Patrick Charles prc9219
CMPS 450
Assignment 2 – Logic Programming
Fall 15

Logic Programming

I.1 Description of Assignment Goals

To familiarize students with the Logical programming concepts and teach students how to change their mentality from imperative programming to logical.

II.1 Language and platform

II.2 Scheme functions

Written in notepad++ v6.5.5 on Windows 10 and run on SWI-Prolog v7.2.0, 64 bit

III.1 Description of Functions

Overview: Functions to be written in the logical language based program, SWI-Prolog and is designed to familiarize students with logical programming concepts and syntax.

III.1.1 relationships

Description:
Defines all mother/father relationships via 2 parameters for 3 family generations. Grandparent, parent, sibling, and cousin are defined via logical clauses.

III.1.2 Maximum/ Minimum of a list of integers

Description:
minimum value function: receives a list of integers and tests if the head is less than the min value so far. If so, the function calls itself with the tail of the list and the new min value and goes through the list again.
Maximum value function: receives a list of integers and tests if the head is greater than the max value so far. If so, the function calls itself with the tail of the list and the new min value and goes through the list again.
Minmax function: is called with 2 parameters which are function calls to the minimum and maximum value functions and returns the values.

III.1.3 Collect user input until zero is reached

Description: reads user input into a list until a zero is reached in which case, it will stop reading user input and output the list of values entered in a list in order of input

III.1.4 Mergesort

Description: receives a list which is split into two lists using split functions. The lists are then sorted recursively based on iterating through the values to determine the least value of the lists and ordered as such. After the lists are sorted, the two lists are merged using, merge function and sorted again.

IV.1 Source code for Prolog functions

IV.1.1 Relationships

filename: 1-relationships.pl

```
%grandparent
%X is grandparent of Z if X is a parent of Z's parent

grandparent(X, Z) :-
   parent(X, Y),
   parent(Y, Z).

%parent
%A is parent of B if  A is  father or mother of B
%according to fact database

parent(A,B) :- father(A,B).
parent(A,B) :- mother(A,B).

%sibling
%have at least one common parent
%test if siblings same person

sibling(X,Y) :- parent(Z,X),
        parent(Z,Y), not(X=Y).

%cousin
%child1 and child2 are cousins if their parents are siblings

cousin(Child1,Child2) :-
   parent(Y1,Child1),
   parent(Y2,Child2),
   sibling(Y1,Y2).


%Fact database

%father
```

```
father(harry, richard).
father(harry, catherine).
father(robert, theresa).
father(robert, elaine).
father(robert, robertjr).
father(richard, patrick).
father(richard, chris).
father(roger, thomas).
father(roger, sara).
father(gary, russell).
father(gary, melissa).
father(robertjr, michael).
father(robertjr, daniel).

%mother

mother(emelda, richard).
mother(emelda, catherine).
mother(doralea,theresa).
mother(doralea, robertjr).
mother(doralea, elaine).
mother(theresa, patrick).
mother(theresa, chris).
mother(catherine,thomas).
mother(catherine, sara).
mother(elaine, russell).
mother(elaine, melissa).
mother(linda, michael).
mother(mother, daniel).
```

IV.1.2 Maximum/ Minimum of a list of integers

filename: maxmin.pl

```
%minmax
%takes list and 2 variables
%computes max and min using helper functions
%and outputs the values

minmax([H|T], Min, Max) :-
        max([H|T], Max), min([H|T], Min).

%max

%takes list and variable(M) to store max
```

```
max([H|T], M) :-
	max(T, H, M).

%puts tail in list. sets head as max.
max([], M, M).

%takes tail as list, compares head of tail to previous tail
%if =<, recursively iterates through list until higher number found
max([H|T], Y, M) :-
	H =< Y, max(T, Y, M).

%if head of tail >previous head, new max is setand
%iterates through list
max([H|T], Y, M) :-
	H > Y, max(T, H, M).

%min

%takes list and variable(M) to store min
min([H|T], M) :-
	min(T, H, M).

%puts tail in list. sets head as min.
min([], M, M).

%takes tail as list, compares head of tail to previous tail
%if =<, recursively iterates through list until lower number found
min([H|T], Y, M) :-
	H =< Y, min(T, H, M).

%if head of tail >previous head, new min is set and
%iterates through list
min([H|T], Y, M) :-
	H > Y, min(T, Y, M).
```

<u>**IV.1.3 Collect user input until zero is reached**</u>

<u>filename</u>: 3-input until zero.pl

```
%start prompt, call read1
start :-
	write('enter number : '),
	read1.

%read number into list
read1:-
	read(X),
        append( [] , X , X),

        %check if 0
        check(X),
        write(X).

%if not 0, read another number. else, stop
check(X):-
        (X  \=  0
        -> read1);
        !.
```

<u>IV.1.5 Mergesort</u>

<u>filename</u>: mergesort.pl

```
%empty list and list with one element returned
mergesort([],[]).
mergesort([X],[X]).

% X = list
% Y = H of split list
% Z = Z of split list
% S, S1, S2 = vars to store sorted lists
% L1, L2 = vars to store split lists
% M = merged list so far
% ZX, ZY = tails of X and Y respectively

%if at least2 elements,split list
mergesort([X,Y|Z],S) :-
        %split list into L1,L2
        split1([X,Y|Z],L1,L2),

        %sort the list1
        mergesort(L1,S1),

        %sort list2
        mergesort(L2,S2),

        %merge 2 sorted lists into S
        merge(S1,S2,S).

%empty list returns empty sorted list
split1([],[],[]).

%single element list returns empty sorted list
split1([X],[X],[]).
```

```
%split list into 2 with first 2 elements as heads of their own list
split1([X,Y|Z],[X|ZX],[Y|ZY]) :-
        split1(Z,ZX,ZY).

%single element and empty lists merge into single element list
merge(X,[],X).
merge([],Y,Y).

%if head <= tail, merge tail of X and tail of Y into M list
merge([X|ZX],[Y|ZY],[X|M]) :-
        X =< Y,
        merge(ZX,[Y|ZY],M).

%%if head > tail, merge head of X and tail of Y into M list
merge([X|ZX],[Y|ZY],[Y|M]) :-
        X > Y,
        merge([X|ZX],ZY,M).
```

<u>IV.1 Test Case for Prolog functions</u>

<u>IV.1.1 Relationships</u>

1. Load file ‘1-relationship.pl’
2. Test grandparent rules
3. Test parent rules
4. Test sibling rules
5. Test cousin rules

<u>Grandparent test:</u>
Query: grandparent(harry, patrick).
Expected result: true
Query: grandparent(harry, michael).
Expected result: false
Query: grandparent(robert, michael).
Expected result: true
Query: grandparent(robert, thomas).
Expected result: false

<u>Results:</u>
```
1 ?- grandparent(harry,patrick).
true .

2 ?- grandparent(harry,michael).
false.
```

```
3 ?- grandparent(robert,michael).
true .

4 ?- grandparent(robert, thomas).
false.
```

<u>Parent test:</u>
Query: parent(harry, richard).
Expected result: true
Query: parent(richard, patrick).
Expected result: true
Query: parent(patrick, richard).
Expected result: false
Query: parent(richard, harry).
Expected result: false

<u>Results:</u>

```
5 ?- parent(harry, richard).
true .

6 ?- parent(richard, patrick).
true .

7 ?- parent(patrick, richard).
false.

8 ?- parent(richard, harry).
false.
```

<u>Sibling test:</u>
Query: sibling(patrick, chris).
Expected result: true
Query: sibling(patrick, michael).
Expected result: false
Query: sibling(richard, theresa).
Expected result: false
Query: sibling(theresa, robertjr).
Expected result: true

<u>Results:</u>

```
9 ?- sibling(patrick, chris).
true .

10 ?- sibling(patrick, michael).
false.
```

```
11 ?- sibling(richard, theresa).
false.

12 ?- sibling(theresa, robertjr).
true
```

Cousin test:

Query: cousin(patrick, michael).
Expected result: true
Query: cousin(thomas, michael).
Expected result: false
Query: cousin(michael, sara).
Expected result: false
Query: cousin(patrick, richard).
Expected result: false
Query: cousin(patrick, sara).
Expected result: true

Results:

```
13 ?- cousin(patrick, michael).
true .

14 ?- cousin(thomas, michael).
false.

15 ?- cousin(michael, sara).
false.

16 ?- cousin(patrick, richard).
false.

17 ?- cousin(patrick, sara).
true
```

Identify relations test:

Query: grandparent(X,patrick).
Expected result: harry
Query: parent(X,patrick).
Expected result: Richard
Query: sibling(X,patrick).
Expected result: chris
Query: cousin(X,patrick).
Expected result: Michael

Results:

```
18 ?- grandparent(X,patrick).
X = harry .

19 ?- parent(X,patrick).
X = richard .

20 ?- sibling(X,patrick).
X = chris .

21 ?- cousin(X,patrick).
X = michael
```

IV.1.2 Maximum/ Minimum of a list of integers

1. Load file ‘2-maxmin.pl’
2. Test min function on a list
3. Test max function on a list
4. Test minmax function on a list

Query: min([5,4,3,0,1,2,43,65,99],X).
Expected result: X = 0
Query: max([5,4,3,0,1,2,43,65,99],X).
Expected result: X = 99
Query: minmax([5,4,3,0,1,2,43,65,99],X,Y).
Expected result: X = 0, Y = 99

Results:

```
1 ?- min([5,4,3,0,1,2,43,65,99],X).
X = 0 .

2 ?- max([5,4,3,0,1,2,43,65,99],X).
X = 99 .

3 ?- minmax([5,4,3,0,1,2,43,65,99],X,Y).
X = 0,
Y = 99
```

IV.1.3 Collect user input until zero is reached

1. Load file ‘3-input until zero.pl’
2. Input numbers followed by a ‘.’ Ending with a 0 to finish

3. Input characters followed by a ‘.’ Ending with a 0 to finish
4. Input strings followed by a ‘.’ Ending with a 0 to finish

Query: start(X).
Numbers to enter: 3,5,7,8,9,0
Expected result: X = [3, 5, 7, 8, 9]
Query: start(X).
Numbers to enter: z,4,3,a,bb,cdb,0
Expected result: X = [z, 4, 3, a, bb, cdb]
Query: start(X).
Numbers to enter: hello, my, name, is, prolog,0
Expected result: X = [hello, my, name, is, prolog]

<u>Results:</u>

```
1 ?- start(X).
enter number : 3.
enter number : |: 5.
enter number : |: 7.
enter number : |: 8.
enter number : |: 9.
enter number : |: 0.

X = [3, 5, 7, 8, 9] .

2 ?- start(X).
enter number : z.
enter number : |: 4.
enter number : |: 3.
enter number : |: a.
enter number : |: bb.
enter number : |: cdb.
enter number : |: 0.

X = [z, 4, 3, a, bb, cdb] .

3 ?- start(X).
enter number : hello.
enter number : |: my.
enter number : |: name.
enter number : |: is.
enter number : |: prolog.
enter number : |: 0.

X = [hello, my, name, is, prolog] .
```

V.1.4 Mergesort

1. Load ‘4-mergesort.pl’
2. Sort empty list
3. Sort list with one element
4. Sort list of many elements

Query: mergesort([],X).
Expected result: X = []
Query: mergesort([666],X).
Expected result: X = [666]
Query: mergesort([100,35,25,65,50,15,10,0,1],X).
Expected result: X = [0,1,10,15,25,35,50,65,100]

Results:

```
1 ?- mergesort([],X).
X = [].

2 ?- mergesort([666],X).
X = [666] .

3 ?- mergesort([100,35,25,65,50,15,10,0,1],X).
X = [0, 1, 10, 15, 25, 35, 50, 65, 100] .
```

V.1.5 Functional Programming vs. Logical Programming

See file “5-functional vs logic prgramming.txt”