# Practical Class 2

## How Prolog Works and Recursion

Objectives:
- How Prolog works & Backtracking
- Recursion in Prolog

### 1. Prolog and Backtracking

Consider the following facts:

```
r(a, b).        s(b, c).
r(a, d).        s(b, d).
r(b, a).        s(c, c).
r(a, c).        s(d, e).
```

a) Without using the Prolog interpreter, indicate all the answers to the following queries:

```
 i.   r(X, Y), s(Y, Z).
ii.   s(Y, Y), r(X, Y).
iii.  r(X, Y), s(Y, Y).
```

b) How many times does Prolog backtrack from the second to the first goal before producing the first answer for each of the queries? Confirm your answers with the Prolog trace mode.

### 2. Backtracking and Search Tree

Consider the following facts and rules:

```
pairs(X, Y):- d(X), q(Y).        d(4).
pairs(X, X):- u(X).              q(4).
u(1).                            q(16).
d(2).
```

a) Without using the Prolog interpreter, draw up a search tree and indicate all the solutions obtained for the query `| ?- pairs(X, Y).`

b) Confirm your answers with the Prolog trace mode.

### 3. Backtracking II, the Sequel

Consider the following facts and rules:

```
a(a1, 1).        b(1, b1).        c(X, Y):- a(X, Z), b(Z, Y).
a(A2, 2).        b(2, B2).        d(X, Y):- a(X, Z), b(Y, Z).
a(a3, N).        b(N, b3).        d(X, Y):- a(Z, X), b(Z, Y).
```

c) Without using the Prolog interpreter, indicate the answers to the following queries:

```
 i.   a(A, 2).
ii.   b(A, foobar).
iii.  c(A, b3).
iv.   c(A, B).
 v.   d(A, B).
```

d) Confirm your answers with the Prolog trace mode.

## 4. Recursion

a) Implement the *factorial(+N, -F)* predicate, which calculates the factorial of a number *N*.

b) Implement *sum_rec(+N, -Sum)* as a recursive predicate to determine the sum of all numbers from one to *N*.

c) Implement the *pow_rec(+X, +Y, -P)* predicate, which recursively determines the result of raising *X* to the power of *Y*.

d) Implement the *square_rec(+N, -S)* predicate, which recursively determines que square of a number *N* (ie, without using multiplications). Suggestion: you may need to use an auxiliary predicate with additional arguments.

e) Implement the *fibonacci(+N, -F)* predicate, which determines the Fibonacci number of order *N*.

f) Implement *collatz(+N, -S)*, which receives a positive integer number *N* and determines the number of steps necessary to reach 1 following the operations set forth by this sequence: if *N* is even, in the next step it will take the value *N/2*; if *N* is odd, in the next step it will take the value *3N+1*.

g) Implement the *is_prime(+X)* predicate, which determines whether *X* is a prime number. Suggestion: a number is prime if it is divisible only by itself and one.

Factorial, sum, square, Fibonacci and Collatz series and primality for some small values:

| N | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Factorial | 1 | 1 | 2 | 6 | 24 | 120 | 720 | 5040 |
| Sum | 0 | 1 | 3 | 6 | 10 | 15 | 21 | 28 |
| Square | 0 | 1 | 4 | 9 | 16 | 25 | 36 | 49 |
| Fibonacci | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 |
| Collatz | - | 0 | 1 | 7 | 2 | 5 | 8 | 16 |
| Prime | no | no | yes | yes | no | yes | no | yes |

## 5. Tail Recursion

a) Implement tail-recursive versions of the *factorial/2*, *sum_rec/2*, pow_rec/3, *square_rec/2* and *fibonacci/2* predicates as described in the previous exercise.

b) Using the Prolog trace mode, compare calls to the regular and tail-recursive versions of *fibonacci/2* (for instance with N = 6). How do you compare the efficiency of both versions?

## 6. Greatest Common Divisor and Least Common Multiple

a) Implement *gcd(+X, +Y, -G)*, which receives two positive integer numbers, *X* and *Y*, and determines the greatest common divisor (gcd) using Euclid's algorithm. This method recursively determines the gcd between two numbers in the following manner: the gcd between *X* and *Y* is *X* when *Y* is zero; otherwise, it is the gcd between *Y* and *X mod Y*.

b) Implement *lcm(+X, +Y, -M)*, which receives two positive integer numbers, *X* and *Y*, and determines the least common multiple, which can be given by *lcm(X, Y) = X.Y / gcd(X, Y)*.

## 7. Family Relations Revisited

Consider the knowledge base from exercise 1 from the previous practical class about family relations.

a) Implement the *ancestor_of(?X, ?Y)* predicate that succeeds if *X* is an ancestor of *Y*.

b) Implement the *descendant_of (?X, ?Y)* predicate that succeeds if *X* is a descendent of *Y*.

c) Implement the *marriage_years(?X, ?Y, -Years)* predicate that succeeds if X and Y were married and are now divorced, and determines the marriage duration, in years.

d) Write queries to answer the following questions:

   i. Who is Gloria's descendant, but not Jay's?

   ii. What ancestors do Haley and Lily have in common?

   iii. To whom were/are both Dede and Gloria married?

e) Suppose that we now have additional predicates in the knowledge base containing the date of birth of everyone. Example:

```
born(jay, 1946-5-23).
born(claire, 1970-11-13).
born(mitchell, 1973-7-10).
```

   i. Implement *before(+X, +Y)*, which receives two dates in the format shown above and succeeds if *X* is before *Y*.

   ii. Implement *older(?X, ?Y, ?Older)*, which unifies *Older* with the oldest person between *X* and *Y*. If *X* and *Y* share the same birthdate, the predicate should fail.

   iii. Implement *oldest(?X)*, which unifies *X* with the oldest person in the knowledge base.

## 8. Red Bull Air Race Revisited

Consider the knowledge base from exercise 3 from the previous practical class about the Red Bull Air Race.

a) Implement the *most_gates(?X)* predicate, which unifies *X* with the circuit that has the highest number of gates.

b) Implement the *least_gates(?X)* predicate, which unifies *X* with the circuit that has the lowest number of gates.

c) Implement the *gate_diff(?X)* predicate, which unifies *X* with the difference of gates between the circuits with the highest and lowest number of gates.

d) Implement *same_team(?X, ?Y)*, which unifies X and Y with pilots racing for the same team.

e) Implement *is_from_winning_team(?P, ?C)*, which unifies *P* with a pilot from the same team as the pilot who won circuit *C*.

## 9. Jobs and Supervisors Revisited

Consider the knowledge base from exercise 5 from the previous practical class about jobs and supervisors. Implement the *superior(+X, +Y)* predicate, which succeeds if person *X* occupies a position that is superior to the position occupied by person *Y*.