Lecture 13

Razvan C. Bunescu
School of Electrical Engineering and Computer Science
bunescu@ohio.edu
Logic Programming

- Introduction
- Predicate Calculus
- Inference Rules
- Logic Programming Languages
- Elements of Prolog
- Applications of Logic Programming
Introduction: Logic Programming Languages

• Express programs in a form of **symbolic logic**:  
  – **First order predicate calculus**:  
    • **Horn clauses**.
  
• Produce results using a logical inference process:  
  – **Resolution**:  
    • **Backward chaining** (top-down resolution).

• Sometimes called **declarative** programming languages:  
  – **declarative**: specify only properties of the results (**what**).  
  – **procedural**: specify procedures for obtaining the results (**how**).
First Order Predicate Calculus

• First Order Logic (FOL) assumes the world contains:
  – Objects: people, houses, numbers, colors, baseball games, wars, …
  – Relations: brother of, bigger than, part of, comes between, …
  – Properties: red, round, prime, multistoried, …
  – Functions: father of, best friend, one more than, plus, …

• Examples:
  – “Evil King John ruled England in 1200.”
    • Objects:
    • Relations:
    • Properties:
Syntax of FOL: Basic Elements

• Used to build terms:
  – Constants: John, 2, A, B, C, ...
  – Functions: sqrt, cosine, fatherOf, leftLegOf, ...
  – Variables: x, y, a, b, s, ...

• Used to build sentences:
  – Predicates: Round, Brother, >, ...
  – Connectives: ¬, ⇒, ∧, ∨, ⇔
  – Equality: =
  – Quantifiers: ∀, ∃
Atomic Sentences

• Every FOL expression is a sentence, which represents a fact.

\[
\text{AtomicSentence} \rightarrow \text{Predicate} \ (\text{Term}_1, \ldots, \text{Term}_n) \\
\quad \mid \text{Term}_1 = \text{Term}_2
\]

\[
\text{Term} \rightarrow \text{Function} \ (\text{Term}_1, \ldots, \text{Term}_n) \\
\quad \mid \text{Constant} \\
\quad \mid \text{Variable}
\]

• Examples:
  – Brother(John, Richard)
  – >(length(leftLegOf(Richard)), length(leftLegOf(John)))
Complex Sentences

- Complex sentences are made from atomic sentences using connectives:
  \( \neg S, S_1 \land S_2, S_1 \lor S_2, S_1 \Rightarrow S_2, S_1 \Leftrightarrow S_2, \)

- Examples:

  \[
  \text{Sibling}(\text{John}, \text{Richard}) \Rightarrow \text{Sibling}(\text{Richard}, \text{John}) \\
  >(1,2) \lor \leq (1,2) \\
  >(1,2) \land \neg >(1,2)
  \]
Truth in FOL

• Sentences are true/false with respect to a model and an interpretation.
  – Model contains objects and relations among them.
  – Interpretation specifies referents for:
    constant symbols → objects
    predicate symbols → relations
    function symbols → functional relations

• An atomic sentence $\text{Predicate}(\text{term}_1,\ldots,\text{term}_n)$ is true iff
the objects referred to by $\text{term}_1,\ldots,\text{term}_n$ are in the relation
referred to by $\text{Predicate}$. 
Model in FOL: Example
Universal Quantification

• $\forall <\text{variables}> <\text{sentence}>$

“Everyone at OU is smart”: $\forall x \text{At}(x, \text{OU}) \Rightarrow \text{Smart}(x)$

• “$\forall x P$” is true in a model $M$ iff $P$ is true for every possible instantiation of $x$ with an object in the model.

• Roughly speaking, equivalent to the conjunction of instantiations of $P$:
  
  $\text{At}(\text{John}, \text{OU}) \Rightarrow \text{Smart}(\text{John})$
  
  $\wedge \text{At}(\text{Richard}, \text{OU}) \Rightarrow \text{Smart}(\text{Richard})$
  
  $\wedge \text{At}(\text{OU}, \text{OU}) \Rightarrow \text{Smart}(\text{OU})$
  
  $\wedge ...$

Lecture 13
Existential Quantification

• \( \exists \) \(<variables>\) \(<sentence>\)

“Someone at OU is smart”: \( \exists x \) At\((x, \text{OU}) \) \(\wedge\) Smart\((x)\)

• “\( \exists x \) \(P\)” is true in a model \( M \) iff \( P \) is true with \( x \) being some possible object in the model.

• Roughly speaking, equivalent to the disjunction of instantiations of \( P \):
  \(\begin{align*}
  \text{At(John, OU)} \wedge \text{Smart(John)} \\
  \lor \text{At(Richard, OU)} \wedge \text{Smart(Richard)} \\
  \lor \text{At(OU, OU)} \wedge \text{Smart(OU)} \\
  \lor \ldots
  \end{align*}\)
Properties of Quantifiers

- \( \forall x \forall y \) is the same as \( \forall y \forall x \)
- \( \exists x \exists y \) is the same as \( \exists y \exists x \)

- \( \exists x \forall y \) is not the same as \( \forall y \exists x \)
  - \( \exists x \forall y \) Loves(x,y)
    - “There is a person who loves everyone in the world”
  - \( \forall y \exists x \) Loves(x,y)
    - “Everyone in the world is loved by at least one person”

- **Quantifier duality**: each can be expressed using the other.
  - \( \forall x \) Likes(x, IceCream) \( \iff \) \( \neg \exists x \neg \) Likes(x, IceCream)
  - \( \exists x \) Likes(x, Broccoli) \( \iff \) \( \neg \forall x \neg \) Likes(x, Broccoli)
Equality

• \( \text{term}_1 = \text{term}_2 \) is true under a given interpretation if and only if \( \text{term}_1 \) and \( \text{term}_2 \) refer to the same object.

• Example: definition of \textit{Sibling} in terms of \textit{Parent}:

\[
\forall x,y \text{Sibling}(x,y) \iff [\neg (x = y) \land \exists m,f \neg (m = f) \land \\
\text{Parent}(m,x) \land \text{Parent}(f,x) \land \text{Parent}(m,y) \land \text{Parent}(f,y)]
\]
Using FOL: The Kinship Domain

- “Brothers are siblings:
  \[ \forall x, y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y) \]

- “One's mother is one's female parent”
  \[ \forall m, c \text{ Mother}(c) = m \Rightarrow (\text{Female}(m) \land \text{Parent}(m, c)) \]

- The “Sibling” relation is symmetric:
  \[ \forall x, y \text{ Sibling}(x, y) \Rightarrow \text{Sibling}(y, x) \]
Inference in Predicate Logic

• **Inference = Reasoning**: any process by which conclusions are reached.

• **Logical Inference** = a procedure that generate a new sentence $\alpha$ that is necessarily true, given that the old sentences in the KB (Knowledge Base) are true.

• **Entailment**: the relation between KB and the new sentence $\alpha$.

  $\text{KB} \models \alpha$
Logical Inference: Example

• Knowledge Base (KB):

\[ \forall x \text{ King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x) \]
\[ \text{King}(\text{John}) \]
\[ \forall y \text{ Greedy}(y) \]
\[ \text{Brother}(\text{Richard}, \text{John}) \]

• Sentence (\( \alpha \)):

\[ \text{Evil}(\text{John}) \]

• Entailment: KB |= \( \alpha \) ?
Inference Rules

• **Resolution** (Herbrand 1965):
  – a complete procedure for proving sentences by refutation.
  – sentences need to be reformulated into a normal form.
    • conjunctive normal form, or
    • implicative normal form.

• **Modus Ponens**:
  – uses unification in a forward-chaining or backward-chaining algorithm; is not complete.
  – sentences need to be reformulated into a canonical form:
    • nonnegated atomic sentences, or
    • **Horn clauses**: \( p_1 \land \ldots \land p_n \Rightarrow q \), where \( p_i \) and \( q \) are atoms.
Modus Ponens: Canonical Form

• Convert sentences into canonical form sentences:
  – **Existential Elimination** and **And Elimination**:
    “∃ x Owns (John, x) ∧ Missile (x)” is replaced with:
    “Owns(John, M1)”
    “Missile(M1)”
  – Transform complex sentences into Horn clauses:
    “¬P₁ ∨ ¬P₂ ⇒ Q” is replaced with P₁ ∧ P₂ ⇒ Q

• Not all sentences can be transformed into the canonical form for Modus Ponens:
  ⇒ Modus Ponens is not a complete inference procedure.
Modus Ponens

**Generalized Modus Ponens:**

- For atomic sentence $p_i$, $p'_i$ and $q$ and a substitution of variables $\theta$ such that $\text{Subst}(\theta, p'_i) = \text{Subst}(\theta, p_i)$ for all $i$:
  
  $p'_1, p'_2, \ldots, p'_n, (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \models \text{Subst}(\theta, q)$

**Example:**

- $p'_1$ is $\text{King}(John)$  
  $p_1$ is $\text{King}(x)$

- $p'_2$ is $\text{Greedy}(y)$  
  $p_2$ is $\text{Greedy}(x)$

- $\theta$ is $\{x/\text{John}, y/\text{John}\}$  
  $q$ is $\text{Evil}(x)$

- $\text{Subst}(\theta, q)$ is $\text{Evil}(\text{John})$
Unification

• We can use Modus Ponens immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$.

$\theta = \{x/\text{John}, y/\text{John}\}$ works

• $\text{Unify}(\alpha, \beta) = \theta$ if $\text{Subst}(\theta, \alpha) = \text{Subst}(\theta, \beta)$

$$\begin{align*}
&\text{p} & \text{q} & \theta \\
&\text{Knows}(\text{John},x) & \text{Knows}(\text{John},\text{Jane}) \\
&\text{Knows}(\text{John},x) & \text{Knows}(y,\text{OJ}) \\
&\text{Knows}(\text{John},x) & \text{Knows}(y,\text{Mother}(y)) \\
&\text{Knows}(\text{John},x) & \text{Knows}(x,\text{OJ})
\end{align*}$$
Unification

- We can use Modus Ponens immediately if we can find a substitution $\theta$ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$.

$\theta = \{x/John,y/John\}$ works

- $Unify(\alpha,\beta) = \theta$ if $Subst(\theta, \alpha) = Subst(\theta, \beta)$

\[
\begin{align*}
  p & \quad q & \quad \theta \\
  Knows(John,x) & \quad Knows(John,Jane) & \quad \{x/Jane\} \\
  Knows(John,x) & \quad Knows(y,OJ) \\
  Knows(John,x) & \quad Knows(y,Mother(y)) \\
  Knows(John,x) & \quad Knows(x,OJ)
\end{align*}
\]
Unification

- We can use Modus Ponens immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$.

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\text{Subst}(\theta, \alpha) = \text{Subst}(\theta, \beta)$

\[
\begin{array}{ccc}
\text{p} & \text{q} & \theta \\
\text{Knows}(\text{John}, x) & \text{Knows}(\text{John}, \text{Jane}) & \{x/\text{Jane}\} \\
\text{Knows}(\text{John}, x) & \text{Knows}(y, \text{OJ}) & \{x/\text{OJ}, y/\text{John}\} \\
\text{Knows}(\text{John}, x) & \text{Knows}(y, \text{Mother}(y)) \\
\text{Knows}(\text{John}, x) & \text{Knows}(x, \text{OJ})
\end{array}
\]
Unification

- We can use Modus Ponens immediately if we can find a substitution $\theta$ such that $\text{King}(x)$ and $\text{Greedy}(x)$ match $\text{King}(\text{John})$ and $\text{Greedy}(y)$.

$\theta = \{x/\text{John}, y/\text{John}\}$ works

- $\text{Unify}(\alpha, \beta) = \theta$ if $\text{Subst}(\theta, \alpha) = \text{Subst}(\theta, \beta)$

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

- We can use Modus Ponens immediately if we can find a substitution $\theta$ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$.

$\theta = \{x/John,y/John\}$ works

- $Unify(\alpha,\beta) = \theta$ if $Subst(\theta, \alpha) = Subst(\theta, \beta)$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows(John,x)</td>
<td>Knows(John,Jane)</td>
<td>${x/Jane}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,OJ)</td>
<td>${x/OJ,y/John}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(y,Mother(y))</td>
<td>${y/John,x/Mother(John)}$</td>
</tr>
<tr>
<td>Knows(John,x)</td>
<td>Knows(x,OJ)</td>
<td>${fail}$</td>
</tr>
</tbody>
</table>
Most General Unifier

• To unify $\text{Knows}(\text{John},x)$ and $\text{Knows}(y,z)$:
  – $\theta = \{y/\text{John}, x/z \}$ or
  – $\theta = \{y/\text{John}, x/\text{John}, z/\text{John} \}$

• The first unifier is **more general** than the second.

• There is a single **most general unifier** (MGU) that is unique up to renaming of variables.
  – $\text{MGU} = \{y/\text{John}, x/z \}$
The Unification Algorithm

function UNIFY(x, y, θ) returns a substitution to make x and y identical
inputs: x, a variable, constant, list, or compound
        y, a variable, constant, list, or compound
        θ, the substitution built up so far

if θ = failure then return failure
else if x = y then return θ
else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
else if COMPOUND?(x) and COMPOUND?(y) then
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
else if LIST?(x) and LIST?(y) then
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
else return failure
The Unification Algorithm

function $UNIFY\text{-}VAR(var, x, \theta)$ returns a substitution

inputs: $var$, a variable
        $x$, any expression
        $\theta$, the substitution built up so far

if $\{var/val\} \in \theta$ then return $UNIFY(val, x, \theta)$
else if $\{x/val\} \in \theta$ then return $UNIFY(var, val, \theta)$
else if $OCCUR\text{-}CHECK?(var, x)$ then return failure
else return add $\{var/x\}$ to $\theta$
Modus Ponens: Example

• Knowledge Base:
  “The law says that it is a crime for an American to sell
  weapons to hostile nations. The country Nono, an enemy
  of America, has some missiles, and all of its missiles were
  sold to it by Colonel West, who is American.”

• Prove that “Colonel West is a criminal”.
Modus Ponens: Example

... it is a crime for an American to sell weapons to hostile nations:
\[ American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \implies Criminal(x) \]

Nono … has some missiles, i.e., \( \exists x \) Owns(Nono,x) \( \land \) Missile(x):
\[ Owns(Nono,M_1) \text{ and } Missile(M_1) \]

… all of its missiles were sold to it by Colonel West
\[ Missile(x) \land Owns(Nono,x) \implies Sells(West,x,Nono) \]

Missiles are weapons:
\[ Missile(x) \implies Weapon(x) \]

An enemy of America counts as "hostile":
\[ Enemy(x,America) \implies Hostile(x) \]

West, who is American …
\[ American(West) \]

The country Nono, an enemy of America …
\[ Enemy(Nono,America) \]
function FOL-BC-Ask(KB, goals, $\theta$) returns a set of substitutions

inputs: $KB$, a knowledge base

goals, a list of conjuncts forming a query

$\theta$, the current substitution, initially the empty substitution $\{\}$

local variables: $ans$, a set of substitutions, initially empty

if goals is empty then return $\{\theta\}$

$q' \leftarrow \text{SUBST}(\theta, \text{FIRST(goals)})$

for each $r$ in $KB$ where $\text{STANDARDIZE-APART}(r) = (p_1 \land \ldots \land p_n \Rightarrow q)$

and $\theta' \leftarrow \text{UNIFY}(q, q')$ succeeds

$ans \leftarrow \text{FOL-BC-Ask}(KB, [p_1, \ldots, p_n|\text{REST(goals)}], \text{COMPOSE}(\theta, \theta')) \cup ans$

return $ans$

- $\text{SUBST}($COMPOSE($\theta_1, \theta_2$), p) = $SUBST($\theta_2$, $\text{SUBST}($$\theta_1$, p$))$
Backward chaining example
Backward chaining example

\[
\text{Criminal(West)} \quad \{x/\text{West}\}
\]

\[
\text{American}(x) \quad \text{Weapon}(y) \quad \text{Sells}(x,y,z) \quad \text{Hostile}(z)
\]
Backward chaining example

```
American(West)  Weapon(y)  Sells(x,y,z)  Hostile(z)
{   }          
```

Lecture 13
Backward chaining example
Backward chaining example

```
Criminal(West)
  \{ x/West, y/MI \}
  \-- American(West) \{ \}
  \-- Weapon(y)
  \-- Sells(x,y,z)
  \-- Hostile(z)
     \-- Missile(y) \{ y/MI \}
```
Backward chaining example

Diagram:

- **Criminal(West)**
  - **American(West)**: {}
  - **Weapon(y)**: {}
  - **Sells(West, M1, z)**: {z/Nono}
    - **Missile(y)**: {y/M1}
    - **Missile(M1)**
    - **Owns(Nono, M1)**
  - **Hostile(z)**
Backward chaining example
Backward chaining example
Backward-Chaining

- Depth-first recursive proof search:
  - Space Complexity is linear in the size of proof.

- Incomplete due to infinite loops:
  - Fix by checking current goal against every goal on stack.

- Inefficient due to repeated subgoals (both success and failure):
  - Fix using caching of previous results (need extra space).

- Widely used for logic programming.
Logic Programming

- **Declarative semantics:**
  - There is a simple way to determine the meaning of each statement.
  - The meaning can be concisely determined from the statement itself.
  - The meaning does not depend on how the statement might be used to solve a problem:
    - Sentences in FOL vs. statements in imperative languages.
    - Simpler than the semantics of imperative languages.
- Programming is **nonprocedural:**
  - Programs do not state how a result is to be computed, but rather the form of the result.
Example: Sorting a List

• Describe the characteristics of a sorted list, not the process of rearranging a list.

\[
\text{sort}(\text{old\_list}, \text{new\_list}) \iff \text{permute}(\text{old\_list}, \text{new\_list}) \land \text{sorted}(\text{new\_list})
\]

\[
\text{sorted}(\text{list}) \iff \forall j \text{ such that } 1 \leq j < n, \text{ list}(j) \leq \text{ list}(j+1)
\]
Logic Programming: Prolog

- University of Aix-Marseille (Colmerauer & Roussel)
  - Natural language processing.
- University of Edinburgh (Kowalski)
  - Automated theorem proving.

- Algorithm = Logic + Control
- Backward-Chaining with Horn clauses
  + bells & whistles.

- Widely used in Europe, Japan (basis of 5th Generation project).
Logic Programming: Prolog

- Program = set of clauses:
  head :- literal₁, ... literalₙ.

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

- Depth-first, left-to-right backward chaining.
- Built-in predicates for arithmetic:
  X is Y*Z+3
- Built-in predicates that have side effects:
  - input and output predicates, assert/retract predicates.
- Closed-world assumption ("negation as failure"):
  - e.g., given alive(X) :- not dead(X).
  - alive(joe) succeeds if dead(joe) fails
Prolog: Syntax (Edinburgh)

- **Term**: a constant, variable, or structure.
- **Constant**: an atom or an integer.
- **Atom**: symbolic value of Prolog.
  - a string of letters, digits, underscores beginning with a lowercase letter.
  - a string of printable ASCII characters delimited by apostrophes.
- **Variable**: any string of letters, digits, and underscores beginning with an uppercase letter.
- **Instantiation**: binding of a variable to a value.
  - Lasts only as long as it takes to satisfy one complete goal
- **Structure**: represents atomic proposition:
  \[
  \text{functor} \ (\text{parameter list})
  \]
Prolog: Fact Statements

• Used for the unconditional assertions.

• Headless Horn clauses:
  female(shelley).
  male(bill).
  father(bill, jake).
Prolog: Rule Statements

• Used for the conditional assertions.

• Headed Horn clause
  – Right side: antecedent (*if* part)
    • May be single term or *conjunction*.
  – Left side: consequent (*then* part)
    • Must be single term.

• *Conjunction*: multiple terms separated by commas.
  – logical AND operator is implied.
Prolog: Example Rules

ancestor(mary, shelley) :- mother(mary, shelley).

- Can use variables (universal objects) to generalize meaning:
  parent(X, Y) :- mother(X, Y).
  parent(X, Y) :- father(X, Y).
  grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
  sibling(X, Y) :- mother(M, X), mother(M, Y),
                  father(F, X), father(F, Y).
For theorem proving, theorem is in form of a sentence that we want system to prove or disprove:

- goal statement, or query.

Same format as headless Horn:

- man(fred)
- Conjunctive sentences and sentences with variables are also legal goals:
  - father(X,mike)
Prolog: Simple Arithmetic

• Prolog supports integer variables and integer arithmetic
• is operator: takes an arithmetic expression as right operand and variable as left operand
  \[ A \text{ is } B / 17 + C \]
• Not the same as an assignment statement!
speed(ford, 100).
speed(chevy, 105).
speed(dodge, 95).
speed(volvo, 80).
time(ford, 20).
time(chevy, 21).
time(dodge, 24).
time(volvo, 24).

distance(X, Y) :- speed(X, Speed),
time(X, Time),
Y is Speed * Time.

Prolog: Example
Prolog: Trace

- Built-in structure that displays instantiations at each step.
- Tracing model of execution - four events:
  - Call (beginning of attempt to satisfy goal)
  - Exit (when a goal has been satisfied)
  - Redo (when backtrack occurs)
  - Fail (when goal fails)
likes(jake, chocolate).
likes(jake, apricots).
likes(darcie, licorice).
likes(darcie, apricots).

trace.
likes(jake, X),
likes(darcie, X).
Prolog: Lists

- Other basic data structure besides atomic sentences.
- *List* is a sequence of any number of elements.
- Elements can be atoms, atomic sentences, or other terms (including other lists)

\[
\text{[apple, prune, grape, kumquat]}
\]
\[
\text{[]} \quad \text{(empty list)}
\]
\[
\text{[X | Y]} \quad \text{(head X and tail Y)}
\]
Examples: Append and Reverse

append([], List, List).
append([Head | List_1], List_2, [Head | List_3]) :-
append (List_1, List_2, List_3).
reverse([], []).
reverse([Head | Tail], List) :-
reverse (Tail, Result),
append (Result, [Head], List).
Examples: Append and Reverse

- **Query:** `append(A, B, [1, 2]).`

- **Answers:**
  - `A=[]    B=[1, 2]`
  - `A=[1, 2] B=[]`
Deficiencies of Prolog

- Resolution order control.
- The closed-world assumption.
- The negation problem.
  - the use of Horn clauses prevents negative conclusions.
- Intrinsic limitations.
  - cannot transform declarative specification of the sorting problem into an efficient sorting algorithm.
Applications of Logic Programming

- Relational database management systems.
- Expert systems.
- Natural language processing.