Lecture 5 - Propositions and Inference

Jesse Hoey
School of Computer Science
University of Waterloo

February 6, 2018

Readings: Poole & Mackworth 2nd ed. chapter 5.1-5.3, and 13.1-13.2
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
Two methods for solving problems:

- **Procedural**
  - “how to” knowledge
  - programs
  - meaning of symbols is meaning of computation
  - languages: C, C++, Java ...

- **Declarative**
  - descriptive knowledge
  - databases
  - meaning of symbols is meaning in world
  - languages: propositional logic, Prolog, relational databases, ...
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

and (conjunction) \( \land \)

or (disjunction) \( \lor \)

not (negation) \( \neg \)

if . . . then . . . (implication) \( \rightarrow \)

. . . if and only if . . . \( \iff \)
### Implication Truth Table

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) If it rains, (B) 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) \hspace{1cm} (B)

If it rains, then I will carry an umbrella
If you don’t study, then you will fail
## Implication Truth Table

<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>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

(A) no rain or I will carry an umbrella  
(B) study or you will fail
If and only if Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$A \leftrightarrow B$</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>F</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 is not a **Tautology**
If it’s raining then the grass is wet
the grass is wet
therefore its raining
Logical Consequence

- \( \{X\} \) is a set of statements
- A set of truth assignments to \( \{X\} \) is an interpretation
- A model of \( \{X\} \) is an interpretation that makes \( \{X\} \) true.
- We say that the world in which these truth assignments hold is a model (a verifiable example) of \( \{X\} \).
- \( \{X\} \) is inconsistent if it has no model
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 $\rightarrow$ 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

\[ \begin{align*}
P1 & : P \rightarrow (\neg H \rightarrow C) \quad P2 : P \rightarrow \neg H \\
D & : P \rightarrow C
\end{align*} \]

<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>T</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>T</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>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>F</td>
<td>T</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>
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

\[
\begin{align*}
P1 : P & \rightarrow (\neg H \rightarrow C) \\
P2 : P & \rightarrow \neg H \\
D : P & \rightarrow C
\end{align*}
\]
Arguments and Models

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 $P_2$: $P_1, P_2 \models D$. 

<table>
<thead>
<tr>
<th>$P$</th>
<th>$C$</th>
<th>$H$</th>
<th>$\neg H \rightarrow C$</th>
<th>$P_1$</th>
<th>$P_2$</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>
Logical Consequence

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

Is this argument valid?
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>¬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.
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)
  \[\neg A \land \ldots \land \ldots \rightarrow \bot \rightarrow A\]
A 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.

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

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}. \\
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}. \\
\text{clouds} & \leftarrow \text{near\_sea} \land \text{cyclone}. \\
\text{wind} & \leftarrow \text{cyclone}. \\
\text{near\_sea}. \\
\text{cyclone}.
\end{align*}
\]
also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}. \\
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}. \\
\text{clouds} & \leftarrow \text{near\_sea} \land \text{cyclone}. \\
\text{wind} & \leftarrow \text{cyclone}. \\
\text{near\_sea}. & \\
\text{cyclone}. & \\
\{\text{near\_sea, cyclone}\} &
\end{align*}
\]
also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\text{rain } \leftarrow \text{clouds } \land \text{wind}.
\]

\[
\text{clouds } \leftarrow \text{humid } \land \text{cyclone}.
\]

\[
\text{clouds } \leftarrow \text{near\_sea } \land \text{cyclone}.
\]

\[
\text{wind } \leftarrow \text{cyclone}.
\]

\[
\text{near\_sea}.
\]

\[
\text{cyclone}.
\]

\[
\{\text{near\_sea, cyclone} \}
\]

\[
\{\text{near\_sea, cyclone, wind} \}
\]
also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}. \\
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}. \\
\text{clouds} & \leftarrow \text{near}_\text{sea} \land \text{cyclone}. \\
\text{wind} & \leftarrow \text{cyclone}. \\
\text{near}_\text{sea}. \\
\text{cyclone}. \\
\{\text{near}_\text{sea},\text{cyclone}\} \\
\{\text{near}_\text{sea},\text{cyclone},\text{wind}\} \\
\{\text{near}_\text{sea},\text{cyclone},\text{wind},\text{clouds}\}
\end{align*}
\]
also known as **forward chaining** - start from facts and use rules to generate all possible atoms

\[
\begin{align*}
\text{rain} & \leftarrow \text{clouds} \land \text{wind}. \\
\text{clouds} & \leftarrow \text{humid} \land \text{cyclone}. \\
\text{clouds} & \leftarrow \text{near}_\text{sea} \land \text{cyclone}. \\
\text{wind} & \leftarrow \text{cyclone}. \\
\text{near}_\text{sea}. \\
\text{cyclone}. \\
\{\text{near}_\text{sea},\text{cyclone}\} \\
\{\text{near}_\text{sea},\text{cyclone},\text{wind}\} \\
\{\text{near}_\text{sea},\text{cyclone},\text{wind},\text{clouds}\} \\
\{\text{near}_\text{sea},\text{cyclone},\text{wind},\text{clouds},\text{rain}\}
\end{align*}
\]
Bottom-up proof (no variables)

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

Sound and Complete
Top-Down Proof (no variables)

start from query and work backwards

\[\text{rain} \leftarrow \text{clouds} \land \text{wind}.\]
\[\text{clouds} \leftarrow \text{humid} \land \text{cyclone}.\]
\[\text{clouds} \leftarrow \text{near}_\text{sea} \land \text{cyclone}.\]
\[\text{wind} \leftarrow \text{cyclone}.\]
\[\text{near}_\text{sea}.\]
\[\text{cyclone}.\]
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.
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
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
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.
es ← clouds ∧ wind
yes ← near_sea ∧ cyclone ∧ wind
yes ← near_sea ∧ cyclone ∧ cyclone ∧ cyclone
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
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
Top-Down Proof (no variables)

Start from query and work backwards

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

\[ \text{yes} \leftarrow \text{rain}. \]
\[ \text{yes} \leftarrow \text{clouds} \land \text{wind} \]
\[ \text{yes} \leftarrow \text{near\_sea} \land \text{cyclone} \land \text{wind} \]
\[ \text{yes} \leftarrow \text{near\_sea} \land \text{cyclone} \land \text{cyclone} \]
\[ \text{yes} \leftarrow \text{near\_sea} \land \text{cyclone} \]
\[ \text{yes} \leftarrow \text{cyclone} \]
\[ \text{yes} \leftarrow \]
solve($q_1 \land \ldots \land q_k$):

\[ ac := \text{"yes} \leftarrow q_1 \land \ldots \land q_k \text{"

repeat

select a conjunct $a_i$ from body of $ac$

choose a clause $C$ from KB with $a_i$ as head

replace $a_i$ 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.
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

\[
\text{health} \leftarrow \text{sustenance} \land \text{shelter}.
\]

\[
\text{sustenance} \leftarrow \text{macdonalds} \land \text{cocacola}.
\]

\[
\text{sustenance} \leftarrow \text{bread} \land \text{water}.
\]

\[
\text{shelter} \leftarrow \text{house}.
\]

\[
\text{house}.
\]

\[
\text{water}.
\]

\[
\text{bread}.
\]

\[
\text{yes} \leftarrow \text{health}.
\]

\[
\text{yes} \leftarrow \text{sustenance} \land \text{shelter}
\]
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
start from query and work backwards

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

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

start from query and work backwards

health $\leftarrow$ sustenance $\land$ shelter.
sustenance $\leftarrow$ macdonalds $\land$ cocacola.
sustenance $\leftarrow$ bread $\land$ water.
shelter $\leftarrow$ house.

house.
water.
bread.

yes $\leftarrow$ health.
yes $\leftarrow$ sustenance $\land$ shelter
yes $\leftarrow$ bread $\land$ water $\land$ shelter
yes $\leftarrow$ bread $\land$ water $\land$ house
yes $\leftarrow$ bread $\land$ water
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
yes ←
Top-Down Interpreter (no variables)

\[ \text{solve}(q_1 \land \ldots \land q_k): \]

\[ ac := \text{"yes} \leftarrow q_1 \land \ldots \land q'_k \]

\[ \text{repeat} \]

\[ \text{select} \text{ a conjunct } a_i \text{ from body of } ac \]

\[ \text{choose} \text{ a clause } C \text{ from KB with } a_i \text{ as head} \]

\[ \text{replace } a_i \text{ in body of } ac \text{ by body of } C \]

\[ \text{until } ac \text{ is an answer} \]

**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.
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.
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 ← bread ∧ water ∧ shelter
start from query and work backwards

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

house.
water.
bread.

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonallds ∧ cocacola ∧ shelter
Top-Down Proof - choice failure

start from query and work backwards

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

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola ∧ shelter
yes ← macdonalds ∧ cocacola ∧ house
yes ← macdonalds ∧ cocacola
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
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.
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.
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
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 ← bread ∧ water ∧ shelter
Top-Down Proof - selection failure

start from query and work backwards

\[ \text{health} \leftarrow \text{sustenance} \land \text{shelter}. \]
\[ \text{sustenance} \leftarrow \text{macdonalds} \land \text{coca cola}. \]
\[ \text{sustenance} \leftarrow \text{bread} \land \text{water}. \]
\[ \text{shelter} \leftarrow \text{house}. \]
\[ \text{house}. \]
\[ \text{coca cola}. \]
\[ \text{bread}. \]

\[ \text{yes} \leftarrow \text{health}. \]
\[ \text{yes} \leftarrow \text{sustenance} \land \text{shelter} \]
\[ \text{yes} \leftarrow \text{bread} \land \text{water} \land \text{shelter} \]
\[ \text{yes} \leftarrow \text{bread} \land \text{water} \land \text{house} \]
Top-Down Proof - selection failure

start from query and work backwards

health ← sustenance $\land$ shelter.
sustenance ← macdonalds $\land$ cocacola.
sustenance ← bread $\land$ water.
shelter ← house.
house.
ocacola.
bread.

yes ← health.
yes ← sustenance $\land$ shelter
yes ← bread $\land$ water $\land$ shelter
yes ← bread $\land$ water $\land$ house
yes ← bread $\land$ water
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 ← bread ∧ water ∧ shelter
yes ← bread ∧ water ∧ house
yes ← bread ∧ water
yes ← water
start from query and work backwards

\begin{align*}
\text{health} & \leftarrow \text{sustenance} \land \text{shelter}, \\
\text{sustenance} & \leftarrow \text{macdonalds} \land \text{coca cola}. \\
\text{sustenance} & \leftarrow \text{bread} \land \text{water}. \\
\text{shelter} & \leftarrow \text{house}. \\
\text{house} & . \\
\text{coca cola} & . \\
\text{bread} & . \\
\text{yes} & \leftarrow \text{health}. \\
\text{yes} & \leftarrow \text{sustenance} \land \text{shelter} \\
\text{yes} & \leftarrow \text{bread} \land \text{water} \land \text{shelter} \\
\text{yes} & \leftarrow \text{bread} \land \text{water} \land \text{house} \\
\text{yes} & \leftarrow \text{bread} \land \text{water} \\
\text{yes} & \leftarrow \text{water} \\
\text{Again - failure with this} & \text{choice}, \text{try the other} \\
\text{(have to try them all)} & .
\end{align*}
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
start again part way through

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

yes ← health.
yes ← sustenance ∧ shelter
yes ← macdonalds ∧ cocacola ∧ shelter
yes ← macdonalds ∧ cocacola ∧ house
Top-Down Proof - selection failure

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

start again part way through

health $\leftarrow$ sustenance $\land$ shelter.
sustenance $\leftarrow$ macdonalds $\land$ cocacola.
sustenance $\leftarrow$ bread $\land$ water.
shelter $\leftarrow$ house.
house.
cocacola.
bread.

yes $\leftarrow$ health.

yes $\leftarrow$ sustenance $\land$ shelter

yes $\leftarrow$ macdonalds $\land$ cocacola $\land$ shelter

yes $\leftarrow$ macdonalds $\land$ cocacola $\land$ house

yes $\leftarrow$ macdonalds $\land$ cocacola

yes $\leftarrow$ macdonalds
Top-Down Proof - selection failure

start again part way through

health ← sustenance ∧ shelter.
s sustenance ← macdonalds ∧ cocacola.
s sustenance ← bread ∧ water.
s shelter ← house.
s house.
s cocacola.
s bread.
s yes ← health.
s yes ← sustenance ∧ shelter
s yes ← macdonalds ∧ cocacola ∧ shelter
s yes ← macdonalds ∧ cocacola ∧ house
s yes ← macdonalds ∧ cocacola
s yes ← macdonalds
Failure with this selection, so we’re done
(since one conjunct is going to fail, the whole thing fails)
solve\( (q_1 \land \ldots \land q_k) \):
\[
    ac := \text{"yes} \leftarrow q_1 \land \ldots \land q_k
\]
repeat
    select a conjunct \(a_i\) from body of \(ac\)
    choose a clause \(C\) from KB with \(a_i\) as head
    replace \(a_i\) in body of \(ac\) by body of \(C\)
until \(ac\) is an answer

**select:** “don’t care nondeterminism”
    any one will do, but be careful: some selections will lead more quickly to solutions!

**choose:** “don’t know nondeterminism”
    have to do them all: can determine the complexity of the problem

Read more about negation as failure in Section 5.6 (not course material)
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?
A well-formed formula is in *conjunctive normal form* (CNF) if it is a conjunction of disjunctions of atoms.

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

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

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

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

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

1. \[ A \land (\neg A \lor B) \land (\neg B) \]

2. \{\{A\}, \{\neg A, B\}, \{\neg B\}\}
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. \((\neg A \lor B) \land (\neg B \lor C) \land (\neg A \land \neg C)\)
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.
Resolution

- 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
Refutation of Modus Ponens

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

show a contradiction

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

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. \[ \bot \]
Transitivity of Implication (again)

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

try to show a contradiction

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

1. \((-A \lor B) \land (-B \lor C) \land \neg(-A \lor C)\)
2. \((-A \lor B) \land (-B \lor C) \land (-A \land \neg C)\)
3. \((-A \lor B) \land (-B \lor C) \land A \land \neg C\)
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

\[ P_1 : P \rightarrow (\neg H \rightarrow C) \quad P_2 : P \rightarrow \neg H \]
\[ D : P \rightarrow C \quad \text{test (refutation of } D) : P_1 \land P_2 \land \neg D \]

\[
(P \rightarrow (\neg H \rightarrow C)) \land (P \rightarrow \neg H) \land \neg (P \rightarrow C) \\
(\neg P \lor (\neg H \rightarrow C)) \land (\neg P \lor \neg H) \land \neg (\neg P \lor C) \\
(\neg P \lor (\neg \neg H \lor C)) \land (\neg P \lor \neg H) \land (\neg \neg P \land \neg C) \\
(\neg P \lor H \lor C) \land (\neg P \lor \neg H) \land (P) \land (\neg C) \\
\left\{\left\{\neg P, H, C\right\}, \left\{\neg P, \neg H\right\}, \{P\}, \{\neg C\}\right\} \\
\left\{\left\{\neg P, H, C\right\}, \left\{\neg P, \neg H\right\}, \left\{\neg P, C\right\}, \{P\}, \{\neg C\}\right\} \\
\left\{\left\{\neg P, H, C\right\}, \left\{\neg P, \neg H\right\}, \left\{\neg P, C\right\}, \{P\}, \{C\}, \{\neg C\}\right\} \\
\bot
\]
Many problems can be formulated as a CNF

- Satisfiability
- Logic circuits
- Gene decoding
- Scheduling
- Air traffic control
- ...
Constraint Satisfaction as CNF

- 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:
  - $y_i$ 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, which 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 $\neg y_i \land \neg z_i$ for all $i$ (Assuming $Y$ and $Z$ have the same domains).
Example Delivery robot: activities $a,b$, times $1,2,3,4$.

Constraints:

$\neg A = 2 \land \neg B = 1 \land A < B$

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
Where propositions stop...

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

\underline{\text{peter is smart and peter is a student}}

\underline{\text{peter will pass}}
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

Ann is smart and Ann is a student $\land$ Ann will pass
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

Lou is smart and Lou is a student

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

All smart students pass

Ann is smart and Ann is a student

Ann will pass
Where propositions stop...

peter is smart \land\ p\ \text{is a student} \rightarrow p\ \text{will pass}
ann is smart \land\ a\ \text{is a student} \rightarrow a\ \text{will pass}
lou is smart \land\ l\ \text{is a student} \rightarrow l\ \text{will pass}

\textbf{All smart students pass}

\text{ann is smart and ann is a student} \rightarrow \text{ann will pass}

\text{smart}(\text{ann}) \land\ \text{student}(\text{ann}) \rightarrow \text{will\_pass}(\text{ann})
Where propositions stop...

\[
\begin{align*}
\text{peter is smart} & \land \text{peter is a student} \quad \rightarrow \quad \text{peter will pass} \\
\text{ann is smart} & \land \text{ann is a student} \quad \rightarrow \quad \text{ann will pass} \\
\text{lou is smart} & \land \text{lou is a student} \quad \rightarrow \quad \text{lou will pass}
\end{align*}
\]

All smart students pass

\[
\begin{align*}
\text{ann is smart and ann is a student} \\
\hline
\text{ann will pass}
\end{align*}
\]

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

**All smart students pass**

Ann is smart and Ann is a student

\[ \begin{align*}
\text{Ann will pass} \\
\forall X (\text{smart}(X) \land \text{student}(X) \rightarrow \text{will\_pass}(X)) \\
\text{smart(ann)} \land \text{student(ann)} \\
\rightarrow \text{will\_pass(ann)}
\end{align*} \]
Beyond propositions: Individuals and Relations

\begin{align*}
\text{in}(\text{kim, r123}). \\
\text{part of} (\text{r123, cs.building}). \\
\text{in}(X, Y) \leftarrow \\
\text{part of}(Z, Y) \land \\
\text{in}(X, Z).
\end{align*}

\begin{align*}
\text{in}(\text{kim, cs.building}).
\end{align*}
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)
Planning under certainty (Poole & Mackworth 2nd ed. Chapter 6.1-6.4)

Supervised Learning (Poole & Mackworth 2nd ed. Chapter 7.1-7.6)