UCLA CS 131 Midterm, Fall 2016
100 minutes total, open book, open notes,
closed computer

Name: Richard Sun Student ID: 904444918

1	2	3	4	5	6	7	total
10	10	20	8	6	21	10	85

1 (10 minutes). A nonterminal N is "nearly terminal" if all rules with N as the left hand side contain only terminal symbols in the right hand side. For example, if N has no rules, or has only a rule with an empty right hand side, then N is nearly terminal. Write an Ocaml function nearly_terminal G N that returns true if N is nearly terminal in the grammar G, and false otherwise. Use the same format for grammars and symbols as in Homeworks 1 and 2.

2 (10 minutes). Consider the two types 'char **' and 'char const * const *' in C and C++. Should the first be a subtype of the second? or vice versa? Justify your two answers by appealing to operations on the types. char** → char* → char

char const* const* → const char* → const char

3. The java.util.Collections class defines a method with this API:

```
static <T> List<T>
  unmodifiableList(List<? extends T> list);
```

Return an unmodifiable view of the specified list. That is, if v = unmodifiableList(u), then v is like u except that you cannot change the list by acting on v; you must change it by acting on u instead.

3a (10 minutes). Suppose ls is of type List<String> and lo is of type List<Object>. Explain why the assignment lo=ls is invalid, but the assignment lo=unmodifiableList(ls) is OK.

3b (10 minutes). Given the above, explain why the type T of the expression unmodifiable(X) cannot be inferred merely by looking at the type of X. What else can affect T? Explain with a brief example.

4. Consider the following syntax, written in ISO EBNF. The start symbol is "syntax". Assume that the "letter" token stands for any ASCII letter, that the "digit" token for any ASCII digit, and that all white space is ignored.

```
syntax = syntax rule, {syntax rule};
syntax rule = meta id, '=', defns list, ';';
defns list = defn, {'|', defn};
defn = term, {',', factor};
term = repeated sequence | meta id | empty;
repeated sequence = '{', defns list, '}';
meta id = letter, {letter | digit};
empty = ;
```

4a (8 minutes). Translate this grammar into an equivalent grammar that uses the OCaml-based notation of Homeworks 1 and 2. Note and remove any blind alleys in your translation. If such a translation is not possible for some reason, explain why not, and go as far in the translation as you can.

4b (5 minutes). If you gave your translated grammar to a working solution to Homework 2, what major problem could happen? Show all the points in the translated grammar where this problem might crop up.

5. For each of the following OCaml expressions, give its type and value.

5a (3 minutes). `let i x = x in i i`

5b (3 minutes).

```
let expr = "3 + 4"
in let lvalue = "a"
   in [expr, ["("; expr; ")"];
       lvalue, ["$"; expr]]
```

5c (6 minutes).

```
let accept_all derivation string =
      Some (derivation, string)
in let accept_empty_suffix derivation =
      function
        | [] -> Some (derivation, [])
        | _ -> None
   in accept_all accept_empty_suffix "aooogah!"
```

6. Suppose you want a wrapper class that contains a value of type T and has a getter 'get' and a setter 'set' to load and store a value. Once you have stored a non-null value, later calls to 'set' are no-ops. You want instances of the wrapper to be safe and fast in a multithreaded application. Consider the following alternative implementations C0, ..., C4 of a wrapper class:

```
class C0<T> {
  T val;
  public void set(T v) {
    if (val == null) { val = v; }
  }
  public T get() { return val; }
}

class C1<T> {
  static volatile int v; int w;
  T val;
  public synchronized void set(T v) {
    if (val == null) { val = v; }
  }
  public T get() { w = v; return val; }
}

class C2<T> {
  T val;
  public synchronized void set(T v) {
    if (val == null) { val = v; }
  }
  public synchronized T get() { return val; }
}

class C3<T> {
  volatile T val;
  public synchronized void set(T v) {
    if (val == null) { val = v; }
  }
  public T get() { return val; }
}

class C4<T> {
  T val;
  public synchronized void set(T v) {
    if (val == null) { val = v; }
  }
  public T get() { return val; }
}
```

6a (15 minutes). Which of the classes C0 through C4 are safe in a multithreaded application? Briefly justify your answers by appealing to the JMM.

6b (10 minutes). Explain the performance implications of C0 through C4. Order the implementations roughly in order of performance, assuming the application has many threads that frequently the 'set' and 'get' methods on the same object.

7 (10 minutes). Suppose I design a new language There-Is-No-Try-Java. This new language is just like Java, except that there is no "try" keyword; one simply writes exactly the same code as in Java, but without the "try". For example, instead of this:

```
try { s = buf.readLine (); }
catch (IOException e)
  { System.out.println (e); }
```

you write this:

```
{ s = buf.readLine (); }
catch (IOException e)
  { System.out.println (e); }
```

Is the syntax of There-Is-No-Try-Java ambiguous? If so, illustrate the ambiguity with a program that can be parsed in two different ways. If not, explain why not.

CS 131 midterm

1. Using HW2 format (start, production function)

```
let nearly_terminal G N =
  let start, rules = G in
  let rec all_terminal rhs =
    match rhs with
    | [] -> true
    | (N _)::_ -> false
    | _::t -> all_terminal t
  in
  let rec check_rhs altlist =
    match altlist with
    | [] -> true
    | h::t when all_terminal h -> check_rhs t
    | _ -> false
  in
  check_rhs (rules N)
```

2. char** should be a subtype of char const * const*, since more operations can be be performed on char **. It is possible to do:

```
char **p = [ ];
p[0] = NULL;
```

but that operation would be illegal if p was declared as a char const *

3a. If you converted a List<String> to List<Object>, then it is possible to insert Objects that are not Strings. Since this violates List<String>'s type constraints the cast is not valid. Since unmodifiable List cannot be changed, inserting invalid items is not a problem.

3b. The return type can affect the type of T. By looking at the type of the variable that the return value is assigned to, T should be whatever type of list that variable has. If X does not extend T, then there is a type error.

4a. type nonterminals =
| Syntax | Syntax_rule | Defns_list | Defn | Term | Repeated_sequence
| Meta_id | Letter | Digit

No mention of blindalleys -3

let iso_ebnf = (Syntax,
(fun x → match x with
~~Nope~~ | Syntax → [[N Syntax_rule]; [N Syntax_rule; N Syntax]]
| Syntax_rule → [[N Meta_id; T "="; N Defns_list; T ";"]]
| Defns_list → [[N Defn;]; [N Defn; T "|"; N Defns_list]]
must separate rules -2 → | Defn → [[N Term]; [T ","; N Factor]]
| Term → [[N Repeated_sequence]; [N Meta_id]; [T ""]]
| Repeated_sequence → [[T "{"; N Defns_list; T "}"]]
| Meta_id → [[N Letter]; [N Meta_id; N Letter]; [N Meta_id; N Digit]]
| Letter → [[T "a"]; [T "b"]; ... [T "z"]]
| Digit → [[T "0"]; [T "1"]; ... [T "9"]]

Richard Sun
904444918

CS 131 Midterm

4b. This grammar is recursive, so HW2 solution may try to search in a loop for a acceptable parse. For example, Meta id → Meta id; Letter will not work with a left most derivation because it will continuously check the Meta id rule without moving on. A rightmost derivation will have trouble with Syntax → Syntax rule; Syntax ✓

5a. 'a = X ✗

5b. (string * (string list)) list = ["3+4" ["(", "3+4", ")"]; "a", ["$", "3+4"]]

5c. Some ('a → ('a, 'b list) option = <fun>, String) = Some (accept_empty_suffix, "aaaaah!") -2 -1

6a. C0: not safe. It is possible for one thread to store a non-null value and another thread (which already did the val == null check) overwrite it.

C1: not safe. The set() function is synchronized, so no thread will overwrite val. However, a cached value of val can be returned by get()

C2: not safe. get() and set() are both synchronized, so there are no race conditions. However, val can be cached.

C3: Safe: set() is synchronized, and val is volatile so it is not cached.

C4: not safe. set() is synchronized, but get() can return a cached value

11

6b. C0 → C4 → C3 → C1 → C2 (faster)

functions with synchronized are the slowest since only one thread can run them at once. volatile variables also slows it down a little because the accesses cannot be optimized.

10

Richar
9044

CS 131 midterm

7a. It is not ambiguous. If a {} block is followed by catch, then it is a try block. But if it is preceded by a catch, then it is a catch block. otherwise, it is a normal block. If the block is part of a if, for, while, etc. and followed by catch, then it's a syntax error.

10