2015

Question 1

a

`i` is a fact.

`ii` is a fact.

`iii` is a rule.

`iii` is the best translation. It's checked that Mary owns `X` if it's a lamb and if it is also white.

b

i

```
false
```

ii

```
X = 3+2
```

iii

```
false
```

iv

```
Argument not properly instantiated
```

v

```
X = 5
```

vi

```
false
```

vii

```
true
```

viii

```
true
```

c

```
split(_, [], [], []).
split(N, [H|T], [H|Small], Big) :- H < N, split(N, T, Small, Big).
split(N, [H|T], Small, [H|Big]) :- H >= N, split(N, T, Small, Big).
```

d

```
pow(_, 0, 1).
pow(N, E, R) :- EN is E-1, pow(N, EN, X), R is X*N.

sumOfPowers(0, A, A).
sumOfPowers(N, Sum, Ans) :- pow(N, N, Power),
                            NewN is N-1,
                            NewSum is Power+Sum,
                            sumOfPowers(NewN, NewSum, Ans).
sumOfPowers(N, S) :- pow(N, N, Power),
                     NewN is N-1,
                     sumOfPowers(NewN, Power, S).
```

Question 2

a

```
member(X, [H|T]) :- X is H, !; member(X, T).
```

b

It is a green cut. A cut is green if it does not change the meaning of the predicate. It should give the same result, but only be more efficient. A cut is red if an equivalent program without the cut doesn't give the same result.

In this example, the cut is green. The program will still evaluate to true if `X` is in the list.

c

```
last(X, [X|[]]) :- !.
last(X, [_|T]) :- last(X, T).
```

d

```
multiple(_, _, 2) :- !.
multiple(N, [H|T], C) :- H is N -> CC is C+1, multiple(N, T, CC); multiple(N, T, C).
multiple(N, L) :- multiple(N, L, 0).
```

e

```
next(A, [H|_]) :- A is H.
next(A, B, [H|T]) :- A is H, next(B, T); next(A, B, T).
```

f

```
next3(A1, A2, A3, B1, A2, A3, L1, L2, L3) :- next(A1, B1, L1),
                                              mem3(A1, A2, A3, L1, L2, L3),
                                              mem3(B1, A2, A3, L1, L2, L3).
next3(A1, A2, A3, A1, B2, A3, L1, L2, L3) :- next(A2, B2, L2),
                                              mem3(A1, A2, A3, L1, L2, L3),
                                              mem3(A1, B2, A3, L1, L2, L3).
next3(A1, A2, A3, A1, A2, B3, L1, L2, L3) :- next(A3, B3, L3),
                                              mem3(A1, A2, A3, L1, L2, L3),
```

```
                                                mem3(A1, A2, B3, L1, L2, L3).

next3(A1, A2, A3, B1, B2, A3, L1, L2, L3) :- next(A1, B1, L1),
                                              next(A2, B2, L2),
                                              mem3(A1, A2, A3, L1, L2, L3),
                                              mem3(B1, B2, A3, L1, L2, L3).
next3(A1, A2, A3, B1, A2, B3, L1, L2, L3) :- next(A1, B1, L1),
                                              next(A3, B3, L3),
                                              mem3(A1, A2, A3, L1, L2, L3),
                                              mem3(B1, A2, B3, L1, L2, L3).
next3(A1, A2, A3, A1, B2, B3, L1, L2, L3) :- next(A2, B2, L1),
                                              next(A3, B3, L1),
                                              mem3(A1, A2, A3, L1, L2, L3),
                                              mem3(A1, B2, B3, L1, L2, L3).
```

Question 3

a

```
s --> u(U), [2], v(V), {U is V*2}.

u(U) --> [0], u(U).
u(NU) --> [1], u(U), {NU is U+1}.
u(0) --> [].

v(NZ) --> [0], v(Z), {NZ is Z+1}.
v(Z) --> [1], v(Z).
v(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, Difference) :- u(In, U, Leftover), v(Leftover, V, Difference), U is V*2.

u([0|List], Count, Result) :- u(List, Count, Result).
```

```
u([1|List], NCount, Result) :- u(List, Count, Result), NCount is Count+1.
u([2|List], 0, List).

v([0|List], NCount, Result) :- v(List, Count, Result), NCount is Count+1.
v([1|List], Count, Result) :- v(List, Count, Result).
v(Leftover, 0, Leftover).
```

d

```
mkList(0, []) :- !.
mkList(V, [V|L]) :- VV is V-1, mkList(VV, L).

s(V) --> {V >= 1, VV is V*2, mkList(VV, L)}, gen(L, VV).

gen(L, V) --> {V >= 0, member(X, L), VV is V-X}, [X], gen(L, VV).
gen(_, 0) --> [].
```