Week 10 Discussion

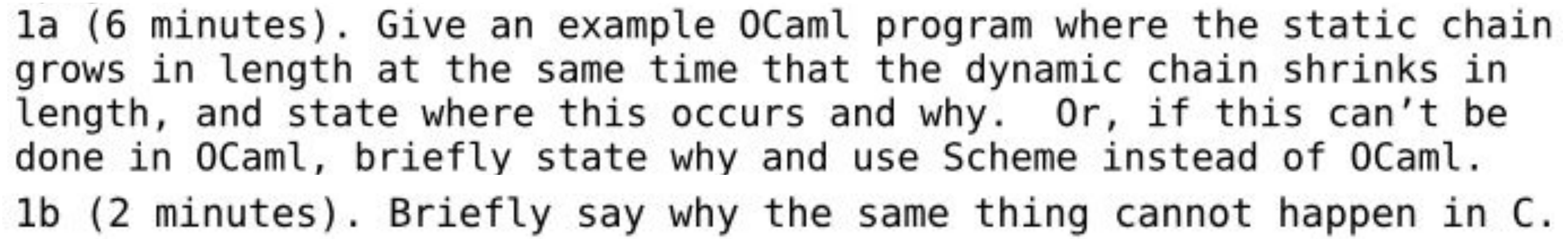
CS 131 Section 1B
3 June 2022
Danning Yu

Announcements

- HW6 released, due 6/3; no late submissions accepted
- Course evaluations available, submit by 6/4 8am
- Final on Tuesday, 6/7 8am-11am
- Homeworks should be submitted on BruinLearn, under Assignments
- Before submitting
  - Make sure your code compiles on SEASnet server
  - Make sure your function signatures are correct
  - Follow all instructions and specifications
  - Do not submit files in a .zip unless told to do so
- Help and starter code from past TAs
  - https://github.com/CS131-TA-team

Spring 2020 Final

Spring 2020 Question 1

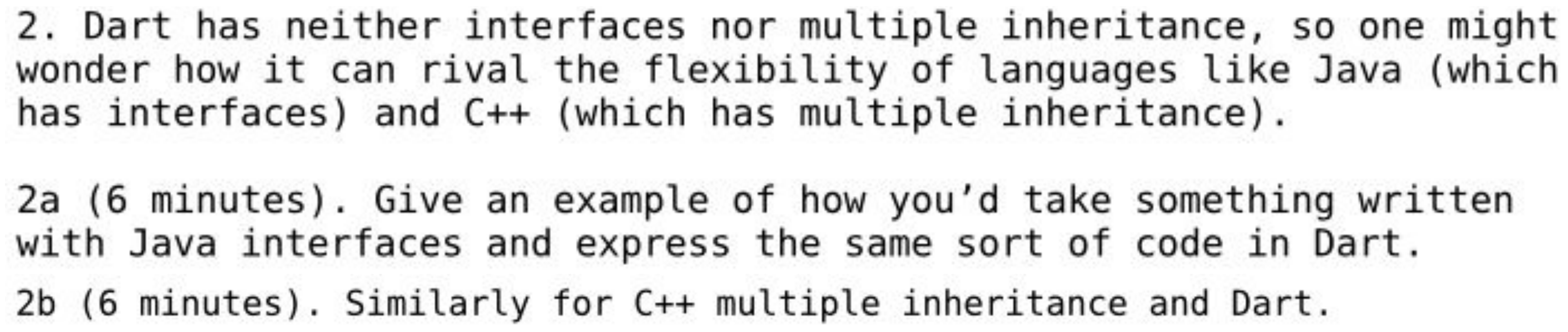


1a (6 minutes). Give an example OCaml program where the static chain grows in length at the same time that the dynamic chain shrinks in length, and state where this occurs and why. Or, if this can't be done in OCaml, briefly state why and use Scheme instead of OCaml.

1b (2 minutes). Briefly say why the same thing cannot happen in C.

- Static chain: formed from nested function calls
- Dynamic chain: formed by functions nested inside other functions
- `func_a` nested 5 levels deep with dynamic chain of length 5
- `func_b` nested 4 levels deep with dynamic chain of length 4
- `func_a` calls `func_b`, causes static chain to increase by 1, dynamic chain to decrease by 1
- Not possible in C because nested function definitions are not allowed

Spring 2020 Question 2



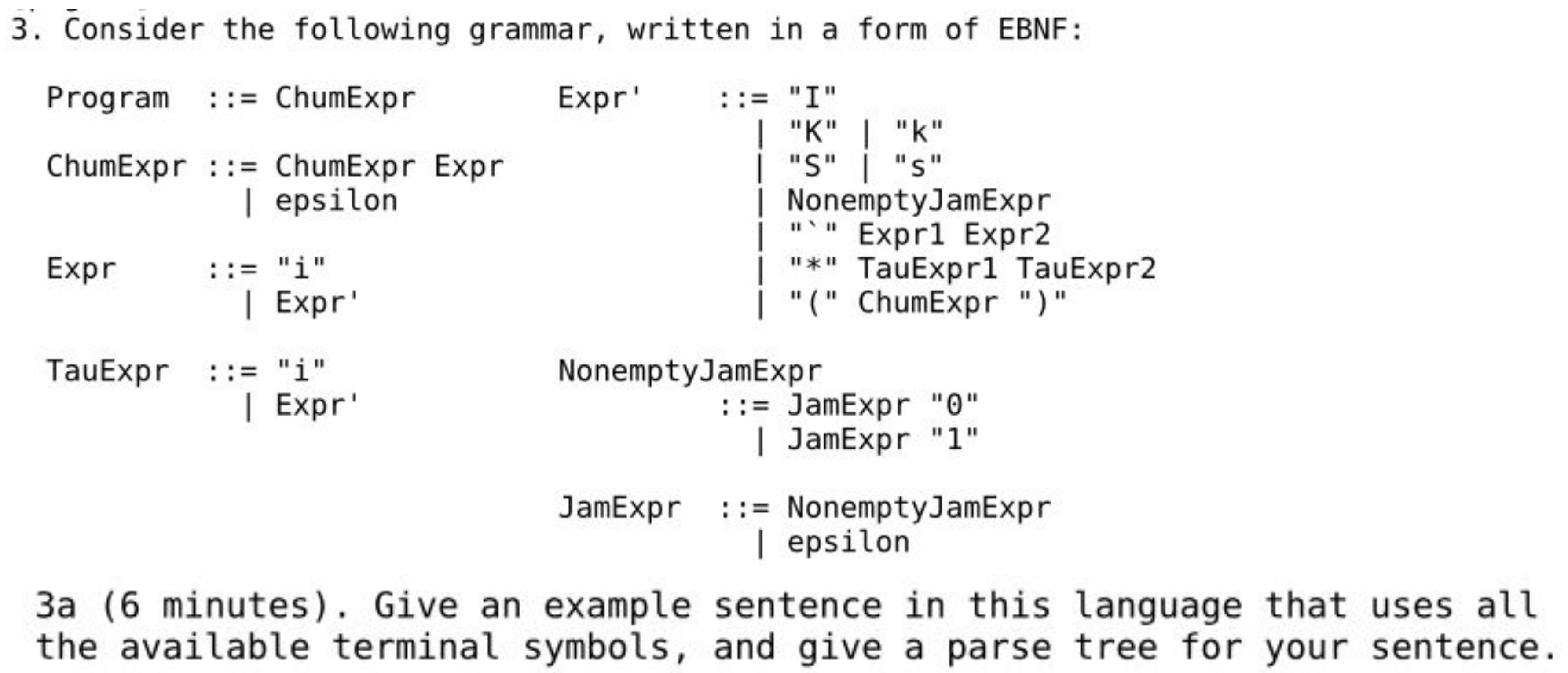
2. Dart has neither interfaces nor multiple inheritance, so one might wonder how it can rival the flexibility of languages like Java (which has interfaces) and C++ (which has multiple inheritance).

2a (6 minutes). Give an example of how you'd take something written with Java interfaces and express the same sort of code in Dart.

2b (6 minutes). Similarly for C++ multiple inheritance and Dart.

- We didn't cover Dart this quarter, so not relevant
- There'll probably be a question about Go, so make sure you're somewhat familiar with the language
  - Especially Go as it relates to templates, since that's what the homework covered

Spring 2020 Question 3a

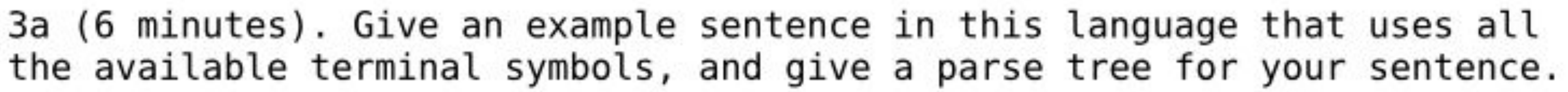


3. Consider the following grammar, written in a form of EBNF:

```
Program  ::= ChumExpr          Expr'    ::= "I"
                                          | "K" | "k"
ChumExpr ::= ChumExpr Expr                | "S" | "s"
           | epsilon                      | NonemptyJamExpr
                                          | "`" Expr1 Expr2
Expr     ::= "i"                          | "*" TauExpr1 TauExpr2
           | Expr'                        | "(" ChumExpr ")"

TauExpr  ::= "i"               NonemptyJamExpr
           | Expr'                      ::= JamExpr "0"
                                          | JamExpr "1"

                               JamExpr  ::= NonemptyJamExpr
                                          | epsilon
```

3a (6 minutes). Give an example sentence in this language that uses all the available terminal symbols, and give a parse tree for your sentence.

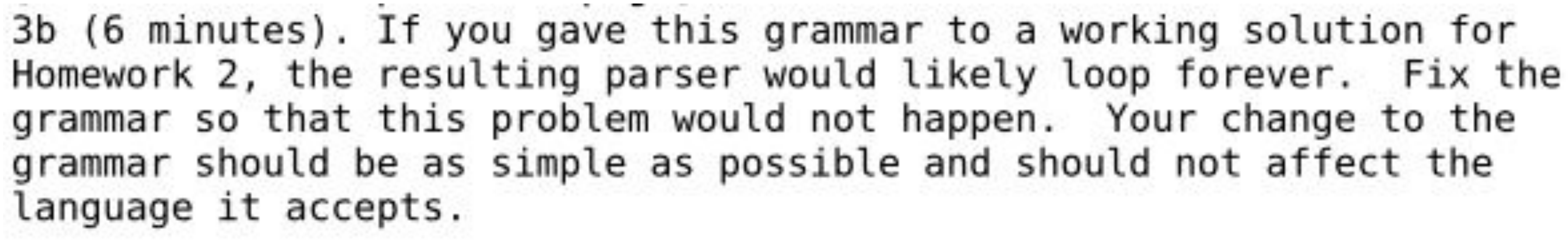
Spring 2020 Question 3a


3a (6 minutes). Give an example sentence in this language that uses all the available terminal symbols, and give a parse tree for your sentence.

```
Program
ChumExpr
ChumExpr Expr
ChumExpr Expr Expr Expr Expr
epsilon Expr Expr Expr Expr Expr Expr Expr Expr
epsilon "i"  Expr' Expr' Expr' Expr' Expr' Expr' Expr' Expr'
epsilon "i"  "I"   "K"     "k" "S"     "s" Expr' Expr' Expr'
epsilon "i"  "I"   "K"     "k" "S"     "s" NonemptyJamExpr Expr'
Expr'
epsilon "i"  "I"  "K"  "k"  "S"   "s"  "0" "1" "`" "i" "i" "*" "i"
    "i" "(" ")"
```

Without quotes: `iIKkSs01`ii*ii()`

Spring 2020 Question 3b



3b (6 minutes). If you gave this grammar to a working solution for Homework 2, the resulting parser would likely loop forever. Fix the grammar so that this problem would not happen. Your change to the grammar should be as simple as possible and should not affect the language it accepts.

- The issue here is that the grammar is left recursive
- Fix `ChumExpr` and `NonemptyJamExpr`

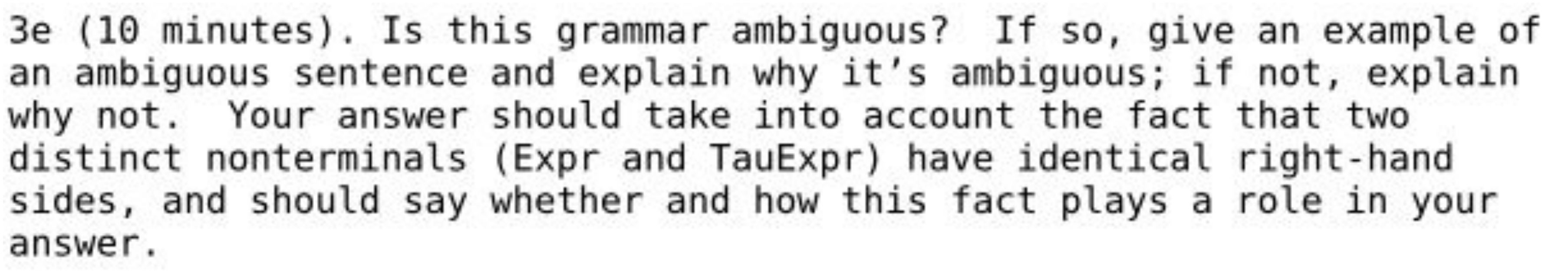
```
ChumExpr ::= Expr ChumExpr | epsilon
NonemptyJamExpr ::= "0" JamExpr | "1" JamExpr
```

Spring 2020 Question 3c & 3d

3c (6 minutes). Use EBNF to make the grammar simpler and easier to understand.
3d (6 minutes). Draw a syntax diagram for the grammar, in the style of Webber. Keep the diagram as simple and uncluttered as possible.

```
Program ::= {Expr}
Expr ::= "i" | Expr'
Expr' ::= "I" | "K" | "k" | "S" | "s" | NonemptyJamExpr
        | {"`"} "i" {"`"} "i" | {"`"} "i" {"`"} "i"
        | "(" [Expr] ")"
      (* note: Expr' has errors in it *)
NonemptyJamExpr ::= (0|1), {(0|1)} // or (0|1)+
JamExpr ::= [NonemptyJamExpr] // or NonEmptyJamExpr?
```

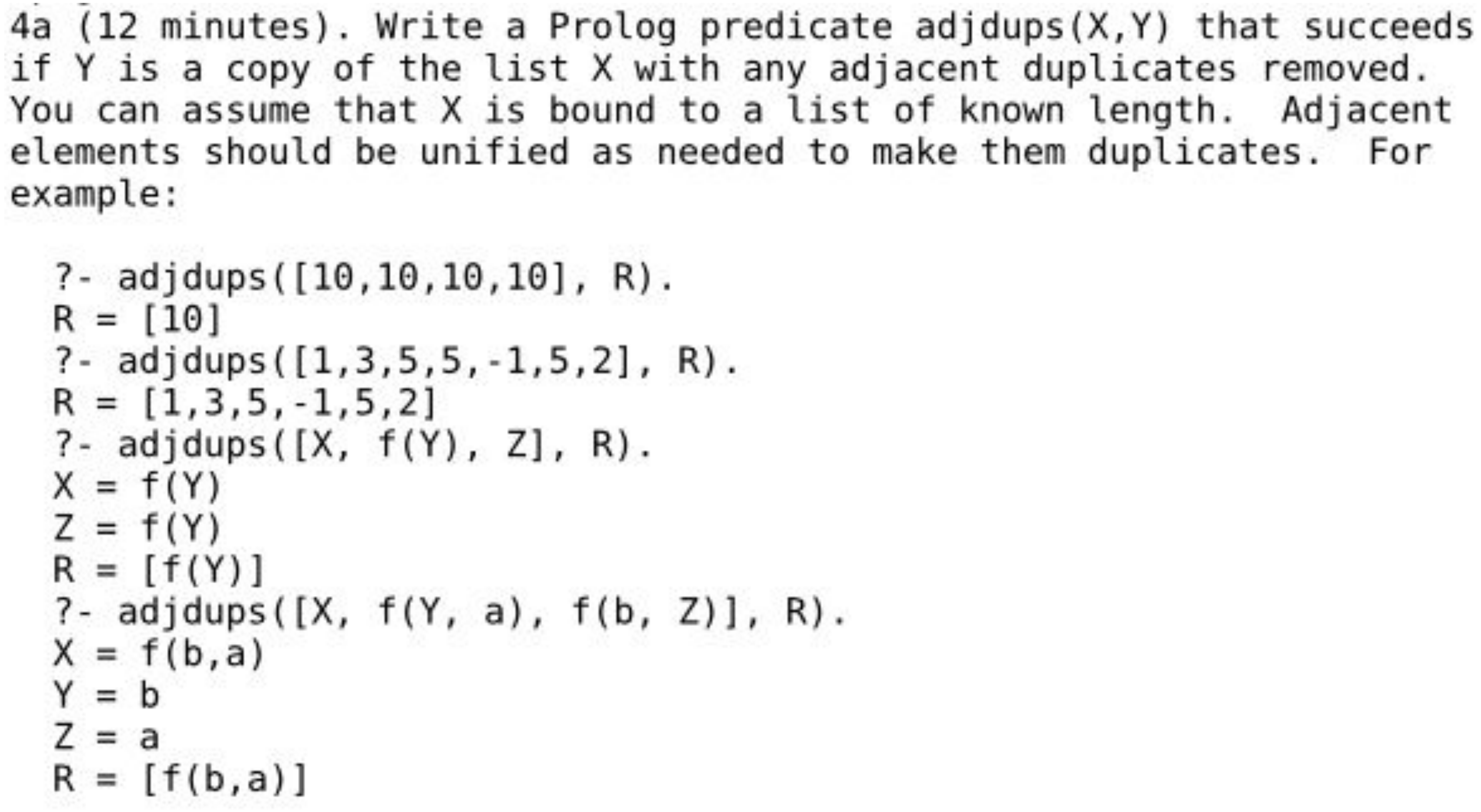
Spring 2020 Question 3e



3e (10 minutes). Is this grammar ambiguous? If so, give an example of an ambiguous sentence and explain why it's ambiguous; if not, explain why not. Your answer should take into account the fact that two distinct nonterminals (Expr and TauExpr) have identical right-hand sides, and should say whether and how this fact plays a role in your answer.

- This grammar is not ambiguous because for every expression, there is only a single parse tree possible for it
- `Expr` and `TauExpr` have the same RHS, `TauExpr` can only be reached if the preceding token is a *, and so there is no chance of confusing it with `Expr`, which requires a ` to precede it
    - In a sense, ` and * are two different operators, such as + and *, and since both operands need to use the same rule, no ambiguity is possible

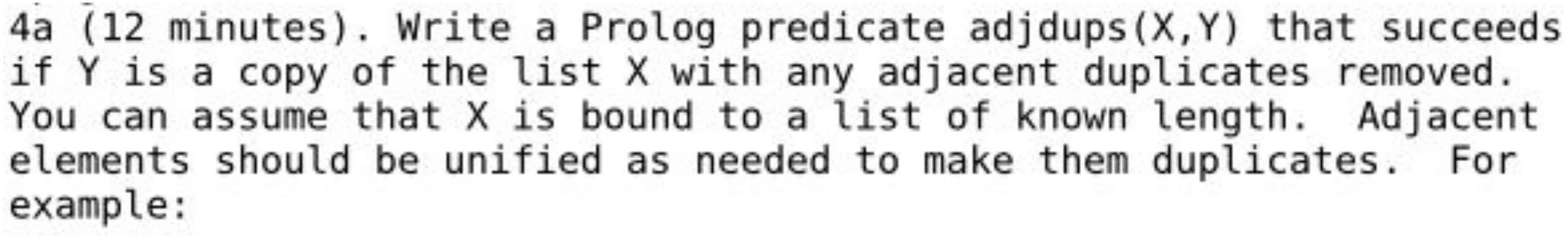
Spring 2020 Question 4a



4a (12 minutes). Write a Prolog predicate adjdups(X,Y) that succeeds if Y is a copy of the list X with any adjacent duplicates removed. You can assume that X is bound to a list of known length. Adjacent elements should be unified as needed to make them duplicates. For example:

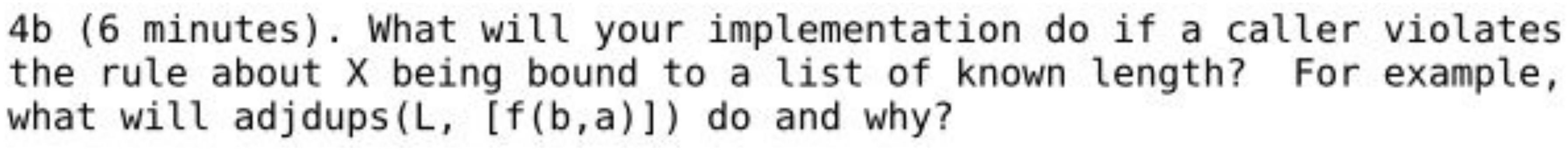
```
?- adjdups([10,10,10,10], R).
R = [10]
?- adjdups([1,3,5,5,-1,5,2], R).
R = [1,3,5,-1,5,2]
?- adjdups([X, f(Y), Z], R).
X = f(Y)
Z = f(Y)
R = [f(Y)]
?- adjdups([X, f(Y, a), f(b, Z)], R).
X = f(b,a)
Y = b
Z = a
R = [f(b,a)]
```

Spring 2020 Question 4a


4a (12 minutes). Write a Prolog predicate adjdups(X,Y) that succeeds if Y is a copy of the list X with any adjacent duplicates removed. You can assume that X is bound to a list of known length. Adjacent elements should be unified as needed to make them duplicates. For example:

```
adjdups([], []).   % base case: nothing = no dups
adjdups([X], [X]). % one element = no dups

% two adjacent duplicates -> skip the first one
adjdups([X,X|Rest], R) :- adjdups([X|Rest], R).

% X is diff, so we return it for sure
adjdups([X,Y|Rest], [X|R]) :- X \= Y, adjdups([Y|Rest], R).
```

Spring 2020 Question 4b



4b (6 minutes). What will your implementation do if a caller violates the rule about X being bound to a list of known length? For example, what will adjdups(L, [f(b,a)]) do and why?

- The given implementation attempts to check the first 2 elements of any list with 2 or more elements, and if they are the same, it skips the first element
- If they are not the same, then it saves the first element in the result
- As a result, if X is bound to a list of unknown length, then Prolog will return infinite solutions, since it can simply keep replicating the first element to create a list with more duplicates
- For example, for the given example, it returns the following answers:

```
L = [f(b,a)] ? ;
L = [f(b,a),f(b,a)] ? ;
L = [f(b,a),f(b,a),f(b,a)] ? ;
```

...and so on

Spring 2020 Question 5

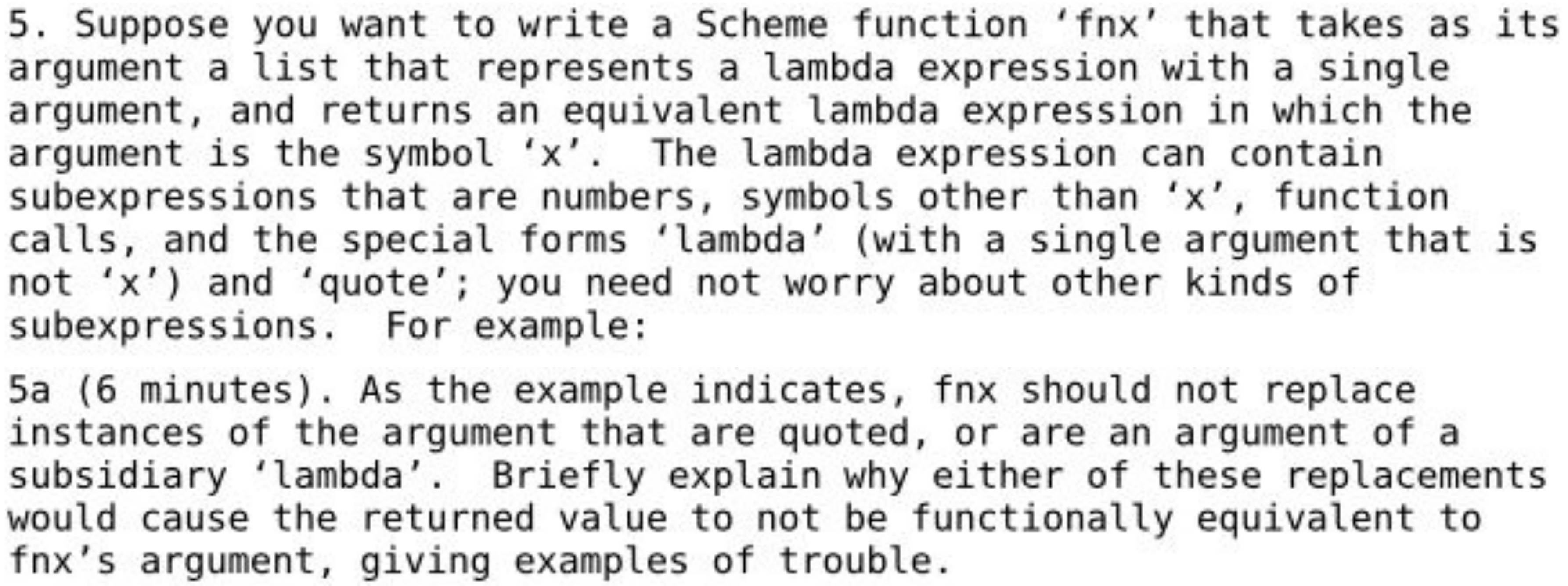
5. Suppose you want to write a Scheme function ‘fnx’ that takes as its argument a list that represents a lambda expression with a single argument, and returns an equivalent lambda expression in which the argument is the symbol ‘x’. The lambda expression can contain subexpressions that are numbers, symbols other than ‘x’, function calls, and the special forms ‘lambda’ (with a single argument that is not ‘x’) and ‘quote’; you need not worry about other kinds of subexpressions. For example:

```
(fnx
 '(lambda (a) (list a '(a 27) ((lambda (a) (f a b -1)) (a a)))))
```

should yield the following list:

```
(lambda (x) (list x '(a 27) ((lambda (a) (f a b -1)) (x x))))
```

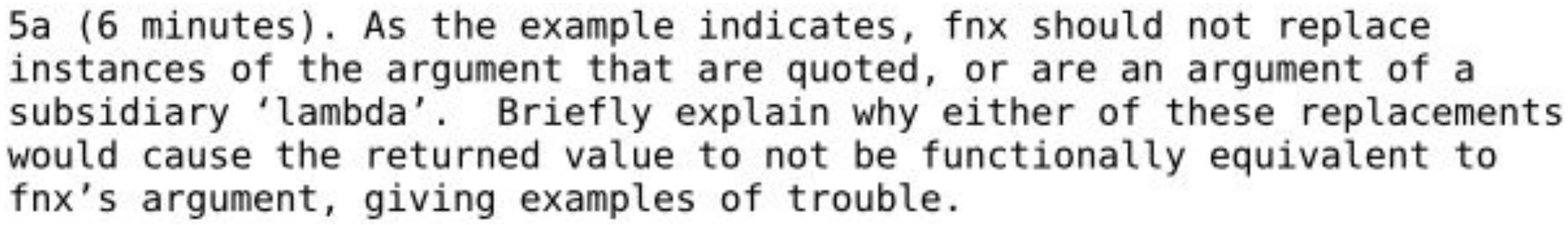
Spring 2020 Question 5a



5. Suppose you want to write a Scheme function ‘fnx’ that takes as its argument a list that represents a lambda expression with a single argument, and returns an equivalent lambda expression in which the argument is the symbol ‘x’. The lambda expression can contain subexpressions that are numbers, symbols other than ‘x’, function calls, and the special forms ‘lambda’ (with a single argument that is not ‘x’) and ‘quote’; you need not worry about other kinds of subexpressions. For example:

5a (6 minutes). As the example indicates, fnx should not replace instances of the argument that are quoted, or are an argument of a subsidiary ‘lambda’. Briefly explain why either of these replacements would cause the returned value to not be functionally equivalent to fnx’s argument, giving examples of trouble.

Spring 2020 Question 5a



5a (6 minutes). As the example indicates, fnx should not replace instances of the argument that are quoted, or are an argument of a subsidiary ‘lambda’. Briefly explain why either of these replacements would cause the returned value to not be functionally equivalent to fnx’s argument, giving examples of trouble.

- Quoted expressions are meant to be interpreted literally
  - A variable in a quoted expression is not meant to be expanded or substituted, so it would be wrong to make the change
  - Example: `'(a b)` is intended to mean the list with elements a and b. Changing this to `'(x b)` would change it to the list with elements x and b, which is not the same
- In lambda expressions, variables inside inner scopes take precedence so that the scope stays static
  - If an argument is redefined in a nested lambda, then this is a completely new binding that is different from the initial variable binding
  - This means that it would be incorrect, although harmless (?), to substitute the variable for `x`
  - Makes more sense to leave the inner lambda binding as is

Spring 2020 Question 5b

5b (24 minutes). Implement ‘fnx’.

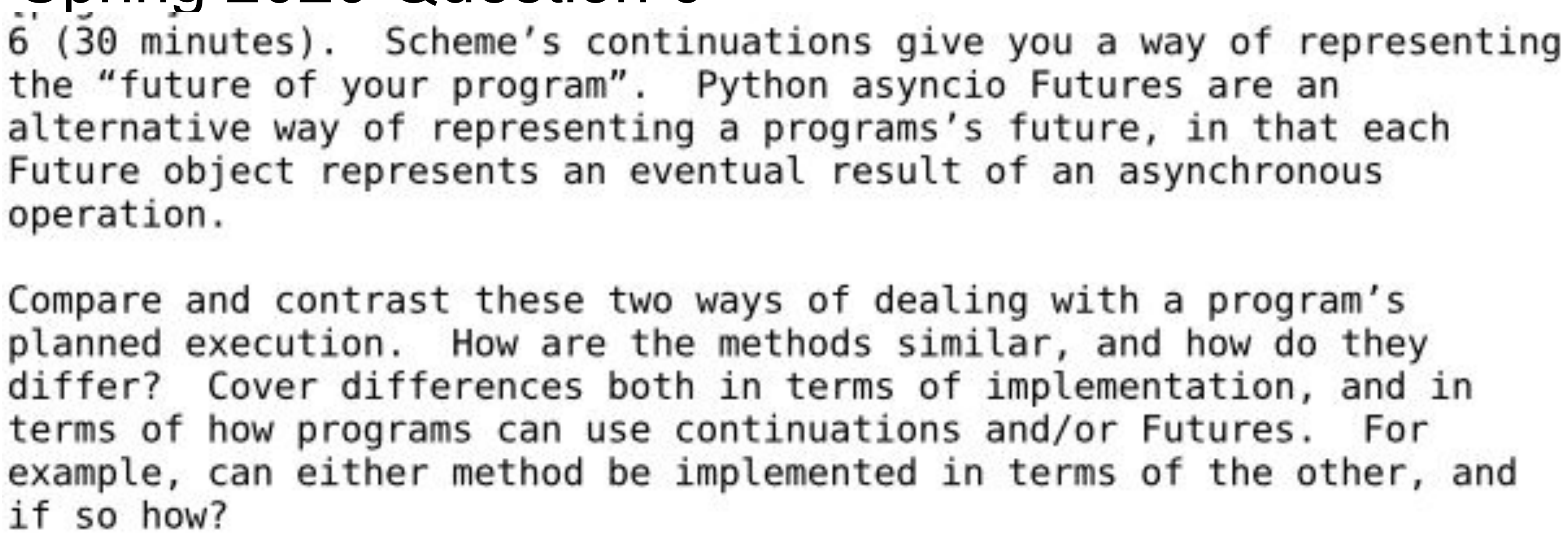
```
(define (fnx input)
  (let [(var-to-sub (caadr input))
        (body (caddr input))]
    (list (car input) '(x) (substitute body var-to-sub #t))))

(define (substitute body var-to-sub is-first)
  (cond
    ((empty? body) '())
    ((equal? (car body) 'quote) body)
    ((and is-first (equal? (car body) 'lambda))
      (if (equal? (caadr body) var-to-sub)
          ;; overload var-to-sub with #f to indicate no
substitution
        (cons 'lambda (substitute (cdr body) #f #t))
        (cons 'lambda (substitute (cdr body) var-to-sub #t)))
    )
    <see next slide for the 2 other conds>
  )
)
```

Spring 2020 Question 5b

5b (24 minutes). Implement ‘fnx’.

```
((list? (car body))
      (cons (substitute (car body) var-to-sub #t)
            (substitute (cdr body) var-to-sub #f))
    )
((and var-to-sub (equal? (car body) var-to-sub))
     ;; var-to-sub is either the variable to change to x
     ;; or it's #f, in which case don't change anything
      (cons 'x (substitute (cdr body) var-to-sub #f))
    )
    (else
      (cons (car body) (substitute (cdr body) var-to-sub #f))
    )
```

Spring 2020 Question 6



6 (30 minutes). Scheme’s continuations give you a way of representing the “future of your program”. Python asyncio Futures are an alternative way of representing a programs’s future, in that each Future object represents an eventual result of an asynchronous operation.

Compare and contrast these two ways of dealing with a program’s planned execution. How are the methods similar, and how do they differ? Cover differences both in terms of implementation, and in terms of how programs can use continuations and/or Futures. For example, can either method be implemented in terms of the other, and if so how?

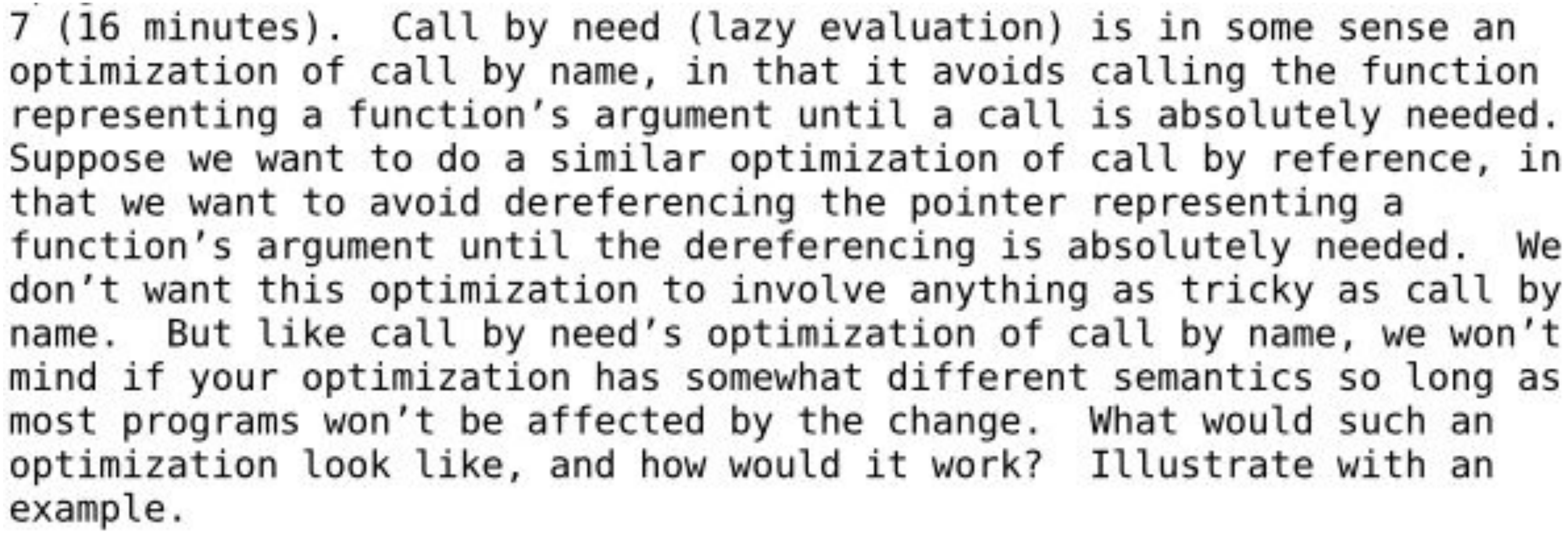
Spring 2020 Question 6

- Both allow you to return from a function after the future or continuation is set up, allowing for nonlinear program execution
- However, they are different in that you can have many futures active at once, and the event loop scheduler chooses which order to execute them, while for continuations, you have fine grained control over the order in which they will execute
- Futures are intended for activity that executes in the background, such as some form of I/O, while continuations are simply points where the code yields, although you could theoretically use them for I/O intensive tasks as well in Scheme code
- Futures can be cancelled, while continuations cannot

Spring 2020 Question 6

- Implementation: both are most likely very similar, with an instruction pointer to indicate where to resume code execution and an environment pointer to retrieve variable values
- Futures suspend themselves (yield) at a particular location in code, while continuations save the `ip` and `ep` of where to resume
- Because they are quite similar, either one could probably be implemented in terms of the other
- To implement a continuation using futures, the code that comes after the `call/cc` would be marked as an async function and one would await on it until it completes
    - Inside that code, if one needs to return immediately, one could immediately cancel the future, which would cause the awaiting to return
- To implement futures using continuations, the continuation would return to some sort of function that uses the result of the code right after the continuation. This way, the program can continue executing some other task while some I/O occurs, and when the I/O completes, it will call the continuation, thus restoring the `ip` and `ep` pair, akin to an `await` statement completing

Spring 2020 Question 7



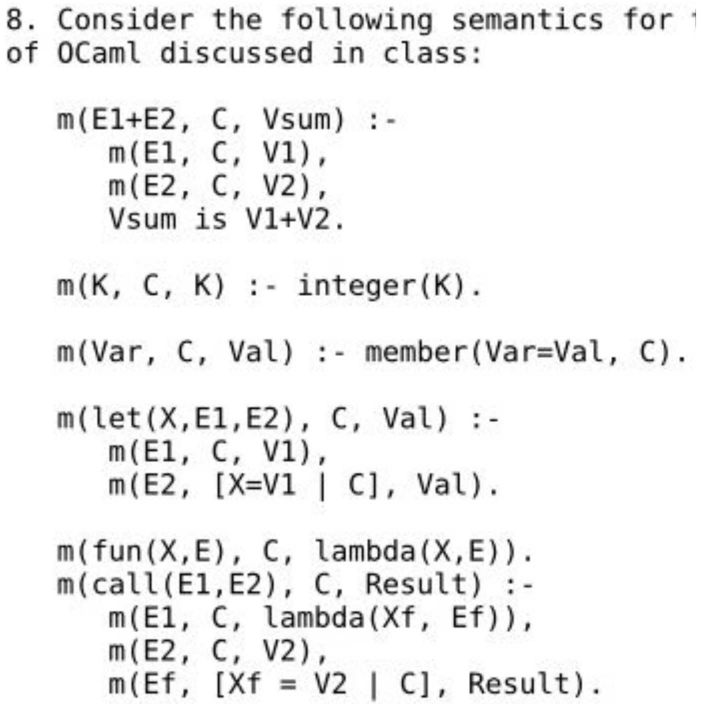
7 (16 minutes). Call by need (lazy evaluation) is in some sense an optimization of call by name, in that it avoids calling the function representing a function's argument until a call is absolutely needed. Suppose we want to do a similar optimization of call by reference, in that we want to avoid dereferencing the pointer representing a function's argument until the dereferencing is absolutely needed. We don't want this optimization to involve anything as tricky as call by name. But like call by need's optimization of call by name, we won't mind if your optimization has somewhat different semantics so long as most programs won't be affected by the change. What would such an optimization look like, and how would it work? Illustrate with an example.

Spring 2020 Question 7

- To avoid dereferencing a pointer until absolutely needed, there are 2 possible cases when we pass in a reference
  - Manipulating the reference directly
  - Performing the dereference to access what it is pointing at
- Manipulating reference directly: (such as pointer arithmetic), we do not do the actual dereferencing.
- Dereferencing the argument: then we actually perform this dereference
- This could be implemented by checking what kind of operation is being carried out at either the code or assembly instruction level is needed to perform the requested operation
- Example

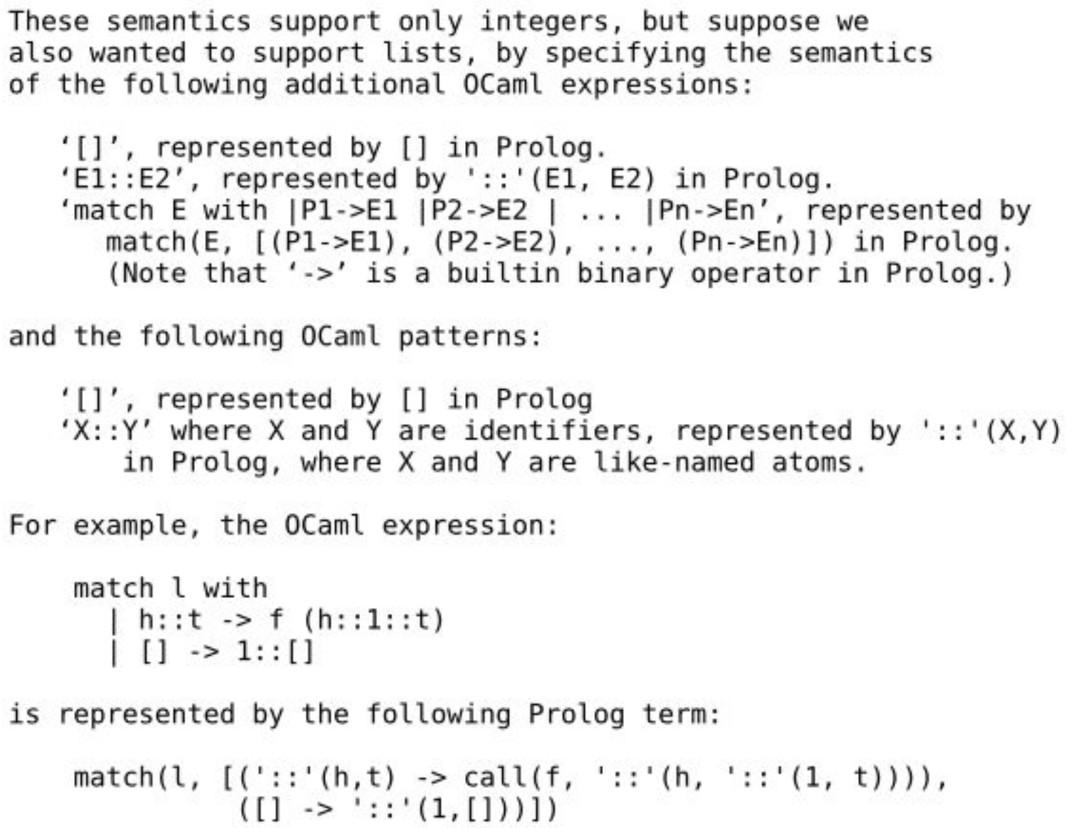
```
int someFunc(int& a){
     &a = &a + 1; // update memory location, no need to dereference
     a = a + 1; //dereference a and increment its value by 1, and save its memory address
     a = a + 2; //no need to dereference again since it's saved
     return a;
}
```

Spring 2020 Question 8



8. Consider the following semantics for
of OCaml discussed in class:

```
   m(E1+E2, C, Vsum) :-
      m(E1, C, V1),
      m(E2, C, V2),
      Vsum is V1+V2.

   m(K, C, K) :- integer(K).

   m(Var, C, Val) :- member(Var=Val, C).

   m(let(X,E1,E2), C, Val) :-
      m(E1, C, V1),
      m(E2, [X=V1 | C], Val).

   m(fun(X,E), C, lambda(X,E)).
   m(call(E1,E2), C, Result) :-
      m(E1, C, lambda(Xf, Ef)),
      m(E2, C, V2),
      m(Ef, [Xf = V2 | C], Result).
```



These semantics support only integers, but suppose we
also wanted to support lists, by specifying the semantics
of the following additional OCaml expressions:

```
   '[]', represented by [] in Prolog.
   'E1::E2', represented by '::'(E1, E2) in Prolog.
   'match E with |P1->E1 |P2->E2 | ... |Pn->En', represented by
     match(E, [(P1->E1), (P2->E2), ..., (Pn->En)]) in Prolog.
     (Note that '->' is a builtin binary operator in Prolog.)
```

and the following OCaml patterns:

```
   '[]', represented by [] in Prolog
   'X::Y' where X and Y are identifiers, represented by '::'(X,Y)
      in Prolog, where X and Y are like-named atoms.
```

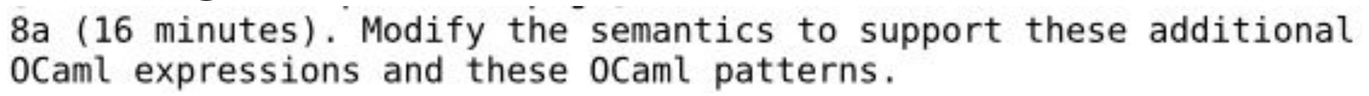
For example, the OCaml expression:

```
    match l with
      | h::t -> f (h::1::t)
      | [] -> 1::[]
```

is represented by the following Prolog term:

```
    match(l, [('::'(h,t) -> call(f, '::'(h, '::'(1, t)))),
              ([] -> '::'(1,[]))])
```

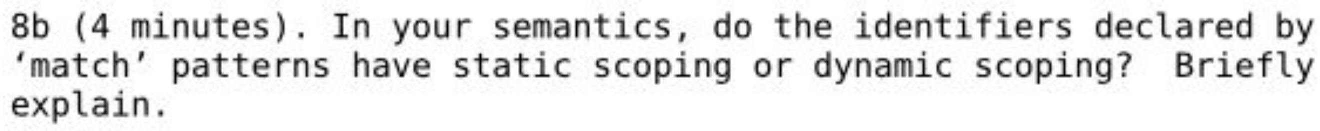
Spring 2020 Question 8a



8a (16 minutes). Modify the semantics to support these additional OCaml expressions and these OCaml patterns.

```
m([], C, []). % empty list is empty list
m('::'(X, Y), C, [X|Y]). % cons: X::Y is [X|Y]
% cons for expressions: evaluate the 2 expressions and cons together
m('::'(E1, E2), C, [V1|V2]) :- m(E1, C, V1), m(E2, C, V2).

m(match(E,[]), C, V). % match: empty match is undefined...?

% match: single match: just return the expression result
m(match(E, [P1->E1]), C, V) :- m(E1, C, V).

% match: if we match first pattern, evaluate E1 and return result
m(match(E, [P1->E1|Rest]), C, V) :- m(P1, C, E), m(E1, C, V).

% match: if we don't match first pattern, match with the rest
m(match(E, [P1->E1|Rest]), C, V) :- m(match(E, Rest), C, V).
```

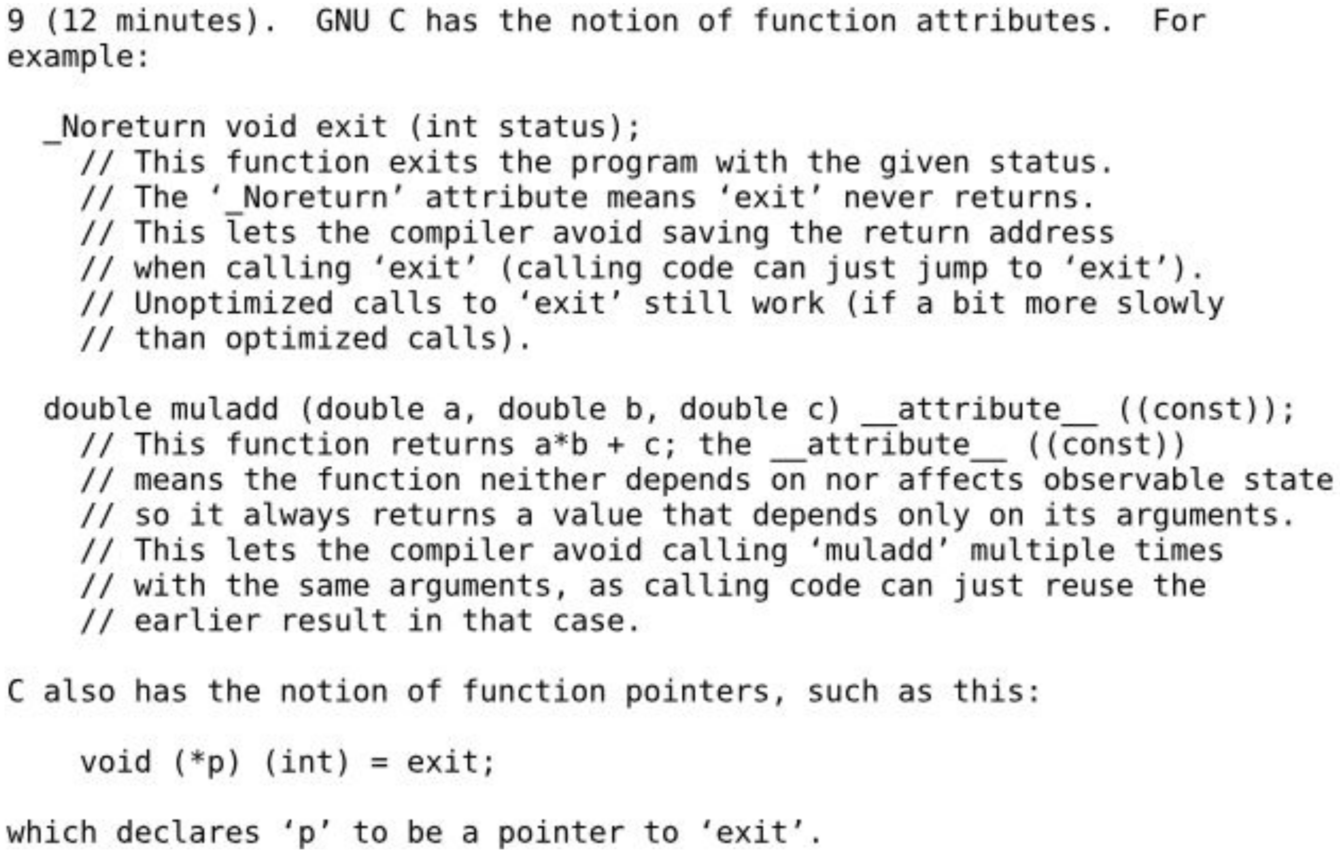
Spring 2020 Question 8b



```
8b (4 minutes). In your semantics, do the identifiers declared by
'match' patterns have static scoping or dynamic scoping?  Briefly
explain.
```

- Static scoping, as the scope of each identifier is only the rule or predicate that they are located in
- The variables in every predicate must be defined in either the head or body

Spring 2020 Question 9



9 (12 minutes). GNU C has the notion of function attributes. For example:

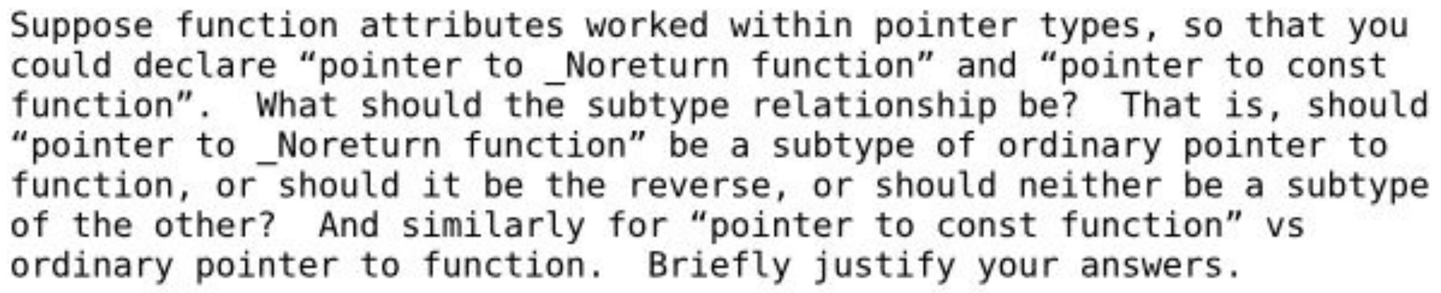
```
  _Noreturn void exit (int status);
    // This function exits the program with the given status.
    // The '_Noreturn' attribute means 'exit' never returns.
    // This lets the compiler avoid saving the return address
    // when calling 'exit' (calling code can just jump to 'exit').
    // Unoptimized calls to 'exit' still work (if a bit more slowly
    // than optimized calls).

  double muladd (double a, double b, double c) __attribute__ ((const));
    // This function returns a*b + c; the __attribute__ ((const))
    // means the function neither depends on nor affects observable state
    // so it always returns a value that depends only on its arguments.
    // This lets the compiler avoid calling 'muladd' multiple times
    // with the same arguments, as calling code can just reuse the
    // earlier result in that case.
```

C also has the notion of function pointers, such as this:

```
    void (*p) (int) = exit;
```

which declares 'p' to be a pointer to 'exit'.

Spring 2020 Question 9



Suppose function attributes worked within pointer types, so that you could declare "pointer to _Noreturn function" and "pointer to const function". What should the subtype relationship be? That is, should "pointer to _Noreturn function" be a subtype of ordinary pointer to function, or should it be the reverse, or should neither be a subtype of the other? And similarly for "pointer to const function" vs ordinary pointer to function. Briefly justify your answers.

- Subtype: a type that provides added functionality
- Pointer to a regular function enables the compiler to create code that dereferences the pointer, calls the function, and obtains a return value
- Pointer to a `_Noreturn` function need not to save the return address
  - Thus, pointer to `_Noreturn` function should be a subtype of a regular function pointer
- Similarly, a pointer to a `const` function can always be replaced by a regular function pointer, so pointers to `const` function is a subtype of regular function pointers
- Alternative way to think about it: if you have a `_Noreturn` function pointer, you can use a regular function pointer in lieu of it, but the other way around will not work: you cannot use a `_Noreturn` function pointer anywhere where a regular function pointer is used

Thank You