HW8 Solutions

Michał Dereziński

March 14, 2015

1 Exercise 7.37

As in the suggestion, we assume that $T$ has tape alphabet $\Gamma = \{a_1, a_2, \ldots, a_n\}$, where $a_1 = 1$ is the only letter permitted in the input and output. The natural numbers for the computed function (input and output) are assumed to be encoded with the length of a string $1^k$. Machine $T'$ will simulate $T$ by encoding the tape letters $a_i$ as $1^i\Delta^{n+1-i}$. The general procedure has three main steps:

1. Convert the input to our encoding. Notice that our changes have to be transparent with respect to the input and output, so the input will still be a natural number written as $1^k$. However, to simulate machine $T$, we have to convert each 1 into $1\Delta^n$ obtaining $(1\Delta^n)^k$.

2. Simulate machine $T$ until it halts. To simulate a single transition, we need to perform several operations.

   (a) We need to read the encoded letter from the tape. This step will take up to $n$ transitions to look through the current segment of the tape, counting the number of 1’s. The counting has to be done using the states, so we will also need $n$ states, each one for the different letter.

   (b) We need to be able to write a new encoded letter. This will also require $n$ states for the purpose of counting.

   (c) We also need to potentially move the head by $n+1$ positions, left or right. This does not require so many states, because we can just go through all the 1’s, and through all the $\Delta$’s until we hit another 1 (two states).
Notice, that along with the counting, the states have to also remember the state information of the simulated machine, so the number of states needed for the simulation phase is roughly \(n \cdot (q + t)\), where \(q\) and \(t\) are the number of states and transitions of machine \(T\), respectively.

3. Finally, after the simulation halts, we need to translate the output back to the original format, by removing the \(\Delta\)'s separating each letter.

2 Exercise 8.4

Let \(L_1, \ldots, L_k\) be recursively enumerable languages forming a partition of \(\Sigma^*\). I will show that \(L_i\) is recursive (for all \(1 \leq i \leq k\)).

Since \(L_1, \ldots, L_k\) is a partition of \(\Sigma^*\), we know that \(\bigcup_{j \neq i} L_j\) is equal to the complement \(L'_i\) of \(L_i\). According to Theorem 8.4, recursively enumerable languages are closed under finite union, so we can conclude that \(L'_i\) is recursively enumerable. Now Theorem 8.7 says that if a language and its complement are both recursively enumerable, then they are also recursive. Therefore, we can conclude that \(L_i\) is recursive, as desired.

2.1 Discussion

Intuitively, for any given input string \(x\), we can always determine exactly which one of \(L_1, \ldots, L_k\) contains \(x\). We can do this by interleaving the computations of the machines accepting \(L_1, \ldots, L_k\) and waiting until one of the machines halts and accepts. We are guaranteed that exactly one of the machines will do so because \(L_1, \ldots, L_k\) form a partition of \(\Sigma^*\).

3 Exercise 8.19 (a)

\[
\begin{align*}
S & \rightarrow T \mid \Lambda \\
T & \rightarrow A B T A B \mid X X \\
B A & \rightarrow A B \\
X A & \rightarrow a \quad a A \rightarrow a a \quad a B \rightarrow a b \quad b B \rightarrow b b \\
B X & \rightarrow b \quad B b \rightarrow b b \quad A b \rightarrow a b \quad A a \rightarrow a a
\end{align*}
\]
4 Exercise 8.38

For the sake of contradiction, suppose that $S - T$ is countable. Then, there exists a list $s_1, s_2, \ldots$ of all members of $S - T$. Since $T$ is countable, we also have a list of all members of $T$: $t_1, t_2, \ldots$. Suppose that we cross out from this sequence all of the elements that are not in $S$. We are left with a new sequence $t_{i_1}, t_{i_2}, \ldots$, that contains elements from $T \cap S$ (which is also countable). Now, consider the sequence:

$$s_1, t_{i_1}, s_2, t_{i_2}, s_3, t_{i_3}, \ldots$$

This sequence counts the elements of the set $(S - T) \cup (T \cap S) = S$, so we showed that $S$ is countable, contradiction.

5 Exercise 8.40 (a)

Suppose to the contrary that the set $S$ of all ternary sequences is countable. Clearly, it has to be infinite. Then there exists a list $s_1, s_2, \ldots$ of all members of $S$. Now consider the sequence $a = (a_1, a_2, \ldots)$ defined by

$$a_i = ((s_i)_i + 1) \mod 3.$$  

We can see that the sequence $a$ is not equal to $s_n$ for any $n$, because $a_n \neq (s_n)_n$. Therefore $a$ is not in the list $s$. But this contradicts the fact that every ternary sequence is in the list $s$. We conclude that $S$ is not countably infinite, and thus it must be uncountably infinite, as desired.

6 Exercise 8.41

6.1 Part a

The set of all three-element subsets of $N$ is countably infinite. Notice that it is equal to $\bigcup_{i=0}^{\infty} A_i$ where

$$A_i = \left\{ \{a, b, c\} \subseteq N \mid \text{max}\{a, b, c\} = i \right\}$$

Each $A_i$ is finite, so their union must be countable.
6.2 Part d

The set of all functions from \( \mathbb{N} \) to \( \{0, 1\} \) is uncountable. Suppose to the contrary that it is countable. Then there exists a list \( f_1, f_2, \ldots \) of all those functions. Now consider the function \( g(n) = 1 - f_n(n) \). We can see that \( g \) is not equal to \( f_n \) for any \( n \), because \( g(n) \neq f_n(n) \). Therefore \( g \) is not in the list \( f \). But this contradicts the fact that every function from \( \mathbb{N} \) to \( \{0, 1\} \) is in the list \( f \). We conclude that the set is not countable, and thus it must be uncountably infinite, as desired.

7 Exercise 8.8

Let \( T \) be a Turing machine accepting a non-recursive language \( L \). Assume for the sake of a contradiction that there are only finitely many strings (call them \( x_1, \ldots, x_n \)) for which \( T \) fails to terminate. I will prove that the complement \( L' \) of \( L \) is recursively enumerable, which (by Theorem 8.7 again) will contradict the fact that \( L \) is not recursive.

Notice that \( L' \) is equal to \( \tilde{L} \cup \{x_1, \ldots, x_n\} \), where \( \tilde{L} \) is the set of strings on which \( T \) crashes or explicitly rejects. Obviously, it suffices to construct a machine \( \tilde{T} \) accepting \( \tilde{L} \). We can easily derive \( \tilde{T} \) from \( T \) by changing “accept” to “reject” and by changing “reject” to “accept” and by being careful to detect crashes before they occur (at which point \( \tilde{T} \) should of course accept).

8 Exercise 8.13

Let \( T \) be a Turing machine enumerating the members of \( L \) in some order. Let us denote these strings by \( x_1, x_2, \ldots \) in the enumeration order.

We cannot assume that this sequence is in canonical order. However, we can define a new Turing machine \( \tilde{T} \) that enumerates a canonically ordered subsequence of \( x_1, x_2, \ldots \). Such a machine would simply start by printing \( x_1 \), and then it would print the next string in the sequence only if it is “greater” (according to the canonical order) than the most recently printed string.

It is obvious that the language \( \tilde{L} \) accepted by \( \tilde{T} \) is recursive (because the strings are printed in canonical order) and that \( \tilde{L} \subseteq L \). It remains to show that \( \tilde{L} \) is infinite.

Suppose to the contrary that \( \tilde{L} \) is finite. According to the definition of \( \tilde{T} \), this would mean that at some point, the strings being enumerated by
are all “less” than one of the previously printed strings (call it \( x_n \)). So apparently \( x_n \) is the “greatest” string in the list. But there are infinitely many distinct strings in the list, and only finitely many strings are “less” than \( x_n \) (thanks to the nice properties of the canonical order). We conclude that there must be a string in the list that is “greater” than \( x_n \), but this contradicts the “maximality” of \( x_n \).