Lecture 12 Lecture 11: Logic

Entailment

Logical Inference: Apply entailment to derive conclusions

- Mathematically, $\alpha \models \beta$ if and only if in every model in which α is true, β is also true.
- Another way: if α is true, then β must also be true.

Inference

- Inference using
  1. Model checking (truth table)
  2. Validity (logical equivelence)
  3. Resolution

- How do we decide if KB $\models \alpha$?

1.

Inference by enumeration

Model checking: enumerates all possible models to check that α is true in all models in which KB is true

```
function TT-ENTAILS?(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic
    symbols ← a list of the proposition symbols in KB and α
    return TT-CHECK-ALL(KB, α, symbols, [ ])

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
  if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true
  else do
      P ← FIRST(symbols); rest ← REST(symbols)
      return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
             TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

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*

Validity and satisfiability 2.

Validity:

- A sentence is valid if it is true in all models
- E.g., $P \vee \neg P$ is valid

Deduction theorem

For any sentences α and β, $\alpha \models \beta$ iff the sentence $(\alpha \Rightarrow \beta)$ is valid

Satisfiability:

- A sentence is satisfiable if it is true in some model
- Determining the satisfiability of sentences in propositional logic was the first problem proved to be NP-complete
- Satisfiability is connected to validity: α is valid iff $\neg\alpha$ is unsatisfiable
- Satisfiability is connected to entailment:

$\alpha \models \beta$ iff the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable (proof by contradiction)

Standard Logic Equivalences

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

A resolution algorithm

3

To prove KB $\models \alpha$, we show that (KB $\wedge \neg\alpha$) is unsatisfiable

(Remember that $\alpha \models \beta$ iff the sentence $(\alpha \wedge \neg\beta)$ is unsatisfiable)

The algorithm:

1. Convert (KB $\wedge \neg\alpha$) to CNF
2. Apply resolution rule to resulting clauses. Each pair with complementary literals is resolved to produce a new clause which is added to the KB
3. Keep going until
   - There are no new cla KB $\not\models \alpha$)
   - Two clauses resolve to yield the empty clause (meaning KB $\models \alpha$)

The empty clause is equivalent to false because a disjunction is true only if one of its disjuncts is true

$$\frac{l_1 \vee l_2, \quad \neg l_2 \vee l_3}{l_1 \vee l_3}$$

$$\frac{l_1 \vee l_2, \quad \neg l_2 \vee l_1}{l_1}$$

Conjunctive Normal Form

- Resolution only applies to sentences of the form $l_1 \vee l_2 \vee \ldots \vee l_k$
- This is called a disjunction of literals
- It turns out that every sentence of propositional logic is logically equivalent to a conjunction of disjunction of literals
- Called Conjunctive Normal Form or CNF

e.g., $\underbrace{(l_1 \vee l_2 \vee l_3 \vee l_4)}_{\text{disjunction}} \wedge (l_5 \vee l_6 \vee l_7 \vee l_8) \wedge \ldots$

conjunction

eg. $(A \vee B) \wedge C$

eg $A \vee B$

eg A

Not CNF

$\neg(A \wedge B)$

$\neg(A \wedge B) \wedge C$

$A \wedge (B \wedge (D \wedge E))$

Combine different facts in the KB (including 5) to check if entailment holds

1. There are no new clauses that can be added (meaning KB $\not\models \alpha$)
2. Two clauses resolve to yield the empty clause (meaning KB $\models \alpha$)

The empty clause is equivalent to false because a disjunction is true only if one of its disjuncts is true

First order logic

Inference Rules

$$\frac{l_1 \vee \cdots \vee l_k, \quad m_1 \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

Modus Ponens	modus Tollens	And-Elimination
$\dfrac{P \quad P \to q}{\therefore q}$	$\dfrac{\neg q \quad P \to q}{\therefore \neg P}$	$\dfrac{P \wedge q}{\therefore P}$

Lecture 14: Probability

Conditional Probability

$$P(A,B) = P(A \mid B)P(B)$$

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

The Axioms of Probability

- Probability values are in the range [0,1] and sum t
  $0 \le P(a) \le 1; \quad \sum_{a \in \Omega} P(a) = 1$
- $P(a \text{ OR } b) = P(a) + P(b) - P(a \text{ AND } b)$

Marginalization

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}, \mathbf{z})$$

or

$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y} \mid \mathbf{z}) P(\mathbf{z})$$

z is over all possible combinations of values of ***Z*** (remember ***Z*** is a set)

Normalization

- In fact, 1/P(*toothache*) can be viewed as a normalization constant for **P**(*Cavity* | *toothache*), ensuring it adds up to 1
- We will refer to normalization constants with the symbol α

$$\mathbf{P}(Cavity \mid toothache) = \alpha \mathbf{P}(Cavity, toothache)$$

$$P(A \mid B) + P(A \mid \neg B) \text{ does not always} = 1$$

$$P(A \mid B) + P(\neg A \mid B) = 1$$

Lecture 15: Bayes Nets

The Full Joint Distribution

$$\prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots, x_1) = \prod_{i=1}^{n} P(x_i \mid parents(x_i))$$

D-separation

Lecture 2: Agents

PEAS Descriptions of Task Environments

Performance, Environment, Actuators, Sensors

Properties of Environments

Fully observable: can access complete state of environment at each point in time	Partially observable: could be due to noisy, inaccurate or incomplete sensor data
Deterministic: next state of the environment completely determined by current state and agent's action	Stochastic: when actions have multiple outcomes, each prescribed by a probability
Episodic: agent's experience divided into independent, atomic episodes in which agent perceives and performs a single action in each episode.	Sequential: current decision affects all future decisions
Static: agent doesn't need to keep sensing while decides what action to take, doesn't need to worry about time	Dynamic: environment changes while agent is thinking (changes with time)
Discrete: (note: applies to states, time, percepts, or actions)	Continuous: continuous values of states and/or actions
Single agent: single decision-making and executing entity	Multiagent: multiple decision-making/executing entities; cooperative or competitive

Rational Agent

Rational agent: for each possible percept sequence, a ratio agent should select an action that is expected to maximize i performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agen ha

Rationality depends on 4 things:

1. Performance measure of success
2. Agent's prior knowledge of environmen
3. Actions agent can perform
4. Agent's percept sequence to date

Types of Agents

Simple Reflex Agent
- Selects actions using only the current percept
- Works on condition-action rules: **if** *condition* **then** *action*

Model-based Reflex
- Maintain some internal state that keeps track of the part of the world it can't see now
- Needs model

Utility-directed Agents
- Utility measures which states are preferable to other states
- Assign numeric values to each possible outcome (utility or "happiness")
- Multidimensional utility (quality, failure rate, etc.)
- Time-dependent utility (hard/soft deadlines)
- Subjective vs. objective utility functions

Goal-directed Agents
- Goal information guides agent's actions (looks to the future)
- Sometimes achieving goal is simple e.g. from a single action
- Other times, goal requires reasoning about long sequences of actions
- Flexible: simply reprogram the agent by changing goals

Learning Agents

Successful agents split task of computing policy in 3 periods:
1. Initially, designers compute some prior knowledge to include in policy
2. When deciding its next action, agent does some computation
3. Agent learns from experience to modify its behavior

Learn from experience to compensate for partial or incorrect prior knowledge

Lecture 3: Uninformed Search

Control Types

Open loop control
- Decision depends on start and goal state
- No sensing at each state

Closed loop control
- Decision depends on each state
- Sensing at each state

Evaluating BFS

Complete?	Yes (if branching factor is finite)
Optimal?	Yes, if step costs are identical. Not guaranteed to be optimal in general
Time Complexity	$O(b^{d+1})$ b – maximum branching factor d – depth of least-cost solution
Space Complexity	Terrible memory usage for large graphs

Evaluary DFS

Yes

NO(could expan some

option with

Uninformed Search

1 • Breadth-first search (open list is FIFO queue) √
2 • Depth-first search (open list is a LIFO queue) √
3 • Uniform-cost search (shallowest node first)
4 • Depth-limited search (DFS with cutoff)
5 • Iterative-deepening search (incrementing cutoff)
6 • Bidirectional search (forward and backward)

4. Depth-limited Search

- Solves infinite path problem by using predetermined depth limit l
- Nodes at depth l are treated as if they have no successors
- Can use knowledge of the problem to determine l (but in general you don't know this in advance)

Complete?	No (If shallowest goal node beyond depth limit)
Optimal?	No (If depth limit > depth of shallowest goal node and we expand a much longer path than the optimal one first)
Time Complexity	$O(b^l)$
Space Complexity	$O(bl)$

Lecture 4: Uninformed and Informed Search

Informed Search

Greedy Best-First Search

- Expands the node closest to the goal, from the list of nodes in the frontier
- Greedy on the heuristic value
- $f(n) = h(n)$

Evaluating Greedy Best-First Search

Complete?	Yes if the graph is finite and the heuristic function is informative (i.e. not 0 at all nodes).
Optimal?	No
Time Complexity	$O(b^m)$
Space Complexity	$O(b^m)$

Greedy Best-First search results in lots of unnecessary nodes being expanded

Heuristic Search: A*

$$f(n) = g(n) + h(n)$$, where

$g(n)$ = cost of path from the initial state to n

$h(n)$ = estimate of the remaining distance

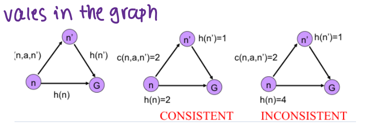
- Expand the node in the open list with the least f value
- When multiple nodes have the same f value, ties can be broken using any rule (first generated node, last generated node, at random, based on heuristic value, etc.)

Admissibility and Consistency

- Admissible heuristic: never overestimates the actual cost to reach a goal. → real cost ≥ all h vales in the graph
- Consistent (or monotone) heuristic:

check all nodes ← $h(n) \le c(n, a, n') + h(n')$ cost of edge

- Every consistent heuristic is also admissible



A*

Heuristic	Guarantees
Admissible	Complete and Optimal
Inadmissible	Complete (optimality depends on problem)
Admissible and consistent	Optimally efficient

Comparing Heuristics

Given two heuristics, how to evaluate which one is better?

- If a heuristic dominates another heuristic, it is strictly better

1. If h_1 and h_2 are admissible, is $\min\{h_1,h_2\}$ admissible? Yes
   - Is it better than h_1 and h_2? No
2. If h_1 and h_2 are admissible, is $\max\{h_1,h_2\}$ admissible? Yes
   - Is it better than h_1 and h_2? Yes

h_1 and h_2 are admissible and h_1 strictly dominates h_2, (i.e. $h_{1(n)} \ge h_{2(n)}$), is h_1 better than h_2? Yes

A* Variants: Iterative Deepening A* (IDA*)

Use iterative deepening (cost-limited search) to reduce memory requirements for A*

- In each iteration do a "cost-limited" depth first search.
  - Cutoff is based on the f-cost (g+h) rather than the depth
- After each iteration, the new cutoff is the smallest f-cost that exceeded the cutoff in the previous iteration
- Not easy to identify when it is more advantageous over A*

A* Variants: Weighted A*

Weighted A*: $f(n) = g(n) + w \times h(n)$
- Local minima
- Trades off optimality for speed
- Orders of magnitude faster than A* in some problems

Lecture 7: Adversarial Search

The Minimax algorithm

1. Generate the whole tree √
2. Label the terminal states with the payoff function
3. Work backwards from the leaves, labeling each state with the best outcome possible for that player
4. Construct a strategy by selecting the the best moves for "Max"
5. Labeling process leads to the "minimax decision" that guarantees maximum payoff, assuming that the opponent is rational

Alpha-Beta Pruning: Intuition

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility valu*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β)) √
    if $v \ge \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS($state$) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if $v \le \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

Effectiveness of Alpha-Beta

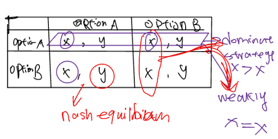
- Depends on order of successors
- Best case: Alpha-Beta reduces complexity from $O(b^m)$ for minimax to $O(b^{m/2})$
- This means Alpha-Beta can lookahead about twice as far as minimax in the same amount of time

Lecture 8: Game Theory & Lecture 9

Dominant Strategies

Suppose a player has two strategies S and S'. We say S dominates S' if choosing S always yields at least as good an outcome as choosing S'.

- S strictly dominates S' if choosing S always gives a better outcome than choosing S' (no matter what the other player does) >
- S weakly dominates S' if there is one set of opponent's actions for which S is superior, and all other sets of opponent's actions give S and S' the same payoff.



Strategies and Equilibria

- Dominant strategy: A player's best move, regardless of what other players do.
- Pareto optimality: A state where no one can be made better off without making someone else worse off (i.e. the best for all players)
- Nash equilibrium: A situation where no player can improve their outcome by changing their strategy, while other players keep theirs the same. same numbers
- Dominant strategy equilibrium: A Nash equilibrium where all players have a dominant strategy.

Mixed Strategies

- Recall that a pure strategy is a deterministic policy i.e. you pick a strategy and play it all the time
- A mixed strategy is a randomized policy i.e. you select your strategy based on a probability distribution
- E.g. Select strategy S1 with probability p and strategy S2 with probability (1-p)
- Is there a mixed strategy Nash Equilibrium in 2 Fingered Morra?