Arithmetic in Prolog

- ✔ In Prolog, arithmetic is performed by building arithmetic expressions out of *evaluable functors*, and evaluating them

- ✔ Evaluable functors are functors that actually return a value, in some contexts

Arithmetic in Prolog

- ✔ Evaluable functors in Prolog

  - ✗ 2-ary evaluable functors:

    `+`

    `-`

    `*`

    `/`

    `//`

    `mod`

  - ✗ 1-ary evaluable functors:

    `-`

    `exp`

    `ln`

    `sin`

    `arcsin`

Arithmetic in Prolog

- ✔ Most of the evaluable functors are defined as infix or prefix operators

  `3 + X` is equivalent to `+(3,X)`

  `Z * 22.2` is equivalent to `*(Z,22.2)`

  `3 + X * 22.2` is equivalent to `+(3,*(X,22.2))`

  `A mod B` is equivalent to `mod(A,B)`

  `-N` is equivalent to `-(N)`

Arithmetic in Prolog

- ✔ In most contexts, the evaluable functors act just like ordinary functors

```
?- X = +(a,b).
X = a + b
yes

?- /(foo,Z) = /(foo,bar)
Z = bar
yes

?- N = 3 * 4.
N = 3 * 4
yes
```

- ✔ However, in some special contexts, the evaluable functors are evaluated: an arithmetic function of the given arguments is computed and the result is returned

- ✔ In these contexts, the arguments to the evaluable functors must be either numbers or variables instantiated to numbers

Arithmetic in Prolog

- ✔ Special contexts in which evaluable functors are evaluated:

  - ✗ `X is E`. The arithmetic expression `E` is evaluated and the variable `X` is instantiated to the result. Fails if `E` is not an arithmetic expression

```
?- N is 3 * 4.
N = 12
yes


?- X is +(a,b).
no


?- X is +(3,4).
X = 7
yes


?- X is +(3,/(4,5)).
X = 3.8
yes


?- X is 3 + 4/5.
X = 3.8


?- X is 3 + Y.
Integer required no
```

Arithmetic in Prolog

- ✔ Special contexts in which evaluable functors are evaluated:
  - ✗ 2-ary numerical comparison predicates

    `=:=, =\=, <, >, =<, >=`

```
?- 277 * 37 > 10000.
yes
```

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

```
?- 1 + 2 = 2 + 1.
no
```

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

```
?- 1 + A =:= B + 2.
Integer required no
```

Arithmetic in Prolog

- ✔ Define a relation `length(L,N)` that is true when `N` is the number of elements in the list `L`

```
length([],0).

length([H|T],N) :- length(T,M), N is M + 1.
```

Note that the order of terms in the body of the rule clause cannot be reversed...

Compare to

```
length([],0).

length([H|T],N) :- length(T,M), N = M + 1.
```

Arithmetic in Prolog

- ✔ Define a relation `max(X,Y,Max)` that is true when `Max` is the larger of the numbers `X, Y`

```
max(X,Y,X) :- X > Y.

max(X,Y,Y) :- Y >= X.
```

Arithmetic in Prolog

- ✔ Define a relation `sumlist(List,Sum)` that is true when `Sum` is the sum of the numbers in `List`

```
sumlist([],0).

sumlist([N|Tail],Sum) :-
        sumlist(Tail,Sum1), Sum is Sum1 + N.
```

Arithmetic in Prolog

- ✔ Define a relation `factorial(N,F)` that is true when `F` is the factorial of the integer `N`.

```
factorial(0,1).

factorial(N,F) :-
    N1 is N-1, factorial(N1,F1), F is N*F1.
```