Introduction

- **Prolog is a logical programming language and stands for PROgramming in LOGic**
- **invented in the early seventies at the University of Marseille**
- **Preferred for AI programming and mainly used in such areas as:**
  - Theorem proving, expert systems, NLP, …
- **Differs from the most common programming languages because it is declarativre language (Traditional programming languages are said to be procedural)**
- **Logical programming is the use of mathematical logic for computer programming.**

Introduction (Cont’d)

- For symbolic, non-numeric computation
- e.g. : parent (tom, bob).
- Parent is a relation between its parameters: tom and bob
- The whole thing is called a clause
- Each clause declares one fact about a relation

Prolog

- Prolog has an interactive interpreter
- After starting SWI-Prolog, the interpreter can start reading your Prolog files and accept your queries.
- To exit Prolog simply type the command ‘halt.’ (Notice the full-stop)
- Prolog program files usually have the extension .pl or .pro

How to run Prolog

- **Starting Prolog on Windows**
  - Start->Programs->SWI-Prolog->Prolog
- you should then see something like this on your screen after you start SWI-Prolog:

```
Welcome to SWI-Prolog (Multi-threaded, Version 5.2.13)
Copyright (c) 1990-2003 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

- Once you have started Prolog you can send commands and queries to the Prolog interpreter by typing them in after the ?-> prompt.
- **listing**
  - makes Prolog show you the facts and rules that are currently loaded

```
1 ?- listing.
Yes
```

- The Yes indicates that the interpreter has succeeded with the command. However, it has not listed anything. This tells us that the knowledge base is empty, And it should be, since you have not actually loaded anything, yet.

How to run Prolog

- **assert**
  We can add a fact or rule to the knowledge base using the assert command.
- Type, assert(likes(mary,john)). at the ?- prompt. This adds the fact like(mary,john) to the knowledge base. We are assert that there is a directed likes relationship between the entity mary to the entity john, i.e. mary likes john. Prolog reponds with Yes to indicate that the fact was successfully asserted. You should get something like:

  -
    ```
    3 ?- assert(likes(mary,john)).
    Yes
    4 ?-
    ```

Statements

- **There are three categories of statements in Prolog:**
  - **Facts:** Those are true statements that form the basis for the knowledge base.
  - **Rules:** Similar to functions in procedural programming (C++, Java…) and has the form of if/then.
  - **Queries:** Questions that are passed to the interpreter to access the knowledge base and start the program.

Facts

- A fact is a one-line statement that ends with a full-stop.
  - parent (john, bart).
  - parent (barbara, bart).
  - male (john).
  - male (bart).
  - female (barbara).

Rules

- **A Rule consists of**
  - a condition part (right-hand side) → body of clause
  - a conclusion part (left-hand side) → head of clause
  - They are separated by ':-' which means 'if'
- **offspring relation**
  - offspring (X, Y) : X is an offspring of Y
  - ∀ X,Y (offspring (X, Y) ← parent (Y, X))
  - $\underbrace{\text{offspring (X, Y)}}_{\textbf{head}}$:- $\underbrace{\text{parent (Y, X)}}_{\textbf{body}}$.

Rules (Cont’d)

- **Variables in head of rules are universally quantified**
- **Variables appearing only in the body are existentially quantified**
- **Rules vs. Facts**
  - A Fact is something unconditionally true
  - Rules specify things that are true if some condition is satisfied

Queries

- **Queries are questions**
- **The engine tries to entail the query (goal) using the Facts and Rules in KB**
- **There are two kinds of answer**
  - Yes/No: parent (tom, bob).
  - Unified Answer/No: parent (X, bob).
- **Other possible answer(s) can be found using semicolon (return for stopping)**
- **For example : parent (X, Y).** →

```
X=pam
Y=bob;

X=tom
Y=bob;

X=tom
Y=liz;

no
```

Queries (Cont'd)

- **Q: Who is a grandparent of Jim? (using parent relationship)**
  - Who is a parent of Jim? Assuming "Y"
  - Who is a parent of "Y"? Assuming "X"
  - ?- parent (Y, jim) , parent (X, Y).
  - If we change the order of them the logical meaning remains the same
- **Q: Who are Tom's grandchildren?**
- **Q: Are Ann and Pat siblings?**

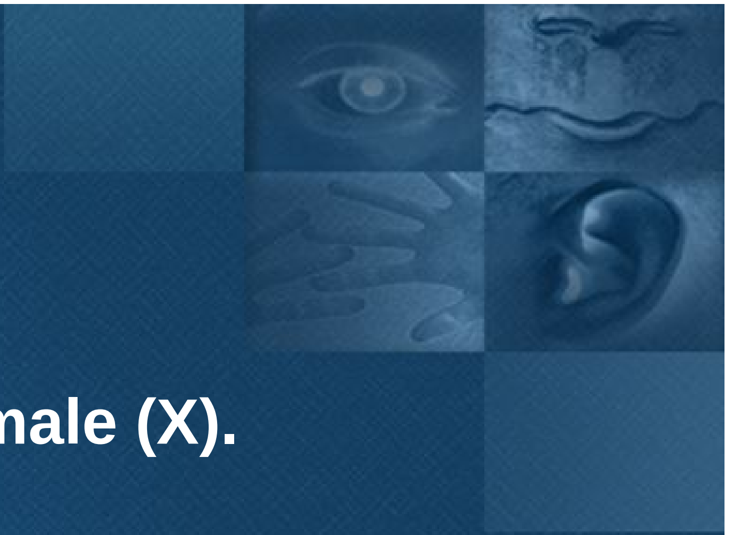
Where the program is written?

- Facts and Rules are stored in one or more files forming our Knowledge Base
- Files containing KB are loaded into the interpreter
- After changing these files, the files should be loaded again to be effective
- Queries are asked in the interactive mode in front of the question prompt: ?-

Reading Files

- **consult (filename).**
  - Reads and compiles a Prolog source file
  - consult ('/home/user/prolog/sample.pl').
- **reconsult (filename).**
  - Reconsult a changed source files.
  - reconsult('/home/user/prolog/sample.pl').
- **[‘filename’].**
  - ['/home/user/prolog/sample.pl'].
- **make.**
  - Reconsult all changed source files.

Examples



- mother (X, Y) :- parent (X, Y) , female (X).
- sister (X, Y) :-

  parent (Z, X) ,

  parent (Z, Y) ,

  female (X).
- What is wrong with this rule?
- Any female is her own sister
- Solution?

Comments

- Multi-line :

```
/* This is a comment
   This is another comment */
```

- Short :

```
% This is also a comment
```

Prolog Syntax

- **Terms in Prolog:**
  - Simple
    - Constants:
      - Atoms
      - Numbers
        - Integer
        - Real
    - Variables
  - Complex Structures

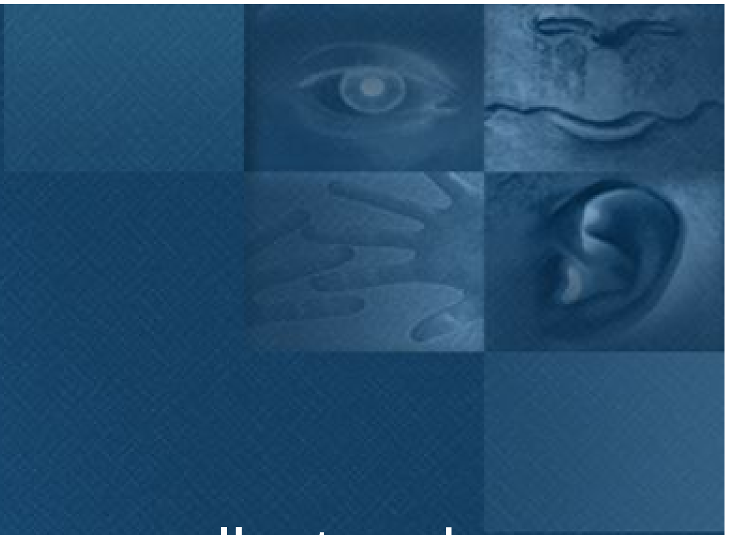
Atoms

- **They should consist of the following set of characters:**
  - The *upper-case letters*
  - The *lower-case letters*
  - The *digits*
  - The *special characters:* +, -, *, /, <, >, =, :, ., &, ~, _
- **Atoms should not start with upper-case letters or underscore and can be followed by any set of characters.**
- **The scope of an atom is the whole program**

Examples of Atoms

- anna, x30, x_, x___y, miss_Jones
- <---> , ==>, … , .:. , ::= (except reserved ones like :-)
- ‘Tom’ , ‘Sarah Jones’ (Useful for having an atom starting with a capital letter)

Numbers



- **Integer**
  - limited to an interval between some smallest and some largest number permitted by a particular Prolog implementation
  - e.g. : 1, 1001 , 0 , -98
- **Real**
  - Not frequently used
  - e.g. : 3.14 , -0.0035 , 100.2

Variables

- **Consists of letters, digits and ‘_’**
- **Starting with an upper-case or an ‘_’**
- **The variable ‘_’ (a single underscore character) is a special one. It's called the anonymous variable.**
- **The scope of a variable is its clause**
  - If the name X15 occurs in two clauses, it represents two different variables.
  - Each occurrence of X15 within the same clause means the same variable

Structures

- Compound Objects
- Each constituent is a simple object or structure.
- e.g. : date (1, jan, 2007)
- Components can be variables.
- Any day in Jan 2007 → date (Day, jan, 2007)

Conjunction and Disjunction

- **Conjunction → ,**
- **Disjunction → ;**
  - **P :- Q ; R.**
  - **P :- Q**
  - **P :- R**
- ‘,’ has more priority
  - **P :- Q , R ; S , T , U .**
  - **P :- (Q , R) ; (S , T , U) .**

Recursion

- Define ancestor relation based on parent relation.
- ancestor (X, Z) :-

  parent (X, Z).
- ancestor (X, Z) :-

  parent (X,Y) , parent (Y, Z).
- ancestor (X, Z) :-

  parent (X, I) , parent (I, Y) , parent (Y, Z).
- **Solution is Recursion**

Recursion

- Remember from functional programming languages

```
void func (int a , int b)
{
          //base case
           if (condition)
              return;
          …
          // recursion
           func (x, y);
          …
}
```

Recursion

- **Rules in Prolog are like functions in procedural programming languages**
- **For recursion we should define the ancestor relation in terms of itself**
- **Base Case :**
  - ancestor(X, Z) :- parent (X, Z).
- **Recursion Step :**
  - ancestor (X, Z) :- parent (X, Y) , ancestor (Y, Z).

How Prolog Answers Questions

- Instead of starting with simple facts given in the program, prolog starts with the goals. In fact, Prolog does goal driven search.
- Using rules, Prolog substitutes the current goals (which matches a rule head) with new sub-goals (the rule body), until the new sub-goals happen to be simple facts.
- Prolog returns the first answer matching the query. When prolog discovers that a branch fails or if you type ‘;’ to get other answers, it backtracks to the previous node and tries to apply an alternative rule at that node.

Example

- **Facts:**
  - parent (pam, bob). parent (tom, bob). parent (tom, liz).
  - parent (bob, ann). parent (bob, pat). parent (pat, jim).
- **Rules:**
  1. ancestor (X, Z) :- parent (X, Z).
  2. ancestor (X, Z) :- parent (X, Y) , ancestor (Y, Z)
- **?- ancestor (tom, pat). (goal)**
- **The rule that appears first, is applied first**
- **Unifying: {tom/X} , {pat/Z}**
  - The goal is replaced by : parent (tom, pat). (sub-goal)
- **Fails** $\Rightarrow$ backtracking

Example (Cont'd)

- **Applying the next rule**
  2. **ancestor (X, Z) :- parent (X, Y) , ancestor (Y, Z)**
- **Unifying: {tom/X} , {pat/Z}**
  - New Goal: parent (tom, Y) , ancestor (Y, pat)
  - Prolog tries to satisfy them in order in which they are written
  - The first one matches one of the facts {bob/Y}
  - Second sub-goal: ancestor (bob, pat)
  - The same steps should be done for this sub-goal

Orders of Clauses and Goals

1. ancestor (X, Z) :- parent (X, Z).
   ancestor (X, Z) :- parent (X, Y) , ancestor (Y, Z).

2. ancestor (X, Z) :- parent (X, Y) , ancestor (Y, Z).
   ancestor (X, Z) :- parent (X, Z).

3. ancestor (X, Z) :- parent (X, Z).
   ancestor (X, Z) :- ancestor (Y, Z) , parent (X, Y).

4. ancestor (X, Z) :- ancestor (Y, Z) , parent (X, Y).
   ancestor (X, Z) :- parent (X, Z).

Orders of Clauses and Goals

- **It turns out that :**
  - The first and second variations are able to reach and answer for ancestor.
  - The third sometimes can and sometimes can’t
  - And the forth can never reach and answer (infinite recursion)
- **“Try simple things first”.**

More about Prolog

- A collection of facts and rules is called a *knowledge base (KB)*.
- Prolog works based on “*first-order predicate logic*”
- Matching corresponds to what is called *“Unification”*.
- *Prolog* has no data types