Examination of

Programming languages and Programming Methodologies

15 January 2022, 9h00 – 12h00

General Guidelines

1. The examination is **closed book** and takes (at most) 3 hours.
2. Use the two ANSWER bundles to write down your answers: use bundle 1 for questions 1,2 and 3 and bundle 2 for question 4.
3. At the back of each answer bundle you will find a number of scrap pages.
4. Make sure that your handwriting is **readable**.
5. Write on **one** side of the sheet only.

Questions

1. *(2 marks)*

   1) Do the following Prolog queries succeed or fail? In case of success, give the bindings of the variables in the query. In case of failure, explain why.

      i. `?- [X,Y|T] = [E,[1,E],[1],E], \+ T = Y.`

      ii. `?- N is 2*4, M = N + 1.`

      iii. `?- [a,[b|X]] = [X|T] .`

   2) What are the domains of the decision variables X, Y and B in CLP(FD) after the following query:

      ```
      ?-[X,Y] ins 0..100, X #<7, X #= Y+3, B #<==> ((X -Y ) #> 0).
      ```

2. *(3 marks)* Consider the following Prolog predicates:

```
test(3,three).
test(a,aa).
test(2,two).

tt(_, [_]).
tt(Xs, [T|Ys]) :- test(D,T), Xs = [D|Ys],!.
tt(Xs, [D|Ys]) :- Xs = [D|Ds], tt(Ds,Ys).
```

   Does the query `?- tt([b,a,3,2],K).` succeed or fail? Sketch its execution by Prolog (e.g. by giving an execution tree). In case of success, indicate how many times it succeeds and give the binding of the variable K for every success. In case of failure, explain why.

3. *(6 marks)* List predicates

1) An element Z of a list of positive integers L is called **left-visible** if for all elements X of L that occur before Z, it is the case that X < Z.
Write the **Prolog** predicate `visibleCount(L, Count)` that succeeds if Count is the number of left-visible elements in the given non-empty list L. Make sure that your recursive predicates are **tail recursive**.

accumulator = Count

```
?- visibleCount([1,3,2,1,4,4,5], C).
C = 4 ;        % 1 3     4   5  are left-visible
false.
?- visibleCount([5,1,3,2,1,4,4,5],C).
C = 1 ;        % 5               is left-visible
false.
```

2) Think of a cycle in a graph as being a sequence of nodes in a graph, such that consecutive nodes are connected by a directed edge, nodes are represented by an integer (also known as their label) and all nodes are distinct except for the first and last node, which are required to be the same. We say that a cycle is **clock-safe** if all but one directed edges point from a smaller node to a larger node.
Consider the cycle consisting of an edge from node 0 to node 3, an edge from node 3 to node 5, an edge from node 5 to node 8, and an edge from node 8 to node 0. This cycle is clock-safe: the edge from 8 to 0 is the only edge not going from a smaller to a larger node.

To represent a cycle with N distinct nodes we use a list L with length N+1. Our example cycle is represented by a list of 5 nodes. The following lists are all four valid representations: [0,3,5,8,0], [3,5,8,0,3], [5,8,0,3,5], [8,0,3,5,8].

Write the **CLP(FD)** predicate `cyclicAscendingList(L)` that succeeds if the list L of N+1 finite domain elements represents a clock-safe cycle with N nodes.
Some examples of queries:

```
?- L = [3,5,8,0,3], cyclicAscendingList(L).
L = [3, 5, 8, 0, 3] ;
false.

?- length(L,5), L ins 0..3, cyclicAscendingList(L), label(L).
L = [0, 1, 2, 3, 0] ;
L = [1, 2, 3, 0, 1] ;
L = [2, 3, 0, 1, 2] ;
L = [3, 0, 1, 2, 3] ;
false.

?- length(L,5), L ins 1..3, cyclicAscendingList(L), label(L).
false.

?- L = [1,2,0,3,1], L ins 0..3, cyclicAscendingList(L).
false % note that L is not clock-safe as 2>0 and 3>1

?- length(L,5), L ins 0..6, L = [1,3,X,6,1],cyclicAscendingList(L).
L = [1, 3, X, 6, 1 ],
X in 4..5 ;
false.
```

4. *(9 marks)* **Use Answer bundle 2 for this question; do not forget to write your name and student number on it.**

Given N married couples. For each couple you get the name of the man and the name of the woman. all different
You may assume that every name occurs only once. The couples arranged a private late night visit to the Eiffel tower and they have to use the old elevator to reach the top floor. The elevator can take at most P persons and there has to be at least one person in the elevator when it goes up or down in order to operate the elevator. As we are dealing with very jealous people it is the case that a woman is not allowed to be together with another man unless her husband is present. This "jealousy" requirement has to be satisfied at the top floor, the ground floor and even in the elevator.

The problem is to schedule the use of the elevator. You get a begin state (that specifies where the elevator is, who is at the ground floor and who is at the top floor. Also the final state is given, namely who has to be at the ground floor and at the top floor. You may assume that the begin and the final state are feasible and compatible with the jealousy requirement.

An example. There are 3 couples: Andy and Ann, Bart and Betsy, and Carl and Cindy. Initially they are all at the ground floor (begin state) and in the final state they are all at the top floor. The elevator is at the ground floor and it can take at most 2 persons. The final state can be reached as follows: Ann and Betsy go up, Ann goes down, Ann and Cindy go up, Cindy goes down, (state a: elevator down; ground: 3 men and Cindy; top: Ann and Betsy) Andy and Bart go up, Andy and Ann go down, Andy and Carl go up, (state b: elevator up; ground: Ann and Cindy; top: 3 men and Betsy) Betsy goes down, Ann and Betsy go up, Ann goes down and Ann and Cindy go up.

Note that we want to write a program that can solve the elevator problem for different values of N and P.

Note that for this question **you may use the predicate sort/2 without defining the predicate yourself.**

1) Represent the information about the N couples as Prolog facts. Give the facts for the 3 couples in the example.
2) Represent a state (namely, the information about where the elevator is, who is at the ground floor and who is at the top floor) as a Prolog term and give the Prolog terms for state a and state b of the example.
3) Write a predicate `samestate(S1,S2)` that succeeds when the given state `S1` actually represents the same situation as the given state `S2`.
4) Given a set of people (men and/or women). Write a predicate that checks the "jealousy" condition. Make sure this predicate is compatible with the previous data representations: you will use this predicate to check the people that are at the ground floor, the top floor and even in the elevator.
5) i. Write a predicate that for a given state computes the possible state transitions (the next states) such that the above rules about the elevator and jealousy are followed: `transition(S1,S2,Move)` succeeds if state `S2` can be reached from the given state `S1` with move `Move`. You can freely choose how to represent `Move`.
   ii. How do you represent the maximum number of people allowed in the elevator (i.e. the parameter P).
6) Write the predicate `solve(S1,S2,Moves)` that computes the `Moves` that are needed to go from the given state `S1` to the given state `S2`.

Success

Gerda Janssens