Comp 348
Principles of Programming Languages
Fall 2020

Assignment 1

Amr Hefny - 40082583
Philippe Carrier - 40153985
Ryan Leyland - 40015165
Sobhan Mehrpour Kevishahi - 40122438

Date: 6th October 2020
Section: U
Professor: Dr. Ali Jannatpour

Question 1 – see Employee.java, Person.java

Question 2 – see Driver.java, SalaryRange.java

Question 3

1. **food(bread, X) = Food(Y, soup)**

   Error;

   Capital “F” Food is not a valid functor.

2. **Bread = soup**

   Unify;

   Bread = soup.

3. **Bread = Soup**

   Unify;

4. **food(bread, X, milk) = food(Y, salad, X)**

   Does Not Unify;

   X can’t be both milk and salad.

5. **manager(X) = Y**

   Unify;

   The entire thing will be unified with Y.

6. **meal(healthyFood(bread), drink(milk)) = meal(X,Y)**

   Unify;

   X= healthyFood(bread)

   Y = drink(milk)

7. **meal(eat(Z), drink(milk)) = [X]**

   Does Not Unify;

   LHS isn’t a list

8. **[eat(Z), drink(milk)] = [X, Y | Z]**

   Unify;

   X = eat(Z) = eat([])

   Y = drink(milk)

   Z = []

9. **f(X, t(b, c)) = f(l, t(Z, c))**

Unify;

X=l

Z=b

10. ancestor(french(jean), B) = ancestor(A, scottish(joe))

Unify;

A = french(jean)

B = scottish(joe)

11. meal(healthyFood(bread), Y) = meal(X, drink(water))

Unify;

X = healthyFood(bread);

Y = drink(water)

12. [H|T] = [a, b, c]

Unify;

H = a

T = [b,c]

13. [H, T] = [a, b, c]

Does Not Unify

LHS has 2 terms, RHS has 3 terms

14. breakfast(healthyFood(bread), egg, milk) = breakfast(healthyFood(Y), Y, Z)

Does Not Unify.

Y cannot be both bread and egg

15. dinner(X, Y, Time) = dinner(jack, cook(egg, oil), Evening)

Unify;

X = jack

Y = cook(egg, oil)

Time = Evening

16. k(s(g), Y) = k(X, t(k))

Unify

X = s(g)

Y = t(k)

17. equation(Z, f(x, 17, M), L*M, 17) = equation(C, f(D, D, y), C, E)

Does Not Unify

D cannot be both x and 17

18. a(X, b(c, d), [H|T]) = a(X, b(c, X), b)

Does Not Unify

b is not a list, so cannot unify with [H|T]

Question 4

1. ? field(hit_transfer, engineering).

Ground

field(hit_transfer, engineering) = field(X,Y) : - course (X,Z), field(Z,Y).

X = hit_transfer; Y = engineering

course(hit_transfer, Z) = course(hit_transfer, mechanical).

Z = mechanical

field(mechanical, engineering) = field(mechanical, engineering).

True.

2. ? lab_number(fine_arts, X).

Non-ground

lab_number(fine_arts, X) = lab_number(fine_arts, 10).

X = 10.

3. ? field(computer, literature).

Ground

field(computer, literature) = field(X,Y) : - course (X,Z), field(Z,Y).

X = computer ; Y = literature

course(computer, Z) does not unify.

False.

4**. ? course(X,Y).**

Non-ground

course(X,Y) = course(hit_transfer, mechanical).

X = hit_transfer;

Y = mechanical.

5. ? student(adrian).

Ground

student(adrian) = student(X) :- student(X,_).

X = adrian.

student(adrian,_) = student(adrian, web_design).

True.

6. ? student(anna, engineering).

Ground

student(anna, engineering) = student(X,Y) :- field(Z,Y), student(X,Z).

X = anna; Y = engineering.

…

field(Z, engineering) = field(X,Y) :- course(X,Z), field(Z,Y).

…

Z = hit_transfer

student(anna, hit_transfer) = student(anna, hit_transfer).

True.

7. ? student(X, engineering).

Non-ground

student(X, engineering) = student(X,Y):- field(Z,Y), student(X,Z).

Y = engineering; X = X;

field(Z, engineering) = … = field(mechanical, engineering) = … =

field(hit_transfer, engineering).

Z = hit_transfer;

student(X, hit_transfer) = student(anna, hit_transfer).

X = anna;

8. ? student(X, fine-arts), course(fine_arts, Y).

Non-ground

student(X, fine-arts) = student(X, Y) :- field(Z,Y), student(X,Z).

Y = fine-arts;

field(Z, fine-arts) = field(X,Y):- course(X,Z), field(Z,Y).

…

field(Z, fine-arts) = field(Z, Y) => fails

False; (No matches found).

9. ? field(_,X).

Non-ground

field(_,X) = field(mechanical, engineering).

X = engineering.

10. ? lab_number(_,X), field(X,Y).

Non-ground

lab_number(_,X) = lab_number(mechanical, 15) => False

lab_number(_,X) = lab_number(fine_arts, 10) => False

lab_number(_,X) = lab_number(X,Z) => False

False; (No matches found).

11. ? lab_number(X,15), field(X,Y).

Non-ground

lab_number(X,15) = lab_number(mechanical, 15).

X = mechanical;

field(mechanical, Y) = field(mechanical, engineering).

Y = engineering;

Output:

X = mechanical.

Y = engineering.

12. ? student(X), !, student(X,_). % note to cut here

Non-ground

student(X) = student(X) :- student(X,_)

student(X,_) = student(anna, hit_transfer).

X = anna;

!

Output:

X = anna;

13. ? student(X), student(X,_), !.

Non-ground

student(X) = student(X) :- student(X,_)

student(X,_) = student(anna, hit_transfer).

X = anna;

student(anna,_) = student(anna, hit_transfer).

!

Output :

X = anna ;

14. ? course(X,_), \+ student(_,X). % \+ is for negation (not)

non-ground

course(X,_) = course(hit_transfer, mechanical).

X = hit_transfer;

student(_,hit_transfer) = student(anna, hit_transfer) => True

\+ True => False

…

X = web_design; (True for adrian) => False

X = design_methods; (True for ava) => False

X = poetry; (True for jack) => False

X = leadership; (True for lee) => False

X = biology;

student(_,biology) = student(X,Y) :- field(Z,Y), student(X,Z).

Y = biology;

field(Z, biology) = field(X,Y) : - course(X,Z), field(Z,Y).

…

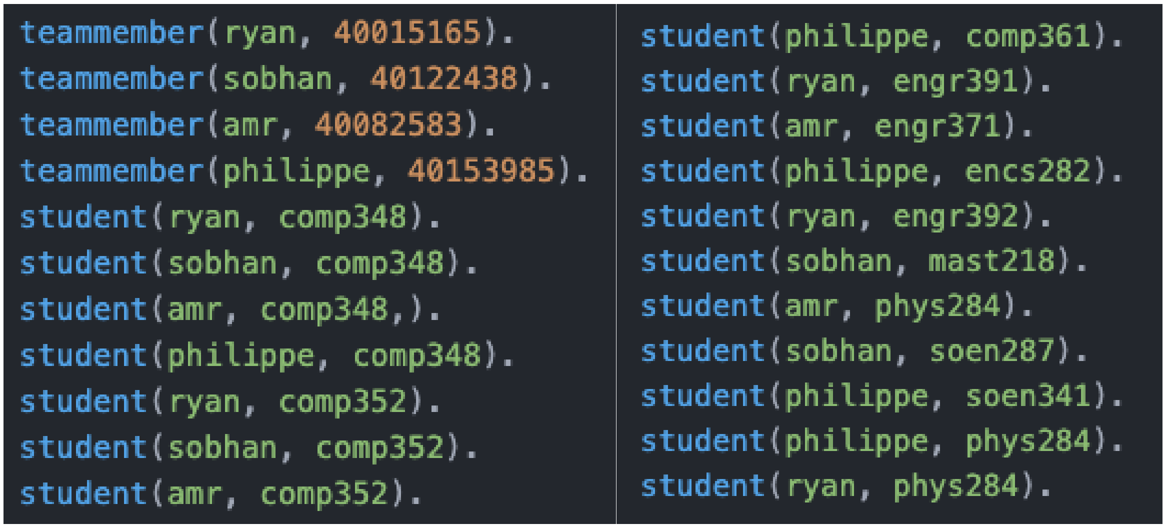
student(_,biology) => false

\+ false => True.

Output:

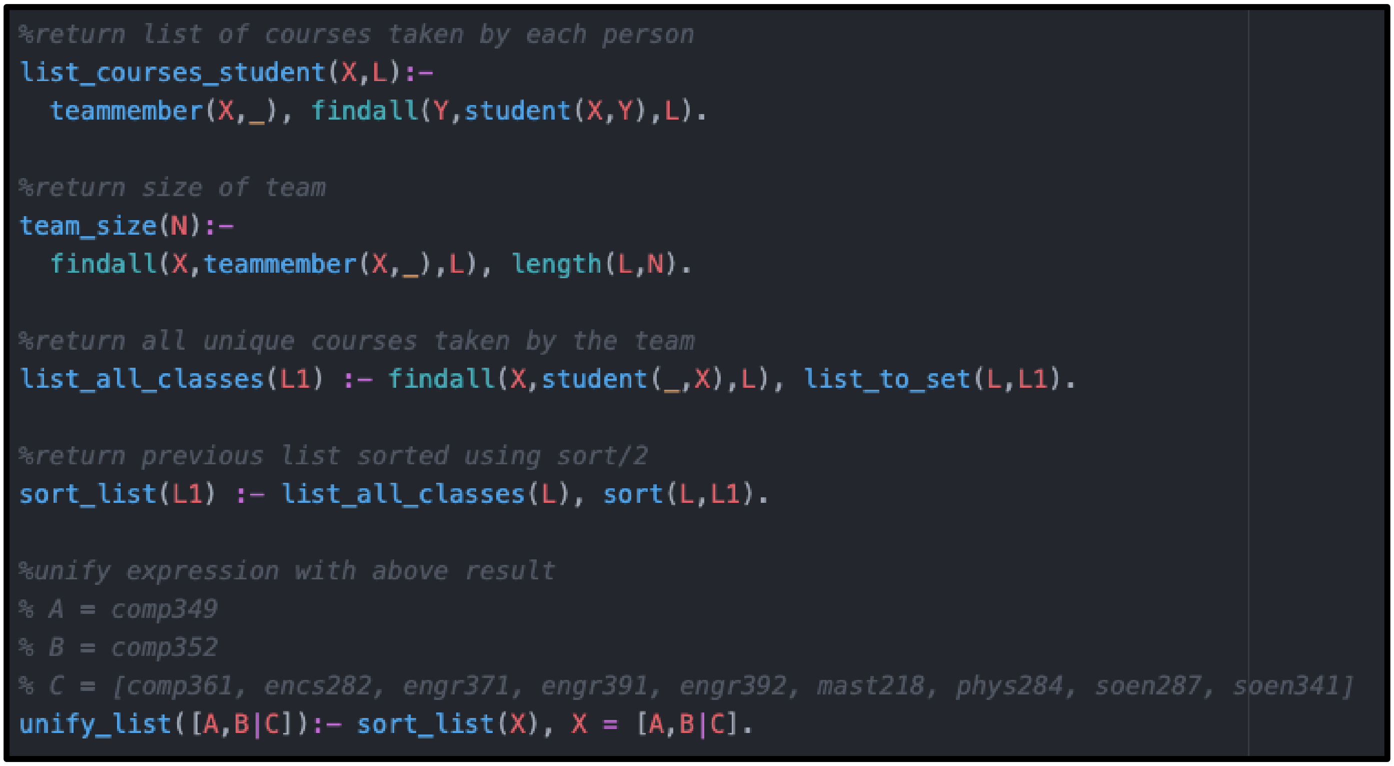
X = biology.

Question 5 – see question5.pl



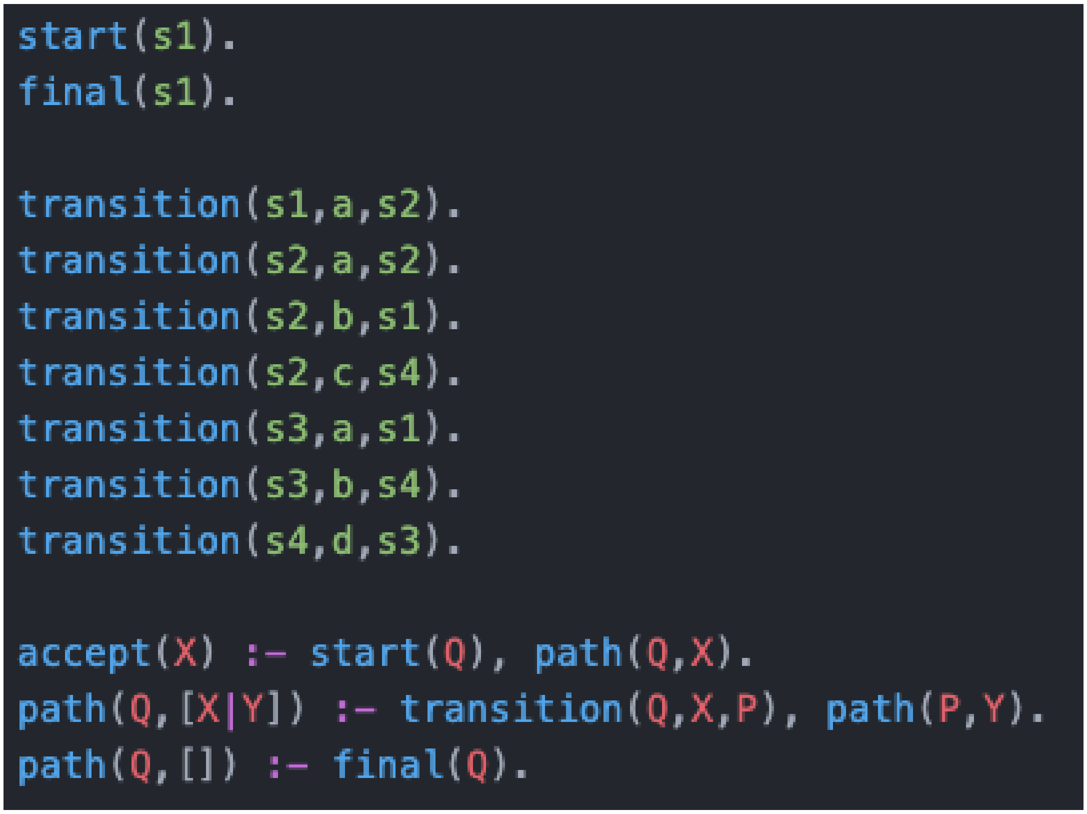
```
teammember(ryan, 40015165).
teammember(sobhan, 40122438).
teammember(amr, 40082583).
teammember(philippe, 40153985).
student(ryan, comp348).
student(sobhan, comp348).
student(amr, comp348,).
student(philippe, comp348).
student(ryan, comp352).
student(sobhan, comp352).
student(amr, comp352).
```

```
student(philippe, comp361).
student(ryan, engr391).
student(amr, engr371).
student(philippe, encs282).
student(ryan, engr392).
student(sobhan, mast218).
student(amr, phys284).
student(sobhan, soen287).
student(philippe, soen341).
student(philippe, phys284).
student(ryan, phys284).
```



```
%return list of courses taken by each person
list_courses_student(X,L):-
  teammember(X,_), findall(Y,student(X,Y),L).

%return size of team
team_size(N):-
  findall(X,teammember(X,_),L), length(L,N).

%return all unique courses taken by the team
list_all_classes(L1) :- findall(X,student(_,X),L), list_to_set(L,L1).

%return previous list sorted using sort/2
sort_list(L1) :- list_all_classes(L), sort(L,L1).

%unify expression with above result
% A = comp349
% B = comp352
% C = [comp361, encs282, engr371, engr391, engr392, mast218, phys284, soen287, soen341]
unify_list([A,B|C]):- sort_list(X), X = [A,B|C].
```

Question 6 – see question6.pl



```
start(s1).
final(s1).

transition(s1,a,s2).
transition(s2,a,s2).
transition(s2,b,s1).
transition(s2,c,s4).
transition(s3,a,s1).
transition(s3,b,s4).
transition(s4,d,s3).

accept(X) :- start(Q), path(Q,X).
path(Q,[X|Y]) :- transition(Q,X,P), path(P,Y).
path(Q,[]) :- final(Q).
```

?- accept([a, a, b]).

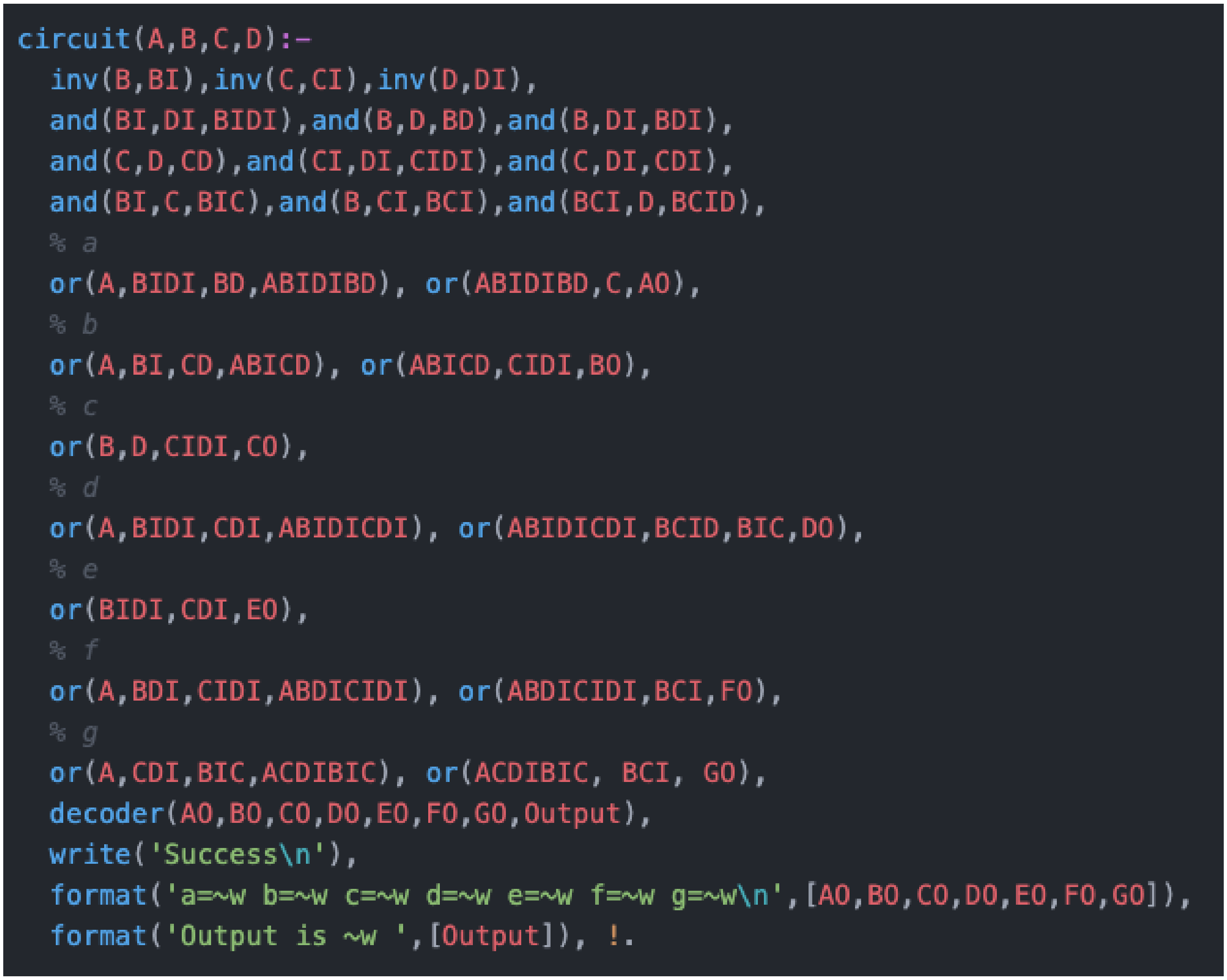
accept([a, a, b]). will return true

It can be seen from the diagram that the FSM starts at s1.

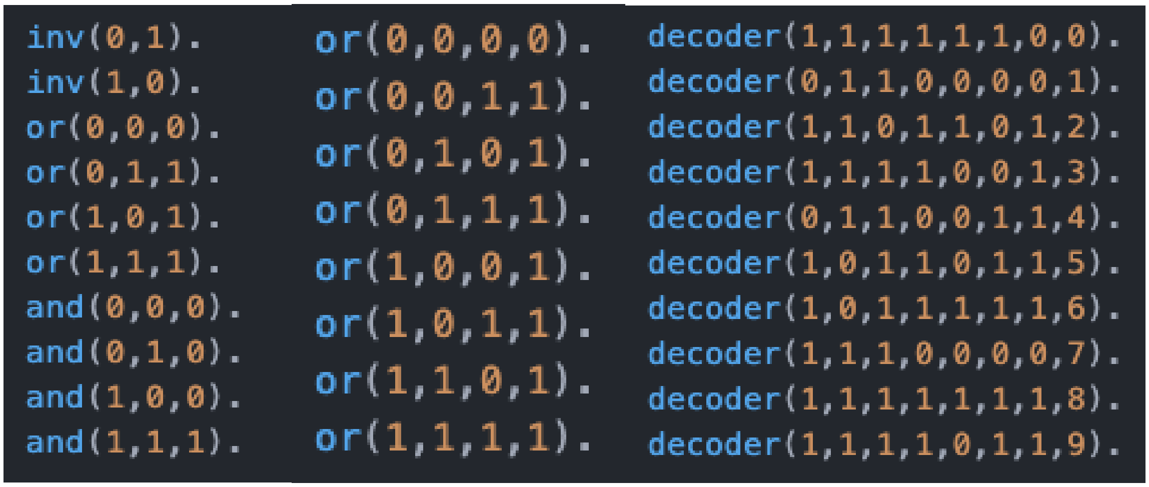
Traversing along the paths a, a and then b will result in the return to s1, which is a final state

Therefore, will return true

Question 7 – see question7.pl



```
circuit(A,B,C,D):-
  inv(B,BI),inv(C,CI),inv(D,DI),
  and(BI,DI,BIDI),and(B,D,BD),and(B,DI,BDI),
  and(C,D,CD),and(CI,DI,CIDI),and(C,DI,CDI),
  and(BI,C,BIC),and(B,CI,BCI),and(BCI,D,BCID),
  % a
  or(A,BIDI,BD,ABIDIBD), or(ABIDIBD,C,AO),
  % b
  or(A,BI,CD,ABICD), or(ABICD,CIDI,BO),
  % c
  or(B,D,CIDI,CO),
  % d
  or(A,BIDI,CDI,ABIDICDI), or(ABIDICDI,BCID,BIC,DO),
  % e
  or(BIDI,CDI,EO),
  % f
  or(A,BDI,CIDI,ABDICIDI), or(ABDICIDI,BCI,FO),
  % g
  or(A,CDI,BIC,ACDIBIC), or(ACDIBIC, BCI, GO),
  decoder(AO,BO,CO,DO,EO,FO,GO,Output),
  write('Success\n'),
  format('a=~w b=~w c=~w d=~w e=~w f=~w g=~w\n',[AO,BO,CO,DO,EO,FO,GO]),
  format('Output is ~w ',[Output]), !.
```



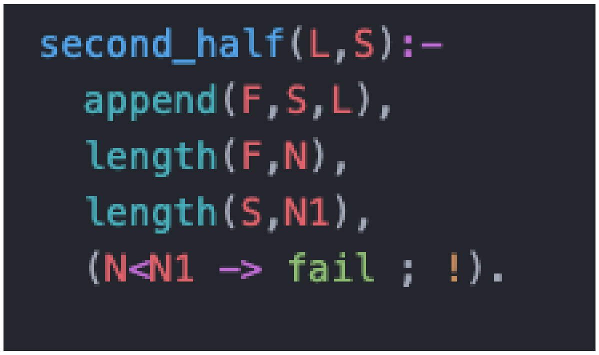
```
inv(0,1).
inv(1,0).
or(0,0,0).
or(0,1,1).
or(1,0,1).
or(1,1,1).
and(0,0,0).
and(0,1,0).
and(1,0,0).
and(1,1,1).
```

```
or(0,0,0,0).
or(0,0,1,1).
or(0,1,0,1).
or(0,1,1,1).
or(1,0,0,1).
or(1,0,1,1).
or(1,1,0,1).
or(1,1,1,1).
```

```
decoder(1,1,1,1,1,1,0,0).
decoder(0,1,1,0,0,0,0,1).
decoder(1,1,0,1,1,0,1,2).
decoder(1,1,1,1,0,0,1,3).
decoder(0,1,1,0,0,1,1,4).
decoder(1,0,1,1,0,1,1,5).
decoder(1,0,1,1,1,1,1,6).
decoder(1,1,1,0,0,0,0,7).
decoder(1,1,1,1,1,1,1,8).
decoder(1,1,1,1,0,1,1,9).
```

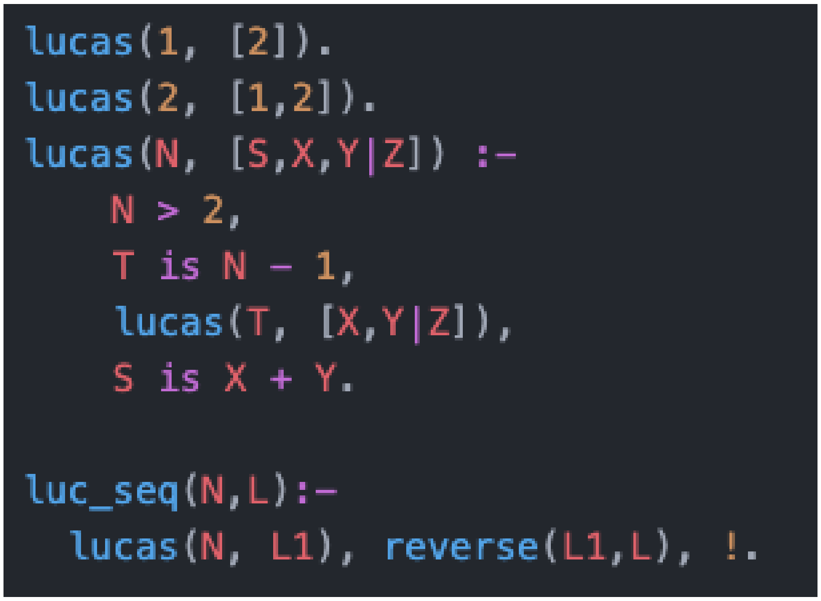
?- circuit(0,1,0,1). → “a=1 b=0 c=1 d=1 e=0 f=1 g=1 Output is 5.”

Question 8 – see question8.pl



```prolog
second_half(L,S):-
  append(F,S,L),
  length(F,N),
  length(S,N1),
  (N<N1 -> fail ; !).
```

Question 9 – see question9.pl



```prolog
lucas(1, [2]).
lucas(2, [1,2]).
lucas(N, [S,X,Y|Z]) :-
    N > 2,
    T is N - 1,
    lucas(T, [X,Y|Z]),
    S is X + Y.

luc_seq(N,L):-
  lucas(N, L1), reverse(L1,L), !.
```