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

Proof Procedures

A logic consists of
- **syntax**: what is an acceptable sentence?
- **semantics**: what do the sentences and symbols mean?
- **proof procedure**: how do we construct valid proofs?

A proof: a sequence of sentences derivable using an inference rule

Logical Connectives

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

If it rains, then I will carry an umbrella
### Implication Truth Table

<table>
<thead>
<tr>
<th></th>
<th></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>

\( (A) \quad (B) \)

- If it rains, then I will carry an umbrella
- If you don’t study, then you will fail

### 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></th>
<th></th>
<th>( A \to B )</th>
<th>((A \to B) \land A )</th>
<th>((A \to B) \land A \to 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 Tollens

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

Modus Tollens is a **Tautology**

- If it’s raining then the grass is wet
- The grass is not wet
- Therefore it’s not raining
Modus Bogus

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A → B</th>
<th>(A → B) \land B</th>
<th>((A → B) \land 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>

Modus Bogus is not a Tautology

If it’s raining then the grass is wet
the grass is wet
therefore its raining

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

P1 : P \rightarrow (\neg H \rightarrow C)  \quad P2 : P \rightarrow \neg H  \quad D : P \rightarrow C

<table>
<thead>
<tr>
<th>P</th>
<th>C</th>
<th>H</th>
<th>\neg H \rightarrow 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>F</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>F</td>
<td>F</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>

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\}.

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

<table>
<thead>
<tr>
<th>P</th>
<th>C</th>
<th>H</th>
<th>( \neg H \rightarrow 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>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</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>F</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 P1 and P2.
Every model of P1 and P2 is a model of D.
Therefore, D is a logical consequence of P1 and P2:
\( P_1, P_2 \models D \).

Logical Consequence

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

Is this argument valid?
Yes!
E: Elvis is Alive
G: Gerry is Alive

<table>
<thead>
<tr>
<th>E</th>
<th>( \neg E )</th>
<th>G</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)
  \( (A \rightarrow B) \land A \rightarrow B \)
- Resolution Refutation (Reductio Ad Absurdum)
  \( (\neg A) \land \ldots \land \neg A \rightarrow \bot \rightarrow A \)

Proofs

- A KB is a set of axioms
- A proof procedure is a way of Proving Theorems
- KB \( \vdash g \) means g can be derived from KB using the proof procedure
- If KB \( \vdash g \), then g is a Theorem
- A proof procedure is sound:
  if KB \( \vdash g \) then KB \( \models g \).
- A proof procedure is complete:
  if KB \( \models g \) then KB \( \vdash g \).
- Two types of proof procedures: bottom up and top down

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 (no variables)
also known as **forward chaining** - start from facts and use rules to generate all possible atoms

```

rain ← clouds ∧ wind.
clouds ← humid ∧ cyclone.
clouds ← near_sea ∧ cyclone.
wind ← cyclone.
near_sea.
cyclone.
{near_sea,cyclone }
{near_sea ,cyclone ,wind }
{near_sea ,cyclone ,wind ,clouds }
{near_sea ,cyclone ,wind ,clouds ,rain }
```
Top-Down Proof (no variables)

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
yes ← cyclone
yes ← near_sea.

Top-Down Proof (no variables)

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
yes ← cyclone
yes ← near_sea.

Top-Down Proof (no variables)

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
yes ← cyclone
yes ← near_sea.
Top-Down Proof (no variables)

start from query and work backwards

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

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

Top-Down Interpreter (no variables)

solve(q₁ ∧...∧ qₖ):
   ac := “yes ← q₁ ∧...∧ qₖ”
   repeat
      select a conjunct aᵢ from body of ac
      choose a clause C from KB with aᵢ as head
      replace aᵢ in body of ac by body of C
   until ac is an answer

Top-Down Proof (no variables)

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.

Top-Down Proof (no variables)

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter
Top-Down Proof (no variables)

start from query and work backwards
health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← bread ∧ water ∧ shelter

Top-Down Proof (no variables)

start from query and work backwards
health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← bread ∧ water ∧ shelter
yes ← bread ∧ water ∧ house
yes ← bread ∧ water
yes ← bread

Top-Down Proof (no variables)

start from query and work backwards
health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← bread ∧ water ∧ shelter
yes ← bread ∧ water ∧ house
yes ← bread ∧ water
yes ← bread

Top-Down Interpreter (no variables)

solve(q1 ∧...∧qk):
ac := “yes ← q1 ∧...∧qk”
repeat
    select a conjunct ai from body of ac
    choose a clause C from KB with ai as head
    replace ai in body of ac by body of C
until ac is an answer

select: “don’t care nondeterminism”
(If one doesn’t give a solution, no point trying others!)
choose: “don’t know nondeterminism”
(if one doesn’t give a solution, others may)
Top-Down Proof - choice failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter

Top-Down Proof - choice failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter

Top-Down Proof - choice failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter

yes ← macdonalds ∧ cocacola ∧ shelter

Top-Down Proof - choice failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter

yes ← macdonalds ∧ cocacola ∧ shelter

yes ← macdonalds ∧ cocacola ∧ house
Top-Down Proof - choice failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
cocacola.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola ∧ shelter
yes ← macdonalds ∧ cocacola ∧ house
yes ← macdonalds ∧ cocacola

If we get a failure with one choice, another may work (and does in this case)

Top-Down Proof - selection failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
cocacola.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola ∧ shelter
yes ← macdonalds ∧ cocacola ∧ house
yes ← macdonalds ∧ cocacola

Top-Down Proof - selection failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
cocacola.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola

Top-Down Proof - selection failure

start from query and work backwards

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
cocacola.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola ∧ shelter
yes ← macdonalds ∧ cocacola ∧ house
yes ← macdonalds ∧ cocacola

If we get a failure with one choice, another may work (and does in this case)
Top-Down Proof - selection failure

<table>
<thead>
<tr>
<th>Line</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>start from query and work backwards</td>
<td>start from query and work backwards</td>
</tr>
<tr>
<td>health ← sustenance ∧ shelter.</td>
<td>health ← sustenance ∧ shelter.</td>
</tr>
<tr>
<td>sustenance ← macdonalds ∧ cocacola.</td>
<td>sustenance ← macdonalds ∧ cocacola.</td>
</tr>
<tr>
<td>sustenance ← bread ∧ water.</td>
<td>sustenance ← bread ∧ water.</td>
</tr>
<tr>
<td>shelter ← house.</td>
<td>shelter ← house.</td>
</tr>
<tr>
<td>house.</td>
<td>house.</td>
</tr>
<tr>
<td>cocacola.</td>
<td>cocacola.</td>
</tr>
<tr>
<td>bread.</td>
<td>bread.</td>
</tr>
<tr>
<td>yes ← health.</td>
<td>yes ← health.</td>
</tr>
<tr>
<td>yes ← sustenance ∧ shelter</td>
<td>yes ← sustenance ∧ shelter</td>
</tr>
<tr>
<td>yes ← bread ∧ water ∧ shelter</td>
<td>yes ← bread ∧ water ∧ shelter</td>
</tr>
<tr>
<td>yes ← bread ∧ water ∧ house</td>
<td>yes ← bread ∧ water ∧ house</td>
</tr>
<tr>
<td>yes ← bread ∧ water</td>
<td>yes ← bread ∧ water</td>
</tr>
<tr>
<td>yes ← water</td>
<td>yes ← water</td>
</tr>
</tbody>
</table>

Again - failure with this choice, try the other
(have to try them all)

start again part way through

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
cocacola.
bread.
yes ← health.
yes ← sustenance ∧ shelter
yes ← bread ∧ water ∧ shelter
yes ← bread ∧ water ∧ house
yes ← bread ∧ water
yes ← water

start again part way through

health ← sustenance ∧ shelter.
sustenance ← macdonalds ∧ cocacola.
sustenance ← bread ∧ water.
shelter ← house.
house.
cocacola.
bread.
yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola ∧ shelter
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?

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

\[
\{(p_1, p_2), \{p_3, p_4, p_5\}, \{p_6, p_7, \ldots\}, \{p_{n-1}, p_n\}\}
\]
## Conjunctive Normal Form - Example 1

### Refutation of Modus Ponens

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

show a contradiction

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

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

## Conjunctive Normal Form - Example 2

### Transitivity of Implication

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

try to show a contradiction

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

1. \( (\neg A \lor B) \land (\neg B \lor C) \land \neg (\neg A \lor \neg C) \)
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. \( \{\neg A, B\}, \{-B, C\}, \{A\}, \{-C\} \)
5. \( \{\neg A, B\}, \{-B, C\}, \{-A, C\}, \{A\}, \{-C\} \)
6. \( \{\neg A, B\}, \{-B, C\}, \{-A, C\}, \{A\}, \{-C\} \)
7. \( \bot \)

## Resolution

### Resolution - Example 1

### Refutation of Modus Ponens

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

show a contradiction

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

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

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

1. \( (\neg A \lor B) \land (\neg B \lor C) \land \neg (\neg A \lor \neg C) \)
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. \( \{\neg A, B\}, \{-B, C\}, \{A\}, \{-C\} \)
5. \( \{\neg A, B\}, \{-B, C\}, \{-A, C\}, \{A\}, \{-C\} \)
6. \( \{\neg A, B\}, \{-B, C\}, \{-A, C\}, \{A\}, \{-C\} \)
7. \( \bot \)

- 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
- Proof by *resolution refutation*: deny the conclusions and show a resolution to \( \bot \).
- 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 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.

\[
P: P \rightarrow (\neg H \rightarrow C)
\]
\[
D : P \rightarrow C \text{  test (refutation of D): } P \land P \land \neg D
\]
\[
(P \rightarrow (\neg H \rightarrow C)) \land (P \rightarrow \neg H) \land \neg(P \rightarrow C)
\]
\[
(\neg P \lor (\neg H \lor C)) \land (\neg P \lor \neg H) \land (\neg P \lor C)
\]
\[
(\neg P \lor \neg H \lor C) \land (\neg P \lor \neg H) \land (\neg P \land \neg C)
\]
\[
(\neg P \lor H \lor C) \land (\neg P \lor \neg H) \land (P \lor \neg C)
\]
\[
\{\{\neg P, H, C\}, \{\neg P, \neg H\}, \{P\}, \{\neg C\}\}
\]
\[
\{\{\neg P, H, C\}, \{\neg P, \neg H\}, \{P\}, \{C\}, \{\neg C\}\}
\]

### Constraint Satisfaction as CNF

- A CSP variable \( Y \) with domain \( \{y_1, \ldots, y_k\} \) can be converted into \( k \) Boolean variables \( \{Y_1, \ldots, Y_k\} \) where \( Y_i \) is true when \( Y \) has value \( y_i \) and false otherwise.
- Thus, \( k \) atoms \( y_1, \ldots, y_k \) are used to represent the CSP variable
- Constraints:
  - \( y_i \) and \( y_j \) cannot both be true when \( i \neq j \): \( \neg y_i \land \neg y_j \) 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, which specifies which assignments are not allowed.
  - Thus, if there are two variables \( Y \) and \( Z \), and a constraint \( Y = Z \), then we have clauses \( \neg y_i \land \neg z_i \) for all \( i \) (Assuming \( Y \) and \( Z \) have the same domains).

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

**Constraints:**
\[
(A \neq 2) \land (B \neq 1) \land (A < B)
\]

Write down the CNF for this CSP

### Where propositions stop...

- Peter is smart \( \land \) Peter is a student \( \rightarrow \) Peter will pass
- Ann is smart \( \land \) Ann is a student \( \rightarrow \) Ann will pass
- Lou is smart \( \land \) Lou is a student \( \rightarrow \) Lou will pass

\[
\text{Peter is smart and Peter is a student} \rightarrow \text{Peter will pass}
\]

\[
\text{Ann is smart and Ann is a student} \rightarrow \text{Ann will pass}
\]

\[
\text{Lou is smart and Lou is a student} \rightarrow \text{Lou will pass}
\]

Where propositions stop...
peter is smart ∧ peter is a student → peter will pass
ann is smart ∧ ann is a student → ann will pass
lou is smart ∧ lou is a student → lou will pass

\[
\text{ann is smart and ann is a student} \quad \text{ann will pass}
\]

\[
\text{All smart students pass} \quad \text{ann is smart and ann is a student} \quad \text{ann will pass}
\]

\[
\forall X (\text{smart}(X) \land \text{student}(X) \rightarrow \text{will_pass}(X))
\]

\[
\text{peter is smart} \land \text{peter is a student} \rightarrow \text{peter will pass}
\]

\[
\text{ann is smart} \land \text{ann is a student} \rightarrow \text{ann will pass}
\]

\[
\text{lou is smart} \land \text{lou is a student} \rightarrow \text{lou will pass}
\]

\[
\text{All smart students pass} \quad \text{ann is smart and ann is a student} \quad \text{ann will pass}
\]

\[
\forall X (\text{smart}(X) \land \text{student}(X) \rightarrow \text{will_pass}(X))
\]

\[
\text{smart(ann)} \land \text{student(ann)} \rightarrow \text{will_pass(ann)}
\]
**Beyond propositions: Individuals and Relations**

**MIU Puzzle**

- **Symbols:** $M, I, U$
- **Axiom:** $MI$
- **Rules:**
  - if $xI$ is a theorem, so is $xIU$
  - $Mx$ 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)