DEERWALK INSTITUTE OF TECHNOLOGY

LAB 4: PROLOG LISTS

(ARTIFICIAL INTELLIGENCE)

SUBMITTED BY:

NAME: SUSHIL AWALE

PROGRAM: B.SC.CSIT (FIFTH SEM)

ROLL NO.: 0540

SECTION: A

SUBMITTED TO:

BIRODH RIJAL

DATE: 13 April 2018

KATHMANDU, NEPAL

2018

1. List operations

i. Write list.

```
writelist ([]) :− nl.
writelist ([H|T]) :− write(H), nl, writelist(T).
```

```
writelist([red, blue, green, white]).
```

```
red
blue
green
white
```

ii. Membership

```
member(X, [X|List]).
member(X, [Element|List]) : −member(X, List).
```

```
member(red, [red, blue, green, white]).
```
true.

```
member(orange, [red, blue, green, white]).
```
false.

iii. Concatenation

```
conc ([], L, L).
```

```
conc ([X|L1], L2, [X|L3]) : −conc(L1, L2, L3).
```

```
conc(['red','blue'], ['green','white'], C).
```

```
C = [red, blue, green, white].
```

iv. Take the n-th element

```
take(1,[H|_],H).
take(N,[_|T],X) : −N1 is N −1,take(N1,T,X).
```

```
take (2, ['red', 'green', 'white'], green).
```
true.

```
take (1, ['red', 'green', 'white'], green).
```
false.

v. Length of a list

```
lengths([],0).
lengths([H|T],N) :− lengths(T,M),N is M + 1.
```

```
lengths([red, green, blue, orange, white], 5).
```
true.

```
lengths([red, green, blue, orange, white], 3).
```
false.

vi. Sum of elements

```
sum([],0).
sum([X|L],Sum) :− sum(L,SL),Sum is X + SL.
```

```
sum([5, 10, 15], 30).
```
true.

vii. Reverse of list

```
reverse([],X,X).
reverse([X|Y],Z,W) :− reverse(Y,[X|Z],W).
```

```
reverse([1, 2, 3], R).
R = [3, 2, 1]
```

viii. Append

```
append([],L,L).
append([H|T],L,[H|TL]) :− append(T,L,TL).
```

```
append([1, 2, 3], [4], [1, 2, 3, 4]).
```
true.

```
append([1, 2, 3], [3], [1, 2, 3]).
```
false.

2. DFA with input as list

DFA with input as list

```
t(0,a,1). t(0,b,2).
t(1,a,1). t(1,b,1).
t(2,a,2). t(2,b,2).
startstate(0). % 0 is a starting state
finalstate(1). % 1 is a final state
```

Implement a predicate checkinput(Start,Input) that checks if a word (here, input) given as a list (e.g. [a,b,b,a,b]) is accepted by the DFA starting from a start state (here State).

- - dfa.pl - -

```
t(0,a,1).
t(0,b,2).
t(1,a,1).
t(1,b,1).
t(2,a,2).
t(2,b,2).
checkinput(Start, []) :- Start is 1.
checkinput(Start, [A|B]) :- t(Start, A, Next), checkinput(Next, B).
```

- - - -

```
checkinput(0, [a]).
```
true.

```
checkinput(1, [a]).
```
true.

```
checkinput(1, [b]).
```
true.

```
checkinput(0, [a, b]).
```
true.

3. Using structures

family(

person(homer,simpson,date(7,may,1960),works(inspector,6000)),

person(marge,simpson,date(7,may,1965),housewife),

[person(bart,simpson,date(7,may,1967),student),

person(lisa,simpson,date(7,may,1965),student)].

Using the family predicate, implement the following relation as rules:

A. **husband(X) : true. if X is someone's husband**

husband(H) :- family(person(H,_,_,_),_,_).

husband (homer).
true.

B. **wife(X) : true. if X is someone's wife**

wife(W) :- family(_,person(W,_,_,_),_).

wife(marge)
true.

C. **child(X) : true. if X is someone's child**

child(X) :- family(_,_,Children), member(person(X,_,_,_), Children).

child(bart).
true.

D. **exists(Person) : true. if the person is in the database**

exists(Person) :- husband(Person);wife(Person);child(Person).

exists(lisa).
true.
exists(sushil).
false.