Declaratieve Talen
Prolog 3

1 Even more classes

Given the following facts about professors and students:

```
teaches(berbers,bs).
teaches(berbers,iw).
teaches(demoen,ab).
teaches(demoen,cc).
teaches(holvoet,bvp).
teaches(moens,bvp).
teaches(danny,ai).
teaches(maurice,ai).
teaches(dedecker,socs).

takes(tom,bs).
takes(tom,bvp).
takes(tom,socs).
takes(maarten,socs).
takes(maarten,bs).
takes(pieter,bvp).
```

Model the following relations *without any double solutions*:

- `takes_same_course(X,Y)`: `X` takes the same course as `Y`.
- `teach_same_course(X,Y)`: `X` teaches a common course with `Y`.
- `teaches_multiple_courses(X)`: `X` teaches more than 1 course.

Use the `findall/3` predicate. Attention, extend the dataset to confirm that your program gives no double solutions. With 'double solutions' we mean solutions where the variables take the same value.

For example:

```
?- takes_same_course(X,Y).

X = tom
```

```
Y = maarten ;

X = tom
Y = pieter ;

X = maarten
Y = tom ;

X = pieter
Y = tom ;

No
```

is a valid solution without any doubles.

Hint:

- The third argument of findall is a list, which is why it can contain doubles, whereas sets can't. Look for a built-in that handles this conversion for you.
- The following code

  ```
  a(1).
  a(2).
  a(2).
  a(3).
  ```

  is equivalent with:

  ```
  a(X) :- member(X,[1,2,2,3]).
  ```

2 A prelude to functional programming

Write a predicate `map(P,Xs,Ys)` that relates a list `Xs` with a list `Ys` using a predicate `P` (with arity 2), as in the following example:

```
inc(X,Y) :- Y is X + 1.

?-  map(inc,[1,2,3],Ys).
Ys = [2,3,4] ? ;
no
```

Note that this behavior already exists as the built-in predicate `maplist/3`. Do not use it for this exercise though.

Hint: Review the manual for `=..` and `call/1` (**not** `call/[2-6]`).

3 Junior interpreter

Extend the mini-meta-interpreter from the lectures with the builtins necessary to interpret `fib/2`. The mini-meta-interpreter from the lectures was defined as follows:

```
interpret((G1,G2)) :-
    !,
    interpret(G1),
    interpret(G2).

interpret(true) :- !.

interpret(Head) :-
    clause(Head,Body),
    interpret(Body).
```

The predicate `fib/2` is defined as follows:

```
fib(0,1).
fib(1,1).
fib(N,F) :-
    N > 1,
    N2 is N - 2,
    fib(N2,F2),
    N1 is N - 1,
    fib(N1,F1),
    F is F1 + F2.
```

What is the time complexity of `fib/2`? Does the time complexity differ between SWI-Prolog and your meta-interpreter?

Now add memoization to your interpreter:

- The interpreter has a memory that is initially empty.
- Every time the interpreter must call a predicate, it first checks whether the answer is in its memory:
  - If the result of the call was saved in memory, return this answer.
  - If the result of the call was not saved in memory, call the predicate and save the answer to memory before returning it.

What is the time complexity of `fib/2` after introducing memoization? What will be the behavior of your interpreter when backtracking? What will be the behavior if the answers contain variables?

4 Infinite Turing Tape

Devise a finite Prolog-representation for the tape of a Turing-machine and its corresponding reading head. This tape has an infinite number of positions. Each position can be empty (represented using '#') or contain a symbol. The tape contains only a finite number of symbols. The reading head is located above a certain position of the tape.

Write predicates that enable representation of the following statements:

- Moving the reading head towards the right.
- Moving the reading head towards the left.
- Read the symbol in the position under the reading head.
- Write a given symbol to the position under the reading head.

Can you think of a data structure where these operations are performed in $\mathcal{O}(1)$?

5 N-queens problem

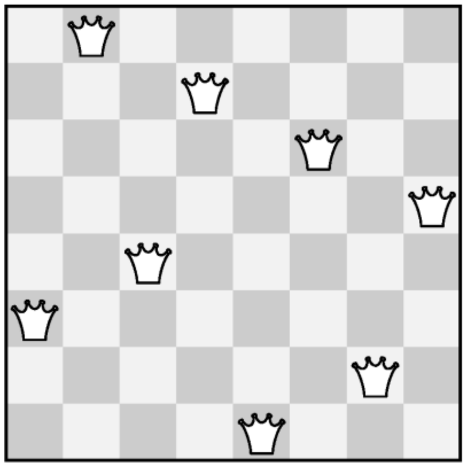
A chessboard consists of 64 squares, divided into 8 rows and 8 columns. In chess, a queen can move horizontally, vertically or diagonally. In each direction, she can travel as far as she wants. The 8-queens problem is a chess problem where 8 queens must be positioned on a chessboard in such a way that no queen can capture another queen in **one** move. We generalize this problem to a N-queen problem where N queens must be positioned on a chessboard with N rows and N columns.

Write a predicate `queens(N,L)` that for a given N returns a list of all possible chessboard positions for the queens. You can represent a chess configuration using a list where the index (1-based) represents the column number, and the value represents the row number. This representation already guarantees that the queens can't attack each other in at least one of the four directions (horizontal, vertical and the two diagonals). For which direction does this representation force such a guarantee? How can you easily enforce this restriction for one of the other three directions?

A naive solution would be to generate all possible configurations for the queens, and then check which configurations are consistent with the imposed restrictions. Checking whether a configuration is consistent is possible by checking for each queen whether she can attack one of the queens to her right in a horizontal, vertical or diagonal move. As the 'can attack' relation between two queens is reflexive, it is not necessary to check whether a queen can attack the queens to her left.

Make sure that your program generates all possible solutions for a given N. The following code fragment and figure 1 show **one** of the 92 solutions for the 8-queens problem. Table 1 documents the number of solutions for a few different values of N.

```
?- queens(8,L).
L = [6, 1, 5, 2, 8, 3, 7, 4] ;
```



Figuur 1: **one** of the 92 solutions for the 8-queens problem.

Number of queens	1	2	3	4	5	6	7	8	9	10
Number of solutions	1	0	0	2	10	4	40	92	352	724

Tabel 1: The number of solutions for a given number of queens.