COMPUTABILITY
HOMEWORK 3 (SLIGHTLY CORRECTED BY DPH)

KRISHNA ROSKIN AND ROBERT B. GRAMACY

1. Problem (10.3)

Theorem 1. Suppose that $L$ is recursively enumerable but not recursive, then any TM accepting $L$ must loop forever on infinitely many inputs.

Proof. Since $L$ is recursively enumerable there exists a TM $T$ accepting $L$. Let $S$ be the set of strings in $\Sigma^*$ on which $T$ loops forever. Now suppose that $S$ is finite. Since $S$ is finite there necessarily exists a TM recognizing $S$. Let $T_s$ be such a machine. Now, construct TM $T_\ell$ which does the following.

1. Simulate $T_s$ on input $x \in \Sigma^*$. If $T_s$ computes 1 on $x$, then let $T_\ell$ crash, thus not accepting $x$. Otherwise go to (2).
2. Simulate $T$ on input $x$. Since $x \notin S$ we can be assured that $T$ will not loop forever. $T$ will either halt (accepting $x$) or crash.

We have that every possible sequence of moves of $T_\ell$ results in either a halt or crash. Therefore, by Theorem 10.1, $L(T_\ell)$ is recursive. However, it is obvious that $L(T_\ell) = L$. But this contradicts that $L$ was not recursive. This means that our assumption that $S$ is finite must be false. Therefore, we may conclude that any TM accepting $L$ must loop forever on infinitely many inputs. ■

2. Problem (10.4)

Claim 1. It is not always true that if $L_1, L_2, \ldots$ are any recursively enumerable subsets of $\Sigma^*$, then $\bigcup_{i=1}^\infty L_i$ is recursively enumerable.

Proof. Consider the language $NSA$ of non-self-accepting strings over \{0, 1\}* as described in the text. Since $NSA$ is in particular a subset of $\Sigma^*$, it is countable. Let $A = a_1, a_2, \ldots$ be one such ordering of the elements of $NSA$. Consider the sets $N_i = \{a_i\}$, each containing just one unique element of the language $NSA$. Clearly, since each $N_i$ contains just one element, it is recursively enumerable. Since $NSA$ is countable we have that

\[
\bigcup_{i=1}^\infty N_i = NSA.
\]

By Theorem 10.9 $NSA$, is not recursively enumerable. Therefore, it is not always true that if $L_1, L_2, \ldots$ are any recursively enumerable subsets of $\Sigma^*$, that $\bigcup_{i=1}^\infty L_i$ is recursively enumerable. ■
3. Problem (10.6)

**Definition 1.** If \( x \) is a string and \( 0 \leq n \leq |x| \) then the \textit{prefix} of \( x \) of length \( n \), written \( \text{pre}(x,n) \), is a string comprised of the first \( n \) letters of \( x \). The \textit{postfix} of \( x \) of length \( n \), written \( \text{post}(x,n) \), is a string comprised of the last \( n \) letters of \( x \).

**Theorem 2.** The class of recursively enumerable languages is closed under concatenation.

**Proof.** Let \( T_1 \) and \( T_2 \) be the TMs that accept the two languages being concatenated. Construct TM \( T \) with five tapes: an input tape, two counter tapes, and two work tapes. The first counter tape is initialized to the equivalent of \( |x| \) and the second to 0. \( T \) will execute the following loop. We first clear both work tapes. Then we copy \( \text{pre}(x,c_1) \) to the first work tape and \( \text{post}(x,c_2) \) to the second work tape, where \( c_1 \) and \( c_2 \) are the value stored on the first and second counter tape respectively. Note that at any time \( \text{pre}(x,c_1) \) concatenated with \( \text{post}(x,c_2) \) is \( x \). We now do a non-deterministic branch. One of the branches decrements \( c_1 \) and increments \( c_2 \) and branches to the beginning of the loop. The second non-deterministic branch, executes \( T_1 \) on the first work tape and then \( T_2 \) on the second work tape. If they both halt, let \( T \) halt. Therefore we have that \( \text{pre}(x,c_1) \in L_1 \) and \( \text{post}(x,c_2) \in L_2 \). If \( T \) tries to decrement from 0 in the first counter tape, we have \( T \) crash. Thus \( T \) will accepts the concatenation of two strings from two recursively enumerable languages and not halt for any other strings. Thus the concatenation of the two recursively enumerable languages is again recursively enumerable. \( \square \)

**Theorem 3.** The class of recursively enumerable languages is closed under the Kleene star.

**Proof.** Let \( T_1 \) be the TM that accepts the language \( L \) in question. We will construct a three-tape TM \( T \) with an input tape, a counter tape, and a work tape. \( T \) is constructed to halt on the empty string. Initialize the counter tape to the equivalent to 1. Now we execute the following loop. Clear the work tape and copy \( \text{pre}(x,c) \) to it, where \( c \) is the value on the counter tape. Now we execute two non-deterministic branches. The first branch increments \( c \) and restarts the loop. The second branch executes \( T_1 \) on the work tape. If \( T_1 \) halts, delete \( \text{pre}(x,c) \) from the input string and execute \( T \) on the remaining input. Note that we have \( T \) crash if \( c > |x| \). Thus \( T \) will halt if and only if there is a sequence of consecutive substrings that \( T_1 \) accepts, thus accepting \( L^* \). Otherwise, it will loop forever or crash on anything not in \( L^* \). \( \square \)

4. Problem (10.8)

First we give two lemmas.

**Lemma 1.** Suppose that \( f : \mathbb{N} \rightarrow \Sigma^* \) is a bijection. If \( f \) is computable, then \( \Sigma^* \) can be enumerated with respect to \( f \).

**Proof.** Let \( T_f \) be the TM that computes \( f \) given a natural number in unary. Construct TM \( T \) with three tapes: an output tape, a counter tape, and a work tape. We begin by copying the counter tape to the work tape and then running \( T_f \) on the work tape which has contents \( \Delta \text{f}^n \) for some \( n \in \mathbb{N} \). This leaves \( \Delta \text{f}(n) \) on the
work tape. Copy \( f(n) \) to the output tape and follow it by a \#. Now increment the counter tape. As this procedure is repeated indefinitely the output tape will contain \( f(0)\#f(1)\#f(2)\# \ldots \). Since \( f \) is surjective, every \( x \in \Sigma^* \) will appear on the output tape. Also, since \( f \) is injective, every \( x \) will appear only once. Thus \( T \) enumerates \( \Sigma^* \).

\[ \square \]

**Lemma 2.** Suppose that \( f : \mathbb{N} \rightarrow \Sigma^* \) is a bijection. If \( f \) is computable, then \( f^{-1} : \Sigma^* \rightarrow \mathbb{N} \) is computable.

**Proof.** By using Lemma 1, let \( T \) be the TM that enumerates \( \Sigma^* \) with respect to \( f \). Now construct TM \( T_{f^{-1}} \) with two tapes: an input tape and a work tape. Execute the following loop. Run \( T \) on the work tape, pausing before it writes a \#. Now compare the just enumerated string to the string on the input tape. If they don’t match, continue enumerating \( \Sigma^* \). If they match, clear the input tape and then reset the head to the beginning of the work tape. Traverse the work tape and for every \#, place a 1 on the input tape and reset the head to beginning of the tape. Thus, the input tape will contain \( \Delta 1^n \) where \( f(n) = x \), the input string. Since \( f \) is subjective, \( T \) will eventually enumerate \( x \), thus will \( T_{f^{-1}} \) halt. Since \( f \) is injective, we know that \( n = f^{-1}(x) \). Thus we have computed \( f^{-1} \).

\[ \square \]

Now we state our result.

**Theorem 4.** Suppose that \( f : \mathbb{N} \rightarrow \Sigma^* \) is a computable, bijective function. Then for any \( L \subseteq \Sigma^* \), \( L \) is recursive if and only if it can be enumerated in order \( f \).

**Proof.** Suppose \( L \) is recursive. Let \( T_L \) be a TM that decides \( L \). By Lemma 1, there exist a TM that enumerates \( \Sigma^* \) with respect to \( f \), call this TM \( T \). Construct \( T' \), a three-tape TM, with an output tape, a work tape and an enumeration tape. \( T' \) will enumerate \( \Sigma^* \) by running \( T \) on the enumeration tape. After a string \( x \in \Sigma^* \) is written, we will copy \( x \) to the work tape. Then we run \( T_L \) on the work tape to see if \( x \in L \). If \( x \in L \), copy \( x\# \) to the output tape. Note that we are guaranteed that \( T_L \) will halt. Otherwise keep enumerating, checking for inclusion in \( L \), and writing to the output tape. Thus only the strings in \( L \) are written to the output tape; furthermore, they are written with respect to \( f \). Thus \( T' \) enumerates \( L \) with respect to \( f \).

Suppose that \( L \) can be enumerated with respect to \( f \). Let \( T_f \) be a TM that performs this enumeration. Construct \( T \) with three tapes: an input tape, a work tape, and an enumeration tape. By Lemma 2 there exist a TM that computes \( f^{-1} \), call this \( T_{f^{-1}} \). Run \( T_{f^{-1}} \) on the input \( x \) to get \( f^{-1}(x) \) on the input tape. Since \( f \) is a bijection, we need only work with this number. Start enumerating \( L \) onto the enumeration tape. As each \( y \in L \) is written, copy it to the work tape and calculate \( f^{-1}(y) \) using \( T_{f^{-1}} \). If \( f^{-1}(y) = f^{-1}(x) \), we halt with \( \Delta 1 \) on the output tape. If \( f^{-1}(y) > f^{-1}(x) \), we halt with \( \Delta 0 \) on the output tape. Otherwise, we continue writing \( y \)'s and testing. Since \( f^{-1}(y) \) is strictly increasing as \( y \in L \) is enumerated by \( T_L \), eventually \( f^{-1}(y) \geq f^{-1}(x) \) (NOTE: this may not be true if \( L \) is finite). Thus \( T \) will always halt, and therefore \( T \) decides \( L \). We conclude that \( L \) is recursive.

\[ \square \]

**NOTE:** the previous proof has a small bug – towards the end it doesn’t handle the case when \( L \) is finite. However, that can easily be fixed by by a case analysis. If \( L \) is infinite, use the above proof. If \( L \) is finite, then \( L \) is automatically recursive (as it can be decided by a finite state machine).
Corollary 1. If $L \subseteq \Sigma^*$ is recursive and is enumerable with respect to $f$, then $f$ is computable.

Therefore, the computability of $f$ is a necessary and sufficient condition.