%==================== TOWER ====================

%Transpose, reference https://stackoverflow.com/questions/4280986/how-to-transpose-a-matrix-in-prolog

transpose([], []).

transpose([F|Fs], Ts) :- transpose(F, [F|Fs], Ts).

transpose([], \_, []).

transpose([\_|Rs], Ms, [Ts|Tss]) :-

lists\_firsts\_rests(Ms, Ts, Ms1), transpose(Rs, Ms1, Tss).

lists\_firsts\_rests([], [], []).

lists\_firsts\_rests([[F|Os]|Rest], [F|Fs], [Os|Oss]) :-

lists\_firsts\_rests(Rest, Fs, Oss).

% Reverse a list, reference https://stackoverflow.com/questions/19471778/reversing-a-list-in-prolog

rev([], List, List). %base case

rev([H|T], List, Acc) :- rev(T, List, [H|Acc]).

check\_len(\_, []).

check\_len(N, [H|T]) :- length(H, N), check\_len(N, T).

find\_count([], \_, Curr\_c, C) :- C = Curr\_c.

find\_count([H|T], Curr\_max, Curr\_c, C):-

H #> Curr\_max, New\_c #= Curr\_c + 1, find\_count(T, H, New\_c, C);

H #< Curr\_max, find\_count(T, Curr\_max, Curr\_c, C).

check\_left([], \_).

check\_left([H\_t|T\_t], [H\_c|T\_c]):-

find\_count(H\_t, 0, 0, H\_c),

check\_left(T\_t, T\_c).

check\_right([], \_).

check\_right([H\_t|T\_t], [H\_c|T\_c]) :-

rev(H\_t, Rev\_row, []),

find\_count(Rev\_row, 0, 0, H\_c),

check\_right(T\_t, T\_c).

valid\_counts(Sol, Cnt) :-

Cnt = counts(T, B, L, R),

transpose(Sol, Sol\_t),

check\_left(Sol\_t, T),

check\_right(Sol\_t, B),

check\_left(Sol, L),

check\_right(Sol, R).

valid\_range(N, T) :- fd\_domain(T, 1, N).

tower(N, T, C) :-

% check dimension of rows

length(T, N),

check\_len(N, T), %square matrix

maplist(fd\_all\_different, T), % all elements are unique in each row

maplist(valid\_range(N), T), % all elements are between 1 and N

% check dimension of C

C = counts(Top, Bot, Left, Right),

length(Top, N),

length(Bot, N),

length(Left, N),

length(Right, N),

% check validity of columns

transpose(T, Tp),

length(Tp, N),

maplist(fd\_all\_different, Tp), % all elements are unique in each row

maplist(valid\_range(N), Tp), % all elements are between 1 and N

% verify solution

valid\_counts(T, C),

maplist(fd\_labeling, T).

%==================== PLAIN\_TOWER ====================

% implement fd\_labeling, reference https://stackoverflow.com/questions/23274402/how-to-make-a-list-of-integers-from-1-to-n

n\_ups(N, Xs) :-

length(Xs, N),

numbered\_from(Xs, 1).

numbered\_from([], \_).

numbered\_from([I0|Is], I0) :-

I1 is I0+1,

numbered\_from(Is, I1).

plain\_labeling(N, P) :-

n\_ups(N, L),

permutation(L, P).

valid\_plain(\_, [], \_, \_).

valid\_plain(N, [H\_t | T\_t], [H\_cl | T\_cl], [H\_cr | T\_cr]) :-

length(T, N),

plain\_labeling(N, T), !,

permutation(T, H\_t),

member(H\_cl, T),

find\_count(H\_t, 0, 0, H\_cl),

reverse(H\_t, RevH\_t),

member(H\_cr, T),

find\_count(RevH\_t, 0, 0, H\_cr),

valid\_plain(N, T\_t, T\_cl, T\_cr).

plain\_tower(N, T, C) :-

% check dimension of rows

length(T, N),

check\_len(N, T),

% check dimension of C

C = counts(Top, Bot, Left, Right),

length(Top, N),

length(Bot, N),

length(Left, N),

length(Right, N),

% check validity of columns

valid\_plain(N, T, Left, Right),

transpose(T, Tp),

valid\_plain(N, Tp, Top, Bot).

%==================== AMBIGUOUS ====================

ambiguous(N, C, T1, T2):-

tower(N, T1, C),

tower(N, T2, C),

T1 \= T2.

import asyncio

import aiohttp

import sys

import json

import time

API\_KEY = 'AIzaSyCfxK4xlxq90OXYSJpb2vjvG8NtPiRovMI'

communications = {

'Goloman': ['Hands', 'Holiday', 'Wilkes'],

'Hands': ['Goloman', 'Wilkes'],

'Holiday': ['Goloman', 'Welsh', 'Wilkes'],

'Welsh': ['Holiday'],

'Wilkes': ['Goloman', 'Hands', 'Holiday']

}

port\_dict = {

'Goloman': 12804,

'Hands': 12805,

'Holiday': 12806,

'Welsh': 12807,

'Wilkes': 12808

}

# Should probably be a dict of dicts, like JSON, but it's not

# Dict of client name to client information

clients = {}

def process\_input(input):

"""

Removes leading and trailing white space from the input string, then returns an array

of white-space separated entries in the string

Parameters

--------------

input: String containing message to process

Returns

--------------

msg\_arr: List of white-space separated entries in the string

"""

return input.strip().split()

def valid\_input(msg):

"""

Determines if the input message array is a valid message

IAMAT messages have exactly four entries, formatted as follows:

IAMAT client location time\_sent

The location is checked to see if it is a valid lat/long pair

WHATSAT messages have exactly four entries, formatted as follows:

WHATSAT client radius num\_entries

The radius is checked to see if it is between 1 and 50 (inclusive)

The number of entries ie checked to see if it is between 1 and 20 (inclusive)

CHANGELOC messages have exactly seven entries, formatted as follows:

CHANGELOC client new\_loc time\_sent time\_received server\_received

All CHANGELOC entries of length exactly 6 are assumed to be valid,

as they should only be sent from server to server

All other message are not valid

Assumes all data types are of the correct type, i.e. all castable

Parameters

--------------

msg: List of important entries of the message, with all entries strings

Returns

--------------

msg\_type: Integer describing type of message

1 if message is a valid "IAMAT" message

2 if message is a valid "WHATSAT" message

3 if message is a valid "CHANGELOC" message

-1 if message is not valid

"""

if len(msg) < 1:

return -1

if msg[0] == "IAMAT":

if len(msg) == 4:

if get\_lat\_long(msg[2]) is not None:

time = None

try:

time = float(msg[3])

except:

pass

if time is None:

return -1

return 1

else:

return -1

else:

return -1

elif msg[0] == "WHATSAT":

if len(msg) == 4:

rad = None

try:

rad = float(msg[2])

except:

pass

if rad is None:

return -1

elif rad > 50 or rad <= 0:

return -1

else:

num\_entries = None

try:

num\_entries = int(msg[3])

except:

pass

if num\_entries is None:

return -1

elif num\_entries > 20 or num\_entries <= 0:

return -1

else:

return 2

else:

return -1

elif msg[0] == "CHANGELOC":

if len(msg) == 6:

return 3

else:

return -1

else:

return -1

def get\_lat\_long(input):

"""

Scans the input string for a latitude-longitude pair

A valid latitude-longitude pair has exactly two +/-'s,

serving as delimiters

The first character must be a +/-

The last character may not be a +/-

Parameters

--------------

input: String containing the latitude-longitude pair

Returns

--------------

lat\_long: Tuple with first entry latitude, second entry longitude, or

None if the input string is not a valid pair

"""

# scan the input string for all instances of +/-

instances = []

for i in range(len(input)):

if input[i] == '+' or input[i] == '-':

instances.append(i)

# More than two +/-

if len(instances) != 2:

return None

# First character not a +/-

if instances[0] != 0:

return None

# Last character is a +/-

if instances[1] == len(input) - 1:

return None

lat\_long = None

try:

lat\_long = float(input[:instances[1]]), float(input[instances[1]:])

except:

pass

return lat\_long

def process\_iamat(msg\_arr, time\_received):

"""

Processes the input message array and combines it with time\_received

to create a properly-formatted array for storage

IAMAT messages have the form

IAMAT client location time\_sent

Parameters

--------------

msg\_arr: List containing input IAMAT message

msg\_arr[0] - 'IAMAT'

msg\_arr[1] - client\_name

msg\_arr[2] - location pair

msg\_arr[3] - time client sent

time\_received: Float containing UNIX time of when server received message

Returns

--------------

client\_info: List containing formatted information for storage

client\_info[0] - 'IAMAT'

client\_info[1] - client\_name

client\_info[2] - location pair

client\_info[3] - time client sent

client\_info[4] - time server received

client\_info[5] - server that received the message

"""

if get\_lat\_long(msg\_arr[2]) is None:

return None

return [msg\_arr[0], msg\_arr[1], msg\_arr[2], msg\_arr[3], str(time\_received), sys.argv[1]]

async def flood\_fill(msg, server\_name):

"""

Asynchronously sends CHANGELOC messages to all servers this server communicates with

Parameters

--------------

msg: String, the message to send

server\_name: String, the name of this server

"""

for s in communications[server\_name]:

log\_file.write("Attempting to open connection with server {0} at port {1}...".format(s, port\_dict[s]))

try:

reader, writer = await asyncio.open\_connection('127.0.0.1', port\_dict[s], loop=loop)

log\_file.write("Success\n")

writer.write(msg.encode())

await writer.drain()

writer.close()

except:

# Could not connect

log\_file.write("Fail\n")

pass

# This doesn't technically have to be asynchronous

async def generate\_output(in\_msg, time\_received):

"""

Asynchronously creates output string for the input message, i.e. what to send to the client

Parameters

--------------

in\_msg: String, the raw message the client sent

time\_received: Float, the time that the server received the message

Returns

--------------

out\_msg: String, the raw message to send back to the client

"""

# Get the message in array form

message = in\_msg.strip().split()

out\_msg = ""

error\_msg = "? {0}".format(in\_msg)

message\_type = valid\_input(message)

# IAMAT

if message\_type == 1:

# Update the client dictionary with new location

msg\_info = process\_iamat(message, time\_received)

if msg\_info is not None:

# Store location, reported client time, time received by server

clients[message[1]] = msg\_info

time\_diff = time\_received - float(message[3])

if time\_diff > 0:

time\_diff = "+" + str(time\_diff)

out\_msg = ("AT {0} {1} {2}\n".format(sys.argv[1], time\_diff, ' '.join(message[1:])))

# Send CHANGELOC messages to all connected servers

asyncio.ensure\_future(flood\_fill('CHANGELOC {0}\n'.format(' '.join(msg\_info[1:])), sys.argv[1]))

else:

out\_msg = error\_msg

# WHATSAT

elif message\_type == 2:

if message[1] not in clients:

out\_msg = error\_msg

else:

client = clients[message[1]]

loc = get\_lat\_long(client[2])

loc = str(loc[0]) + "," + str(loc[1])

rad = float(message[2]) \* 1000

url = 'https://maps.googleapis.com/maps/api/place/nearbysearch/json?key={0}&location={1}&radius={2}'.format(API\_KEY, loc, rad)

time\_diff = float(client[4]) - float(client[3])

if time\_diff > 0:

time\_diff = "+" + str(time\_diff)

out\_msg = "AT {0} {1} {2} {3} {4}\n".format(client[5], time\_diff, client[1], client[2], client[3])

async with aiohttp.ClientSession() as session:

async with session.get(url) as resp:

response = await resp.json()

response['results'] = response['results'][:int(message[3])]

out\_msg += json.dumps(response, indent=3)

out\_msg += "\n\n"

else:

out\_msg = error\_msg

return out\_msg

async def handle\_input(reader, writer):

"""

Asynchronous method that is called in a loop until a KeyboardInterrupt

Reads the input from the reader, writes corresponding output to the writer

Adapted from asyncio.readthedocs.io/en/latest/tcp\_echo.html

Parameters

--------------

reader: StreamReader instance

writer: StreamWriter instance

"""

# Read the buffer data

data = await reader.readline()

time\_received = time.time()

in\_msg = data.decode()

# The data already ends in a newline

log\_file.write("RECEIVED: " + in\_msg)

# Format the message to be in list form

message = in\_msg.strip().split()

# The CHANGELOC case is handled independently since there is no output to the client/other servers when received

if message[0] == "CHANGELOC" and valid\_input(message):

# Check if the time in the message was sent later than the time currently in the dict

if message[1] not in clients:

clients[message[1]] = message

asyncio.ensure\_future(flood\_fill('CHANGELOC {0}\n'.format(' '.join(message[1:])), sys.argv[1]))

else:

# This location came after the one currently stored in the dictionary

# Don't floodfill if this is the second time you've received the message

if message[3] > clients[message[1]][3]:

clients[message[1]] = message

asyncio.ensure\_future(flood\_fill('CHANGELOC {0}\n'.format(' '.join(message[1:])), sys.argv[1]))

else:

out\_msg = await generate\_output(in\_msg, time\_received)

log\_file.write("SENDING: " + out\_msg)

writer.write(out\_msg.encode())

await writer.drain()

def main():

"""

Main function

Parses command-line arguments to make sure they are valid

Opens a log file for this specific server

Runs the handle\_input function asynchronously in a loop

Adapted from asyncio.readthedocs.io/en/latest/tcp\_echo.html

"""

if len(sys.argv) != 2:

print("Bad args")

sys.exit(1)

if sys.argv[1] not in port\_dict:

print("Bad server name")

sys.exit(1)

global log\_file

log\_file = open(sys.argv[1] + "\_log.txt", "w+")

global loop

loop = asyncio.get\_event\_loop()

coro = asyncio.start\_server(handle\_input, '127.0.0.1', port\_dict[sys.argv[1]], loop=loop)

server = loop.run\_until\_complete(coro)

# print("Initializing server {0} at port {1}".format(sys.argv[1], port\_dict[sys.argv[1]]))

try:

loop.run\_forever()

except KeyboardInterrupt:

pass

server.close()

loop.run\_until\_complete(server.wait\_closed())

loop.close()

# The log won't update until the server gets ^C'ed, which is probably bad for a server log

log\_file.close()

if \_\_name\_\_ == '\_\_main\_\_':

main()

import java.util.concurrent.atomic.AtomicIntegerArray;

class GetNSetState implements State {

private AtomicIntegerArray value;

private byte maxval;

GetNSetState(byte[] v) {

value = new AtomicIntegerArray(v.length);

for (int i = 0; i < v.length; i++) {

value.set(i, (int) v[i]);

}

maxval = 127;

}

GetNSetState(byte[] v, byte m) {

value = new AtomicIntegerArray(v.length);

for (int i = 0; i < v.length; i++) {

value.set(i, (int) v[i]);

}

maxval = m;

}

public int size() { return value.length(); }

public byte[] current() {

byte[] curr = new byte[value.length()];

for (int i = 0; i < value.length(); i++) {

curr[i] = (byte) value.get(i);

}

return curr;

}

public boolean swap(int i, int j) {

int val\_i = value.get(i);

int val\_j = value.get(j);

if (val\_i <= 0 || val\_j >= maxval) {

return false;

}

value.set(i, val\_i-1);

value.set(j, val\_j+1);

return true;

}

}

import java.util.concurrent.locks.ReentrantLock;

class BetterSafeState implements State {

private byte[] value;

private byte maxval;

private ReentrantLock l;

BetterSafeState(byte[] v) {

value = v; maxval = 127; l = new ReentrantLock();

}

BetterSafeState(byte[] v, byte m) {

value = v; maxval = m; l = new ReentrantLock();

}

public int size() { return value.length; }

public byte[] current() { return value; }

public boolean swap(int i, int j) {

l.lock();

if (value[i] <= 0 || value[j] >= maxval) {

l.unlock();

return false;

}

value[i]--;

value[j]++;

l.unlock();

return true;

}

}

type ('nonterminal, 'terminal) symbol =

| N of 'nonterminal

| T of 'terminal;;

type ('nonterminal, 'terminal) parse\_tree =

| Node of 'nonterminal \* ('nonterminal, 'terminal) parse\_tree list

| Leaf of 'terminal;;

(\* convert grammar \*)

let rec prod\_func rules non\_term = match rules with

| [] -> []

(\* if lhs matches the argument,

\* add to the alternative list and restart\_symbolsively go on to the rest of rules

\* else restart\_symbolsively go on to the rest of rules \*)

| h::t -> if (fst h) = non\_term then (snd h)::(prod\_func t non\_term)

else (prod\_func t non\_term);;

(\* grammar: (start symbol, construct a production function for the rhs rules) \*)

let convert\_grammar g = (fst g), (prod\_func (snd g));;

(\* parse tree leaves \*)

let rec parse\_tree\_helper = function

| [] -> []

| h::t -> match h with

| Leaf lf -> lf::(parse\_tree\_helper t)

| Node (nonterminal, terminal) -> (parse\_tree\_helper terminal)@(parse\_tree\_helper t);;

let parse\_tree\_leaves tree = parse\_tree\_helper [tree];;

(\* matcher \*)

let rec matcher start\_symbol prod\_func rules accept frag = match rules with

| [] -> None (\* no rules are left to check \*)

| h\_rules::t\_rules ->

let head\_matcher = matcher\_helper h\_rules prod\_func accept frag in (\* \*)

match head\_matcher with

| None -> matcher start\_symbol prod\_func t\_rules accept frag (\* can't match the start\_symbolrent rule \*)

| \_ -> head\_matcher (\* otherwise, match the header \*)

and matcher\_helper start\_rule prod\_func accept frag = match start\_rule with

| [] -> accept frag

| \_ -> match frag with

| [] -> None

| h\_frag::t\_frag -> match start\_rule with

| [] -> None

| (N nonterminal)::t1 ->

let other\_rules = (prod\_func nonterminal)

and accept2 = (matcher\_helper t1 prod\_func accept)

in matcher nonterminal prod\_func other\_rules accept2 frag

| (T terminal)::t2 -> if terminal = h\_frag then matcher\_helper t2 prod\_func accept t\_frag

else None

let make\_matcher gram = fun accept frag -> matcher (fst gram) (snd gram) ((snd gram) (fst gram)) accept frag;;

(\* make parser \*)

let rec parser\_matcher start prod\_func rules accept frag = match rules with

| [] -> None

| h\_rules::t\_rules ->

let head\_matcher = parser\_matcher\_helper start prod\_func h\_rules accept frag in (\* \*)

match head\_matcher with

| None -> parser\_matcher start prod\_func t\_rules accept frag (\* can't match the start\_symbolrent rule \*)

| Some t -> Some(h\_rules::t)

and parser\_matcher\_helper start prod\_func rules accept frag = match rules with

| [] -> accept frag

| \_ -> match frag with (\* cmp is the list of strings so hd is start\_symbolrent fragment item in the list of string \*)

| [] -> None

| h\_frag::t\_frag -> match rules with

| [] -> None

| (N nonterminal)::t1 ->

let other\_rules = (prod\_func nonterminal)

and accept2 = (parser\_matcher\_helper start prod\_func t1 accept)

in parser\_matcher start prod\_func other\_rules accept2 frag

| (T terminal)::t2 ->

if terminal = h\_frag then parser\_matcher\_helper start prod\_func t2 accept t\_frag

else None

;;

let make\_parser\_matcher gram accept frag =

parser\_matcher (fst gram) (snd gram) ((snd gram) (fst gram)) accept frag

;;

let rec parser start\_symbol pmatcher = match start\_symbol with

| [] -> (pmatcher, [])

| h::t ->

((fst (parser t (fst (parser\_helper h pmatcher)))),

(snd (parser\_helper h pmatcher))::(snd (parser t (fst (parser\_helper h pmatcher)))))

and parser\_helper start\_symbol pmatcher = match start\_symbol with

| (T terminal) -> (match pmatcher with

| [] -> ([], Leaf terminal)

| h::t -> (h::t, Leaf terminal))

| (N nonterminal) -> match pmatcher with (\* Non terminal here\*)

| [] -> ([], Node (nonterminal, []))

| h::t -> ((fst (parser h t)), Node (nonterminal, (snd (parser h t))))

;;

let make\_parser\_helper gram pmatcher = parser [N (fst gram)] pmatcher;;

let wrapper frag = match frag with

| [] -> Some []

| \_ -> None

;;

let make\_parser gram = fun frag -> match (make\_parser\_matcher gram wrapper frag) with

| None -> None

| Some [] -> None

| Some t -> Some (List.hd (snd (make\_parser\_helper gram t)));;

fun<T> everyNth(lst: List<T>, N: Int) : List<T>{

if (N <= 0 || N > lst.size) {

println("Error: please provide a valid N")

return emptyList()

}

var ret = listOf<T>()

for (i in 0..(lst.size-1)) {

if ((i+1) % N == 0) { // every Nth element

ret += lst[i]

}

}

return ret

}

#lang racket

;check indices

(define (helper-Ind arr sing ind)

(cond

[(equal? (car arr) sing) ind]

[else (helper-Ind (cdr arr) sing (add1 ind))]

)

)

(define (var-fixate a b)

(string->symbol (string-append (symbol->string a) "!" (symbol->string b)))

)

(define (rec-lambda a b c d e)

(if (and (equal? a b) '())

;both are empty

(list c d e)

(if (not (equal? (car a) (car b))) (rec-lambda (cdr a) (cdr b) (cons (var-fixate (car a) (car b)) c) (cons (car a) d) (cons (car b) e)) (rec-lambda (cdr a) (cdr b) c d e)

)

)

)

; helper function for lambd

;fE is first expression and sE is second expression

(define (substitute fE sE singl dual)

(if (equal? '() fE) '()

(if (list? (car fE)) (cons (substitute (car fE) sE singl dual) (substitute (cdr fE) sE singl dual))

(if (member (car fE) singl)

(cons (let ((ct (helper-Ind singl (car fE) 0))) (list-ref dual ct)) (substitute (cdr fE) sE singl dual))

(cons (car fE) (substitute (cdr fE) sE singl dual))

)

)

)

)

(define (dbl term)

(car (car term))

)

(define (fix-FHelper a b c d)

(cons (expr-compare (car a) (car b)) (fix-Fst (cdr a) (cdr b) c d))

)

(define (fix-Flist a b c d)

(cons (fix-Fst (car a) (car b) c d) (fix-Fst (cdr a) (cdr b) c d))

)

(define (fix-SHelper a b c d)

(cons (expr-compare (car a) (car b)) (fix-Snd (cdr b) (cdr a) c d))

)

(define (fix-Slist a b c d)

(cons (fix-Snd (car b) (car a) c d) (fix-Snd (cdr b) (cdr a) c d))

)

(define (fix-Fst a b c d)

(cond

[(equal? '() a) '()]

[(and (equal? #t (list? (car a))) (equal? #f (list? (car b)))) (expr-compare a b)]

[(and (equal? #f (list? (car a))) (equal? #t (list? (car b)))) (expr-compare a b)]

[(and (list? (car a)) (equal? (dbl a) 'lambda))(fix-FHelper a b c d)]

[(and (list? (car a)) (equal? (dbl b) 'lambda))(fix-FHelper a b c d)]

[(and (list? (car a)) (equal? (dbl a) 'λ))(fix-FHelper a b c d)]

[(and (list? (car a)) (equal? (dbl b) 'λ))(fix-FHelper a b c d)]

[(list? (car a)) (fix-Flist a b c d)]

[(member (car a) d) (cons (let ((ct (helper-Ind d (car a) 0))) (list-ref c ct)) (fix-Fst (cdr a) (cdr b) c d))]

[else (cons (car a) (fix-Fst (cdr a) (cdr b) c d))]))

(define (fix-Snd b a c d)

(cond

[(equal? '() b) '()]

[(and (equal? #t (list? (car a))) (equal? #f (list? (car b)))) (expr-compare a b)]

[(and (equal? #f (list? (car a))) (equal? #t (list? (car b)))) (expr-compare a b)]

[(and (list? (car b)) (equal? (dbl a) 'lambda))(fix-SHelper a b c d)]

[(and (list? (car b)) (equal? (dbl b) 'lambda))(fix-SHelper a b c d)]

[(and (list? (car b)) (equal? (dbl a) 'λ))(fix-SHelper a b c d)]

[(and (list? (car b)) (equal? (dbl b) 'λ))(fix-SHelper a b c d)]

[(list? (car b)) (fix-Slist a b c d)]

[(member (car b) d) (cons (let ((ct (helper-Ind d (car b) 0))) (list-ref c ct)) (fix-Snd (cdr b) (cdr a) c d))]

[else (cons (car b) (fix-Snd (cdr b) (cdr a) c d))]))

(define (sItem? a)

(if (list? (cadr a))

(not (pair? (cadr a)))

#t

)

)

;first checks if list

; returns true if its not a pair or list

(define (lambda-compare a b)

(cond

[(and (equal? #f (sItem? a))(equal? #f (sItem? b)))

(if (equal? (length (cadr a)) (length (cadr b)))

(let ((fixate (rec-lambda (cadr a) (cadr b) '() '() '())))

(cond

[(or (equal? (car a) 'λ) (equal? (car b) 'λ))

(cons 'λ

(expr-compare

(fix-Fst (cdr a) (cdr b) (car fixate) (cadr fixate))

(fix-Snd (cdr b) (cdr a) (car fixate) (car (cddr fixate))))

)]

[else

(cons 'lambda

(expr-compare

(fix-Fst (cdr a) (cdr b) (car fixate) (cadr fixate))

(fix-Snd (cdr b) (cdr a) (car fixate) (car (cddr fixate))))

)

]

)

)

(list-compare a b)

)]

[(and (equal? #t (sItem? a))(equal? #f (sItem? b))) (list-compare a b)]

[(and (equal? #f (sItem? a))(equal? #t (sItem? b))) (list-compare a b)]

[else

(let ((fixate (rec-lambda (list (cadr a)) (list (cadr b)) '() '() '())))

(cond

[(or (equal? (car a) 'λ) (equal? (car b) 'λ))

(cons 'λ

(expr-compare

(fix-Fst (cdr a) (cdr b) (car fixate) (cadr fixate))

(fix-Snd (cdr b) (cdr a) (car fixate) (car (cddr fixate))))

)]

[else

(cons 'lambda

(expr-compare

(fix-Fst (cdr a) (cdr b) (car fixate) (cadr fixate))

(fix-Snd (cdr b) (cdr a) (car fixate) (car (cddr fixate))))

)

]

)

)

]

)

)

;;;;

(define (loop-compare a b)

(cond

[ (and (not(equal? '() a)) (not (equal? '() b)))

(cons (expr-compare (car a) (car b)) (loop-compare (cdr a) (cdr b)))]

[ (and (equal? '() a) (not (equal? '() b))) '()]

[ (and (not(equal? '() a)) (equal? '() b)) '()]

[else '()]

)

)

;recursively loop

(define (list-compare a b)

(cond

[(equal? a b) a] ;checks if they are equal terms

[(and (boolean? a)(boolean? b))

(if a (if b #t '%) (if b '(not %) #f))]

[else (list 'if '% a b)]))

;compare base case

(define (checker a b)

(cond

[(equal? (car a) 'lambda) (lambda-compare a b)]

[(equal? (car a) 'quote) (list-compare a b)]

[(equal? (car a) 'λ) (lambda-compare a b)]

[else (loop-compare a b)]

)

)

(define (checkFirst a b)

(cond

[(equal? 'lambda (car a)) (lambda-compare a b)]

[(equal? 'lambda (car b)) (lambda-compare a b)]

[(equal? 'λ (car a)) (lambda-compare a b)]

[(equal? 'λ (car b)) (lambda-compare a b)]

[(equal? 'quote (car a)) (list-compare a b)]

[(equal? 'quote (car b)) (list-compare a b)]

[(equal? 'if (car a)) (list-compare a b)]

[(equal? 'if (car b)) (list-compare a b)]

[else (loop-compare a b)]

)

)

;if statement checker

(define (expr-compare a b)

(if (and (list? a) (list? b))

(cond

[(equal? (length a) (length b))

(if (equal? (car a) (car b))

(checker a b)

(checkFirst a b)

)]

[else (list-compare a b)]

)

(list-compare a b)

)

)

(define (test-expr-compare x y)

(cond

[(and (equal? (eval (list 'let '((% #t)) (expr-compare x y) )) (eval x))

(equal? (eval (list 'let '((% #f)) (expr-compare x y) )) (eval y))

) #t]

[else #f]

)

)

#|

(test-expr-compare '(let ((k 3)) k) '(let ((k 5)) 5))

|#

;TEST EXPRESSION X AND Y

(define test-expr-x

(list

#t

#f

#t

#f

10

10

;bools and nums

'(1 5 10)

'(cons a b)

'(cons a b)

'(cons a b)

'(list)

'(if a b c)

;lists

'(+ 12 (- 20 12))

'(quote (3 4))

; lambda special for

'((lambda (a) (f a)) 1)

'((lambda (a) (f a)) 1)

'(λ (x y) (+ x y))

'(let ((a b)) a)

'(if a b c)

''((λ (a) a) c)

'(+ #f ((λ (a b) (f a b)) 1 2))

'(quoth (a b))

'((lambda (a) (eq? a ((λ (a b) ((λ (a b) (a b)) b a))

a (lambda (a) a))))

(lambda (b a) (b a)))

;pulled from test cases

)

)

(define test-expr-y

(list

#t

#f

#f

#t

10

20

;bools and nums

'(1 5 10)

'(cons a b)

'(cons a c)

'(list a b)

'(list d)

'(x a b c)

;lists

'(+ 12 (- 20 12))

'(quote (3 4))

; lambda special form

'((lambda (a) (f a)) 1)

'((λ (a) (g a)) 2)

'(λ (x y) (+ x y))

'(let ((a b)) a)

'(if a b c)

''((lambda (b) b) d)

'(+ #t ((lambda (a c) (f a c)) 1 2))

'(quoth (a c))

'((λ (a) (eqv? a ((lambda (b a) ((lambda (a b) (a b)) b a))

a (λ (b) a))))

(lambda (a b) (a b)))

)

)