Moti Begna

CSCI 5106

Homework 8

Problem 1

Using only the *append* relation, formulate queries to determine the following:

a) The third element of a list

Solution: append([_, _, X|_], [_], L).

d) Whether a list is a concatenation of three copies of the same list.

Solution - append(X, Y, L), append(X, X, Y).

/* returns false on failure, assigns X and Y on success */

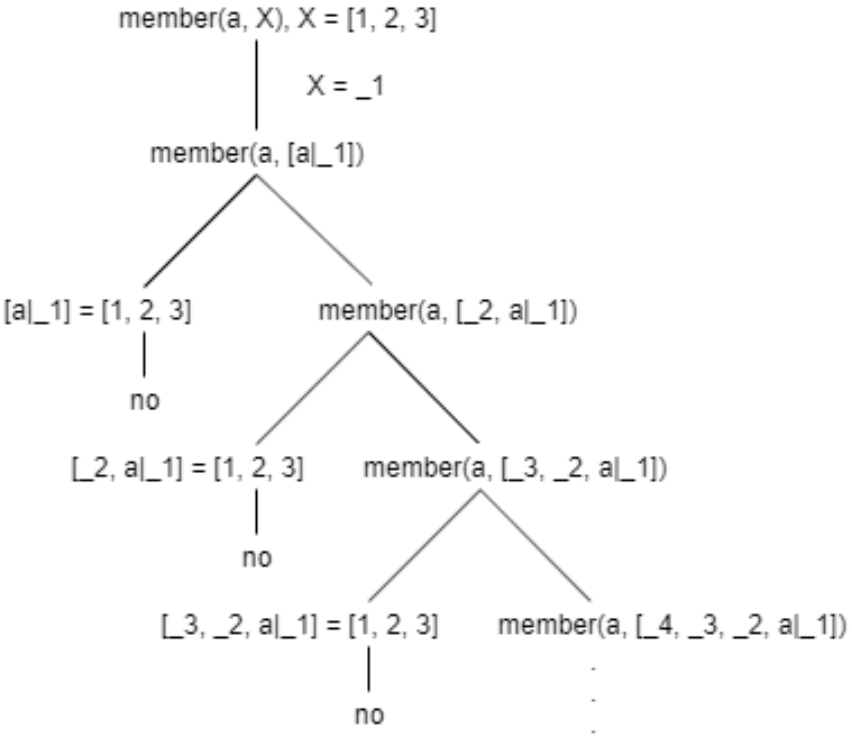
Problem 2

```
/* Problem 2.1: Is a list L1 a permutation of another list L2? */

permutation(L1, L2) :- length(L1, X1), length(L2, X2), X2 is X1, isMember(L1, L2).

isMember([], _).
isMember([H|T], L2) :- member(H, L2), isMember(T, L2).

/* Problem 2.2: Does a list have an even number of elements? */

evenList(L) :- length(L, X), 0 is mod(X, 2).
```

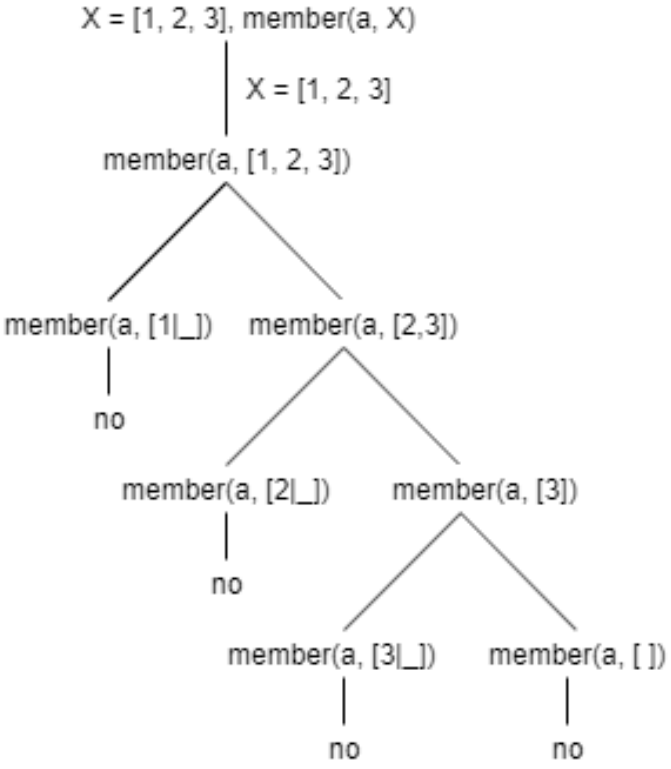
Problem 3

In part a, the prolog query first assigns the variable X to a specified list, and then runs a member relation on the value ‘a’ and X, while part b does this in the opposite direction. Based on the definitions of the member relations from the exercises, the member query in part a would match the value ‘a’ to the head of the list X, and then recurse over the tail of the list until matching against an empty list. In each instance, the match would fail since ‘a’ does not exist within the list X. In any case, the query as a whole would terminate. This greatly differs from part b, however, since the list X is constantly being reassigned. Here, a call to member on the value ‘a’ and X assigns the list X, firstly, as a list with ‘a’ as it’s head. Then, instead of the query X = [1, 2, 3] assigning the list to X, it would check to see if the list is *equal* to X. And because this would end up failing, a call to member would once again be called, only this time, ‘a’ becomes the second element of X. This procedure would run indefinitely, constantly assigning ‘a’ to the next index of the list X, and then checking its equality with the list [1, 2, 3]. This is due to the fact that X is not assigned when member(a, X) is called, thus X can be any list so long as a exists somewhere within it.

Part a)


member(a, X), X = [1, 2, 3]
X = _1
member(a, [a|_1])
[a|_1] = [1, 2, 3]
member(a, [_2, a|_1])
no
[_2, a|_1] = [1, 2, 3]
member(a, [_3, _2, a|_1])
no
[_3, _2, a|_1] = [1, 2, 3]
member(a, [_4, _3, _2, a|_1])
no

Part b)


X = [1, 2, 3], member(a, X)
X = [1, 2, 3]
member(a, [1, 2, 3])
member(a, [1|_])
member(a, [2,3])
no
member(a, [2|_])
member(a, [3])
no
member(a, [3|_])
member(a, [ ])
no
no

Problem 4

```
/* Problem 4.1 In order to describe an abstract syntax representation for
logical expressions, we can use a linear notation in which, using
the following predicates, we can define the logical operators and, or, not. Thus,
we can represent logical expressions such as and(not(var(a)), or(var(a), var(b)))*/

and(E1,E2):- E1, E2.
or(E1,E2):- E1; E2.
not(E):- \+ E.

/* Problem 4.2:  The assignment of truth values for propositional variables
 within a logical expression can be represented as a list of "tuples" containing
 the name of the variable and the truth assignment for it. As an example,
 [(p, true), (q, true)] is an assignment list whereby the propositional
 variables q and p within some given expression both have the value true.*/

/*Problem 4.3 A predicate that returns the truth value of a logical Expression E
and an assignment L of truth values for propositional variables. Example,
istrue(and(var(a), not(var(b))), [(a, true), (b, false)]) returns true.*/

istrue(E, L):- evaluate(E, L).

evaluate(and(E1,E2), L):- and(evaluate(E1, L), evaluate(E2, L)).
evaluate(or(E1,E2), L):- or(evaluate(E1, L), evaluate(E2, L)).
evaluate(not(E), L):- not(evaluate(E, L)).
evaluate(var(E), L):- member((E, true), L).

/*Problem 4.4 A predicate that takes a logical expression E and returns a list of
all the propositional variables appearing in that list. */

varsOf(E, LST):- varList(E, LST).

varList(and(E1,E2), LST):- varList(E1,E1LIST), varList(E2,E2LST), union(E1LIST,E2LST,LST).
varList(or(E1,E2), LST):- varList(E1,E1LIST), varList(E2,E2LST), union(E1LIST,E2LST,LST).
varList(not(E), LST):- varList(E,ELIST), union([],ELIST,LST).
varList(var(E), LST):- union([], [E], LST).
```

```
/*Problem 4.5 A predicate that takes a logical expression E and succeeds when E
 is a tautology, i.e. when every assignment for the expression evaluates to true. */

isTaut(E) :- not(isNotTaut(E)).
isNotTaut(E) :- varsOf(E, Varlist), isAssignment(Varlist, L, [true, false]),
   not(istrue(E, L)).

isAssignment([VAR],LST, TFLIST):- member(ASSGN, TFLIST), append([], [(VAR, ASSGN)], LST).
isAssignment([VAR|VARLIST],LST, TFLIST):- member(ASSGN, TFLIST),
   isAssignment(VARLIST,RETLST,TFLIST), append([(VAR, ASSGN)], RETLST, LST).
```