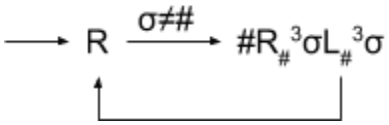
Luke Jaffe
Worked with: Emily Moxie
Theory of Computation
Due 03/23/16

Problem Set 6

Problem 1.
Using the notation for chaining Turing Machines shown in class, create a Turing Machine that takes two binary numbers separated by a space # and swaps them, so if ▸#1010#001 is the input, ▸#001#1010 is on the tape when the machine halts. You may assume 0,1, and # are the entire alphabet and that each number is non-empty. Feel free to define new subroutine machines in terms of the ones shown in class; you can also use the copying or shift-left machines described there. Please give a brief description of how your machine works. (The answer key copies the first number past the second, then "backspaces" to erase the first copy; you can use any strategy as long as the final result is correct.)

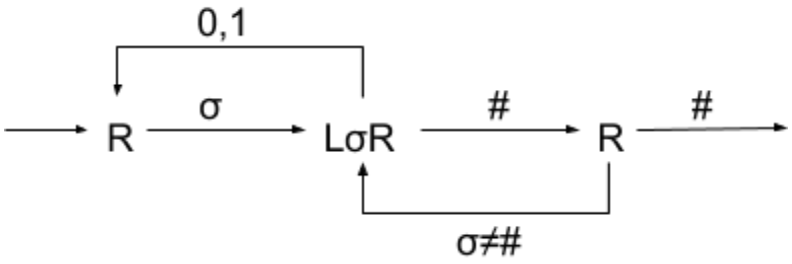
Let us define a modified version of the copying machine "C" as follows:



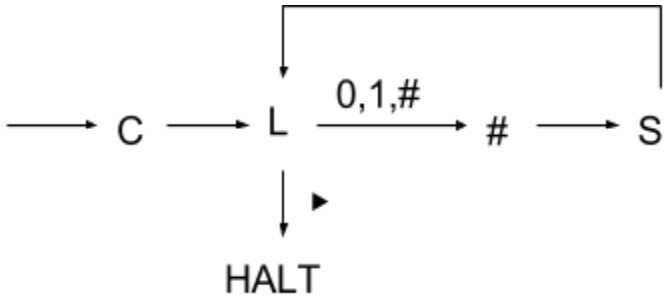
On a tape with two space-delimited strings, this machine will copy the first string past the second string (again delimited by a single space).

In the example given, the machine will start with the sequence:
▸#1010#001#
And after executing this copying machine, progress to the sequence:
▸#1010#001#1010#
With the current tape head symbol underlined.

Next, our machine must shift the second two space-delimited strings to the left, so as to overwrite the original first string. To do so, we will first construct the following machine, which we can call "S":



This machine will copy two space-delimited strings one square to the left. But it has one problem: it will conjoin the second string on the tape (first of the two copied strings) to the first string on the tape, such that we cannot delimit between the two. Therefore, before executing this machine, we must move left and write a space. While we are doing so, we can also check for the start character, as that is our halting condition. The final machine is constructed as follows:



First, we copy the first string past the second string. Then, we enter a loop to “backspace” the two copied strings so they are at the start of the tape (after a space). At the start of this loop, we check to see if the symbol to the left is the start symbol. If so, the copying has completed, and the machine will halt. This works because our copying machine also copies the space before the two copied strings. Then, we overwrite the symbol to a space, so we can always delimit between the start of the copied strings, and the string which is being overwritten. Finally, we shift both copied strings, including the space before them and the space between them, one square left. This loop repeats until the halting condition is reached, at which point the two strings will be “reversed” from their starting position.

Problem 2.

A Diophantine equation is a polynomial equation of possibly more than one unknown, such as $ab^2 + c^2 = 2$, where the solution must consist of all integers. (a=1,b=1,c=1 is one solution to the example.) Show that {E: E is a Diophantine equation with at least one solution} is recursively enumerable by describing an algorithm that will halt on E (eventually!) iff E is in the language. You may assume the Church-Turing Thesis — that is, you may assume that operations that are clearly computable are legal without describing low-level details. (You do not need any advanced mathematics for this problem.)

Let us define a Turing Machine M, which takes as input a Diophantine equation and a set of values for the variables in that equation. M accepts if the set of values satisfies the equation and rejects if it does not. The simplest way to find an input which halts is to enumerate all possible values for the input variables, and test them with the equation as inputs to machine M. Then, let us define a Turing Machine T which performs this enumeration, iterating through all possible values for the inputs for a given equation, and halting if machine M accepts on the given

equation and inputs. The machine T will continue on to the next possible input if M rejects, until it finds an input (if one exists) which causes M to accept. The trick is to define the manner in which we enumerate these inputs.

Let us use the given example equation ($ab^2 + c^2 = 2$) to show how we should enumerate these inputs. Suppose we enumerate as follows:

a=0, b=0, c=0 $ab^2 + c^2 = 0$
a=0, b=0, c=1 $ab^2 + c^2 = 1$
a=0, b=0, c=2 $ab^2 + c^2 = 4$
…

Since the integers are countably infinite, this enumeration will never reach a solution, even though we know one exists (a=1, b=1, c=1). Thus, we must enumerate the possible inputs in a manner which will "eventually" cover all possible values for all possible inputs. Let us define this method as follows: First, iterate through all combinations of inputs which contain only 0:
(a=0, b=0, b=0)
Then, iterate through all combinations of inputs which contain 0 and 1, except for those already listed:
(a=0, b=0, b=1)
(a=0, b=1, b=0)
(a=0, b=1, b=1)
(a=1, b=0, b=0)
(a=1, b=0, b=1)
(a=1, b=1, b=0)
(a=1, b=1, b=1)
Since we must also cover negative integers, we then iterate through all combinations of inputs which contain 0, 1, -1, except for those already covered. Then 0, 1, -1, 2, then 0, 1, -1, 2, -2, and so on. We can see that this method will eventually enumerate all possible values for all variables in the equation, without getting stuck incrementing/decrementing one variable forever. We have already demonstrated that our Turing Machine T will halt on a working equation and input. Since we can enumerate all possible values for the variables in the equation, and since the machine will halt on any correct inputs and otherwise keep trying inputs, we have proven that the given language is recursively enumerable.

Problem 3.
Show that the recursively enumerable languages are closed under Kleene star, so if L is a recursively enumerable language, so is L* = {w*: w is in L}. (Hint: A "nonstandard" Turing machine is useful here, though not required.)

Let the Turing Machine M be the machine which semidecides the language L. We know this machine exists, because it is given that L is recursively enumerable. Then, let us define a new Turing Machine P which takes as input <M,s>, where s is a string in L*. If the input s is empty, P

accepts. Otherwise, P takes the string s and runs it on machine M. If M accepts at some point, then P marks the place in the string s where M has accepted, and feeds a new string to M called s', where s' is s starting at the symbol after the marked place. P continues to run s' on M until s' is empty, at which point P accepts. If P ever feeds M an input s or s' which is not in L, then M will not halt, and thus P will not halt. In this way, P will accept on all strings s in L*, and will run forever on all strings not in L*.

Problem 4.

Prove $E_{\subseteq}$ = {<R1,R2>: R_1 and R_2 are regular expressions and $L(R_1) \subseteq L(R_2)$} is decidable by reducing it to the decidable language E_{DFA} = {<A>| A is a DFA and L(A) = ∅}.

First, let us note that DFAs are closed under the difference operation, i.e. there is a construction defined as DFA_1 - DFA_2 = DFA_3, where $L(DFA_1) - L(DFA_2) = L(DFA_3)$.

If $L(R_1) \subseteq L(R_2)$, $L(R_1) - L(R_2)$ must be the empty set i.e. there must be no elements of $L(R_1)$ which are not in $L(R_2)$. Further, we have an if-and-only-if (iff) relation, since if $L(R_1) - L(R_2)$ is the empty set, then $L(R_1) \subseteq L(R_2)$.

Let E be the Turing Machine which decides E_{DFA} on input A, accepting iff A is a DFA and L(A) = ∅.

Create a Turing Machine S that does the following:
1. Run the regex-to-NFA algorithm on R_1 and R_2 to produce N_1 and N_2 respectively.
2. Run the NFA-to-DFA algorithm on N_1 and N_2 to produce D_1 and D_2 respectively.
3. Run the difference construction to get D_1 - D_2 = D_3
4. Run the machine E on D_3, accepting iff it accepts.

This machine E will therefore accept on <R1,R2> iff $L(R_1) \subseteq L(R_2)$. Since we can reduce $E_{\subseteq}$ to E_{DFA}, $E_{\subseteq}$ is decidable.

Problem 5.

a.) Prove the "time-limited halting problem" {: M is a deterministic Turing Machine that halts on input w in t or fewer steps} is decidable.

Let us define a Turing Machine P which takes <M,t,w> as input. P executes machine M on input w with a few special rules. First, P has a counter variable C which starts at 0. Each time M takes a step, P increments C by one. If C reaches the value t+1, then P rejects. If M finishes executing before C reaches t+1, then P accepts or rejects according to which M does.This will ensure that the machine M can only accept in t or fewer steps, but will always have halted after t+1 steps. Since the machine P halts on any input w, the language is decidable.

b*.) Prove the “space-limited halting problem” {: M is a deterministic Turing Machine that halts on input w without the head moving past the kth tape square} is decidable. (Hint: What must be true if a configuration of M is repeated?)

Let us define a Turing Machine P which takes <M,k,w> as input. P executes machine M on input w with a few special rules. First, P has a counter variable C which starts at 0. Each time M steps right, P increments C by one. Each time M steps left, P decrements C by one. If C reaches the value k+1, then P rejects. If M finishes executing before C reaches k+1, then P accepts or rejects according to which M does.

Our machine P also has one more rule: If a configuration of M is repeated, P rejects. This is because a repeated configuration means M is inside an infinite loop, and will never halt on its own. A repeated configuration implies an infinite loop because if a configuration produces itself after some number of steps, then it will do so again infinitely. Let us go further by proving that if a Turing Machine is constrained not to repeat configurations, then it must halt or reach square k+1. This is because the set of all configurations which can exists within k squares on a Turing Machine is finite. In order to reach a configuration outside of this set, the Turing Machine must utilise a square past k.

We assume that configuration history can be recorded and checked on a Turing Machine. Since M cannot enter an infinite loop, and will always halt past the kth tape square, it must eventually halt. Therefore, the language is decidable.