Propositional Logic

Reading: Chapter 7.1, 7.3 – 7.5

[Partially based on slides from Jerry Zhu, Louis Oliphant and Andrew Moore]

Logic

- If a problem domain can be represented formally, then a decision maker can use logical reasoning to make rational decisions.
- Several types of logic:
  - Propositional Logic (Boolean logic)
  - First-Order Logic (aka first-order predicate calculus)
  - Non-Monotonic Logic
  - Markov Logic
- A logic includes:
  - syntax: what is a correctly-formed sentence?
  - semantics: what is the meaning of a sentence?
  - Inference procedure (reasoning, entailment): what sentence logically follows given knowledge?

Propositional Logic

- A symbol in PL is a symbolic variable whose value must be either True or False, and which stands for a natural language statement that could be either true or false
  - A = “Smith has chest pain”
  - B = “Smith is depressed”
  - C = “It is raining”

Propositional Logic Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence</td>
<td>AtomicSentence</td>
</tr>
<tr>
<td>AtomicSentence</td>
<td>True</td>
</tr>
<tr>
<td>Symbol</td>
<td>P</td>
</tr>
<tr>
<td>ComplexSentence</td>
<td>¬</td>
</tr>
<tr>
<td></td>
<td>(</td>
</tr>
</tbody>
</table>

BNF (Backus-Naur Form) grammar for Propositional Logic

Well-formed sentence examples:

- \( \neg P \lor ((\text{True} \land R) \Rightarrow Q) \Rightarrow S \)
- \( \neg(P \lor Q) \land \Rightarrow S \)

Well-formed ("wff" or "sentence")

Not well-formed
Propositional Logic Syntax

- \((-P \lor ((\text{True} \land R) \iff Q)) \Rightarrow S)\)

- **Means True**
- **Means “Not”**
- **Means “Or” – disjunction**
- **Means “And” – conjunction**
- **Means “If-then” – implication**
- **Means “iff” – biconditional**

Propositional symbols must be specified

---

Semantics

- An **interpretation** is a complete True / False assignment to all propositional symbols
  - Example symbols: P means “It is hot”, Q means “It is humid”, R means “It is raining”
  - There are 8 interpretations (TTT, ..., FFF)
- The semantics (meaning) of a sentence is the set of interpretations in which the sentence evaluates to True
- Example: the semantics of the sentence P \lor Q is the set of 6 interpretations:
  - P=True, Q=True, R=True or False
  - P=True, Q=False, R=True or False
  - P=False, Q=True, R=True or False
- A **model** of a set of sentences is an interpretation in which all the sentences are true

---

Evaluating a Sentence under an Interpretation

- Calculated using the definitions of all the connectives, recursively

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>\neg P</th>
<th>P \land Q</th>
<th>P \lor Q</th>
<th>P \Rightarrow Q</th>
<th>P \iff Q</th>
</tr>
</thead>
<tbody>
<tr>
<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>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</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>
</tr>
</tbody>
</table>

- Pay attention to \(\Rightarrow\)
  - “5 is even implies 6 is odd” is True!
  - If P is False, regardless of Q, P \Rightarrow Q is True
  - No causality needed: “5 is odd implies the Sun is a star” is True
Understanding “⇒”

- This is an operator. Although we call it “implies” or “implication,” do not try to understand its semantic form from the name. We could have called it “foo” instead and still defined its semantics the same way.
- A ⇒ B “means” A is sufficient but not necessary to make B true.
- Example:
  - Let A be "has a cold" and B be "drink water"
  - A ⇒ B can be interpreted as "should drink water" when “has a cold.”
  - However, you can drink water even when you do not have a cold. Thus A ⇒ B is still true when A is not true.

Example

\((\neg P \lor (Q \land R)) \Rightarrow Q\)

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>\neg P</th>
<th>Q\land R</th>
<th>\neg P \lor (Q \land R)</th>
<th>\neg P \lor (Q \land R) \Rightarrow Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Satisfiable: a sentence that is true under some interpretation(s)

Deciding satisfiability of a sentence is NP-complete

Example

\(((P \land R) \Rightarrow Q) \land P \land R \land \neg Q\)
**Example**

\[ ((P \land R) \Rightarrow Q) \land P \land R \land \neg Q \]

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>\neg Q</th>
<th>R \land \neg Q</th>
<th>P \land R \land \neg Q</th>
<th>P \land R</th>
<th>P \land R \Rightarrow Q</th>
<th>final</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Unsatisfiable:** a sentence that is false under all interpretations

Also called inconsistent or a contradiction

**Example**

\[ (P \Rightarrow Q) \lor (P \land \neg Q) \]

**Knowledge Base (KB)**

- A knowledge base, KB, is a set of sentences
  
  Example KB:
  - ChuckGivingLecture \leftrightarrow (TodayIsMonday \lor TodayIsWednesday \lor TodayIsFriday)
  - \neg ChuckGivingLecture
  
  - It is equivalent to a single long sentence: the conjunction of all sentences
    - (ChuckGivingLecture \leftrightarrow (TodayIsMonday \lor TodayIsWednesday \lor TodayIsFriday)) \land \neg ChuckGivingLecture

- A model of a KB is an interpretation in which all sentences in KB are true
**Entailment**

- **Entailment** is the relation of a sentence $\beta$ \textit{logically following} from other sentences $\alpha$ (e.g., KB)
  \[ \alpha \models \beta \]
- $\alpha \models \beta$ if and only if, in every interpretation in which $\alpha$ is true, $\beta$ is also true; i.e., whenever $\alpha$ is true, so is $\beta$;
  all models of $\alpha$ and also models of $\beta$
- Deduction theorem: $\alpha \models \beta$ if and only if $\alpha \Rightarrow \beta$ is valid (always true)
- Proof by contradiction (refutation, \textit{reductio ad absurdum}): $\alpha \models \beta$ if and only if $\alpha \land \neg \beta$ is unsatisfiable
- There are $2^n$ interpretations to check, if KB has $n$ symbols

**Deductive Inference**

- Say you write a program that, according to you, proves whether a sentence $\beta$ is entailed by $\alpha$
- The thing your program does is called \textit{deductive inference}
- We don’t trust your inference program (yet), so we write things your program finds as
  \[ \alpha \vdash \beta \]
- It reads “$\beta$ is \textit{derived from} $\alpha$ by your program”
- What properties should your program have?
  - **Soundness**: the inference algorithm only derives entailed sentences. That is, if $\alpha \vdash \beta$ then $\alpha \models \beta$
  - **Completeness**: all entailment can be inferred.
    That is, if $\alpha \models \beta$ then $\alpha \vdash \beta$

**Soundness and Completeness**

- **Soundness** says that any wff that follows deductively from a set of axioms, KB, is valid (i.e., true in all models)
- **Completeness** says that all valid sentences (i.e., true in all models of KB), can be proved from KB and hence are theorems
Method 1: Inference by Enumeration

Also called Model Checking or Truth Table Enumeration

LET: \( KB = A \lor C, B \lor \neg C \) \( \beta = A \lor B \)

QUERY: \( KB \models \beta ? \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>( A \lor C )</th>
<th>( B \lor \neg C )</th>
<th>( KB )</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<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>true</td>
<td>false</td>
<td>true</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>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</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>
</tr>
</tbody>
</table>

\text{NOTE: The computer doesn't know the meaning of the proposition symbols}

So, all logically distinct cases must be checked to prove that a sentence can be derived from \( KB \)

Rows where all of sentences in \( KB \) are true are the models of \( KB \)

\( \beta \) is entailed by \( KB \) if all models of \( KB \) are models of \( \beta \), i.e., all rows where \( KB \) is true, \( \beta \) is also true

In other words: \( KB \models \beta \) is valid

Inference by Enumeration

\( \vdash \) is the symbol for entailment in propositional logic. It means that \( \beta \) is entailed by \( KB \).

\( \models \) is the symbol for logical consequence. It means that \( \beta \) is a logical consequence of \( KB \).

\( \beta \) is valid if and only if \( \vdash \beta \) and \( \models \beta \).
Method 2: Natural Deduction using Sound Inference Rules

Goal: Define a more efficient algorithm than enumeration that uses a set of inference rules to incrementally deduce new sentences that are true given the initial set of sentences in KB, plus uses all logical equivalences

Logical Equivalences

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

You can use these equivalences to derive or modify sentences

Sound Inference Rules

- **Modus Ponens** (Latin for "mode that affirms")
  \[
  \frac{\alpha \rightarrow \beta, \alpha}{\beta}
  \]

- **And-Elimination**
  \[
  \frac{\alpha \land \beta}{\alpha}
  \]

Note: Prove that an inference rule is **sound** by using a truth table, e.g.:

\[
\begin{array}{|c|c|c|c|c|}
\hline
P & Q & P \rightarrow Q & P \land (P \rightarrow Q) & Q \\
\hline
T & T & T & T & T \\
T & F & F & F & T \\
F & T & F & T & T \\
F & F & T & F & T \\
\hline
\end{array}
\]

- Implication-Elimination, IE (Modus Ponens, MP)
- And-Elimination, AE
- And-Introduction, AI
- Or-Introduction, OI
- Double-Negation Elimination, DNE
**Inference Rules**

- Each inference rule formalizes the idea that “A infers B” (A ⊢ B) in terms of “logically follows” (A ⊨ B)
- Doesn’t say anything about deductibility – just says for each interpretation that makes A true, that interpretation also makes B true

**Question**

What’s the difference between

≡ (logical equivalence)

⊨ (entails)

⊢ (derived from; infers)

**Natural Deduction = Constructing a Proof**

- A Proof is a sequence of inference steps that leads from α (i.e., KB) to β (i.e., query)
- This is a search problem!

KB:
1. (P ∧ Q) ⇒ R
2. (S ∧ T) ⇒ Q
3. S
4. T
5. P

Query:
R

**Proof by Natural Deduction**

1. S        Premise (i.e., given sentence in KB)
2. T        Premise
3. S ∧ T    Conjunction(1, 2) (And-Introduction)
4. (S ∧ T) ⇒ Q  Premise
5. Q        Modus Ponens(3, 4)
6. P        Premise
7. P ∧ Q    Conjunction(5, 6)
8. (P ∧ Q) ⇒ R  Premise
9. R        Modus Ponens(7, 8)  Last line is query sentence
Monotonicity Property

- Note that natural deduction relies on the monotonicity property of Propositional Logic:

  Deriving a new sentence and adding it to KB does NOT affect what can be entailed from the original KB

- Hence we can incrementally add new true sentences that are derived in any order

- Once something is proved true, it will remain true

Proof by Natural Deduction

KB:
1. TomGivingLecture ⇔ (TodayIsTuesday ∨ TodayIsThursday)
2. ¬ TomGivingLecture

Query:
¬ TodayIsTuesday

Resolution Rule of Inference

- Resolution Rule of Inference

\[
\frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma}
\]

- Examples

\[
\begin{align*}
A \lor B, \neg B & \quad A \lor B \lor \neg C \lor D, \neg A \lor \neg E \lor F \\
A & \quad B \lor \neg C \lor D \lor \neg E \lor F
\end{align*}
\]

called “unit resolution”
Resolution

• Take any two “clauses” where one contains some symbol, and the other contains its complement (negative)
  \[ P \lor Q \lor R \quad \neg Q \lor S \lor T \]

• Merge (resolve) them, by throwing away the symbol and its complement, to obtain their resolvent clause:
  \[ P \lor R \lor S \lor T \]

• If two clauses resolve and there’s no symbol left, you have derived False, aka the empty clause

Method 3: Resolution Refutation

• Show KB \models \alpha by proving that KB \land \neg \alpha is unsatisfiable, i.e., deducing False from KB \land \neg \alpha
• Your algorithm can use all the logical equivalences to derive new sentences, plus:
  • Resolution rule: a single inference rule
    ▪ Sound: only derives entailed sentences
    ▪ Complete: can derive any entailed sentence
    ▪ Resolution is refutation complete:
      \[ KB \models \beta, \text{ then } KB \land \neg \beta \models False \]
    ▪ But the sentences need to be preprocessed into a special form
    ▪ But all sentences can be converted into this form

Resolution Refutation Algorithm

1. Add negation of query to KB
2. Pick 2 sentences that haven’t been used before and can be used with the Resolution Rule of inference
3. If none, halt and answer that the query is NOT entailed by KB
4. Compute resolvent and add it to KB
5. If False in KB
   ▪ Then halt and answer that the query IS entailed by KB
   ▪ Else Goto 2

Conjunctive Normal Form (CNF)

1. Replace all \iff using iff/biconditional elimination
   ▪ \[ \alpha \iff \beta \equiv (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha) \]
2. Replace all \Rightarrow using implication elimination
   ▪ \[ \alpha \Rightarrow \beta \equiv \neg \alpha \lor \beta \]
3. Move all negations inward using
   ▪ double-negation elimination
     \[ \neg \neg \alpha \equiv \alpha \]
   ▪ de Morgan’s rule
     \[ \neg (\alpha \lor \beta) \equiv \neg \alpha \land \neg \beta \]
     \[ \neg (\alpha \land \beta) \equiv \neg \alpha \lor \neg \beta \]
4. Apply distributivity of \lor over \land
   ▪ \[ \alpha \land (\beta \lor \gamma) \equiv (\alpha \land \beta) \lor (\alpha \land \gamma) \quad + 1 \text{ more} \]
Convert Sentence into CNF

A ↔ (B ∨ C) starting sentence

\((A \implies (B \lor C)) \land ((B \lor C) \implies A)\) iff/biconditional elimination

\((\neg A \lor B \lor C) \land (\neg (B \lor C) \lor A)\) implication elimination

\((\neg A \lor B \lor C) \land (\neg B \land \neg C) \lor A\) move negations inward

\((\neg A \lor B \lor C) \land (\neg B \land \neg C) \land (\neg C \lor A)\) distribute \lor over \land

called a “clause”

Resolution Refutation Steps

• Given KB and \(\beta\) (query)
• Add \(\neg \beta\) to KB, and convert all sentences to CNF
• Show this leads to False (aka “empty clause”). Proof by contradiction
• Example KB:
  • \(A \leftrightarrow (B \lor C)\)
  • \(\neg A\)
• Example query: \(\neg B\)

Resolution Refutation Preprocessing

• Add \(\neg \beta\) to KB, and convert to CNF:

  a1: \(\neg A \lor B \lor C\)
  a2: \(\neg B \lor A\)
  a3: \(\neg C \lor A\)
  b: \(\neg A\)
  c: B

  • Want to reach goal: False (empty clause)

Resolution Refutation Example

<table>
<thead>
<tr>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1: (\neg A \lor B \lor C)</td>
</tr>
<tr>
<td>a2: (\neg B \lor A)</td>
</tr>
<tr>
<td>a3: (\neg C \lor A)</td>
</tr>
<tr>
<td>b: (\neg A)</td>
</tr>
<tr>
<td>c: B</td>
</tr>
</tbody>
</table>
Resolution Refutation Example

a1: \( \neg A \lor B \lor C \)
a2: \( \neg B \lor A \)
a3: \( \neg C \lor A \)
b: \( \neg A \)
c: \( B \)

Step 1: resolve a2, c: A

Step 2: resolve above and b: empty clause / false

Example

• Given:
  - \( P \lor Q \)
  - \( P \Rightarrow R \)
  - \( Q \Rightarrow R \)

• Prove: \( R \)

Example

• Given:
  - \( (P \Rightarrow Q) \Rightarrow Q \)
  - \( (P \Rightarrow P) \Rightarrow R \)
  - \( (R \Rightarrow S) \Rightarrow \neg(S \Rightarrow Q) \)

• Prove: \( R \)
Efficiency of the Resolution Refutation Algorithm

- Run time can be exponential in the worst case
  - Often much faster
- Factoring: if a new clause contains duplicates of the same symbol, delete the duplicates
  \[ P \lor R \lor P \lor T \equiv P \lor R \lor T \]
- If a clause contains a symbol and its complement, the clause is a tautology and is useless; it can be thrown away
  a1: \((\neg A \lor B \lor C)\)
  a2: \((\neg B \lor A)\)
  Resolvent of a1 and a2 is: \(B \lor C \lor \neg B\)
  Which is valid, so throw it away

Resolution Refutation Strategies

- Resolution refutation proofs can be thought of as search:
  - reversed construction of search tree (leaves to root)
  - leaves are KB clauses and \(\neg\)query
  - resolvent is new node with arcs to parent clauses
  - root is False clause

Resolution Refutation Strategies

- Breadth-First
  - level 0 clauses: KB clauses and \(\neg\)query
  - level k clauses: resolvents computed from 2 clauses:
    - one of which must be from level k-1
    - other from any earlier level
  - compute all possible level 1 clauses, then all possible level 2 clauses, etc.
  - complete but very inefficient

Resolution Refutation Strategies

- Input Resolution
  - \(P\) and \(Q\) can be resolved if at least one is from the set of original clauses, i.e., KB and \(\neg\)query
  - proof trees have a single "spine" (see Fig. 9.11)
  - Modus Ponens is a form of input resolution since each step is used to generate a new fact
  - complete for FOL KB with only Horn clauses
Resolution Refutation Strategies

- Linear Resolution
  - a slight generalization of input resolution
  - \( P \) and \( Q \) can be resolved if:
    - at least 1 is from the set of original clauses
    - or \( P \) must be an ancestor of \( Q \) in the proof tree
  - complete

Method 4: Chaining with Horn Clauses

- Resolution is more powerful than we need for many practical situations
- A weaker form: Horn clauses
  - A Horn clause is a disjunction of literals with at most one positive
    
    \[
    \neg R \lor \neg P \lor Q \quad \text{no}
    \]
    
    \[
    \neg R \lor \neg P \lor Q \quad \text{yes}
    \]

- \( KB \) = conjunction of Horn clauses
- What’s the big deal?
  - \( \neg R \lor \neg P \lor Q \)
  - \( \equiv \neg (R \land P) \lor Q \)
  - \( \equiv ? \)

Horn Clauses

\[
\neg R \lor \neg P \lor Q
\]

\[
\equiv \neg (R \land P) \lor Q
\]

Every rule in KB is in this form

- \( P \) (special case, no negative literals): fact
- \( \neg R \lor \neg P \) (special case, no positive literal): goal clause

The big deal:

- KB easy for humans to read
- Natural forward chaining and backward chaining algorithms; proof easy for humans to read
- Can decide entailment with Horn clauses in time \text{linear} with KB size

But …

- Can only ask atomic queries

Horn Clauses

- Only 1 rule of inference needed:

  \[
  \text{Generalized Modus Ponens}
  \]

  \[
  P, Q, (P \land Q) \Rightarrow R
  \]

  \[
  R
  \]
**Forward Chaining**

- "Apply" any rule whose premises are satisfied in the KB
- Add its conclusion to the KB until query is derived

```
P ⇒ Q
L ∧ M ⇒ P
B ∧ L ⇒ M
A ∧ P ⇒ L
A ∧ B ⇒ L
A
B
```

query: Q

- Forward chaining with Horn clause KB is complete

---

**Backward Chaining**

- Forward chaining problem: can generate a lot of irrelevant conclusions
  - Search forward, start state = KB, goal test = state contains query
- Backward chaining
  - Work backwards from goal to premises
  - Find all implications of the form
    ```
    (...) ⇒ query
    ```
  - Prove all the premises of one of these implications
  - Avoid loops: check if new subgoal is already on the goal stack
  - Avoid repeated work: check if new subgoal
    1. Has already been proved true, or
    2. Has already failed
Backward Chaining

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Backward Chaining

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

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

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

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
Backward Chaining

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

Forward vs. Backward Chaining

- **Forward chaining** is data-driven
  - May perform lots of work irrelevant to the goal
- **Backward chaining** is goal-driven
  - Appropriate for problem solving
  - Time complexity can be much less than linear in size of KB
- Some form of bi-directional search may be even better

Prolog: A Logic Programming Language

- A Program =
  - a set of logic sentences as Horn clauses
    - called the database (DB), i.e., the KB
    - ordered by programmer
  - executed by specifying a query to be proved
    - uses **backward-chaining**
    - uses **depth-first search** on the ordered facts and rules
    - searches until a solution is found
Prolog Basic Syntax

- **Database:**
  - **Fact:** a positive literal
    
    \[ F(x) \quad \text{in Prolog:} \ F(X). \]
    
    initial capital variables universally quantified
  - **Rules:** \( \geq 1 \) negative and 1 positive literals
    if antecedent(s) then consequent
    
    \[ A_1 \wedge A_2 \wedge \ldots \wedge A_n \Rightarrow C \quad \text{in Prolog:} \quad C :- A_1, A_2, \ldots, A_n. \]

- **Query:**
  - **FOL:** \( Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \)
    in Prolog: \( ?- Q_1, Q_2, \ldots, Q_n. \)
  - query variables implicitly existentially quantified

Problems with Propositional Logic

- Consider the game “minesweeper” on a 10 x 10 field with only one land mine

- How do you express the knowledge, with Propositional Logic, that the squares adjacent to the land mine will display the number 1?

- Intuitively with a rule like

\[
\text{Landmine}(x,y) \Rightarrow \text{Number1}(\text{Neighbors}(x,y))
\]

but Propositional Logic **cannot** do this

Problems with Propositional Logic

- Propositional Logic has to say, e.g. for cell (3, 4):

\[
\begin{align*}
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_2_3 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_2_4 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_2_5 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_3_3 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_3_5 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_4_3 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_4_4 \\
\text{Landmine}_3_4 & \Rightarrow \text{Number1}_4_5 \\
\end{align*}
\]

- And similarly for each of Landmine_1_1, Landmine_1_2, Landmine_1_3, ..., Landmine_10_10

- Difficult to express large domains concisely
- Don’t have objects and relations
- First-Order Logic is a powerful upgrade
Other Logic Systems

Logics are characterized by what they commit to as "primitives"

<table>
<thead>
<tr>
<th>Logic</th>
<th>What Exists in World</th>
<th>Knowledge States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional</td>
<td>facts</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>First-Order</td>
<td>facts, objects, relations</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Temporal</td>
<td>facts, objects, relations, times</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Probability Theory</td>
<td>facts</td>
<td>degree of belief 0..1</td>
</tr>
<tr>
<td>Markov</td>
<td>facts, objects, relations</td>
<td>degree of belief 0..1</td>
</tr>
</tbody>
</table>

First-Order Logic (FOL)

Also known as First-Order Predicate Calculus (FOPC)

- Constants: Bob, 2, Madison, ...
- Functions: Income, Address, Sqrt, ...
- Predicates: Sister, Teacher, ≤, ...
- Variables: x, y, a, b, c, ...
- Connectives: ∧, ∨, ¬, ⇒, ⇔
- Equality: =
- Quantifiers: ∀, ∃

FOL Syntax: Quantifiers

**Universal quantifier:** ∀<variable> <sentence>

- Means the sentence is true for all values of x in the domain of variable x
- Main connective typically ⇒ forming if-then rules
  - All humans are mammals becomes in FOL:
    \[ ∀x \text{ Human}(x) \Rightarrow \text{Mammal}(x) \]
    i.e., for all x, if x is a human then x is a mammal
  - Mammals must have fur becomes in FOL:
    \[ ∀x \text{ Mammal}(x) \Rightarrow \text{HasFur}(x) \]
    for all x, if x is a mammal then x has fur

FOL Syntax: Quantifiers

**Existential quantifier:** ∃<variable> <sentence>

- Means the sentence is true for some value of x in the domain of variable x
- Main connective is typically ∧
  - Some humans are old becomes in FOL:
    \[ ∃x \text{ Human}(x) \land \text{Old}(x) \]
    there exist an x such that x is a human and x is old
  - Mammals may have arms. becomes in FOL:
    \[ ∃x \text{ Mammal}(x) \land \text{HasArms}(x) \]
    there exist an x such that x is a mammal and x has arms
Fun with Sentences

- Good people always have friends.
  
  **could mean:** All good people have friends.
  
  \[ \forall x \ (\text{Person}(x) \land \text{Good}(x)) \Rightarrow \exists y (\text{Friend}(x,y)) \]

- Busy people sometimes have friends.
  
  **could mean:** Some busy people have friends.
  
  \[ \exists x \ (\text{Person}(x) \land \text{Busy}(x) \land \exists y (\text{Friend}(x,y)) \]

- Bad people never have friends.
  
  **could mean:** Bad people have no friends.
  
  \[ \forall x \ (\text{Person}(x) \land \text{Bad}(x)) \Rightarrow \neg \exists y (\text{Friend}(x,y)) \]  
  or equivalently: No bad people have friends.
  
  \[ \neg \exists x \ (\text{Person}(x) \land \text{Bad}(x) \land \exists y (\text{Friend}(x,y)) \]

---

What You Should Know

- A lot of terms
- Use truth tables (inference by enumeration)
- Natural deduction proofs
- Conjunctive Normal Form (CNF)
- Resolution Refutation algorithm and proofs
- Horn clauses
- Forward chaining algorithm
- Backward chaining algorithm

---

Fun with Sentences

- There is exactly one shoe.
  
  \[ \exists x \ (\text{Shoe}(x) \land \forall y (\text{Shoe}(y) \Rightarrow (x=y)) \]

- There are exactly two shoes.
  
  \[ \exists x, y \ (\text{Shoe}(x) \land \text{Shoe}(y) \land \neg (x=y) \land \forall z (\text{Shoe}(z) \Rightarrow (x=z) \lor (y=z)) \]