CS 162 **Fall 2015**

Homework 7 Solutions

November 19, 2015 *Timothy Johnson*

1. Exercise 9.2.4, pg. 391.
   Let $L_1, L_2, \ldots, L_k$ be a collection of languages over alphabet Σ such that:

   - For all $i \neq j$, $L_i \cap L_j = \emptyset$; i.e., no string is in two of the languages.
   - $L_1 \cup L_2 \cup \ldots \cup L_k = \Sigma^*$; i.e., every string is in one of the languages.
   - Each of the languages L_i, for $i = 1, 2, \ldots, k$ is recursively enumerable.

   Prove that each of the languages is therefore recursive.

   To show that each of these languages is recursive, we need to be able to determine for each input which of our k languages contains it. Then we know (based on the first property) that every other language does not contain it.

   Since each language is RE, for each language we can construct a Turing machine with a single tape that accepts all inputs in that language.

   To determine which language contains each input, we construct a Turing machine with k tapes. First, we copy the input to every tape. Then we run each of the k machines for our languages in parallel on different tapes. Some of these may reject the input, and some may run forever. But since our input is in some language, one of the machines will eventually accept.

   If machine i accepts, then we know that our input is in the language L_i, and not in any of the other languages. QED.

   Bonus version: To have this problem make sense for an infinite number of languages, we need to have a function that describes all of them.

   Let f be a computable function on the natural numbers. Then let $\mathcal{L}$ be the collection of languages accepted by the infinite set of Turing machines $\{M_{f(i)} \mid i \geq 0\}$. Note that since each language is the language of a Turing machine, it is implied that each one if recursively enumerable. Assume that:

   - For all $i \neq j$, $L(M_{f(i)}) \cap L(M_{f(j)}) = \emptyset$; i.e., no string is in two different languages.
   - $\cup_i L(M_{f(i)}) = \Sigma^*$; i.e., every string is in one of the languages.

   Prove that each of the languages is recursive.

   Given an input w, we need to find the value of i for which $w \in L(M_{f(i)})$. Our previous proof doesn't work, because we can't run an infinite number of machines in parallel simultaneously.

However, we can run any finite number of these machines in parallel. So we'll proceed in rounds. In round k we'll run each of the first k machines in our list for k steps.

Then suppose that w is accepted by machine k_1 after k_2 steps. In round $\max(k_1, k_2)$, we'll find the language that accepts w. Given that it's accepted by machine k_1, we know that every other machine will either reject it or run forever.

Therefore, each of the given languages is recursive. QED.

2. Exercise 9.2.6(b,c,d), pg. 392.
Determine whether the recursive and/or the recursively enumerable languages are closed under the following operations. You may give informal, but clear, constructions to show closure.

   - Intersection.
   - Concatenation.
   - Kleene closure (star).

   Recursive languages are closed under all three of these operations.

   Suppose that L_1 and L_2 are recursive languages. Then there is a Turing machine M_1 that accepts any input in L_1 and rejects any input not in L_1. Similarly, there is a Turing machine M_2 that accepts any input in L_2 and rejects any input not in L_2. We need to show that for each of our new languages, we will accept if w is in the language, and reject if w is not in the language.

   - To determine if some input is in the intersection $L_1 \cap L_2$, we run both M_1 and M_2 on the input. We accept if both of these accept, and reject if either one of them rejects.
   - To determine if some input is in the concatenation L_1L_2, we first nondeterministically guess where to divide our input w into two parts, w_1w_2. Then we run M_1 on w_1 and M_2 on w_2. We accept if both of these accept, and reject if either of these reject.

     If $w \in L_1L_2$, then there is some place at which we can split our string such that M_1 accepts w_1, and M_2 accepts w_2. So our nondeterministic machine will accept this input.

     Otherwise, if $w \notin L_1L_2$, then there is no place at which we can divide our string such that M_1 accepts w_1 and M_2 accepts w_2. So our nondeterministic machine will reject.
   - Lastly, suppose that we want to determine if $w \in L_1^*$. If w is empty, we accept. Otherwise, we first nondeterministically guess how many different parts into which we need to break our input. Then we guess where to put all of the breaks in our input, and run M_1 on each part. We accept if M_1 accepts every part.

     If $w \in L_1^*$, then there is some way in which we can break our strings so that M_1 will accept each part. Otherwise, we will have to reject.

   Recursively enumerable languages are also closed under intersection, concatenation, and Kleene star.

Suppose that M_1 and M_2 accept the recursively enumerable languages L_1 and L_2. We need to show that if w is in our new language, it will be accepted.

- To determine if $w \in L_1 \cap L_2$, we run both M_1 and M_2 on the input. If w is in the intersection, then both machines will eventually accept, so we will accept the input.
- To determine if $w \in L_1 L_2$, we nondeterministically guess where to break the string into $w_1 w_2$, and run M_1 on w_1 and M_2 on w_2. Then if w is in the language, M_1 and M_2 will both eventually accept.
- To determine if $w \in L_1^*$, we nondeterministically guess how many parts into which we need to break our input, and where to put the breaks. Then we run M_1 on each part. If w is in the language, then each of these will accept.

3. The Big Computer Corp. has decided to bolster its sagging market share by manufacturing a high-tech version of the Turing machine, called BWTM, that is equipped with *bells* and *whistles*. The BWTM is basically the same as your ordinary Turing machine, except that each state of the machine is labeled either a "bell-state" or a "whistle-state". Whenever the BWTM enters a new state, it either rings the bell or blows the whistle, depending on which state it has just entered. Prove that it is undecidable whether a given BWTM M, on given input w, ever blows the whistle.

   We reduce from the problem of deciding whether a Turing machine accepts a given input. For each Turing machine, we set each final state to be a whistle state, and every other state to be a bell state.

   Now suppose that we can decide if, for a given input w, our BWTM ever enters a whistle state. Then we can determine whether the original Turing machine accepts w. But we already know that this is undecidable. Therefore, by this reduction, determining if a BWTM ever enters a whistle state is undecidable. QED.

4. Exercise 9.3.3, pg. 400.
   Show that the language of codes for TM's M that, when started with a blank tape, eventually writes a 1 somewhere on the tape is undecidable.

   We will reduce from the problem of deciding whether a Turing machine accepts a given input. If 1 was in the tape alphabet of our original machine, then we replace it with a different character. That way, we have a Turing machine that never writes a 1 on its tape.

   Next we modify each transition that goes to an accepting state, so that it writes a 1 on its tape. Therefore, this Turing machine now writes a 1 on its tape if and only if the original Turing machine halts when started with a blank tape. Since the halting problem is undecidable, this new problem is also undecidable. QED.

5. Exercise 9.3.6(b), pg. 400

Let L be the set of codes for TM's that never make a move left on any input. Show that L is decidable.

We will perform a breadth-first search of the possible states of our Turing machine, until we have either discovered every reachable state, or found a reachable state on which it moves left.

To be more precise, we begin in the start state, and check all possible starting input characters (including a blank symbol for empty input). If any of these transitions moves left, then we're done. Otherwise, we take all of the possible states that we could have moved to, and look at all of the possible characters in the second position in our input (including a blank).

If at any step we have added no new states, then we stop, since we have found every reachable state. Let the number of states be $|Q|$. If there is no reachable state from which we could move left, we will stop after at most $|Q|$ rounds. Otherwise, if there is some reachable state from which we could move left, we will reach it in at most $|Q|$ rounds.

Therefore, since this algorithm will run in finite time for both codes in L and codes not in L, L is decidable. QED.