Programming Paradigms Exercise (3)
Model Answers

Marco Montali Thomas Tschager

2nd Semester 2017/18

1. (a)

```
min_elem([Min],Min).

min_elem([Min|Tail], Min):-
  min_elem(Tail,TailMin),
  Min =< TailMin.

min_elem([H|Tail],Min):-
  min_elem(Tail,Min),
  Min < H.
```

(b)

query) Tries to prove `minElem([19,3,29], X)`.
- 1.1) Tries to prove first case `minElem([Min1], X)`.
  - 1.1) Fails to match `[Min1]` to `[19,3,29]`.
- 2.1) Tries to prove second case `minElem([Head1|Tail1], X)`.
  - 2.1) Matches `Head1` to `19` and `Tail1` to `[3,29]`.
  - 2.2) Tries to prove `minElem([3,29], TailMin1)`.
    - 1.1) Tries to prove first case `minElem([Min2], TailMin1)`.
      - 1.1) Fails to match `[Min2]` to `[3,29]`.
    - 2.1) Tries to prove second case `minElem([Head2|Tail2], TailMin1)`.
      - 2.1) Matches `Head2` to `3` and `Tail2` to `[29]`.
      - 2.2) Tries to prove `minElem([29], TailMin2)`.
        - 1.1) Tries to prove first case `minElem([Min3], TailMin2)`.
          - 1.1) Matches `[Min3]` to `[29]`.
        - 1.1) Has proven `minElem([Min3], TailMin2)` for `TailMin2 = 29`.
      - 2.2) Has proven `minElem([29], TailMin2)` for `TailMin2 = 29`.
      - 2.3) Proves `Head2 =< TailMin2` $\Leftrightarrow$ `3 =< 29`.
      - 2.4) Proves `Min2 is Head2`, for `Min2 = 3`
    - 2.1 Has proven `minElem([Head2|Tail2], TailMin1)` for `TailMin1 = 3`.
  - 2.2) Has proven `minElem([3,29], TailMin1)` for `TailMin1 = 3`.
  - 2.3) Fails to prove `Head1 =< TailMin1` $\Leftrightarrow$ `19 =< 3`.
- 3.1) Tries to prove third case `minElem([Head3|Tail3], X)`.
  - 3.1) Matches `Head3` to `19` and `Tail3` to `[3,29]`.
  - 3.2) Tries to prove `minElem([3,29], TailMin3)`.
    - 1.1) Tries to prove first case `minElem([Min4], TailMin3)`.

1.1) Fails to match `[Min4]` to `[3,29]`.
2.1) Tries to prove second case `minElem([Head4|Tail4], TailMin3)`.
2.1) Matches `Head4` to `3` and `Tail4` to `[29]`.
2.2) Tries to prove `minElem([29], TailMin4)`.
1.1) Tries to prove first case `minElem([Min5], TailMin4)`.
1.1) Matches `[Min5]` to `[29]`.
1.1) Has proven `minElem([Min5], TailMin4)` for `TailMin4` = 29.
2.2) Has proven `minElem([29], TailMin4)` for `TailMin4` = 29.
2.3) Proves `(Head4 =< TailMin4)` ⇔ `(3 =< 29)`.
2.4) Proves `(Min4 is Head4)` for `Min4` = 3
2.1) Has proven `minElem([Head4|Tail4], TailMin3)` for `TailMin3` = 3.
3.2) Has proven `minElem([3,29], TailMin3)` for `TailMin3` = 3.
3.3) Proves `(Head3 < TailMin3)` ⇔ `(19 < 3)`.
3.4) Proves `(X is 3)`, for `X` = 3.
3.1) Has proven `minElem([Head3|Tail3], X)` for X = 3.
query) Has proven `minElem([19,3,29], X)` for X = 3.

(c)
```
min_elem([Min],Min):-!.

min_elem([Min|Tail], Min):-
  min_elem(Tail,TailMin),
  Min =< TailMin,!.

min_elem([_|Tail],Min):-
  min_elem(Tail,Min).
```

The first `!` is required because the singleton list `[X]` also matches with the pattern `[H|T]`, with `H/X` and `T/[]`. Also the second `!` is required because the two recursive definitions may both seem to match, but they are in fact mutually exclusive. Once `!` is used, the third clause only matches if the second fails, and therefore we do not need to apply the test anymore.

2.
```
revlist([],[]).
revlist([H|T],RL):-
  revlist(T,RT),
  append(RT,[H],RL).
```

3. (a)
```
is_a_list([]).
is_a_list([_|_]).

elem(X):- \+ is_a_list(X).

make_flat([],[]).

make_flat([H|T],[H|Tflat]):-
  elem(H),!,
  make_flat(T,Tflat).

make_flat([H|T],L):-
```

```
  make_flat(H,Hflat),
  make_flat(T,Tflat),
  append(Hflat,Tflat,L).
```

Notice the usage of `!` to impose mutual exclusion between the different definitions of `make_flat`.

(b) Declaratively, the answers are *all* nested lists that, once flattened, give `[a,b,c]` as a result. Procedurally, by using the program above, Prolog returns:

```
X = [a, b, c]
X = [a, b, c, []]
X = [a, b, c, [], []]
X = [a, b, c, [], [], []]
X = [a, b, c, [], [], [], []]
X = [a, b, c, [], [], [], [], []]
X = [a, b, c, [], [], [], [], [], []]
X = [a, b, c, [], [], [], [], [], []|...]
...
```

There are in fact infinitely many lists following this pattern, whose flattening results in `[a,b,c]`.

4.
```
equal([], []).
equal([H|T1], [H|T2]) :-
  equal(T1, T2).

dom([H1], [H2]) :- H1 < H2.
dom([H|T1], [H|T2]) :-
  \+ equal(T1, T2),
  dom(T1,T2), !.
dom([H1|T1], [H2|T2]) :-
  H1 < H2,
  dom(T1,T2), !.
```

5.
```
% belongsTo(A,B,X) is true if X belongs to the interval [A,B].
belongsTo(B, B, B) :- !.

belongsTo(A, B, X) :-
  A < B,
  X is A.

belongsTo(A, B, X) :-
  A < B,
  A1 is A+1,
  belongsTo(A1, B, X).

notPrime(X) :-
  MaxDivisor is X div 2,
  belongsTo(2,MaxDivisor,Divisor),
  M is X mod Divisor,
  M = 0.

isPrime(X) :- \+ notPrime(X).

firstPrimeBetween(A,B,X) :-
  belongsTo(A,B,X),
  isPrime(X),!.
```

6.
```
perkm(1,1000,0.08).
perkm(1001,10000,0.04).
perkm(10001,20000,0.02).
perkm(20001,99999999,0.0).

euro(0,0).
euro(Km, Price) :-
  perkm(Min,Max,P1),
  Km >= Min,
  Km =< Max,
  R is Min-1,
  euro(R, P2),
  Price is (Km-R)*P1+P2.
```