

**Note re Honesty:** I wanted to make explicit that it is not okay and will be considered cheating to explicitly search for solutions to a problem or to read handed-out solutions to problems from previous years of 601. Please ask if you have any questions about this policy.

**Typo on HW1 1b Answer:** While the analysis was correct we forgot to list the last group of equivalence classes. Here is the corrected first paragraph of the solution.

**1b.** We claim that the equivalence classes of  $\sim_{A_1}$  are exactly  $[c], [a^i], [a^i b], [a^{i+1}bc]$ , for all  $i \in \mathbf{N}$ . We first argue that all these classes are different. The first equivalence class,  $[c]$ , contains all words no extension of which is in  $A_1$ . The strings in the classes  $[a^i]$  can be appended with all strings of the form  $a^j b c^{2(i+j)}$  to get a word from  $A_1$ , the strings from  $[a^i b]$  can only be appended with  $c^{2i}$  to form a valid word from  $A_1$ . Finally, the strings from  $[a^{i+1}bc]$  can only be appended with  $c^{2i+1}$  to form a valid word from  $A_1$ . Thus all these classes are different.

**Problems:**

1. [25 pts.] Do problem 2.8.4, page 52 of [P] concerning Kolmogorov complexity. Hint for (b): remember that this definition is up to some fixed additive constant,  $c$ , so you are really being asked to show that for a fixed constant,  $c$ ,  $K(x) \leq |x| + c$ . Hint for (c): count how many strings there are of length  $n$ .
  
2. Define a  $k$ -head non-writing TM,  $M = (Q, \Sigma, \delta, s, F)$ ,  $\delta : Q \times \Sigma^k \rightarrow Q \times (\{\leftarrow, -, \rightarrow\})^k$ , to be a TM that has  $k$  read-only heads, all on the input string:  $\triangleright w_1 w_2 \dots w_n \sqcup$ . The heads are all started at the leftmarker and not allowed to move off the  $n + 2$ -character string.  $M$  accepts by entering an accept state. You may assume that once it enters an accept state it stays there forever.
  - (a) [5 pts.] To get a feel for this, construct a 2-head, non-writing TM that accepts exactly the palindromes over  $\{0, 1\}^*$ . Please describe the machine clearly, in English.
  - (b) [20 pts.] Show that every language accepted by a  $k$ -head non-writing TM is in L. (The converse holds as well, i.e. L is equal to the set of languages accepted by some multi-head non-writing TM, but you do not need to prove this.)
  
3. [25 pts.] Prove that a non-empty set  $S \subseteq \mathbf{N}$  is recursive iff there is a total recursive function,  $f$ , such that  $f$  is non-decreasing, i.e.,  $\forall n (f(n) \leq f(n + 1))$ , and  $S$  is the range of  $f$ , i.e.,  $S = f(\mathbf{N}) = \{f(n) \mid n \in \mathbf{N}\}$ .
  
4. Define Bloop – bounded looping – as the following tiny subset of the programming language Ruby [There is no need to know Ruby, the part we are using is completely self explanatory. The only advantage is that you can actually run all the definitions in Ruby.]
 

Bloop1. there is only one type: the natural number, i.e., non-negative integer.

Bloop2. there are no built in functions except the successor function  $s(n) = n+1$  and the constant 0. (Since Ruby doesn't actually have the built in function,  $s$ , we place the following definition at the top of our bloop programs, but this is the only use of "+" or "1".)

```
def s(n)
return(n + 1)
end
```

Bloop3. no functions may be called recursively, i.e., you can only call functions that have been previously defined in an acyclic way.

Bloop4. the only iteration method is ".times":

```
v.times do body end
```

meaning do body  $v$  times, where we mean the value of  $v$  before beginning this loop. Loops may not be nested.

Bloop5. there are no global variables, only the variables locally defined inside the definition of a function.

Bloop6. there are no nested function definitions.

Bloop7. each line inside a bloop definition is either an assignment:  $v = f(u_1, \dots, u_k)$ , where  $u_1 \dots u_k$  are variables and  $f$  is a previously defined bloop function, or the start of an iteration: " $v$ .times do" where  $v$  is a variable, or an end or return statement.

Bloop8. Define the bloop functions to be exactly the set of functions definable in a bloop program.

Bloop9. Each function definition must end with a statement of the form return(v) where  $v$  is a variable, and return statements may occur nowhere else. No uninitiated variable may appear in any statement except of course as the left side of an assignment.

For examples, see the Ruby programs: [www.cs.umass.edu/~immerman/cs601/bloop1.rb](http://www.cs.umass.edu/~immerman/cs601/bloop1.rb) and [www.cs.umass.edu/~immerman/cs601/bloop2.rb](http://www.cs.umass.edu/~immerman/cs601/bloop2.rb)

Use bloop1.rb to compute hyperexp( $n$ ) for  $n = 0, \dots, 4$ , but don't go beyond this! You can use bloop2.rb to compute hyperexp( $n$ ) for  $n = 0, \dots, 5$ , but don't go beyond this.

- (a) [5 pts.] Write a bloop program that defines the functions  $m1(n) = n - 1$  if  $n > 0$  else 0 and the function  $sub(a, b) = max(a - b, 0)$ . [Hint: it is a bit tricky to get  $m1$ . I suggest that you use a loop that starts with a value of 0 and  $n$  times, increments the value. To get  $n - 1$  you can copy the value to the ans variable **before** you increment it.]
- (b) [20 pts.] Show by induction on the length of the bloop program that every bloop function is total recursive, i.e., it is computable by a Turing machine that halts on all inputs. [Hint: I want you to think about how you could translate a general bloop program into a TM, but your solution does not need to have any explicit TM's. Rather I want you to figure out the tasks that a TM simulating a bloop program would have to do, convince yourself that you can do each of these tasks on a TM, and sketch simply but clearly why TMs can perform each of these tasks. The purpose of the problem is to get a handle on the power of TM's that are guaranteed to halt on all inputs. We will later see that the bloop functions compute exactly the Primitive Recursive Functions. This is a large subclass of the total recursive functions.]