FIU FLORIDA INTERNATIONAL UNIVERSITY

Logic Programming and PROLOG (IV)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences

PROLOG. Arithmetic

- **PROLOG** is mainly a language for **symbolic computation** where the need for numerical calculation is comparatively modest.
- Some of the predefined operators can be used for basic arithmetic operations.
- These are:

+	addition
-	subtraction
*	multiplication
/	division
div	integer division
mod	modulo, the reminder of integer division

PROLOG. Arithmetic (cont...)

- The following question is a naive attempt to request arithmetic computation:

  ?- X = 1 + 2.

  PROLOG will "quietly" answer

  X = 1 + 2

- and **not X = 3**. The reason is simple: the expression 1 + 2 merely denotes a PROLOG term where + is the functor and 1 and 2 are its arguments.
- There is nothing in the above goal to force PROLOG to actually activate the addition operation.
- A special predefined operator, **is**, is provided to solve this problem. The is operator will force evaluation.

  ?- X is 1 + 2.

  X = 3

3

PROLOG. Arithmetic (cont...)

- **Examples:**

  ?- X is 3/2, Y is 3 div 2.

  X = 1.5 % real division

  Y = 1 % integer division

- The **precedence of the predefined arithmetic operators** is such that the associativity of arguments with operators is the same as normally in mathematics.
- Parentheses can be used to indicate different associations.
- Arithmetic is also involved when comparing numerical values.
- We can, for example, test whether the product of 277 and 37 is greater than 10000 by the query:

  ?- 277*37 > 10000.

  true

- Note that, similarly to **is**, the "**>**" operator also forces the evaluation.

4

PROLOG. Arithmetic (cont...)

- The **comparison operators in PROLOG** are:

X > Y	X is greater than Y
X < Y	X is less than Y
X >= Y	X is greater than or equal to Y
X =< Y	X is less than or equal to Y
X =:= Y	the values of X and Y are equal
X =\= Y	the values of X and Y are not equal

5

PROLOG. Arithmetic (cont...)

- **Notice the difference between the matching operators "=" and "=:="**
- **X = Y**, the goal will cause the matching of the objects X and Y, and will, if X and Y match, possibly instantiate some variables in X and Y. There will be no evaluation

```
?- 1 + 2 = 2 + 1.
false

?- 1 + A = B + 2.
A = 2
B = 1
```

- **X =:= Y** causes the arithmetic evaluation and cannot cause any instantiation of variables.

```
?- 1 + 2 =:= 2 + 1.
true
```

6

PROLOG. Arithmetic (cont...)

- **Example:**

  Computing the **greatest common divisor** (**gcd**)

- **Strategy (Euclide's Algorithm):**
  - Given two positive integers, X and Y , their **gcd**, D, can be found according to three cases:

    (1) If X and Y are equal then D is equal to X.

    (2) If X < Y then D is equal to the greatest common divisor of X and the difference Y - X.

    (3) If Y < X then do the same as in case (2) with X and Y interchanged.
- In **PROLOG:**

  gcd(X, X, X).

  gcd(X, Y, D) :- X < Y, Y1 is Y - X, gcd(X, Y1, D).

  gcd(X, Y, D) :- Y < X, gcd(Y, X, D).
- The last rule could be equivalently replaced by:

  gcd(X, Y, D) :- Y < X, X1 is X - Y, gcd(X1, Y, D).

7

PROLOG. Arithmetic (cont...)

- **Example:**

  Counting the items in the list (**length** of the list)

- **Strategy:**

  (1) If the list is empty then its length is 0.

  (2) If the list is not empty then List, [Head|Tail]; then its length is equal to 1 plus the length of the Tail.
- In **PROLOG**:

  length([], 0).

  length([H|Tail],N) :- length(Tail,N1), N is 1 + N1.
- An example of length can be:

  ?- length([a,b,[c,d],e], N).

  N = 4

Note that in the second clause of length, the two goals of the body cannot be swapped. The reason for this is that N1 has to be instantiated before the goal **N is 1 + N1** can be processed.

8

PROLOG. Arithmetic (cont...)

- **Example:**

  Counting the items in the list (**length** of the list)

  length([], 0).
  length([H|Tail],N) :- length(Tail,N1), N is 1 + N1.

- It is interesting to see what happens if we try to program the length **relation** without the use of **is**.

  length1([], 0).
  length1([H|Tail],N) :- length1(Tail,N1), N = 1 + N1.

- Now the query:

  ?- length1([a,b,[c,d],e], N).

- will produce the answer:

  N = 1+(1+(1+(1+0))).

  The addition was never explicitly forced and was therefore not carried out at all!

9

PROLOG. Arithmetic (cont...)

- **Example:**

  Factorial of n, n! = 1*2*3*…*(n-1)*n

  ?- fact(4,F).
  F = 24

- In **PROLOG**:

  fact(0,1).
  fact(N,F):- fact(N1,F1),N is N1+1,F is F1*N.

And this is correct?

fact(0,1).
fact(N,F):- N is N1+1,fact(N1,F1),F is F1*N.

And this?

fact(0,1).
fact(N,F):- N > 0, N1 is N-1, fact(N1,F1), F is F1*N.

10

PROLOG. Arithmetic (cont...)

- **Example:**

  If a list is a set

  Definition: A **list** represents a **set** if no element repeats.

  So **set** verifies if each element occurs only once in a list.

  ?- set([a,b,c,c,d]).

  false

  ?- set([a,b,c]).

  true

  ?- set([a,[b,a],c]).

  true

  ?- set([]).

  true

- In **PROLOG**:

  set([]).

  set([X|T]):- not(member(X,T)), set(T).

11

PROLOG. Arithmetic (cont...)

- **Example:**

  Power: N^k, where k is a natural number

  ?- power(2,3,P).

  P = 8

  ?- power(2,0,P).

  P = 1

- In **PROLOG**:

  power(N,0,1): - !.

  power(N,K,P):- K1 is K-1,

  power(N,K1,P1),

  P is P1*N.

  If we use "!" (cut) then PROLOG never try both clauses!

12

PROLOG. Arithmetic (cont...)

- **Example:**

  Fibonacci sequence.
  The Fibonacci sequence f(1), f(2), f(3), … is:
  1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …
  f(1) = f(2)= 1; f(n) = f(n-2) + f(n-1), if n > 2 (n≥3)
  ?- fib(6,F).
  F = 8

- In **PROLOG**:

```
fib(1,1).
fib(2,1).
fib(N,F):- N >= 3,
        N1 is N-1,
        N2 is N-2,
        fib(N1,F1), fib(N2,F2),
        F is F1+F2.
```

13

PROLOG. Arithmetic (cont...)

- **Example:**

  Counts the occurrences of an element in a list
  (Occurrences on the **superficial** level)
  ?- count(a,[a,b,c,[a,c],a],C).
  C = 2
  We can also generate a list with N identical elements
  ?- count(a,X,10).
  X = [a,a,a,a,a,a,a,a,a,a]

- In **PROLOG**:

```
count(X,[],0).
count(X,[X|T],N):- count(X,T,N1), N is N1+1.
count(X,[Y|T],N):- count(X,T,N).
```

14

PROLOG. Arithmetic (cont...)

- **Example:**

  Absolute value of a number (abs)

  ?- abs(0,A).
  A = 0
  ?- abs(-9,A).
  A = 9
  ?- abs(-9,9).
  true
  ?- abs(-9,8).
  false
  ?- abs(X,8).
  X = 8

- In **PROLOG**:

  abs(X,X):- X >= 0, !.

  abs(X,Y):- Y is -X.

This is correct?

abs(X,X):- X >= 0.

abs(X,Y):- Y is -X.

15

PROLOG. Arithmetic (cont...)

- **Example:**

  Selects the integers from a list (using **integer())**

  ?- integer(a).
  false
  ?- integer(4.23).
  false
  ?- select([1,a,3,b],I).
  I = [1,3]
  ?- select([1,[a,2],3,4],I).
  I = [1,3,4]

- In **PROLOG**:

  select([],[]).

  select([H|T],[H|I]):- integer(H), select(T,I), !.

  select([H|T],I):- select(T,I).

16

PROLOG. Arithmetic (cont...)

- **Example:**

  Verifies if a list has an even number of elements.
  ?- even([a,b,c,d]).
  true

- In **PROLOG**:

  even([_,_]).

  even([_,_|T]):- even(T).

17

PROLOG. Arithmetic (cont...)

- **Example:**

  Determine whether a given integer number is prime.
  ?- is_prime(24).
  false
  ? is_prime(17).
  true

- In **PROLOG**:

```
is_prime(2).
is_prime(3).
is_prime(P) :- integer(P), P > 3,
    P mod 2 =\= 0,
    not(has_factor(P,3)).
% has_factor(N,L) :- N has an odd factor F >= L.
has_factor(N,L) :- N mod L =:= 0.
has_factor(N,L) :- L * L < N, L2 is L + 2, has_factor(N,L2).
```