# Advanced Prolog Features

## Extending Prolog: Arithmetic

- Although Prolog allows the standard arithmetic operators (such as + - * and /) to be used in arithmetic expressions, there is no similar convenient notation for calculating factorials or to perform other less common but sometimes useful operations such as adding the squares of two numbers.
- The built-in predicate **is/2** is used for evaluating arithmetic expressions. It is not permitted for the Prolog programmer to redefine this by adding new operators (or by any other means) and any attempt to do so would lead to a system error.



## Extending Prolog: Arithmetic…

- There are techniques that can be used by the programmer to get the same effect, so that expressions involving new arithmetic operators such as 6! are permitted or even so that the definitions of standard operators such as + and - are changed.
- The key to this is to define a replacement for the **is/2** operator. This new operator will be called **iss/2**. There are two steps involved:



## Extending Prolog: Arithmetic…

*Step 1: Define **iss/2** to be an operator*

- Enter the goal:

  **?- op(710,xfx,iss).**

- The above goal should be entered at the system prompt or as a directive at the beginning of a program.
- The third argument **(iss)** of the **op/3** goal is the name of the operator.
- The first argument (710) is the *precedence* of the iss/2 operator.
- The second argument **(xfx)** denotes that **iss** is an infix operator, which takes two arguments and will be written between them.

## Extending Prolog: Arithmetic…

*Step 2: Define the **iss/2** operator*

- The simplest definition of **iss/2** would be the single line:

  X iss Y:-X is Y.

- This would simply make the operator **iss** equivalent to the built-in operator **is**
- Example:
  - **?- Z iss 6+sqrt(25)-2.**
  - **Z = 9**



## Extending Prolog: Arithmetic…

- An improved attempt at defining **iss** is as follows:
  - The effect of using **iss** in combination with the different types of arithmetic operator (+ - * / etc.) is specified explicitly in the first eight clauses.

```
Y iss A+B:-Y is A+B,!.
Y iss A-B:-Y is A-B,!.
Y iss A*B:-Y is A*B,!.
Y iss A/B:-Y is A/B,!.
Y iss A//B:-Y is A//B,
Y iss A^B:-Y is A^B,!.
Y iss +A:-Y is A,!.
Y iss -A:- Y is -A,!.
Y iss X:- Y is X,!.
```



## Extending Prolog: Arithmetic…

- All other cases (e.g. expressions involving **sqrt**, **sin** etc.) are dealt with by the final clause. With these definitions the operator **iss** still has the same effect as **is**.
- **Examples:**
  - **?- Y iss 6+4*3-2.**

    **Y = 16**
  - **?- X iss 3,Y iss X+5.6-3*10+100.5.**

    **X = 3 ,**

    **Y = 79.1**



## Extending Prolog: Arithmetic…

- **Examples:**
  - **?-Y iss (8+4)/3-(6*7).**

    **Y = -38**
  - **?- A=3,B=4,Y iss sqrt(A*A+B*B).**

    **A = 3 ,**

    **B = 4 ,**

    **Y = 5**
  - **?-Y iss 6+sqrt(25).**

    **Y = 11**
  - **?-Y iss 6+sqrt(10+15).**

    **Y = 11**

## Defining ! as a Factorial Operator

- Starting from this more elaborate definition of **iss**, we can now add further operators.
- The mathematical function *factorial* is defined only for integer arguments.
- The value of 'factorial 6' is 6 x 5 x 4 x 3 x 2 x 1 and is written as 6!
- In general the value of N! is *Nx(N-1)!*. This leads to a two-line recursive definition of a predicate **factorial/2**:

```
factorial(1,1):-!.
factorial(N,Y):-N1 is N-1,factorial(N1,Y1),
   Y is N*Y1.
```



## Defining ! as a Factorial Operator…

- It is assumed that the first argument will always be an integer or a variable bound to an integer and the second argument is an unbound variable.
- Then, for example, the product 6 x 5 x 4 x 3 x 2 x 1 can be found by
  - **?- factorial(6,Y).**

    **Y = 720**



## Defining ! as a Factorial Operator…

- This predicate can now be used to define a new arithmetic operator **!** which will enable terms such as 6! or N! to be written when evaluating an arithmetic expression using the **iss** predicate
- As usual, there are two actions required to do this:



## Defining ! as a Factorial Operator…

*Step 1: Define ! to be an operator*

- This can be done by entering the goal

  **?- op(200,xf,!).**

- The atom **xf** denotes that **!** is a postfix operator, which will appear after its argument, e.g. 6!. Its precedence is 200.

## Defining ! as a Factorial Operator…

*Step 2: Define the ! Predicate*

- Using the definition of the factorial predicate already given, the **!** operator (or rather its effect when used together with the **iss** operator) can be defined by adding the following clause to the definition of **iss**, say as the first line.

```
Y iss N!:-N1 iss N,factorial(N1,Y),!.
```



## Defining ! as a Factorial Operator…

- This allows the exclamation mark character to be used in a convenient way to represent factorials.
  - **?-Y iss 6!.**

    **Y = 720**
  - **?-Y iss (3+2)!.**

    **Y = 120**
- However, there is a flaw in the definition of **iss**. Entering a goal such:

  **?-Y iss 5!+6!.**
- will cause Prolog to crash with an error message such as 'Function Not Defined'.



## Defining ! as a Factorial Operator…

- The reason is that to evaluate this expression, Prolog makes use of the definition of the + operator, which is:

  **Y iss A+B:-Y is A+B,!.**
- This causes it to try to evaluate the goal:

  **Y is 5!+6!**
- which causes an error as in this context **5!** and **6!** are not numbers. They have no meaning at all outside their definition for the **iss** predicate.



## Defining ! as a Factorial Operator…

- The most satisfactory way of dealing with this problem is to modify the definition of the **iss** operator so that its arguments are themselves evaluated using **iss** before adding, multiplying etc. their values
- This requires every clause in the definition of **iss/2** to be modified, expect for the last, and gives the following revised program:

## Defining ! as a Factorial Operator…

```
?- op(710,xfx,iss).
?- op(200,xf,!).
factorial(1,1):-!.
factorial(N,Y):-N1 is N-1,factorial(N1,Y1),Y is N*Y1.
Y iss N!:-N1 iss N,factorial(N1,Y),!.
Y iss A+B :-A1 iss A,B1 iss B,Y is A1+B1,!.
Y iss A-B :-A1 iss A,B1 iss B,Y is A1-B1,!.
Y iss A*B :-A1 iss A,B1 iss B,Y is A1*B1,!.
Y iss A/B :-A1 iss A,B1 iss B,Y is A1/B1,!.
Y iss A//B :-A1 iss A,B1 iss B,Y is A1//B1,!.
Y iss A^B :-A1 iss A,B1 iss B,Y is A1^B1,!.
Y iss +A :-Y iss A,!.
Y iss -A :- A1 iss A,Y is -A1,!.
Y iss X :- Y is X,!.
```



## Defining ! as a Factorial Operator…

- With the new definition of the + operator, if either of its arguments is an expression such as 5! it is converted to a number before it is used
- If an argument is a number, it is 'converted' to itself by the final clause
- The **!** operator now works as expected
- When the goal **Y iss 5!+6!** is evaluated, the system first applies **iss** to **5!** and to **6!** producing the numbers 120 and 720, respectively, and then adds them together.



## Defining ! as a Factorial Operator…

- **Examples:**
  - **Y iss 6!.**

    **Y = 720**
  - **Y iss (3+2)!.**

    **Y = 120**
  - **?- Y iss 5!+6!.**

    **Y = 840**



## Defining ! as a Factorial Operator…

- **?- Y iss 4+2,Z iss Y!+3!-4!.**

  **Y = 6 ,**

  **Z = 702**
- **?- Y iss (3!)!.**

  **Y = 720**
- **?- Y iss -(3!).**

  **Y = -6**

## Defining ! as a Factorial Operator…

- Note that the above definition of **iss** is still not watertight. Expressions such as **sqrt(3!)** will cause the system to crash.
- This can be overcome by adding additional clauses such as:

  **Y iss sqrt(A):-A1 iss A,Y is sqrt(A1),!.**
- for all the arithmetic functions, such as **sqrt**, **tan** and **sin** with which it is intended to use the new operator.



## Defining ** as a Sum of Squares Operator

- As well as factorial, we can define new operators that perform any operations we wish.
- For example we might want to have an infix operator ****** that returns the sum of the squares of its two arguments. This can be defined as follows.



## Defining ** as a Sum of Squares Operator…

*Step 1: Define ** to be an operator*

- This can be done by entering the goal

  **?- op(200,yfx,**).**
- This specifies that ** is an infix operator, which will appear between its two arguments, e.g. 3**4. Its precedence is 200.



## Defining ** as a Sum of Squares Operator…

*Step 2: Define the ** operator*

- The ** operator can be defined by adding the following clause to the definition of **iss**, anywhere except the final line (which is a 'catch all').

  **Y iss A**B:- A1 iss A, B1 iss B,Y is A1*A1+B1*B1,!.**

## Defining ** as a Sum of Squares Operator…

- **Examples:**
  - **?- Y iss 3**2.**

    **Y = 13**
  - **?- Y iss (3**2)+2.**

    **Y = 15**
  - **?- Y iss 6+3**4+8+1**2-10.**

    **Y = 34**
  - **?- Y iss (3**1)**(2**1).**

    **Y = 125**



## Redefining Addition and Subtraction

- Now that we have the **iss** predicate we can even use it to 'redefine' addition and subtraction if we wish.
- If we change the following clauses in the definition of **iss:**

```
Y iss A+B:- A1 iss A,B1 iss B,Y is A1+B1,!.
Y iss A-B:- A1 iss A,B1 iss B,Y is A1-B1,!.
```

**to**

```
Y iss A+B:- A1 iss A,B1 iss B,Y is A1-B1,!.
Y iss A-B:- A1 iss A,B1 iss B,Y is A1+B1,!.
```



## Redefining Addition and Subtraction…

- The effect will be to cause + to subtract and - to add
- Examples:
  - **?- Y iss 6+4.**

    **Y = 2**
  - **?- Y iss 6-4.**

    **Y = 10**



## END