually appropriate and relevant to the query posed to me.

In short, the technology that powers AI language models like me is based on computer logic and mathematical algorithms that enable us to understand and generate human-like responses.

Regenerate response

Why Logic?

- New life for an old technology:
 - Deep Learning and Logic Programming are starting to merge today for building Explainable AI (XAI) systems.
 - Logic Explained Networks:



From Search to Logical Reasoning

- Representation of Knowledge and the Reasoning Processes are central to the entire field of AI
- A knowledge-based system (KBS) is a computer program that reasons and uses a knowledge base to solve complex problems.

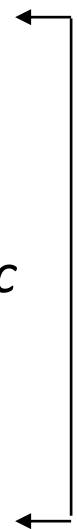
Knowledge and Intelligence



The Knowledge-Based Approach

- Agents that ***know***
 - Achieve competence by being told new knowledge or by learning
 - Achieve adaptability by updating their knowledge
 - > *Knowledge representation*
 - State of the world, properties and evolution of the world; goals of the agent, actions and their effect
- Agents that ***reason***
 - Use knowledge to deduce course of actions
 - > *Knowledge inference*

Logic



Knowledge-Based Agents

- **Knowledge base (KB)**
 - Set of sentences i.e., representations of facts (DB)
 - Knowledge representation language
- Adding and querying knowledge
 - **Tell**: add a sentence to the KB
 - **Ask**: retrieve knowledge from the KB
 - Answers must *follow* from what has been **Tell**'ed (told)
- Inference mechanism
 - Determine what follows from the KB

Problem Formulation of KBS

- Knowledge Based System
 - States: Instances of the KB (sets of sentences)
 - Use **Tell** to build the KB
 - e.g. **Tell(KB, “Smoke \Rightarrow Fire”)**
 - Tell(KB, “Fire \Rightarrow Call_999”)**
 - ...
 - Tell(KB, “Smoke”)**
 - Operators: Add / Infer a new sentence
 - Goal: Answer a query
 - Use **Ask** to query the KB
 - e.g. **Ask(KB, “? Call_999”)**

A Generic Knowledge-Based Agent

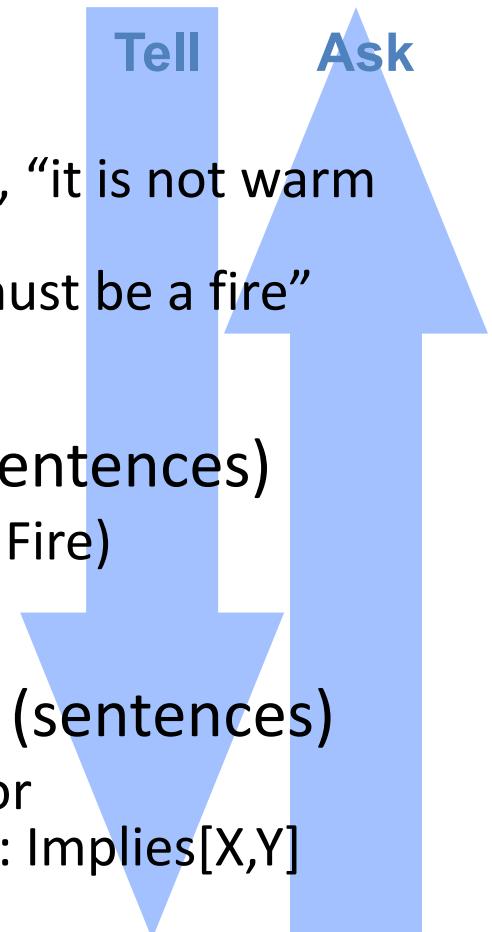
```
function KB-Agent (percept) returns action
    static KB,           // a knowledge base
    t                  // a time counter, initially 0

    Tell (KB, Make-Percept-Sentence (percept, t))
    action  $\leftarrow$  Ask (KB, Make-Action-Query (percept, t))
    Tell (KB, Make-Action-Sentence (action, t))
    t  $\leftarrow$  t + 1
    return action
```

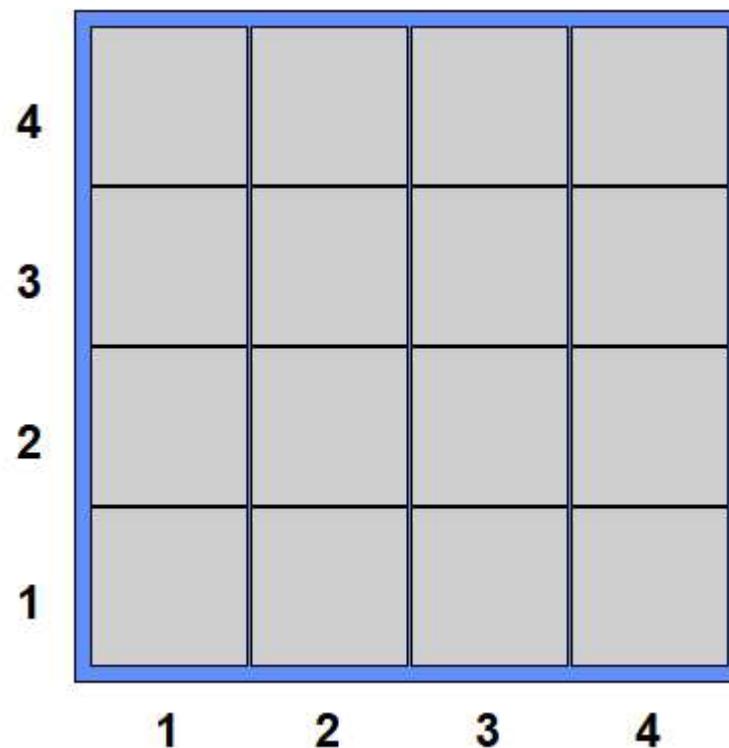
- > 3 steps: interpretation, inference, execution
- > KB: background knowledge (observed)
+ acquired information (deduced)

Levels of Knowledge

- **Epistemological level**
 - Declarative description of knowledge
 - e.g. facts: “there is smoke in the kitchen”, “it is not warm enough”
rules: “if there is smoke then there must be a fire”
- **Logical level**
 - Logical encoding of knowledge (into sentences)
 - e.g. facts: Smoke; rules: Implies(Smoke, Fire)
- **Implementation level**
 - Physical representation of knowledge (sentences)
 - e.g. - the string “Implies(Smoke, Fire)”, or
- a “1” entry in a 2-dimensional array: Implies[X,Y]

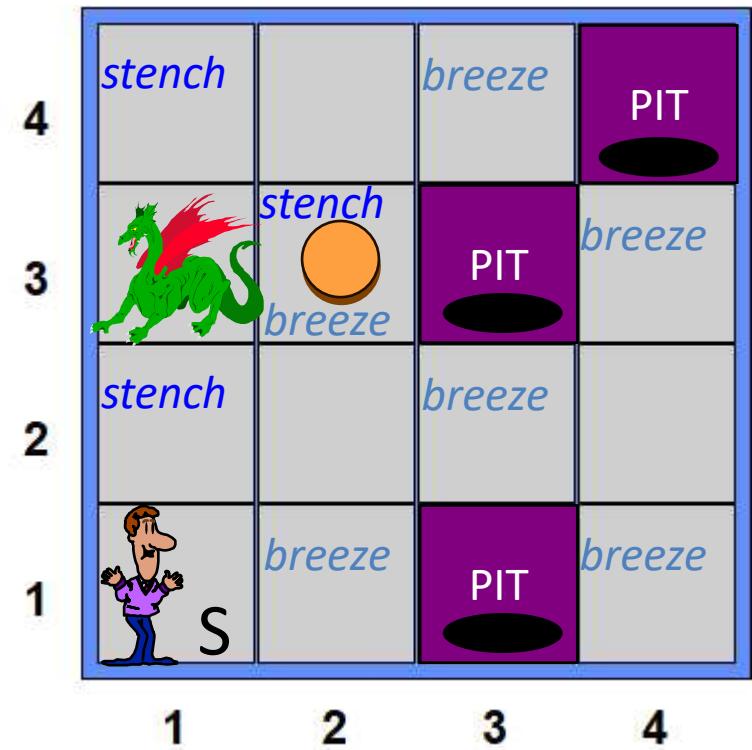


Example: the Wumpus World



Example: the Wumpus World

- Problem description (PAGE)
 - Environment
 - Grid of squares, walls;
 - Agent, gold, pits, wumpus.
 - Goal
 - Find the gold, return to S at [1,1].
 - Percepts
 - A list of 5 symbols, e.g. [Stench, Breeze, Glitter, Bump, Scream];
 - Agent's location *not* perceived.
 - Actions
 - Go-Forward, Turn-Left, Turn-Right, Grab, Shoot (1 arrow only), Climb.



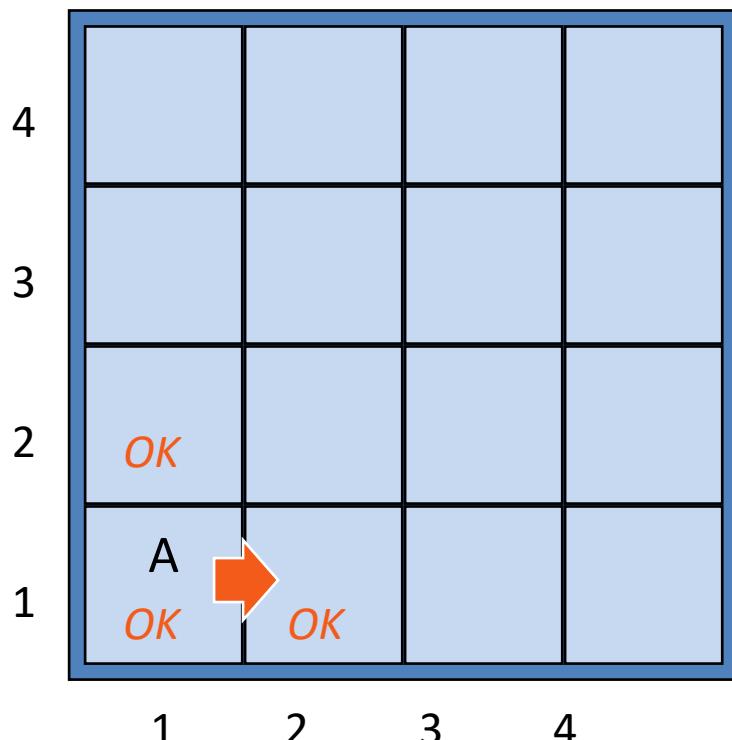
The Wumpus World

- Problem description (cont'd)
 - Initial state
 - Agent at [1,1]; gold, pits and wumpus in random squares.
 - Path-cost
 - Climbing out with the gold: +1000 (without: 0) • Each action: -1
 - Getting killed (pit or wumpus): -10000
 - Knowledge
 - “In all squares adjacent to the one where the wumpus is, the agent will perceive a stench.”
 - “In all squares adjacent to a pit, the agent will perceive a breeze.”
 - In the square where the gold is, the agent will perceive a glitter.”
 - When walking into a wall, the agent will perceive a bump.”
 - When the wumpus is killed, the agent will perceive a scream.”

Acting and Reasoning in the Wumpus World

(0) Initial state

[nil, nil, nil, nil, nil]



A = Agent

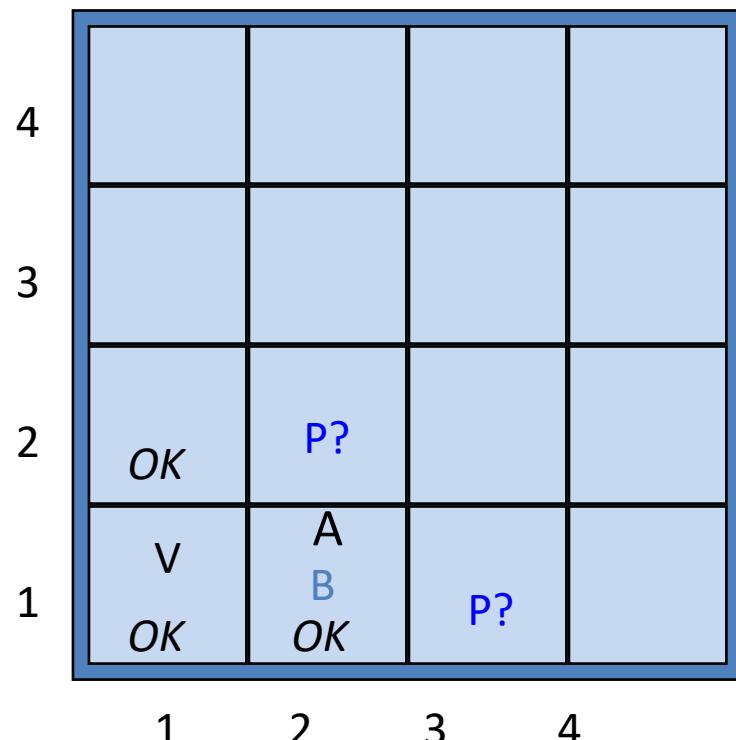
B = Breeze

G = Glitter, Gold

OK = Safe square

(1) after {F}

[nil, Breeze, nil, nil, nil]



P = Pit

S = Stench

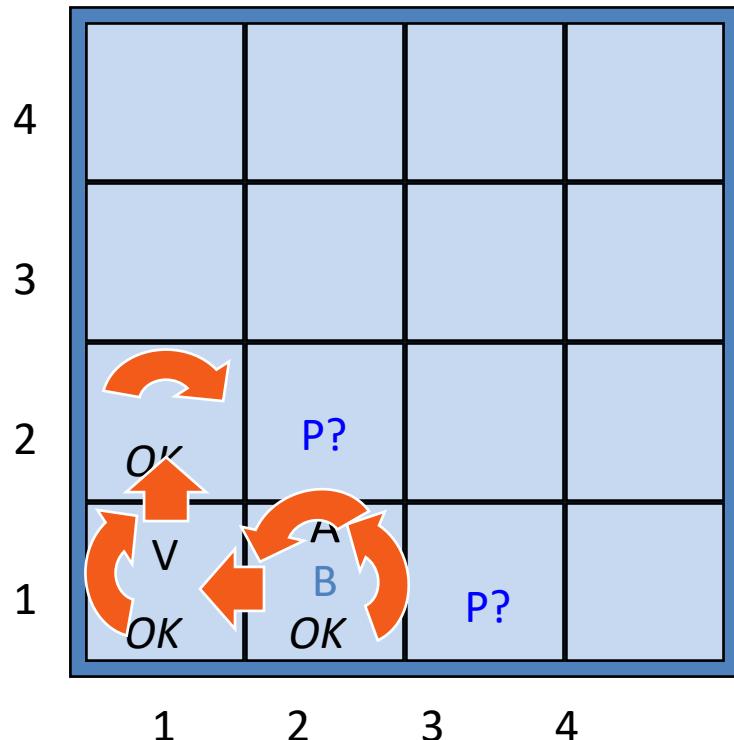
V = Visited

W = Wumpus

Acting and Reasoning in the Wumpus World

(1) after {F}

[nil, Breeze, nil, nil, nil]



A = Agent

B = Breeze

G = Glitter, Gold

OK = Safe square

P = Pit

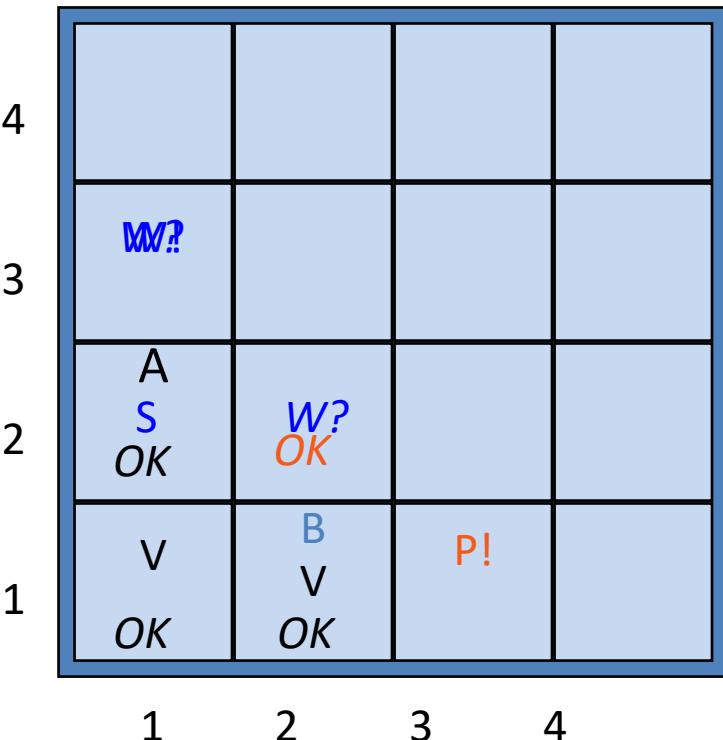
S = Stench

V = Visited

W = Wumpus

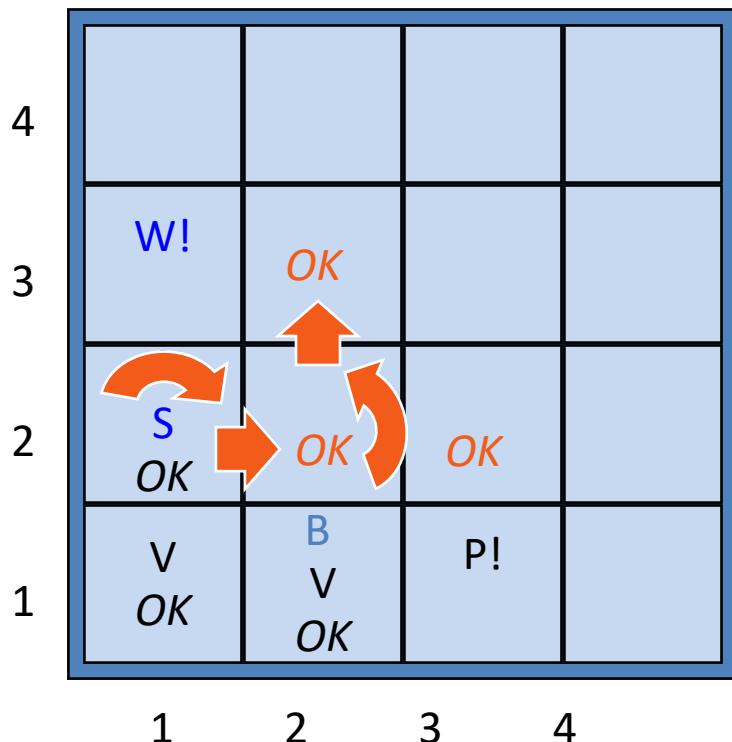
(6) after {F, L, L, F, R, F}

[Stench , nil, nil, nil, nil]



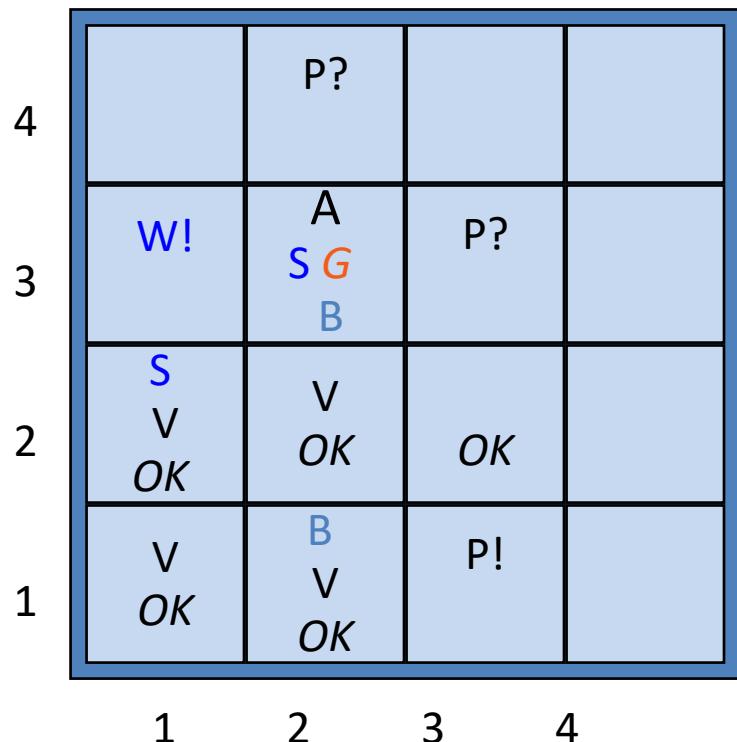
Acting and Reasoning in the Wumpus World

(6) after {F, L, L, F, R, F}
 [Stench , nil, nil, nil, nil]



A = Agent
 G = Glitter, Gold
 B = Breeze
 OK = Safe square

(10) after {F, L, L, F, R, F ,R, F, L, F}
 [Stench, Breeze, Glitter, nil, nil]



P = Pit
 S = Stench
 V = Visited
 W = Wumpus

Summary

- **Intelligent agents need ...**
 - Knowledge about the world to make good decisions.
- **Knowledge can be ...**
 - Defined using a knowledge representation language.
 - Stored in a knowledge base in the form of sentences.
 - Inferred, using an inference mechanism and rules.



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 8b – Logical Agent

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Recap

- **Intelligent agents need ...**
 - Knowledge about the world to make good decisions.
- **Knowledge can be ...**
 - Defined using a knowledge representation language.
 - Stored in a knowledge base in the form of sentences.
 - Inferred, using an inference mechanism and rules.

Recap

```
function KB-Agent (percept) returns action
    static KB,           // a knowledge base
    t                  // a time counter, initially 0

    Tell (KB, Make-Percept-Sentence (percept, t))
    action  $\leftarrow$  Ask (KB, Make-Action-Query (percept, t))
    Tell (KB, Make-Action-Sentence (action, t))
    t  $\leftarrow$  t + 1
    return action
```

- > 3 steps: interpretation, inference, execution
- > KB: background knowledge (observed)
+ acquired information (deduced)

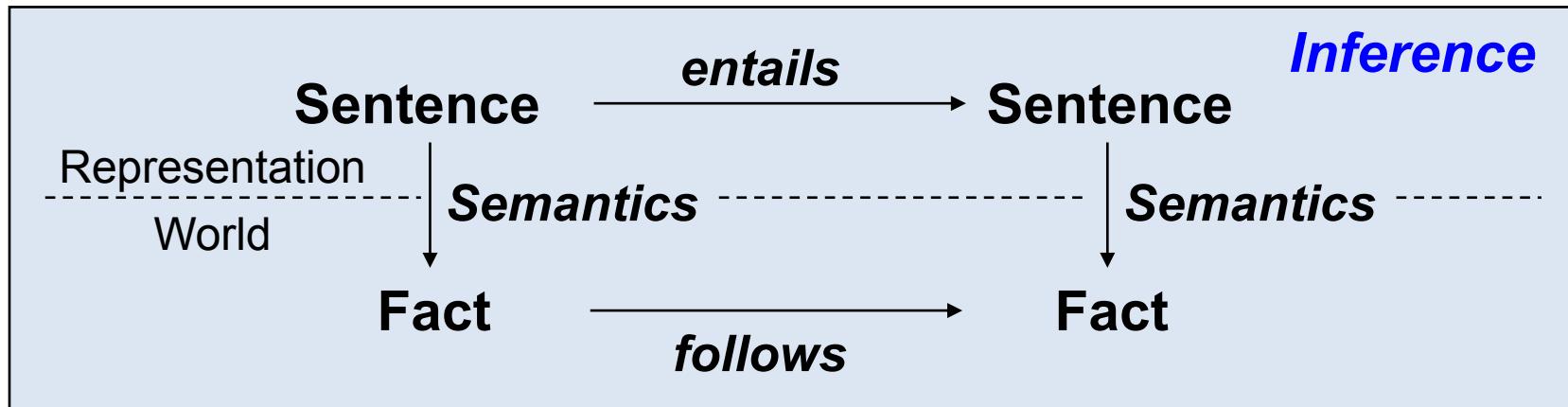
Knowledge Representations

- **Knowledge representation (KR)**
 - KB: set of sentences → need to
 - Express knowledge in a (computer-) tractable form
- **Knowledge representation language**
 - Syntax – implementation level
 - Possible configurations that constitute sentences
 - Semantics – knowledge level
 - Facts of the world the sentences refer to
 - e.g. language of arithmetics: x, y numbers
sentence: “ $x \geq y$ ”, semantics: “greater or equal”



Reasoning and Logic

- **Logic**
 - Representation + Inference = Logic
 - Where representation = syntax + semantics
- **Reasoning**
 - Construction of new sentences from existing ones
- **Entailment** as logical inference
 - the relationship between sentences whereby one sentence will be true if all the others are also true



Deduction and Induction

- **Mechanical reasoning**
 - Example
 - If a chord sequence is tonal, then it can be generated by a context-sensitive grammar.
 - The twelve-bar blues has a chord sequence that is tonal. |-
 - The twelve-bar blues has a chord sequence that can be generated by a context-sensitive grammar.
- **Deductive inference**
 - KB: Monday \Rightarrow Work, Monday |- Work sound
- **Inductive inference**
 - KB: Monday \Rightarrow Work, Work |- Monday unsound!
 - Generalization e.g., “all swans are white ...”

Deduction and Induction



Entailment and Inference

- **Entailment**

- Generate sentences that are necessarily true, given that the existing sentences are true
- Notation: $\text{KB} \models \alpha$

- e.g. Wumpus world:

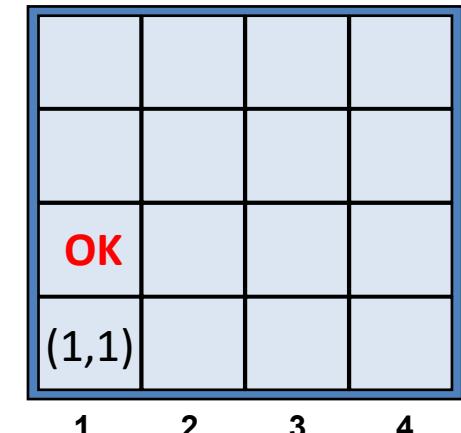
$\{ \neg S(1,1), \neg B(1,1) \} \models \text{``OK}(2,1)''$

Arithmetics:

$\{ x \geq y, y \geq z \} \models \text{``}x \geq z\text{''}$

- **Inference**

- The act or process of **deriving logical conclusions from premises known or assumed to be true.**
- **Tell**, given KB: $(\text{KB} \models \alpha) !$
- **Ask**, given KB and α : $(\text{KB} \models \alpha) ?$



Properties of Inference

- *Can be described by the sentences it derives, $KB \models \alpha_I$*
- **Soundness**
 - Generate only **entailed** sentences
 - Proof: sequence of operations of a sound inference
 - Record of operations that generate a specific entailed sentence
 - e.g. “Smoke \Rightarrow Fire” and “Smoke” \models “Fire”
 - “Fire \Rightarrow Call_911” and “Fire” \models “Call_911”
- **Completeness**
 - A proof can be found **for any** entailed sentence

An Example of Sound Inference

- Sentence: x
 - Semantics: an expression; can be a single symbol or number, the concatenation of 2 expressions, etc.
- Sentence: $x y$
 - Semantics: an expression which refers to a quantity that is the product of the quantities referred to by each of the expressions
- Sentence: $x = y$
 - Semantics: the 2 expressions on each side of “=” refer to the same quantity
- A sound inference: from $E = mc^2$
$$T_1 \geq T_2 \models E T_1 \geq mc^2 T_2$$

Is this a Sound Inference?

- Sentence: x
 - Semantics: an expression; can be a single symbol or number, the concatenation of 2 expressions, etc.
- Sentence: $x y$
 - Semantics: an expression which refers to a quantity that is the product of the quantities referred to by each of the expressions
- Sentence: $x = y$
 - Semantics: the 2 expressions on each side of “=” refer to the same quantity
- A sound inference? from $E = mc^2$
$$T_1 > T_2 \models E T_1 \geq mc^2 T_2$$

Knowledge Representation Languages

- **Formal (programming) languages**
 - Good at describing algorithms and data structures
 - e.g. the Wumpus world as a 4x4 array, $\text{World}[2,2] \leftarrow \text{Pit}$
 - Poor at representing incomplete / uncertain information
 - e.g. “there is a pit in [2,2] or [3,1]”, or “...a wumpus *somewhere*”
 - \rightarrow not expressive enough
- **Natural languages**
 - Very expressive (too much, thus very complex)
 - More appropriate for communication than representation
 - Suffer from ambiguity
 - e.g. “It’s hot!”
 - e.g. “small cats and dogs” compared to “ $-x + y$ ”.

Properties of Representations

- *KR languages should combine the advantages of both programming and natural languages.*
- **Desired properties**
 - Expressive
 - Can represent everything we need to.
 - Concise
 - Unambiguous
 - Sentences have a unique interpretation.
 - Context independent
 - Interpretation of sentences depends on semantics only.
 - Effective
 - An inference procedure allows to create new sentences.

Properties of Semantics

- **Interpretation (meaning)**
 - *Correspondence between sentences and facts*
 - Arbitrary meaning, fixed by the writer of the sentence
 - e.g. Natural languages: meaning fixed by usage (cf. dictionary)
exceptions: encrypted messages, codes (e.g. Morse)

 - Systematic relationship: compositional languages
 - *The meaning of a sentence is a function of the meaning of its parts.*
 - Truth value
 - A sentence make a claim about the world → TRUE or FALSE
 - *Depends on the interpretation and the state of the world*
 - e.g. Wumpus world: S(1,2) true if means “Stench at [1,2]” and the world has a wumpus at either [1,3] or [2,2].

Properties of Inference

- **Definition**

- *Inference (reasoning) is the process by which conclusions are reached*
- Logical inference (deduction) is the process that implements entailment between sentences

- **Useful properties**

- Valid sentence (tautology)
 - iff TRUE under all possible interpretations in all possible worlds.
 - e.g. “S or $\neg S$ ” is valid, “S(2,1) or $\neg S(2,1)$ ”, etc.
- Satisfiable sentence
 - iff there is some interpretation in some world for which it is TRUE
 - e.g. “S and $\neg S$ ” is unsatisfiable

Inference and Agent Programs

- **Inference in computers**
 - Does not know the interpretation the agent is using for the sentences in the KB
 - Does not know about the world (actual facts)
 - Knows only what appears in the KB (sentences)
 - e.g. Wumpus world: doesn't know the meaning of "OK", what a wumpus or a pit is, etc. – can only see: $\text{KB} \models "[2,2] \text{ is OK}"$
 - > *Cannot reason informally*
 - does not matter, however, if $\text{KB} \models "[2,2] \text{ is OK}"$ is a valid sentence
- **Formal inference**
 - Can handle arbitrarily complex sentences, $\text{KB} \models P$

Different Logics

- **Formal logic**
 - Syntax
 - A set of rules for writing sentences
 - Semantics
 - A set of rules (constraints) for relating sentences to facts
 - Proof theory / inference procedure
 - A set of rules for deducing entailments of sentences
- **Propositional logic**
 - Symbols, representing propositions (facts)
 - Boolean connectives, combining symbols
 - e.g. “Hot” or “Hot and Humid”

Different Logics

- **First-order logic**
 - Objects and predicates, representing properties of and relations between objects
 - Variables, Boolean connectives and quantifiers
 - e.g. “Hot(x)”, “Hot(Air)” or “Hot(Air) and Humid(Air)”
- **Temporal logic**
 - World ordered by a set of time points (intervals)
- **Probabilistic and fuzzy logic**
 - Degrees of belief and truth in sentences
 - e.g. “Washington is a state” with belief degree 0.4, “a city” 0.6,
“Washington is a large city” with truth degree 0.6

Different Degrees of Truth

- *Q: Is there a tuna sandwich in the refrigerator?*
- *A: 0.5 !*
- **Probabilities**
 - There *is* or there *isn't* (50% chance either way).
- **Measures**
 - There is *half* a tuna sandwich there.
- **Fuzzy answer**
 - There is *something* there, but it *isn't really* a tuna sandwich. Perhaps it is some other kind of sandwich, or a tuna salad with no bread...

The Commitments of Logics

Formal (KR) Language	Ontological commitment (what exists in the world)	Epistemological commitment (what an agent believes about facts)
Propositional logic	facts	true / false / unknown
First-order logic	facts, objects, relations	true / false / unknown
Temporal logic	facts, objects, rel., times	true / false / unknown
Probability logic	facts	degree of belief 0...1
Fuzzy logic	degrees of truth 0...1	degree of belief 0...1

Summary

- **A representation language is defined by ...**
 - A syntax, which specify the structure of sentences, and
 - A semantics, which specifies how the sentences relate to facts in the world.
- **Inference is ...**
 - The process of deducing new sentences from old ones.
 - Sound if it derives true conclusions from true premises.
 - Complete if it can derive all possible true conclusions.
- **Logics ...**
 - Make different commitments about what the world is made of and what kind of beliefs we can have about facts.



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 9a – Propositional Logic

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Logic is a Formal Language

- **Propositions:**
 - Anil is Intelligent
 - Anil is hardworking
 - If Anil is Intelligent and Anil is Hardworking, then Anil scores a high mark

Propositional Logic

- **Syntax** of the representation language specifies all the sentences that are well-formed.
- **Semantics** of the language defines the truth of each sentence with respect to each possible world.

Elements of Propositional Logic

- **Symbols**

- Logical constants: TRUE, FALSE
- Propositional symbols: P, Q, etc. (uppercase)
- Logical connectives: \wedge , \vee , \Leftrightarrow , \Rightarrow , \neg
- Parentheses: ()

- **Sentences**

- Atomic sentences: constants, propositional symbols
- Combined with connectives, e.g. $P \wedge Q \vee R$
also wrapped in parentheses, e.g. $(P \wedge Q) \vee R$

Elements of Propositional Logic

- Anil is intelligent = Intelligent(Anil)
- Anil is hardworking = Hardworking(Anil)
- Objects and relations or Functions

Anil hardworking, intelligent

- A proposition (statement) can be true or false

} Propositions

Logical Connectives

- **Conjunction** Λ
 - Binary op., e.g. $P \Lambda Q$, “P and Q”, where P, Q are the *conjuncts*
- **Disjunction** \vee
 - Binary op., e.g. $P \vee Q$, “P or Q”, where P, Q are the *disjuncts*
- **Implication** \Rightarrow
 - Binary op., e.g. $P \Rightarrow Q$, “P implies Q”, where P is the *premise* (antecedent) and Q the *conclusion* (consequent)
 - Conditionals, “if-then” statements, or rules
- **Equivalence** \Leftrightarrow
 - Binary op., e.g. $P \Leftrightarrow Q$, “P equivalent to Q”
 - Biconditionals
- **Negation** \neg
 - Unary op., e.g. $\neg P$, “not P”

Syntax of Propositional Logic

(Backus-Naur Form)

Sentence	→	AtomicSentence ComplexSentence
AtomicSentence	→	LogicalConstant PropositionalSymbol
ComplexSentence	→	(Sentence) Sentence LogicalConnective Sentence \neg Sentence
LogicalConstant	→	TRUE FALSE
PropositionalSymbol	→	P Q R ...
LogicalConnective	→	Λ \vee \Leftrightarrow \Rightarrow \neg

Precedence (from highest to lowest): \neg , Λ , \vee , \Rightarrow , \Leftrightarrow

e.g.: $\neg P \Lambda Q \vee R \Rightarrow S$ (not ambiguous), equal to: $((\neg P) \Lambda Q) \vee R \Rightarrow S$

Semantics of Propositional Logic

- **Validity**
 - A sentence is valid if it is true in all models.
 - Valid sentences are known as **tautologies**.
 - Every valid sentence is logically equivalent to *True*.

Semantics of Propositional Logic

- **Satisfiability**
 - A sentence is satisfiable if it is true in some models.
 - Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.
 - Most problems in computer sciences are satisfiability problems.
 - E.g., Constraint satisfaction problem, Search problems.

Semantics of Propositional Logic

- **Interpretation of symbols**
 - Logical constants have fixed meaning
 - True: always means the fact is the case; valid
 - False: always means the fact is not the case; unsatisfiable
 - Propositional symbols mean “whatever they mean”
 - e.g.: P “we are in a pit”, etc.
 - Satisfiable, but not valid (true only when the fact is the case)
- **Interpretation of sentences**
 - Meaning derived from the meaning of its parts
 - Sentence as a combination of sentences using connectives
 - Logical connectives as (boolean) functions:

TruthValue f (TruthValue, TruthValue)

Example 1

- Let P stands for Intelligent(Anil)
 - Let Q stands for Hardworking(Anil)
-
- What does $P \wedge Q$ mean?
 - What does $P \vee Q$ mean?
-
- $P \wedge Q$, $P \vee Q$ are compound propositions

Example 2

- Use parenthesis to ensure that the syntax is completely unambiguous:
 - A: John likes Kate.
 - B: John likes Chocolate.
 - C: John buys Chocolate
- $(A \wedge B) \Rightarrow C$
 - If John likes Kate and John likes Chocolate, John buys Chocolate
- $A \wedge (B \Rightarrow C)$
 - John likes Kate, and
 - If John likes Chocolate, then John buys Chocolate

Semantics of Propositional Logic

- **Interpretation of connectives**

- **Truth-table**

- Define a mapping from input to output

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

- Interpretation of sentences by decomposition

- e.g.: $\neg P \wedge Q \vee R \Rightarrow S$, with $P \leftarrow F, Q \leftarrow T, R \leftarrow F, S \leftarrow T$:

$$\neg P \leftarrow T$$

$$((\neg P) \wedge Q) \vee R \leftarrow T$$

$$(\neg P) \wedge Q \leftarrow T$$

$$(((\neg P) \wedge Q) \vee R) \Rightarrow S \leftarrow T$$

Validity and Inference

- Testing for validity
 - Using truth-tables, checking all possible configurations
 - e.g.: $((P \vee Q) \wedge \neg Q) \Rightarrow P$

P	Q	$P \vee Q$	$\neg Q$	$(P \vee Q) \wedge \neg Q$	$((P \vee Q) \wedge \neg Q) \Rightarrow P$
False	False	False	True	False	True
False	True	True	False	False	True
True	False	True	True	True	True
True	True	True	False	False	True

- The proposition says:
 - If $((P \vee Q) \wedge \neg Q)$ is True, then P is True.
 - If $((P \vee Q) \wedge \neg Q)$ is False, then ? (didn't specify, so P can be either True or False) -> overall, this proposition is *valid*

Summary

- **Valid** sentence – TRUE under all interpretations
- **Satisfiable** sentence – TRUE under at least 1 interpretation
- **Unsatisfiable** sentence – FALSE under all interpretations

Exercise

A	B	C	$A \wedge B$	$B \Rightarrow C$	$(A \wedge B) \Rightarrow C$	$A \wedge (B \Rightarrow C)$
T	T	T				
T	T	F				
T	F	T				
T	F	F				
F	T	T				
F	T	F				
F	F	T				
F	F	F				



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 9b – Propositional Logic

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Recap

Sentence	→	<u>AtomicSentence</u> <u>ComplexSentence</u>
AtomicSentence	→	<u>LogicalConstant</u> <u>PropositionalSymbol</u>
ComplexSentence	→	(Sentence) Sentence <u>LogicalConnective Sentence</u> \neg Sentence
LogicalConstant	→	TRUE FALSE
PropositionalSymbol	→	P Q R ...
LogicalConnective	→	Λ \vee \Leftrightarrow \Rightarrow \neg

Precedence (from highest to lowest): \neg , Λ , \vee , \Rightarrow , \Leftrightarrow

e.g.: $\neg P \Lambda Q \vee R \Rightarrow S$ (not ambiguous), equal to: $((\neg P) \Lambda Q) \vee R \Rightarrow S$

Recap

A	B	C	$A \wedge B$	$B \Rightarrow C$	$(A \wedge B) \Rightarrow C$	$A \wedge (B \Rightarrow C)$
T	T	T				
T	T	F				
T	F	T				
T	F	F				
F	T	T				
F	T	F				
F	F	T				
F	F	F				

Recap

A	B	C	$A \wedge B$	$B \Rightarrow C$	$(A \wedge B) \Rightarrow C$	$A \wedge (B \Rightarrow C)$
T	T	T	T	T	T	T
T	T	F	T	F	F	F
T	F	T	F	T	T	T
T	F	F	F	T	T	T
F	T	T	F	T	T	F
F	T	F	F	F	T	F
F	F	T	F	T	T	F
F	F	F	F	T	T	F

Literal and Clause

- **Literal:** A single proposition or its negation:
 - Example: $P, \neg P$
- A **clause:** A propositional formula formed from a finite collection of literals and logical connectives:
 - Example: $P \vee Q \vee \neg R$

Rules of Inference

- **Sound inference rules**
 - Pattern of inference, that occur again and again
 - Soundness proven once and for all (truth-table)
$$\frac{\alpha}{\beta}$$
- **Classic rules of inference**
 - Implication-Elimination, or *Modus Ponens (MP)*
 - $$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

e.g., Cloudy \wedge Humid \Rightarrow Rain |= Rain
Cloudy \wedge Humid

Rules of Inference

- **Classic rules of inference**

- And-Elimination

$$\bullet \frac{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}{\alpha_i}$$

e.g. Cloudy \wedge Humid \models Cloudy
Cloudy \Rightarrow NoSunTan

- And-Introduction

$$\bullet \frac{\alpha_1, \alpha_2, \dots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}$$

e.g. Cloudy, Humid
Cloudy \wedge Humid \Rightarrow Rain

- Or-Introduction

$$\bullet \frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \dots \vee \alpha_n}$$

- Double-Negation-Elimination

$$\bullet \frac{\neg \neg \alpha}{\alpha}$$

Rules of Inference

- **Resolution**
 - A technique of inference
 - Suppose x is a **literal** and S_1 and S_2 are two propositional sentences represented in the **clausal form**
- If $(x \vee S_1) \wedge (\neg x \vee S_2)$. Then, we get $(S_1 \vee S_2)$
 - Here, $(S_1 \vee S_2)$ is the **resolvent**,
 - x is **resolved upon**

Rules of Inference

- **The resolution rule of inference**
 - Unit Resolution
 - The **resolvent** is one in which at least one of the parent clauses is **a unit clause** (i.e., a single literal)
 - $$\frac{\alpha \vee \beta, \neg\beta}{\alpha}$$
- e.g., Monday \vee Tuesday, \neg Monday \models Tuesday

same as MP:
$$\frac{P \Rightarrow Q, P}{Q} \quad \text{i.e. } \frac{\neg\beta \Rightarrow \alpha, \neg\beta}{\alpha}$$

Rules of Inference

- **The resolution rule of inference**
 - Full Resolution

$$\bullet \quad \frac{\alpha \vee \beta, \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

Truth-table
for the
resolution

α	β	γ	$\alpha \vee \beta$	$\neg \beta \vee \gamma$	$\alpha \vee \gamma$
False	False	False	False	True	False
False	False	True	False	True	True
False	True	False	True	False	False
False	True	True	True	True	True
True	False	False	True	True	True
True	False	True	True	True	True
True	True	False	True	False	True
True	True	True	True	True	True

Equivalence Rules

- Equivalent notations

- e.g., MP:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta} \quad \left. \begin{array}{l} 1) \alpha \Rightarrow \beta, \alpha \dashv \beta \\ 2) \alpha \Rightarrow \beta, \alpha \vDash \beta \\ \\ 3) \frac{\alpha}{\beta} \\ \\ 4) ((\alpha \Rightarrow \beta) \wedge \alpha) \Rightarrow \beta \end{array} \right\}$$

Equivalence Rules

- **Equivalence rules**

- Associativity:

$$\alpha \wedge (\beta \wedge \gamma) \Leftrightarrow (\alpha \wedge \beta) \wedge \gamma$$

$$\alpha \vee (\beta \vee \gamma) \Leftrightarrow (\alpha \vee \beta) \vee \gamma$$

- Distributivity:

$$\alpha \wedge (\beta \vee \gamma) \Leftrightarrow (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$$

$$\alpha \vee (\beta \wedge \gamma) \Leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

- De Morgan's Law:

$$\neg(\alpha \vee \beta) \Leftrightarrow \neg\alpha \wedge \neg\beta$$

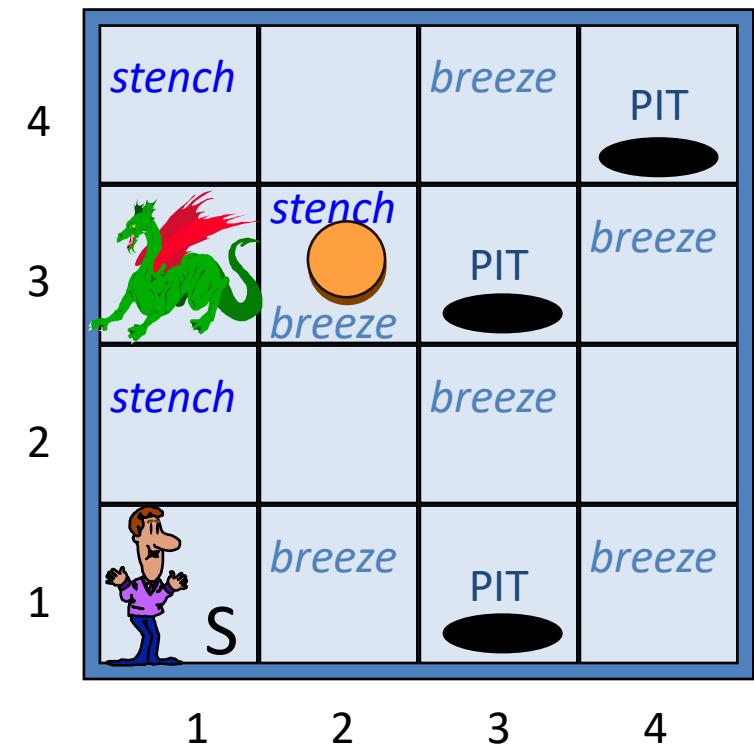
$$\neg(\alpha \wedge \beta) \Leftrightarrow \neg\alpha \vee \neg\beta$$

Complexity of Inference

- Proof by truth-table
 - Complete
 - The truth-table can always be written.
 - Exponential time complexity
 - A proof involving N proposition symbols requires 2^N rows.
 - In practice, a proof may refer only to a small subset of the KB.
- Monotonicity
 - Knowledge always increases
 - if $\text{KB}_1 \models \alpha$ then $(\text{KB}_1 \cup \text{KB}_2) \models \alpha$
 - Allows for local rules,
e.g., Modus Ponens $\alpha \Rightarrow \beta, \alpha \vdash \beta$

Relooking at the Wumpus World

- A reasoning agent
 - Propositional logic as the “programming language”
 - Knowledge base (KB) as problem representation
 - Percepts
 - Knowledge – sentences
 - Actions
 - Rule of inference (e.g., Modus Ponens) as the algorithm that will find a solution



The Knowledge Base

- **TELLing the KB: percepts**

- Syntax and semantics

- Symbol **S11**, meaning
“there is stench at [1,1]”

- Symbol **B12**, meaning
“there is breeze at [1,2]”

- ...

- **Percept sentences**

- Partial list:

- $\neg S_{11}$, $\neg B_{11}$, $\neg G_{11}$, ...
 - $\neg S_{21}$, B_{21} , $\neg G_{21}$, ...
 - S_{12} , $\neg B_{12}$, $\neg G_{12}$, ...

- ...

[Stench , nil, nil, nil, nil]

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

The Knowledge Base

- **TELLing the KB: knowledge**
 - Rules about the environment

- “All squares adjacent to the wumpus have stench.”

$$S12 \Rightarrow W11 \vee W12 \vee W22 \vee W13$$

- “A square with no stench has no wumpus and adjacent squares have no wumpus either.”

$$\neg S11 \Rightarrow \neg W11 \wedge \neg W21 \wedge \neg W12$$

$$\neg S21 \Rightarrow \neg W11 \wedge \neg W21 \wedge \neg W22 \wedge \neg W31$$

$$\neg S12 \Rightarrow \neg W11 \wedge \neg W12 \wedge \neg W22 \wedge \neg W13$$

[Stench , nil, nil, nil, nil]

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

Finding the Wumpus

- **Checking the truth-table**
 - Exhaustive check: every row for which KB is true also has W13 true
 - 12 propositional symbols, i.e.
 $S_{11}, S_{21}, S_{12}, W_{11}, W_{21}, W_{12}, W_{22}, W_{13}, W_{31}, B_{11}, B_{21}, B_{12}$
 - $2^{12} = 4,096$ rows
 - > *possible, but lengthy*
- **Reasoning by inference**
 - Application of a sequence of inference rules (proof)
 - Modus Ponens, And-Elimination, and Unit-Resolution

$KB \Rightarrow W_{13}$



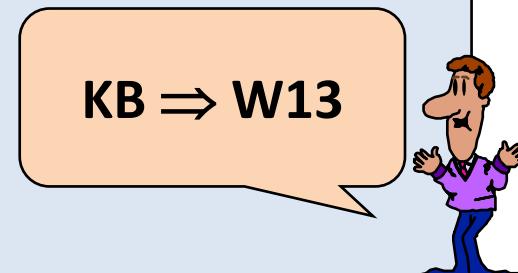
Proof for “ $\text{KB} \Rightarrow \text{W13}$ ”

Knowledge Base	Inferences
$\begin{array}{lll} \neg S11, & \neg B11, & \neg G11, \\ \neg S21, & B21, & \neg G21, \\ S12, & \neg B12, & \neg G12, \end{array}$ <p>R1: $\neg S11 \Rightarrow \neg W11 \wedge \neg W21$ $\qquad\qquad\qquad \wedge \neg W12$</p> <p>R2: $\neg S21 \Rightarrow \neg W11 \wedge \neg W21$ $\qquad\qquad\qquad \wedge \neg W22 \wedge \neg W31$</p> <p>R3: $\neg S12 \Rightarrow \neg W11 \wedge \neg W12$ $\qquad\qquad\qquad \wedge \neg W22 \wedge \neg W13$</p> <p>R4: $S12 \Rightarrow W11 \vee W12 \vee W22$ $\qquad\qquad\qquad \vee W13$</p>	<ol style="list-style-type: none"> 1. Modus Ponens: $\neg S11, \mathbf{R1}$ $- \neg W11 \wedge \neg W21 \wedge \neg W12$ 2. And-Elimination: \blacklozenge $- \neg W11, \neg W21, \neg W12$ 3. Modus Ponens: $\neg S21, \mathbf{R2}$ $- \neg W11 \wedge \neg W21 \wedge \neg W22$ $\qquad\qquad\qquad \wedge \neg W31$ 4. And-Elimination: \blacklozenge $- \neg W11, \neg W21, \neg W22, \neg W31$

Proof for “ $\text{KB} \Rightarrow \text{W13}$ ”

Knowledge Base	Inferences
$\neg S11, \quad \neg B11, \quad \neg G11,$ $\neg S21, \quad B21, \quad \neg G21,$ $S12, \quad \neg B12, \quad \neg G11,$ R1: $\neg S11 \Rightarrow \neg W11 \wedge \neg W21$ $\qquad\qquad\qquad \wedge \neg W12$ R2: $\neg S21 \Rightarrow \neg W11 \wedge \neg W21$ $\qquad\qquad\qquad \wedge \neg W22 \wedge \neg W31$ R3: $\neg S12 \Rightarrow \neg W11 \wedge \neg W12$ $\qquad\qquad\qquad \wedge \neg W22 \wedge \neg W13$ R4: $S12 \Rightarrow W11 \vee W12 \vee W22$ $\qquad\qquad\qquad \vee W13$	$\text{KB} += \neg W11, \neg W21, \neg W12,$ $\qquad\qquad\qquad \neg W22, \neg W31$ 5. Modus Ponens: $S12, \text{R4}$ $ - \quad (\textcolor{blue}{W13 \vee W12 \vee W22})$ $\qquad\qquad\qquad \vee W11$ 6. Unit-Resolution: $\blacklozenge, \neg W11$ $ - \quad (\textcolor{blue}{W13 \vee W12}) \vee W22$

Proof for “ $\text{KB} \Rightarrow \text{W13}$ ”

Knowledge Base	Inferences
$\neg S11, \quad \neg B11, \quad \neg G11,$ $\neg S21, \quad B21, \quad \neg G21,$ $S12, \quad \neg B12, \quad \neg G12,$ R1: $\neg S11 \Rightarrow \neg W11 \wedge \neg W21$ $\qquad\qquad\qquad \wedge \neg W12$ R2: $\neg S21 \Rightarrow \neg W11 \wedge \neg W21$ $\qquad\qquad\qquad \wedge \neg W22 \wedge \neg W31$ R3: $\neg S12 \Rightarrow \neg W11 \wedge \neg W12$ $\qquad\qquad\qquad \wedge \neg W22 \wedge \neg W13$ R4: $S12 \Rightarrow W11 \vee W12 \vee W22$ $\qquad\qquad\qquad \vee W13$	$\text{KB} += \neg W11, \neg W21, \neg W12,$ $\neg W22, \neg W31,$ $(W13 \vee W12) \vee W22$ 7. Unit-Resolution: $\blacklozenge, \neg W22$ $ - \quad \textcolor{blue}{W13 \vee W12}$ 8. Unit-Resolution: $\blacklozenge, \neg W12$ $ - \quad \textcolor{blue}{W13}$ 

From Knowledge to Actions

- **TELLing the KB: actions**

- Additional rules

- e.g. “if the wumpus is 1 square ahead then do not go forward”

$A12 \wedge \text{East} \wedge W22 \Rightarrow \neg \text{Forward}$

$A12 \wedge \text{North} \wedge W13 \Rightarrow \neg \text{Forward}$

...

- **ASKing the KB**

- Cannot ask “which action?”
but “should I go forward?”

[Stench , nil, nil, nil, nil]

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

A Knowledge-Based Agent Using Propositional Logic

```
function Propositional-KB-Agent (percept) returns action
    static   KB,           // a knowledge base
            t             // a time counter, initially 0

    Tell (KB, Make-Percept-Sentence (percept, t))
    foreach action in the list of possible actions
        do
            if Ask (KB, Make-Action-Query (t, action)) then
                Tell (KB, Make-Action-Sentence (action, t))
                t ← t + 1
            return action
    end
```

Limits of Propositional Logic

- **A weak logic**
 - Too many propositions to TELL the KB
 - e.g., the rule “if the wumpus is 1 square ahead then do not go forward” needs 64 sentences (16 squares x 4 orientations)!
 - Result in increased time complexity of inference
 - Handling change is difficult
 - Need time-dependent propositional symbols
 - e.g., A11 means “the agent is in square [1,1]” - when?
 - at t = 0: A11-0;
 - at t = 1: A21-1;
 - at t = 2: A11-2
 - Need to rewrite rules as time-dependent
 - e.g., $A12-0 \wedge \text{East-0} \wedge W22-0 \Rightarrow \neg \text{Forward-0}$
 - $A12-2 \wedge \text{East-2} \wedge W22-2 \Rightarrow \neg \text{Forward-2}$

Summary

- **Propositional logic ...**
 - Commits only to the existence of facts.
 - Has simple syntax and semantics and is therefore limited.



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 10a – First-Order Logic

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Recap

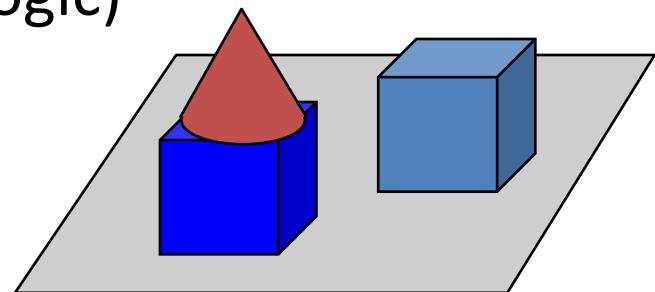
- A **sentence** is either True or False under an interpretation (or a world).
 - When it is True, we say that it is satisfiable (or is a model).
 - When it is False, then we say that it is unsatisfiable.
 - When a sentence is satisfiable under all interpretations (or worlds), then we say that it is a valid sentence or a tautology.
- Given that N objects are used in a **KB**, there will be 2^N possible interpretations (or worlds).

Recap

- **Soundness and Completeness**
 - Refer to an inference procedure
 - We do not say a sentence is sound or complete. An inference procedure is sound if we can derive True sentence from True sentences. That is, it implements entailment.
 - MP implements entailments
 - An inference procedure is complete if we can derive the proof of all entailed sentences (or valid sentences).

Representing Knowledge

- **Knowledge-based agent**
 - Have representations of the world in which they operate
 - Use those representations to infer what actions to take
- **Ontological commitments**
 - The world as facts (propositional logic)
 - The world as objects (first-order logic)
with properties *about* each object,
and relations *between* objects
 - e.g. the blocks world:
 - Objects: cubes, cylinders, cones, ...
Properties: shape, colour, location, ...
Relations: above, under, next-to, ...



First-Order Logic (FOL)

- **A very powerful KR scheme**
 - Essential representation of the world
 - Deal with objects, properties, and relations.
 - Simple, generic representation
 - Does not deal with specialized concepts such as categories, time, and events.
 - Universal language
 - Can express anything that can be programmed.
 - Most studied and best understood
 - More powerful proposals still debated.
 - Less powerful schemes too limited.

Propositional vs. First-Order Logic

- Aristotle's syllogism

- Socrates is a man. All men are mortal. Therefore Socrates is mortal.

Statement	Propositional Logic	First-Order Logic
“Socrates is a man.”	SocratesMan, S43	Man(Socrates), P52(S21)
“Plato is a man.”	PlatoMan, S157	Man(Plato), P52(S99)
“All men are mortal.”	MortalMan S421 Man \Rightarrow Mortal S9 \Rightarrow S4	$\text{Man}(x) \Rightarrow \text{Mortal}(x)$, P52(V1) \Rightarrow P66(V1)
“Socrates is mortal.”	MortalSocrates S957 S43 \wedge S421 – S957 ?!?	Mortal(Socrates) V1 \leftarrow S21, ... – P66(S21)

Syntax and Semantics of FOL

- **Sentences**
 - Built from quantifiers, predicate symbols, and terms
- **Terms**
 - Represent objects
 - Built from variables, constant and function symbols
- **Constant symbols**
 - Refer to (“name”) particular objects of the world
 - The object is specified by the interpretation
 - e.g. “John” is a constant , may refer to “John, king of England from 1199 to 1216 and younger brother of Richard Lionheart”, or my uncle, or ...

Syntax and Semantics of FOL

- **Variables**
 - Refer to any object of the world
 - e.g. x , person, ... as in $\text{Brother}(\text{KingJohn}, \text{person})$.
 - Can be substituted by a constant symbol
 - e.g. $\text{person} \leftarrow \text{Richard}$, yielding $\text{Brother}(\text{KingJohn}, \text{Richard})$.
- **Terms**
 - Logical expressions referring to objects
 - Include constant symbols (“names”) and variables.
 - Make use of function symbols.
e.g. $\text{LeftLegOf}(\text{KingJohn})$ to refer to his leg without naming it
 - Compositional interpretation
 - e.g. $\text{LeftLegOf}(), \text{KingJohn} \rightarrow \text{LeftLegOf}(\text{KingJohn})$.

Predicate and Function Symbols

- **Predicate symbols**
 - Refer to particular relations on objects
 - Binary relation specified by the interpretation
e.g. Brother(KingJohn, RichardLionheart) → T or F
 - A n -ary relation if defined by a set of n -tuples
 - Collection of objects arranged in a fixed order
e.g. {⟨KingJohn, RichardLionheart⟩, ⟨KingJohn, Henry⟩, ... }
- **Function symbols**
 - Refer to functional relations on objects
 - Many-to-one relation specified by the interpretation
e.g. BrotherOf(KingJohn) → a person, e.g. Richard (not T/F)
 - Defined by a set of $n+1$ -tuples
 - Last element is the function value for the first n elements.

Functions

- A function of arity n takes n objects of type W_1, \dots, W_n as inputs and returns an object of type W .
- Example:

– $\text{Plus}(3, 4) = 7$

The diagram illustrates the decomposition of a functional term. A red arrow points from the text "Functional Term" to the term $\text{Plus}(3, 4)$. Another red arrow points from the text "Object Terms" to the arguments 3 and 4 within the parentheses.

Predicates

- Predicates are like functions except that their return type is True or False.
- Example:
 - Greater-Than(3, 4)=False

Sentences in FOL

- **Atomic sentences**
 - State facts, using terms and predicate symbols
 - e.g. Brother(Richard, John).
 - Can have complex terms as arguments
 - e.g. Married(FatherOf(Richard), MotherOf(John)).
 - Have a truth value
 - Depends on both the interpretation and the world.
- **Complex sentences**
 - Combine sentences with connectives
 - e.g. Father(Henry, KingJohn) \wedge Mother(Mary, KingJohn)
 - Connectives identical to propositional logic
 - i.e.: \wedge , \vee , \Leftrightarrow , \Rightarrow , \neg

Sentence Equivalence

- ***There are many ways to write a logical statement in FOL***

– Example

- $A \Rightarrow B$ equivalent to $\neg A \vee B$
“rule form” “complementary cases”

 $\text{Dog}(x) \Rightarrow \text{Mammal}(x)$ $\neg \text{Dog}(x) \vee \text{Mammal}(x)$
“dogs are mammals” “either it’s not a dog or it’s a mammal”
- $A \wedge B \Rightarrow C$ equivalent to $A \Rightarrow (B \Rightarrow C)$
- Proof: $A \wedge B \Rightarrow C \Leftrightarrow \neg(A \wedge B) \vee C \Leftrightarrow (\neg A \vee \neg B) \vee C$
 $\Leftrightarrow \neg A \vee \neg B \vee C \Leftrightarrow \neg A \vee (\neg B \vee C)$
 $\neg P \vee Q \Leftrightarrow P \Rightarrow Q \Leftrightarrow \neg A \vee (B \Rightarrow C) \Leftrightarrow A \Rightarrow (B \Rightarrow C)$

Sentences in Normal Form

- *There is only one way to write a logical statement using a Normal Form of FOL*
 - Example $\neg B \Rightarrow \neg A$
 - $A \Rightarrow B$, $A \wedge B \Rightarrow C$ equivalent to $\neg A \vee B$, $\neg A \vee \neg B \vee C$
“Implicative Normal Form” “Conjunctive Normal Form”
- *Rewriting logical sentences allows to determine whether they are equivalent or not*
 - Example
 - $A \wedge B \Rightarrow C$ and $A \Rightarrow (B \Rightarrow C)$
both have the same CNF: $\neg A \vee \neg B \vee C$
- *Using FOL is the most convenient, but using a Normal Form is the most efficient*

Sentence Verification

- ***Rewriting logical sentences helps to understand their meaning***
 - Example
 - $\text{Owns}(x,y) \Rightarrow (\text{Dog}(y) \Rightarrow \text{AnimalLover}(x)) \quad A \Rightarrow (B \Rightarrow C)$
 - $\text{Owns}(x,y) \wedge \text{Dog}(y) \Rightarrow \text{AnimalLover}(x) \quad A \wedge B \Rightarrow C$
“A dog owner is an animal lover”
- ***Rewriting logical sentences helps to verify their meaning is as intended***
 - Example
 - “*Dogs all have the same enemies*”
 $\text{Dog}(x) \wedge \text{Enemy}(z, x) \Rightarrow (\text{Dog}(y) \Rightarrow \text{Enemy}(z, y)) \quad \text{same as}$
 $\text{Dog}(x) \wedge \text{Dog}(y) \wedge \text{Enemy}(z, x) \Rightarrow \text{Enemy}(z, y)$

Universal Quantifier \forall

- Express properties of collections of objects
 - Make a statement about *every* objects w/out enumerating
 - e.g. “All kings are mortal
 - King(Henry) \Rightarrow Mortal(Henry) \wedge
 - King(John) \Rightarrow Mortal(John) \wedge
 - King(Richard) \Rightarrow Mortal(Richard) \wedge
 - King(London) \Rightarrow Mortal(London) \wedge
 - ...
 - instead: $\forall x, \text{King}(x) \Rightarrow \text{Mortal}(x)$
 - Note: the semantics of the implication says $F \Rightarrow F$ is TRUE.
 - Thus, for those individuals that satisfy the premise $\text{King}(x)$, the rule asserts the conclusion $\text{Mortal}(x)$
 - But, for those individuals that do not satisfy the premise, the rule makes no assertion.

Using the Universal Quantifier

- The implication (\Rightarrow) is the natural connective to use with the universal quantifier (\forall)
 - Example
 - General form: $\forall x P(x) \Rightarrow Q(x)$
 - e.g. $\forall x \text{Dog}(x) \Rightarrow \text{Mammal}(x)$ “all dogs are mammals”
 - Use conjunction? $\forall x P(x) \wedge Q(x)$
 - e.g. $\forall x \text{Dog}(x) \wedge \text{Mammal}(x)$
 - same as $\forall x P(x)$ and $\forall x Q(x)$
 - e.g. $\forall x \text{Dog}(x)$ and $\forall x \text{Mammal}(x)$

All dogs are mammals. All mammals are dogs.

→ yields a very strong statement (too strong! i.e. *incorrect*)

Existential Quantifier \exists

- Express properties of some particular objects
 - Make a statement about one object without naming it
 - e.g., “King John has a brother who is king”
 - $\exists x, \text{Brother}(x, \text{KingJohn}) \wedge \text{King}(x)$
 - instead of
 - Brother(Henry, KingJohn) \wedge King(Henry) \vee
 - Brother(London, KingJohn) \wedge King(London) \vee
 - Brother(Richard, KingJohn) \wedge King(Richard) \vee
 - ...

Using the Existential Quantifier

- The conjunction (Λ) is the natural connective to use with the existential quantifier (\exists)
 - Example
 - General form: $\exists x P(x) \Lambda Q(x)$
 - e.g., $\exists x \text{Dog}(x) \Lambda \text{Owns}(\text{John}, x)$, “John owns a dog”
 - Use Implication? $\exists x P(x) \Rightarrow Q(x)$
 - e.g., $\exists x \text{Dog}(x) \Rightarrow \text{Owns}(\text{John}, x)$
 - Could be true for all x such that $P(x)$ is false
 - e.g., $\text{Dog}(\text{Garfield_the_cat}) \Rightarrow \text{Owns}(\text{John}, \text{Garfield_the_cat})$
 - yields a very weak statement (too weak! i.e. *useless*)





NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 10b – First-Order Logic

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Recap

- **First-Order Logic:**
 - Allows descriptions of relations and properties.
 - Allows the descriptions of relations and properties in a compounded manner.
 - Permits the use of constant and variables.
 - Allows general statements to be made.
 - Separation of inference from the representation of knowledge.

Recap

- **Normal Forms**
 - $P \Rightarrow Q$ can be rewritten as $\neg P \vee Q$
- **Auto documentation with predicate naming**
 - Reading from left to right
 - e.g., grandfather(Philip, William)
- **For all** quantifier (universal for all models) is used with implication connective
 - \forall with \Rightarrow
- **Existential** quantifier (satisfiable for at least one model) is used with and connective
 - \exists with \wedge

Recap

Generation of complex sentences

Composition via

Connectives – expressiveness

i.e. complex relations

Generalization – universal statement

– existential statement

Nesting and Mixing Quantifiers

- **Combining \forall and \exists**
 - Express more complex sentences
 - e.g., “if x is the parent of y , then y is the child of x ”:
 - $\forall x, \forall y \text{ Parent}(x, y) \Rightarrow \text{Child}(y, x)$
 - “everyone has a parent”: $\forall x, \exists y \text{ Parent}(y, x)$
 - Semantics depends on quantifiers ordering
 - e.g., $\exists y, \forall x \text{ Parent}(y, x)$
 - “there is someone who is everybody’s parent” ?
- **Well-formed formula (WFF)**
 - Sentences with all variables properly quantified

Connections between Quantifiers

- **Equivalences**

- Using the negation (*hence only one quantifier is needed*)

$$\forall x P(x) \Leftrightarrow \neg \exists x \neg P(x)$$

- e.g. "everyone is mortal":

$$\forall x \text{ Mortal}(x) \Leftrightarrow \neg \exists x \neg \text{Mortal}(x)$$

- De Morgan's Laws

•	$\forall x P \Leftrightarrow \neg \exists x \neg P$	$P \wedge Q \Leftrightarrow \neg(\neg P \vee \neg Q)$
	$\forall x \neg P \Leftrightarrow \neg \exists x P$	$\neg P \wedge \neg Q \Leftrightarrow \neg(P \vee Q)$
	$\neg \forall x P \Leftrightarrow \exists x \neg P$	$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$
	$\neg \forall x \neg P \Leftrightarrow \exists x P$	$\neg(\neg P \wedge \neg Q) \Leftrightarrow P \vee Q$

Equality Predicate Symbol

- **Need for equality**
 - State that two terms refer to the same object
 - e.g., $\text{Father}(\text{John}) = \text{Henry}$, or
 - $=(\text{Father}(\text{John}), \text{Henry})$
 - Useful to define properties
 - e.g. “King John has two brothers”:
 - $\exists x, y \text{ Brother}(x, \text{KingJohn}) \wedge \text{Brother}(y, \text{KingJohn}) \wedge \neg(x=y)$

Grammar of First-Order Logic

(Backus-Naur Form)

Sentence

→ AtomicSentence | (Sentence)
| Sentence Connective Sentence
| \neg Sentence
| Quantifier Variable, ... Sentence

AtomicSentence

→ Predicate(Term, ...) | Term = Term

Term

→ Function(Term, ...) | Constant | Variable

Connective

→ \wedge | \vee | \Leftrightarrow | \Rightarrow

Quantifier

→ \forall | \exists

Constant

→ A | X_1 | John | ...

Variable

→ a | x | person | ...

Predicate

→ P() | Colour() | Before() | ...

Function

→ F() | MotherOf() | SquareRootOf() | ...

Using First-Order Logic

- **Knowledge domain**
 - A part of the world we want to express knowledge about
- **Example of the kinship domain**
 - Objects: people e.g., Elizabeth, Charles, William, etc.
 - Properties: gender i.e., male, female
 - Unary predicates: Male() and Female()
 - Relations: kinship e.g., motherhood, brotherhood, etc.
 - Binary predicates: Parent(), Sibling(), Brother(), Child(), etc.
 - Functions: MotherOf(), FatherOf()
 - > Express facts e.g., Charles is a male
and rules e.g., the mother of a parent is a grandmother

Sample Functions and Predicates

- **Functions**

$$\forall x,y \text{ FatherOf}(x)=y \Leftrightarrow \text{Parent}(y,x) \wedge \text{Male}(y)$$

$$\forall x,y \text{ MotherOf}(x)=y \Leftrightarrow \text{Parent}(y,x) \wedge \text{Female}(y)$$

- **Predicates**

→ $\forall x,y \text{ Parent}(x,y) \Leftrightarrow \text{Child}(y,x)$

$$\forall x,y \text{ Grandparent}(x,y) \Leftrightarrow \exists z, \text{Parent}(x,z) \wedge \text{Parent}(z,y)$$

$$\forall x,y \text{ Sibling}(x,y) \Leftrightarrow \neg x=y \wedge \exists z, \text{Parent}(z,x) \wedge \text{Parent}(z,y)$$

$$\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x)$$

- **Potential problems**

- Self-definition (causes infinite recursion)

- e.g., $\forall x,y \text{ Child}(x,y) \Leftrightarrow \text{Parent}(y,x)$ following the above

TELLing and ASKing

- **TELLing the KB**
 - Assertion: add a sentence to the knowledge base
 - e.g.
TELL(KB, $\forall x,y \text{ MotherOf}(x)=y \Leftrightarrow \text{Parent}(y,x) \wedge \text{Female}(y)$)
and so on, then
TELL(KB, $\text{Female}(\text{Elizabeth}) \wedge \text{Parent}(\text{Elizabeth},\text{Charles}) \wedge \text{Parent}(\text{Charles},\text{William})$)
- **ASKing the KB**
 - Query: retrieve/infer a sentence from the knowledge base
 - Yes/No answer
 - e.g. ASK(KB, Grandparent(Elizabeth, William))
 - Binding list, or substitution
 - e.g. ASK(KB, $\exists x \text{ Child}(\text{William}, x)$) yields {x / Charles}

Inferences Rules for FOL

- Inference rules from Propositional Logic
 - *Modus Ponens*
 - $$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$
 - And-Elimination
 - $$\frac{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}{\alpha_i}$$
 - Or-Introduction
 - $$\frac{}{\alpha_1 \vee \alpha_2 \vee \dots \vee \alpha_n}$$
 - Double-Negation-Elimination
 - $$\frac{\neg\neg\alpha}{\alpha}$$
 - And-Introduction
 - $$\frac{\alpha_1, \alpha_2, \dots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}$$
 - Resolution
 - $$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Universal Elimination

- $\forall x, \text{Likes}(x, \text{flower})$
- Substituting x by Shirin gives
- Likes(Shirin, flower)
- The substitution should be done by a constant term.
- In this way, the \forall quantifier can be eliminated.

Existential Elimination/Introduction

- **Existential Elimination**
 - $\exists x, \text{likes}(x, \text{flower})$
 - Can be changed to:
 - $\text{likes}(\text{Person}, \text{flower})$
 - As long as the person is not in the knowledge base.
- **Existential Introduction**
 - $\text{Likes}(\text{Marry}, \text{flower})$
 - Can be written as:
 - $\exists x, \text{likes}(x, \text{flower})$

Inferences Rules with Quantifiers

- **Substitutions**

- SUBST(θ , α):
- binding list θ applied to a sentence α
 - e.g., $\text{SUBST}(\{x / \text{John}, y / \text{Richard}\}, \text{Brother}(x, y)) = \text{Brother}(\text{John}, \text{Richard})$

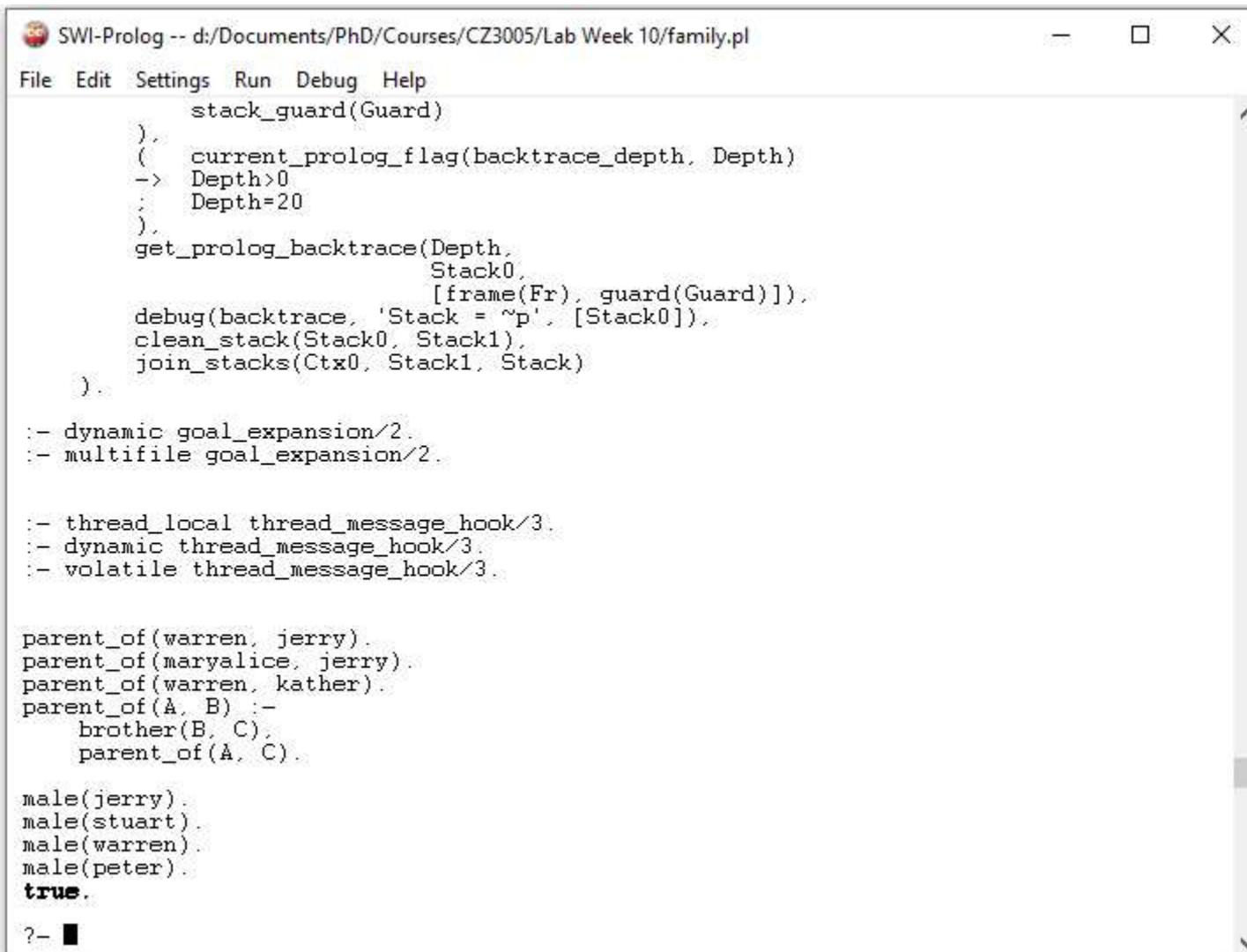
Working Example with Prolog

- SWI-Prolog offers a comprehensive free Prolog environment.
- Since its start in 1987, SWI-Prolog development has been driven by the needs of real world applications.
- SWI-Prolog is widely used in research and education as well as commercial applications.
- Download it here: <https://www.swi-prolog.org/>
- Let's see the “royal family” example together



SWI Prolog

Working Example with Prolog



The screenshot shows the SWI-Prolog IDE interface with a file named `family.pl`. The code is as follows:

```
SWI-Prolog -- d:/Documents/PhD/Courses/CZ3005/Lab Week 10/family.pl
File Edit Settings Run Debug Help
stack_guard(Guard)
).
( current_prolog_flag(backtrace_depth, Depth)
-> Depth>0
; Depth=20
).
get_prolog_backtrace(Depth,
Stack0,
[frame(Fr), guard(Guard)]),
debug(backtrace, 'Stack = ~p', [Stack0]),
clean_stack(Stack0, Stack1),
join_stacks(Ctx0, Stack1, Stack)
).

:- dynamic goal_expansion/2.
:- multifile goal_expansion/2.

:- thread_local thread_message_hook/3.
:- dynamic thread_message_hook/3.
:- volatile thread_message_hook/3.

parent_of(warren, jerry).
parent_of(maryalice, jerry).
parent_of(warren, kather).
parent_of(A, B) :-
    brother(B, C),
    parent_of(A, C).

male(jerry).
male(stuart).
male(warren).
male(peter).
true.
?- ■
```



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 11a – Default Logic

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Learning Goals

Understanding the:

- Basics of Reiter's Default Logic (RDL)
- Limitations of RDL

Default Logic

- Standard logic can only express that something is true or that something is false.
- Default logic is proposed to formalize reasoning with default assumptions.
- It can express facts like
 - “by default, something is true”.

Unicorns



- **Given:**
 - If the unicorn is mythical, then it is immortal.
 - If it is not mythical, then it is a mortal mammal.
 - If it is either immortal or a mammal, then it is horned.
 - The unicorn is magical if it is horned.
- **Question:**
 - Based on Modus Ponens, can we show that the unicorn is mythical, magical and horned?

Unicorns



- **KB:**

Mythical $\Rightarrow \neg$ Mortal

\neg Mythical \Rightarrow Mortal

\neg Mythical \Rightarrow Mammal

\neg Mortal \Rightarrow Horned

Mammal \Rightarrow Horned

Horned \Rightarrow Magical

- **Assumptions**

I can believe

$$\frac{: \text{Mythical}}{\text{Mythical}}$$

or

$$\frac{: \neg \text{Mythical}}{\neg \text{Mythical}}$$

Mythical

\neg Mythical

\neg Mortal

Mammal

Horned

Horned

Magical

Magical

Default Logic

Prerequisite : Justification₁, …, Justification_n

Conclusion

According to this default:

- If we believe that Prerequisite is true;
- AND each of Justifications is consistent with our current beliefs;
- THEN, we are led to believe that the Conclusion is true.

Default Logic Syntax

$$\frac{\alpha(x) : \beta_1(x), \dots, \beta_n(x)}{\gamma(x)}$$

where $x = x_1, \dots, x_m$, and $\alpha(x), \beta_1(x), \dots, \beta_n(x), \gamma(x)$ are formulae whose free variables are among x_1, \dots, x_m .

The default is **applied** by substituting c (the ground instance) into α and β to infer γ :

- Trigger: $\alpha(c)$ belongs to our set of beliefs.
 - Justification: the set of our beliefs is consistent with each $\beta(c)$.

Our KB does not entail $\neg\beta(c)$

Default Theory

Delta: Set of Defaults

$\langle \Delta, \Phi \rangle$

Phi: Set of “Facts”

Example – the default rule that “birds typically fly”:

- $\Delta = \left\{ \frac{\text{bird}(x) : \text{flies}(x)}{\text{flies}(x)} \right\}$
 - This rule means that, "if x is a bird, and it can be assumed that x flies, then we can conclude that x flies".
- $\Phi = \{\text{bird}(\text{Tweety}), \text{cat}(\text{Sylvester})\}$

Default Theory

A common default assumption:

- What is not known to be true is believed to be false.
- This is known as the Closed-World Assumption.
- It is formalized in default logic using a default as follows for every fact F.

$$\frac{: \neg F}{\neg F}$$

Types of Defaults

- Normal Defaults:
$$\frac{\alpha(x) : \gamma(x)}{\gamma(x)}$$
- Semi-Normal Defaults:
$$\frac{\alpha(x) : \beta(x)}{\gamma(x)}$$
, where $\beta(x) \vdash \gamma(x)$

E.g.,
$$\frac{bird(x) : flies(x) \wedge \neg swim(x)}{flies(x)},$$

where $flies(x) \wedge \neg swim(x) \vdash flies(x)$

Types of Defaults

- Open Defaults (Default Schemas) have unbounded variables, e.g., x

$$\frac{\alpha(x) : \beta_1(x), \dots, \beta_n(x)}{\gamma(x)}$$

- Closed (Grounded) Defaults use ground terms, e.g., $x=c$

$$\frac{\alpha(c) : \beta_1(c), \dots, \beta_n(c)}{\gamma(c)}$$

Reiter Inference with Default Theory

- Guess the extension Ξ (pronounced as “Xi”)
- Initialise beliefs $\Xi^* = \Phi$
- (loop over) c-ground instance of an (unused) default
$$\frac{\alpha(x) : \beta(x)}{\gamma(x)}:$$
 - Check two conditions
 - Triggered?: $\Xi^* \vdash \alpha(c)$
 - Justified?: $\beta(c)$ is consistent with Ξ
 - If yes: update beliefs $\Xi^* \leftarrow \Xi^* \cup \{\gamma(c)\}$
- (end loop)
- If $\Xi = \Xi^*$ then extension found/confirmed

Reiter Extension: Compact Description

Given a default theory $\langle \Delta, \Phi \rangle$, Ξ is an extension if and only if

$$\Xi = \text{Cn} \left(\bigcup_{i=1}^{\infty} \Xi_i \right)$$

where:

- $\Xi_0 = \Phi$
- $\Xi_{i+1} = \Xi_i \cup \left\{ \gamma(c) \mid \frac{\alpha(x): \beta_1(x), \dots, \beta_n(x)}{\gamma(x)} \in \Delta, \Xi_i \vdash \alpha(c), \neg\beta_1(c), \dots, \neg\beta_n(c) \notin \Xi \right\}$

Cn means applying any known inference rules to expand the KB

Example

Given Theory: $T = \left\langle \Delta = \left\{ \frac{\text{bird}(x) : \text{flies}(x)}{\text{flies}(x)} \right\}, \Phi = \{\text{bird}(\text{Tweety}), \text{cat}(\text{Sylvester})\} \right\rangle$,

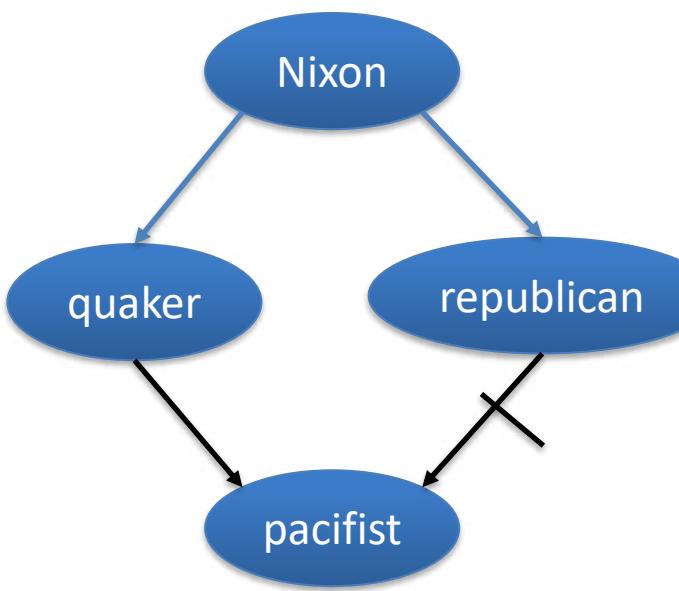
- Guess the extension = $Cn(\{\text{flies}(\text{Tweety})\} \cup \Phi)$
- Our initial knowledge is $F = \Phi$
- Sylvester-instance of default not applicable:
 - not hold $\Phi \vdash \text{bird}(\text{Sylvester})$
- $\Phi \vdash \text{bird}(\text{Tweety})$ and $\text{flies}(\text{Tweety})$ is consistent with F
- $F = \Phi \cup \{\text{flies}(\text{Tweety})\}$
- No more default rules to apply
- An extension is reached

Implications of Default Theory

- A default rule can be applied to a theory
 - if its precondition is entailed by the theory; and
 - its justifications are all consistent with the theory.
- The application of a default rule leads to the addition of its consequence to the theory.
- Other default rules may then be applied to the resulting theory.
- When the theory is such that no other default can be applied, the theory is called an extension of the default theory.

Nixon Diamond

The default rules may be applied in different orders, and this may lead to different extensions. E.g.:



Let Theory $T = \langle \Delta, \Phi \rangle$

- $= \left\{ \frac{\frac{quaker(x) : pacifist(x)}{pacifist(x)},}{\frac{republican(x) : \neg pacifist(x)}{\neg pacifist(x)}} \right\}$
- $\Phi = \{quaker(Nixon), republican(Nixon)\}$

Nixon Diamond

Given $\langle \Delta, \Phi \rangle$

- $\Delta = \left\{ \frac{\text{quaker}(x) : \text{pacifist}(x)}{\text{pacifist}(x)}, \frac{\text{republican}(x) : \neg \text{pacifist}(x)}{\neg \text{pacifist}(x)} \right\}$
- $\Phi = \{\text{quaker}(\text{Nixon}), \text{republican}(\text{Nixon})\}$

There are **two** extensions:

1. One that contains: $\text{pacifist}(\text{Nixon})$
2. .. and the one that contains: $\neg \text{pacifist}(\text{Nixon})$

Entailment

- A default theory can have 0, 1 or more extensions.
- Entailment of a formula from a default theory can be defined in one of two ways:
 - Skeptical:
 - a formula is entailed by a default theory if it is entailed by all its extensions.
 - Credulous:
 - a formula is entailed by a default theory if it is entailed by at least one of its extensions.

Example

- The Nixon diamond example theory has two extensions:
 - one in which Nixon is a pacifist; and
 - one in which Nixon is not a pacifist.
- Thus, we have:
 - Neither $\text{Pacifist}(\text{Nixon})$ nor $\neg\text{Pacifist}(\text{Nixon})$ are skeptically entailed.
 - Both $\text{Pacifist}(\text{Nixon})$ and $\neg\text{Pacifist}(\text{Nixon})$ are credulously entailed.
- The credulous extensions of a default theory can be inconsistent with each other.



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 11b – Default Logic (Process)

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Learning Goals

Understanding the:

- Improved variant of Reiter's Default Logic (RDL)

Recap

- A default rule can be applied to a theory
 - if its precondition is entailed by the theory; and
 - its justifications are all consistent with the theory.
- The application of a default rule leads to the addition of its consequence to the theory.
- Other default rules may then be applied to the resulting theory.
- When the theory is such that no other default can be applied, the theory is called an extension of the default theory.
- The default rules may be applied in different orders, and this may lead to different extensions.

Makinson Approach

- Order ***ground*** instances of defaults in Δ : d_1, d_2, \dots
- Initialize beliefs $\Xi_0 = \Phi$ and used defaults set $\Delta_0 = \emptyset$
- Define Ξ_{n+1} from Ξ_n ,
 - Find $d = \frac{\alpha(c) : \beta_1(c), \dots, \beta_n(c)}{\gamma(c)} \notin \Delta_n$ such that
 - Triggered?: $\Xi_n \vdash \alpha(c)$
 - Justified?: Ξ_n is consistent with $\beta_1(c), \dots, \beta_m(c)$
 - If $\Xi_n \cup \{\gamma(c)\}$ is consistent with each $\beta'(c')$ in $\Delta_n \cup \{d\}$
 - $\Xi_{n+1} = \Xi_n \cup \{\gamma(c)\}$, and $\Delta_{n+1} = \Delta_n \cup \{d\}$
 - else **abort -- no extension for this order of defaults**
- The extension is $\Xi = \bigcup_{i \geq 0} \Xi_i$

Makinson Approach

- No extension guessing
 - **Choose** the order of defaults in Δ : d_1, d_2, \dots
- There still may be more than one possible extension
 - **Different orders** of defaults can lead to different Ξ
- We get the same extensions as in Reiter's approach
 - If they exist at all

Remember ...

Delta: Set of Defaults

$\langle \Delta, \Phi \rangle$

Phi: Set of “Facts”

$Cn()$

Cn means applying any known
inference rules to expand the KB

Operational Semantics

Given a default theory $T = \langle \Delta, \Phi \rangle$, let $\Pi = (\delta_0, \delta_1, \dots)$ be (a finite or infinite) sequence of (closed) defaults from Δ without multiple occurrences.

$\Pi[k]$ denotes the initial segment of sequence Π with length k .

Model of the world

Each sequence Π is associated with two sets:

- $In(\Pi) = Cn(\Phi \cup \{consequence(\delta) | \delta \text{ occurs in } \Pi\})$
- $Out(\Pi) = \{\neg\phi | \phi \in justifications(\delta) \text{ for some } \delta \text{ in } \Pi\}$

Everything I have denied the existence of as a result of my assumptions (i.e., the negation of all the justifications)

Example

Consider $T = \langle \Delta, \Phi \rangle$ with $\Phi = \{\alpha\}$ and defaults from Δ :

$$\delta_1 = \frac{\alpha : \neg\beta}{\neg\beta}, \quad \delta_2 = \frac{\beta : \gamma}{\gamma}$$

For $\Pi_a = (\delta_1)$ we have

$$In(\Pi_a) = Cn(\{\alpha, \neg\beta\}), Out(\Pi_a) = \{\beta\}$$

For $\Pi_b = (\delta_2, \delta_1)$ we have

$$In(\Pi_b) = Cn(\{\alpha, \neg\beta\}), Out(\Pi_b) = \{\beta\}$$

Process, Successful, Closed

Π is a process of $T = \langle \Delta, \Phi \rangle$ iff default δ_k is applicable to $In(\Pi[k])$ for every k such that δ_k occurs in Π .

Let Π be a process. We define:

- Π is **successful** iff $In(\Pi) \cap Out(\Pi) = \emptyset$ (*Nothing in the out set can be inferred from the in set*);
Otherwise, it **fails**.
- Π is **closed** iff every $\delta \in \Delta$ that is applicable to $In(\Pi)$ already occurs in Π .

Example

Consider $T = \langle \Delta, \Phi \rangle$ with $\Phi = \{\alpha\}$ and defaults from Δ :

$$\delta_1 = \frac{\alpha : \neg\beta}{\eta}, \quad \delta_2 = \frac{\text{true} : \gamma}{\beta}$$

$\Pi_1 = (\delta_1)$ is successful,

$In(\Pi_1) = Cn(\alpha, \eta)$ and $Out(\Pi_1) = \{\beta\}$
but not closed, since δ_2 is applicable, too.

$\Pi_2 = (\delta_1, \delta_2)$ is closed, but not successful

$In(\Pi_2) = Cn(\alpha, \eta, \beta)$ and $Out(\Pi_2) = \{\beta, \neg\gamma\}$,
 $In(\Pi_2) \cap Out(\Pi_2) = \beta$

$\Pi_3 = (\delta_2)$ is a closed and successful process T

$In(\Pi_3) = Cn(\alpha, \beta)$ and $Out(\Pi_3) = \{\neg\gamma\}$,
 $In(\Pi_3) \cap Out(\Pi_3) = \emptyset$

Extension

- Let $T = \langle \Delta, \Phi \rangle$ be a default theory. A set of formulae Σ is an **extension** of T iff there is some ***closed and successful*** Π such that $\Sigma = In(\Pi)$.
- To **find a successful** process: generate a process Π , test whether $in(\Pi) \cap Out(\Pi) = \emptyset$. If not, then backtrack (try another process).

Process Tree

$T = \langle \Delta, \Phi \rangle$ be a default theory. A **process tree** is a tree $G = (V, E)$ such that all nodes $v \in V$ are labelled with two sets of formulae:

- an In-set $\text{In}(v)$ and
- an Out-set $\text{Out}(v)$.

The root of G is labelled with $C_n(\Phi)$ as the In-set and \emptyset as the Out-set. Every $e \in E$ denotes a default application and is labelled by it.

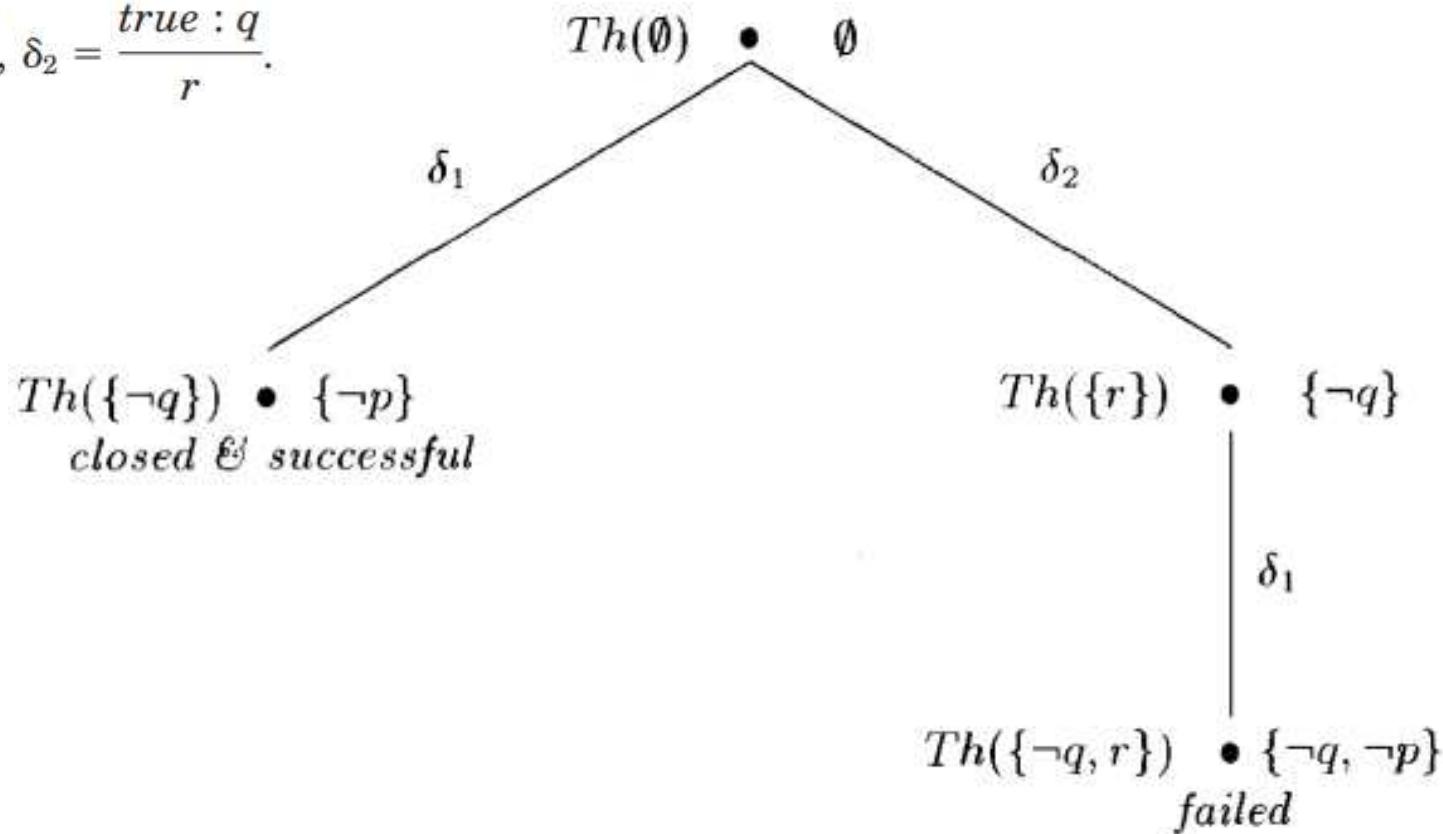
A process is thus a path in G starting from the root.

A node $v \in V$ is **expanded** if $\text{In}(v) \cap \text{Out}(v) = \emptyset$. Otherwise, it is a “failed” leaf of the tree.

Process Tree Example

Let $T = (W, D)$ be the default theory
with $W = \emptyset$ and $D = \{\delta_1, \delta_2\}$ with

$$\delta_1 = \frac{\text{true} : p}{\neg q}, \quad \delta_2 = \frac{\text{true} : q}{r}.$$



Process Tree: Properties

- A process is thus a path in G starting from root.
- A node $v \in V$ is **expanded** if $In(v) \cap Out(v) = \emptyset$.
- Otherwise, it is a “failed” leaf of the tree.
- Expanded $v \in V$ has a child node, w_δ , for every $\delta = \frac{\alpha : \beta_1, \dots, \beta_n}{\gamma}$
 - w_δ does not appear on the path from the root to v
 - δ is applicable to $In(v)$
 - w_δ connected to v by an edge labelled with δ
 - w_δ is labelled with $In(w_\delta) = Cn(In(v) \cup \{\gamma\})$ and $Out(w_\delta) = Out(v) \cup \{\neg\beta_1, \dots, \neg\beta_n\}$



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 12a – Fuzzy Logic

Yu Han

han.yu@ntu.edu.sg

*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



Learning Goals

Understanding the:

- Basic definitions and terminology
- Set-theoretic operations
- Membership Function (MF) formulation
 - MFs parameterization
 - Linguistic modifier/hedges

Example: Safe Autonomous Vehicles

- Autonomous Cars implement Duty of Care
 - an individual should exercise “reasonable care” while performing acts that could harm others
- “On a Formal Model of Safe and Scalable Self-driving Cars”, by Shalev-Swartz, Shammah, and Shashua, arXiv 1708.06374
 - Responsibility Sensitive Safety – mathematical safety assurance
 - System design that adheres to the mathematical model



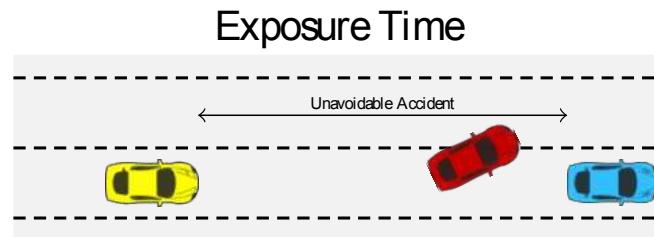
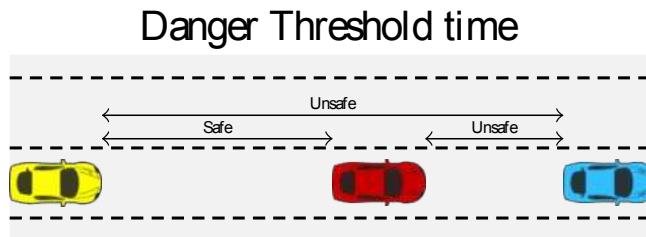
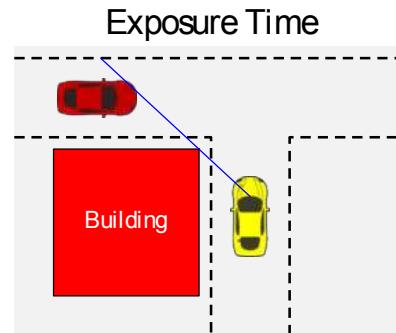
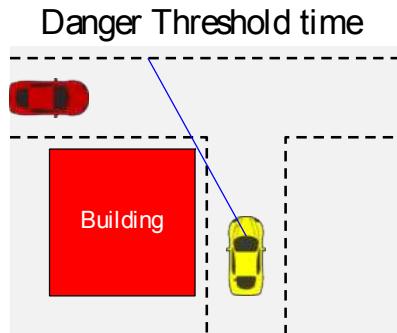
Example: Safe Autonomous Vehicles

- Responsibility Sensitive Safety (RSS)
 - Do not hit someone from behind
 - Do not cut-in recklessly
 - Right-of-way is given, not taken
 - Be careful of areas with limited visibility
 - If you can avoid an accident without causing another one, you must do so

Example: Safe Autonomous Vehicles

- Responsibility Sensitive Safety (RSS)
 - Do not hit someone from behind
 - Even if not your fault?
 - Do not cut-in recklessly
 - Right-of-way is given, not taken
 - How to resolve polite deadlocks?
 - Be careful of areas with limited visibility
 - If you can avoid an accident without causing another one, you must do so
 - Emergency breaking can cause whiplash

Example: Safe Autonomous Vehicles



“On a Formal Model of Safe and Scalable Self-driving Cars”,
by Shalev-Swartz, Shammah, and Shashua, arXiv 1708.06374

Example: Safe Autonomous Vehicles

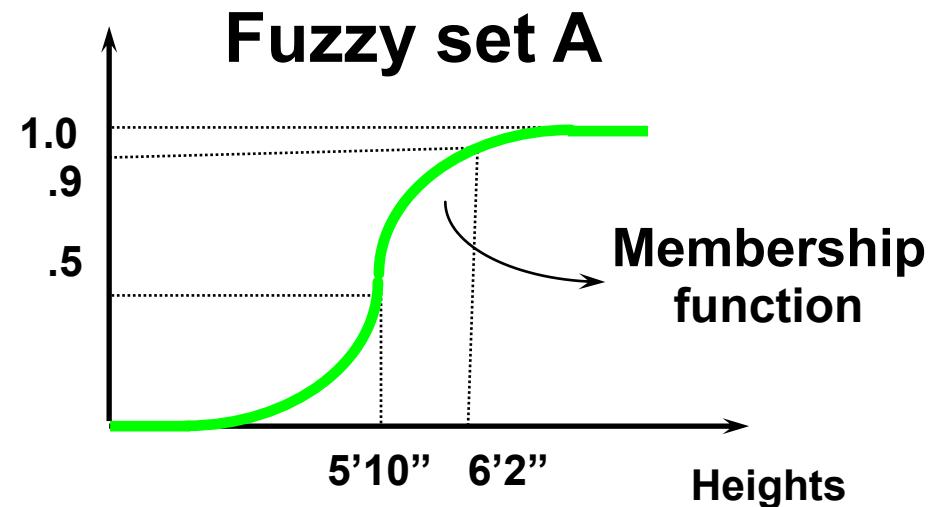
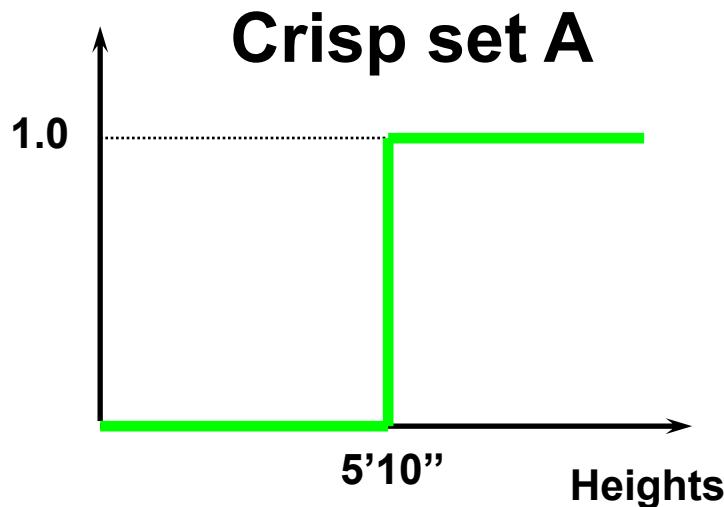
- Use of Semantic Action Space
 - Not “drive for 5.33 kilometers, then reduce speed at the rate of 1 m/s^2 ”
 - Slow down as you approach red light to stop at the line.
 - IF approach red light, THEN slow down and stop

"On a Formal Model of Safe and Scalable Self-driving Cars",
by Shalev-Swartz, Shammah, and Shashua, arXiv 1708.06374

Fuzzy Sets

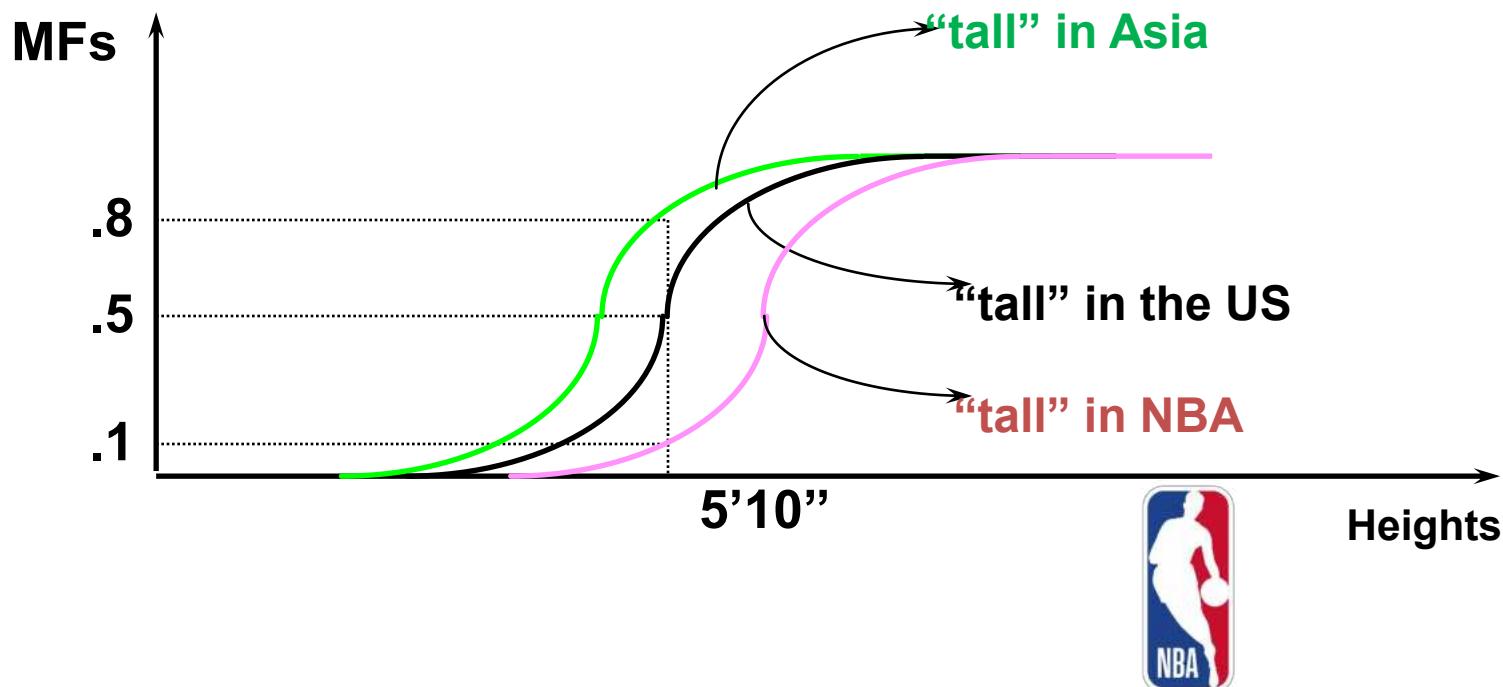
- Sets with fuzzy boundaries

A = Set of tall people



Membership Functions (MFs)

- Characteristics of MFs:
 - Subjective measures
 - Not probability functions



Fuzzy Sets

Formal definition:

A fuzzy set A in X is expressed as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) \mid x \in X\}$$

Crisp input

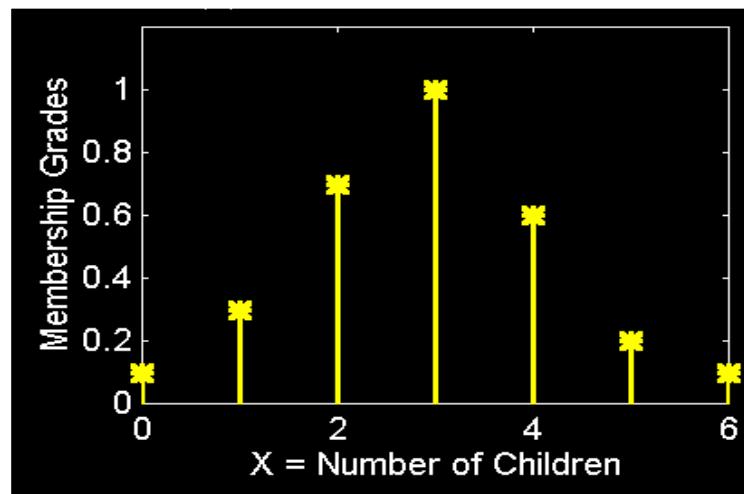
Fuzzy set **Membership function (MF)** **Universe or universe of discourse**

The diagram illustrates the formal definition of a fuzzy set A . It shows the set $A = \{(x, \mu_A(x)) \mid x \in X\}$ with three main components: 'Crisp input' (the element x), 'Membership function (MF)' ($\mu_A(x)$), and 'Universe or universe of discourse' (X). Curved arrows point from the labels to their respective parts in the set definition.

A fuzzy set is totally characterized by a membership function (MF).

Fuzzy Sets – Discrete Universes

- Fuzzy set C = “desirable city to live in”
 $X = \{\text{SF, Boston, LA}\}$ (discrete and non-ordered)
 $C = \{(\text{SF}, 0.9), (\text{Boston}, 0.8), (\text{LA}, 0.6)\}$
- Fuzzy set A = “sensible number of children to have”
 $X = \{0, 1, 2, 3, 4, 5, 6\}$ (discrete universe)
 $A = \{(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)\}$



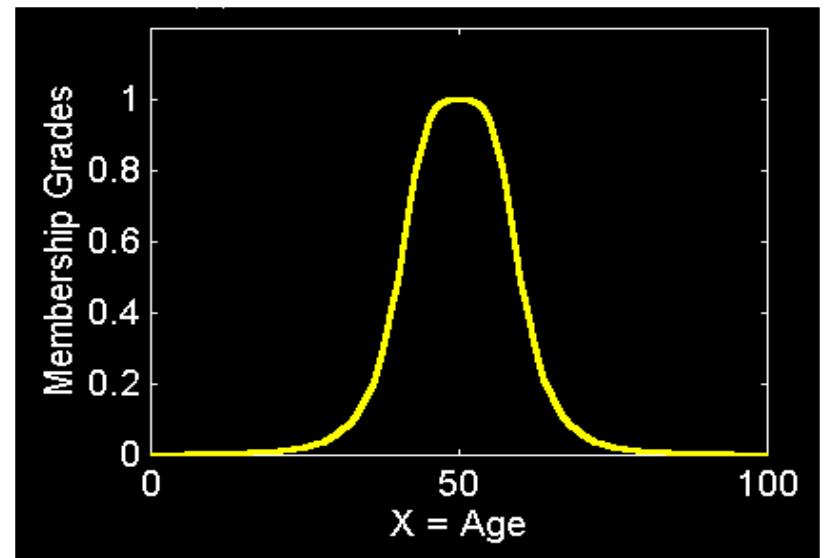
Fuzzy Sets – Continuous Universes

- Fuzzy set B = “about 50 years old”

X = Set of positive real numbers (continuous)

$$B = \{(x, \mu_B(x)) \mid x \text{ in } X\}$$

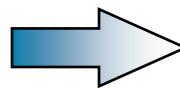
$$\mu_B(x) = \frac{1}{1 + \left(\frac{x - 50}{10}\right)^2}$$



Alternative Notation

A fuzzy set A can be alternatively denoted as follows:

X is discrete

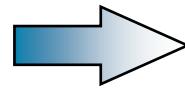


$$A = \sum_{x_i \in X} \mu_A(x_i) / x_i$$

$X = \{0, 1, 2, 3, 4, 5, 6\}$ (discrete universe)

$A = \{(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)\}$

X is continuous



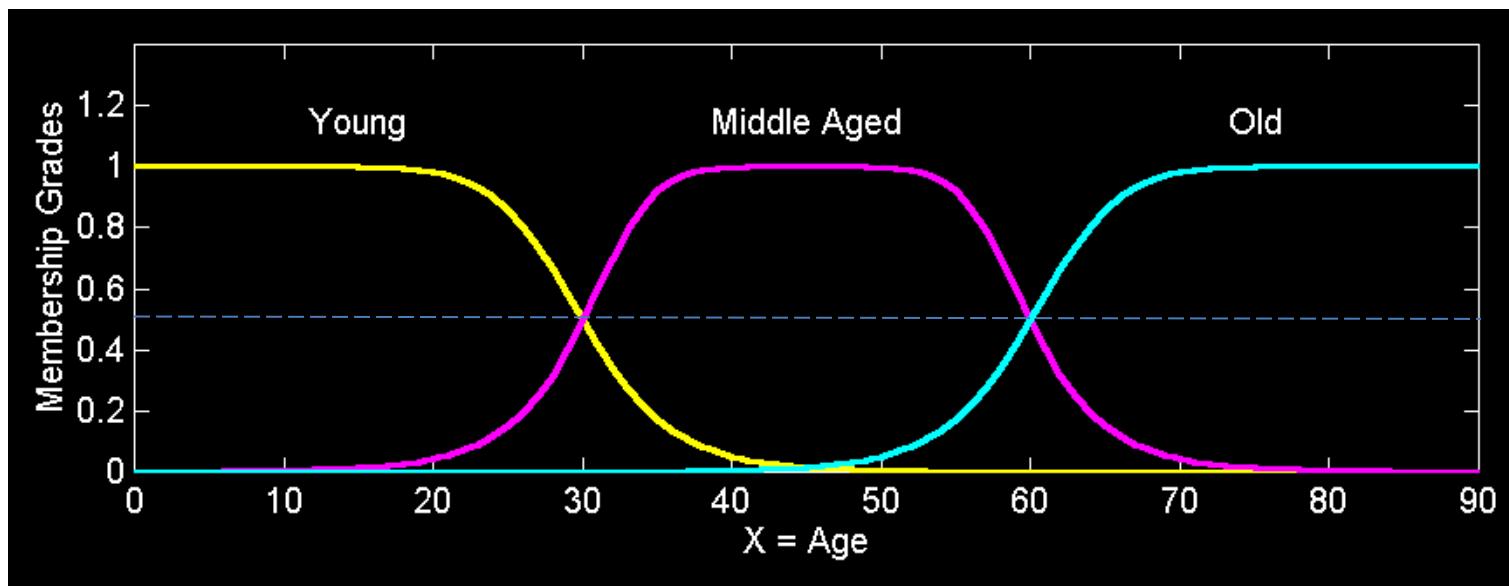
$$A = \int_X \mu_A(x) / x$$

Note that Σ and integral signs stand for the union of membership grades;

“/” stands for a marker and does not imply division.

Fuzzy Partition

- Fuzzy partitions formed by the linguistic values “young”, “middle aged”, and “old”:



Non-Pseudo Partitioning

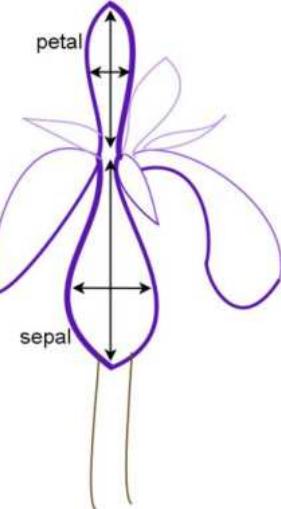
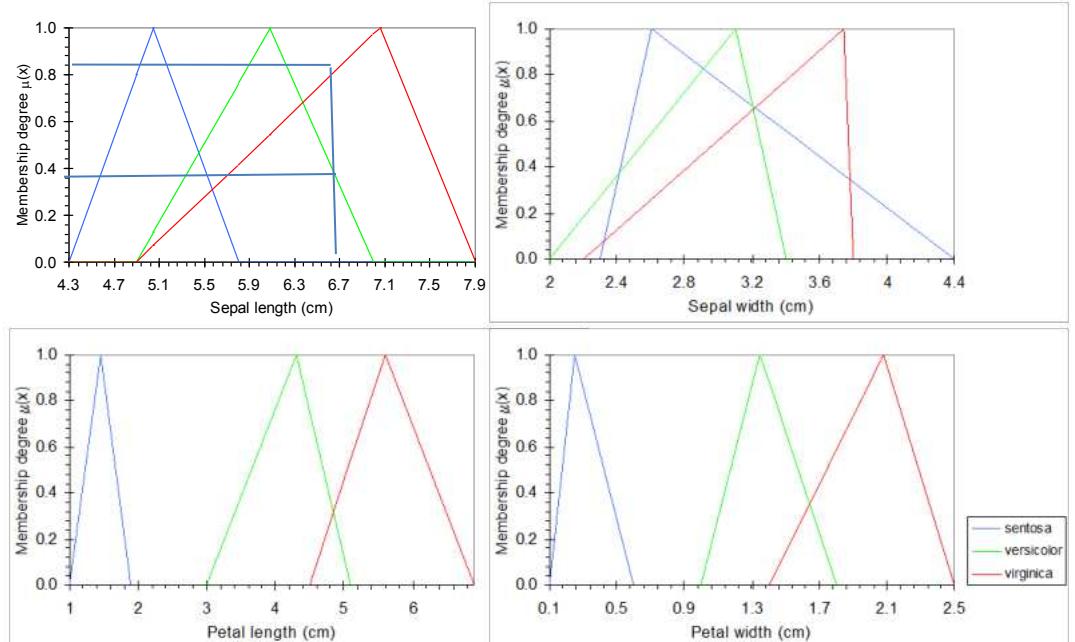
- Let c the set of membership functions that fuzzy partition the space of x .
- This fuzzy space is **non-Pseudoly partitioned** when:

Each MF value is normal and convex

$$\sup_x (\mu_{i,i \in c}(X)) = 1$$

Summation of MF values at X is NOT 1

$$\sum_{i=1}^c \mu_{i,i \in c}(X) \neq 1$$



Iris flower

Pseudo Partitioning

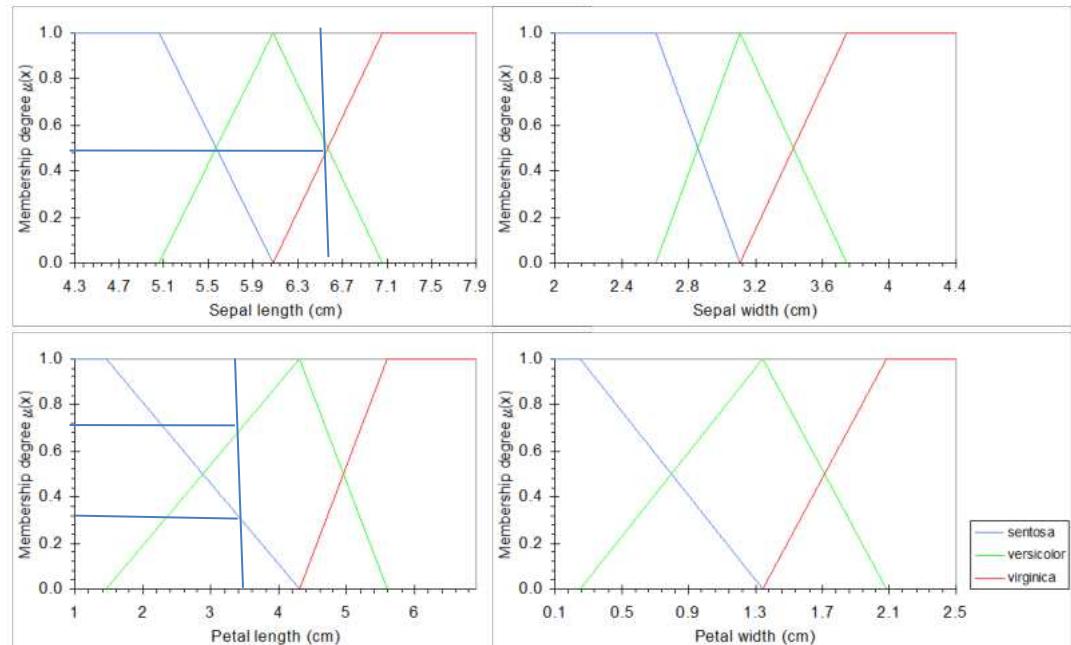
- Let c be the set of membership functions that fuzzy partition the space of x .
- This fuzzy space is **Pseudoly partitioned** when:

Each MF value is normal and convex

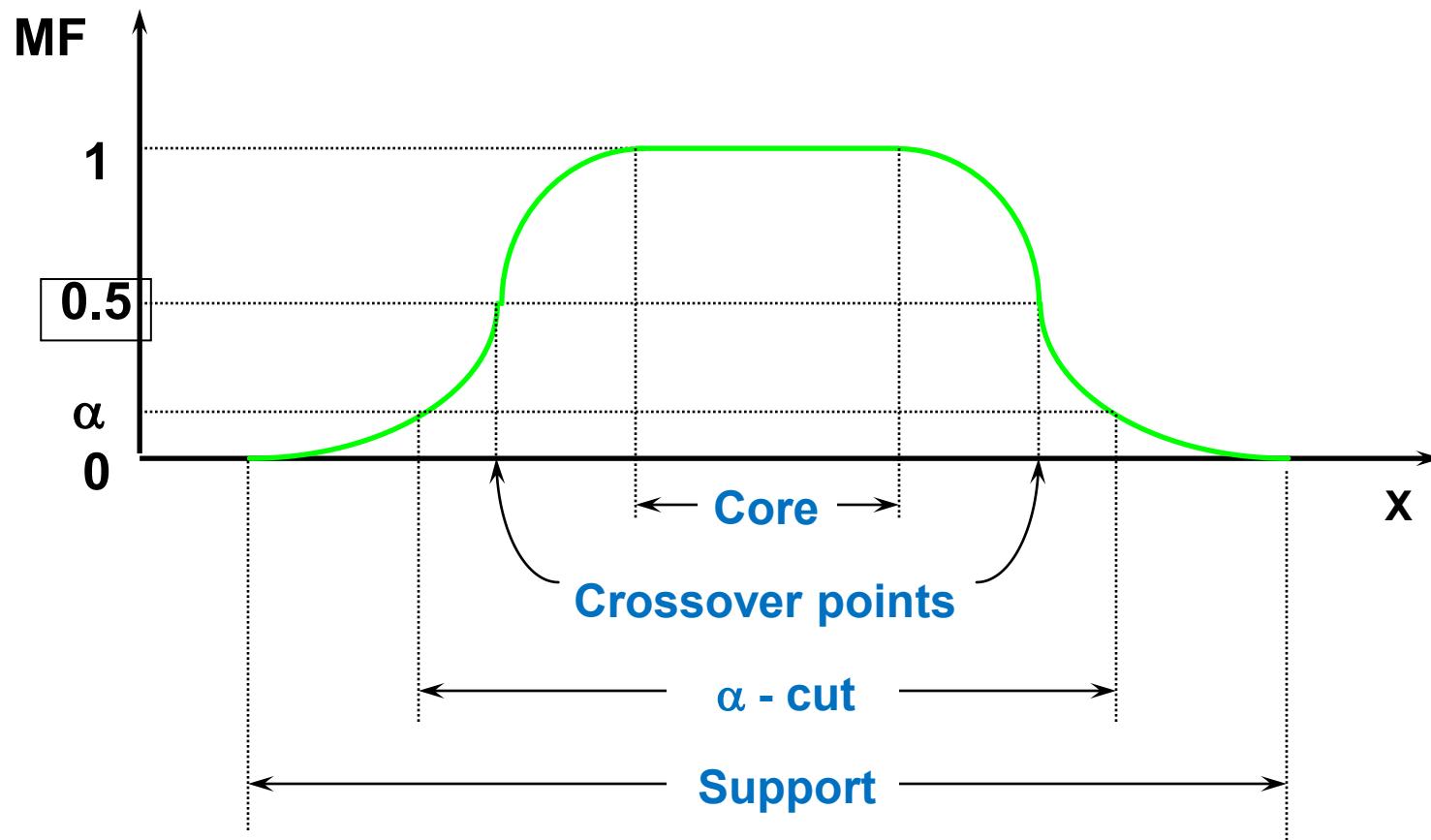
$$\sup_x (\mu_{i,i \in c}(X)) = 1$$

Summation of MF values at X is 1

$$\sum_{i=1}^c \mu_{i,i \in c}(X) = 1$$



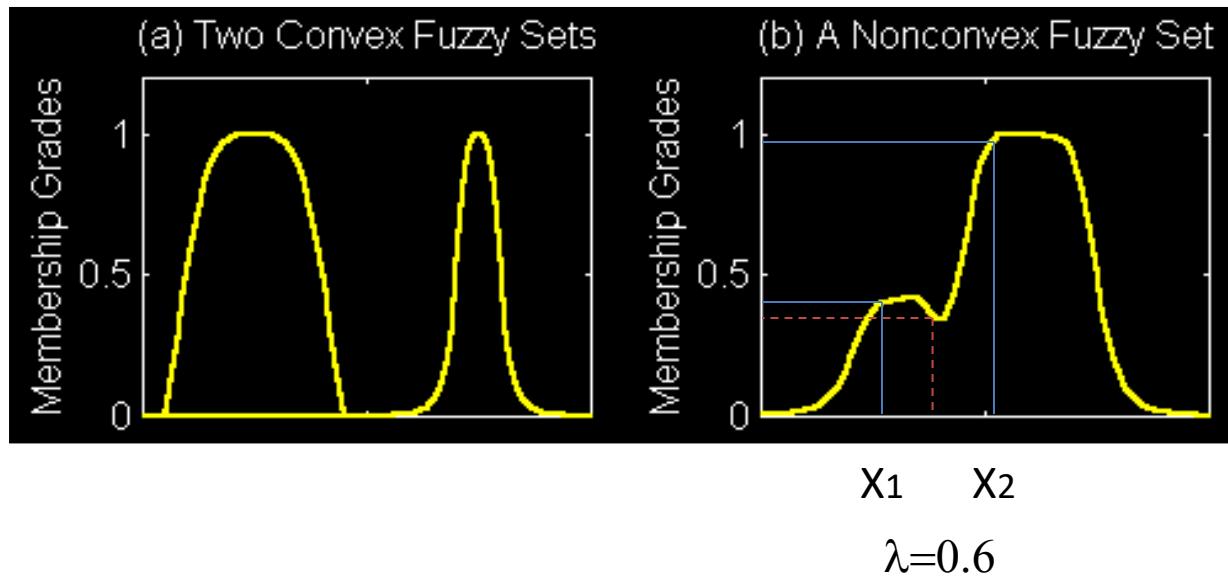
MF Terminology



Convexity of Fuzzy Sets

A fuzzy set A is convex if for any λ within $[0, 1]$:

$$\mu_A(\lambda x_1 + (1 - \lambda) x_2) \geq \min(\mu_A(x_1), \mu_A(x_2))$$



Alternatively, A is convex if all its *α -cuts* are convex.

Set-Theoretic Operations

- Subset:

$$A \subseteq B \Leftrightarrow \mu_A \leq \mu_B$$

- Complement:

$$\overline{A} = X - A \Leftrightarrow \mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

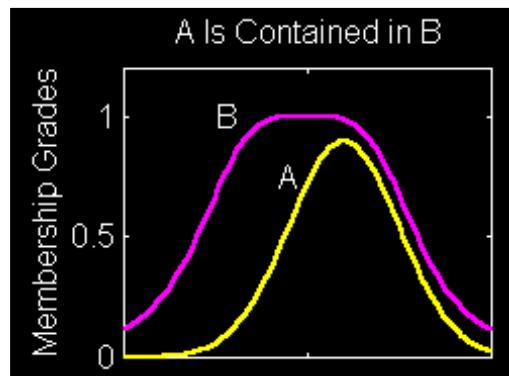
- Union: (OR - Disjunction)

$$C = A \cup B \Leftrightarrow \mu_c(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

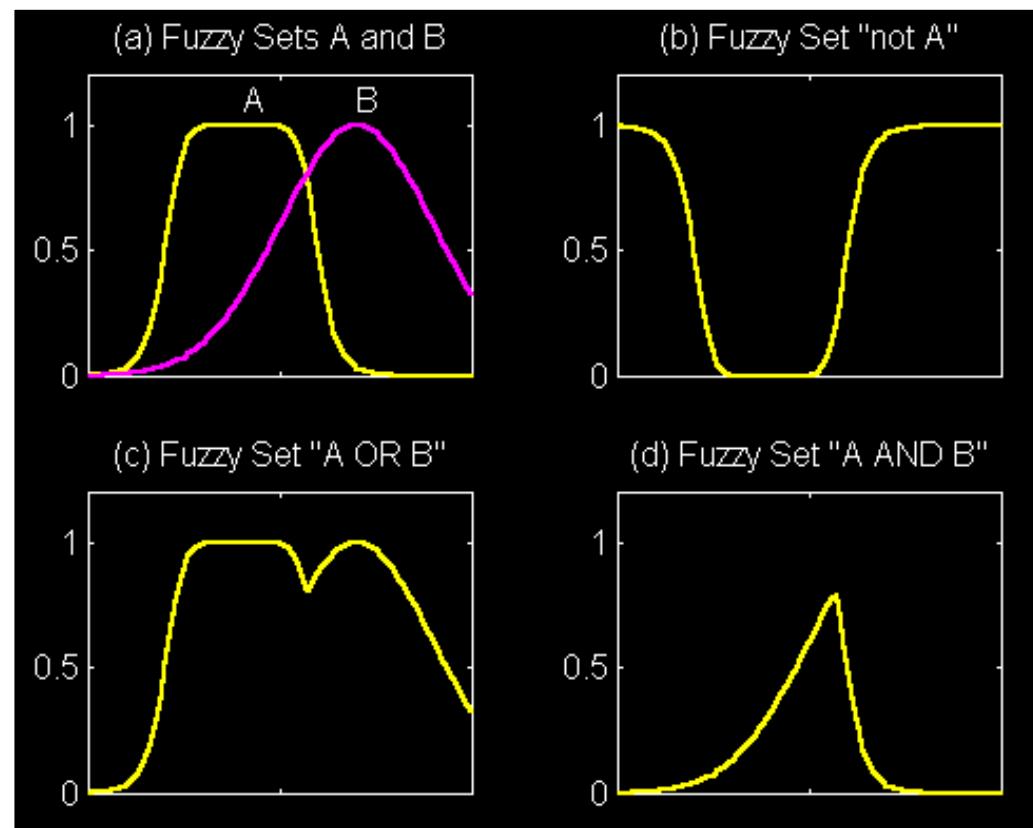
- Intersection: (AND – Conjunction)

$$C = A \cap B \Leftrightarrow \mu_c(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

Set-Theoretic Operations



$$\mu_A \leq \mu_B$$

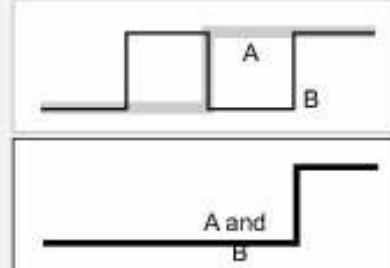


$$\max(\mu_A(x), \mu_B(x))$$

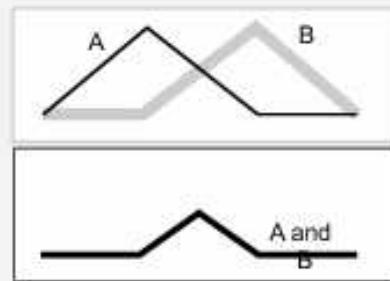
$$\min(\mu_A(x), \mu_B(x))$$

Fuzzy Logical Operation

Two-valued logic



Multivalued logic



AND
 $\min(A,B)$

T-norm

OR
 $\max(A,B)$

S-norm/T-conorm

NOT
 $(1-A)$

MF Formulation

Triangular MF:

$$trimf(x; a, b, c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right)$$

Trapezoidal MF:

$$trapmf(x; a, b, c, d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right)$$

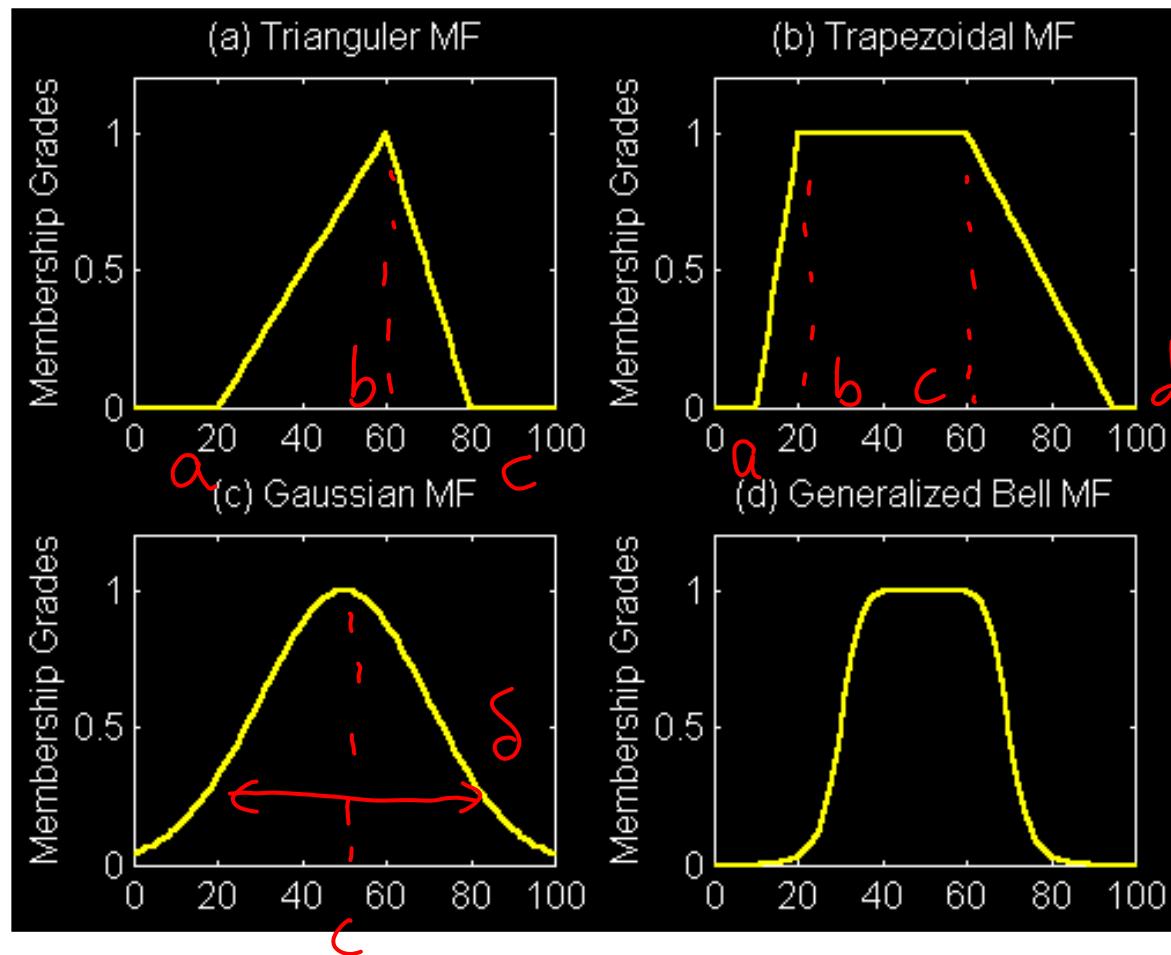
Gaussian MF:

$$gaussmf(x; a, b, c) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

Generalized bell MF:

$$gbellmf(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{b}\right|^{2b}}$$

MF Formulation



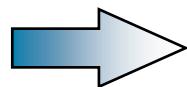
MF Formulation

Sigmoidal MF:

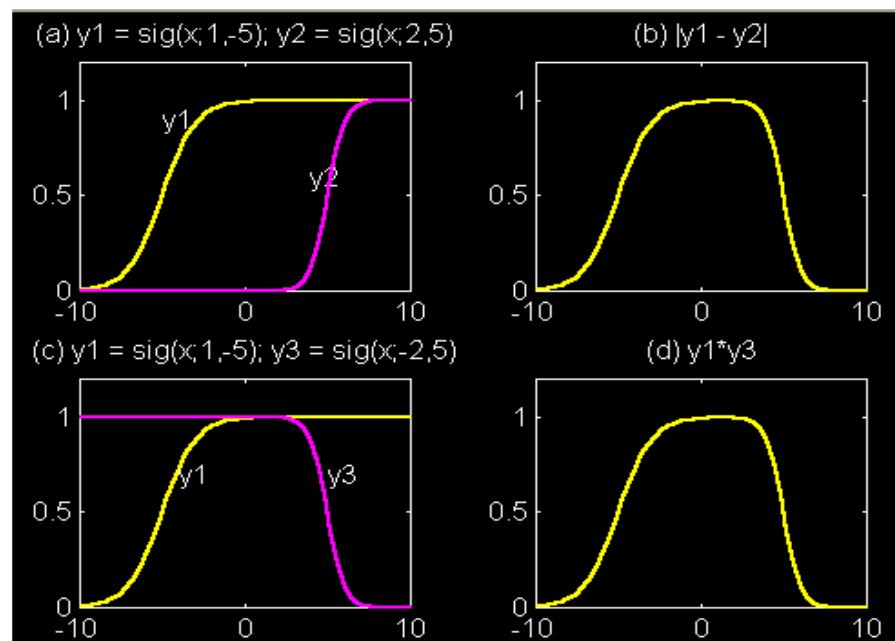
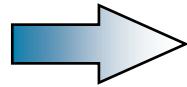
$$\text{sigmf } (x; a, b, c) = \frac{1}{1 + e^{-a(x-c)}}$$

Examples:

Absolute difference
of two sig. MFs



Product of two sig. MFs



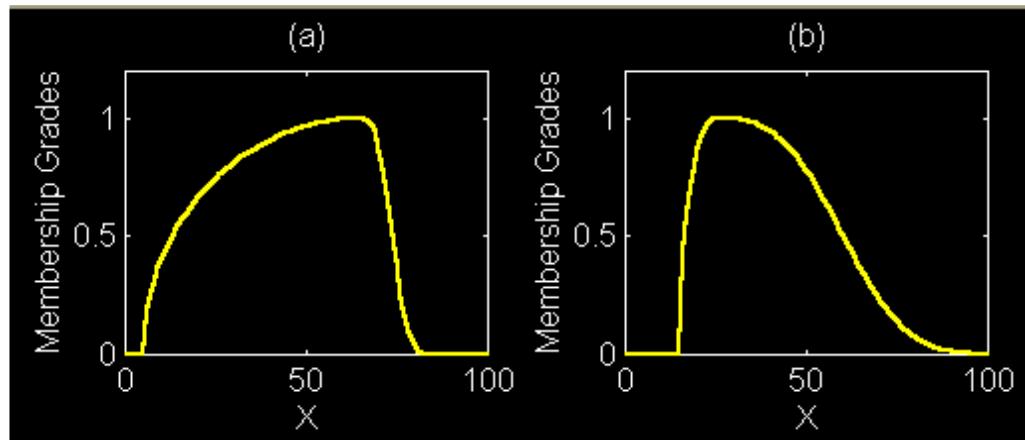
MF Formulation

Left-Right MF:

$$LR(x; c, \alpha, \beta) = \begin{cases} F_L\left(\frac{c-x}{\alpha}\right), & x < c \\ F_R\left(\frac{x-c}{\beta}\right), & x \geq c \end{cases}$$

Example: $F_L(x) = \sqrt{\max(0, 1-x^2)}$ $F_R(x) = \exp(-|x|^3)$

c=65
a=60
b=10



c=25
a=10
b=40



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CZ3005 Artificial Intelligence

Week 12b – Fuzzy Logic

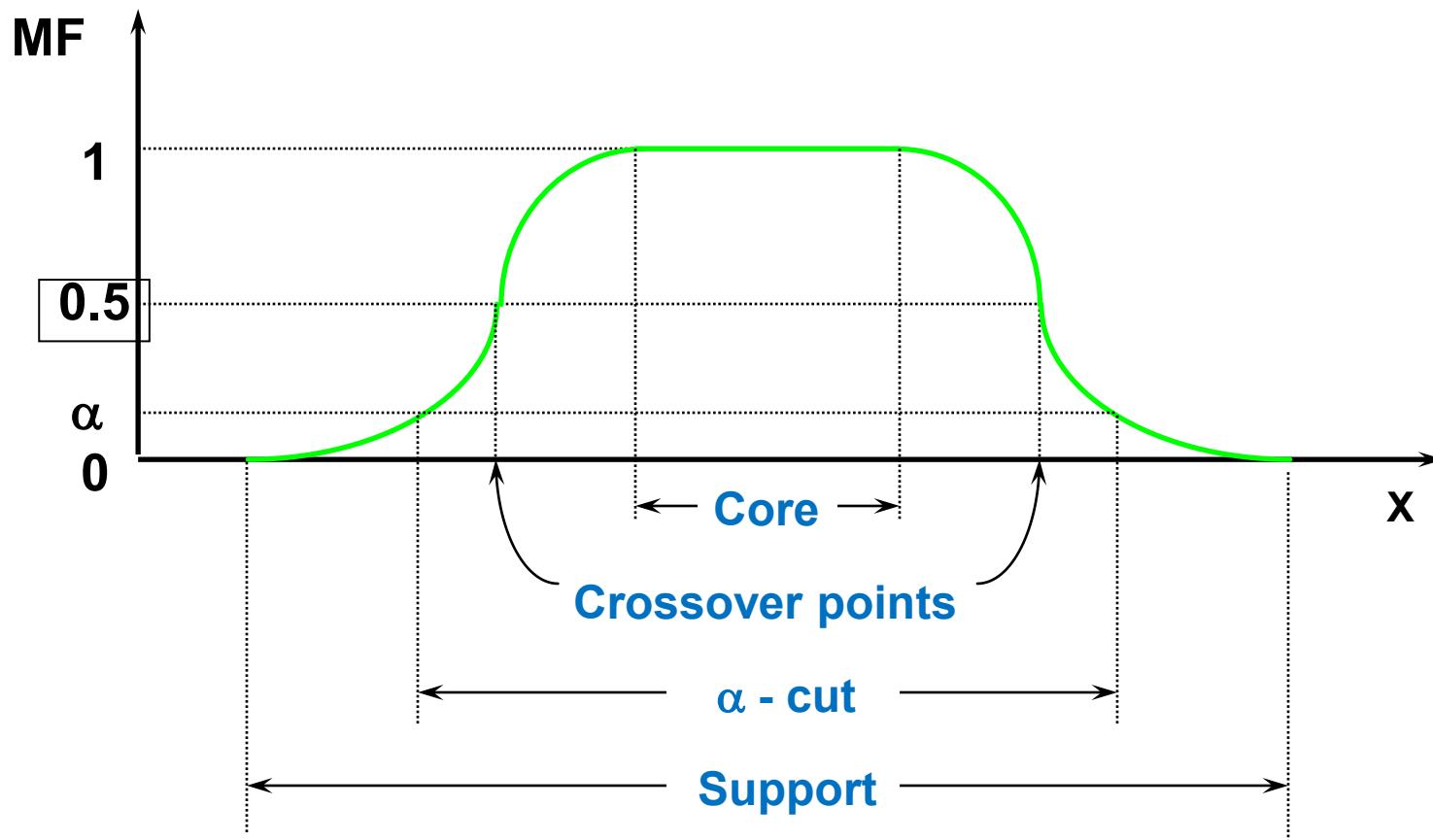
Yu Han

han.yu@ntu.edu.sg

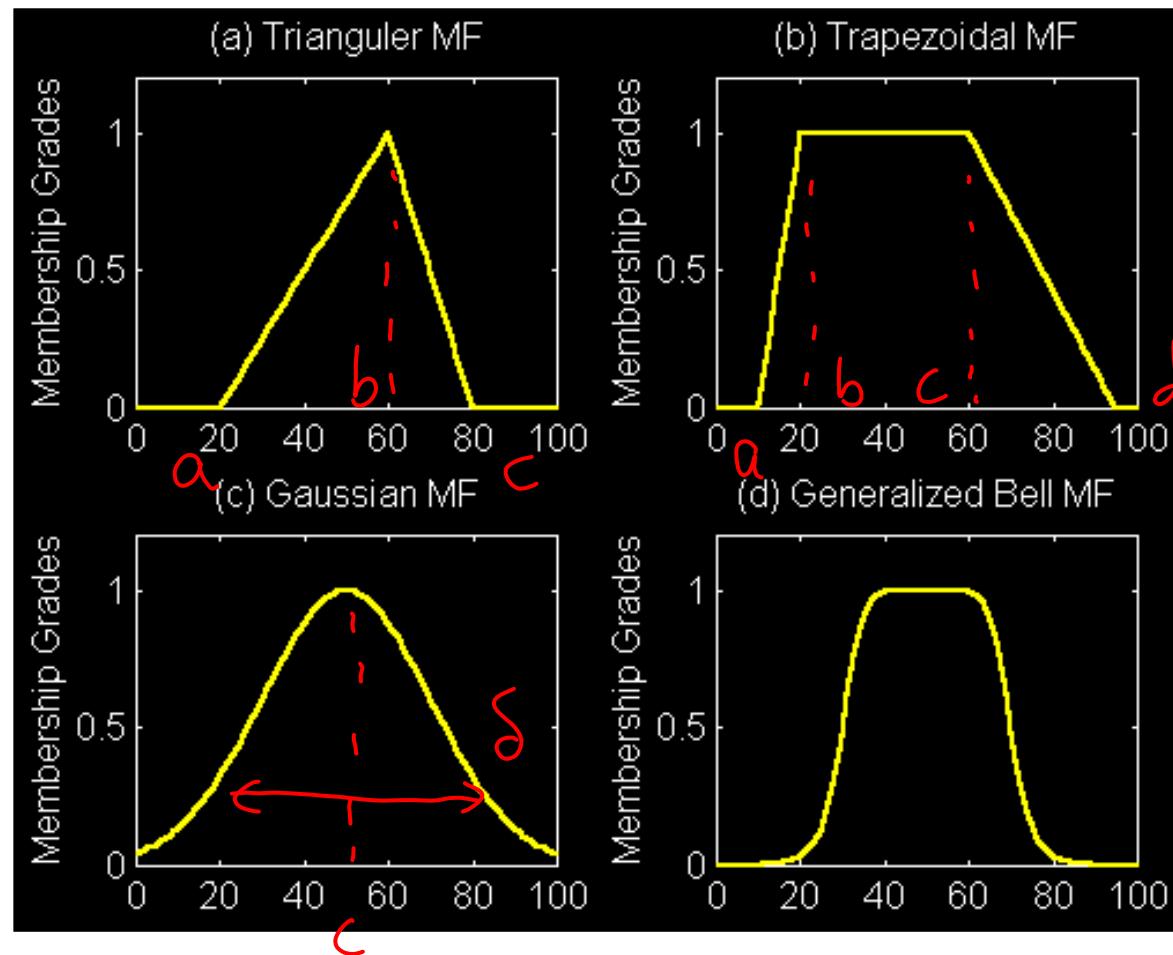
*Nanyang Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University*



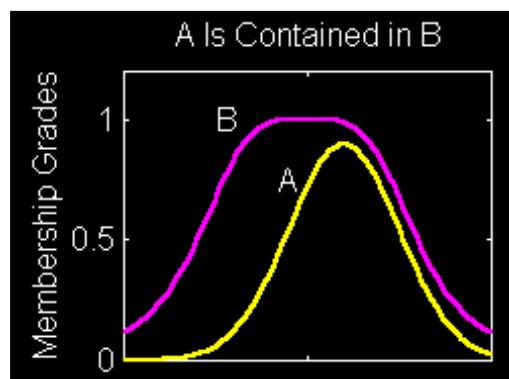
Recap – MF Terminology



Recap – MF Formulation

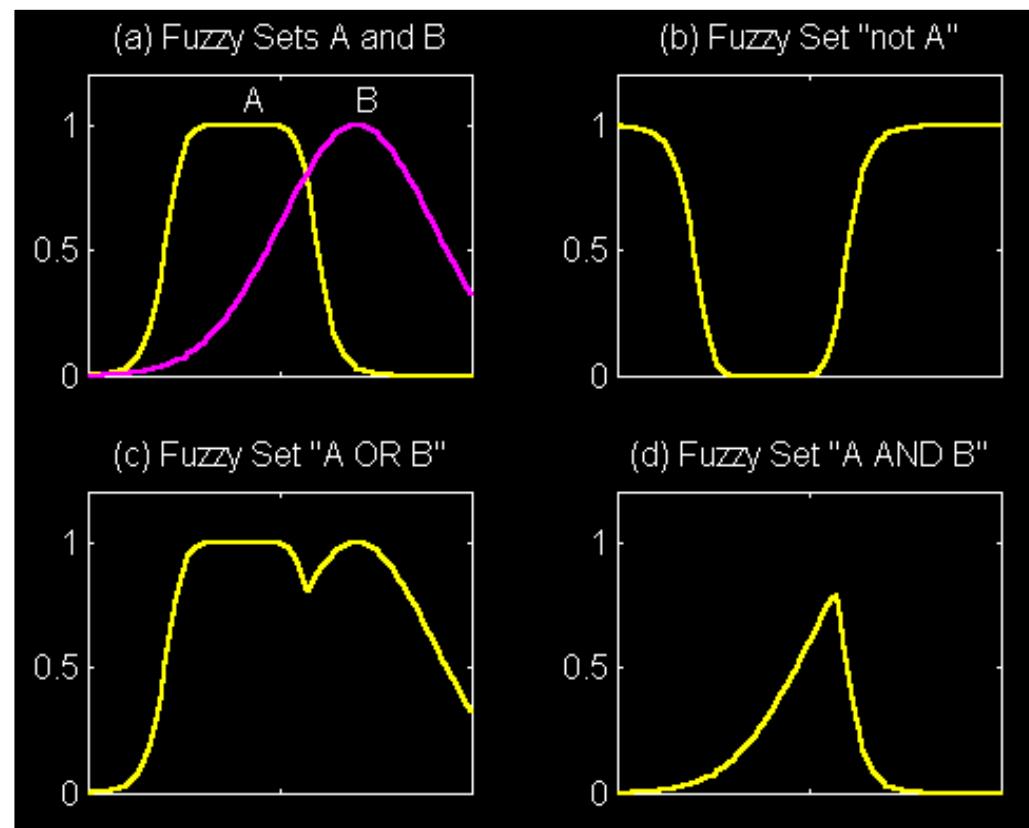


Recap – Set-Theoretic Operations



$$\mu_A \leq \mu_B$$

Subset



$$\max(\mu_A(x), \mu_B(x))$$

Union

$$1 - \mu_A(x) \quad \text{Complement}$$

$$\min(\mu_A(x), \mu_B(x))$$

Intersection

Learning Goals

Understanding the:

- Linguistic modifier/hedges
- More on fuzzy union, intersection, and complement
 - Fuzzy complement
 - Fuzzy intersection and union
 - Parameterized T-norm and T-conorm
- Fuzzy Rule Based System
 - Fuzzy Rule, Fuzzy Inference, and defuzzification

Linguistic Hedge - Modifiers

- Linguistic Hedge/Modifiers are operations that modify the meaning of a term – fuzzy label (fuzzy set).
 - “*very Tall*”, the word *very* modifies “*TALL*” which is a fuzzy set.
- Other modifiers are:
 - “*more or less*” (morl), “*possibly*”, and “*definitely*”

Linguistic Hedge – Modifiers

- very $a = a^2$
- morl $a = a^{0.5}$
- extremely $a = a^3$
- slightly $a = a^{0.333}$
- somewhat $a = \text{morl } a \text{ and not slightly } a$

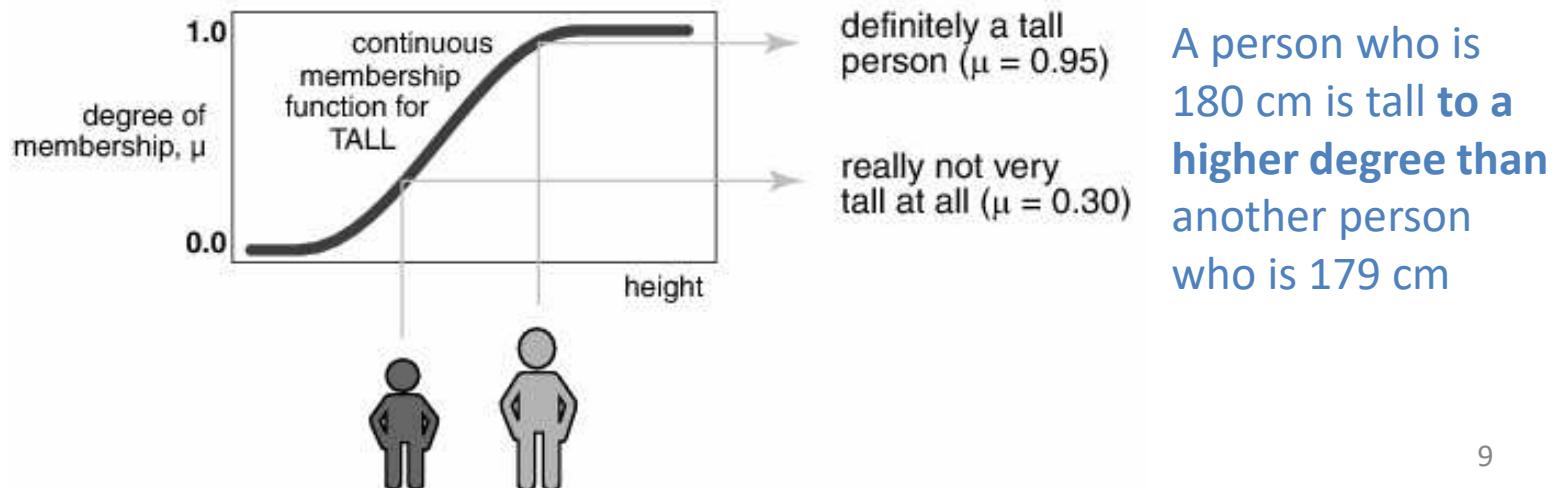
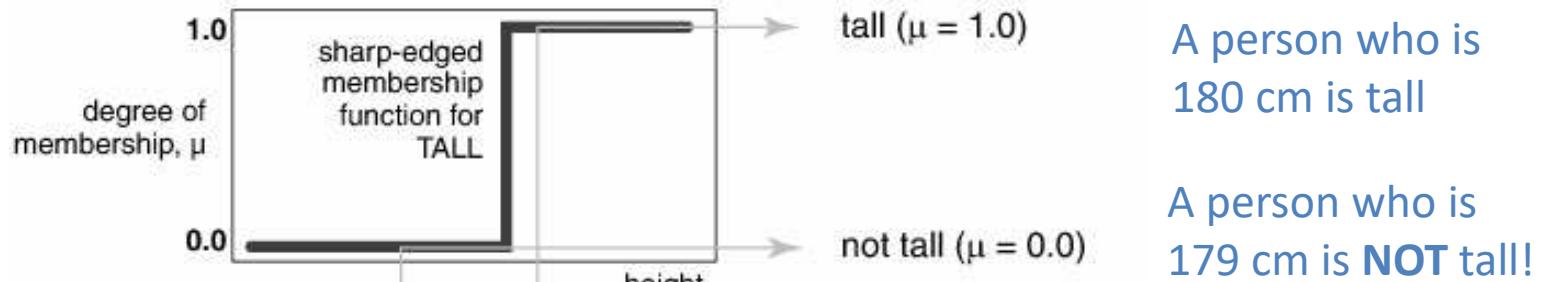
E.g., young = [1/0, 0.6/20, 0.1/40, 0.0/60, 0.0/80]

very young = young^2
= [1/0, 0.36/20, 0.01/40, 0.0/60, 0.0/80]

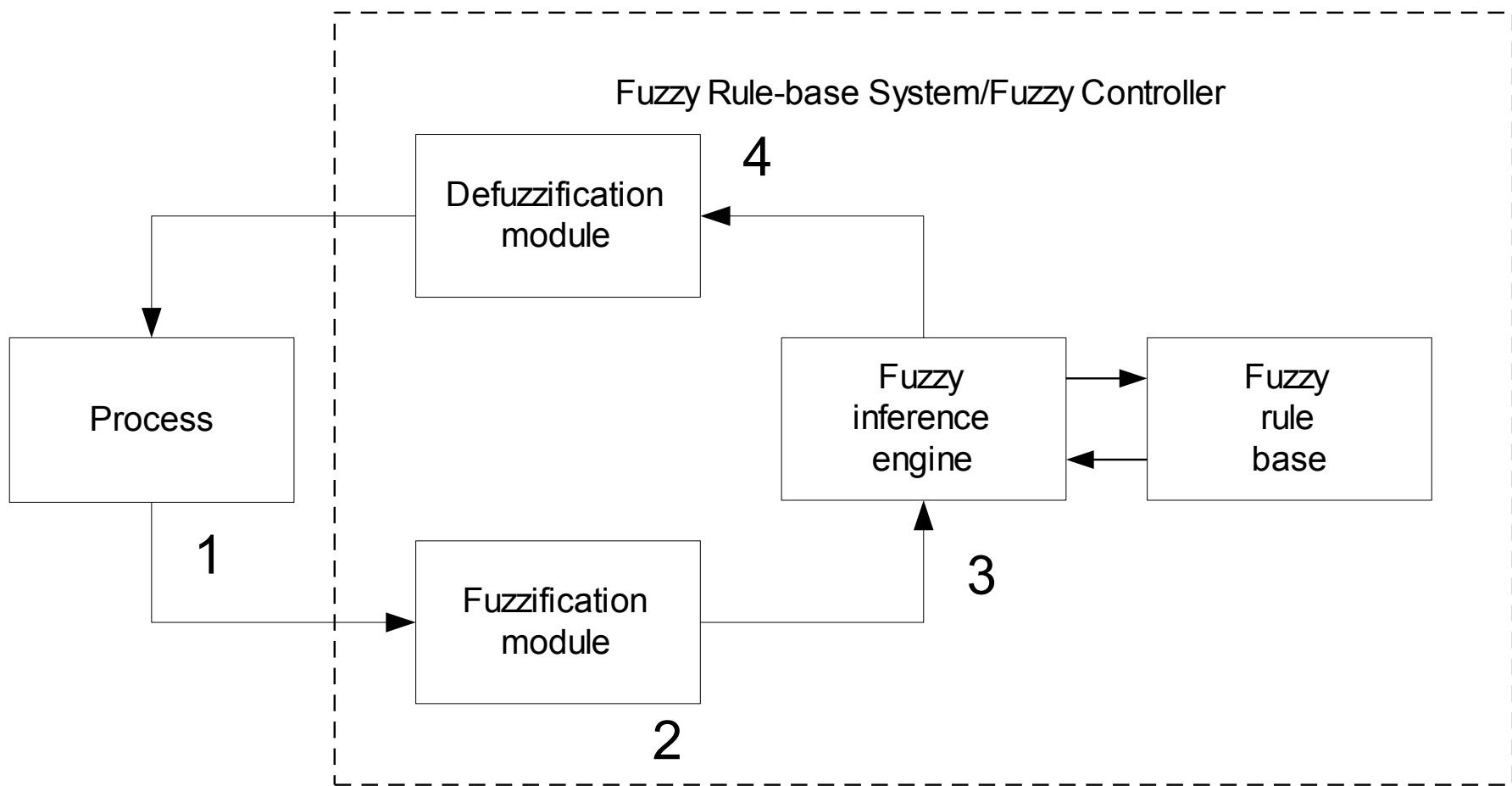
Summary of Membership Functions

- Fuzzy sets allow the description of **vague concepts** (e.g., *SLOW*, *MEDIUM* and *FAST*) for a fuzzy variable (e.g., *SPEED*)
- This provides the semantics (concepts) to linguistic rules involving fuzzy variables.
e.g., The *SPEED* is *FAST*.
- The fuzzy set admits the possibility of **partial memberships** in it. (e.g., Friday is sort of a weekend day, the weather is rather hot).

Semantic Meaning & Partial Membership



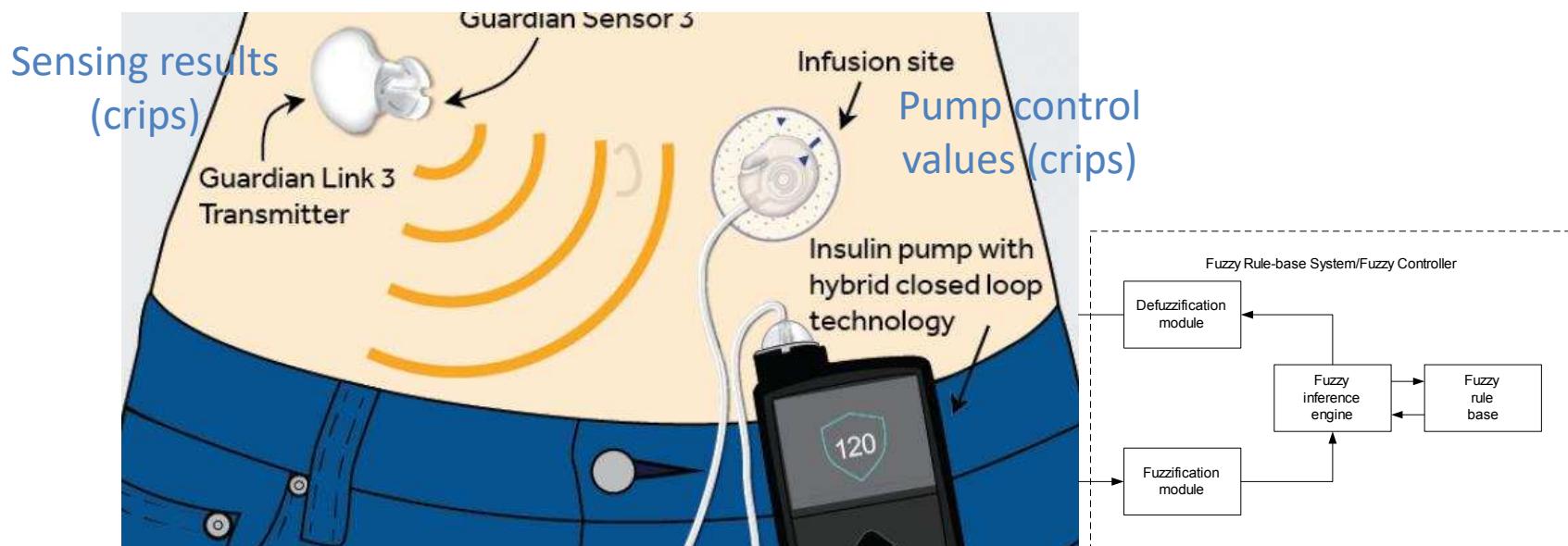
Fuzzy Rule-Based (FRB) Systems



Operation of FRB Systems

1. Firstly, measurements are taken of all variables from the process.
2. Next, these measurements are converted into appropriate fuzzy sets to express measurement uncertainties - **fuzzification**.
3. The fuzzified measurements are then used by the inference engine to evaluate the control rules stored in the **fuzzy rule base**. - (fuzzy rules defined with fuzzy (linguistic) variables using fuzzy labels – fuzzy sets)
4. The result of this evaluation is one or several fuzzy sets defined on the universe of possible control actions. This fuzzy set is then converted, in the final step of the cycle, into **a single crisp value** or **a vector of values** which best represents the resulting fuzzy set or sets – **defuzzification**.

Example FRB Continuous Insulin Pump



Components of a Fuzzy Rule

- A single fuzzy if – then rule:
 - If x is A then y is B .

fuzzy antecedent fuzzy conclusion

A and B are linguistic values defined by fuzzy sets on the range of discourse for fuzzy variables x and y .

E.g., If *service* is *good* then *tips* is *average*

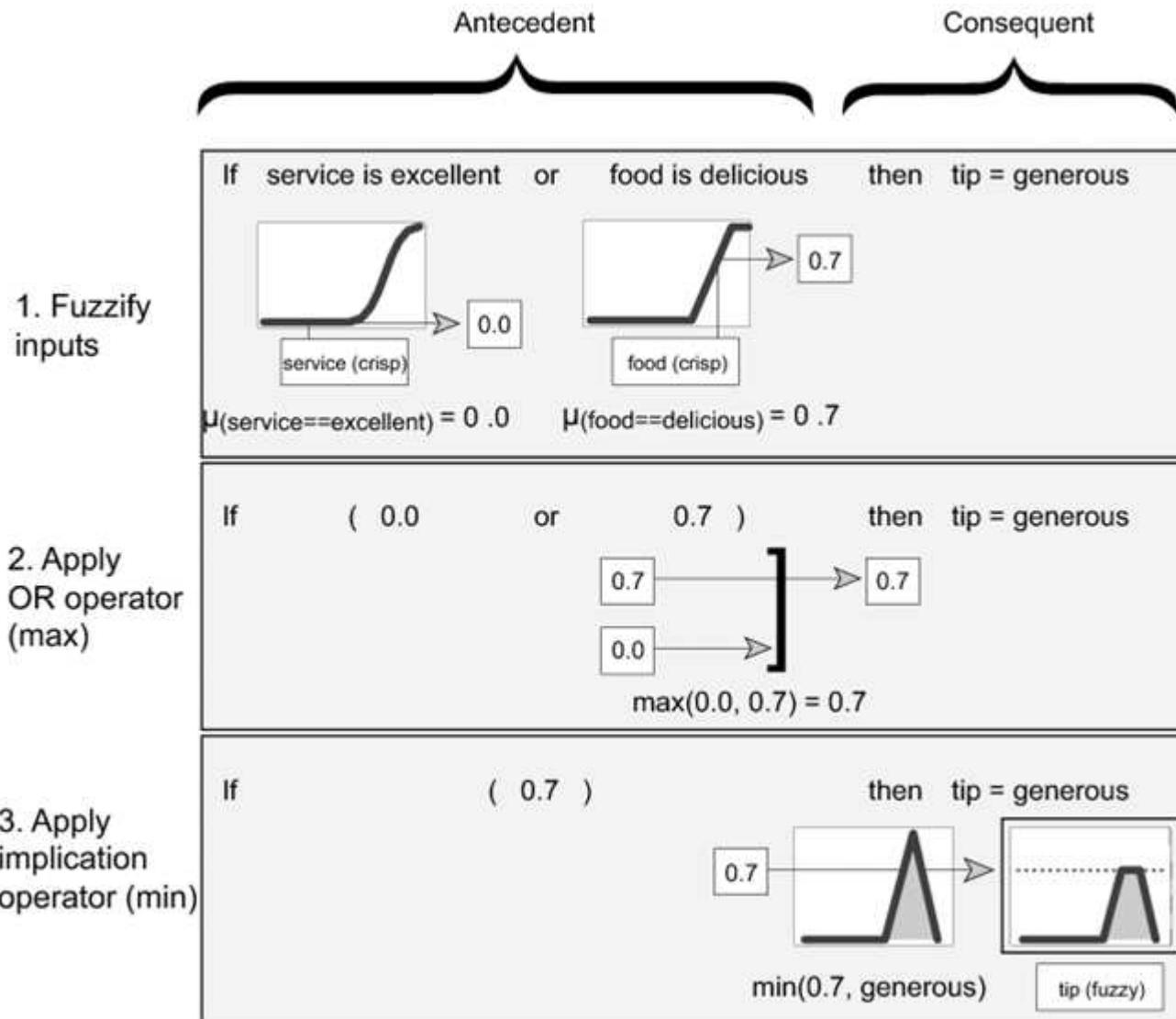
x A y B

Interpreting fuzzy if-then rule:

- Evaluating antecedent (*fuzzifying input and necessary fuzzy operators*)
- Applying the result to the conclusion/consequent (*known as implication*)



Fuzzy Inference – Example



Steps of Fuzzy Inference

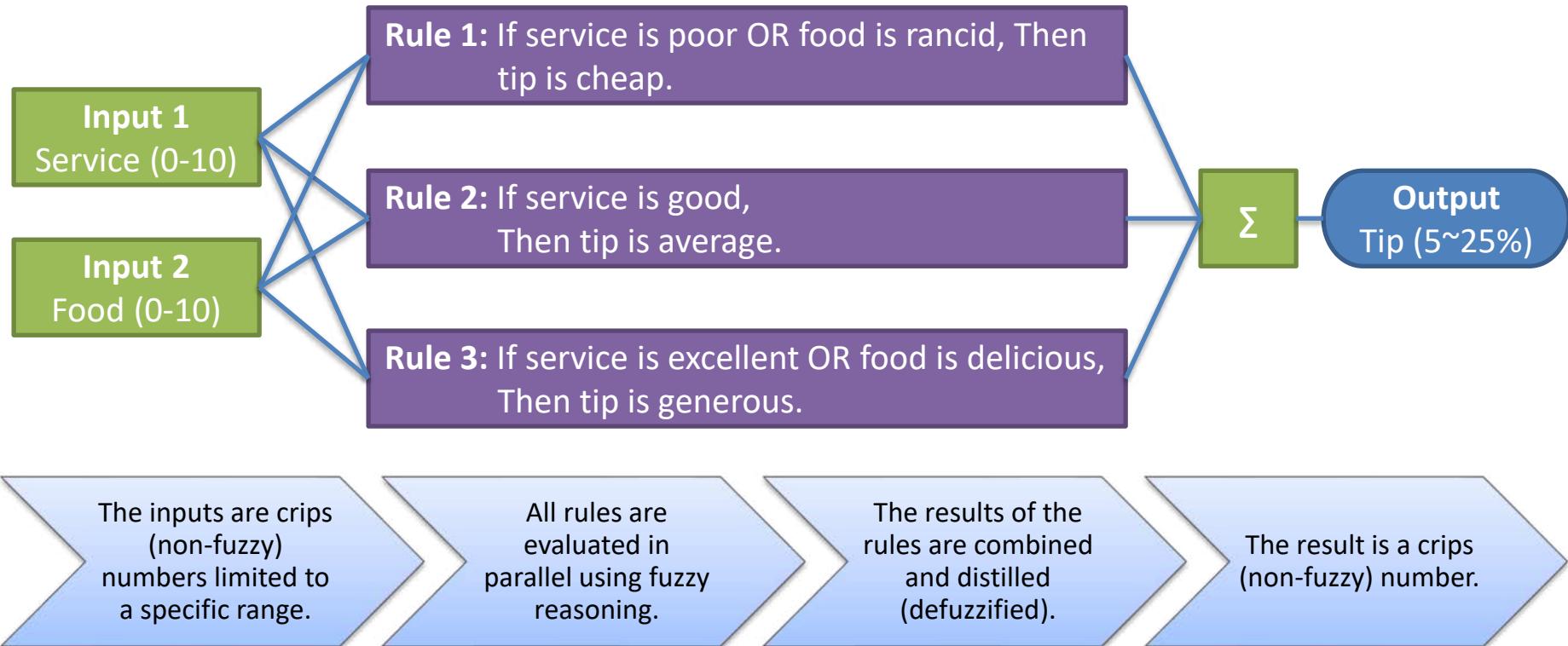
1. **Fuzzify inputs:** Resolve all fuzzy statements in the antecedent to a degree of membership between 0 and 1
2. **Apply Fuzzy Operators:** If there are multiple parts to the antecedent, apply fuzzy logic operators (AND, OR, etc.) and resolve the antecedent to a single number between 0 and 1.
3. **Apply Implication Method:** Use the degree of support for the entire rule to shape the output fuzzy set. The consequent of a fuzzy rule assigns an entire fuzzy set to the output. This fuzzy set is represented by a membership function that is chosen to indicate the qualities of the consequent.
4. **Aggregation of the consequents across all rules (if there are multiple rules).**
5. **Defuzzification**

Fuzzy Inference – Cont'd

- Fuzzy inference is the process of formulating the mapping from a given input to an output using fuzzy logic.
- The process of fuzzy inference involves all of the pieces that are described in the previous sections:
 - Membership Functions
 - Logical Operations
 - If-Then Rules

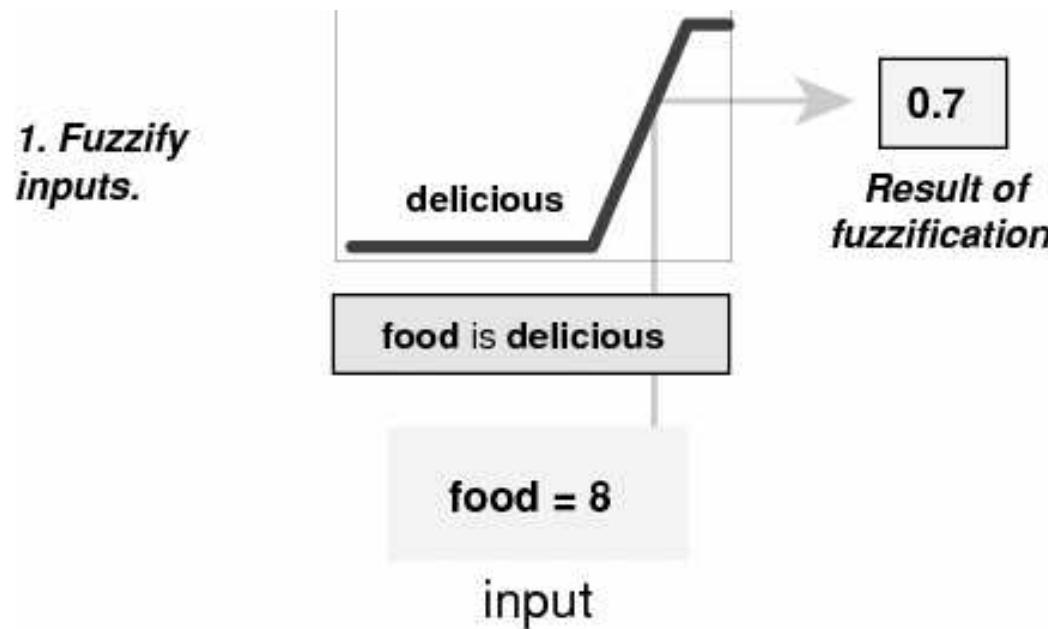
Example – Multiple Rules

Consider the example of the service for a dinner for two



Step 1: Fuzzification

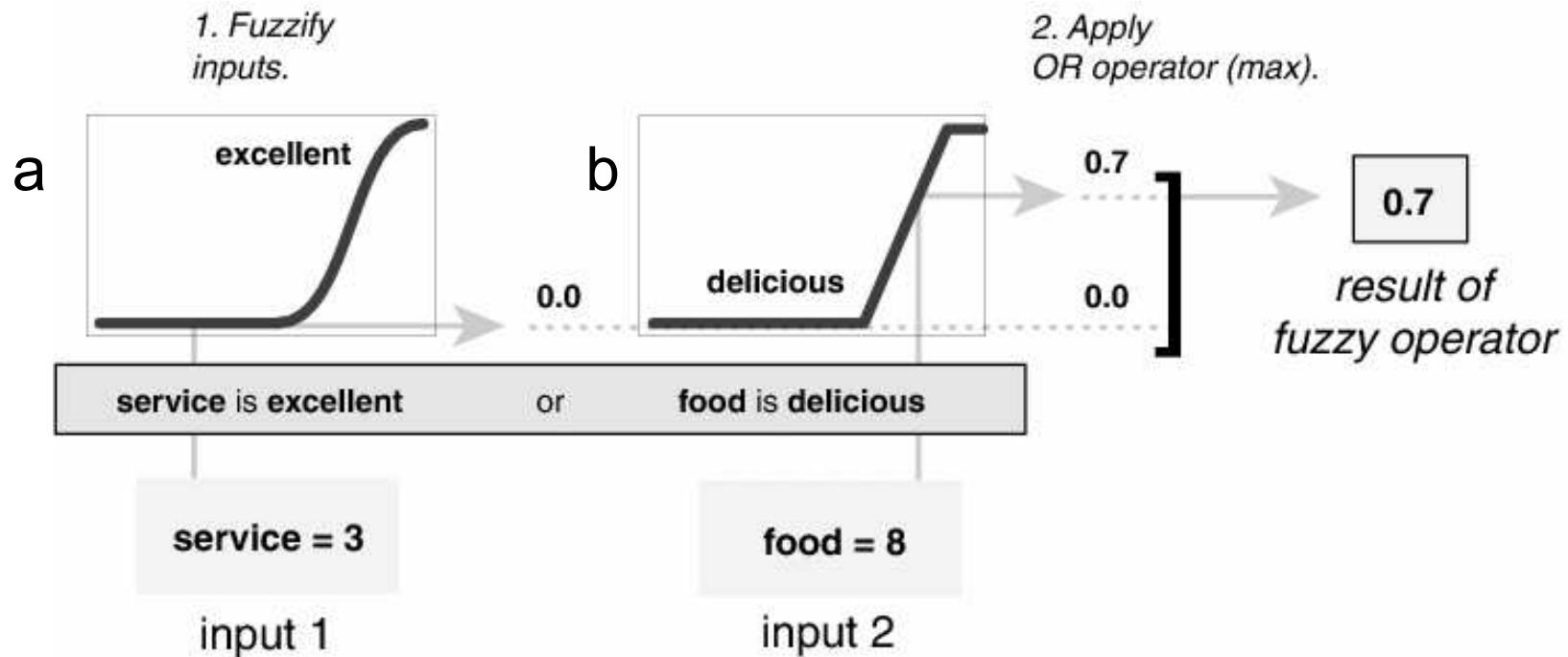
- Fuzzification of the input amounts to either a table lookup or a function evaluation.



- In this manner, each input is fuzzified over all the qualifying membership functions required by the rules.

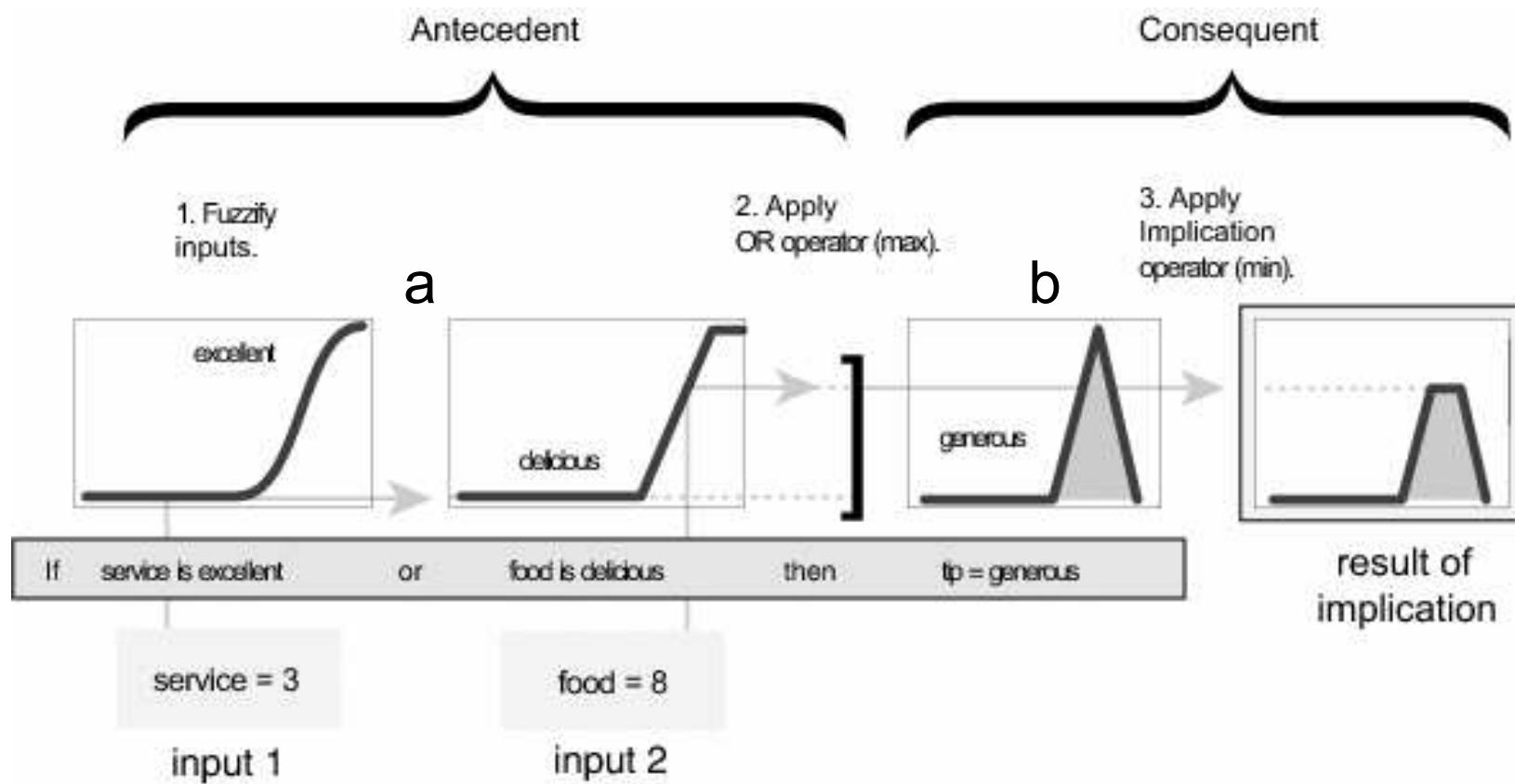
Step 2: Applying Fuzzy Operators

- OR - $\max(a, b)$



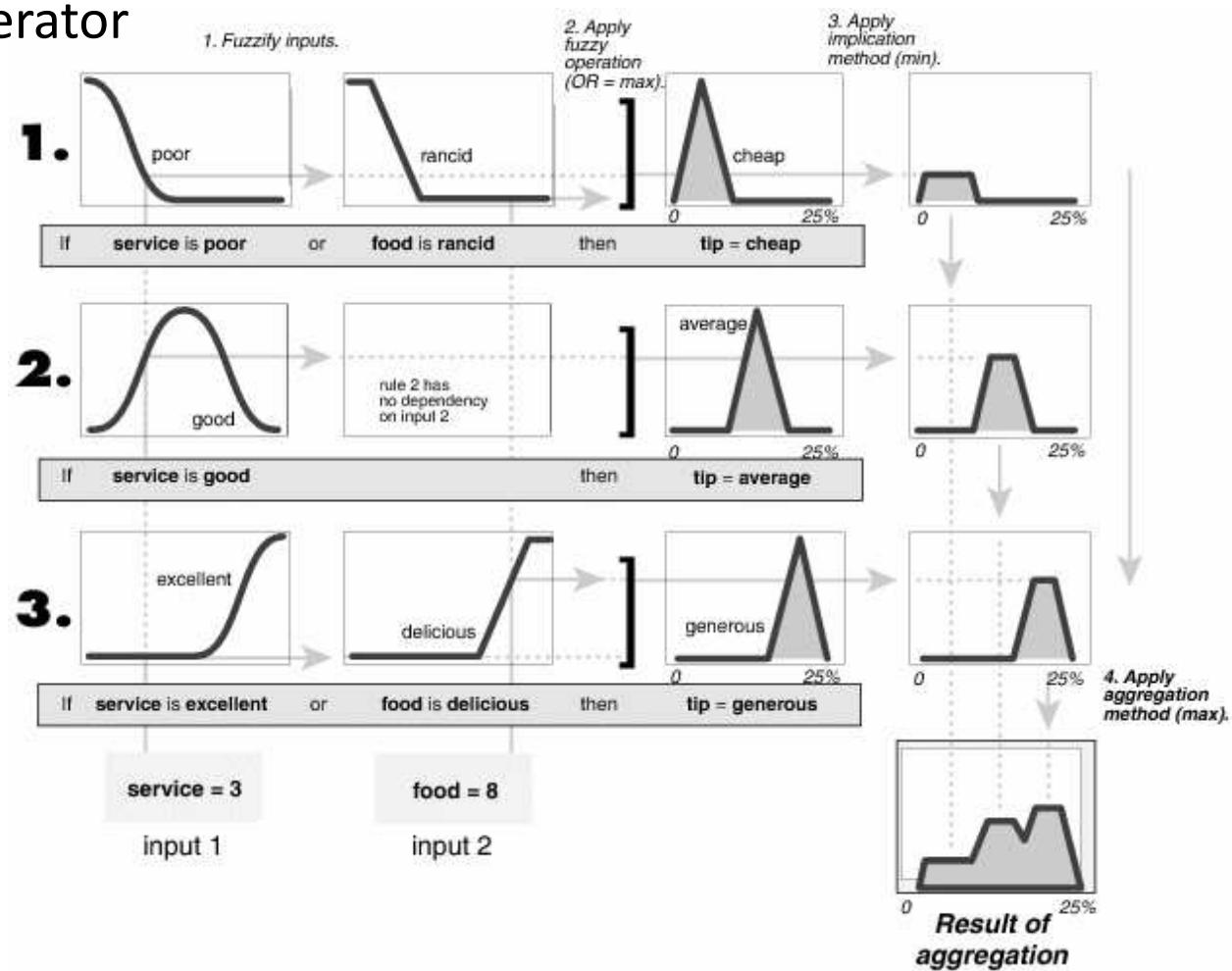
Step 3: Applying Implication Method

- $\min(a, b)$, Rule 3:



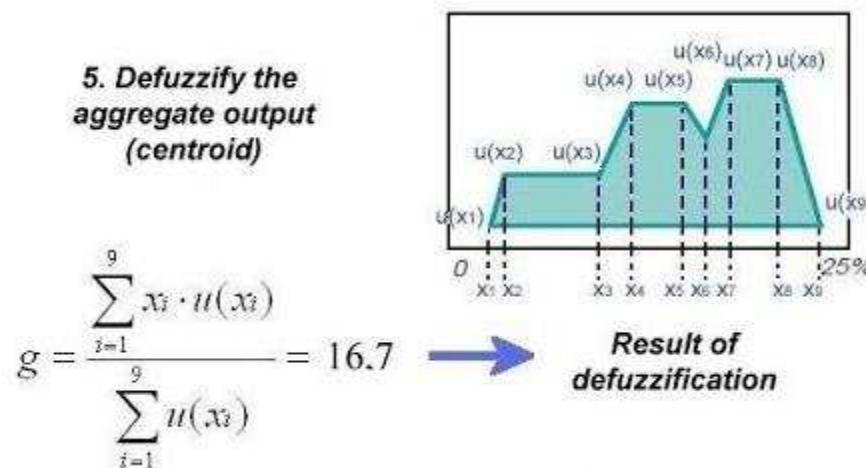
Step 4: Aggregating All Outputs (3 Rules)

- Aggregation is the process by which the fuzzy sets that represent the outputs of each rule are combined into a single fuzzy set.
- E.g., max operator



Step 5: Defuzzification

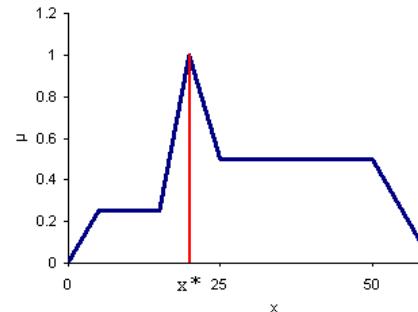
- The input for the defuzzification process is a fuzzy set (the aggregate output fuzzy set)
- The output is a single number.



- The final tip to pay for a rating of service=3 and food=8 is 16.7%.

Types of Defuzzifications

- Max-membership $\sup_x(\mu(X))$ defuzzification method



- Centroid defuzzification

$$x^* = \frac{\int \mu_i(x) x \, dx}{\int \mu_i(x) \, dx}$$

- Weighted Average Defuzzification Technique

$$x^* = \frac{\sum_{i=1}^n m^i w_i}{\sum_{i=1}^n m^i}$$