Logic

- A proposition or sentence is an expression that can be evaluated into a truth-value (true or false). It is made up of symbols representing variables, which can be either true or false.

- Propositions can be built from symbols using negation ($\neg$), conjunction ($\land$), disjunction ($\lor$), implication ($\Rightarrow$), and biconditional implication ($\Leftrightarrow$). A literal is a symbol or its negation.

- A Horn clause is a proposition that is either a single literal or of the form $(B \land C \land \ldots) \Rightarrow A$. Equivalently, it is a clause (a proposition of the form $(A \lor B \lor \ldots)$) in which at most one of the literals is true. A proposition in Horn form is a conjunction of Horn clauses.

- A proposition in conjunctive normal form (CNF) is a conjunction of disjunctions: $(A \lor B \lor \ldots) \land (C \lor D \lor \ldots) \land \ldots$. Any proposition can be made into this form.

- A model is a possible world, an assignment of truth-values to all of the variables. For any proposition $\alpha$, $M(\alpha)$ is the set of models in which $\alpha$ is true. A proposition is valid if it is true in all models.

- A set of propositions can be stored in a knowledge base. A knowledge base $KB$ entails a proposition $\alpha$ (written $KB \models \alpha$) if $\alpha$ is true in all worlds where $KB$ is true – that is, if $M(KB) \subseteq M(\alpha)$. (Note the direction! Propositional logic is monotonic, which means that adding propositions can only add knowledge, it can never remove existing knowledge. The more propositions that are known, the fewer models are consistent with the knowledge, so the smaller the set is.) Two propositions (or sets of propositions) $\alpha$ and $\beta$ are logically equivalent if $\alpha \models \beta$ and $\beta \models \alpha$.

- Deduction Theorem: $KB \models \alpha$ whenever the proposition $KB \Rightarrow \alpha$ is valid.

- Propositions can be proven using model checking, which enumerates the possible models. Or, they can be proven using logical inference, deriving propositions from other propositions. A procedure $i$ derives a proposition $\alpha$ from a knowledge base $KB$ (written $KB \vdash_i \alpha$) if $\alpha$ can be derived from $KB$ by $i$. A procedure $i$ is sound if it generates only true propositions, and is complete if it generates all true propositions.

- Some useful procedures (inference rules):
  
  Modus ponens: If $A$, and $A \Rightarrow B$, then $B$.
  
  Modus tollens: If $A \Rightarrow B$, and $\neg B$, then $\neg A$.
  
  Resolution: If $A \lor B$, and $\neg A$, then $B$. 
• Whether a knowledge base entails a proposition $\alpha$ can be checked using **forward chaining** (which uses modus ponens to derive new propositions, until it finds $\alpha$ or doesn’t) or **backward chaining** (which works backward from $\alpha$ to find propositions that prove it).

• A proposition is **satisfiable** if there exists some assignment of values to its variables that makes it true. To prove a proposition unsatisfiable, one can add its negation to the knowledge base and apply the resolution procedure until False is shown to be entailed.

• The **DPLL** algorithm checks satisfiability of a CNF proposition by using depth-first search while taking advantage of some facts about logic to take shortcuts:
  
  · A clause is true if any of its literals is true. A proposition is false if any of its clauses is false.
  
  · Pure symbol heuristic: If a symbol always appears with the same sign (always $A$ or always $\neg A$), that literal can be assigned to true.
  
  · Unit clause heuristic: If a clause has only one literal (or all the other literals are assigned false), then that literal must be assigned to true.

• The **WalkSAT** algorithm checks satisfiability of a CNF proposition by using the “min-conflict heuristic,” starting with a random assignment and flipping variables’ values to those that maximize the number of satisfied clauses. (So it cannot definitively conclude that a proposition is unsatisfiable.)

• **First-order logic** is a superset of propositional logic. It allows quantifiers (exists ($\exists$) and for all ($\forall$)), objects (things other than True/False), variables (whose values can be objects), functions (which can return things other than truth-values), and relations (which have truth-values).
Exercises

1. (AIMA 7.4) Which of the following are correct?

   (a) \( \text{False} \models \text{True} \).
   (b) \( \text{True} \models \text{False} \).
   (c) \( (A \land B) \models (A \iff B) \).
   (d) \( A \iff B \models A \lor B \).
   (e) \( A \iff B \models \neg A \lor B \).
   (f) \( (A \land B) \Rightarrow C \models (A \Rightarrow C) \lor (B \Rightarrow C) \).
   (g) \( (C \lor (\neg A \land \neg B)) \equiv ((A \Rightarrow C) \land (B \Rightarrow C)) \).
   (h) \( (A \lor B) \land \neg (A \Rightarrow B) \) is satisfiable.
   (i) \( (A \iff B) \land (\neg A \lor B) \) is satisfiable.

2. (AIMA 7.20) Convert the following set of sentences to clausal form.

   (a) \( A \iff (B \lor E) \).
   (b) \( E \Rightarrow D \).
   (c) \( C \land F \Rightarrow \neg B \).
   (d) \( E \Rightarrow B \).
   (e) \( B \Rightarrow F \).
   (f) \( B \Rightarrow C \).

   Give a trace of the execution of DPLL on the conjunction of these clauses.

3. (AIMA 7.12) Use resolution to prove \( (\neg A \land \neg B) \) from the clauses in exercise 2.

4. Consider the interpretation \( i = \{ A = \text{True}, B = \text{False}, C = \text{True}, D = \text{False} \} \) For each of these sentences, indicate whether it’s valid, unsatisfiable, not valid, but it holds in \( i \), or not unsatisfiable, but fails in \( i \).

   (a) \( A \Rightarrow \neg A \)
   (b) \( \neg(A \land B) \Rightarrow (\neg A \lor \neg B) \)
   (c) \( B \Rightarrow C \land D \)
   (d) \( A \Rightarrow C \land D \)
   (e) \( (A \land C) \iff (B \land D) \)
   (f) \( A \lor B \lor C \lor D \)
   (g) \( D \iff \neg D \)
def dpll(clauses, symbols, model):
    "See if the clauses are true in a partial model."
    unknown_clauses = []  ## clauses with an unknown truth value
    for c in clauses:
        val = pl_true(c, model)
        if val == False:
            return False
        if val != True:
            unknown_clauses.append(c)
    if not unknown_clauses:
        return model
    P, value = find_pure_symbol(symbols, unknown_clauses)
    if P:
        return dpll(clauses, removeall(P, symbols), extend(model, P, value))
    P, value = find_unit_clause(unknown_clauses, model)
    if P:
        return dpll(clauses, removeall(P, symbols), extend(model, P, value))
    P, symbols = symbols[0], symbols[1:]
    return (dpll(clauses, symbols, extend(model, P, True)) or
            dpll(clauses, symbols, extend(model, P, False)))