Overview

• Last time
  – Logic for KR in general; Propositional Logic; Natural Deduction

• Today
  – Entailment, satisfiability and validity
  – Normal forms
    • Negation normal form
    • Conjunctive normal form
  – Satisfiability as a search problem
    • DPLL algorithm

• Learning outcomes covered today:

Distinguish the characteristics, and advantages and disadvantages, of the major knowledge representation paradigms that have been used in AI, such as production rules, semantic networks, propositional logic and first-order logic;

Solve simple knowledge-based problems using the AI representations studied;

Propositional Logic for KR

• Given a knowledge base KB and a property α, check if KB ⊨ α
  – Use truth tables
  – Prove α from KB
  – Relate with validity and satisfiability
  – Davis-Putnam algorithm

Validity and Satisfiability

• A formula is said to be valid (or a tautology) iff it is true under every interpretation.

• A formula is said to be satisfiable (or consistent) iff it is true under at least one interpretation.

• A formula is said to be unsatisfiable (or inconsistent or contradictory) iff it is not made true under any interpretation.

• If a formula φ is a valid then ¬φ is unsatisfiable.
Validity, Satisfiability and Entailment

Implications for Knowledge Representation

- **Deduction Theorem:**
  \( \text{KB} \models \alpha \) if and only if \( (\text{KB} \Rightarrow \alpha) \) is valid
- Or, . . .
  \( \text{KB} \models \alpha \) if and only if \( (\text{KB} \land \neg \alpha) \) is unsatisfiable
  *reductio ad absurdum*

Satisfiability Checking

- Given a knowledge base \( \text{KB} \) and a property \( \alpha \), check if \( (\text{KB} \land \neg \alpha) \) is satisfiable
  - If not satisfiable, \( \alpha \) is implied by \( \text{KB} \)
  - Otherwise, an interpretation of propositions would give us a *countermodel*

Countermodel (I)

To check if
\[
(\text{hot} \land \text{smoky} \Rightarrow \text{fire}) \\
\land (\text{alarm_beeps} \Rightarrow \text{smoky}) \\
\land (\text{fire} \Rightarrow \text{switch_on_sprinklers}) \\
\land (\text{alarm_beeps} \Rightarrow \text{switch_on_sprinklers})
\]

we form
\[
(\text{hot} \land \text{smoky} \Rightarrow \text{fire}) \\
\land (\text{alarm_beeps} \Rightarrow \text{smoky}) \\
\land (\text{fire} \Rightarrow \text{switch_on_sprinklers}) \\
\land (\neg \text{alarm_beeps} \Rightarrow \text{switch_on_sprinklers})
\]

Countermodel (II)

But
\[
(\text{hot} \land \text{smoky} \Rightarrow \text{fire}) \\
\land (\text{alarm_beeps} \Rightarrow \text{smoky}) \\
\land (\text{fire} \Rightarrow \text{switch_on_sprinklers}) \\
\land (\neg \text{alarm_beeps} \Rightarrow \text{switch_on_sprinklers})
\]

is true under the interpretation \( I \):
\[
I(\text{hot}) = F \\
I(\text{smoky}) = T \\
I(\text{fire}) = F \\
I(\text{alarm_beeps}) = T \\
I(\text{switch_on_sprinklers}) = F
\]
Example: Truth Tables and Satisfiability

Using a truth table show whether 
\((p \Rightarrow q) \lor (q \Rightarrow p)\)
is a tautology, consistent or inconsistent.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>(p⇒q)</th>
<th>(q⇒p)</th>
<th>(p⇒q) ∨ (q⇒p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Equivalent Formulae

• Two formulae A and B are equivalent, written \(A \equiv B\) iff A and B have the same truth values for every interpretation.
• Show 
\((p \Rightarrow q) \equiv (\neg p \lor q)\)
• Draw up a truth table for \((p \Rightarrow q)\) and \((\neg p \lor q)\) and check their truth values are the same.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>(p⇒q)</th>
<th>(¬p)</th>
<th>(¬p ∨ q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Efficiency

• A truth table contains \(2^n\) rows
• Its construction requires \(2^n\) steps . .
• Can we do better than that?
  – Not really: theory says that this is a very hard problem
  – In practice, not so bad, if we can use heuristics to identify lines we don’t need to check
  – But we have to transform \((K \land \neg \alpha)\) into a normal form

Equivalent Transformations (I)

• Where A, B and C are propositions or propositional formulae and T and F are true and false respectively
• Idempotent laws 
  \(A \land A \equiv A\)
  \(A \lor A \equiv A\)
• Associative laws 
  \((A \land B) \land C \equiv A \land (B \land C)\)
  \((A \lor B) \lor C \equiv A \lor (B \lor C)\)
• Commutative laws 
  \(A \land B \equiv B \land A\)
  \(A \lor B \equiv B \lor A\)
• Distributive laws 
  \(A \land (B \lor C) \equiv (A \land B) \lor (A \land C)\)
  \(A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)\)
Equivalent Transformations (II)

- Identity laws
  \[ A \land T \equiv A \]
  \[ A \lor F \equiv A \]
  \[ A \land F \equiv F \]
  \[ A \lor T \equiv T \]

- Complement laws
  \[ A \land \neg A \equiv F \]
  \[ A \lor \neg A \equiv T \]
  \[ \neg \neg A \equiv A \]
  \[ \neg T \equiv F \]
  \[ \neg F \equiv T \]

- De Morgan’s laws
  \[ \neg (A \land B) \equiv \neg A \lor \neg B \]
  \[ \neg (A \lor B) \equiv \neg A \land \neg B \]

- Laws for \( \Rightarrow \) and \( \leftrightarrow \)
  \[ A \Rightarrow B \equiv \neg A \lor B \]
  \[ A \leftrightarrow B \equiv (A \Rightarrow B) \land (B \Rightarrow A) \]

- We can use these laws to simplify expressions and to prove equivalences.

Examples

- Simplify the following expression:
  \[ \neg (\neg P \land \neg Q) \]
  \[ \neg (\neg P \land \neg Q) \text{ given} \]
  \[ \equiv (\neg \neg P \lor \neg \neg Q) \text{ de Morgan’s laws} \]
  \[ \equiv (P \lor Q) \text{ Complement laws} \]

- Prove the following equivalence:
  \[ \neg (\neg (P \land Q) \lor P) \equiv F \]
  \[ \neg (\neg (P \land Q) \lor P) \text{ given} \]
  \[ \equiv \neg ((\neg P \lor \neg Q) \lor P) \text{ de Morgan’s laws} \]
  \[ \equiv \neg (\neg Q \lor (\neg P \lor P)) \text{ Commutative laws} \]
  \[ \equiv \neg (\neg Q \lor T) \text{ Associative laws} \]
  \[ \equiv \neg T \text{ Complement laws} \]
  \[ \equiv F \text{ Complement laws} \]

Negation Normal Form

- It is often useful to transform formulae into normal forms. These are logically equivalent formulae but have syntactically different forms that may be more suitable for reasoning with.

- There are several normal forms: Negation Normal Form, Clausal Form, Disjunctive Normal Form and Conjunctive Normal Form. We are most interested in Conjunctive Normal Form (CNF).

- A formula is in negation normal form if negations appear only in front of propositions and the only operators are \( \land \), \( \lor \) and \( \neg \).

- First remove the \( \Rightarrow \) and \( \leftrightarrow \) operators. Then apply de Morgan’s laws and remove double negations (complement laws), until in the correct form.

Conjunctive Normal Form

- A formula is in Conjunctive Normal Form if it is of the form
  \[ A_1 \land A_2 \land \ldots \land A_k \]
  where each \( A_i \) is a disjunction of propositions or their negations.

- Example
  \[ (p \lor q) \land r \land (\neg p \lor \neg r \lor s) \text{ is in CNF.} \]
  \[ \neg (p \lor q) \land r \land (\neg p \lor \neg r \lor s) \text{ is not in CNF.} \]
  \[ (p \lor q) \land r \land (p \Rightarrow (\neg r \lor s)) \text{ is not in CNF.} \]
**Conjunctive Normal Form**

- To translate into CNF, first translate into NNF. Then apply distribution laws or commutativity laws until in the correct form.

- **Any** well-formed formula of propositional logic can be rewritten, using the previous equivalences, as an equivalent formula of CNF.

**Example**

- Translate \((\neg(p \lor \neg q) \lor r) \Rightarrow p\) into CNF.

  - We first translate into NNF and obtain \(((p \lor \neg q) \land \neg r) \lor p\).

  - Then transform into CNF

    \[((p \lor \neg q) \land \neg r) \lor p\]
    \[\equiv ((p \lor \neg q) \land \neg r) \lor (\neg r \lor p)\]
    \[\equiv (p \lor \neg q \lor p) \land (\neg r \lor p)\]

**Exercise**

**Satisfiability as a Search Problem**

- Given a formula in CNF, can we find an assignment of truth values to propositions that satisfies it?

  - **States** are *partial assignments* - some propositions get values, some are possibly unassigned.
  - **Actions** are deciding whether a (yet unassigned) proposition is true or false.
  - **Initial state**: empty partial assignment.
  - **Goal state**: an assignment making the formula true.
Algorithm as Satisfiability for CNF

• A complete backtracking algorithm – Davis-Putnam paper (1960)
• The version presented (DPLL) is described in a paper by Davis, Logemann, and Loveland (1962).
• Naive Approach: every proposition is either true or false in any interpretation.

Search Space

• Consider \((\neg p \lor \neg r) \land (p \lor q) \land (r \lor \neg q)\)

---

Partial Assignment

• Another idea: Simplify formula with a partial assignment

\(\neg p \lor \neg r) \land (p \lor q) \land (r \lor \neg q)\); let \(p = \text{True}\)

\(\neg \text{True} \lor \neg r) \land \text{True} \land (r \lor \neg q)\)

\(\text{False} \lor \neg r) \land \text{True} \land (r \lor \neg q)\)

\(\neg r \land (r \lor \neg q)\) can only be true if \(r = \text{False}\)

\(\neg \text{False} \land (\text{False} \lor \neg q)\)

\(\neg q\) can only be true if \(\neg q = \text{True}\) (so \(q = \text{False}\))

\(\text{True}\)

• Given a good order for partial assignments this can be very helpful. DPLL provides heuristics for choosing partial assignments. Only in the worst case we will need to try everything.

Likewise

• \((\neg p \lor \neg r) \land (p \lor q) \land (r \lor \neg q)\); let \(p = \text{False}\)

\(\neg \text{False} \lor \neg r) \land (\text{False} \lor q) \land (r \lor \neg q)\)

\(\text{True} \lor \neg r) \land q \land (r \lor \neg q)\)

\(\text{True} \land q \land (r \lor \neg q)\)

\(q \land (r \lor \neg q)\) can only be true if \(q = \text{True}\)

\(\text{True} \land (r \lor \neg \text{True})\)

\((r \lor \text{False})\)

\(r\) can only be true if \(r = \text{True}\)

\(\text{True}\)
Reduced Search Space

- Consider \((\neg p \lor \neg r) \land (p \lor q) \land (r \lor \neg q)\)

Only 5 nodes!

Algorithm Structure

- Search through possible assignments of propositions
- Simplify formulae with partial assignments
  - Unit clause propagation
  - Pure literal elimination

Unit Clause

- A clause with just one literal is called a unit clause
  - e.g. \([q] \land (\neg r \lor \neg q) \land (r \lor s)\)
- Literal's value can be uniquely assigned
  - q must be set to True
- Unit clause propagation:
  Check if a formula in CNF has a unit clause, C.
  Set the value of the literal in C such that C is True
  - Notice that some other clauses may become unit
    - e.g. \([q] \land (\neg r \lor \neg q) \land (r \lor s)\) reduces to \(\neg r \land (r \lor s)\)
  - r must be set to False
  - \(\neg r \land (r \lor s)\) reduces to s

Pure Literal

- A pure literal is a literal that always appears with the same “sign” in all clauses.
  - e.g. \([p] \land (\neg q \lor \neg r) \land (\neg q \lor r)\)
- Making a pure literal True makes some clauses True, but no clause False
  - e.g. \([p] \land (\neg q \lor \neg r) \land (\neg q \lor r)\) reduces to \(\neg q \land (\neg q \lor r)\)
  - \(\neg q \land (\neg q \lor r)\) reduces to True
- Pure literal elimination:
  Check if a formula in CNF has a pure literal, l
  Set the value of l to True
DPPL(φ)
• Given a propositional formula φ, DPLL(φ) does the following:
  • If φ is True then return True;
  • If φ is False then return False;
  • Pick a proposition p in φ
    – If DPLL(Simplify(φ, p)) is True then return True;
    – If DPLL(Simplify(φ, ¬p)) is True then return True;
  • Return False;

Simplify(φ, L)
• Given a propositional formula φ and a literal l, simplify(φ, l) does the following:
  – Delete every clause that contains l from φ;
  – Delete ¬l from all clauses in φ
  – Apply exhaustively unit clause propagation and pure literal elimination;
  – Fail (return False) if positive and negative unit clauses for the same literal.

Applications
• There are now several efficient SAT solvers based on DPLL available on the web. They are used in a number of applications.
  – Hardware and software verification
    • IBM, Intel, . . .
  – Planning
  – Scheduling
  – . . .

Summary
• We have considered issues concerning efficiency in propositional reasoning
• This has covered equivalent transformations and normal forms
  – Negation normal form
  – Conjunctive normal form
• We have considered how satisfiability can be viewed as a search problem
  – Looked at an algorithm for satisfiability in CNF

• Next time
  – Proof method resolution for propositional logic