Introduction to Artificial Intelligence

Instructor: Wei-Min Shen

Week 5.1
Status Check

• Projects
  – Project 1, grades
  – Project 2, assignment is out

• Questions?
Today’s Lecture

• Inferences in First Order Logic (FOL)
  – Modus Ponens with variables
    • Unification: finding the values for variables
  – Forward chaining
    • E.g. production systems, ACT, SOAR
  – Backward chaining (aka Logic Programming)
    • E.g., Prolog
  – Resolution
    • Convert to CNF, eliminating P, -P.
Inferences in FOL

• Proofs – extend propositional logic inference to deal with quantifiers

• Unification

• Generalized Modus Ponens

• Forward and backward chaining
  – inference rules and reasoning program

• Completeness
  – Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set

• Resolution
  – inference procedure that is complete for any set of sentences

• Logic programming (the same as backward chaining)
Truth Depends on Interpretation

Representation 1:
A  B
ON(A,B) True
ON(B,A) False

Representation 2:
A  B
ON(A,B) False
ON(B,A) True
Logic as Representation of World

Representation

Sentence

entails

Sentence

“Refers to” (semantics)

World

Facts

follows

Facts

“Refers to” (semantics)
Desired Inference Procedures

representation

sentences

entail

inference

sentence

world

facts

follows - from

fact
FOL Inference Rules (The Seven Basics)

◊ **Modus Ponens or Implication-Elimination:** (From an implication and the premise of the implication, you can infer the conclusion.)

\[
\alpha \Rightarrow \beta, \quad \alpha \\
\beta
\]

◊ **And-Elimination:** (From a conjunction, you can infer any of the conjuncts.)

\[
\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n \\
\alpha_i
\]

◊ **And-Introduction:** (From a list of sentences, you can infer their conjunction.)

\[
\alpha_1, \alpha_2, \ldots, \alpha_n \\
\alpha_1 \land \alpha_2 \land \ldots \land \alpha_n
\]

◊ **Or-Introduction:** (From a sentence, you can infer its disjunction with anything else at all.)

\[
\alpha_i \\
\alpha_1 \lor \alpha_2 \lor \ldots \lor \alpha_n
\]

◊ **Double-Negation Elimination:** (From a doubly negated sentence, you can infer a positive sentence.)

\[
\neg \neg \alpha \\
\alpha
\]

◊ **Unit Resolution:** (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

\[
\alpha \lor \beta, \quad \neg \beta \\
\alpha
\]

◊ **Resolution:** (This is the most difficult. Because \( \beta \) cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

\[
\alpha \lor \beta, \quad \neg \beta \lor \gamma \\
\alpha \lor \gamma \
\]

or equivalently

\[
\neg \alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma \\
\neg \alpha \Rightarrow \gamma
\]
Reminder

- **Ground term**: A term that does not contain a variable.
  - A constant symbol
  - A function applies to some ground term

- `{x/a}`: substitution/binding list

- Examples:
  - Married(x, y) with {x/Mary, y/John}
  - Results in Married(Mary, John)
Three New Rules

The three new inference rules for FOL (compared to propositional logic) are:

• **Universal Elimination (UE):**
  for any sentence $\beta$, variable $x$ and ground term $O$,
  
  \[
  \forall x \beta \\
  \beta \{x/O\}
  \]
  
  e.g., from $\forall x \text{Likes}(x, \text{Candy})$ and $\{x/\text{Joe}\}$ we can infer $\text{Likes}(\text{Joe}, \text{Candy})$

• **Existential Elimination (EE):**
  for any sentence $\beta$, variable $x$ and constant symbol $k$ not in KB,
  
  \[
  \exists x \beta \\
  \beta \{x/k\}
  \]
  
  e.g., from $\exists x \text{Kill}(x, \text{Victim})$ we can infer $\text{Kill}(\text{Murderer}, \text{Victim})$, if Murderer new symbol

• **Existential Introduction (EI):**
  for any sentence $\beta$, variable $x$ not in $\beta$ and ground term $g$ in $\beta$,
  
  \[
  \beta \\
  \exists x \beta \{g/x\}
  \]
  
  e.g., from $\text{Likes}(\text{Joe}, \text{Candy})$ we can infer $\exists x \text{Likes}(x, \text{Candy})$
Examples of Inferences/Proofs

Sound inference: find $\alpha$ such that $KB \models \alpha$.
Proof process is a search, operators are inference rules.

E.g., Modus Ponens (MP)

\[
\frac{\alpha, \quad \alpha \Rightarrow \beta}{\beta} \quad \frac{At(Joe, UCB)}{OK(Joe)} \quad \frac{At(Joe, UCB) \Rightarrow OK(Joe)}{OK(Joe)}
\]

E.g., And-Introduction (AI)

\[
\frac{\alpha \quad \beta \quad OK(Joe) \quad CSMajor(Joe)}{\alpha \land \beta \quad OK(Joe) \land CSMajor(Joe)}
\]

E.g., Universal Elimination (UE)

\[
\frac{\forall x \quad \alpha \quad \forall x \quad At(x, UCB) \Rightarrow OK(x)}{\alpha\{x/\tau\} \quad At(Pat, UCB) \Rightarrow OK(Pat)}
\]

$\tau$ must be a ground term (i.e., no variables)
## Example Proof

| Bob is a buffalo | 1. $Buffalo(Bob)$  
| Pat is a pig     | 2. $Pig(Pat)$     
| Buffaloes outrun pigs | 3. $\forall x, y \; Buffalo(x) \land Pig(y) \Rightarrow Faster(x, y)$ 
| Bob outruns Pat  |
| AI 1 & 2 | 4. $Buffalo(Bob) \land Pig(Pat)$ |
Example Proof

UE 3, \{x/Bob, y/Pat\} 5. \textit{Buffalo}(Bob) \land \textit{Pig}(Pat) \Rightarrow \textit{Faster}(Bob, Pat)
Example Proof

| MP 4 & 5 | 6. $Faster(Bob, Pat)$ |
Search with Primitive Example Rules

Operators are inference rules
States are sets of sentences
Goal test checks state to see if it contains query sentence

\[
\begin{array}{cccc}
1 & 2 & 3 \\
\Rightarrow & AI & 1 & 2 \\
& 1 & 2 & 3 & 4 \\
& \Rightarrow & UE & 3 \{x/\text{Bob}, y/\text{Pat}\} \\
& 1 & 2 & 3 & 4 & 5 \\
& \Rightarrow & MP & 5 & 6 \\
& 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

AI, UE, MP is a common inference pattern

**Problem:** branching factor huge, esp. for UE

**Idea:** find a substitution that makes the rule premise match some known facts
\[ \Rightarrow \text{a single, more powerful inference rule} \]
More Inference in FOL

• Proofs – extend propositional logic inference to deal with quantifiers

• Unification

• Generalized modus ponens

• Forward and backward chaining – inference rules and reasoning program

• Completeness – Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set

• Resolution – inference procedure that is complete for any set of sentences

• Logic programming
Unification

A substitution $\sigma$ unifies atomic sentences $p$ and $q$ if $p\sigma = q\sigma$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{OJ})$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td></td>
</tr>
</tbody>
</table>

Goal of unification: finding the substitution $\sigma$:

$\{x/\text{Jane}\}$
$\{x/\text{OJ}, y/\text{John}\}$
$\{y/\text{John}, x/\text{Mother}(\text{John})\}$
Unification

**Idea:** Unify rule premises with known facts, apply unifier to conclusion

E.g., if we know \( q \) and we have \( \text{Knows}(John, x) \Rightarrow \text{Likes}(John, x) \)
then we conclude

\[
\text{Likes}(John, Jane) \\
\text{Likes}(John, OJ) \\
\text{Likes}(John, \text{Mother}(John))
\]
Extra Example for Unification

How to **unify P and Q** below?

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student(x)</td>
<td>Student(Bob)</td>
<td>{x/Bob}</td>
</tr>
<tr>
<td>Sells(Bob, x)</td>
<td>Sells(x, coke)</td>
<td>{x/coke, x/Bob}</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Is this correct?</strong></td>
</tr>
</tbody>
</table>

What about \{ x/VendingMachine \} ?
## Extra Example for Unification

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student($x$)</td>
<td>Student(Bob)</td>
<td>${x/\text{Bob}}$</td>
</tr>
<tr>
<td>Sells(Bob, $x$)</td>
<td>Sells($y$, coke)</td>
<td>${x/\text{coke}, y/\text{Bob}}$</td>
</tr>
</tbody>
</table>
More Unification Examples

1 – unify(P(a,X), P(a,b)) \( \sigma = \{ X/b \} \)
2 – unify(P(a,X), P(Y,b)) \( \sigma = \{ Y/a, X/b \} \)
3 – unify(P(a,X), P(Y,f(a))) \( \sigma = \{ Y/a, X/f(a) \} \)
4 – unify(P(a,X), P(X,b)) \( \sigma = \text{failure} \)

Note: If P(a,X) and P(X,b) are independent, then we can replace X with Y and get the unification to work.
More Inference in FOL

• Proofs – extend propositional logic inference to deal with quantifiers

• **Unification**
• Generalized *modus ponens*
• **Forward and backward** chaining – inference rules and reasoning program
• **Completeness** – Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set
• **Resolution** – inference procedure that is complete for any set of sentences
• Logic programming
Generalized Modus Ponens (GMP)

\[
p_1', p_2', \ldots, p_n', \quad (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \quad \frac{}{q\sigma}
\]

where \( p_i'\sigma = p_i\sigma \) for all \( i \)

E.g. \( p_1' = \text{Faster(Bob,Pat)} \)
\( p_2' = \text{Faster(Pat,Steve)} \)
\( p_1 \land p_2 \Rightarrow q = \text{Faster}(x, y) \land \text{Faster}(y, z) \Rightarrow \text{Faster}(x, z) \)
\( \sigma = \{x/\text{Bob}, y/\text{Pat}, z/\text{Steve}\} \)
\( q\sigma = \text{Faster}(\text{Bob, Steve}) \)

GMP used with KB of definite clauses (exactly one positive literal):
either a single atomic sentence or
(conjunction of atomic sentences) \( \Rightarrow \) (atomic sentence)
All variables assumed universally quantified
Soundness of GMP

Need to show that

\[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \models q\sigma \]

provided that \( p_i'\sigma = p_i\sigma \) for all \( i \)

Lemma: For any definite clause \( p \), we have \( p \models p\sigma \) by UE

1. \( (p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\sigma = (p_1\sigma \land \ldots \land p_n\sigma \Rightarrow q\sigma) \)

2. \( p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1'\sigma \land \ldots \land p_n'\sigma \)

3. From 1 and 2, \( q\sigma \) follows by simple MP
Properties of GMP

• Why is GMP and efficient inference rule?

  - It takes **bigger steps**, combining several small inferences into one

  - It takes **sensible steps**: uses eliminations that are guaranteed to help (rather than random UEs)

  - It uses a precompilation step which converts the KB to **canonical form** (Horn sentences)

Remember: sentence in Horn from is a conjunction of Horn clauses (clauses with at most one positive literal).
Horn form

• We convert sentences to Horn form as they are entered into the KB
• Using Existential Elimination and And Elimination

• e.g., \( \exists x \) \( \text{Owns} (\text{Nono}, x) \land \text{Missile}(x) \) becomes

\[
\text{Owns} (\text{Nono}, M), \\
\text{Missile}(M)
\]

(with \( M \) a new symbol that was not already in the KB)
More Inference in FOL

• Proofs – extend propositional logic inference to deal with quantifiers

• Unification

• Generalized modus ponens

• Forward and backward chaining
  – inference rules and reasoning program

• Completeness – Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set

• Resolution – inference procedure that is complete for any set of sentences

• Logic programming
Forward Chaining

• Forward Chaining acts like a breadth-first search at the top level, with depth-first sub-searches.

• Since the search space spans the entire KB, a large KB must be organized in an intelligent manner in order to enable efficient searches in reasonable time.

When a new fact $p$ is added to the KB
   for each rule such that $p$ unifies with a premise
       if the other premises are known
       then add the conclusion to the KB and continue chaining

Forward chaining is data-driven
   e.g., inferring properties and categories from percepts
Forward Chaining Example

Add facts 1, 2, 3, 4, 5, 7 in turn.
Number in [] = unification literal; √ indicates rule firing

1. \( \text{Buffalo}(x) \land \text{Pig}(y) \Rightarrow \text{Faster}(x, y) \)
2. \( \text{Pig}(y) \land \text{Slug}(z) \Rightarrow \text{Faster}(y, z) \)
3. \( \text{Faster}(x, y) \land \text{Faster}(y, z) \Rightarrow \text{Faster}(x, z) \)
4. \( \text{Buffalo}(\text{Bob}) \) [1a, ×]
5. \( \text{Pig}(\text{Pat}) \) [1b, √] \(\Rightarrow\) 6. \( \text{Faster}(\text{Bob}, \text{Pat}) \) [3a, ×], [3b, ×]
   [2a, ×]
7. \( \text{Slug}(\text{Steve}) \) [2b, √]
   \(\Rightarrow\) 8. \( \text{Faster}(\text{Pat}, \text{Steve}) \) [3a, ×], [3b, √]
   \(\Rightarrow\) 9. \( \text{Faster}(\text{Bob}, \text{Steve}) \) [3a, ×], [3b, ×]
Backward Chaining

When a query $q$ is asked
  if a matching fact $q'$ is known, return the unifier
  for each rule whose consequent $q'$ matches $q$
    attempt to prove each premise of the rule by backward chaining

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

Backward chaining is the basis for logic programming, e.g., Prolog
Backward Chaining Example

1. \( \text{Pig}(y) \land \text{Slug}(z) \Rightarrow \text{Faster}(y, z) \)
2. \( \text{Slimy}(z) \land \text{Creeps}(z) \Rightarrow \text{Slug}(z) \)
3. \( \text{Pig}(\text{Pat}) \)
4. \( \text{Slimy}(\text{Steve}) \)
5. \( \text{Creeps}(\text{Steve}) \)

To be proved →

- \( \text{Faster}(\text{Pat}, \text{Steve}) \)
  - 1. \( \{y/\text{Pat}, z/\text{Steve}\} \)
  - 3. \( \{\} \)
  - 2. \( \{z/\text{Steve}\} \)
  - 4. \( \{\} \)
  - 5. \( \{\} \)
More Inference in FOL

• Proofs – extend propositional logic inference to deal with quantifiers

• **Unification**

• Generalized *modus ponens*

• **Forward and backward chaining**
  – inference rules and reasoning program

• **Completeness** (proved by)
  – Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved by inferences from that set

• **Resolution**
  – inference procedure that is complete for any set of sentences

• Logic programming
More Inference in FOL

- Proofs – extend propositional logic inference to deal with quantifiers

- Unification
- Generalized modus ponens
- Forward and backward chaining
  - inference rules and reasoning program

- Completeness
  - Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set

- Resolution
  - inference procedure that is complete for any set of sentences

- Logic programming
Resolution

Entailment in first-order logic is only semidecidable:

- can find a proof of $\alpha$ if $KB \models \alpha$
- cannot always prove that $KB \not\models \alpha$

Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

Resolution is a refutation procedure:

- to prove $KB \models \alpha$, show that $KB \land \neg \alpha$ is unsatisfiable

Resolution uses $KB$, $\neg \alpha$ in CNF (conjunction of clauses)

Resolution inference rule combines two clauses to make a new one:

- KB: \{famous(x)\}

To prove: famous(John)?

Inference continues until an empty clause is derived (contradiction)
Resolution Example

Given: KB in CNF content

\[
\neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x, y, z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) \\
\neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono}, x) \lor \text{Sells}(\text{West}, x, \text{Nono}) \\
\neg \text{Enemy}(x, \text{America}) \lor \text{Hostile}(x) \\
\neg \text{Missile}(x) \lor \text{Weapon}(x) \\
\text{Owns}(\text{Nono}, M_1) \\
\text{American}(\text{West}) \]

Prove: Criminal(\text{West})
Example: Criminal(West)?
More Inference in FOL

• Proofs – extend propositional logic inference to deal with quantifiers

• Unification
• Generalized modus ponens
• Forward and backward chaining
  – inference rules and reasoning program

• Completeness
  – Gödel’s theorem: for FOL, any sentence entailed by another set of sentences can be proved from that set

• Resolution
  – inference procedure that is complete for any set of sentences

• Logic programming
Logic Programming: Prolog

- A useful programming/prototyping paradigm
- A Prolog program is a set of Horn clauses
  head :- literal₁, ... literalₙ.

  criminal(X):-
  american(X),weapon(Y),sells(X,Y,Z),hostile(Z).

- Proof is depth-first, top-down, left-to-right backward chaining
  - With efficient unification and retrieval of matching clauses

- Built-in predicates and syntactic sugar
  - Arithmetic: X is Y*Z+3
  - Lists: (cons 1 (cons 2 (cons 3 '()))) can be written in Prolog as
    .(1,. (2,. (3, [])))
  or [1|[2|[3|[[]]]]]
  or [1,2,3]
Prolog Example

Appending two lists to produce a third (recursively):

```prolog
append([], Y, Y).
// Appending an empty list to a list yields the same list
append([X|L], Y, [X|Z]) :- append(L, Y, Z).
// Appending a list [X|L] onto a list Y yields [X|Z]
// if Z is the result of appending L to Y
```
Prolog Example

- Appending two lists to produce a third:
  
  ```prolog
  append([], Y, Y).
  append([X|L], Y, [X|Z]) :- append(L, Y, Z).
  ```

- Query: `append([1], [2], C) ?`
  
  Unify: `append([1|[]], [2], [1|Z])`
  
  `{X=1, L=[], Y=[2], C=[1|Z]}`

  Subgoal: `append([], [2], Z)`
  
  Unify: `append([], [2], [2])`  
  
  `{Z=[2]}`

  Answer: `C=[1|2]=[1, 2]`
Other Issues in BC/Prolog

- Incomplete due to infinite loops
  
  \[
  \begin{align*}
  \text{path}(X,Z) & :\text{link}(X,Z). \\
  \text{path}(X,Z) & :\text{path}(X,Y), \text{link}(Y,Z).
  \end{align*}
  \]

- Inefficient due to repeated subgoals (both success & failure)
  - Can cache previous results
    - \textit{Memoization} of results of subgoals
      - Analogous to \textit{dynamic programming} that Forward Chaining already does in saving intermediate results as facts in KB
    - Trades off space for time

- Overhead due to interpretation for index lookup, unification and recursive call stack
  - Compile program into something much more efficient

- No \textit{Occurs-Check}, so can yield incorrect answers