LOGIC PROGRAMMING

Lecture#4

Arithmetic and Logical Operators

1. Arithmetic

- The most important one is the built-in predicate **is/2**, which is predefined as an infix operator and thus is written between its two arguments.

Examples:

```
?- X is -3.
X = -3.
?- X is 10.5+4.7*2.
X = 19.9.
?- Y is 10, Z is Y+1.
Y = 10,
Z = 11.
```

1. Arithmetic (cont.)

Operation	Meaning
X+Y	the sum of X and Y
X-Y	the difference of X and Y
X*Y	the product of X and Y
X/Y	the quotient of X and Y
X//Y	the 'integer quotient' of X and Y
X^Y	X to the power of Y
-X	the negative of X
abs(X)	the absolute value of X
sin(X)	the sine of X (for X measured in degrees)
cos(X)	the cosine of X (for X measured in degrees)
max(X,Y)	the larger of X and Y
min(X,Y)	the smaller of X and Y
sqrt(X)	the square root of X

1. Arithmetic (cont.)

Examples:

```
?- X is 30,Y is 5,Z is X+Y+X*Y+sin(X).
X = 30 ,
Y = 5 ,
Z = 185.5.
?- 10 is 7+13-11+9.
false.
?- 18 is 7+13-11+9.
true.
```

- Note that a goal such as X is X+1 will always fail, whether or not X is bound.

```
?- X is 10,X is X+1.
false.
```

1. Arithmetic (cont.)

To increase a value by one requires a different approach.

```
/*Correct version */
increase(N,M):-M is N+1.
```

?- increase(4,X).
X = 5

Operator Precedence in Arithmetic Expressions:

Refer to Page 62 for details.

1. Arithmetic (cont.)

Relational Operators:

- The infix operators =:= =\= > >= < =< are a special type known as *relational operators*.
- ***Examples***:

```
?- 4 > 2.
true.
?- 99 >= 109.
false.
?- 88+15-3 =:= 110-5*2.
true.
?- 100 =\= 99.
true.
```

2. Equality Operators

- There are three types of relational operator for testing equality and inequality available in Prolog.
- The first type is used to compare the values of arithmetic expressions. The other two types are used to compare terms.

1.a. Arithmetic Expression Equality =:=

E1=:=E2 succeeds if the arithmetic expressions E1 and E2 evaluate to the same value.

```
?- sqrt(36)+4=:=5*11-45.
```

true.

To check whether an integer is odd or even we can use the user-defined **checkeven/1** predicate defined below.

2. Equality Operators (cont.)


```
checkeven(N) :- M is N//2 , N =:= 2*M.
```

Goals:

?- checkeven(9).

false.

?- checkeven(8).

true.

1.b. Arithmetic Expression Inequality =\=

- E1=\=E2 succeeds if the arithmetic expressions E1 and E2 do not evaluate to the same value.

Example:

?- 10 =\= 8+3.

true.

2. Equality Operators (cont.)

2.a. Terms Identical ==

- Both arguments of the infix operator == must be terms.
- The goal Term1==Term2 succeeds if and only if Term1 is identical to Term2.
- Any variables used in the terms may or may not already be bound, but no variables are bound as a result of evaluating the goal.
- **Examples:**

```
?- likes(X,prolog) == likes(X,prolog).
true.
?- likes(X,prolog) == likes(Y,prolog).
false.
?- 6+4 == 7+3.
false.
?- D == 3.
false.
```

2. Equality Operators (cont.)

2.b. Terms Not Identical \==

- Term1 \==Term2 tests whether Term1 is not identical to Term2. The goal succeeds if Term1==Term2 fails. Otherwise it fails.
- ***Example***:

  ?- p(X) \== p(Y).

  true.

2. Equality Operators (cont.)

3.a. Terms Identical With Unification =

- The term equality operator = is similar to == with one vital (and often very useful) difference.
- The goal *Term1=Term2 succeeds if terms Term1 and Term2 unify, i.e.* there is some way of binding variables to values which would make the terms identical. If the goal succeeds, such binding actually takes place.
- ***Examples:***

```
?- p(X) = p(10).
X = 10.
?- likes(X,prolog) = likes(john,Y).
X = john,
Y = prolog.
?- likes(X,prolog)=likes(Y,ada).
false.
?- likes(X,prolog)=likes(Y,prolog).
X = Y.
```

2. Equality Operators (cont.)


```
?- R = 0, R =:= 0.
R = 0.

?- R = 0, R =:= 2.
false.

?- 6+4 = 6+2.
false.

?- 2+W = 2 + 3.
W = 3.
```

2. Equality Operators (cont.)

3.b. Non-Unification Between Two Terms \=

- The goal Term1 \=Term2 succeeds if Term1=Term2 fails, i.e. the two terms cannot be unified. Otherwise it fails.
- **Examples**:

```
?-  6+4 \= 3+7.
true.
?- likes(X,prolog) \= likes(john,Y).
false.
?- likes(X,prolog) \= likes(X,ada).
true.
```

3. Logical Operators

The \+ Operator:

- It is used to give negation.
- The negated goal succeeds if the original goal fails and fails if the original goal succeeds.

Examples: refer to the animal program in Page 14 & 15:

```
?- \+dog(fido).
false.

?- dog(fred).
true.

?- \+dog(fred).
false.
```

3. Logical Operators (cont.)

The \+ Operator:

- **Example**:

```
large_animal(X):- \+ dog(X) , large(X).
```

Goals:

```
?- large_animal(fido).
false.
?- large_animal(mary).
true.
```

3. Logical Operators (cont.)

The Disjunction Operator (;):

- The disjunction operator ;/2 (written as a semicolon character) is used to represent 'or'.
- **Examples:**

```
?- 6<3;7 is 5+2.
true.

?- 6*6=:=36;10=8+3.
true.
```