

CMSC330 - SUMMER 2006 - MIDTERM 2

INSTRUCTOR: GUILHERME FONSECA

- Write all answers in the answers book provided.
- You can keep the exam. Only return the answers book.
- You are allowed to consult one letter-size paper, handwritten on one side. Besides that, the exam is closed book, closed notes.
- There are 10 question, totaling 100 points, in this exam. You have 1 hour and 20 minutes to finish it.

(1) (15 points) Give short answers:

- (a) Garbage collection reduces memory leaks. Give one short fragment of C language code that causes a memory leak.
- (b) In the C programming language, a function can be passed to another function through function pointers. Why are closures more powerful than function pointers?
- (c) Precisely describe a language that is **context-sensitive**, but **not context-free**. You can describe it in English, or using the operators discussed in class.

(2) (10 points) The *map* function can be written in OCaml as:

```
let rec map f l =
  match l with
  [] -> []
  | (h::t) -> (f h)::(map f t)
```

There is already a *collect* function in **Ruby** that has essentially the same behavior. Without using Ruby's *collect* function, write a *map* function in **Ruby** that does the same thing, using codeblocks. You can use any other Ruby features and methods, but not the *collect* method. The following Ruby code should work with your *map* function and return Array of the square of the elements:

```
map([5, 5, 3, 1]) {|x| x*x}      # Returns [25, 25, 9, 1]
```

(3) (15 points) What is the exact type of *f* in each of the OCaml lines below. Your answer must be written using the OCaml syntax for specifying types.

- (a) `let f = (false, 1 + 2)`
- (b) `let f x = x`
- (c) `let f x y = x*x > y*y`
- (d) `let f (x,y) = (x+3)::y`
- (e) `let f x y z = x::(y z)`

- (4) (10 points) Write a **partition** function in OCaml that receives a value p and a list ℓ . The **partition** function should return a tuple of two lists of elements from ℓ . The first list in the tuple consists of the elements that are less than p (according to the OCaml $<$ operator). The second list in the tuple consists of the elements that are greater than or equal to p (according to the OCaml \geq operator). Do not use any library function. Feel free to write helper functions. The lists you return should have the elements in the same order as they were in the original list.

```
partition 10 [11; 12; 7; 9; 23; 5; 8; 12]
(* Returns ([7; 9; 5; 8], [11; 12; 23; 12]) *)
```

The type of your function should be:

```
'a -> 'a list -> ('a list * 'a list)
```

- (5) (10 points) Suppose some programmer wrote two OCaml functions f and g , that are supposed to do exactly the same thing. The problem is that you are not sure if both implementations are correct. First, you confirmed that the types of f and g are correct. To make sure the functions are the same, you would like to give each element x from a list of test inputs to both $f(x)$ and $g(x)$, and answer whether $f(x) = g(x)$ for all values x in the list. The function *testEq* that you need to write takes 3 arguments using currying. The arguments are, in order: the first function f , the second function g , and the list of test inputs.

You need to write a function *testEq* with the following behavior:

```
let f1 x = x+x;;
let f2 x = 2*x;;
let f3 x = x*x;;

testEq f1 f2 [0;3;4;7];;    (* returns true *)
testEq f2 f3 [0;3;4;7];;    (* returns false *)
testEq f2 f3 [0;2];;        (* returns true *)
```

The two functions that will be passed to *testEq* take a single argument and return a value that can be compared with OCaml's $=$ operator. Besides that, the functions f and g have no side effect. Feel free to write other helper functions, but you must write the OCaml code for all functions that you use.

- (6) (8 points) Write a **regular grammar** for the following language over the alphabet $\Sigma = \{a, b\}$:

$\{w \mid w \text{ contains at most one } b\}$

- (7) (10 points) Write a **context-free grammar** for the following language over the alphabet $\Sigma = \{a, b, c\}$:

$\{a^i b^j c^{2i+j} \mid i, j \geq 0\}$

- (8) (12 points) Consider the following BNF grammar. This grammar generates strings similar but not necessarily identical to Prolog clauses. Quotes are not part of the string and are used to indicate the terminal symbols that are not alphanumeric.

$$\begin{aligned} \langle \text{clause} \rangle &::= \langle \text{func} \rangle \text{ ":" } \langle \text{conj} \rangle \text{ "."} \\ \langle \text{conj} \rangle &::= \langle \text{conj} \rangle \text{ "," } \langle \text{disj} \rangle \mid \langle \text{disj} \rangle \\ \langle \text{disj} \rangle &::= \langle \text{disj} \rangle \text{ ";" } \langle \text{func} \rangle \mid \langle \text{func} \rangle \mid \varepsilon \\ \langle \text{func} \rangle &::= \langle \text{id} \rangle \text{ "(" } \langle \text{func} \rangle \text{ ")" } \mid \langle \text{id} \rangle \\ \langle \text{id} \rangle &::= x \mid y \mid z \end{aligned}$$

- (a) Which operator has higher precedence: `,` or `;`?
 (b) What is the associativity of the `,` operator?
 (c) What is the associativity of the `;` operator?

* Tell whether each of the following expressions can or cannot be generated by the grammar above. (Just **Yes** or **No**, no justification necessary.)

- (d) $x(y):-.$
 (e) $:-x(y).$
 (f) $x(y):-y(y(z)),z(y);x.$
 (g) $x(x):-y(y):-z(z).$
 (h) $x(y(z)):-x().$

- (9) (5 points) Write a formal regular expression for the language below. The alphabet is $\Sigma = \{a, b, c\}$. The only operators allowed are concatenation, $*$ and $|$ (do not write a Ruby regular expression).

$\{w \mid \text{all consonants are adjacent to a consonant on at least one side in } w \}$

Notice that b and c are the only consonants in the alphabet Σ . For example, a , bb , bc , and $aabbabcba$ are in the language, but b , ab , and aba are not.

- (10) (5 points) Convert the following NFA to a NFA without ε transitions.

