Artificial Intelligence

Propositional Logic

Outline

- Propositional (Boolean) logic
- Equivalence, validity, satisfiability
- Inference rules and theorem proving
    - forward chaining
    - backward chaining
    - resolution

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas
- The proposition symbols P_1, P_2 etc are sentences
  - If S is a sentence, $\neg S$ is a sentence (negation)
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	false	true	false

With these symbols, 8 possible models, can be enumerated automatically.
Rules for evaluating truth with respect to a model *m*:

$\neg S$	is true iff	S is false	
$S_1 \wedge S_2$	is true iff	S_1 is true and	S_2 is true
$S_1 \vee S_2$	is true iff	S_1 is true or	S_2 is true
$S_1 \Rightarrow S_2$	is true iff	S_1 is false or	S_2 is true
i.e.,	is false iff	S_1 is true and	S_2 is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true	

Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (true \vee false) = true \wedge true = true$$

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Inference by enumeration

- Depth-first enumeration of all models is sound and complete

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

- For n symbols, time complexity is $O(2^n)$, space complexity is $O(n)$

Logical equivalence

- Two sentences are logically equivalent iff true in same models: α ≡ β iff α ⊨ β and β ⊨ α

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

Validity and satisfiability

- A sentence is valid if it is true in all models,
  e.g., True, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

- Validity is connected to inference via the Deduction Theorem:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

- A sentence is satisfiable if it is true in some model
  e.g., $A \lor B$, C

- A sentence is unsatisfiable if it is true in no models
  e.g., $A \land \neg A$

- Satisfiability is connected to inference via the following:
  $KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable

Proof methods

- Proof methods divide into (roughly) two kinds:
  - Application of inference rules
    - Legitimate (sound) generation of new sentences from old
    - Proof = a sequence of inference rule applications
      Can use inference rules as operators in a standard search algorithm
    - Typically require transformation of sentences into a normal form
  - Model checking
    - truth table enumeration (always exponential in n)
    - improved backtracking
    - heuristic search in model space (sound but incomplete)

Resolution

- A powerful rule of inference for propositional logic.
- Works only on Conjunctive Normal Form (CNF)
  - literal: an atomic proposition symbol or a negation of the symbol
  - clausal sentence: either a literal or a disjunction of literals
  - CNF: conjunction of clausal forms
- Resolution inference rule (for CNF):
  $P_1 \vee \ldots \vee P_n, Q_1 \vee \ldots Q_m$
  $\models P_1 \vee \ldots \vee P_{i-1} \vee P_{i+1} \vee \ldots \vee P_n \vee Q_1 \vee \ldots \vee Q_{j-1} \vee Q_{j+1} \vee \ldots \vee Q_m$
  where P_i and Q_j are complementary literals
- Resolution is sound and complete for propositional logic

Conversion to CNF

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
- Move $\neg$ inwards using de Morgan's rules and double-negation.
- Apply distributivity law ($\wedge$ over $\vee$) and flatten the sentence.

Resolution algorithm

- Proof by contradiction, i.e., show (KB ∧ ¬ α) unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each Ci, Cj in clauses do
            resolvents ← PL-RESOLVE(Ci, Cj)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Forward and backward chaining

- Horn Form (restricted):
  KB = conjunction of Horn clauses
  - Horn clause:
    - proposition symbol; or
    - (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- Modus Ponens (for Horn Form): complete for Horn KBs

  $\alpha_1, \ldots, \alpha_n, \ \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta \vDash \beta$

- Can be used with forward chaining or backward chaining.
- These algorithms are very natural and run in linear time.

Forward chaining

- Idea: fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found
- Forward chaining is sound and complete for Horn KB

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

   while agenda is not empty do
       p ← POP(agenda)
       unless inferred[p] do
           inferred[p] ← true
           for each Horn clause c in whose premise p appears do
               decrement count[c]
               if count[c] = 0 then do
                   if HEAD[c] = q then return true
                   PUSH(HEAD[c], agenda)
   return false
```

Proof of completeness

- FC derives every atomic sentence that is entailed by KB
  - FC reaches a fixed point where no new atomic sentences are derived
  - Consider the final state as a model m, assigning true/false to symbols
  - Every clause in the original KB is true in m
    $a_1 \land \ldots \land a_k \Rightarrow b$
  - Hence m is a model of KB
  - If KB $\models$ q, q is true in every model of KB, including m

Backward chaining

- Idea: work backwards from the query q:

  to prove q by BC,
  check if q is known already, or
  prove by BC all premises of some rule concluding q

- Avoid loops: check if new subgoal is already on the goal stack

- Avoid repeated work: check if new subgoal

  - has already been proved true, or

  - has already failed

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - syntax: formal structure of sentences
  - semantics: truth of sentences with respect to models
  - entailment: necessary truth of one sentence given another
  - inference: deriving sentences from other sentences
  - soundness: derivations produce only entailed sentences
  - completeness: derivations can produce all entailed sentences
- Resolution is complete for propositional logic
- Forward, backward chaining are linear-time, complete for Horn clauses
- Propositional logic lacks expressive power