ECS 140A Discussion 5

Ty Feng, Chris Fernandez

2:10-3:50pm Thursday, Sept 7 2023

UCDAVIS

Today's agenda

Homework 4

Prolog Exercises

Erlang

Q&A

HW4 Submission

Autograder will be used for HW4, so be sure to name your files correctly:

- Q1-4 -> “hw4.pl”
- Q5 -> “sudoku.pl”

Make sure `swipl hw4/sudoku.pl` can run your programs without producing any errors.

Hint for HW 4, problem 4

- Base case: both L1 and L2 are empty, return true.
- Check if L1 and L2 are of the same length
- Split L1 into head H and tail T
- If head H of L1 is a member in L2
- Remove first instance of element H from L2 and return the result as L3
- Recurse on T and L3

HW4 Problem 5: Sudoku

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Fill in the grid so that every row, every column, and every 3x3 box contains the digits 1 through 9.

HW4 Problem 5: Sudoku

9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Constraints?

Each row has valid numbers and they are unique from each other

Each column has valid numbers and they are unique from each other

Each 3x3 box has valid numbers and they are unique from each other

Minidoku

Let's talk about a smaller version of the problem. Given a list of three elements where some of the elements are numbers and some are empty variables:

[3,Y,1] for example

can you find a way to fill the variables with numbers so that all the numbers are taken from the set of valid numbers {1, 2, 3} and are all different?

How would you make Prolog do it?

Minidoku

```
sudoku(L) :- [X1,X2,X3] = L,

             worthy([X1,X2,X3]).
```

What does it mean for the values to be worthy? The problem description tells us that the values must be valid (drawn from some specified set of values) and that they must all be different.

Minidoku

```
sudoku(L) :- [X1,X2,X3] = L,
                worthy([X1,X2,X3]).
  worthy(L) :- valid(L),diff(L).
```

What does it mean to be valid?

Minidoku

```
sudoku(L) :- [X1,X2,X3] = L,
                worthy([X1,X2,X3]).
  worthy(L) :- valid(L),diff(L).
  valid([H]) :- validval(H).
  valid([H|T]) :- validval(H),valid(T).
```

What is validval?

Minidoku

```
sudoku(L) :- [X1,X2,X3] = L,
             worthy([X1,X2,X3]).

 worthy(L) :- valid(L),diff(L).

 valid([H]) :- validval(H).

 valid([H|T]) :- validval(H),valid(T).

 validval(1).

 validval(2).

 validval(3).
```

What does it mean for all the elements of the list to be different?

Minidoku

```
sudoku(L) :- [X1,X2,X3] = L,
             worthy([X1,X2,X3]).

worthy(L) :- valid(L),diff(L).
valid([H]) :- validval(H).
valid([H|T]) :- validval(H),valid(T).
validval(1).
validval(2).
validval(3).

diff([H]).
diff([H|T]) :- not(member(H,T)),diff(T).
member(H,[H|T]).
member(X,[H|T]) :- member(X,T).
```

9x9 Sudoku

Based on Minidoku, what you need to do are:

1. Take in a nested list of 9 lists of 9 elements
2. Check every row is `worthy`
3. Check every column consists of unique numbers
    a. Why? No need to check `valid` because we already checked in step 2
4. Check every 3x3 box consists of unique numbers
    a. Why? No need to check `valid` because we already checked in step 2
5. Valid numbers are 1..9

That's it! Easy, right?

Prolog practice

Navigate to tyfeng.com/ecs140a/discussion5.html

To run Prolog programs:

$ swipl filename.pl

?- trace. % trace your code step by step. Useful to see recursion and backtracking

?- halt. % exit out of swipl

?- procedureName(data, X). % solve for X. X is a variable whose value will be returned by the program.

Prolog practice

Problem 1: Write a procedure

same_length(L1, L2)

which returns true if both lists L1 and L2 have the same number of elements. For example,

?- same_length([a,b,c],[1,2,3])

true.

?- same_length([a,b,c],[1, 2]).

false.

Prolog practice

Problem 1: Write a procedure

same_length(L1, L2)

which returns true if both lists L1 and L2 have the same number of elements. For example,

?- same_length([a,b,c],[1,2,3])

true.

Solution:

```
same_length([], []).
same_length([_|T1], [_|T2]) :- same_length(T1, T2).
```

Prolog practice

Problem 2: Write a procedure

sum_list(L1, S)

which returns true if S is the sum of all elements in the list L1. For example,

?- sum_list([1,2,3,4,5], 15)

true.

This should also work:

?- sum_list([1, 2, 3, 4, 5], X).

X = 15.

Prolog practice

Problem 2: Write a procedure

sum_list(L1, S)

which returns true if S is the sum of all elements in the list L1. For example,

?- sum_list([1,2,3,4,5], 15)

true.

Solution:

```
sum_list([], 0).
sum_list([H|T], Sum) :-
    sum_list(T, Rest),
    Sum is H + Rest.
```

Prolog practice

Problem 3: Write a procedure

even_elements(L1, L2)

which returns true if list L2 contains every second element of list L1. For example,

?- even_elements([a,b,c,d,e,f],[b,d,f])

true.

This should also work:

?- even_elements([1,2,3,4,5,6], X).

X = [2, 4, 6].

Prolog practice

Problem 3: Write a procedure

even_elements(L1, L2)

which returns true if list L2 contains every second element of list L1. For example,

?- even_elements([a,b,c,d,e,f],[b,d,f])

true.

Solution:

```
even_elements([], []).
even_elements([_], []).
even_elements([_, Y| T1], [Y|T2]) :- even_elements(T1, T2).
```

Prolog practice

Problem 4: Write a procedure

triple(L1, L2)

which returns true if every element in list L1 appears thrice in L2, according to the pattern in the following example:

```
?- triple([a,b,c],[a,a,a,b,b,b,c,c,c]).
true.
```

Note that the elements that are duplicated are also grouped together. These cases should also work:

```
?- triple(X,[a,a,a,b,b,b,c,c,c]).
X = [a, b, c].
?- triple([a,b,c],X).
X = [a, a, a, b, b, b, c, c, c].
```

Prolog practice

Problem 4: Write a procedure

triple(L1, L2)

which returns true if every element in list L1 appears thrice in L2, according to the pattern in the following example:

```
?- triple([a,b,c],[a,a,a,b,b,b,c,c,c]).
true.
```

Solution:

```
triple([], []).
triple([H|T1], [H,H,H|T2]) :- triple(T1, T2).
```

Prolog practice

Problem 5: Write a procedure

remove_first_elem(X, L1, L2)

which returns true if list L2 is the result of removing the first instance of element X from list L1.

For example,

remove_first_elem(1, [2, 1, 3, 1, 3], [2, 3, 1, 3])

returns true as 1 is removed from the original list [2, 1, 3, 1, 3] to form the new list [2, 3, 1, 3].

Prolog practice

Problem 5: Write a procedure

remove_first_elem(X, L1, L2)

which returns true if list L2 is the result of removing the first instance of element X from list L1.

Solution:

```
remove_first_elem(_, [], []).
remove_first_elem(X, [X|T], T).
remove_first_elem(X,[H|T1],[H|T2]) :- remove_first_elem(X,T1,T2).
```

Prolog practice

Problem 6: Write a procedure

remove_all_elements(X, L1, L2)

which returns true if list L2 is the result of removing all occurrences of element X from list L1. For example,

?- remove_all_elements(a, [a,b,c,a,d,a], [b,c,d])

true.

This should also work:

?- remove_all_elements(a, [a,b,c,a,d,a], X).

X = [b, c, d].

Prolog practice

Problem 6: Write a procedure

remove_all_elements(X, L1, L2)

which returns true if list L2 is the result of removing all occurrences of element X from list L1.

Solution:

```
remove_all_elements(_, [], []).
remove_all_elements(X, [X|T], L2) :- member(X, [X|T]),
remove_all_elements(X, T, L2).
remove_all_elements(X, [H|T], [H|L2]) :- not(member(X, [H|T])),
remove_all_elements(X, T, L2).
```

Erlang

- Erlang is a concurrent, functional programming language designed for building scalable, fault-tolerant, and highly available systems.
- Erlang excels at handling concurrent processes, making it ideal for building real-time, distributed systems.
- It embraces functional programming principles, allowing for clean, maintainable code.
- Erlang was built for distributed computing, making it suitable for building distributed and fault-tolerant applications.
- Erlang is widely used in telecommunications, gaming, instant messaging, and many other industries where high concurrency and reliability are critical.

Download Erlang: https://www.erlang.org/

Syntax

- Erlang files will have the .erl extension.
- File should contain name of module at the start. Module name should be same as name of the file.
- Functions that are written should be exported out in order to use outside the module.
- When defining a function, specify it's arity(number of arguments) as well.
- In order to run the file, save the program and open terminal in that folder.
- Type erl to start erlang
- Load file and compile with c(module-name). (remember to use the period)
- Call the function with module-name:function-name.
- You can have multiple functions in the same file.

Example1: addnumbers.erl

- A program to add numbers
  -module(addnumbers).
  -export([add2/2]).
  add2(A,B) -> A+B.

Example1: addnumbers.erl

- A program to add numbers
  -module(addnumbers).
  -export([add2/2]).
  add2(A,B) -> A+B.

- C:\Users\chris fernandez\Downloads>erl
  1> c(addnumbers).
  {ok,addnumbers}
  2> addnumbers:add2(3,4).
  7

Example2: addnumbers1.erl

- ```
  -module(addnumbers1).
  -export([add2/2]).
  -export([add3/3]).
  -export([add4/4]).
  add2(A,B) -> A+B.
  add3(A,B,C) -> A+B+C.
  add4(A,B,C,D) -> A+B+C+D.
  ```
  ```

Example2: addnumbers1.erl

- -module(addnumbers1).
  -export([add2/2]).
  -export([add3/3]).
  -export([add4/4]).
  add2(A,B) -> A+B.
  add3(A,B,C) -> A+B+C.
  add4(A,B,C,D) -> A+B+C+D.

- 1> c(addnumbers1).
  {ok,addnumbers1}
  2> addnumbers1:add2(1,2).
  3
  3> addnumbers1:add3(1,2,3).
  6
  4> addnumbers1:add4(1,2,3,4).
  10

Example3: factorial.erl

- -module(factorial).
  -export([fact/1]).
  fact(0) -> 1;
  fact(N) when N>0 -> N * fact(N-1).

- 1> c(factorial).
  {ok,factorial}
  2> factorial:fact(3).
  6
  3> factorial:fact(4).
  24

- Note: Use semicolons between cases, not periods.

Example4: fibonacci.erl

- -module(fibonacci).
  -export([fib/1]).
  fib(0) -> 0;
  fib(1) -> 1;
  fib(N) when N>1 -> fib(N - 1) + fib(N - 2).

- 1> c(fibonacci).
  {ok,fibonacci}
  2> fibonacci:fib(0).
  0
  3> fibonacci:fib(1).
  1
  4> fibonacci:fib(2).
  1
  5>fibonacci:fib(3).
  2

Practice: mylist.erl

1) mycount

Write a function, mycount, that takes in a list and a value and returns the number of times the value appears in the list.
mylist:mycount([1,2,3,1,2,1,3,1], 1) -> 4

Practice: mylist.erl

1) mycount

Write a function, mycount, that takes in a list and a value and returns the number of times the value appears in the list.
mylist:mycount([1,2,3,1,2,1,3,1], 1) -> 4

```
-module(mylist).
-export([mycount/2]).
mycount([], _) -> 0;
mycount([H|T], V) when H == V ->1 + mycount(T, V);
mycount([_|T], V) -> mycount(T, V).
```

Practice: mylist.erl

2) mytake

Write a function, mytake, that takes in a positive integer n and a list, returns a new list with the first n elements of the original list.
mylist:mytake(3,[1,2,3,1,2,1,3,1]) -> [1,2,3]

Practice: mylist.erl

2) mytake

Write a function, mytake, that takes in a positive integer n and a list, returns a new list with the first n elements of the original list.
mylist:mytake(3,[1,2,3,1,2,1,3,1]) -> [1,2,3]

```
-module(mylist).
-export([mytake/2]).
mytake(_, []) -> [];
mytake(0, _) -> [];
mytake(N, [H | T]) -> [H | mytake(N - 1, T)].
```

Practice: mylist.erl

3) myproduct

Write a function myproduct that takes a list of numbers and returns the product of all the numbers in the list.

mylist:myproduct([1,2,3,1,2,1,3,1]) -> 36

Practice: mylist.erl

3) myproduct

Write a function myproduct that takes a list of numbers and returns the product of all the numbers in the list.

mylist:myproduct([1,2,3,1,2,1,3,1]) -> 36

```
-module(mylist).
-export([myproduct/1]).
myproduct([]) -> 1;
myproduct([H | T]) ->H * myproduct(T).
```

Practice: mylist.erl

4) myappend

Write a function myappend that takes 2 lists and returns the concatenated list.

mylist:myappend([1,2,3], [1,2,1,3,1]) -> [1,2,3,1,2,1,3,1]

Practice: mylist.erl

4) myappend

Write a function myappend that takes 2 lists and returns the concatenated list.

mylist:myappend([1,2,3], [1,2,1,3,1]) -> [1,2,3,1,2,1,3,1]

```
-module(mylist).
-export([myappend/2]).
myappend([], L) -> L;
myappend([H|L1], L2) -> [H|myappend(L1, L2)].
```

Practice: mylist.erl

5) myzip

Write a function, myzip, that takes in two lists and returns a new list of pairs, where each pair contains one element from each list.

mylist:myzip([a,b,c],[1,2,3]) => [{a,1},{b,2},{c,3}]

mylist:myzip([1,2,3], [1,2]) -> [{1,1},{2,2}]

Practice: mylist.erl

5) myzip

Write a function, myzip, that takes in two lists and returns a new list of pairs, where each pair contains one element from each list.
mylist:myzip([a,b,c],[1,2,3]) => [{a,1},{b,2},{c,3}]
mylist:myzip([1,2,3], [1,2]) -> [{1,1},{2,2}]

```
-module(mylist).
-export([myzip/2]).
myzip([], _) -> [];
myzip(_, []) -> [];
myzip([H1 | T1], [H2 | T2]) ->  [{H1, H2} | myzip(T1, T2)].
```