2016

Notes

Anything in *italics* is a note.

Question 1

a

i) `X=1.`
ii) `false.`
iii) `false.`
iv) `true.`
v) `false.`
vi) `true.`
vii) `false.`
viii) `false`
ix) `ERROR: Arguments are not sufficiently instantiated.`
x) `L=[].`

b

```
isSet([]).
isSet([H|T]) :- not(member(H, T)) -> isSet(T).
```

c

Solution thanks to Michael Vu:

```
moreThanOne(List) :- moreThanOne(List, []).
moreThanOne([], L) :- length(L, X), X>1.
moreThanOne([H|T], []) :- moreThanOne(T, [H]).
moreThanOne([H|T], L) :- member(H, L) -> moreThanOne(T, L); moreThanOne(T, [H|L]).
```

d

Solution thanks to Michael Vu:

```
moreThan(List, Num) :- moreThan(List, Num, []).
moreThan([], Num, X) :- length(X, Length), Length > Num.
moreThan([H|T], Num, []) :- moreThan(T, Num, [H]).
moreThan([H|T], Num, L) :- member(H, L) -> moreThan(T, Num, L); moreThan(T, Num, [H|L]).
```

Question 2

a

i

```
fac(0, 1).
fac(N, F) :- NN is N-1, fac(NN, FF), F is N*FF.
```

ii

```
factorial(0, F, F).
factorial(N, A, F) :- AA is N*A,
                      NN is N-1,
                      factorial(NN, AA, F).
fact(N, F) :- factorial(N, 1, F).
```

b

i

```
fib(0, 1).
fib(1, 1).
fib(N, F) :- N2 is N-2,
             fib(N2, FF),
             N1 is N-1,
             fib(N1, FFF),
             F is FF+FFF.
```

ii

```
fibonacci(0, _, F, F).
fibonacci(N, P1, P2, F) :- NN is N-1,
```

```
                            FF is P1+P2,
                            fibonacci(NN, P2, FF, F).
fib(N, F) :- fibonacci(N, 1, 1, F).
```

Question 3

a

```
s --> a(N), b(M), c(K), {M+N =< K}.

a(NN) --> [a], a(N), {NN is N+1}.
a(0) --> [].

b(MM) --> [b], b(M), {MM is M+1}.
b(0) --> [].

c(KK) --> [c], c(K), {KK is K+1}.
c(0) --> [].
```

b

Difference lists represent the information about grammatical categories as the different between two lists. The first list in the pair of lists is what needs to be consumed, and the second list is what should be left behind. These are useful for checking DCGs as the first list should be entirely consumed by our recogniser and leave an empty list behind.

c

```
s(In, Diff) :- a(In, L1, N), b(L1, L2, M), c(L2, Diff, K), M+N =< K.

a(['a'|T], Res, NN) :- a(T, Res, N), NN is N+1.
a(Res, Res, 0).

b(['b'|T], Res, MM) :- b(T, Res, M), MM is M+1.
b(Res, Res, 0).

c(['c'|T], Res, KK) :- c(T, Res, K), KK is K+1.
c(Res, Res, 0).
```

d

```
s(N, L) --> a(S),
            {NN is N*2},
            {f(S, NN, L)}.

% Assemble list of input
a([A|X]) --> [A], a(X).
a([]) --> [].

f(S, N, L) :- g(S, N, L).
f([H|T], N, L) :- append(T, [H], S),
                  g(S, N, L).

% Generate output
g(_, 0, []).
% Don't change order of set
g(S, N, [H|X]) :- N>1,
                  append([H|[]], T, S),
                  length([H], HL),
                  NN is N-HL,
                  g([H|T], NN, X).
% Change order of set
g(S, N, [H|X]) :- N>0,
                  append([H|[]], T, S),
                  append(T, [H], NS),
                  length([H], HL),
                  NN is N-HL,
                  g(NS, NN, X).
```