Review of Last Time

- $\models$ means “logically follows”
- $\vdash$ means “can be derived from”
- If your inference algorithm derives only things that follow logically from the KB, the inference is sound
- If everything that follows logically from the KB can be derived using your inference algorithm, the inference is complete

Entailment Applied to the Wumpus World

Let us consider the models that support the conclusion $\alpha_1 = \text{"There is no pit in } [1,2].\text{"} $ We draw a line marked with $\alpha_1$ around these models. In every model in which KB is true, $\alpha_1$ is also true. Therefore KB $\models \alpha_1$

Inference: Model Checking

- Suppose we want to know if KB $\models \neg P_{1,2}$?
- In the 3 models in which KB is true, $\neg P_{1,2}$ is also true

<table>
<thead>
<tr>
<th>R_{1,1}</th>
<th>P_{1,1}</th>
<th>P_{1,2}</th>
<th>P_{2,1}</th>
<th>P_{2,2}</th>
<th>R_{1}</th>
<th>R_{2}</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</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>true</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>true</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>true</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>true</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>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</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>false</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>false</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>false</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>false</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>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Complexity

- If the KB and $\alpha$ contain $n$ symbols in total, what is the time complexity of the truth table enumeration algorithm?
- Space complexity is $O(n)$ because the actual algorithm uses DFS

The really depressing news

- Every known inference algorithm for propositional logic has a worst-case complexity that is exponential in the size of the input
- But some algorithms are more efficient in practice

You can’t handle the truth!
Logical equivalence

- Intuitively: two sentences $\alpha$ and $\beta$ are logically equivalent (i.e. $\alpha \equiv \beta$) if they are true in the same set of models.
- Formally: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$.
- Can prove this with truth tables.

Standard Logic Equivalences

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg \neg \alpha & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \rightarrow \beta) & \equiv (\neg \beta \rightarrow \neg \alpha) \quad \text{contraposition} \\
(\alpha \leftrightarrow \beta) & \equiv (\alpha \lor \neg \beta) \quad \text{implication elimination} \\
(\alpha \equiv \beta) & \equiv ((\alpha \rightarrow \beta) \land (\beta \rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg (\alpha \lor \beta) & \equiv (\neg \alpha \land \neg \beta) \quad \text{de Morgan} \\
\neg (\alpha \land \beta) & \equiv (\neg \alpha \lor \neg \beta) \quad \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv (\alpha \land \beta) \lor (\alpha \land \gamma) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]

In the above, $\alpha$, $\beta$, and $\gamma$ are arbitrary sentences of propositional logic.

Validity

- A sentence is valid if it is true in all models.
- E.g. $P \lor \neg P$ is valid.
- Valid sentences = Tautologies.
- Tautologies are vacuous.

Deduction theorem

For any sentences $\alpha$ and $\beta$, $\alpha \models \beta$ iff the sentence $(\alpha \Rightarrow \beta)$ is valid.

Satisfiability

- A sentence is satisfiable if it is true in some model.
- A sentence is unsatisfiable if it is true in no models.
- Determining the satisfiability of sentences in propositional logic was the first problem proved to be NP-complete.
- Satisfiability is connected to validity: $\alpha$ is valid iff $\neg \alpha$ is unsatisfiable.
- Satisfiability is connected to entailment: $\alpha \models \beta$ iff the sentence $(\alpha \land \neg \beta)$ is unsatisfiable (proof by contradiction).

CW: Exercise

- Is the following sentence valid?
  
  \[(A \Rightarrow B) \lor (\neg A \Rightarrow \neg B)\]

Proof methods

How do we prove that $\alpha$ can be entailed from the KB?

1. Model checking e.g. check that $\alpha$ is true in all models in which KB is true.
2. Inference rules.
Inference Rules

1. Modus Ponens
   \[ \alpha \implies \beta, \quad \alpha \quad \implies \beta \]

2. And-Elimination
   \[ \alpha \land \beta \quad \implies \alpha \]

These are both sound inference rules. You don’t need to enumerate models now.

Other Inference Rules

- \((\alpha \land \beta) \equiv (\beta \land \alpha)\) commutativity of \(\land\)
- \((\alpha \lor \beta) \equiv (\beta \lor \alpha)\) commutativity of \(\lor\)
- \((\alpha \land (\beta \land \gamma)) \equiv (\alpha \land (\beta \land \gamma))\) associativity of \(\land\)
- \((\alpha \lor (\beta \lor \gamma)) \equiv (\alpha \lor (\beta \lor \gamma))\) associativity of \(\lor\)
- \(\neg(\neg \alpha) \equiv \alpha\) double-negation elimination
- \((\alpha \implies \beta) \equiv (\neg \beta \implies \neg \alpha)\) contraposition
- \((\alpha \implies \beta) \equiv (\neg \alpha \lor \beta)\) implication elimination
- \((\alpha \implies \beta) \equiv ((\alpha \implies \beta) \land (\beta \implies \alpha))\) biconditional elimination
- \((\neg(\neg \alpha) \lor (\alpha \lor \beta)) \equiv (\neg \alpha \lor \neg \beta)\) de Morgan
- \((\neg(\neg \alpha) \lor (\alpha \lor \beta)) \equiv (\neg \alpha \lor \neg \beta)\) de Morgan
- \((\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))\) distributivity of \(\lor\) over \(\land\)
- \((\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))\) distributivity of \(\lor\) over \(\land\)

All of the logical equivalences can be turned into inference rules e.g.

\[
\alpha \iff \beta \quad \implies \quad (\alpha \implies \beta) \land (\beta \implies \alpha)
\]

Example

Given the following KB, can we prove \(\neg R\)?

KB:

\[ P \implies \neg(Q \lor R) \]

\[ P \]

Proof:

\(\neg(Q \lor R)\) by Modus Ponens

\(\neg Q \land \neg R\) by De Morgan’s Law

\(\neg R\) by And-Elimination

Proofs

- A sequence of applications of inference rules is called a proof
- Instead of enumerating models, we can search for proofs
- Proofs ignore irrelevant propositions
- 2 methods:
  - Go forward from initial KB, applying inference rules to get to the goal sentence
  - Go backward from goal sentence to get to the KB

In-class Exercise

If it is October, there will not be a football game at OSU
If it is October and it is Saturday, I will be in Corvallis
If it doesn’t rain or if there is a football game, I will ride my bike to OSU
Today is Saturday and it is October
If I am in Corvallis, it will not rain

Can you prove that I will ride my bike to OSU?

Monotonicity

- Proofs only work because of monotonicity
- Monotonicity: the set of entailed sentences can only increase as information is added to the knowledge base
- For any sentences \(\alpha\) and \(\beta\), if \(KB \models \alpha\) then \(KB \land \beta \models \alpha\)
Resolution

• An inference rule that is sound and complete
• Forms the basis for a family of complete inference procedures
• Here, complete means refutation completeness: resolution can refute or confirm the truth of any sentence with respect to the KB

Resolution

• Here’s how resolution works (\(\neg l_2\) and \(l_2\) are called complementary literals):

\[
\frac{l_i \lor l_2, \quad \neg l_2 \lor l_i}{l_i \lor l_i}
\]

• Note that you need to remove multiple copies of literals (called factoring) i.e.

\[
\frac{l_i \lor l_2, \quad \neg l_i \lor l_i}{l_i}
\]

• If \(l_i\) and \(m_j\) are complementary literals, the full resolution rule looks like:

\[
\frac{l_i \lor \cdots \lor l_i, \quad m_i \lor \cdots \lor m_i}{l_i \lor \cdots \lor l_i, \lor \cdots \lor l_i \lor \cdots \lor m_i \lor \cdots \lor m_i}
\]

Conjunctive Normal Form

• Resolution only applies to sentences of the form \(l_1 \lor l_2 \lor \cdots \lor l_k\)
• This is called a disjunction of literals
• It turns out that every sentence of propositional logic is logically equivalent to a conjunction of disjunction of literals
• Called Conjunctive Normal Form or CNF
  e.g. \((l_1 \lor l_2 \lor l_3) \land (l_4 \lor l_5 \lor l_6) \land \cdots\)

• k-CNF sentences have exactly k literals per clause
  e.g. A 3-CNF sentence would be \((l_1 \lor l_2 \lor l_3) \land (l_4 \lor l_5 \lor l_6)\)

Recipe for Converting to CNF

1. Eliminate \(\leftrightarrow\), replacing \(\alpha \leftrightarrow \beta\) with \((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)\)
2. Eliminate \(\Rightarrow\), replacing \(\alpha \Rightarrow \beta\) with \(\neg \alpha \lor \beta\)
3. Move \(\neg\) inwards using:
   - \(\neg (\neg \alpha) = \alpha\) (double-negation elimination)
   - \(\neg (\alpha \land \beta) = \neg \alpha \lor \neg \beta\) (De Morgan’s Law)
   - \(\neg (\alpha \lor \beta) = \neg \alpha \land \neg \beta\) (De Morgan’s Law)
4. Apply distributive law \((\alpha \lor (\beta \land \gamma)) = ((\alpha \lor \beta) \land (\alpha \lor \gamma))\)

In-class Exercise

KB

<table>
<thead>
<tr>
<th>Person (\Rightarrow) Mortal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socrates (\Rightarrow) Person</td>
</tr>
</tbody>
</table>

Can we show that:

\[\text{KB} \models (\text{Socrates} \Rightarrow \text{Mortal})?\]

Exercise

• Convert the following sentence to CNF.

\[(B \lor C) \Rightarrow D\]
A resolution algorithm

To prove KB |= α, we show that (KB ∧ ¬α) is unsatisfiable
(Remember that α |= β iff the sentence (α ∧ ¬β) is unsatisfiable)

The algorithm:
1. Convert (KB ∧ ¬α) to CNF
2. Apply resolution rule to resulting clauses. Each pair with complementary literals is resolved to produce a new clause which is added to the KB
3. Keep going until
   - There are no new clauses that can be added (meaning KB |= α)
   - Two clauses resolve to yield the empty clause (meaning KB |= α)

The empty clause is equivalent to false because a disjunction is true only if one of its disjuncts is true.

In-class Exercise

KB
Person ⇒ Mortal
Socrates ⇒ Person

Can we show that:
KB |= (Socrates ⇒ Mortal)?

CW: Exercise

• Suppose the KB contains the following sentences in CNF.
  1. ¬C ∨ E
  2. ¬P ∨ E
  3. ¬E ∨ ¬R
  4. ¬A ∨ ¬P ∨ E
• Does KB |= ¬R?

Resolution Pseudocode

```java
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 C1, C2 in clauses do
      newResolvent ← PL-RESOLVE(C1, C2)
      if newResolvent contains the empty clause then return true
      new ← new ∪ newResolvent
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

Things you should know

• Understand the syntax and semantics of propositional logic
• Know how to do a proof in propositional logic using inference rules
• Know how to convert arbitrary sentences to CNF
• Know how resolution works