CMSC330 Spring 2008 Final Exam Solutions

1. (9 pts) Programming languages
   a. (3 pts) Give an example of a language feature (not including library functions) in Ruby & OCaml whose syntax is different but semantics are the same.

Semantics	Ruby Syntax	OCaml Syntax
Conditional statement	**foo( ) if true**	**if true then foo( )**
Nested if statement	**if … then … elsif … then … else … end**	**if … then … else if … then … else**
Assign value to x	**x = 1;**	**x := 1**
Structural equality of strings	**a == b**	**a = b**
Floating point operations	**1.0 + 1.1**	**1.0 +. 1.1**

   b. (3 pts) Give an example of a language feature (not including library functions) in Ruby & Java whose syntax is the same but semantics are different.

Statement	Ruby Semantics	Java Semantics
x = 1;	**Declare new variable x & assign value 1 to x**	**Assign 1 to closest x in static scope**
3 + 4	**Method invocation: 3.+(4)**	**Add 3 & 4**
if (x)	**Check x == nil**	**Check x == (boolean) true**
"#{x}"	**String for x.to_s**	**String "#{x}"**
x == "str"	**Structural equality (contents)**	**Physical equality (address)**

   c. (3 pts) Describe a feature of an old programming language which proved vexing for programmers.
      Spaces in identifiers, no reserved keywords, dynamic scoping, etc…

2. (12 pts) Ruby features
   What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

   a.
```
a = [2,3]
a[3] = "c"
a.push("b")
puts a
```
   // (3 pts) 2 3 nil c b

   b.
```
if  "CMSC 330" =~ /([0-9]+)/ then
    puts $1
else
    puts "Error"
end
```
   // (3 pts) 330

   c.
```
s = "CMSC 330 is too HARD!"
s.scan(/[A-Z]+/) { |x| puts x }
```
   // (3 pts) CMSC HARD

   d.
```
h = {2 => 3, 1 => 2, 0=>1}
puts "#{h[1]} #{h[2]}"
```
   // (3 pts) 2 3

3. (20 pts) Ruby programming

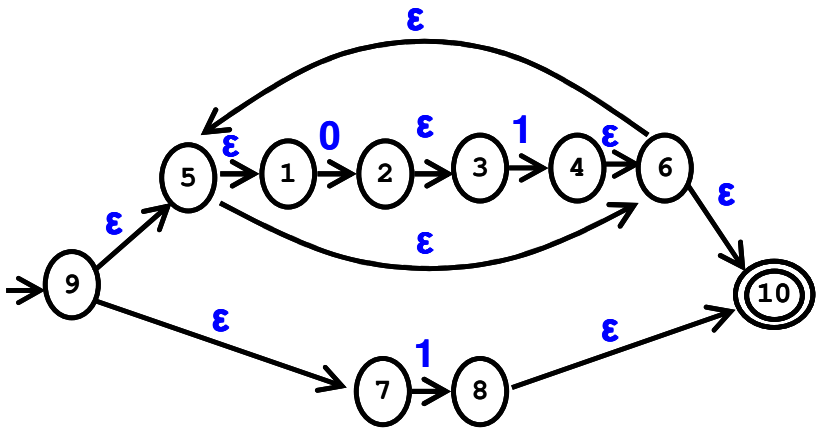
Consider the following programming problem. We need to read from a text file a list of cars and the years they were produced. The file contains a number of lines of the form <car> <year>, where <car> is composed of letters, <year> is composed of digits, and the two are separated by a single space. Write a Ruby program to read in the list of cars and their model years, and output the list of cars and years in sorted order, with each car on a separate line. Any lines not in the correct format should be ignored.

Skeleton Code	Example Input	Example Output

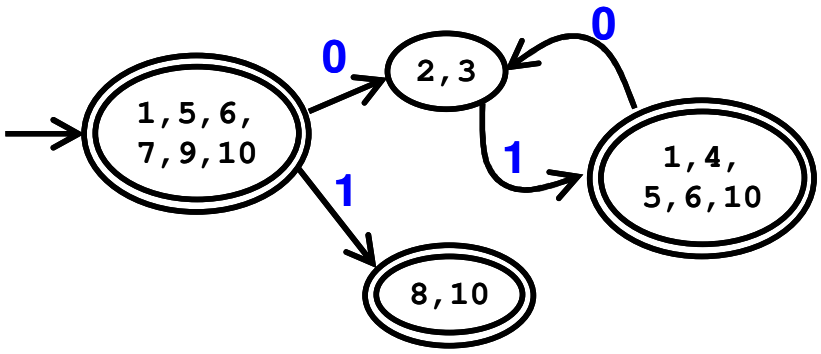
file = File.new(ARGV[0], "r")
until file.eof? do
  line = file.readline
  …
end
// a = Array.new ; a.sort!
// print 1 ; puts 2
	yaris 2008
	prius 2004
	prius 2007
	rav4 1998
	yaris 2006
	prius 2006
	camry 1983
		camry 1983
		prius 2004 2006 2007
		yaris 2006 2008

```
cars = { }                                  // (2 pt) Allocate Hash
file = File.new(ARGV[0], "r")               // (2 pt) Open & read file
until file.eof? do
   line = file.readline
   if line =~ /([a-zA-Z]+) ([0-9]+)/        // (4 pt) RE for entry
      if ! cars[$1]                         //     lowercase only is fine
         cars[$1] = Array.new               // (2 pt) init new Hash entry
      end
      cars[$1].push($2)                     // (2 pt) store model year
   end
end
labels = cars.keys                          // (2 pt) get & sort list of cars
labels.sort!
labels.each { |x|
   print "#{x} ="
   cars[x].sort!.each { |y| print " #{y}" } // (2 pt) get & sort model year
   puts                                     // (2 pts) print format correct
}                                           // (2 pts) print data correct
```

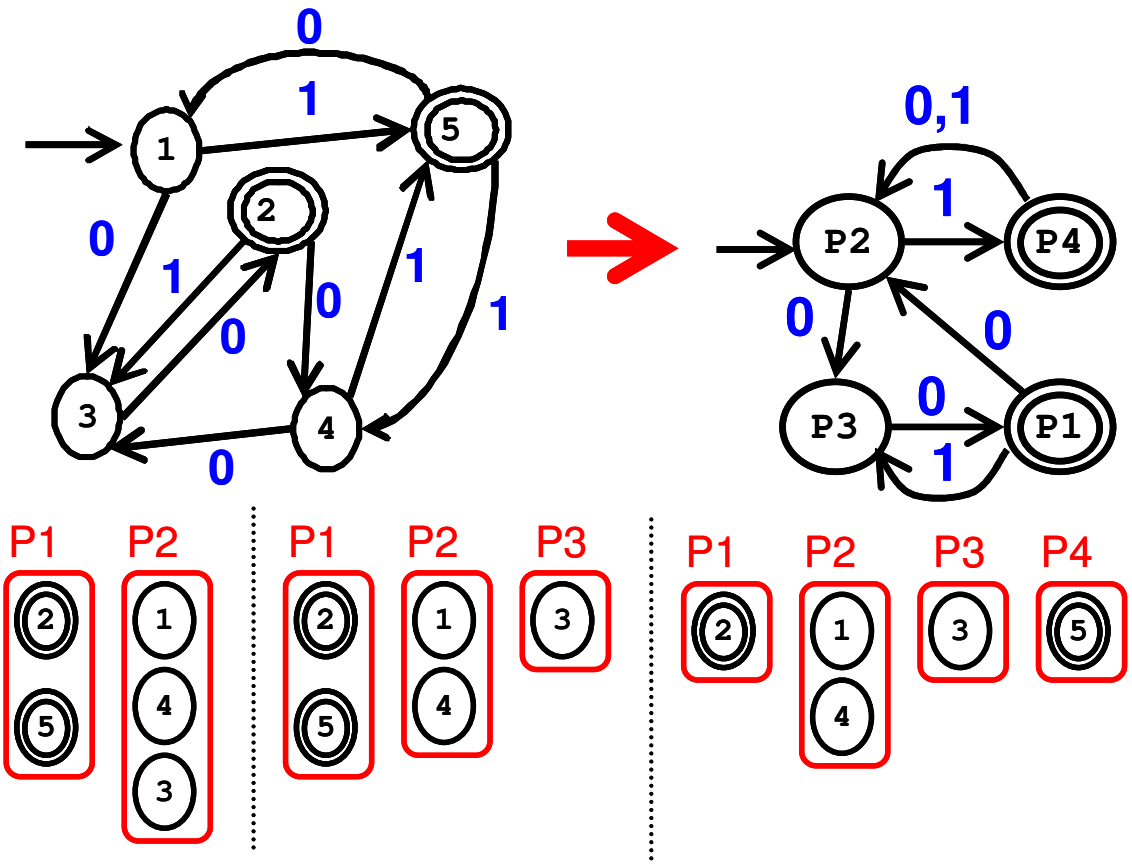
4. (30 pts) Regular expressions finite automata
   a. (6 pts) Reduce the regular expression (01)* | 1 to an NFA, using the algorithm shown in class.



   b. (12 pts) Reduce the NFA to a DFA using the subset algorithm. Show the NFA states represented by each DFA state.



c. (12 pts) Minimize the following DFA using Moore's algorithm. Show the entire sequence of partitions created (list the DFA states in each partition), and explain why each partition was split.



5. (10 pts) Grammars and parsing

   a. (4 pts) Write a grammar for $a^x b^y a^z$, where $x = y - z$, for $x,y,z \geq 0$

      S → AB
      A → aAb | epsilon **// A → aAb applied x times**
      B → bBa | epsilon **// B → bBa applied z times**

   b. (6 pts) Rewrite the following grammar

      S → S and S | not S | true | false

      so that "and" is right associative & has higher precedence than "not".

      S → not A | A **// "not" → lowest precedence**
      A → B and A | B **// "and" → right associative**
      B → true | false

6. (24 pts) OCaml Types & Type Inference

   Give the type of the following OCaml expressions:

   a. (3 pts) let f x = match x with **// 'a list list -> 'a list**
      [] -> []
      | (h::_) -> h

   b. (3 pts) let f x y = y (x+1) **// int -> (int -> 'a) -> 'a**

   Write an OCaml expression with the following types:

   c. (3 pts) int -> (int -> 'a) -> 'a **// Example: let f x y = y (x+1)**
   d. (3 pts) (int -> 'a) -> int -> 'a **// Example: let f x y = x (y+1)**

   What are the values of the following OCaml expressions? If an error exists, describe the error.

   e. (3 pts) let x = x in 1 **// Error: unbound x**
   f. (3 pts) let x = 1 in if x > 0 then x else print_string "Error"
      // Error: types of if then (type = int) and else (type = ( )) do not match
   g. (3 pts) (fun x -> if x > 0 then x else (–x)) 2 **// 2**
   h. (3 pts) Are type inference and strong typing orthogonal language features? Explain.

      No.
      1) Without strong typing, a variable's type may change (many implicit casts) making it difficult to infer what that variable's type is supposed to be.
      2) Strong typing ensures a variable's type does not change, making it easier to infer what that variable's type is supposed to be.

7. (14 pts) OCaml Polymorphic Types

Consider a OCaml module Bst that implements a binary search tree:

```
module Bst = struct
  type bst = Empty | Node of int * bst * bst
  let empty = Empty              (* empty binary search tree          *)
  let is_empty = function        (* return true for empty bst         *)
     Empty -> true
    | Node (_, _, _) -> false
   (* Implement the following functions
           val height : bst -> int
           val post_fold : ('a -> int -> 'a) -> 'a -> bst -> 'a
    *)
     let rec height =            (* height of bst      *)
     let rec post_fold f a t =   (* apply f to nodes of t in postorder    *)
  end
```

a. (6 pts) Implement height. You may use Pervasives.max ('a -> 'a -> 'a), which returns the greater of its two arguments.

```
let rec height = function
    Empty -> 0
  | Node (_, left, right) -> 1+(Pervasives.max (height left) (height right))
```

b. (8 pts) Implement post_fold as a version of fold which performs a *postorder* traversal of the tree (visits in order 1: left subtree, 2: right subtree, 3: node).

```
let rec post_fold f a t = match t with
    Empty -> a
  | Node (m, left, right) -> f (post_fold f (post_fold f a left) right) m
```

Example:

```
let s = Node(9,Node(5,Node(3,empty,empty),empty),Node(12,empty,empty)) ;;
height s ;;                                        (* returns 3 *)
List.rev (post_fold (fun a m -> m::a) [] s) ;;   (* returns [3;5;12;9] *)
```

8. (12 pts) Recursive Descent Parser in OCaml

The example OCaml recursive descent parser 15-parseArith_fact.ml employs a number of shortcuts. For instance, the function parseT handles the grammar rules for

T → F * T | F

directly instead of rewriting the grammar, creating the following productions:

T → ? C → ?

You must identify where code corresponding to parseC was inserted directly in the code for parseT, in the comments below:

```
let rec parseT lr =                          (* parseT *)
  let x = parseF lr in                       (* 1: ? → ? *)
  match !lr with                             (* parseC *)
    | ('*'::t) ->                            (* 2: if lookahead = First( ? ) *)
      lr := t;                               (* 3: ? → ? *)
      Prod (x,parseT lr)
    | _ -> x                                 (* 4: ? → ? *)
```

a. (2 pts) What rule should have been applied to the productions for T?
   Left factoring
b. (2 pts) What productions for T & C would be created by applying the rule?
   T → F C C → * T | epsilon
c. (2 pts) What production should appear in place of ? in comment 1?
   T → F C
d. (2 pts) What sentential form should appear in place of ? in comment 2?
   *** T**
e. (2 pts) What production should appear in place of ? in comment 3?
   C → * T
f. (2 pts) What production should appear in place of ? in comment 4?
   C → epsilon

9. (12 pts) Function arguments, scoping, & FP vs. OOP
   a. (4 pts) Explain why upwards funargs are more difficult to implement than downwards funargs.
      Because activation records / stack frames must be stored on the heap, since a function's local variables may be accessed even after the function returns.
   b. (4 pts) Explain why dynamic scoping can be more confusing than static scoping
      Because the binding for a symbol depends on the sequence of function calls at runtime, since symbols are bound to the closest active function scope.
   c. (4 pts) Describe how OOP may be used to simulate functional programming.
      Functions may be passed as arguments by passing an object with a known method, using interfaces to specify what method must be present.

10. (12 pts) Parameter passing

a. (2 pts) Describe lazy evaluation.

Arguments to functions are evaluated at the last possible moment, just before they're needed in the function body.

b. (3 pts) Explain how to implement lazy evaluation in a language using call-by-value.

Place function arguments int the body of thunks (function with no arguments), and pass the thunks instead. Within the procedure body, replace formal parameters with thunk invocations.

Consider the following C code.

```
void swap(int f, int g) {
  int tmp = f;    f = g;    g = tmp;
}
int main( ) {
   int i = 2;
   int a[] = {2, 0, 1};
   swap(i, a[i]);
   printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

c. (2 pts) Give the output if C uses call-by-value

2 2 0 1 (no effect).

d. (2 pts) Give the output if C uses call-by-reference

1 2 0 2 (swap i & a[2])

e. (3 pts) Give the output if C uses call-by-name

1 2 2 1 (i = 2, tmp = i, i = a[i], a[i] = tmp)

11. (18 pts) Polymorphism

a. (3 pts) Explain the difference between subtype polymorphism & overloading.

With subtype polymorphism a function foo(A x) may accept 2 arguments of different types A & B if B is a subclass of A. With overloading a function foo( ) may accept 2 arguments of different types A & B if 2 copies of the function are defined: foo(A x) & foo(B x).

b. (3 pts) Explain why subtype polymorphism causes problems for container classes (e.g., Set, Map, Stack).

Container of B is not a subclass of Container of A, even if B is a subclass of A.

OR

Because trying to write a polymorphic container class requires downcasting objects (from subclass) removed from a container, where the legality of the downcast cannot be determined until runtime (possibly causing an exception).

c. (4 pts) Generic programming is used in object-oriented programming languages to solve these problems. Name and describe the technique used to implement generic programming in Java without changing the JVM (Java virtual machine).

Translation with type erasure is used to implement generic programming by having the Java compiler erase parametric types and add downcasts where needed.

Consider the following Java classes:

```
class A { … }
class B extends A { … }
class C extends A { … }
```

HashSet<E> is a class supporting generics in the Java 1.5 class library.
Explain why the following code is or is not legal

d. (2 pts) int count(HashSet<?> s) { … } … count(new HashSet<C>( ));
Legal since C matches ?.

e. (3 pts) int count(HashSet<? extends A> s) { … } … count(new HashSet<C>( ));
Legal since C is a subclass of A (i.e., C extends A).

In more detail: Actual parameter type (HashSet<C>) matches formal parameter type (HashSet<? extends A>), since "? extends A" can match A and its subclasses B & C.

f. (3 pts) int count(HashSet<? super B> s) { … } … count(new HashSet<C>( ));
Illegal since C is not a superclass of A (i.e., C super B is not true).

In more detail: Actual parameter type (HashSet<C>) does not match formal parameter type (HashSet<? super B>), since "? super B" can match B and its superclasses A & Object.

12. (16 pts) Java multithreading

a. Using Java locks & conditions, you must implement a synchronization construct called Partner. A Partner object is created with a value 0. When a thread calls the method next( ), it waits until a 2nd thread also calls next( ). The 2nd thread returns the current Partner value and continues without blocking, waking up the 1st thread. The next thread to invoke next( ) and acquire the lock for Partner will increment the Partner value and return its previous (non-incremented value). Be careful, since a 3rd thread may invoke next( ) before the 1st thread can wake up and be partnered with the 2nd thread, forcing the 1st thread to wait for a new partner!

```
class Partner {
        public int next( ) { … }
}

Partner p = new Partner( );
thread1.run( ) { … p.next( ); } // blocks & returns 0 after thread 2 calls next( )
thread2.run( ) { … p.next( ); } // next( ) returns 0 immediately
…
thread3.run( ) { … p.next( ); } // blocks & returns 1 after thread 4 calls next( )
thread4.run( ) { … p.next( ); } // next( ) returns 1 immediately
```

```
public class Partner {
        int current = 0;                                    // shared modifiable data
        int partnerNum = 0;                                 // shared modifiable data
        Lock lock = new ReentrantLock();
        Condition ready = lock.newCondition();

        public int next( ) throws InterruptedException {
                int returnVal;
                lock.lock();                // prevent data race on current / partnerNum
                current++;                                  // # of threads at Partner
                if (current == 2) {                         // partner thread is waiting
                        ready.signalAll();                  // wake up partner thread
                        returnVal = partnerNum;             // return value = current partnerNum
                                                            // continue execution
                }
                else {
                        while (current < 2) {               // if current = 1, wait for partner thread
                                ready.await();              // sleep until partner available
                        }                                   // use while in case threads cut in front
                        returnVal = partnerNum;             // return value = current partnerNum
                        partnerNum++;           // increment partnerNum for next pair of threads
                        current -= 2;                       // reduce # of threads by 2
                }
                lock.unlock();                              // release lock
                return returnVal;                           // return partnerNum
        }
}
```

Code is similar to the barrier example in the homework. Differences include

- **Keeping track of an additional field partnerNum to determine the number returned by next( ) for each pair of partnered threads.**
- **Instead of an asynchronous call to reset( ), code in next( ) explicitly decrements the number of threads by 2 after a pair of threads partner up.**
- **Next( ) must return a value. The return statement must be outside the lock( ) unlock( ) region to prevent deadlock.**

13. (6 pts) Memory allocation & garbage collection

Consider the following Java code.

```
public foo(Object a[ ] ) {
        a[0] = new Object( );   // object 1
        a[1] = new Object( );   // object 2
        a[2] = new Object( );   // object 3
        a[1] = a[2];
}
```

a. (3 pts) What object(s) are garbage when foo( ) returns? Explain why.

Object 2 since a[1] references object 3 after a[1] = a[2], and there are no other references to object 2.

b. (3 pts) Describe why garbage collection may be preferred over manual memory allocation.

Less programmer effort, fewer errors introduced due to incorrect frees, memory can be compacted, performance may actually be better.

14. (8 pts) Markup & query languages

a. (5 pts) Creating your own XML tags, write an XML document that organizes the following information: Obama and Hillary are Democrats, McCain is a Republican.

```
<candidates>
    <candidate>
        <name>Obama</name>
        <party>Democrat</party>
    </candidate>
    <candidate>
        <name>Hillary</name>
        <party>Democrat</party>
    </candidate>
    <candidate>
        <name>McCain</name>
        <party>Republican</party>
    </candidate>
</candidates>
```

b. (3 pts) Explain how query languages are different from programming languages.

Query languages are designed to retrieve and manage data in databases and information systems, and (usually) cannot be used to program arbitrary algorithms (i.e., not Turing-complete languages).

15. (20 pts) Lambda calculus & encodings

Evaluate the following λ-expressions as much as possible

a. (3 pts) (λx.λy.x) a b

(λx.λy.x) a b → (λy.a) b → a

b. (3 pts) (λz.λy.z y) y z

(λz.λy.z y) y z → (λz.λa.z a) y z → (λa.y a) z → y z

Using the appropriate λ-calculus encodings

c. (6 pts) Given:

or = λx.λy.((x true) y)
true = λx.λy.x
false = λx.λy.y

Prove : or false true = true (false or true ↔ or false true)

or false true **// replacing or w/ encoding**
= (λx.λy.((x true) y)) false true **// β-reduction: x → false**
= (λy.((false true) y)) true **// β-reduction: y → true**
= (false true) true **// replacing false w/ encoding**
= ((λx.λy.y) true) true **// β-reduction: x → true**
= (λy.y) true **// β-reduction: y → true**
= true

d. (8 pts) Given:

0 = λf.λy.y
1 = λf.λy.f y
2 = λf.λy.f (f y)
M + N = λx.λy.(M x)((N x) y)

Prove : 0 + 2 = 2 (0 + 2 ↔ + 0 2)

(+ 0 2) **// replacing + w/ encoding**
= λx.λy.(0 x)((2 x) y) **// replacing 0 w/ encoding**
= λx.λy.((λf.λy.y) x)((2 x) y) **// β-reduction: 1st f → x**
= λx.λy.(λy.y)((2 x) y) **// β-reduction: 2nd y → (2 x) y**
= λx.λy.(2 x) y **// replacing 2 w/ encoding**
= λx.λy.((λf.λy.f (f y)) x) y **// β-reduction: 1st f → x**
= λx.λy.(λy.x (x y)) y **// β-reduction: 2nd y → y**
= λx.λy.x (x y) **// α-conversion: replace x with f**
= λf.λy.f (f y) **// apply encoding for 2**

= 2

16. (7 pts) Operational Semantics

a. (3 pts) Describe the purpose of analyzing the semantics of a program using techniques such as operational semantics.

To formally (and precisely) specify the meaning of a program, in order to verify the correctness of an algorithm through formal verification.

b. (4 pts) Write a proof in operational semantics that "(+ a b)" has the value 3, if the value of "a" is 1 and the value of "b" is 2.

$$\frac{a \rightarrow 1 \qquad b \rightarrow 2}{(+\ a\ b) \rightarrow 3}$$

17. (16 pts) Extra credit

a. (10 pts) Given:

Z = λf. (λx. f (λy. x x y)) (λx. f (λy. x x y))

fact = λf. λn.if n = 0 then 1 else n * (f (n-1))

What happens when you try to evaluate (Z fact) 1? You do not need to expand if.

(Z fact) 1 **// replacing Z w/ encoding**
= ((λf.(λx.f (λy. x x y)) (λx.f (λy. x x y))) fact) 1
= ((λx. fact (λy. x x y)) (λx. fact (λy. x x y))) 1
= (fact (λy. (λx. fact (λy. x x y)) (λx. fact (λy. x x y)) y)) 1
= (fact (λx. fact (λy. x x y)) (λx. fact (λy. x x y))) 1
= (fact (Z fact)) 1
= if 1 = 0 then 1 else 1 * ((Z fact) 0)
= 1 * ((Z fact) 0)
= 1 * (fact (Z fact) 0)
= 1 * (if 0 = 0 then 1 else 0 * ((Z fact) (-1))
= 1 * 1
= 11

b. (6 pts) Given

Y = λf.(λx.f (x x)) (λx.f (x x))

What is the relationship between Y and Z?

Both are (fixed point) combinators that can be used to achieve recursion for fact. The difference is that with Z fact is passed (λy. x x y), while with Y fact is passed (x x). This simply changes when the arguments must be evaluated, if the language is call-by-value instead of call-by-name.