Cairo University
Faculty of Computers & Information

Artificial Intelligence course (AI)
Lab#1

Lab objective:

- To introduce prolog,
- Make student familiar with SWI-Prolog

Part I: Prolog Introduction

Artificial Intelligence Programming focuses on the following techniques:

- Knowledge representation and manipulation
- Database construction and management
- State-space search
- Planning
- Text parsing and Definite Clause Grammar

Prolog

- Prolog is short for **Pro**gramming **Log**ic
- Specially good for symbolic computation
- Not suitable for problems involving numeric calculations
- No global variables
- Variables in prolog get their values by being matched to constants in facts or rules
- Recursion is employed instead of iterative constructs

Getting started with Prolog

Prolog programs are based on a database containing facts and rules that define relations between objects.

- Clauses of prolog are of three types: facts, rules and questions.
- Facts as seen define things that are unconditionally (always) true.
- Rules define things that are true depending on given conditions.

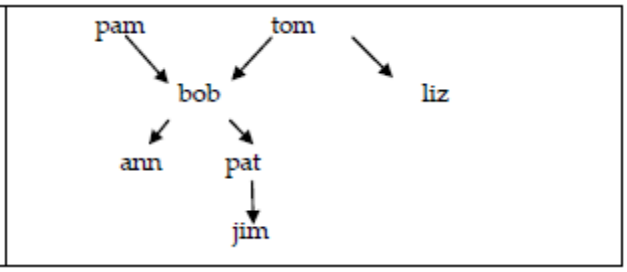
1. Defining facts:

Prolog is a first order predicate language. So, Defining facts in prolog can be simply done by stating the n-tuples of objects that satisfy the relation.

Example 1:

The following predicates (or ***clauses***) define the family tree relations on the right side.

```
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
```



- Facts are constructs that are always true.
- Each clause declares one fact about the parent relation. The first clause for example is a particular instance of the parent relation. Such an instance is also called a relationship. In general, a relation is defined as the set of all its instances.
- Clauses should start with lower-case letter.
- Each clause terminates with a full stop.
- The arguments of relations can be: concrete objects (***constants/atoms***) or general objects (***variables)***.
- Variables start with upper case letters.

A question can be:

```
parent(bob,pat).
True
parent(bob,X).
X=ann;
X=pat;
```

2. Defining Rules:

Rules consist of:

- The head part: conclusion on the left hand side of the rule
- The body part: conditions in the right hand side of the rule

The body is a list of goals separated by commas (conjunction) and sometimes (rarely used) semicolon (disjunction).

Prolog rules of the same name forms ***procedures***.

Example 2:

The off spring relationship can be defined as follows:

```
For all X and Y
Y is an offsring of X if
X is a parent of Y
```

The corresponding prolog clause which has the same meaning is:

```
offspring(Y,X):-
   parent(X,Y).
```

The following rule defines the grandparent relation:

```
grandparent(X,Y):-
   parent(X,Z),
   parent(Z,Y).
```

":-" means if.

3. Recursive Rules:

The key idea of the recursive rule is: when defining something, we can use or call the same thing that has not been yet completely defined. Such definitions are in general called recursive definitions. Recursive programming is in fact one of the fundamental principles in prolog programming.

Example 3:

Let us define the predecessor relation. This relation can be defined in terms of the parent relation. The whole definition can be expressed with two rules. The first rule will define the direct predecessor and the second rule for the indirect predecessors. We say that X is an indirect predecessor of Z if there is a parentship chain of people between X and Z. so the definition of the predecessor can be as follows:

```
For all X and Z,
X is a predecessor of Z if
There is a Y such that
X is a parent of Y and
Y is a predecessor of Z.
```

A prolog clause with the above meaning is as follows:

```
predecessor(X,Z):-
      parent(X,Z).

predecessor(X,Z):-
      parent(X,Y),
      predecessor(Y,Z).
```

A question can be:

```
predecessor(pam,X).
X=bob;
X=ann;
X=pat;
X=jim;
```

4. Prolog data objects:

Atoms and numbers:

The alphabets that make up symbols are:

- `A < Z, a < Z.`
- `The set of digits: 0 < 9`
- Special characters such as: `_ + - * / < > = : & . ~`

Atoms need to start with a letter. Ex: **person**. Strings of characters enclosed in single quotes are also atoms. This makes it possible to have an atom that starts with a capital letter. '**Tom**', '**South Africa**'.

Numbers are integers or real numbers. Real numbers are not commonly used in prolog because it is primarily a language for symbolic non numeric computations.

Variables:

Variable symbols are symbol expressions beginning with an uppercase character. E.g., X, Y, Z, Building.

Anonymous variables (_): When a variable appears in clause only once, we do not have to invent a name for it. we can use the underscore instead, **For example:**

```
hasachild(X):-
   parent(X,_).
```

Notice that each time the underscore occurs in a clause it represents a new anonymous variable.

If the underscore occurs in a prolog question, then its value is not output when the prolog answers the question. **For example:** `parent(X,_).` Will list the name of parents only.

Structures:

They are objects that have several components. The components themselves can in turn be structures.

Example 1:

Constant components:

- date(1,may,2001).
- color_of(house_of(neighbor(joe))).
- `maximum(maximum(7, 18), add_one(18)).`

Variable components:

- date(Day,may,2001).
- color_of(house_of(neighbor(X))).
- maximum(maximum(7, X), add_one(X)).

All structured objects can be pictured as trees.

The root of the tree is the ***functor*** and the offsprings of the root are the components. If a component is also a structure, then it is a subtree of the tree that corresponds to the whole structured objects.

The number of elements (arguments) in the domain is called ***the arity*** of the structure.

We can use more than clause with the same functor name but with different arity, **for example:**

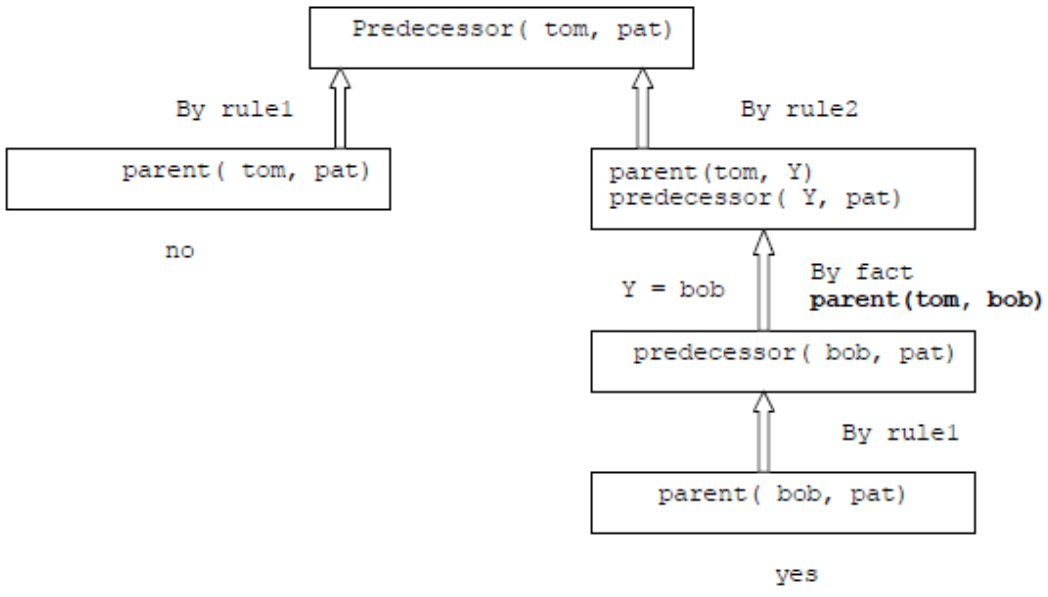
- point(1,1).
- point(1,1,1).

The ***lexical scope*** of variable names is one clause. This means that if the name X occurs in two clauses, then it signifies two different variables. But the occurrence of X within the same clause means the same variable.	

5. How prolog answers questions:

- A question to prolog is always a sequence of one or more goals. To answer a question, prolog tries to satisfy all the goals. What does it mean to satisfy a goal? To satisfy a goal means to demonstrate that the goal is true, assuming that the relations in the program are true.
- If the question contains variables, prolog also had to find what are the particular objects (in place of variables) for which the goal is satisfied. The particular instantiation of variables to these objects is displayed to the user.
- If prolog cannot demonstrate for some instantiations of variables that the goals logically flow from program, then the prolog's answer to the question will be no.
- The prolog finds the proof sequence in the inverse order (i.e. in a goal driven manner). prologs starts with the goal and, using rules, substitutes the current goals with new sub goals, until new goals happen to be simple facts. If some goal fails, the prolog ***backtracks*** to try another alternative way to derive the top goal until success or fail to find adequate substitution.

Example 4: The following shows the execution trace of predecessor example**:**
predecessor(tom, pat).



6. Unification & Matching

- Basic idea of **unification** is variables replacement:
  - other variables
  - constants
  - function expressions

  Given two terms, we say that they match if:
  - They are identical, or
  - The variables in both terms can be instantiated to objects in such away that after the substitution of variables by these objects the terms become identical.

- **Matching** is a process that takes two terms as input and checks whether they can match. If they do not match we say that the process fails, if they do match we say that the process succeeds and it also instantiates the variables in both terms to such values that make the terms identical.

- The prolog often begins the matching process when finds the operator '='. Matching in prolog always results in ***the most general unifier MGU*** instantiation. Why?

- Because simply MGU commits the variables to the least possible extent, thus leaving the greatest possible freedom for further unifications if further matches are required (*see the second problem in example2 below*).

- The general rules for matching S and T are:

  - If S and T are constants then they match only if they are the same thing.
  - If S is a variable and T is anything, then they match and S is bound to T, and vice versa.
  - If S and T are structures then they match only when they have the same principal functor and all their corresponding components match.

Example 5:

1. date(D,M,2001) and date(D1,may,Y) match and give the following:

   D=D1
   M=may
   Y=2001

2. date(D,M,2001) = date(D1,may,Y1), date(D,M,2001) = date(15,M,Y) give the following:

D=D1
M=may
Y1=2001 *after satisfying the first goal*
D=15
D1=15
M=may
Y1=2001
Y=2001 *after satisfying the second goal*

Part II: Getting familiar with SWI-Prolog

- SWI-Prolog Prolog is a prolog programming environment
- The prolog code is written in a text file (or more than one file) using a standard text editor.
- Files are saved in **.pl** files
- SWI-Prolog is then instructed to read the programs from these files (which is called *consulting* the files)

Reading in programs

- To input a program from a file, use the query:
  |? - consult ('C: \myfile.pl').

- Facts and rules written in the file are stored by SWI-Prolog for later use
- The consult query can also take multiple file names as argument:
  |? – consult ([file1, file2, file3]).

Using the terminal to write code

- Prolog code may be even typed on the terminal. However, it's only recommended to use such an option if the constructs are not needed permanently, and are few in number.
- To enter clauses by the terminal, type in the query:

|? – consult (user).

- *user* is a built-in predicate indicating that the input stream employed will be the keyboard.

Prolog Examples on SWI-Prolog

Define the following relations into prolog:

- Father, mother.

```
% father( *X, *Y), where the first argument is the father of the
second
```

```
father(X,Y):-
      parent(X,Y),
      male(X).

% mother( *X, *Y), where the first argument is the mother of the
second

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

- Brother, sister.

```
% brother( *X, *Y), where the first argument is the brother of
the second

brother(X,Y):-
      parent(Z,X),
      parent(Z,Y),
      male(X),
      X \== Y.
% sister( *X, *Y), where the first argument is the sister of the
second

sister(X,Y):-
      parent(Z,X),
      parent(Z,Y),
      female(X),
      X \== Y.
```

Queries and Directives

- Suppose that a membership clause is written as:
  `parent(bob,ann).`
  `parent(bob,pat).`
- The full syntax of a query is a "?-" followed by a sequence of goals

`|?- parent(bob,pat).`

- Remember that prolog programs must terminate with a "."
- If the goals specified in the query have been satisfied, and if there are no variables in this query, then the system answers *yes.*
- If a variable is included in the query, the final value of the variable is displayed. So the output for query:

```
|?- parent(bob,X).
      X=ann;
      X=pat;
```

Once the output of a query is displayed, SWI-Prolog accepts one letter commands corresponding to certain actions:

Y [or enter key] : Accepts the fetched solution
N[or ";"] : Rejects the solution, and forces the search for another solution

Example:

Trace the following program:

```
big(bear).
big(elephant).
small(cat).
brown(bear).
black(cat).
gray(elephant).
dark(Z):-black(Z).
dark(Z):-brown(Z).
```

Answer:

```
?-trace.
?- dark(X),big(X).
1        1 Call: dark(_463) ?
         2      2 Call: black(_463) ?
         2      2 Exit: black(cat) ?
?        1      1 Exit: dark(cat) ?
         3      1 Call: black(cat) ?
         3      1 Exit: black(cat) ?
X = cat ? ;
         1      1 Redo: dark(cat) ?
         4      2 Call: brown(_463) ?
         4      2 Exit: brown(bear) ?
         1      1 Exit: dark(bear) ?
         5      1 Call: black(bear) ?
         5      1 Fail: black(bear) ?
no
```

Exercises:

1. Use a recursive rule to check whether a number is odd or even.
2. In the last exercise, Try to write predicates in different order, explain what happens and why.