Programming Paradigms Exercise (3)
Model Answers

Marco Montali        Thomas Tschager

2nd Semester 2017/18

1. (a) \texttt{min\_elem([\textit{Min}],\textit{Min}).}

\texttt{min\_elem([\textit{Tail}], \textit{Min}):-}
\texttt{min\_elem(Tail, Tail\_Min),}
\texttt{Min =< Tail\_Min.}

\texttt{min\_elem([\textit{H}|\textit{Tail}], \textit{Min}):-}
\texttt{min\_elem(Tail, Min),}
\texttt{Min < H.}

(b) \texttt{query} \texttt{Tries to prove minElem([19, 3, 29], X).}
1.1) \texttt{Tries to prove first case minElem([\textit{Min} 1], X).}
1.1) \texttt{Fails to match [Min 1] to [19, 3, 29].}
2.1) \texttt{Tries to prove second case minElem([\textit{Head} 1 | \textit{Tail} 1], X).}
2.1) \texttt{Matches Head 1 to 19 and Tail 1 to [3, 29].}
2.2) \texttt{Tries to prove minElem([3, 29], \textit{Tail\_Min 1}).}
2.2) \texttt{Fails to match [Min 2] to [3, 29].}
2.2) \texttt{Tries to prove second case minElem([\textit{Head} 2 | \textit{Tail} 2], \textit{Tail\_Min 1}).}
2.2) \texttt{Matches Head 2 to 3 and Tail 2 to [29].}
2.2) \texttt{Tries to prove minElem([29], \textit{Tail\_Min 2}).}
1.1) \texttt{Tries to prove first case minElem([\textit{Min} 3], \textit{Tail\_Min 2}).}
1.1) \texttt{Fails to match [Min 3] to [29].}
1.1) \texttt{Has proven minElem([Min 3], Tail\_Min 2) for Tail\_Min 2 = 29.}
2.2) \texttt{Has proven minElem([29], Tail\_Min 2) for Tail\_Min 2 = 29.}
2.3) \texttt{Proves Head 2 =< Tail\_Min 2 = 3 =< 29.}
2.4) \texttt{Proves Min 2 is Head 2, for Min 2 = 3}
2.1) \texttt{Has proven minElem([\textit{Head} 2 | \textit{Tail} 2], \textit{Tail\_Min 1}) for Tail\_Min 1 = 3.}
2.2) \texttt{Has proven minElem([3, 29], \textit{Tail\_Min 1}) for Tail\_Min 1 = 3.}
2.3) \texttt{Fails to prove Head 1 =< Tail\_Min 1 =< 19 =< 3.}
3.1) \texttt{Tries to prove third case minElem([\textit{Head} 3 | \textit{Tail} 3], X).}
3.1) \texttt{Matches Head 3 to 19 and Tail 3 to [3, 29].}
3.2) \texttt{Tries to prove minElem([3, 29], \textit{Tail\_Min 3}).}
1.1) \texttt{Tries to prove first case minElem([\textit{Min} 4], \textit{Tail\_Min 3}).}
1. Fails to match \([\text{Min}_4]\) to \([3,29]\).

2.1) Tries to prove second case \(\text{minElem}(\text{Head}_4 | \text{Tail}_4), \text{TailMin}_4)\).

2.1) Matches \(\text{Head}_4\) to 3 and \(\text{Tail}_4\) to \([29]\).

2.2) Tries to prove \(\text{minElem}(\text{Tail}_4), \text{TailMin}_4)\).

2.2) Matches \([\text{Min}_4]\) to \([29]\).

2.1) Has proven \(\text{minElem}(\text{Head}_4 | \text{Tail}_4), \text{TailMin}_4)\) for \(\text{TailMin}_4 = 29\).

2.2) Has proven \(\text{minElem}(\text{Tail}_4), \text{TailMin}_4)\) for \(\text{TailMin}_4 = 29\).

2.3) Proves \((\text{Head}_4 \leq \text{TailMin}_4) \iff (3 \leq 29)\).

2.4) Proves \((\text{Min}_4 \text{ is } \text{Head}_4) \iff \text{Min}_4 = 3\).

2.1) Has proven \(\text{minElem}(\text{Head}_4 | \text{Tail}_4), \text{TailMin}_3)\) for \(\text{TailMin}_3 = 3\).

3.2) Has proven \(\text{minElem}(\text{Tail}_3), \text{TailMin}_3)\) for \(\text{TailMin}_3 = 3\).

3.3) Proves \((\text{Head}_3 < \text{TailMin}_3) \iff (19 < 3)\).

3.4) Proves \((X \text{ is } 3)\), for \(X = 3\).

query) Has proven \(\text{minElem}(\text{Tail}_3, X)\) for \(X = 3\).

(c) \(\text{min\_elem}(\text{Min}, \text{Min}) :- !.\)

\[
\text{min\_elem}(\text{Min}|\text{Tail}), \text{Min} :- \\
\text{min\_elem}(\text{Tail}, \text{TailMin}), \\
\text{Min} =< \text{TailMin}, !.
\]

\[
\text{min\_elem}([_|\text{Tail}], \text{Min}) :- \\
\text{min\_elem}(\text{Tail}, \text{Min}).
\]

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. \(\text{revlist}([], []).\)

\[
\text{revlist}(\text{H}|\text{T}, \text{RL}) :- \\
\text{revlist}(\text{T}, \text{RT}), \\
\text{append}(\text{RT}, \text{H}, \text{RL}).
\]

3. (a) \(\text{is\_a\_list}([]).\)

\[
\text{is\_a\_list}([_|_]).\]

\[
\text{elem}(X) :- \neg \text{is\_a\_list}(X).
\]

\[
\text{make\_flat}([], []).\]

\[
\text{make\_flat}(\text{H}|\text{T}, \text{H}|\text{T\_flat}) :- \\
\text{elem}(\text{H}), !, \\
\text{make\_flat}(\text{T}, \text{T\_flat}).
\]

\[
\text{make\_flat}(\text{H}|\text{T}, \text{L}).
\]
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, [], [], [], [], [], []]

... 

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 \[A,B\].

\[
\text{belongsTo}(\text{A, B, X}) :- \\
\text{A < B,} \\
\text{X is A.}
\]

\[
\text{belongsTo}(\text{A, B, X}) :- \\
\text{A < B,} \\
\text{A1 is A+1,} \\
\text{belongsTo(A1, B, X).}
\]

\[
\text{notPrime}(\text{X}) :- \\
\text{MaxDivisor is X div 2,} \\
\text{belongsTo(2,MaxDivisor,Divisor),} \\
\text{M is X mod Divisor,} \\
\text{M = 0.}
\]

\[
\text{isPrime}(\text{X}) :- \neg \text{notPrime}(\text{X}).
\]

\[
\text{firstPrimeBetween}(\text{A,B,X}) :- \\
\text{belongsTo(A,B,X),} \\
\text{isPrime(X),!}.
\]

6. perkm(1,1000,0.08).

\[
\text{perkm}(1001,10000,0.04). \\
\text{perkm}(10001,20000,0.02). \\
\text{perkm}(20001,99999999,0.0).
\]

\[
\text{euro}(0,0). \\
\text{euro}(\text{Km, Price}) :- \\
\text{perkm(Min,Max,P1),} \\
\text{Km >= Min,} \\
\text{Km <= Max,} \\
\text{R is Min-1,} \\
\text{euro(R, P2),} \\
\text{Price is (Km-R)*P1+P2.}
\]