CS 188: Artificial Intelligence

Propositional Logic: Semantics, Inference, Agents

Instructor: Sergey Levine and Stuart Russell

University of California, Berkeley
You can think about deep learning as equivalent to ... our visual cortex or auditory cortex. But, of course, true intelligence is a lot more than just that, you have to recombine it into higher-level thinking and symbolic reasoning, a lot of the things classical AI tried to deal with in the 80s. ... We would like to build up to this symbolic level of reasoning — maths, language, and logic. So that’s a big part of our work.

Demis Hassabis, CEO of Google Deepmind
Knowledge

- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - *Tell* it what it needs to know (or have it *Learn* the knowledge)
  - Then it can *Ask* itself what to do—answers should follow from the KB
- Agents can be viewed at the *knowledge level* i.e., what they *know*, regardless of how implemented
- A single inference algorithm can answer any answerable question
  - Cf. a search algorithm answers only “how to get from A to B” questions

<table>
<thead>
<tr>
<th>Knowledge base</th>
<th>Domain-specific facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference engine</td>
<td>Generic code</td>
</tr>
</tbody>
</table>
Logic

- **Syntax**: What sentences are allowed?
- **Semantics**: What are the possible worlds? Which sentences are true in which worlds? (i.e., definition of truth)

\[ \alpha_1, \alpha_2, \alpha_3 \]

Syntaxland Semanticsland
Examples

- **Propositional logic**
  - Syntax: $P \lor \neg Q \land R$; $X_1 \iff (\text{Raining} \Rightarrow \text{Sunny})$
  - Possible world: \{P=true, Q=true, R=false, S=true\} or 1101
  - Semantics: $\alpha \land \beta$ is true in a world iff $\alpha$ is true and $\beta$ is true (etc.)

- **First-order logic**
  - Syntax: $\forall x \exists y P(x,y) \land \neg Q(\text{Joe}, f(x)) \Rightarrow f(x)=f(y)$
  - Possible world: Objects $o_1, o_2, o_3$; $P$ holds for $<o_1, o_2>$; $Q$ holds for $<o_1, o_3>$; $f(o_1)=o_1$; $\text{Joe}=o_3$; etc.
  - Semantics: $\phi(\sigma)$ is true in a world if $\sigma=\sigma_j$ and $\phi$ holds for $\sigma_j$; etc.
**Inference: entailment**

- **Entailment**: $\alpha \models \beta$ ("$\alpha$ entails $\beta$" or "$\beta$ follows from $\alpha$") iff in every world where $\alpha$ is true, $\beta$ is also true
  - I.e., the $\alpha$-worlds are a subset of the $\beta$-worlds [$\text{models}(\alpha) \subseteq \text{models}(\beta)$]
- In the example, $\alpha_2 \models \alpha_1$
- (Say $\alpha_2$ is $\neg Q \wedge R \wedge S \wedge W$
  $\alpha_1$ is $\neg Q$)
Inference: proofs

- A proof is a demonstration of entailment between $\alpha$ and $\beta$
- Method 1: model-checking
  - For every possible world, if $\alpha$ is true make sure that is $\beta$ true too
  - OK for propositional logic (finitely many worlds); not easy for first-order logic
- Method 2: theorem-proving
  - Search for a sequence of proof steps (applications of inference rules) leading from $\alpha$ to $\beta$
  - E.g., from $P \land (P \Rightarrow Q)$, infer $Q$ by Modus Ponens
- **Sound** algorithm: everything it claims to prove is in fact entailed
- **Complete** algorithm: every that is entailed can be proved
What’s the connection between complete inference algorithms and complete search algorithms?

- **Answer 1:** they both have the words “complete...algorithm”
- **Answer 2:** they both solve any solvable problem
- **Answer 3:** Formulate inference as a search problem
  - Initial state: KB contains $\alpha$
  - Actions: apply any inference rule that matches KB, add conclusion
  - Goal test: KB contains $\beta$

Hence any complete search algorithm (BFS, IDS, ...) yields a complete inference algorithm...

provided the inference rules themselves are strong enough
Propositional logic syntax: The gruesome details

- Given: a set of proposition symbols \( \{X_1, X_2, \ldots, X_n\} \)
  - (we often add True and False for convenience)
- \( X_i \) is a sentence
- If \( \alpha \) is a sentence then \( \neg \alpha \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \land \beta \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \lor \beta \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \Rightarrow \beta \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \Leftrightarrow \beta \) is a sentence
- And p.s. there are no other sentences!
Propositional logic semantics: The unvarnished truth

function PL-TRUE?(\(\alpha\), model) returns true or false

if \(\alpha\) is a symbol then return Lookup(\(\alpha\), model)

if \(\text{Op}(\alpha) = \lnot\) then return not(PL-TRUE?(Arg1(\(\alpha\)), model))

if \(\text{Op}(\alpha) = \land\) then return and(PL-TRUE?(Arg1(\(\alpha\)), model), PL-TRUE?(Arg2(\(\alpha\)), model))

if \(\text{Op}(\alpha) = \Rightarrow\) then return or(PL-TRUE?(Arg1(\(\alpha\)), model), not(PL-TRUE?(Arg2(\(\alpha\)), model)))

etc. (Sometimes called “recursion over syntax”)
PacMan facts

- If Pacman is at 3,3 at time 16 and goes North and there is no wall at 3,4 then Pacman is at 3,4 at time 17:
  - \[\text{At}_3,3_16 \land \text{N}_16 \land \neg \text{Wall}_3,4 \Rightarrow \text{At}_3,3_17\]

- At time 0 Pacman does one of four actions:
  - \[(\text{W}_0 \lor \text{E}_0 \lor \text{N}_0 \lor \text{S}_0)\]
  - \[\neg(\text{W}_0 \land \text{E}_0) \land \neg(\text{W}_0 \land \text{S}_0) \land \ldots\]
Simple theorem proving: Forward chaining

- Forward chaining applies Modus Ponens to generate new facts:
  - Given $X_1 \land X_2 \land \ldots X_n \Rightarrow Y$ and $X_1, X_2, \ldots, X_n$
  - Infer $Y$
- Forward chaining keeps applying this rule, adding new facts, until nothing more can be added
- Requires KB to contain only **definite clauses**:
  - (Conjunction of symbols) $\Rightarrow$ symbol; or
  - A single symbol (note that $X$ is equivalent to $\text{True} \Rightarrow X$)
Forward chaining algorithm

**function** `PL-FC-ENTAILS?(KB, q)` **returns** true or false

count ← a table, where `count[c]` is the number of symbols in c’s premise
inferred ← a table, where `inferred[s]` is initially false for all s
agenda ← a queue of symbols, initially symbols known to be true in KB

**while** agenda is not empty **do**
  p ← Pop(agenda)
  **if** p = q **then return** true
  **if** inferred[p] = false **then**
    inferred[p] ← true
  **for each** clause c in KB where p is in c.premise **do**
    decrement `count[c]`
    **if** `count[c]` = 0 **then** add c.conclusion to agenda

return false
Forward chaining example: Proving Q

**CLAUSES**
- P ⇒ Q
- L ∧ M ⇒ P
- B ∧ L ⇒ M
- A ∧ P ⇒ L
- A ∧ B ⇒ L
- A
- B

**COUNT**
- 1
- 2
- 2
- 2
- 2
- 0
- 0

**INFERRRED**
- A false true
- B false true
- L false true
- M false true
- P false true
- Q false true

**AGENDA**
- A
- B
- L
- M
- P
- Q
Properties of forward chaining

- Theorem: FC is sound and complete for definite-clause KBs
- Soundness: follows from soundness of Modus Ponens (easy to check)
- Completeness proof:
  1. FC reaches a fixed point where no new atomic sentences are derived
  2. Consider the final *inferred* table as a model \( m \), assigning true/false to symbols
  3. Every clause in the original KB is true in \( m \)
     Proof: Suppose a clause \( a_1 \land \ldots \land a_k \Rightarrow b \) is false in \( m \)
     Then \( a_1 \land \ldots \land a_k \) is true in \( m \) and \( b \) is false in \( m \)
     Therefore the algorithm has not reached a fixed point!
  4. Hence \( m \) is a model of KB
  5. If KB \( \models q \), \( q \) is true in every model of KB, including \( m \)
Simple model checking

**function** TT-ENTAILS?(KB, α) **returns** true or false

return TT-CHECK-ALL(KB, α, symbols(KB) U symbols(α), {})

**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
  P ← first(symbols)
  rest ← rest(symbols)
  return and (TT-CHECK-ALL(KB, α, rest, model U {P = true})
              TT-CHECK-ALL(KB, α, rest, model U {P = false}))
Simple model checking, contd.

- Same recursion as backtracking
- $O(2^n)$ time, linear space
- We can do much better!
A sentence is **satisfiable** if it is true in at least one world (cf. CSPs!)

Suppose we have a hyper-efficient SAT solver; how can we use it to test entailment?

- Suppose $\alpha \models \beta$
- Then $\alpha \Rightarrow \beta$ is true in all worlds
- Hence $\neg(\alpha \Rightarrow \beta)$ is false in all worlds
- Hence $\alpha \land \neg\beta$ is false in all worlds, i.e., unsatisfiable

So, add the negated conclusion to what you know, test for (un)satisfiability; also known as **reductio ad absurdum**

Efficient SAT solvers operate on **conjunctive normal form**
Conjunctive normal form (CNF)

- Every sentence can be expressed as a conjunction of clauses.
- Each clause is a disjunction of literals.
- Each literal is a symbol or a negated symbol.

Conversion to CNF by a sequence of standard transformations:

- \( \text{At}_1,1,0 \Rightarrow (\text{Wall}_0,1 \Leftrightarrow \text{Blocked}_W_0) \)
- \( \text{At}_1,1,0 \Rightarrow ((\text{Wall}_0,1 \Rightarrow \text{Blocked}_W_0) \land (\text{Blocked}_W_0 \Rightarrow \text{Wall}_0,1)) \)
- \( \neg \text{At}_1,1,0 \lor ((\neg \text{Wall}_0,1 \lor \text{Blocked}_W_0) \land (\neg \text{Blocked}_W_0 \lor \text{Wall}_0,1)) \)
- \( (\neg \text{At}_1,1,0 \lor \neg \text{Wall}_0,1 \lor \text{Blocked}_W_0) \land \)
  \( (\neg \text{At}_1,1,0 \lor \neg \text{Blocked}_W_0 \lor \text{Wall}_0,1) \)

- Replace biconditional by two implications:
  \( \alpha \Rightarrow \beta \) by \( \neg \alpha \lor \beta \)

- Distribute \( \lor \) over \( \land \)
Efficient SAT solvers

- **DPLL (Davis-Putnam-Logemann-Loveland)** is the core of modern solvers
- Essentially a backtracking search over models with some extras:
  - **Early termination**: stop if
    - all clauses are satisfied; e.g., \((A \lor B) \land (A \lor \neg C)\) is satisfied by \(\{A=true\}\)
    - any clause is falsified; e.g., \((A \lor B) \land (A \lor \neg C)\) is satisfied by \(\{A=false,B=false\}\)
  - **Pure literals**: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., \(A\) is pure and positive in \((A \lor B) \land (A \lor \neg C) \land (C \lor \neg B)\) so set it to true
  - **Unit clauses**: if a clause is left with a single literal, set symbol to satisfy clause
    - E.g., if \(A=false\), \((A \lor B) \land (A \lor \neg C)\) becomes \((false \lor B) \land (false \lor \neg C)\), i.e. \((B) \land (\neg C)\)
    - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.
DPLL algorithm

function DPLL(clauses, symbols, model) returns true or false

if every clause in clauses is true in model then return true
if some clause in clauses is false in model then return false

P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
if P is non-null then return DPLL(clauses, symbols–P, model∪{P=value})

P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols–P, model∪{P=value})

P ← First(symbols); rest ← Rest(symbols)
return or(DPLL(clauses, rest, model∪{P=true}), DPLL(clauses, rest, model∪{P=false}))
Efficiency

- Naïve implementation of DPLL: solve ~100 variables
- Extras:
  - Variable and value ordering (from CSPs)
  - Divide and conquer
  - Caching unsolvable subcases as extra clauses to avoid redoing them
  - Cool indexing and incremental recomputation tricks so that every step of the DPLL algorithm is efficient (typically O(1))
    - Index of clauses in which each variable appears +ve/-ve
    - Keep track number of satisfied clauses, update when variables assigned
    - Keep track of number of remaining literals in each clause
- Real implementation of DPLL: solve ~10000000 variables
SAT solvers in practice

- Circuit verification: does this VLSI circuit compute the right answer?
- Software verification: does this program compute the right answer?
- Software synthesis: what program computes the right answer?
- Protocol verification: can this security protocol be broken?
- Protocol synthesis: what protocol is secure for this task?
- Planning: *how can I eat all the dots???*
A knowledge-based agent

**function** KB-AGENT(percept) **returns** an action

**persistent**: KB, a knowledge base
- t, an integer, initially 0

TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))

action ← ASK(KB, MAKE-ACTION-QUERY(t))

TELL(KB, MAKE-ACTION-SENTENCE(action, t))

t ← t+1

**return** action
Example: Partially observable Pacman

- Pacman has to act given only local perception
  - Four Boolean percept variables for wall in each direction
- What knowledge does he need to begin with?
  - **Sensor model**: sentences stating how the current percept variables are determined by the current state variables
  - **Transition model**: sentences stating how the next-state variables are determined by the current state variables and Pacman’s action
  - **Initial conditions**: what Pacman knows about the initial state
  - **Domain constraints**: what is generally true, e.g., Pacman can do one thing at a time and be in one place at a time
Pacman variables

- Pacman’s location
  - $\text{At}_{1,1,0}$ (Pacman is at [1,1] at time 0) $\text{At}_{3,3,4}$ etc

- Wall locations (these do not change with time)
  - $\text{Wall}_{0,0}$, $\text{Wall}_{0,1}$ etc

- Percepts
  - $\text{Blocked}_W_0$ (blocked by wall to my West at time 0) etc.

- Actions
  - $\text{W}_0$ (Pacman moves West at time 0), $\text{E}_0$ etc.

- $N \times N$ world for $T$ time steps \( \Rightarrow N^2T + N^2 + 4T + 4T = O(N^2T) \) variables

- $2^{N^2T}$ possible worlds! $N=10$, $T=100 \Rightarrow 10^{3010}$ worlds (each a “history”)
Sensor model

- State facts about how Pacman’s percepts arise...
- Pacman perceives a wall to the West at time $t$ if and only if he is in $x,y$ and there is a wall at $x-1,y$ ....
  - $\text{Blocked}_W_0 \iff ((\text{At}_1\text{,}1\text{,}_0 \land \text{Wall}_0\text{,}1) \lor (\text{At}_1\text{,}2\text{,}_0 \land \text{Wall}_0\text{,}2) \lor (\text{At}_1\text{,}3\text{,}_0 \land \text{Wall}_0\text{,}3) \lor \ldots)$

How many of these sentences? How big are they?
Quiz

- What is wrong with sentences like
  - $\text{At}_{1,1,0} \land \text{Wall}_{0,1} \Rightarrow \text{Blocked}_W_{0}$
    - If you are at $[1,1]$ at time 0 and there is a wall in $[0,1]$, the west percept is blocked

- True but *incomplete*!
  - They say “under these conditions the percept variable is true”
  - They don’t say when it is false
  - In particular, they allow for worlds where the percept is always true!!
    - *Unintended or non-standard models*
Transition model

- How does each state variable or fluent at each time gets its value?
- State variables for POPacman are $\text{At}_{x,y,t}$, e.g., $\text{At}_{3,3,17}$
- A state variable gets its value according to a successor-state axiom
  - $X_t \iff [X_{t-1} \land \neg(\text{some action}_{t-1} \text{ made it false})] \lor [\neg X_{t-1} \land (\text{some action}_{t-1} \text{ made it true})]$
- For Pacman location:
  - $\text{At}_{3,3,17} \iff [\text{At}_{3,3,16} \land \neg((\neg \text{Wall}_{3,4} \land \text{N}_{16}) \lor (\neg \text{Wall}_{4,3} \land \text{E}_{16}) \lor \ldots)] \lor [\neg \text{At}_{3,3,16} \land ((\text{At}_{3,2,16} \land \neg \text{Wall}_{3,3} \land \text{N}_{16}) \lor (\text{At}_{2,3,16} \land \neg \text{Wall}_{3,3} \land \text{N}_{16}) \lor \ldots)]$
Initial state

- Pacman may know its initial location:
  - $\text{At}_{1,1_0} \land \neg \text{At}_{1,2_0} \land \neg \text{At}_{1,3_0} \ldots$
- Or, it may not:
  - $\text{At}_{1,1_0} \lor \text{At}_{1,2_0} \lor \text{At}_{1,3_0} \lor \ldots \lor \text{At}_{3,3_0}$
- We also need a **domain constraint** – exactly one thing at a time
  - $\neg (W_0 \land E_0) \land \neg (W_0 \land S_0) \land \ldots$
  - $\neg (W_1 \land E_1) \land \neg (W_1 \land S_1) \land \ldots$
  - $\ldots \land (W_0 \lor E_0 \lor N_0 \lor S_0) \land \ldots$
State estimation

- **State estimation** means keeping track of what’s true now.
- A logical agent can just ask itself!
  - E.g., ask whether $\text{KB} \land <\text{actions}> \land <\text{percepts}> \models \text{Wall}_2,2$
  - This is “lazy”: it involves reasoning about one’s whole life history at each step!
- A more “eager” form of state estimation:
  - After each action and percept
    - For each state variable $X_t$
      - If $X_t$ is entailed, add to KB
      - If $\neg X_t$ is entailed, add to KB
Planning as satisfiability

- Given a hyper-efficient SAT solver, can we use it to make plans?
- Yes, for fully observable, deterministic case:
  - planning problem is solvable iff there is some satisfying assignment
  - solution obtained from truth values of action variables
- For $T = 1$ to infinity, set up the KB as follows and run SAT solver:
  - Initial state, domain constraints
  - Transition model sentences up to time $T$
  - Goal is true at time $T$
- Read off action variables from solution
Summary

- One possible agent architecture: knowledge + inference
- Logics provide a formal way to encode knowledge
  - A logic is defined by: syntax, set of possible worlds, truth condition
- Logical inference computes entailment relations among sentences
- SAT solvers based on DPLL provide incredibly efficient inference
- Logical agents can construct plans by asking whether there is a future in which the goal is achieved