Course Code: **CS4767**

Course Title: **Programming Techniques for AI using Prolog**

Prolog LAB Manual Using Visual Prolog 5.0

Prepared By:

Dr. Umair Abdullah

Shaheed Zulfikar Ali Bhutto Institute of Science and Technology

Islamabad Campus, Pakistan

2016

TABLE OF CONTENTS

Visual Prolog Environment _____ 3
Starting the Visual Prolog's Visual Development Environment _____ 3
LAB 1: Basic Prolog Program structure _____ 7
LAB 2: Querying the Prolog _____ 9
LAB 3: Defining new relations using existing facts _____ 11
LAB 4: Defining Recursive Relations _____ 14
LAB 5: How Prolog Answers Questions _____ 16
LAB 6: Encoding knowledge in Prolog _____ 17
LAB 7: Building a phone directory and querying it _____ 19
LAB 8: Output to user _____ 21
LAB 9: Input from User _____ 23
LAB 10: Input and Processing Integer _____ 24
LAB 11: Processing Numbers: Calculating Factorial _____ 26
LAB 12: Finding GCD of two numbers _____ 27
LAB 13: Working with Lists: Printing all elements of a list _____ 28
LAB 14: List Membership _____ 29
LAB 15: Substituting Item from list _____ 30
LAB 16: Finding Maximum of from a List _____ 31

Visual Prolog Environment

This section describes the basic operation of the Visual Prolog system focusing on running the examples of Prolog Programming course. I assume, that you have experience using the Graphical User Interface system. This might be either 16-bit Windows 3.x or 32-bit Windows (95/98 and NT/2000). You should thus know about using menus, closing and resizing windows, loading a file in the **File Open** dialog etc. If you do not have this knowledge, you should not start off trying to create an application that runs under this environment. You must first learn to use the environment.

If you are a beginner to Prolog, you don't want to mix learning the Prolog language with the complexity of creating Windows applications with event handling and all the Windows options and possibilities. The codes for the examples are platform independent: They can run in DOS text mode, under UNIX, or with the adding of a little extra code for the User Interface handling, in a Windows environment.

I do suggest that you at an early stage work your way through the Guided Tour in the Getting Started book and try to compile some of the examples in the VPI subdirectory. This gives you an impression what can be done with Visual Prolog - just so you know what you should be able to do, when you have learned to master Visual Prolog.

Starting the Visual Prolog's Visual Development Environment

The Visual Prolog's installation program will install a program group with an Icon, which are normally used to start Visual Prolog's Visual Development Environment (**VDE**). However, there are many ways to start an application in the GUI World.

If the Visual Development Environment had an open project (a .PRJ or .VPR file) the last time the VDE was closed on your computer, it will automatically reopen this project next time it starts.

If while installation of Visual Prolog's from CD you had checked ON the **Associate 32-bit VDE with Project File Extensions PRJ & VPR**, then you can simply double click on a project file (file name extension .PRJ or .VPR). The Visual Development Environment will be activated and the selected project will be loaded.

To run most examples in this manual, you should use Visual Development Environment's utility **Test Goal**. The **Test Goal** utility can be activated with the menu item **Project | Test Goal** or simply by the hot key **Ctrl+G**. For correct running of these examples with the **Test Goal** utility, the VDE should use the special settings of loaded projects. I recommend you to create and always use the following special **TestGoal** project.

To run with the **Test Goal** utility, Language Tutorial's examples require that some non-default settings of Visual Prolog's compiler options should be specified. These options can be specified as the project settings with the following actions:

1. Start Visual Prolog's VDE.

If this is the first time you start the VDE, then it does not have a loaded project and you will see the picture like this (also you will be informed that the default Visual Prolog initialization file is created):

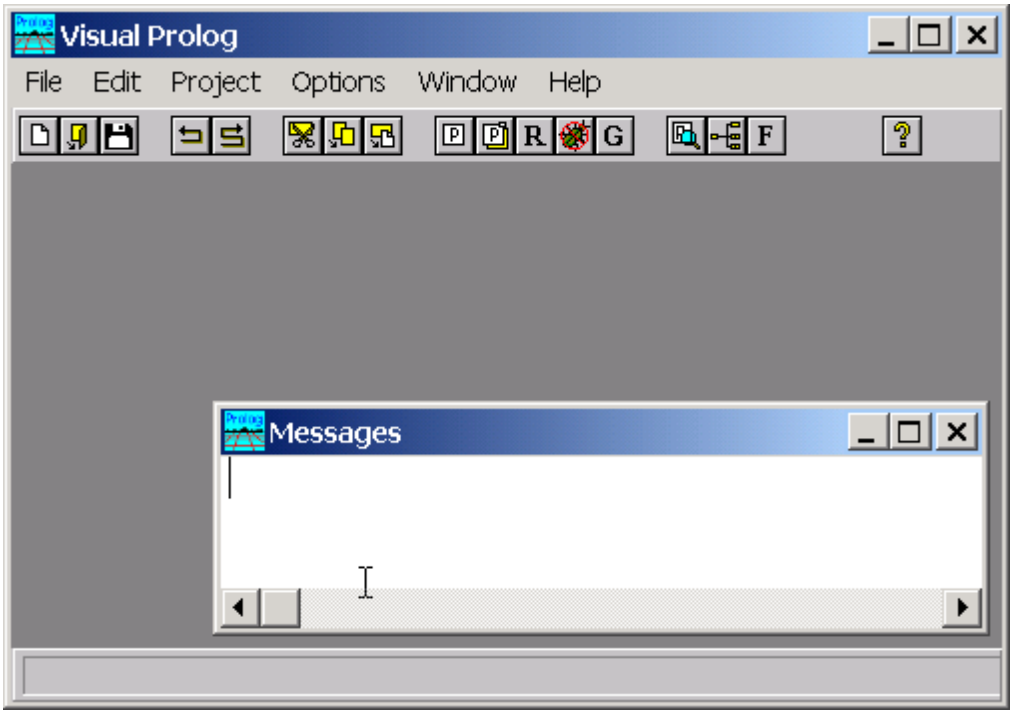


Figure 1.1: Start Visual Development Environment

2. Start creation of a new project.

Select **Project | New Project** menu item, the **Application Expert** dialog will be activated.

3. Specify the **Base Directory** and the **Project Name**.

Opening an Editor Window

To create a new edit window, you can use the menu command **File | New**. This will bring up a new editor window with the title "NONAME". The VDE's editor is a fairly standard text editor. You can use cursor keys and the mouse as you are used to in other editors. It supports **Cut**, **Copy** and **Paste, Undo** and **Redo**, which you can all activate from the **Edit** menu. Also the **Edit** menu shows the accelerator keys associated for these actions. The editor is documented in the *Visual Development Environment* manual.

Running and Testing a Program

To check, that your system is set up properly, you should try to type in the following text in the window:

```
GOAL write("Hello world"),nl.
```

This is what is called a GOAL in the Prolog terminology, and this is enough to be a program that can be executed. To execute the GOAL, you should activate the menu item **Project | Test Goal**, or just press the accelerator key **Ctrl+G**. If your system is installed properly, your screen will look like the following:

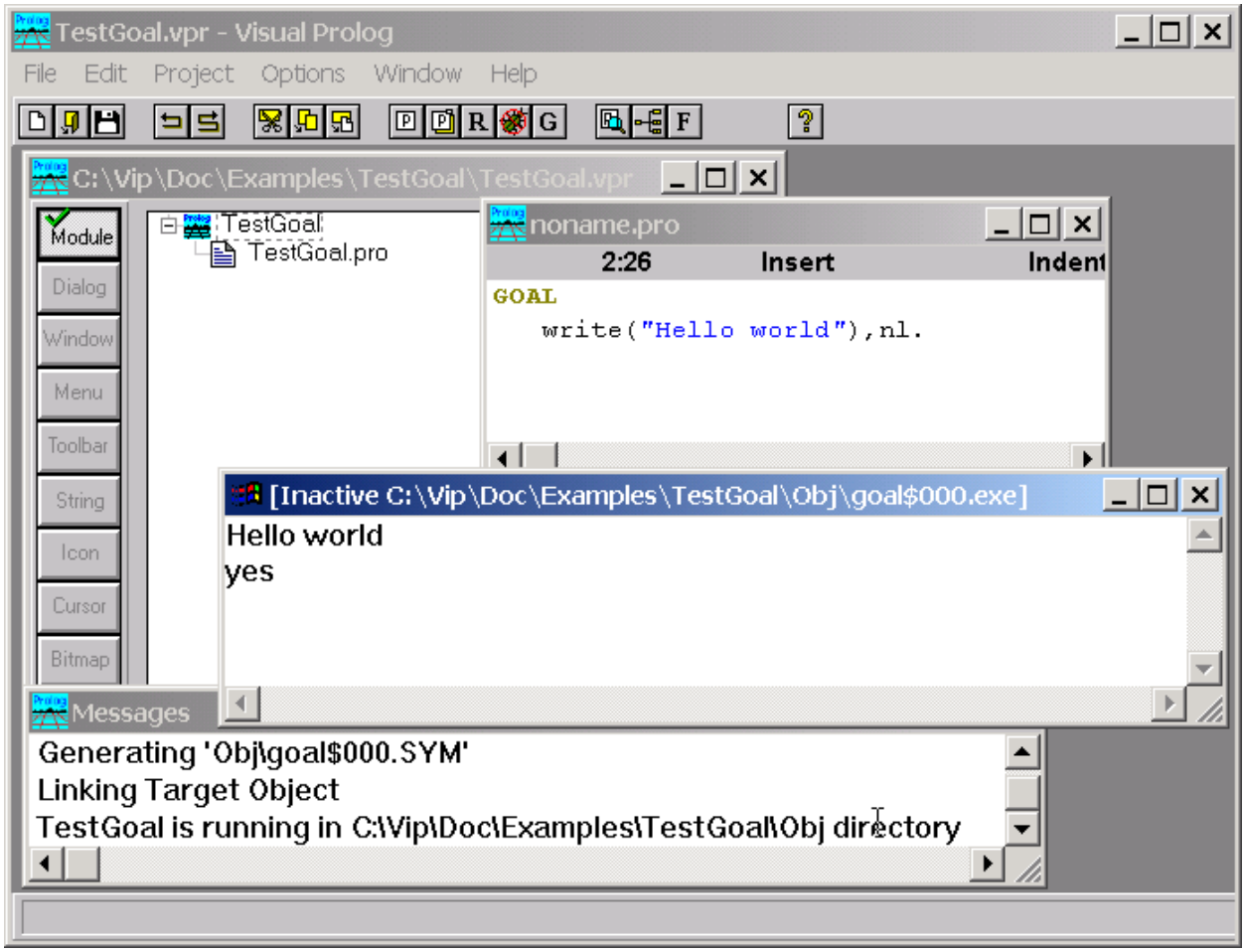


Figure 1.5: The "Hello world" test

The result of the execution will come up in a separate window (on this figure it has title: `[Inactive C:\Vip\Doc\Examples\TestGoal\Obj\goal$000.exe]`), which you must close before you can test another GOAL.

Handling Errors

If you, like all programmers do, happen to make some errors in your program and try to compile the program, then the VDE will display the **Errors (Warnings)** window, which contains the list of detected errors. You can double click on one of these errors to come to the position of the error in the source text. (Beginning with Visual Prolog v. 5.3, you can click on one of these errors and press the hot key **F1** to display extended information about the selected error.) In previous versions, you can press **F1** to display Visual Prolog's on-line Help. When in the Help window, click the **Search** button and type in an error number; the help topic with extended information about the error will be displayed.

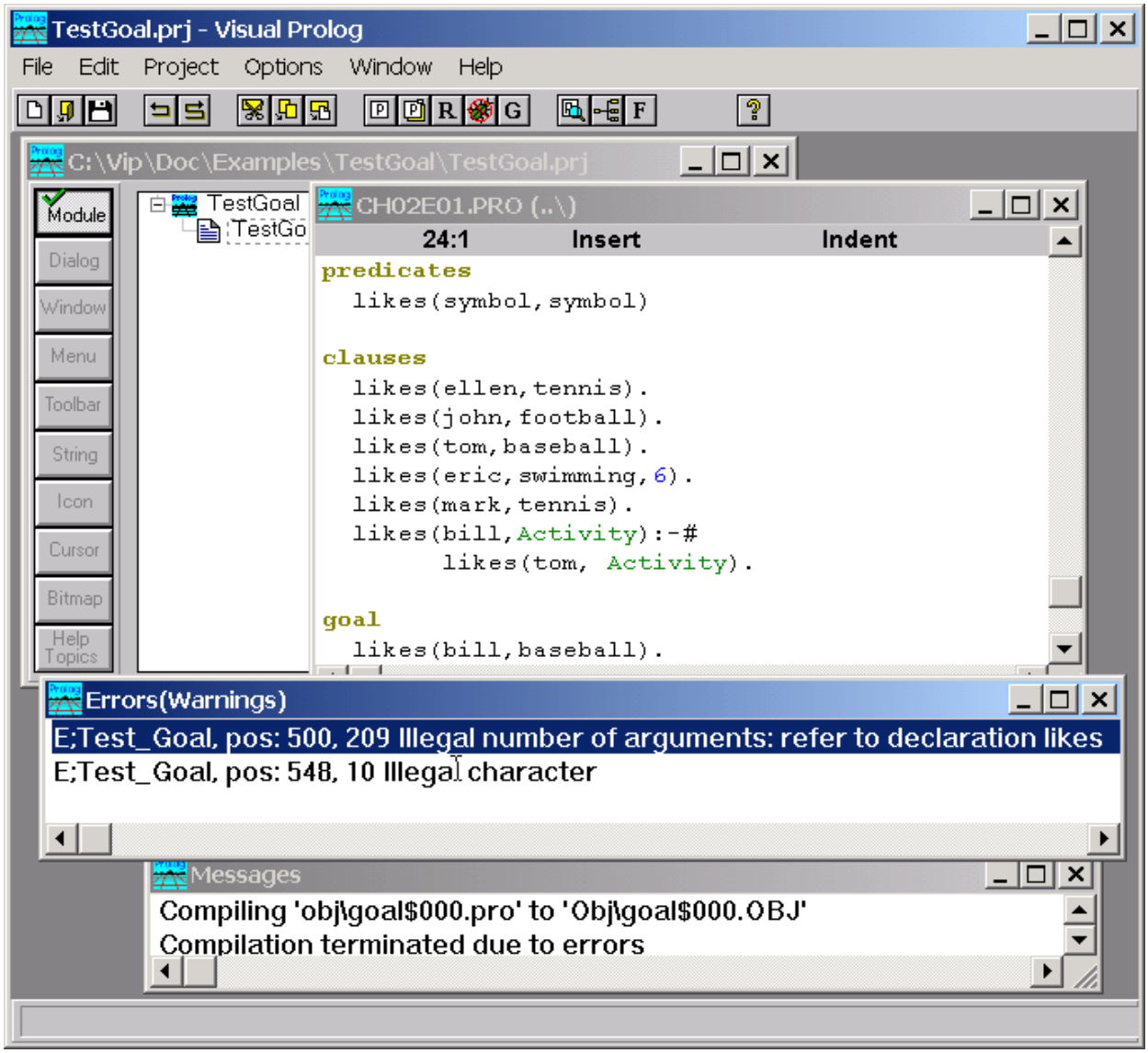


Figure 1.6: Handling Errors

LAB 1: Basic Prolog Program structure

Visual Prolog's Basic Program Sections

Generally, a Visual Prolog program includes four basic program sections. These are the **clauses** section, the **predicates** section, the **domains** section, and the **goal** section.

• The **clauses** section is the heart of a Visual Prolog program; this is where you put the facts and rules that Visual Prolog will operate on when trying to satisfy the program's goal.

• The **predicates** section is where you declare your predicates and the domains (types) of the arguments to your predicates. (You don't need to declare Visual Prolog's built-in predicates.)

• The **domains** section is where you declare any domains you're using that aren't Visual Prolog's standard domains. (You don't need to declare standard domains.)

The **goal** section is where you put the starting goal for a Visual Prolog program.

The Clauses Section

The **clauses** section is where you put all the facts and rules that make up your program. Most of the discussion is centered around the clauses (facts and rules) in your programs; what they convey, how to write them, and so on.
If you understand what facts and rules are and how to write them in Prolog, you know what goes in the **clauses** section. Clauses for a given predicate must be placed together in the **clauses** section; a sequence of clauses defining a predicate is called a *procedure*.

When attempting to satisfy a goal, Visual Prolog will start at the top of the **clauses** section, looking at each fact and rule as it searches for a match. As Visual Prolog proceeds down through the **clauses** section, it places internal pointers next to each clause that matches the current sub-goal. If that clause is not part of a logical path that leads to a solution, Visual Prolog returns to the set pointer and looks for another match

The Predicates Section

If you define your own predicate in the **clauses** section of a Visual Prolog program, you *must* declare it in a **predicates** section, or Visual Prolog won't know what you're talking about. When you declare a predicate, you tell Visual Prolog which domains the arguments of that predicate belong to.
Visual Prolog comes with a wealth of built-in predicates. You don't need to declare any of Visual Prolog's built-in predicates that you use in your program. The *Visual Prolog* on-line help gives a full explanation of the built-in predicates.
Facts and rules define predicates. The **predicates** section of the program simply lists each predicate, showing the types (domains) of its arguments. Although the **clauses** section is the heart of your program, Visual Prolog gets much of its efficiency from the fact that you also declare the types of objects (arguments) that your facts and rules refer to.

The Domains Section

In traditional Prolog there is only one type - **the term**. I have the same in Visual Prolog, but I are declaring what the domains of the arguments to the predicates actually are.

Domains enable you to give distinctive names to different kinds of data that would otherwise look alike. In a Visual Prolog program, objects in a relation (the arguments to a predicate) belong to domains; these can be pre-defined domains, or special domains that you specify.

The Goal Section

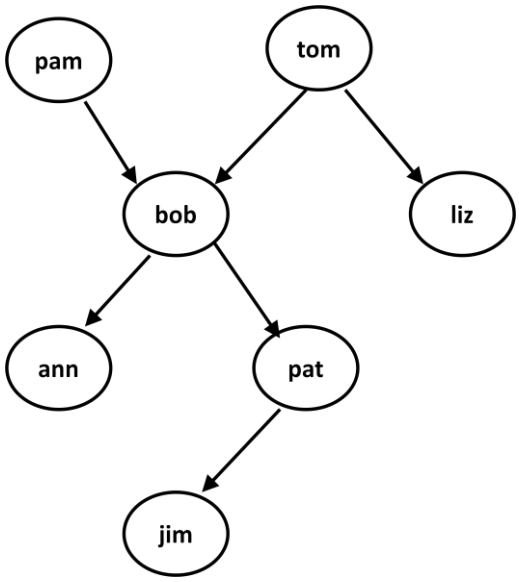
Essentially, the **goal** section is the same as the body of a rule: it's simply a list of sub-goals. There are two differences between the **goal** section and a rule:

1. The **goal** keyword is *not* followed by :-.
2. Visual Prolog automatically executes the goal when the program runs.

It's as if Visual Prolog makes a call to **goal**, and the program runs, trying to satisfy the body of the goal rule. If the sub-goals in the **goal** section all succeed, then the program terminates successfully. If, while the program is running, a sub-goal in the **goal** section fails, then the program is said to have failed. (Although, from an external point of view, there isn't necessarily any difference; the program simply terminates.)

Problem Statement

Implement the given family tree in the form of Prolog Clauses. Every arrow in the diagram represents a Parent relation. For example pam is parent of bob, tom is parent of bob and so on.



Solution:

%In clauses section we shall encode this knowledge.

CLAUSES

```
parent( pam, bob). % Pam is a parent of Bob

parent( tom, bob).

parent( tom, liz).

parent( bob, ann).

parent( bob, pat).

parent( pat, jim).
```

LAB 2: Querying the Prolog

Problem Statement:

On the basis of family tree implemented in last Lab, write Prolog statement to query Prolog engine for following questions;

- Is Bob a parent of Pat?
- Who is Liz's parent?
- Who are Bob's children?
- Who is a parent of whom?
- Is bob grandparent of Jim?
- Who is a grandparent of ann?
- Who are Tom's grandchildren?
- Do Ann and Pat have a common parent?
-

Solution:

- **Is Bob a parent of Pat?**
    - **?- parent(bob, pat).**
- **Who is Liz's parent?**
    - **?- parent(X, liz).**
- **Who are Bob's children?**
    - **?- parent(bob, X).**

- **Who is a parent of whom?**
  - **Find X and Y such that X is a parent of Y.**
  - **?- parent(X, Y).**
- **Is bob grandparent of Jim?**
  - **?- parent(pat, jim), parent(bob, pat).**
- **Who is a grandparent of ann?**
  - **?- parent(Y, ann), parent(X, Y).**
- **Who are Tom's grandchildren?**
  - **?- parent(tom, X), parent(X, Y).**
- **Do Ann and Pat have a common parent?**
  - **?- parent(X, ann), parent(X, pat).**

LAB 3: Defining new relations using existing facts

- It is easy in Prolog to define a relation.
- The user can query the Prolog system about relations defined in the program.
- A Prolog program consists of clauses. Each clause terminates with a full stop.
- The arguments of relations can be
  - Atoms: concrete objects or constants
  - Variables: general objects such as X and Y
- Questions to the system consist of one or more goals.
- An answer to a question can be either positive (succeeded) or negative (failed).
- If several answers satisfy the question then Prolog will find as many of them as desired by the user.

Problem Statement:

Extend the Family tree program implemented in previous labs by adding the gender information given as follow; Pam, Liz, Ann and Pat are female, while Tom, Bob and Jim are male persons.

Using this information define the following relations;

- Define the “mother” relation:
- Define the “grandparent” relation:
- Define the “sister” relation

Also depict the given relations with the help of diagrams.

Solution:

Facts:

```
female( pam).     % Pam is female
female( liz).
```

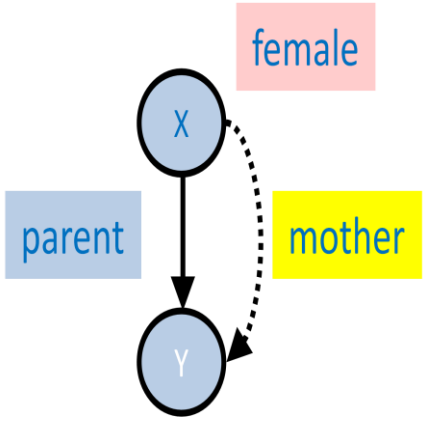
```
female( ann).
female( pat).
male( tom).          % Tom is male
male( bob).
male( jim).
```

Define the "offspring" relation:

Fact: offspring(liz, tom).

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

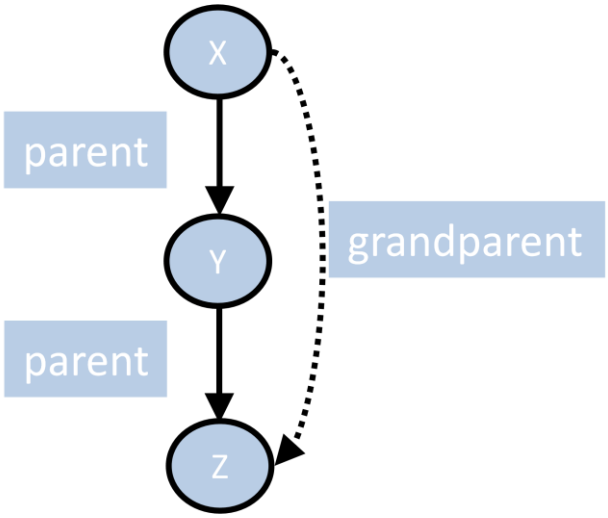
- For all X and Y,
  Y is an offspring of X if
  X is a parent of Y.



- "mother" relation:

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

– For all X and Y,
  X is the mother of Y if
  X is a parent of Y and
  X is a female.



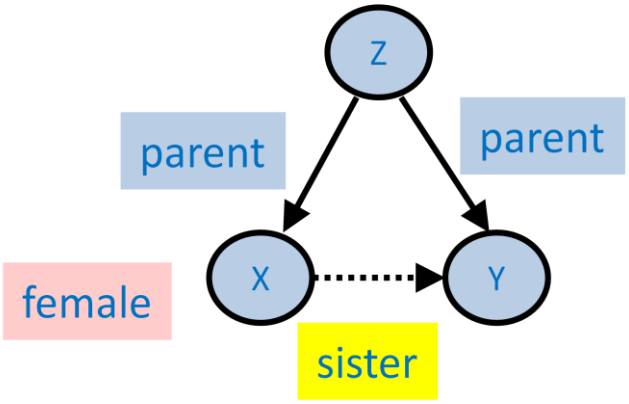
- "grandparent" relation:

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

- "sister" relation:

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

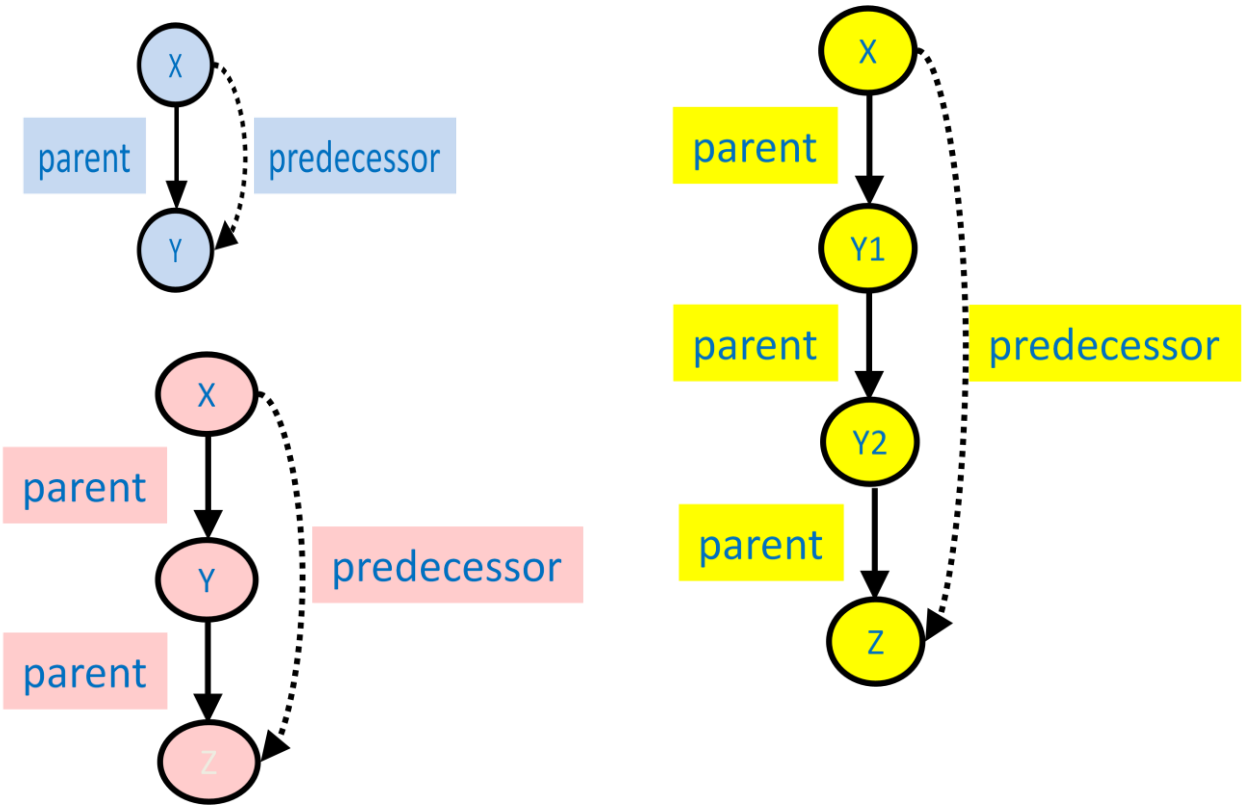
  - different (X, Y) is satisfied if and only if X and Y are not equal.



LAB 4: Defining Recursive Relations

Problem Statement:

Define the "predecessor(X, Y)" relation. It should be recursive to satisfy all the conditions depicted below;



Solution:

- Defining the "predecessor" relation

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

– For all X and Z,

X is a predecessor of Z if X is parent of Z

OR there is a Y such that

(1) X is a parent of Y and

(2) Y is a predecessor of Z.

LAB 5: How Prolog Answers Questions

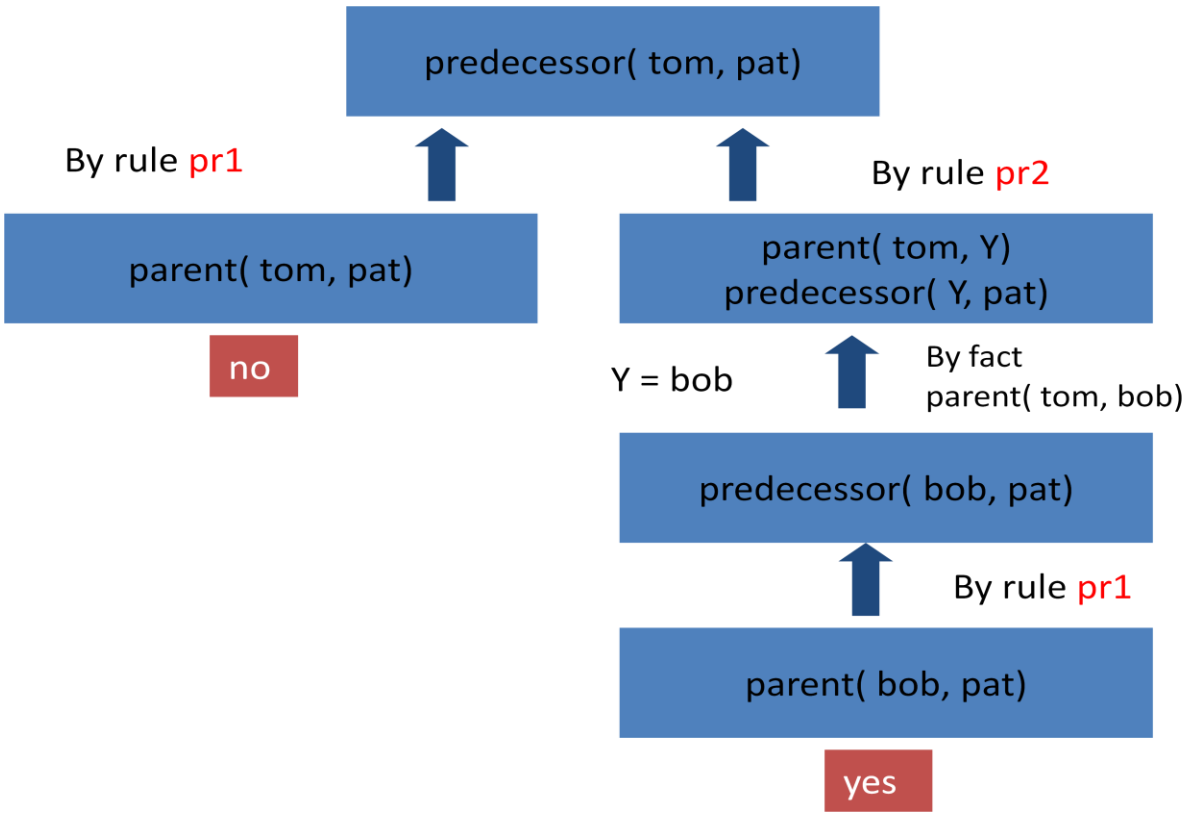
Problem Statement:

By using Prolog clauses given below, trace the recursive predecessor relation in order to demonstrate how Prolog engine answers queries.

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

```
predecessor( X, Z)  :- parent( X, Z).          % Rule pr1
predecessor( X, Z)  :- parent( X, Y),          % Rule pr2
                       predecessor( Y, Z).
```

Solution:



LAB 6: Encoding knowledge in Prolog

Problem Statement:

Encode following knowledge in Prolog and answer the given queries;

- Asif likes to play tennis, while Umer and Fahad play football. Kashif regularly plays hockey. Tariq, who is friend of Asif plays every sports which Asif plays. Moreover, all of these persons also like to play cricket.
- Queries
- Who plays tennis ?
- Which sports Tariq plays ?
- Which game is played by which person ?

Solution:

```
CLAUSES

play(asif, tennis).

play(umer, footbal).

play(fahad, footbal).

play(kashif, hockey).

Play(tariq, Sport):-

     play(asif, Sport).

Play(Person, criket):-

     play(Person, Sport).
```

%answering queries

GOAL

```
% Who plays tennis ?
play(Who, tennis),
% Which sports Tariq plays ?
play(tariq, Sports),
% Which game is played by which person ?
play (Person, Game).
```

LAB 7: Building a phone directory and querying it

Problem Statement:

- **Create knowledge base in Prolog for data given below;**

Person Name	City Code	Phone no
Kashif	051	544525
Asif	042	124536
Fahad	051	879546
Tariq	021	112223
Noman	051	555555
Jamil	042	665655
Aslam	051	481526

City Name	Code
Islamabad	051
Lahore	042
Karachi	021

Write Prolog goal statements to answer the following questions on the basis of tabular knowledge given above;

- What is phone number of Kashif ?
- Does Asif belongs to Lahore ?

- Where does Noman lives ?
- Does Fahad and Tariq lives in same city ?
- Who lives in same city where Kashif lives ?
- Relations
- Belongs_To (Person, City)
- Same_City (Person, Person)

Solution:

```
clauses

phone_no (kashif , 051, 544525).

phone_no (asif, 042, 124536).

phone_no (fahad, 051, 879546).

phone_no (tariq, 021, 112223).

phone_no (noman, 051, 555555).

phone_no (jamil, 042, 665655).

phone_no (aslam, 051, 481526).

city_code(islamabad, 051).

city_code(lahore, 042).

city_code(karachi, 021).

belongsto(Person, City):-

     phone_no(Person, D, N),

     city_code(City, D).

same_city(P1, P2):-

    phone_no(P1, D, N),

    phone_no(P2, D, M),

    P1 <> P2.
```

LAB 8: Output to user

Problem Statement:

Define can_swim relation for the animals which live in water. Use output statement in goal portion to display results to the user.

Solution:

```
predicates
  type(symbol,symbol)
  is_a(symbol,symbol)
  lives(symbol,symbol)
  can_swim(symbol)
clauses
  type(ungulate,animal).
  type(fish,animal).
  is_a(zebra,ungulate).
  is_a(herring,fish).
  is_a(shark,fish).
  lives(zebra,on_land).
  lives(frog,on_land).
  lives(frog,in_water).
  lives(shark,in_water).
  can_swim(Y):-
       type(X, animal),
```

```
      is_a(Y,X),
      lives(Y, in_water).
goal
  can_swim(What),
  write("A ",What," can swim\n").
```

LAB 9: Input from User

Following functions can be used to input data from user at runtime.

- readchar(X).
- readinteger(X).
- readln(X).

Problem Statement:

Define reference relation to store phone numbers of four persons (use dummy data). And in goal portion write input statements to input query data from user at runtime.

Solution:

```
predicates
        reference(symbol,symbol)
goal
        write("Please type a name :"),
        readln(The_Name),
        reference(The_Name,Phone_No),
        write("The phone number is ",Phone_No).
clauses
        reference("Albert", "01-123456").
        reference("Betty",  "01-569767").
        reference("Carol",  "01-267400").
        reference("Dorothy","01-191051").
```

LAB 10: Input and Processing Integer

Problem Statement:

Write a Prolog program to demonstrate how Prolog engine can input numbers and process those numbers. As an example you may define simple addition and multiplication procedures in Prolog.

Solution:

```
domains
  product,sum = integer


predicates
  add_em_up(sum,sum,sum).
  multiply_em(product,product,product).
  nums.
clauses
  add_em_up(X,Y,Sum):-
   Sum=X+Y.
  multiply_em(X,Y,Product):-
   Product=X*Y.
  nums:-
  readint(X),
  readint(Y),
  add_em_up(X,Y,Sum),
  write(Sum),nl.
```

```
goal


  %nums.
  readint(X),
  readint(Y),
  add_em_up(X,Y,Ans),
  write(Ans),nl.


%add_em_up(2,3,Sum),
  %multiply_em(3,4, M).
```

LAB 11: Processing Numbers: Calculating Factorial

Problem Statement:

Write Prolog procedure to calculate factorial of given number.

Solution:

```
domains
  num= integer
predicates
  fact1(num, num).
clauses
  fact1(0,1).
  fact1(N,Result) :-
    N > 0,
    N1 = N-1,
    fact1(N1,Result1),
    Result = Result1*N.
goal
  %fact1(4, A).
  readint(X),
  fact1(X,Ans),
  write(X, " factorial is ", Ans),nl.
```

LAB 12: Finding GCD of two numbers

Problem Statement:

How Prolog program can be used to find Greatest Common Divisor of two integers.

Solution:

```
1. predicates
2.    gcd(integer, integer, integer).
3. clauses
4.      gcd( X, X, X).
5.      gcd( X, Y, D) :-
6.           X<Y,
7.           Y1 = Y-X,
8.           gcd( X, Y1, D).
9.      gcd( X, Y, D) :-
10.               Y<X,
11.               gcd( Y, X, D).
12.     goal
13.        gcd(39,26,D).
```

LAB 13: Working with Lists: Printing all elements of a list

- In Prolog, a *list* is an object that contains an arbitrary number of other objects within it.
- [dog, cat, canary]
- ["valerie ann", "jennifer caitlin", "benjamin thomas"]
- Heads and Tails: A list is really a recursive compound object. It consists of two parts: the head, of list which is the first element, and the tail, which is a list comprising all the subsequent elements. *The tail of a list is always a list; the head of a list is an element.*

Problem Statement:

Define a simple list of integer in Prolog and demonstrate how Prolog can display all the numbers stored in the list.

Solution:

```
1. DOMAINS
2. list = integer* /* or whatever type you wish to use */
3. PREDICATES
4. write_a_list(list)
5. CLAUSES
6. write_a_list([]). /* If the list is empty, do nothing more. */
7. write_a_list([H|T]):-
8. /* Match the head to H and the tail to T, then... */
9. write(H),nl,
10.     write_a_list(T).
11.     GOAL
12.     write_a_list([1, 2, 3]).
```

LAB 14: List Membership

Problem Statement:

Define Prolog procedure to check the membership of given item from the given list.

Solution:

```
1. DOMAINS
2. namelist = name*
3. name = symbol
4. PREDICATES
5. member(name, namelist)
6. CLAUSES
7. member(Name, [Name|_]).
8. member(Name, [_|Tail]):-
9. member(Name,Tail).
10.     GOAL
11.     member(susan, [ian, susan, john]).
```

LAB 15: Substituting Item from list

Problem Statement:

Write a generic 'substitute' function in Prolog, which should allow replacement of an old number with new number from a list of given number. Demonstrate this functionality on list of numbers.

Solution:

```
1. DOMAINS
2. list = integer*
3. PREDICATES
4.      substitute(integer,integer, list, list)
5. CLAUSES
6.      substitute(X, N, [], []).
7.      substitute(X, N, [H|T], [H|Y]):-
8.           X <> H,
9.           substitute(X, N, T, Y).
10.
11.          substitute(X, N, [X|T],[N|Y]):-
12.               substitute(X, N, T, Y).
13.     GOAL
14.          substitute(1, 10, [1,2,1,3,1,4],X).
```

LAB 16: Finding Maximum of from a List

Problem Statement:

How Prolog can be used to find Maximum number from a list of numbers. Define max relation in Prolog.

Solution:

```
1. DOMAINS
2. list = integer*
3. PREDICATES
4.      max(integer, list, integer)
5. CLAUSES
6.      max(M, [], M).
7.      max(M, [H|T],X):-
8.           X >= H,
9.           max(M, T, X).
10.          max(M, [H|T],X):-
11.               H > X,
12.               max(M, T, H).
13.
14.     GOAL
15.     max (M, [10,2,10,30,1,4],0).
```

Prolog LAB Manual Using Visual Prolog 5.0

This is first edition of Prolog Lab Manual is based on contents covered in the course of “Programming Techniques using Prolog” taught during Spring 2016 semester by Dr. Umair Abdullah at SZABIST. This Lab manual presents practical formulation of Prolog topics in very simple way.