CIS 511 Homework 3

Stephen Phillips, Dagaen Golomb

February 23, 2015

Problem 1

Show a Turing Machine that accepts the language $L = \{x \in a, b, c^* \mid x \text{ contains more } a\text{'s than } b\text{'s and } c\text{'s combined}\}$

We define the Turing Machine $M = \langle Q, q_0, \Sigma, \Gamma, \delta, q_A, q_R \rangle$ as follows. First we define the alphabets:

$$\Sigma = \{a, b, c\}$$
$$\Gamma = \{a, b, c, \dot{a}, \dot{b}, \dot{c}, \sqcup\}$$

The dotted symbols' indicate marked symbols which we have already read and accounted for. We will define the transition function, with the set Q being implicitly defined by the states that appear in the transition function. First step for this to work is to shift everything left so we know where the end of the tape is, like we did in class. After this, we have a blank symbol ($\sqcup$) at each end of the tape.

The following table describes what to do in each state given what is read from the tape. We provide a short description of the state's purpose to ease in understanding how the machine works. The starting state is q_0 and the final states are q_A and q_R which halt and accept or reject, respectively.

	a	b	c	$\dot{a}$	$\dot{b}$	$\dot{c}$	$\sqcup$	Description
q_0	q_0,a,R	q_1,$\dot{b}$,L	q_1,$\dot{c}$,L	q_0,$\dot{a}$,R	q_0,$\dot{b}$,R	q_0,$\dot{c}$,R	q_4,$\sqcup$,L	Scanning for a b or c, so far $a = b + c$.
q_1	q_1,a,L	q_1,b,L	q_1,c,L	q_1,$\dot{a}$,L	q_1,$\dot{b}$,L	q_1,$\dot{c}$,L	q_2,$\sqcup$,R	Marked a b or a c, return to left of tape.
q_2	q_3,$\dot{a}$,L	q_2,b,R	q_2,c,R	q_2,$\dot{a}$,R	q_2,$\dot{b}$,R	q_2,$\dot{c}$,R	q_R,$\sqcup$,R	Scan for an a to balance b and c's.
q_3	q_3,a,L	q_3,b,L	q_3,c,L	q_3,$\dot{a}$,L	q_3,$\dot{b}$,L	q_3,$\dot{c}$,L	q_0,$\sqcup$,R	Marked an a, return to left of tape.
q_4	q_A,a,L	q_4,b,L	q_4,c,L	q_4,$\dot{a}$,L	q_4,$\dot{b}$,L	q_4,$\dot{c}$,L	q_R,$\sqcup$,R	No more b or c's, scan left for a's.
q_R								Rejecting state
q_A								Accepting state

Problem 2

Show that a language is recognizable by a queuing automaton if and only if it is recognizable by a Turing Machine.

- ($\Longleftarrow$) If we have a Turing recognizable language, by definition there must be a Turing Machine that recognizes it. Therefore if we can make a queuing automaton replicate the actions of the Turing Machine we have shown that Turing recognizable languages are recognized by queuing automata. We will show that a queuing automaton can simulate a TM by explaining how it can simulate a step in a TM.

  To do this, we simplify the start and say that the queuing automaton pushes a 'start of tape' symbol followed by the entire string x, followed by a blank symbol, into its queue before computation begins. Now we just need to show how to perform each action a TM could perform. It always reads a symbol and writes a symbol. The two main differences is whether it moves left or right after a particular step. We will fix our simulated TM head at front of the queue (where items are removed).

  When we perform an action that moves left, we pull the read symbol from the queue and push the written symbol to the queue. This effectively moves all symbols right one spot which simulates the head moving left one.

When we perform an action that moves right, we must do some more work. First, we mark the symbol currently at the end of the queue. We then pull the read symbol from the queue and push the written symbol to the queue. Now, we continually pull the symbol at the front of the queue and reinsert it to the rear. Once we reach our marked symbol we remove the mark. We have now shifted all symbols left one, simulating a right move by the head.

- ($\Longrightarrow$) This is the simpler direction. It is intuitive that something as powerful as a TM can recognize anything a queuing automaton can. We can prove this by showing how a TM can simulate and queuing automaton.

  Given a description of a queuing automaton, a TM can simulate it as so: first, copy the input to a tape and use another tape to hold the queue. For the queue tape we will use the left end as the rear of the queue (where items are added) and the right end as the front. We will use a special marker (or blank) to mark the left of the queue tape. Now, we simply step over the input, always moving right since that is all the queuing automaton can do. For each push operation we move all of the queue tape right one spot (except the left end marker) and insert the new item in the created gap right after the left end marker. For each pull operation, we simply remove the right-most item and replace it with a blank.

  Since we can simulate a queuing automaton with a TM, it is clear any language recognizable by a queuing automaton is also recognizable by a TM.

Problem 3

Show that Turing recognizable languages are closed under Kleene Star.

To do this we consider the Turing recognizable language L, and the machine that recognizes it M. On a given input x, there are a large but finite number of ways to split the string into substrings. For a given partition, we can test if all the strings are in L using the following subroutine:

function A($x_1, x_2, \ldots, x_n$)
  Create list to store which runs on strings x_i have halted
  for $i = 1, \ldots, \infty$ **do**
    for j in incomplete list **do**
      Simulate the next step of $M(x_j)$
      if $M(x_j)$ has halted **then**
        If M rejects x_j, reject
        If M accepts x_j, remove j from incomplete list
      end if
    end for
    if Incomplete list empty **then**
      Accept
    end if
  end for
end function

This halts and accepts if and only if all the $x_i \in L$. It is fairly apparent, as it rejects if even one of the x_i gets rejected by M and accepts only after M has halted and accepted all of the x_i. If neither of these happens, one of the x_i must have made M loop forever, in which case A also loops forever since it will keep waiting until M halts before it halts.

Using this subroutine, we can get the following to accept L^*:

function N(x)
  Let $n = |x|$
  for $i = 1, \ldots, \infty$ **do**
    for $j = 1, \ldots, n$ **do**
      for All divisions of x into i strings, $x_1, x_2, \ldots, x_j$ **do**
        Run one step of $A(x_1, x_2, \ldots, x_j)$
        If $A(x_1, x_2, \ldots, x_i)$ halts and accepts, accept

end for
end for
end for
Reject
end function

This machine N accepts if and only if $x \in L^*$.

- ($\Longleftarrow$) If $x \in L$ then by definition there exist strings $x_1, \ldots, x_n \in L$ such that $x = x_1 \ldots x_n$. As N goes through all possible partitions of x, this partition will be found and therefore $A(x_1, \ldots x_n)$ will accept, and therefore N will accept.
- ($\Longrightarrow$) If N accepts x, then A must have accepted for some $i \geq 1$ and with some partition $x = x_1 \ldots x_i$. But $A(x_1, \ldots, x_i)$ accepts if and only if all the x_j are in L, which means by definition x must be in L^*. If it wasn't then no such partition could have been formed and none of the $A(x_1, \ldots, x_i)$ calls would have accepted. So therefore N only accepts strings in L^*

As N accepts all of L^* and only accepts strings in L^* the language of N must be L^*. Since this was done for arbitrary Turing recognizable language L it is true for all Turing recognizable languages.

Problem 4

We show that a language L is decidable if and only if it is enumerable in the standard string order.

- ($\Longleftarrow$) If L has an enumerable in the standard string order, then there exists an Turing machine, the enumerator E, that does exactly that: output all the strings in the language in the standard string order. Using that enumerator we can create a decider D:

  function D(x)
  for All strings y output by enumerator E **do**
  If $y = x$, accept
  If y comes after x in the standard string order, reject
  end for
  end function

  If E truly outputs all the strings in the language in the standard string order, then if x is in the language, it will eventually output x. Then D will accept when it is output by E. If it is not in the language, then since E outputs all stings in the language in the standard string order, when we see a string y that comes after x in that order, we know we will never see x in the language, so we will reject. So therefore the language of D is L. And it is a decider, since x has some finite index in the standard ordering of strings, one of these two outcomes will be reached in finite time. Therefore we will halt in finite time on all strings.

- ($\Longrightarrow$) If L is decidable, there must be some decider D that decided the language. Therefore we can make the following enumerator E:

  function E(z)
  Ignore input z
  for All strings x in the standard string order **do**
  If $D(x)$ accepts, output x
  end for
  end function

  As D is a decider, for every string x it will halt in finite time. Since it only accepts strings in L, E can only output strings in L. Since we are enumerating stings in the standard order, decide once whether or not to output them, we therefore must output the strings in L in the standard sting order, and E functions as we wanted it to.

Problem 5

Show that the language $L = \{\langle G \rangle \mid G \text{ is a CFG over } \{0,1\} \text{ and } L \cap 1^* \neq \varnothing\}$ is Turing decidable.

We use a similar method as finding if a CFG generates the empty language. To assist us, we have two types of marks on the symbols, $\dot{V}$ and $\ddot{V}$. The first is to mark that we have seen it and it can generate a string in 1^*. The second is to mark we have seen it but it cannot generate a string in 1^*.

The idea is that we look through the production rules and find strings in 1^*. We start at production rules that have right hand sides with only terminals, specifically terminal strings with only 1 in it. For every production rule we find with only 1 in it, we mark the corresponding variable V with the first kind of mark $\dot{V}$. If there are variables that can generate terminal strings but not ones in 1^*, then we mark them with the second kind of symbol $\ddot{V}$. We denote the first group S_1 and the second group S_2.

Now we look at production rules with right hand sides with only symbols in $S_1 \cup S_2$. If one of these right hand sides has only symbols in S_1 we place the corresponding variable from the left hand side into S_1. The rest of the variables we put into S_2. We repeat this until we have all variables in $S_1 \cup S_2$. If the starting symbol $S \in S_1$ then we know that this grammar generates strings in 1^*, and we accept. Otherwise we reject.

Problem 6

Show that $L = \{\langle A \rangle \mid L(A) = \varnothing\}$ is co-Turing recognizable.

To show this is co-Turing recognizable, we must show that $\bar{L}$ is Turing recognizable. In other words $\bar{L} = \{\langle A \rangle \mid L(A) \neq \varnothing\}$. This is clearly recognizable, since we can just simulate A on all strings by the usual dove-tailing method. This is described in detail below, described by the machine M.

function M(x)
  for $i = 1, \ldots, \infty$ **do**
    for x in the first i strings in Σ^* **do**
      Run $A(x)$ for i steps
      If $A(x)$ halts and accepts, accept
    end for
  end for
end function

So this machine will run forever on machines A which have the empty language which is fine since this is a recognizer so it does not need to halt on strings not in the language. However if A has even one string in its language, it will accept, since A must halt and accept that string in some finite time and therefore M will as well.

As $\bar{L}$ is recognizable, L must be co-Turing recognizable.

Problem 7

Show that a language C is Turing recognizable if and only if there exists a decidable language D such that $C = \{x \mid \exists y : \langle x, y \rangle \in D\}$.

The idea is that y is the number of steps it takes for a recognizer of C takes to accept x. The complete proof is as follows:

- ($\Longleftarrow$) If D is a decidable language, then there must exist a Turing machine N_D that decided the language. We construct the recognizer of C as follows:

  function M_C(x)
    for All y in Σ^* in Lexicographic order **do**
      If $N_D(\langle x, y \rangle)$ accepts, accept
    end for
  end function

  This loops forever for x not in C, but as N_D is guaranteed to halt on every iteration of the loop, if $x \in C$ then we will find the y such that $\langle x, y \rangle \in D$ and halt and accept. This satisfies all that we need the decider M_C to do, so we show that if such a language D exists, then C is Turing recognizable.

- ($\Longrightarrow$) If C is recognizable, then we can easily construct such a language. Since C is recognizable then there exists a Turing Machine N_C that recognizes the language. Then we construct the corresponding M_D:

  function M_D(x,y)
  Let $n = |y|$
  Run $N_C(x)$ for n steps
  If $N_C(x)$ halted and accepted within n steps, accept
  Otherwise Reject
  end function

  Since this only runs N_C for $|y|$ steps, it will halt on every input. If $x \in C$ then that means N_C must halt and accept after some number of steps, say m. That means that $\forall y : |y| > m, M_D(x, y)$ accepts. Therefore, the condition that there exists a y is satisfied.

Problem 8

- (a) Let L be the language of Kolmogorov random strings, i.e. $L = \{x : \text{x is Kolmogorov random}\}$. Recall that a string is Kolmogorov random if its Kolmogorov complexity $K(s) \geq n - 2$, where $n = \text{length}(s)$.
  Now, suppose L is decidable, so there exists a program P that decides it. Recall that greater than $1/2$ of all strings on a particular length are Kolmogorov random. Let x be the first Kolmogorov random string of length $2^{|P|}$. The following program Q prints x:

  function Q
  Enumerate string of length $2^{|P|}$
  for Each string y in the enumeration **do**
  Run P(y)
  if P(y) = YES, halt and accept
  end for
  end function

  Now, the size of program Q is the size of program P, plus a constant amount of code to form the enumeration and looping. Thus, $|Q| = |P| + c << 2^{|P|}$.
  But x is Kolmogorov random, and we have just created a program with length $|Q| < length(x) - 2 = 2^{|P|} - 2$, so x is actually not Kolmogorov random. This is a contradiction. Therefore, P cannot exist. Thus, if no decider for L can exist, then L must be undecidable.

- (b) Given a decider for L, we decide L_{Kol} as follows: we take the string under question as input. We then generate programs of increasing length and use the decider for L to determine if they halt given no input. If they do, we run them. We then take their output and compare to the string in question. If it matches, we have found a shortest program that outputs the given string. Knowing its Kolmogorov complexity, its simple to assert whether it is Kolmogorov random or not. Note that we know that we will eventually find a program that halts and outputs the string in question, since there always exists the program that simply stores the string and copies it to output when executed.

- (c) It follows from (a) and (b) that L must be undecidable, since if it were we could use it to decide a problem that we know to be undecidable (L_{Kol}).