Logic Languages

- Programs as a form of symbolic logic and logical inference as computational mechanism
- Declarative languages
  - Specify problem not algorithm
  - Non-procedural
- Declarative semantics
  - Meaning of statement determined from statement itself independent of context
- Program is a set of facts, propositions and queries
  - Run-time infers answers to queries based on propositions and facts using logical inference

Predicate Calculus

- Proposition – a statement that may or may not be true
  - Objects and relationships between them
- Symbolic logic
  - Express propositions
  - Express relationships
  - Describe how new propositions can be inferred
- Logic languages use first-order predicate calculus
**Propositions**

- **Objects**
  - Constant
  - A specific object
  - Variable
  - Can represent different objects at different times
- **Atomic propositions**
  - Consist of compound terms
- **Compound term**
  - Functor - name of relation as functional symbol
  - Tuple - set of elements of the relation
- **Propositions**
  - Have no intrinsic semantics
  - Proposition can be a fact or a query
  - Compound proposition built from atomic propositions and logical connectors
  - Variables appear via quantifiers

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
<td>( \neg )</td>
<td>( \neg a )</td>
<td>not ( a )</td>
</tr>
<tr>
<td>conjunction</td>
<td>( \land )</td>
<td>( a \land b )</td>
<td>( a ) and ( b )</td>
</tr>
<tr>
<td>disjunction</td>
<td>( \lor )</td>
<td>( a \lor b )</td>
<td>( a ) or ( b )</td>
</tr>
<tr>
<td>equivalence</td>
<td>( \equiv )</td>
<td>( a \equiv b )</td>
<td>( a ) is equivalent to ( b )</td>
</tr>
<tr>
<td>implication</td>
<td>( \rightarrow )</td>
<td>( a \rightarrow b )</td>
<td>( a ) implies ( b )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( a \equiv b )</td>
<td>( a ) implies ( b )</td>
</tr>
</tbody>
</table>

\( a \land b \equiv c \quad a \lor b \equiv d \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal</td>
<td>( \forall X, P )</td>
<td>For all ( X ), ( P ) is true.</td>
</tr>
<tr>
<td>existential</td>
<td>( \exists X, P )</td>
<td>There exists a value of ( X ) such that ( P ) is true.</td>
</tr>
</tbody>
</table>

\( \forall X \) \( \text{woman}(X) \quad \exists X, \text{mother}(X, X) \) \( \text{mother}(X) \)
Clausal Form

- A standardized form of expressing a proposition

**Syntax**
- Antecedent
  - Right hand side
  - Anded terms
- Consequent
  - Left hand side
  - Ored terms
- Explicit quantification not required
  - Existential not required
  - Universal implied
- If antecedent is true then the consequent is true
  - If all terms in antecedent true then at least one term in consequent is true

Resolution

- Resolution is an inference rule that allows inferred propositions to be derived from given propositions
  - Transitivity
- Process
  - Left hand sides are ored together
  - Right hand sides are anded together
  - Any term on both left & right is removed
- Unification
  - Resolution requires finding values of variables that allow matching to succeed
  - One choice might fail then backtrack and make another choice
  - Resolution of general proposition is not computationally effective
- Horn clause
  - Headed – single proposition on left hand side
  - State relationships
  - Headless – empty left hand side
  - State facts
Prolog

- Program
  - Series of statements
- Term
  - Constant
    - Atom
      - Begins with lowercase letter
    - Integer
  - Variable
    - Begins with uppercase letter
    - Instantiated during resolution
- Structure
  - Atomic proposition
  - Functor name is an atom
  - Parameters are atoms, variables or structures

- Facts
  - Database of assumed information
  - Headless Horn clause
- Rules
  - Headed Horn clause
  - Related to theorems in mathematics
  - If antecedent (rhs) is true then consequent (lhs – head) is true
  - Antecedent can contain conjunctions (and)
    - No need for operator since and implied
    - Separated by commas
  - Variables used for generalization
- Goals
  - Headless Horn clause
  - System tests inference
    - Yes – can be inferred
    - No – cannot be inferred from facts and rules
Inference

- In compound proposition, each term is a subgoal
- Must produce a chain of rules and facts that connects the goal to a fact
- Variables must be unified
- Backward chaining
  - Start from facts and try to find sequence that leads to goal
  - Forward chaining
  - Start from goal and try to find sequence to facts
- Depth-first
  - Prove each subgoal in turn
- Breadth-first
  - Develop proofs of subgoal in parallel
- Backtracking
  - Abandon subgoal that cannot be proved and back up to an alternative for previous subgoal
Examples

- Distance travelled
  - Arithmetic
  - Trace
- Likes
- Backtracking
- Append
  - Lists
    - Head, tail

```
P₁ : - Pₐ
P₂ : - Pₐ
Q : - Pₐ

man(bob).
man(bob).
father(bob).
man(X) :- father(X).

A is B / 17 + C.
```

```
speed(ford, 100).
speed(chevy, 105).
speed(volvo, 80).
time(ford, 20).
time(chevy, 31).
time(volvo, 24).
distance(X, Y) :- speed(X, Speed),
              distance(X, Time),
              Y is Speed * Time.

distance(chevy, CHEVY_DISTANCE).
```
likes(jake, chocolate).
likes(jake, apricot).
likes(darcie, licorice).
likes(darcie, apricot).
likes(jake, X), likes(darcie, X).

(1) 1 Call: likes(jake, _0) ?
(2) 1 Call: likes(darcie, chocolate)
(3) 1 Exit: likes(jake, apricot)
(4) 1 Exit: likes(darcie, apricot)
(5) 1 Exit: likes(darcie, apricot)

[apples, prunes, grape, rumput]

[New_List_Head | New_List_Tail]

[Element_1 | List_2]

append([], List, List).
append([Head | List_1], List_2, [Head | List_1]) : =
append(List_1, List_2, List_2).

append([bob, 50], [jake, darcie], Family).

(1) 1 Call: append([bob, 50], [jake, darcie], _0) ?
(2) 2 Call: append([50], [jake, darcie], _0) ?
(3) 3 Exit: append([], [jake, darcie], [jake, darcie])
(4) 2 Exit: append([50], [jake, darcie], [j0, jake, darcie])
(5) 1 Exit: append([bob, 50], [jake, darcie],
 [bob, 50, jake, darcie])
Family « [bob, j0, jake, darcie]