Programming Languages
CSCI-GA.2110.001 Fall 2021

Midterm Exam

Please either put your answers directly on this sheet or put them in a plain text, Word, or PDF document. Then, upload the document to the Assignments→Midterm Exam tab on the course website.

1. True/False. Circle or indicate the correct response. 20 points (2 points each)

   (a) T F The call stack is used to store the variables and code for each function in a program.

   (b) T F A package specification in Ada is used to declare those things (functions, procedures, types, variables, etc.) that are visible outside of the package body.

   (c) T F A higher order function, such as the MAP function in Scheme, is a function that operates on other functions (e.g. takes other functions as parameters and returns other functions as results).

   (d) T F An interpreter always executes a program directly without translating any of the program into machine code.

   (e) T F Parallelism is different from concurrency since parallelism actually requires the simultaneous execution of portions of a program, while concurrency does not.

   (f) T F In a block-structured, statically-scoped imperative language, in order for a function F to be called, a stack frame for every function surrounding the definition of F (i.e. in which F is nested at some depth) must already be on the stack.

   (g) T F A regular expression can be used to express the set of all strings of the form $\texttt{a}^n\texttt{b}^n$, that is, consisting of n `a`'s followed by n `b`'s, for any $n \geq 0$.

   (h) T F The code pointer (CP) in a closure representing a function `f` points to the portion of the stack frame where `f`'s code is stored.

   (i) T F In a dynamically typed language, types are associated with values rather than variables.

   (j) T F Any set of strings that can be defined by a CFG can also be defined by a regular expression.

2. Multiple Choice. Circle or indicate all statements that are correct, if any, for each question (there may be more than one correct statement).

28 points
(4 points each)

   (a) The following strings would not be in the set defined by the regular expression
   `(a | ε) (b | (cd)*)*`

      i. `abcdcdcdb`
      ii. `abcbcd`
      iii. `bcdcdcdb`
      iv. `bcdcdbbc`

   (b) Given the following Scheme function,

```
(define (bar x y)
  (cond ((null? x) y)
        ((= (car x) 7) (cons 'a (bar (cdr x) y)))
        (else (cons (car x) (bar (cdr x) y)))))
```

   assuming `x` and `y` are both lists of numbers,

      i. `bar` returns a list containing no occurrences of 7.
      ii. `bar` returns a list of the elements of `x` and `y`, where each 7 in `x` has been replaced by the symbol `a`
      iii. `bar` returns a list of the elements of `x` and `y`
      iv. if `x` contains no 7's, then `bar` behaves like `append`

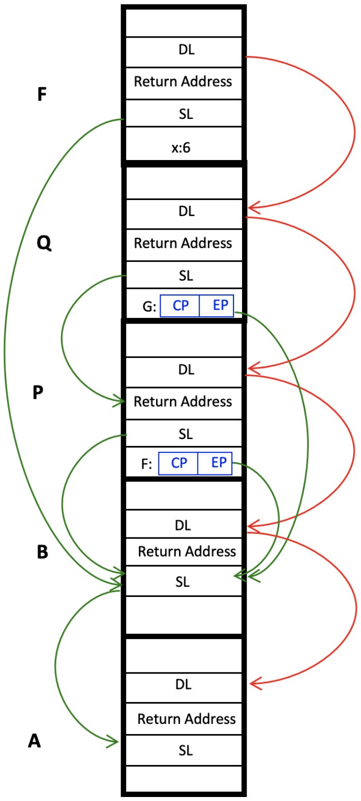
(c) Assuming static scoping, in the program whose call stack is shown to the right,

i. the function Q has been called by the function B

ii. the actual function represented by the formal parameter F is defined two scopes inside of A (i.e. in a function that is defined in A).

iii. the function Q is defined in the same scope as P is.

iv. the actual functions represented by the formal parameters F and G are defined in the same scope



(d) A CFG is ambiguous if

i. two different parse trees can represent derivations of the same string

ii. two different productions have the same non-terminal on the left hand side of the arrow.

iii. two different derivations can produce the same string

iv. it has a non-terminal that doesn't appear in any production

(e) The following are examples of concurrency.

i. no assumption can be made about the relative order of execution of two statements in a program

ii. if one block of a program consists of statements A, B, C and D and another block consists of statements W, X, Y, Z, then the order of execution is required to be A, W, B, X, C, Y, D, Z

iii. two statements in a program are executed simultaneously on two different cores of a processor

iv. the APPEND function in Scheme.

(f) The following CFGs define the set of strings of the form $\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m\mathtt{d}^k$ (that is, n `a`'s followed by m `b`'s, followed by m `c`'s, followed by k `d`'s), for any $m \geq 0$, $n \geq 0$, and $k \geq 0$.

i. `S -> aS | Sd | T`
`T -> bTc |` ϵ

ii. `S -> aSd | T`
`T -> bTc | U`
`U -> bUc |` ϵ

iii. `S -> TUV |` ϵ
`T -> Ta |` ϵ
`U -> bUc |` ϵ `V -> Vb |` ϵ

iv. `S -> aS | bSc | Sd`ϵ

(g) Suppose you wanted to write a Scheme function (max-all L), where L is a list containing numbers and nested lists (i.e. the only non-lists in L are numbers). For example,
`(max-all '(3 (4 (8 7) 5) 2 (1 3)))`
should return 8. You can assume that neither L nor any nested list within L is empty in the original call to max-all. Also, it is OK, but not necessary, for (max-all N), where N is a number, to return N. Assume nothing about the numbers in L.

The recursive reasoning would be:

i. Base Case: L is the empty list, in which case return 0.
Assumption: (max-all L1) finds the maximum number in L1, for any list L1 smaller than L.
Step: If the CAR of L is a number, return the larger of the CAR of L and the result of calling max-all on the CDR of L. Otherwise, return the larger of the result of calling max-all on the CAR of L and the result of calling max-all on the CDR of L.

ii. Base Case: L is a number, in which case return that number.
Assumption: (max-all L1) finds the maximum number in L1, for any list L1 smaller than L.
Step: return the larger of the result of calling max-all on the CAR of L and the result of calling max-all on the CDR of L.

iii. Base Case: L is a number, in which case return the number.
Assumption: (max-all L1) finds the maximum number in L1, for any L1 smaller than L.
Step: If L contains only one element, return the result of calling max-all on the CAR of L. Otherwise, return the larger of the result of calling max-all on the CAR of L and the result of calling max-all on the CDR of L.

iv. Base Case: L contains only one element, which is a number. Return the CAR of L.
Assumption: (max-all L1) finds the maximum number in L1, for any list L1 smaller than L.
Step: If L contains only one element, return the result of calling max-all on the CAR of L. Else, if the CAR of L is a number, return the larger of the CAR of L and the result of calling max-all on the CDR of L. Else, return the larger of the result of calling max-all on CAR L and the result of calling max-all on the CDR L.

3. Fill in the blanks. 24 points total

(a) In Scheme, given the following definition of the function `bar`,

4 points

`(define (bar L M)   (` append (car L) `M))`

when (bar '((1 2 3) (4 5 6)) '(7 8 9)) is called, it returns `(1 2 3 7 8 9)`.

(b) In the following Ada code,

4 points

```
procedure Prog is
   task T1 is entry Start; end T1;
   task body T1 is
   begin
      accept Start do
         put("Hello"); New_line;
      end Start;
   end T1;

   task T2;
   task body T2 is
   begin
      T1.Start;
      put("Goodbye"); New_line;
   end T2;
begin --Prog
   null;
end Prog;
```

the printing is not concurrent because the printing of "Hello" is inside the accept block, while T2 waits.

4 points

(c) Fill in the blanks in the program below such that if the language were statically scoped, it would print 15, but if the program were dynamically scoped, it would print 10. For any blank whose value doesn't matter in either case, write "25" in it.

```
program foo;
   x: integer :=   15   ;
   procedure f()
   begin
      print(x);
   end f;

   procedure g(procedure h)
      x: integer :=   10   ;
   begin
      h();
   end g;

   procedure k()
      x: integer :=   25   ;
   begin
      g(f);
   end k;
begin -- foo
  k();
end foo;
```

8 points

(d) For each parameter passing mechanism, fill in the blanks to indicate what the C program below would print, assuming C used that parameter passing mechanism. Four numbers are printed by the program.

Pass by value: 121 22 23 0 Pass by reference: 22 122 23 1

Pass by value-result: 22 22 23 1 Pass by name: 21 123 23 1

Assume that the address of the actual parameter is only computed once for pass by value-result.

```
int x = 0;
int a[3] = {21, 22, 23}; // initializes the three elements of a[]

void g(int w, int z)
{
  w = w + 1;
  z = z + 1;
  a[x] = a[x] + 100;
}

int main()
{
  g(x, a[x]);
  printf("%d %d %d ", a[0], a[1], a[2]);  // prints the 3 elements of a[]
  printf("%d\n", x);  // prints x
}
```

As in C, assume the first element of array a[] is a[0].

4 points

(e) The result of compiling and running the following Ada program (if it compiles) is ______________________.
Otherwise, it doesn't compile because A is not visible outside of Q and cannot be used in main.

```
with text_io;
use text_io;

procedure main is

   package Q is
      function Bar return integer;
      procedure Foo(X:integer);
   end Q;

   package body Q is
      A: integer := 0;

      function Bar return integer is
      begin
         return A;
      end Bar;

      procedure Foo(x:integer) is
      begin
         A := X;
      end Foo;
   end Q;

   use Q;
   package int_io is new integer_io(integer);
   use int_io;

begin
   Foo(5);
   A := A + 1;
   Put(Bar);
end main;
```