CZ3005:

Module 2: Intelligent Agent & Search

1. Types of Environment

Accessible?: Have access to complete state of environment

Deterministic?: Next state of environment completely determined by current state & actions selected by the agent

Episodic? | Sequential?: Each episode is not affected by previous actions taken

Static? Dynamic?: Env^{n} Δ when agent is deliberating?

discrete? continuous: Limited # of distinct percepts & actions

Design of Problem-Solving Agent → 1. Goal Form^{n} 2. Problem Form^{n} 3. Search process ⇒ uninformed / informed 4. Action Execution.

2. What is a well defined problem Formulation.

Def of problem: info used by agent to decide what to do.

Specification: → Initial state.

→ Action state (i.e avail actions (successor func)) → cost Func^{n}: path cost. g(n), Σ all action step cost along path

→ State space (i.e states reachable from initial state)

· Sol^{n} path: seq. of action from one state to another

→ Goal state predicate

· single state enumerated list of states, abstract properties.

Solution: path (seq. of operators) from initial State to State that satisfies goal-test.

Frontier (# Candidates nodes for expansion)

2. Search strategies

completeness: does it always find sol^{n} if one exists

Time Complexity: How long? # of nodes expanded (Explored Nodes)

Space Complexity: max # nodes in mem. (in Explored Nodes)

optimality: Does it always find the best (least-cost) sol^{n}?

b: max branching Factor of search tree

d: depth of least cost sol^{n}

A) Uninformed Search

	What?	Time	Space	Complete	optimal
BFS	Expand shallowest unexpanded node (FIFO)	$O(b^d)$	$O(b^d)$	Yes	Yes, when cost are =
UCS	FIFO w priority queue to path cost g(n) to order elements	# nodes w path cost $g \leq$ cost of optimal sol^{n}	=	Yes	Yes
DFS	Expand deepest unexpanded node in LIFO stack + backtrack	$O(b^m)$. If sol^{n} dense, faster than BFS	$O(bm)$	infinite depth: No finite depth with loops: No · with repeated state check: Yes finite w/o loops	No
Depth Limited	DFS with cutoff in max depth of a path	$O(b^l)$	$O(bl)$	Yes, if $l > d$	No

· Systematic generation of new states (→ Goal test)

· Inefficient (Exponential space & time complexity)

B) 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 eval^{n} fun^{n} to estimate desirability of each node.

Types of eval Fun^{n}

1. Path cost fun^{n}: g(n)
   - cost from initial state to search node (n)
   - No info on cost towards goal
   - Need to estimate cost toward goal

2. Heuristic f(n)
   - Estimated cost of cheapest path from n to goal state h(n)
   - Exact cost cannot be determined
   - depends only on state at that node
   - h(n) < Real cost

ucs uses:
g(n): cost to reach n (Past Exp)
Inefficient but optimal & complete

→ cost from n to goal (Future pred^{n})

	What?	Time	Space	Complete	Optimal
Greedy:	Expand node closest to goal h(n) at eval func: E.g straight line dist from n to Bucharest	$O(b^m)$	$O(b^m)$	No	But ↓ search space ⟷ No
A*	combines Greedy & ucs: Eval. f(n) = g(n) + h(n) Estimated total cost of path through n to goal	Exponential in len of sol^{n}	All generated nodes in mem Exponential in len sol^{n}		

Module 3: Constrain Satisfaction & Adversarial Search

Constrain Satisfaction problem

State: defined by variables, V_i with values from Domain, D_i

constrains - **Goal test:** a set of constraints specifying allowable combinations of values for subsets of variables.

Goal: discover some state that satisfies a given set of constrains.

Consistent | legal Assignment: assignment that don't violate any constrains

Solution: Assignment with every variable given a value | complete & assignment satisfies all constrains.

Initial state:

* use search strategies to find sol^{n} to csp

→ Each state is a node in the tree

→ Solutions are at bottom of tree, use DFS

Possible heuristics for CSP

1. Order to assign variables
2. Order of values to try for each variable
3. Implications of current variable assignment to other unassigned variables

- (i) **Forward checking:** propagates info from unasigned to assigned values, keeps track of remaining legal values for unassigned var & terminates search when any var has no legal values ⇒ prevents unnecessary expansion of nodes when the node has no more possible values & ↑ speed of search.
- (ii) **constraint prog^{n}:** prog^{n} of implications of a constraints on one var onto other var, whereby nodes are expanded if they don't violate the constraint

WAYS to solve CSP Problem

1. **Standard search** - uninformed Algo. e.g. backtracking.

→ IDEA: do not waste time searching when constraints have already been violated.

- Goes back to previous node if current node violates some constrains.

2. **Most constraining variable:**

- ↓ branching factor of future choices by selecting var that is involved in the largest # of constrains on unassigned var
- ↓ domain of many other unassigned var.

3. **Least constraing variable**

- choose the value that leaves max flexibility for subsequent var assignment

4. **Min conflict Heuristics**

- hive an initial random assignment, select var that violate constrain, Δ assign^{n} of var to value than min # of violated constrains
- local heuristic search, may not lead to sol^{n}

Adversarial Search - Games as Search problems.

- Abstraction : Ideal rep^{n} of real world
  Accurate formulation: state space rep^{n}
- complexity: Abstract by not simple.
- time limited: complete search difficult → uncertainty on actions desirability.

Types

	deterministic (↱ Randomness in game)	Chance
Perfect Info (each player has complete info about his opponent pos^{n} & about choices avail to him)	chess, checkers, Go, othello	Backgammon, Monopoly
Imperfect Info.		Bridge, Poker, Scrabble, Nuclear war

Min-max Search Strategy.

→ Maximise one's own utility & minimize the opponent's. Assump^{n}: opponent does the same.

3 step.

1. Generate entire game tree down to terminal states
2. (Calc utility.)
   A. Assess the utility of each terminal state
   B. Determine the best utility of parents of terminal state
   C. Repeat the process for parent until root is reached
3. Select the best moves (i.e move with highest utility)

(Assumes perfect decisions: No time limit)

For Imperfect Decisions.

i.e Time & space requirements → complete game tree search is intractable → impractical to make perfect decisions

Modif^{n} to Min-max Algo

1. Replace utility function by Eval^{n} func (An estimated desirability of pos^{n}) (expected utility of game from given pos^{n})

Requirements of Eval^{n} fun^{n}
- Computationally Efficient
- Agrees w utility func on terminal states
- Accurately reflects chances of winning

(A tradeoff btw accuracy & efficiency.)

2. Partial Search tree.
   E.g. depth-limit
   - Replace terminal test by cut-off test

* Quiescent search
→ eval func only applied to quiescent pos^{n}
↳ pos^{n} that are not likely to have a large variation in eval in the near future → pretty confident w eval func^{n}

Module 3: Markov Decision Process.

MDP components: $\{S, T, A, R\}$ (S: State, T: transition func, A: Action, R: Reward)

Solution: policy: $\pi(s)$ mapping from states to actions

optimal policy max accumulated rewards over trajectory

$\tau = \langle s_1, A_1, R_1, \ldots S_T, A_T, R_T \rangle$

$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \leq \frac{R_{max}}{1-\gamma}$ $(0 < \gamma < 1)$

Value Function: $V^{\pi}(s)$ ⇒ Ave of multiple trajectories.

- Refers to expected sum of discounted rewards if agent executes optimal policy π starting in state s

$V^{\pi}(s) = E_{\pi}[G_t | S_t = s]$

Action value of state action pair : $Q^{\pi}(s,a) = E_{\pi}[G_t | S_t = s, A_t = a]$

$V^{\pi}(s) = \sum_{a \in A} Q^{\pi}(s,a)\pi(a|s)$

Bellman Eqn.

$Q(s,a) = \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma * V(s')]$ (γ: discount factor; V(s'): Value of state(s'))

- Q: value of taking action a in state s
- $P(s'|s,a)$: prob of transitioning to state s' from s after taking action a
- $r(s,a,s')$: immediate reward after taking action a in state s & transitioning to s'

To choose optimal policy, choose action that max Q value for each state

optimal action $\pi^*(s) = \text{argmax}_a Q(s,a)$

$V(s) = \max_{a \in A} Q(s,a)$

Optimal value function V^* is derived from optimal Q values by taking the max over all possible actions

Value Iteration Function

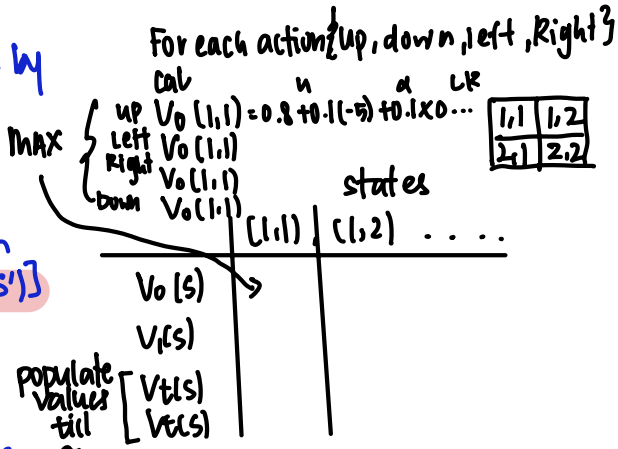
Initialise $V_0(s)$ for all s

For $i = 0, 1, \ldots$ $V_{i+1}(s) = \max_a \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$ ($Q_{i+1}(s,a)$)

Repeat this process until converges.

- use V^* to find optimal policy/action in each state.

$\pi^*(s) = \text{argmax}_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_i(s')] = \text{argmax}_a Q_i(s,a)$



Policy Iteration

Start with some initial policy, π_0 & alternate btw:

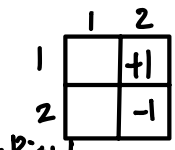
- Policy evaluation: calculate $V^{\pi_i}(s)$ for every state, s

$$V^{\pi_i}(s) = \sum_{s'} P(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V^{\pi_i}(s')]$$

- Policy improvement: calculate new policy π_{i+1} based on update utilities

$$\pi_{i+1}(s) = \underset{a \in A}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^{\pi_i}(s')]$$

until process converges



→ Initial Policy $\pi(S_{11})$ = Right
$\pi(S_{21})$ = Right

→ Policy Eval
$V(S_{11})$ =
$V(S_{21})$

$\pi(S_{11})$ = Right
π(Right) = 0.79
π(Left) 0.775 } max
⋮

* Algo

1. Initialize V(s) & π(s) for all states s
2. policy Evaluation (calculate the V)
   a. Repeat
   b. Δ ← 0
   c. For each state s:
   d. v ← V(s)
   e. V(s) ← $\sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V(s')]$
   f. Δ ← max(Δ, |v − V(s)|)
   g. until Δ < w (small threshold)
3. Policy Improvement (calculate new policy π)
   a. policyStable ← True
   b. For each state s:
   c. b ← π(s)
   d. π(s) ← $\operatorname{argmax}_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$
   e. if b ≠ π(s), then policyStable ← False
   f. if policyStable ← True, then stop, go to step 2

① calculate

	value
states s_0	V(s₀) = 8.03
s_1	V(s₁) = 11.2
s_2	V(s₂) = 8.9

② calculate optimal policy

π^*	a_0	a_1	Actions
s_0	5	6	
s_1			
s_2			

⇒ optimal policy
$\pi(s_0) = a_1$
$\pi(s_1) = 11$
$\pi(s_3) = 11$

Module 5: Reinforcement Learning

Reinforcement Learning: learn value func^{n} (policy) w/o the transition function $P(S'|s,a)$ → don't know what happens after taking an action

1. Monte Carlo (use of randomness)

- Solve of problem by generating suitable random numbers & observing the fraction of numbers by obeying some properties

→ only for evaluating & finding $V(S)$ = expected cumulative reward for states for given policy.

* MC for estimating V(s)

→ using some policy, calculate accumulated rewards for states in each episode

→ $V(S)$ is the ave of accumulated rewards over all episodes.

First visit MC policy Eval^{n}

- Initialize
  - $\pi \leftarrow$ policy to be evaluated.
  - $V \leftarrow$ an arbitrary state-value fun
  - Returns(s) $\leftarrow$ an empty list for all states
- Repeat many times.
  - Generate an episode with π
  - For each state appearing in the episode
    - $R \leftarrow$ Return first occurrance of s
    - Append R to Returns(s)
    - $V(S) \leftarrow$ ave (Returns(s))

$\frac{1}{i}\sum_{i} G_t$ → i : # of episodes for state S_j

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

$= \sum_{i} \gamma^i r_{t+1}$

* MC for estimating Q-values.

→ objective: find optimal policy by estimating action-value func $Q(S)$

→ calculate accumulated reward for state action pairs instead of state. Ave acc^{n} rewards across all episodes.

	S_1	S_2	S_3
a_1	Q_{11}	Q_{21}	Q_{31}
a_2	Q_{12}	Q_{22}	Q_{33}

	cell 1	cell 2	cell 3
R_t	8.1	9.3	9.9
U_t	1	5	4

✓ At cell 1,2,3 choose R_t.

→ Selecting an action: $\pi'(s) = \underset{a \in A}{\operatorname{argmax}} Q^{\pi}(s,a)$

mc-control Algo

$Q_\pi[s,a] \sim \frac{1}{N}\sum_{i=1}^{N} G_{i,s}$

- Initialize
  - $Q(s,a) \leftarrow$ arbitrary
  - Returns$(s,a) \leftarrow$ empty list → small #
  - $\pi \leftarrow$ an arbitrary ε-soft policy (random policy)
- Repeat many times.
  - (a) Generate an episode with π
  - (b) For each state appearing in the episode
    - $R \leftarrow$ Return first occurrence of s
    - Append R to Return(s,a)
    - $Q(s,a) \leftarrow$ ave(Returns(s,a))
  - (c) For each s in episode
    - $a^* \leftarrow \text{argmax}_a Q(s,a)$
    - For all $a \in A(s)$:
      $\pi(s,a) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/\lvert A(s)\rvert & \text{if } a=a^* \\ \varepsilon/\lvert A(s)\rvert & \text{if } a\neq a^* \end{cases}$

(a), (b) = policy eval; (c) = policy improvement

2. Q-Learning.

→ mc needs the whole trajectory.

→ Q learning uses only one step trajectory (s,a,r,s') to estimate Q-value.

→ update Q-value of s,a pair immediately after every observation.

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

α: learning rate; γ: discount factor

Algo params: step size $\alpha \in (0,1]$, small $\varepsilon > 0$

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

Loop for each episode:

  Initialize S

  Loop for each step of episode: (maxval; ε-Greedy)

    Choose A from S using policy derived from Q

    Take action A, observe R, S'

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

    $S \leftarrow S'$

  until S is terminal

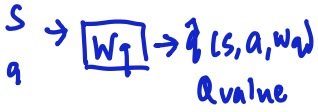
→ Maintain Q table:

$Q = S\begin{bmatrix} & \text{action} & \\ & & \end{bmatrix}$ State-action pairs.

3. DQN

→ Q-Learning needs to maintain Q table, not feasible in large state/action space

→ use N.N as black box to replace table



Module 6: Game Theory.

Nash Equilibrium.

- Each agent is selfish
- Each agent makes decision based on what he think others will do
- No one can do better by changing strategy alone

Eg Prisoners Dilemma

		confess		silent	
		A	B	A	B
(A)	confess	5yrs	5yrs	0yrs	20yrs
	silent	A 20yrs	B 0yrs	A 1yrs	B 1yrs

(B)

dominant move
- confessing always has a higher payoff

cooperation is for both to
But since agent selfish →
will always choose to defect

Pure Strategy N.E

- For two strategies S_1 (for $\bar{i}$) & S_2 (for $\bar{j}$) in N.E
- Assume: 1. Agent $\bar{i}$ plays S_1 and agent $\bar{j}$ can do no better than play S_2
  2. Agent $\bar{j}$ plays S_2 & agent $\bar{i}$ can do no better than play S_1
- Neither agent has any incentive to deviate from N.E

But: 1) Not every interaction has pure strategy N.E
2) Some scenarios may have >1 pure strategy N.E

"specific unchanging choices throughout game"

Mixed Strategy N.E

- No pair of strategies forms a pure strategy N.E:
  - Whatever pair of strategy chosen, somebody wish they had done something else

∴ Soln: Allow mixed strategy
- play 'heads' with prob 0.5
- play 'tails' with prob 0.5

"probabilistic choice to max output"

e.g matching Pennies Payoff Matrix

	$\bar{i}$ heads	$\bar{j}$ tails
$\bar{j}$ heads	1, -1	-1, 1
$\bar{j}$ tails	-1, 1	1, -1

same face ⇒ $\bar{i}$ wins
different face ⇒ $\bar{j}$ wins

Week 8: Logical Agents.

KB Approach

A. Agents that Know
   ↳ By telling new knowledge
   ↳ By updating "

B. Agents that reason
   ↳ use knowledge to deduce course of action

Knowledge Representation
Knowledge Inference
Logic

Knowledge Based Agents.

- Have representations of the world which they operate
- use those representations to infect what action to take

KB

- Set of sentences / Rep of facts
- KR language — tell / Ask
- Also infer from KB.

KB-System

- States: Instances of KB
  → Build by using tell
- Operators
  → Add/infer new sentence
- Goal: Answer Query
  → use Ask

Levels of knowledge in KB

Range from background (observed) & deduced

1) Epistemological – Declarative Description of knowledge
   e.g facts: smoke in kitchen

2) Logical – Logical encoding of knowledge
   e.g facts: smoke
   Rules: Implies (Smoke, Fire)

3) Implementation – Physical representation of knowledge (sentences)
   e.g Implies (Smoke, Fire)

Knowledge Representations.

KR language = Syntax + Semantics (Logic)
(Implementation level) (Knowledge level)

* Logic = Inference + Representation (Syntax + Semantics)

A) (Inference)

* Reasoning: construction of new knowledge from existing ones. Act of deriving logical conclusions from known premise / assumed to be true

Mechanical Reasoning.

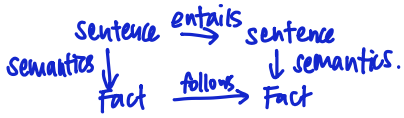
A) Deductive Inference (sound)
   KB: Monday → work, Monday ⊢ work

B) Inductive Inference (unsound)
   KB: Monday → work, work ⊢ Monday

Tell: give KB : (KB | α)!
Ask: given KB ⊢ α : (KB |= α)?

(Form of logical Inference)

Entailment : Returns true given existing sentence return true



* useful properties

Sentence is T/F under interpretation of world

A) valid Sentence (tautology)

iff TRUE under all interpretations e.g S OR ¬S

B) Satisfiable

iff TRUE under some interpretations e.g S and ¬S in some world

Refers to inference procedure

c) Soundness (e.g MP)

- sound if it never proves a false statement; only derives conclusions that are true whenever premise are true

d) completeness

- complete if it can prove all true (valid) statements

B) Representations (Syntax & Semantics)

*Desired properties of Representations

- Expressive
- Concise
- context independent
- Effective
- unambigious

* Desired properties of semantics

- correspondence btw sentence & facts
- Arbitrary meaning, fixed by writer of sentence.
- systematic RIS : Meaning of sentence is a function of the meaning of its parts
- makes claim about world ⇒ Returns T/F

Week 9: propositional logic

* Syntax: of the representation language, specifies all sentences that are well formed.
* Semantics: defines truth of each sentence w.r.t each possible world

(1) Syntax (Backus Naur Form)

Syntax of Propositional Logic
(Backus-Naur Form)

Sentence	→	AtomicSentence | ComplexSentence
AtomicSentence	→	LogicalConstant | PropositionalSymbol
ComplexSentence	→	(Sentence) | Sentence LogicalConnective Sentence | ¬Sentence
LogicalConstant	→	TRUE | FALSE
PropositionalSymbol	→	P | Q | R | …
LogicalConnective	→	$\wedge$ | $\vee$ | $\Leftrightarrow$ | $\Rightarrow$ | $\neg$

Precedence (from highest to lowest): $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
e.g.: $\neg P \wedge Q \vee R \Rightarrow S$ (not ambiguous), equal to: $(((\neg P) \wedge Q) \vee R) \Rightarrow S$

7

(2) validity

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)

10

* Types

Literal: single proposition / negative e.g p or $\neg p$

Clause: propositions formed from finite collections of literals & logical connectivities. e.g $p \vee q \vee r$

Classic Rules of Inference

1. MP/Implict Elimination: $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$
2. And Elimination $\frac{\alpha_1 \wedge \alpha_2 \dots \wedge \alpha_n}{\alpha_i}$
3. And Introduction: $\frac{\alpha_1, \alpha_2 \dots \alpha_n}{\alpha_1 \wedge \alpha_2 \dots \wedge \alpha_n}$
4. Or Introduction $\frac{\alpha_1}{\alpha_1 \vee \alpha_2 \dots \alpha_n}$
5. Double Negation $\frac{\neg\neg\alpha}{\alpha}$

Resolution

1. Unit Resolution $\frac{\alpha \vee \beta, \neg\beta}{\alpha}$
2. Full Resolution: $\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$

Equivalence Rules

- **Equivalent notations**
  - e.g., MP:

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

1) $\alpha \Rightarrow \beta,\ \alpha \ \vdash \beta$

2) $\alpha \Rightarrow \beta,\ \alpha \ \models \beta$

3) $\dfrac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$

4) $((\alpha \Rightarrow \beta) \wedge \alpha) \Rightarrow \beta$

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

Limitations of propositional logic

Given N objects are used in KB, the will be 2^N interpretations/worlds.

Week 10: First order Logic

Week 11: Default Logic

- Standard logic only express something is T/F based on explicitly given rules in KB.
- Default logic is used to formalize reasoning with default assumptions. (i.e common sense ⇒ if nothing in KB conflicts with commonsense, the infer something else)

* Default Theory = Default Rules + KB.

* Default Rules

prereq → $\frac{\alpha(x) : \beta_1(x), \ldots \beta_n(x)}{\gamma(x)}$ ← Justification

conclusion

Applied by substituting x with c (ground instance) to α & β to infer γ

$\alpha(c)$ ⇒ belongs to our set of beliefs

Each $\beta(c)$ ⇒ consistent with our set of beliefs ⇒ infer $\gamma(c)$ as True

Types of Default Rules

- 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., $\frac{bird(x) : has\_wings(x)}{flies(x)}$,

where $has\_wings(x) \vdash flies(x)$

Types of Default Rules

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

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

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

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

* Default Theory $\langle \Delta, \Phi \rangle$ → Phi: set of facts/given KB

Set of Default Rules

e.g $\langle \{\frac{bird(x) : flies(x)}{flies(x)}\}, \{bird(Tweenty), cat(Sylvester)\} \rangle$

* Types of Default Logic Inference

(1) Reiter Default Logic Inference (RDL)

* Algo

* Example

Reiter Default Logic (RDL) Inference

- Guess the extension Ξ (pronounced as "Xi")
- Initialise beliefs $\Xi^* = \Phi$ ⇒ initialise initial belief to fact part in KB.
- (loop over) c-ground instance of an (unused) default $\frac{\alpha(x) : \beta(x)}{\gamma(x)}$:
  - Check two conditions → is c a member of class α
    - Triggered?: $\Xi^* \vdash \alpha(c)$
    - Justified?: $\beta(c)$ is consistent with Ξ ⇒ look through KB to see if any fact contradict β(x)
  - If yes: update beliefs $\Xi^* \leftarrow \Xi^* \cup \{\gamma(c)\}$ → Add new sentence to KB that $\gamma(c)$ is true
- (end loop)
- If $\Xi = \Xi^*$ then extension found/confirmed

The extension is added to our KB as new knowledge

Example

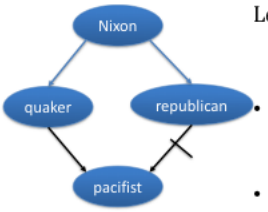
Given Theory: $T = \langle \Delta = \{\frac{bird(x) : flies(x)}{flies(x)}\}, \Phi = \{bird(Tweety), cat(Sylvester)\} \rangle$ ← KB

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

*Limitations

RDL Limitation - Nixon Diamond

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



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

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

RDL Limitation - Nixon Diamond

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

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

There are **two** extensions:

1. One that contains: **pacifist(Nixon)**
2. .. and the one that contains: **¬ pacifist(Nixon)**

if analyse republican first → the conclusion will be diff ∴ conflicts

Conflicting conclusions

Addressing the RDL Limitation

- 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: (Relaxed)
    - a formula is entailed by a default theory if it is entailed by at least one of its extensions.

decision maker's properties

OR

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 Pacifist(Nixon) nor ¬Pacifist(Nixon) are skeptically entailed.
  - Both Pacifist(Nixon) and ¬Pacifist(Nixon) are credulously entailed.
- The credulous extensions of a default theory can be inconsistent with each other.

(2) Makinson Approach

Makinson Approach

- Order ***ground*** instances of defaults in Δ: $d_1, d_2, \ldots$
- Initialize beliefs $\Xi_0 = \Phi$ and used defaults set $\Delta_0 = \emptyset$
- Define Ξ_{n+1} from Ξ_n,
  - Find $d = \frac{\alpha(c) : \beta_1(c), \ldots, \beta_n(c)}{\gamma(c)} \notin \Delta_n$ such that
    - Triggered?: $\Xi_n \vdash \alpha(c)$
    - Justified?: Ξ_n is consistent with $\beta_1(c), \ldots, \beta_m(c)$
  - 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 $\Xi = \bigcup_{i \geq 0} \Xi_i$

4

Makinson Approach

- No extension guessing
  - **Choose** the order of defaults in Δ: $d_1, d_2, \ldots$
- 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

5

Operational Semantics

Given a default theory $T = \langle \Delta, \Phi \rangle$, let $\Pi = (\delta_0, \delta_1, \ldots)$ 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

initial set of facts

union w/ conclusions for

Each sequence Π is associated with two sets:

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

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

For any default rule that we have considered ⇒ add negation of justification portion of those rules

7

Example

if α T ¬β is T

β : T

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

nothing conflicts when γ : T

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

¬β : T

Added to in set

Adds negation of justification.

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

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

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

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

↳ R_2 triggered first but is not supported by KB ∴ Apply R_1 ∴ same conc!

8

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.

If every default rule in the current default theory that's applicable has already occured and been extended

9

Example

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

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

$\Pi_1 = (\delta_1)$ is successful,
$In(\Pi_1) = Cn(\alpha, \eta)$ and $Out(\Pi_1) = \{\beta\}$
but not closed, since δ_2 is applicable, too.

- No overlap btw In set & out set
- No rules that says it ⊢ or ⊢ ¬ T ⇒ then β ⇒ T

All rules explored

$\Pi_2 = (\delta_1, \delta_2)$ is closed, but not successful
$In(\Pi_2) = Cn(\alpha, \eta, \beta)$ and $Out(\Pi_2) = \{\beta, \neg\gamma\}$,
$In(\Pi_2) \cap Out(\Pi_2) = \beta$

conflict

Nothing concludes ¬ T ⇒ ∴ β ⇒ T ⇒ add to In & negation of ¬β out

$\Pi_3 = (\delta_2)$ is a closed and successful process T
$In(\Pi_3) = Cn(\alpha, \beta)$ and $Out(\Pi_3) = \{\neg\gamma\}$,
$In(\Pi_3) \cap Out(\Pi_3) = \emptyset$

δ_1 : α applies but
- justification β : T conflicts w/ δ_2 ¬β ∴ is non conclusive
- δ_1 triggered but not successfully extended: N.A

10

Extension

Process Tree

Extension

- Let $\mathrm{T} = \langle \Delta, \Phi \rangle$ be a default theory. A set of formulae Ξ is an **extension** of T iff there is some ***closed and successful*** Π such that $\Xi = In(\Pi)$.

- To **find a successful** process: generate a process Π, test whether $\mathrm{in}(\Pi) \cap Out(\Pi) = \emptyset$. If not, then backtrack (try another process).

(3) Process Tree
extended: ***

Process Tree

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

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

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

A process is thus a path in G starting from the root.
A node $\mathrm{v} \in V$ is **expanded** if $In(v) \cap Out(v) = \emptyset$. Otherwise, it is a "failed" leaf of the tree.

Add negation of justification into outset

MaKinson Approach Δ ϕ

Consider $\langle \{\frac{\alpha : \neg\beta}{\mu}, \frac{true : \gamma}{\beta}\}, \{\alpha\} \rangle$

Added into the in set

Let $S_1 = \frac{\alpha : \neg\beta}{\mu}$ $S_2 = \frac{true : \gamma}{\beta}$

Find possible sequence. $\pi_1 = (S_1)$, $\pi_2 = (S_1, S_2)$, $\pi_3 = S_2$ $\pi_4 = (S_2, S_1)$

$\pi_1 = S_1$

$In(\pi_1) = Cn(\alpha, \mu)$

$out(\pi_1) = \{\beta\}$

- successful : $In(\pi_1) \cap out(\pi_1) = \emptyset$
- Not closed : δ_2 applies

$\pi_2 = (S_1, S_2)$

$In(\pi_2) = Cn(\alpha, \mu, \beta)$

$out(\pi_2) = \{\beta, \neg\gamma\}$

$In(\pi_2) \cap out(\pi_2) = \beta$

Not successful ⇒ Fails

Closed: All applicable rules extended

$\pi_3 = (S_2)$

$In(\pi_3) = Cn(\alpha, \beta)$

$out(\pi_3) = \{\neg\gamma\}$

$In(\pi_3) \cap out(\pi_3) = \emptyset$

closed: δ_1 triggered but not extended due to conflicts $\neg\beta$

Successful

*if closed & successful

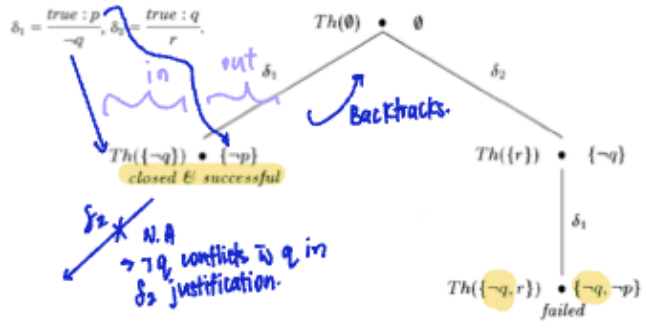
$\square = In(\pi)$

Process tree

Process Tree Example

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

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



13

Process Tree: Properties

- A process is thus a path in G starting from root.
- A node v ∈ V is **expanded** if $In(v) \cap Out(v) = \emptyset$.
- Otherwise, it is a "failed" leaf of the tree.
- Expanded v ∈ V has a child node, w_δ, for every $\delta = \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$
  - 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 $Out(w_\delta) = Out(v) \cup \{\neg\beta_1, \ldots, \neg\beta_n\}$

14