Problem Solving

Two methods for solving problems:

- **Procedural**
  - devise an algorithm
  - program the algorithm
  - execute the program

- **Declarative**
  - identify the knowledge needed
  - encode the knowledge in a representation (knowledge base - KB)
  - use logical consequences of KB to solve the problem

Readings: Poole & Mackworth 2nd ed. chapter 5.1-5.3, and 13.1-13.2
### Logical Connectives

- **and (conjunction)** $\land$
- **or (disjunction)** $\lor$
- **not (negation)** $\neg$
- **if . . . then . . . (implication)** $\rightarrow$
- **... if and only if . . .** $\leftrightarrow$

### Implication Truth Table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A \rightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

(A) (B)

**Note:** often logical statements with implication are written backwards: $A \rightarrow B$ is the same as $B \leftarrow A$.

**If it rains, then I will carry an umbrella**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A \rightarrow B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

(A) (B)

**If it rains, then I will carry an umbrella**

**If you don't study, then you will fail**

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A \rightarrow B$</th>
<th>$A \land \neg B$</th>
<th>$\neg (A \land \neg B)$</th>
<th>$\neg A \lor B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</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>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

(A) (B)

**no rain or I will carry an umbrella**

**study or you will fail**
If and only if Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A ↔ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\[ A \leftrightarrow B \equiv (A \rightarrow B) \land (B \rightarrow A) \]

De Morgan’s Laws

\[ A \lor B \equiv \neg (\neg A \land \neg B) \]

it rains OR I play football

not true that ( it doesn’t rain AND I don’t play football )

\[ A \land B \equiv \neg (\neg A \lor \neg B) \]

I’m a politician AND I lie

not true that ( I’m not a politician OR I tell the truth)

---

Modus Ponens

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
<th>(A → B) ∧ A</th>
<th>((A → B) ∧ A) → B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</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>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Modus Ponens is a Tautology
If it’s raining then the grass is wet
it’s raining
therefore the grass is wet

---

Modus Tolens

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
<th>(A → B) ∧ B</th>
<th>((A → B) ∧ B) → ¬A</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</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>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Modus Tolens is a Tautology
If it’s raining then the grass is wet
the grass is not wet
therefore it’s not raining
Modus Bogus

A countercase is **not** a Tautology
If it’s raining then the grass is wet
the grass is wet
therefore its raining

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
<th>(A → B) ∧ B</th>
<th>((A → B) ∧ B) → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Logical Consequence

A statement, A, is a logical consequence of a set of statements \( \{X\} \), if A is true in every model of \( \{X\} \).

If, for every set of truth assignments that hold for \( \{X\} \) (for every model of \( \{X\} \)), some other statement (A) is always true, then this other statement is a logical consequence of \( \{X\} \).

Argument Validity

An argument is **valid** if any of the following is true:

- the conclusions are a logical consequence of the premises.
- the conclusions are true in every model of the premises
- there is no situation in which the premises are all true, but the conclusions are false.
- argument → conclusions is a tautology (always true)

(These four statements are identical)
Arguments and Models

P1: If I play hockey, then I'll score a goal if the goalie is not good
P2: If I play hockey, the goalie is not good
D: Therefore, if I play hockey, I'll score a goal

P: I play hockey
C: I'll score a goal
H: the goalie is good

\[ P_1 : P \to (\neg H \to C) \quad P_2 : P \to \neg H \]
\[ D : P \to C \]

<table>
<thead>
<tr>
<th>P</th>
<th>C</th>
<th>H</th>
<th>\neg H \to C</th>
<th>P1</th>
<th>P2</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</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>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<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>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Each row is an interpretation: an assignment of T/F to each proposition. In all the green lines, the premises are true: these interpretations are models of \( P_1 \) and \( P_2 \). Every model of \( P_1 \) and \( P_2 \) is a model of \( D \). Therefore, \( D \) is a logical consequence of \( P_1 \) and \( P_2 \):

\[ P_1, P_2 \models D. \]

Logical Consequence

P1: Elvis is Dead
P2: Elvis is Not Dead
D: Therefore, Jerry is Alive

E: Elvis is Alive
J: Jerry is Alive

\[ P_1 : P \to (\neg H \to C) \quad P_2 : P \to \neg H \]
\[ D : P \to C \]

<table>
<thead>
<tr>
<th>P</th>
<th>C</th>
<th>H</th>
<th>\neg H \to C</th>
<th>P1</th>
<th>P2</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</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>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<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>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

Is this argument valid?
Logical Consequence

P1: Elvis is Dead
P2: Elvis is Not Dead
D: Therefore, Jerry is Alive

Is this argument valid?
Yes!
E: Elvis is Alive
J: Jerry is Alive

<table>
<thead>
<tr>
<th>E</th>
<th>¬E</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

An argument is valid if there is no situation in which the premises are all true, but the conclusions are false. But here, there is no model of the premises, so the argument is valid.

Deduction and Proof

Given a knowledge base, we want to prove things that are true. We can use

- Truth Table
- Natural Deduction
- Semantic Tableaux
- Axiomatic Logic (Modus Ponens)

((A → B) ∧ A) → B

- Resolution Refutation (Reductio Ad Absurdum)

(¬A) ∧ ... ∧ ... → ⊥) → A

Proofs

A Knowledge Base (KB) is a set of axioms
A proof procedure is a way of proving theorems
KB ⊢ g means g can be derived from KB using the proof procedure
If KB ⊢ g, then g is a theorem
A proof procedure is sound: if KB ⊢ g then KB ⊨ g.
A proof procedure is complete: if KB ⊨ g then KB ⊢ g.

Complete Knowledge

- we assume a closed world:
  - the agent knows everything (or can prove everything)
  - if it can’t prove something: must be false
  - negation as failure

- other option is an open world:
  - the agent doesn’t know everything
  - can’t conclude anything from a lack of knowledge
Bottom-up proof

also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\text{rain} \leftarrow \text{clouds} \land \text{wind}.
\]
\[
\text{clouds} \leftarrow \text{humid} \land \text{cyclone}.
\]
\[
\text{clouds} \leftarrow \text{near}_{\text{sea}} \land \text{cyclone}.
\]
\[
\text{wind} \leftarrow \text{cyclone}.
\]
\[
\text{near}_{\text{sea}}.
\]
\[
\text{cyclone}.
\]

\{\text{near}_{\text{sea}},\text{cyclone} \}\{
\text{near}_{\text{sea}},\text{cyclone},\text{wind} \}\{
\text{near}_{\text{sea}},\text{cyclone},\text{wind},\text{clouds} \}\{
\text{near}_{\text{sea}},\text{cyclone},\text{wind},\text{clouds},\text{rain} \}
also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}.
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}.
\text{clouds} & \leftarrow \text{near}\_\text{sea} \land \text{cyclone}.
\text{wind} & \leftarrow \text{cyclone}.
\text{near}\_\text{sea}.
\text{cyclone}.
\{\text{near}\_\text{sea},\text{cyclone}\} \\
\{\text{near}\_\text{sea},\text{cyclone},\text{wind}\} \\
\{\text{near}\_\text{sea},\text{cyclone},\text{wind},\text{clouds}\} \\
\{\text{near}\_\text{sea},\text{cyclone},\text{wind},\text{clouds},\text{rain}\}
\end{align*}
\]

\[
C := \{\};
\text{repeat}
\hspace{1em} \text{select } r \in KB \text{ such that}
\hspace{1em} \cdot \ r \text{ is } h \leftarrow b_1 \land \ldots \land b_m
\hspace{1em} \cdot \ b_i \in C \ \forall \ i
\hspace{1em} \cdot \ h \notin C
\hspace{1em} C := C \cup \{h\}
\text{until no more clauses can be selected}
\]

---

**Top-Down Proof**

start from query and work backwards

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}.
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}.
\text{clouds} & \leftarrow \text{near}\_\text{sea} \land \text{cyclone}.
\text{wind} & \leftarrow \text{cyclone}.
\text{near}\_\text{sea}.
\text{cyclone}.
\text{Start with query: if } \text{rain} \text{ is proved, "yes" is the logical result (the answer to the question)}
\text{yes} & \leftarrow \text{rain}.
\end{align*}
\]

---

**Bottom-up proof**

Bottom-up proof

also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}.
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}.
\text{clouds} & \leftarrow \text{near}\_\text{sea} \land \text{cyclone}.
\text{wind} & \leftarrow \text{cyclone}.
\text{near}\_\text{sea}.
\text{cyclone}.
\{\text{near}\_\text{sea},\text{cyclone}\} \\
\{\text{near}\_\text{sea},\text{cyclone},\text{wind}\} \\
\{\text{near}\_\text{sea},\text{cyclone},\text{wind},\text{clouds}\} \\
\{\text{near}\_\text{sea},\text{cyclone},\text{wind},\text{clouds},\text{rain}\}
\end{align*}
\]

---

**Top-Down Proof**

start from query and work backwards

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}.
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}.
\text{clouds} & \leftarrow \text{near}\_\text{sea} \land \text{cyclone}.
\text{wind} & \leftarrow \text{cyclone}.
\text{near}\_\text{sea}.
\text{cyclone}.
\text{Start with query: if } \text{rain} \text{ is proved, "yes" is the logical result (the answer to the question)}
\text{yes} & \leftarrow \text{rain}.
\end{align*}
\]
start from query and work backwards

rain ← clouds ∧ wind.
clouds ← humid ∧ cyclone.
clouds ← near_sea ∧ cyclone.
wind ← cyclone.
near_sea.
cyclone.

yes ← rain.
yes ← clouds ∧ wind

yes ← near_sea ∧ cyclone ∧ wind
yes ← near_sea ∧ cyclone ∧ cyclone

start from query and work backwards

rain ← clouds ∧ wind.
clouds ← humid ∧ cyclone.
clouds ← near_sea ∧ cyclone.
wind ← cyclone.
near_sea.
cyclone.

yes ← rain.
yes ← clouds ∧ wind
yes ← near_sea ∧ cyclone ∧ wind
yes ← near_sea ∧ cyclone ∧ cyclone
yes ← near_sea ∧ cyclone ∧ cyclone
Top-Down Proof

start from query and work backwards

\[ \text{rain} \leftarrow \text{clouds} \land \text{wind}. \]
\[ \text{clouds} \leftarrow \text{humid} \land \text{cyclone}. \]
\[ \text{clouds} \leftarrow \text{near}\_\text{sea} \land \text{cyclone}. \]
\[ \text{wind} \leftarrow \text{cyclone}. \]
\[ \text{near}\_\text{sea}. \]
\[ \text{cyclone}. \]

\[ \text{yes} \leftarrow \text{rain}. \]
\[ \text{yes} \leftarrow \text{clouds} \land \text{wind}. \]
\[ \text{yes} \leftarrow \text{near}\_\text{sea} \land \text{cyclone} \land \text{wind}. \]
\[ \text{yes} \leftarrow \text{near}\_\text{sea} \land \text{cyclone} \land \text{cyclone}. \]
\[ \text{yes} \leftarrow \text{near}\_\text{sea} \land \text{cyclone}. \]
\[ \text{yes} \leftarrow \text{cyclone}. \]

Towards Automated Methods

A proof procedure gives us a method for deriving theorems.
Therefore, given a knowledge base of assumptions, we can 'prove' things and know they are tautologies (they are logical consequences of our knowledge base).

but ....
The method is difficult and requires some know-how - how could we make it work more automatically?

Top-Down Interpreter

\[
\text{solve}(q_1 \land \ldots \land q_k):
\begin{align*}
ac & := \text{"yes} \leftarrow q_1 \land \ldots \land q_k\text{"} \\
\text{repeat} \\
& \quad \text{select a conjunct } q_i \text{ from body of } ac \\
& \quad \text{choose a clause } C \text{ from KB with } q_i \text{ as head} \\
& \quad \text{replace } q_i \text{ in body of } ac \text{ by body of } C \\
& \quad \text{until } ac \text{ is an answer}
\end{align*}
\]
A well-formed formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of atoms.

\[(p_1 \lor p_2) \land (p_3 \lor p_4 \lor p_5) \land (p_6 \lor p_7 \lor \ldots) \land (p_{n-1} \lor p_n)\]

Convert a propositional formula to CNF:
1. Eliminate \(\leftrightarrow\) using \(A \leftrightarrow B \equiv (A \rightarrow B) \land (B \rightarrow A)\)
2. Eliminate \(\rightarrow\) using \(A \rightarrow B \equiv \neg A \lor B\)
3. Use deMorgan’s laws to push \(\neg\) into atoms
4. Use \(\neg
\neg A \equiv A\) to eliminate double negatives
5. Use distributive law to complete \(A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)\)

write \((p_1 \lor p_2) \land (p_3 \lor p_4 \lor p_5) \land (p_6 \lor p_7 \lor \ldots) \land (p_{n-1} \lor p_n)\) as \(\{\{p_1, p_2\}, \{p_3, p_4, p_5\}, \{p_6, p_7, \ldots\}, \ldots, \{p_{n-1}, p_n\}\}\)

**Conjunctive Normal Form - Example 1**

Refutation of Modus Ponens

\[A \land (A \rightarrow B) \vdash B\]

show a contradiction \(\bot\): means “false”

If our refutation leads to a contradiction, it must be “false”, so the conclusion must be true

\[A \land (A \rightarrow B) \land \neg B \vdash \bot\]

1. \(\neg A \lor B \land \neg B \lor C \land \neg (\neg A \lor C)\)
2. \(\neg A \lor B \land \neg B \lor C \land (\neg \neg A \land \neg C)\)
3. \(\neg A \lor B \land \neg B \lor C \land A \land \neg C\)
4. \(\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}\)

We will demonstrate using resolution on slide 28

**Conjunctive Normal Form - Example 2**

Transitivity of Implication

\[((A \rightarrow B) \land (B \rightarrow C)) \rightarrow (A \rightarrow C)\]

try to show a contradiction

\[(A \rightarrow B) \land (B \rightarrow C) \land \neg (A \rightarrow C) \vdash \bot\]

1. \(\neg A \lor B \land \neg B \lor C \land \neg (\neg A \lor C)\)
2. \(\neg A \lor B \land \neg B \lor C \land (\neg \neg A \land \neg C)\)
3. \(\neg A \lor B \land \neg B \lor C \land A \land \neg C\)
4. \(\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}\)

- A complementary pair of propositions is \(p_i, \neg p_i\)
- Can show that two clauses with a complementary pair: \(\{\{A, B\}, \{C, \neg B\}\} \equiv \{\{A, B\}, \{C, \neg B\}, \{A, C\}\}\)
- That is, since \(B\) and \(\neg B\) cannot both be true, one of \(A\) or \(C\) has to be true, otherwise the whole formula is false
- Therefore, we can resolve \(\{A, B\}, \{C, \neg B\}\) into \(\{A, C\}\)
- This means that \(\{A, C\}\) is true whenever \(\{A, B\}, \{C, \neg B\}\) is true
- So we can add \(\{A, C\}\) to the statement without changing the truth value
- \(\{\{A\}, \{\neg A\}\}\) resolves to \(\bot\)
Resolution

- Proof by resolution refutation: deny the conclusions and show a resolution to ⊥.
- Resolve clauses - adds new clauses that are true whenever the existing clauses are true
- If you can find a contradiction, then
  - the existing clauses cannot all be true
  - If the premises are all true, the refutation of the conclusion must be false,
  - so the argument is valid
- If you cannot find a contradiction after resolving all clauses
  - the refutation of the conclusion must be true
  - so the argument is invalid

Resolution - Example 1

Refutation of Modus Ponens

\[ A \land (A \rightarrow B) \vdash B \]

show a contradiction

\[ A \land (A \rightarrow B) \land \neg B \vdash \bot \]

1. \[ A \land (\neg A \lor B) \land (\neg B) \]
2. \[ \{\{A\}, \{\neg A, B\}, \{\neg B\}\} \]
3. \[ \{\{A\}, \{\neg A, B\}, \{B\}, \{\neg B\}\} \]
4. \[ \bot \]

Resolution - Example 2

Transitivity of Implication (again)

\[ ((A \rightarrow B) \land (B \rightarrow C)) \rightarrow (A \rightarrow C) \]

try to show a contradiction

\[ (A \rightarrow B) \land (B \rightarrow C) \land \neg(A \rightarrow C) \vdash \bot \]

1. \[ (\neg A \lor B) \land (\neg B \lor C) \land \neg(\neg A \lor C) \]
2. \[ (\neg A \lor B) \land (\neg B \lor C) \land (\neg A \land \neg C) \]
3. \[ (\neg A \lor B) \land (\neg B \lor C) \land A \land \neg C \]
4. \[ \{\{\neg A, B\}, \{\neg B, C\}, \{A\}, \{\neg C\}\} \]
5. \[ \{\{\neg A, B\}, \{\neg B, C\}, \{\neg A, C\}, \{A\}, \{\neg C\}\} \]
6. \[ \{\{\neg A, B\}, \{\neg B, C\}, \{\neg A, C\}, \{A\}, \{C\}, \{\neg C\}\} \]
7. \[ \bot \]

Resolution - Example 3

P1: If I play hockey, then I’ll score a goal if the goalie is not good
P2: If I play hockey, the goalie is not good
D: if I play hockey, I’ll score a goal

P: I play hockey, C: I’ll score a goal, H: the goalie is good
\[ P1 : P \rightarrow (\neg H \rightarrow C) \quad P2 : P \rightarrow \neg H \quad D : P \rightarrow C \]

test (refutation of D): \[ P1 \land P2 \land \neg D \]

\[
(P \rightarrow (\neg H \rightarrow C)) \land (P \rightarrow \neg H) \land \neg(P \rightarrow C)
\]

\[
(\neg P \lor (\neg H \rightarrow C)) \land (\neg P \lor \neg H) \land (\neg P \lor \neg C)
\]

\[
(\neg P \lor \neg H \lor C) \land (\neg P \lor \neg H) \land (\neg P \lor \neg C)
\]

\[
(\neg P \lor H \lor C) \land (\neg P \lor H) \land (P \lor \neg C)
\]

\[
\{\{\neg P, H, C\}, \{\neg P, \neg H\}, \{P\}, \{\neg C\}\}
\]

\[
\{\{\neg P, H, C\}, \{\neg P, \neg H\}, \{\neg P, C\}, \{P\}, \{\neg C\}\}
\]

\[
\{\{\neg P, H, C\}, \{\neg P, \neg H\}, \{\neg P, C\}, \{P\}, \{\neg C\}\}
\]
rain ← clouds ∧ wind.
clouds ← humid ∧ cyclone.
clouds ← near_sea ∧ cyclone.
wind ← cyclone.
near_sea.
cyclone.

prove rain by converting to CNF and resolving

Many problems can be formulated as a CNF
- Satisfiability
- Logic circuits
- Gene decoding
- Scheduling
- Air traffic control
- ...

A CSP variable $Y$ with domain $\{v_1, \ldots, v_k\}$ can be converted into $k$ Boolean variables $\{Y_1, \ldots, Y_k\}$ where $Y_i$ is true when $Y$ has value $v_i$ and false otherwise.

Thus, $k$ atoms $y_1, \ldots, y_k$ are used to represent the CSP variable

Constraints:
- exactly one of $y_1, \ldots, y_k$ must be true:
  - $y_i$ and $y_j$ cannot both be true when $i \neq j$: $\neg y_i \lor \neg y_j$ for $i < j$
  - at least one of the $y_i$ must be true: $y_1 \lor \ldots \lor y_k$
- There is a clause for each false assignment in each constraint that specifies which assignments are not allowed.
- Thus, if there are two variables $Y$ and $Z$, and a constraint $Y \neq Z$, then we have clauses $\neg y_i \lor \neg z_i$ for all $i$ (Assuming $Y$ and $Z$ have the same domains).

Example Delivery robot: activities $a, b$, times $1, 2, 3, 4$.

constraints:
$(A \neq 2) \land (B \neq 1) \land (A < B)$

We have two 8 variables in the CNF:

$a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$

where $a_i$ means $A = i$ is true and $b_i$ means $B = i$ is true.

Constraints saying that $A$ (and $B$) must be exactly one value:

$\neg a_i \lor \neg a_j$ for $i < j$

$\neg b_i \lor \neg b_j$ for $i < j$

Domain constraints $\neg a_2$ and $\neg b_1$

The binary constraint $A < B$ has one $\neg(a_i \land b_j)$ for all $j \leq i$
Beyond propositions: Individuals and Relations

First order example

KB can contain relations: `part_of(C,A)` is true if C is a "part of" A (in the world)

KB can contain quantification: `part_of(C,A)` holds `∀C,A`

proof procedure is the same, with a few extra bits to handle relations & quantification

```prolog
symptom(runny_nose,flu).
symptom(fever,flu).
symptom(fever,hepatitis).
symptom(chills,flu).
symptom(chills,hypothermia).
symptom(aches,flu).
symptom(rash,hepatitis).
has_symptom(john,fever).
has_symptom(john,runny_nose).
has_symptom(mary,chills).
has_symptom(mary,rash).

has_condition(Person,Condition):-
symptom(Symptom,Condition),
has_symptom(Person,Symptom).
```

MIU Puzzle

Symbols: M, I, U

Axiom: MI

Rules:
- if xI is a theorem, so is xIU
- Mxx is a theorem, so is Mxx
- in any theorem, III can be replaced by U
- UU can be dropped from any string

Starting from MI, can you generate MU? (use top-down or bottom-up)

Next:

- Planning under certainty (Poole & Mackworth 2nd ed. Chapter 6.1-6.4)
- Supervised Learning (Poole & Mackworth 2nd ed. Chapter 7.1-7.6)