Recall: Wumpus World

1

2

3

4

Wumpus

You (Agent)
Recall: Wumpus KB

Knowledge Base (KB) includes the following sentences:

- Statements currently known to be true:
  - \( \neg P_{1,1} \)
  - \( \neg B_{1,1} \)
  - \( B_{2,1} \)

- Properties of the world: E.g., "Pits cause breezes in adjacent squares"
  - \( B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1}) \)
  - \( B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1}) \)
  (and so on for all squares)

Recall from last time:

- \( m \) is a model of a sentence \( \alpha \) if \( \alpha \) is true in \( m \)
- \( M(\alpha) \) is the set of all models of \( \alpha \)
- \( KB \models \alpha \) (KB "entails" \( \alpha \)) iff \( M(KB) \subseteq M(\alpha) \)

Is there no pit in [1,2]?

\( KB \models \neg P_{1,2} \)?
Inference by Truth Table Enumeration

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$P_{3,2}$</th>
<th>KB</th>
<th>$\neg P_{1,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

In all models in which KB is true, $\neg P_{1,2}$ is also true.
Therefore, KB $\models \neg P_{1,2}$
Another Example

Is there a pit in [2,2]?

Inference by Truth Table Enumeration

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$KB$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>false</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

$P_{2,2}$ is false in a model in which $KB$ is true
Therefore, $KB \not\models P_{2,2}$
Inference by TT Enumeration

• Algorithm: Depth-first enumeration of all models (see Fig. 7.10 in text for pseudocode)

• Algorithm is sound & complete

• For $n$ symbols:
  • time complexity = $O(2^n)$, space = $O(n)$

Concepts for Other Techniques:

Logical Equivalence

Two sentences are logically equivalent iff they are true in the same models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \implies \beta) & \equiv (\neg\beta \implies \neg\alpha) \quad \text{contraposition} \\
(\alpha \iff \beta) & \equiv (\neg\alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \iff \beta) & \equiv ( (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) ) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg\alpha \lor \neg\beta) \quad \text{de Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg\alpha \land \neg\beta) \quad \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
Concepts for Other Techniques: Validity and Satisfiability

- A sentence is **valid** if it is true in all models (a tautology)
  
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:
  
  $KB \models a$ if and only if $(KB \Rightarrow a)$ 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: $KB \models a$ if and only if $(KB \land \neg a)$ is unsatisfiable (proof by contradiction)

Inference/Proof Techniques

- Two kinds (roughly):

  **Model checking**
  - Truth table enumeration (always exponential in $n$)
  - Efficient backtracking algorithms,
    e.g., Davis-Putnam-Logemann-Loveland (DPLL)
  - Local search algorithms (sound but incomplete)
    e.g., randomized hill-climbing (WalkSAT)

  **Successive application of inference rules**
  - Generate new sentences from old in a sound way
  - Proof = a sequence of inference rule applications
  - Use inference rules as successor function in a standard search algorithm
Inference Technique I: Resolution

Terminology:
Literal = proposition symbol or its negation
E.g., A, ¬A, B, ¬B, etc.

Clause = disjunction of literals
E.g., (B ∨ ¬C ∨ ¬D)

Resolution assumes sentences are in
Conjunctive Normal Form (CNF):
sentence = conjunction of clauses
E.g., (A ∨ ¬B) ∧ (B ∨ ¬C ∨ ¬D)

Conversion to CNF

E.g., B_{1,1} ↔ (P_{1,2} ∨ P_{2,1})

1. Eliminate ↔, replacing α ↔ β with (α ⇒ β) ∧ (β ⇒ α).
   (B_{1,1} ⇒ (P_{1,2} ∨ P_{2,1})) ∧ ((P_{1,2} ∨ P_{2,1}) ⇒ B_{1,1})

2. Eliminate ⇒, replacing α ⇒ β with ¬α ∨ β.
   (¬B_{1,1} ∨ P_{1,2} ∨ P_{2,1}) ∧ (¬(P_{1,2} ∨ P_{2,1}) ∨ B_{1,1})

3. Move ¬ inwards using de Morgan's rules and double-negation:
   (¬B_{1,1} ∨ P_{1,2} ∨ P_{2,1}) ∧ ((¬P_{1,2} ∧ ¬P_{2,1}) ∨ B_{1,1})

4. Apply distributivity law (∧ over ∨) and flatten:
   (¬B_{1,1} ∨ P_{1,2} ∨ P_{2,1}) ∧ (¬P_{1,2} ∨ B_{1,1}) ∧ (¬P_{2,1} ∨ B_{1,1})

   This is in CNF - Done!
Resolution motivation

There is a pit in [1,3] or
There is a pit in [2,2]
There is no pit in [2,2]

There is a pit in [1,3]

More generally,

\[ \ell_1 \lor \ldots \lor \ell_k, \quad \neg \ell_i \]

\[ \ell_i \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \]

Inference Technique: Resolution

- General Resolution inference rule (for CNF):

\[ \ell_1 \lor \ldots \lor \ell_k, \quad m_1 \lor \ldots \lor m_n \]

\[ \ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n \]

where \( \ell_i \) and \( m_j \) are complementary literals.

E.g., \( P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2} \)

\[ P_{1,3} \]

- Resolution is sound for propositional logic
Resolution

Soundness of resolution inference rule
(Recall logical equivalence $A \Rightarrow B \equiv \neg A \lor B$)

\[-(l_1 \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow l_i \]
\[-m_j \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n)\]

\[-(l_i \lor \ldots \lor l_{i-1} \lor l_{i+1} \lor \ldots \lor l_k) \Rightarrow (m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n)\]

(since $l_i = \neg m_j$)

Resolution algorithm

- To show $KB \models \alpha$, use proof by contradiction, i.e., show $KB \land \neg \alpha$ unsatisfiable

```
function PL-Resolution(KB, \alpha) returns true or false
    clauses \leftarrow the set of clauses in the CNF representation of $KB \land \neg \alpha$
    new \leftarrow \{ \}
    loop do
        for each $C_i, C_j$ in clauses do
            resolvents \leftarrow PL-Resolve($C_i, C_j$)
            if resolvents contains the empty clause then return true
            new \leftarrow new \cup resolvents
        if new \subseteq clauses then return false
        clauses \leftarrow clauses \cup new
    end loop
    return false
```

**Resolution example**

Given no breeze in [1,1], prove there's no pit in [1,2]

\[ KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \lnot B_{1,1} \text{ and } \alpha = \lnot P_{1,2} \]

Resolution: Convert to CNF and show \( KB \land \lnot \alpha \) is unsatisfiable
Resolution example

Empty clause
(i.e., KB \land \neg \alpha \text{ unsatisfiable})

Inference Technique II:
Forward/Backward Chaining

- Require sentences to be in Horn Form:
  
  \( KB = \text{conjunction of Horn clauses} \)

  Horn clause =
  
  - proposition symbol or
  - \("(\text{conjunction of symbols}) \Rightarrow \text{symbol}"\)
    (i.e. clause with at most 1 positive literal)

  E.g., \( KB = C \land (B \Rightarrow A) \land (C \land D \Rightarrow B) \)

- F/B chaining based on "Modus Ponens" rule:
  
  \[ a_1, \ldots, a_n, a_1 \land \ldots \land a_n \Rightarrow \beta \]

  \[ \beta \]

  Complete for Horn clauses

- Very natural and linear time complexity in size of KB
Forward chaining

- Idea: fire any rule whose premises are satisfied in $KB$, add its conclusion to $KB$, until query $q$ is found

$P \Rightarrow Q$
$L \land M \Rightarrow P$
$B \land L \Rightarrow M$
$A \land P \Rightarrow L$
$A \land B \Rightarrow L$
$A$
$B$

Query = “Is Q true?”  

AND-OR Graph

Forward chaining algorithm

```
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
```

Forward chaining is sound & complete for Horn KB
Forward chaining example

Query = Q
(i.e. "Is Q true?")
Forward chaining example

Forward chaining example
Forward chaining example

Forward chaining example
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 goal stack

Avoid repeated work: check if new subgoal
1. has already been proved true, or
2. has already failed

Backward chaining example
Backward chaining example

Backward chaining example
Backward chaining example

Backward chaining example
Backward chaining example

Backward chaining example
Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing, e.g., object recognition, routine decisions
- FC may do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving, e.g., How do I get an A in this class? e.g., What is my best exit strategy out of the classroom? e.g., How can I impress my date tonight?
- Complexity of BC can be much less than linear in size of KB

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination
   - A clause is true if any literal is true.
   - A sentence is false if any clause is false.

2. Pure symbol heuristic
   - Pure symbol: always appears with the same "sign" in all clauses.
   - e.g., In the three clauses (A ∨ ~B), (~B ∨ ~C), (C ∨ A), A and B are pure, C is impure.
   - Make a pure symbol literal true.

3. Unit clause heuristic
   - Unit clause: only one literal in the clause
   - The only literal in a unit clause must be true.
The DPLL algorithm

function DPLL-SATISFIABLE?(s) returns true or false
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(clauses, symbols, [])

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, |P = value|model)
P, value ← FIND-UNIT-CLAUSE(clauses, model)
if P is non-null then return DPLL(clauses, symbols-P, |P = value|model)
P ← FIRST(symbols); rest ← REST(symbols)
return DPLL(clauses, rest, |P = true|model) or
DPLL(clauses, rest, |P = false|model)

Next Time

• WalkSAT
• Logical Agents: Wumpus
• First-Order Logic
• To Do:
  Project #2
  Finish Chapter 7
  Start Chapter 8