Solution Set 7

1.

2. (Graded by Brian Terry) Every infinite recursive set is in the range of a strictly increasing recursive function.

   Let $S \subseteq N$ be an infinite recursive subset of the natural numbers. Define the recursive function f as follows: $f(0) =$ the least element of S. Since S is infinite there has to be some element in S, since S is recursive we can check whether $0 \in S$, $1 \in S$, et.c. until the answer is affirmative. Thus $f(0)$ is well defined. In general $f(x)$ is the least element of $S - \{f(0), \ldots, f(x-1)\}$.

   By construction f is strictly increasing, and the range of f is S.

3. Graded by David Schollmeyer (solutions in hardcopy only.)

4. Graded by Jim Morris and Patrick Donahue. Solution due to Jim Morris.

   (a) This is proved by diagonalization. Take one axis of a diagonalization table to be any infinite sequence of different strings and the other axis to be all languages assumed enumerable. Let $L^* = \{s_k \ : \ s_k \notin L_k\}$, where s_k is the k'th string and L_k is the k'th language. This condition creates a new language that is different to every other language by virtue of changing the one element on the diagonal for each language. Since this new language is not in the table, then we add it to the table but it would have to be different than itself and this leads to a contradiction that proves that the set of all languages is uncountable.

   (b) Σ is the alphabet, $k = |\Sigma|$. Then k-adic(x) is a one-one correspondence from Σ^* to N.

   (c) It is impossible to represent an uncountable set (the set of all languages) by a countable set (the set of all strings). If a mapping was available between the strings and the languages, that would imply that the languages could be mapped to N which we proved in part (a) to be wrong; thus, the mapping is not possible.

   (d) All recursive languages can be represented by a DTR. All DTRs can be encoded as a string. And since all strings are countable, the recursive languages are countable.

   Let L be set of recursive languages, that by the above can be enumerates as $\{L_1, L_2, L_3, ...\}$. Let B be $2^{\Sigma^*} - L$. Assume B is countable with enumeration $\{K_1, K_2, K_3, \ldots\}$, then $\{L1, K1, L2, K2, ...\}$ is an enumeration of 2^{Σ^*}. This is a contradiction so the set of all non-recursive languages must be uncountable.

(e) All r.e. languages can be represented by a NTA. NTAs can be encoded as different strings. And since all strings are countable, the r.e. languages are countable.

We establish that the complement of r.e. languages are uncountable by an argument similar to part (d).

(f) Let L be the set of all r.e. languages. Let $\overline{L}$ be the set of all co-r.e. languages (Note that co-r.e. $\neq$ non-r.e.) Let $B = 2^{\Sigma^*} - L - \overline{L}$. The arguments from part (d) and (e) show that B is uncountable, whereas $L \cup \overline{L}$ is countable. Thus, B contains a language not in $L \cup \overline{L}$. In other words, somewhere in that uncountable set is a non-re and non-co-re language.

5. We present a solution adapted from one by William Shulman.

(a) We will show that $\overline{K}_{xy} \leq_m K_{PQ}$. Let total recursive $f(x, y) = (A, B)$, where A is the program that runs (a hardcoded) input y on the machine x, and B is a program that loops forever. Then

$$(x, y) \in \overline{K}_{xy} \Leftrightarrow f(x, y) \in K_{PQ}$$

By Proposition 7.29 (iv), if K_{PQ} was r.e., $\overline{K}_{xy}$ would be r.e. too, which would contradict Theorem 7.36.

(b) If there does exist a program D that can produce a proof of equivalence, then we have a r.e. machine for K_{PQ}: On input (P, Q), supply this input to D. If D halts with a correct proof of equivalence (as checked by the verifier) - note that this is guaranteed to happen if P and Q are equivalent - accept, else reject.
Since we know that the language is not r.e., such a D cannot exist.

Remarks: Note that (a) implies that TMs P and Q can be equivalent (accept the same language) and yet there may be no *proof* of this fact. For, if there was always a proof of equivalence whenever two TMs were equivalent, we could build a r.e. proof producer by enumerating all proofs, and using the verifier to check if a enumerated 'proof' is a correct proof equivalence. Thus, another (equivalent) reason there is no machine D as above is that proofs D is alleged to produce may not even exist.

Even more interestingly, the many of you who did parts (a) and (b) right will be happy to note that you have in fact proved a version of Gödel's Incompleteness Theorem (FB. pp 569-574). Alan Turing, in his 1937 breakthrough paper proving the undecidability of the halting problem (Theorem 7.21), did observe that Gödel Incompleteness emerged as a ("trivial", in his words) consequence of undecidability. Nevertheless, this is a somewhat deep connection, and you should be happy you have some feel for it now.

(c) We show $K_{inatt} \leq_m K_{PQ}$ via $f(x) = (A, B)$, where A is a DTA that ignores input and runs inattentive x, and B always halts.

(d) Since K_{PQ} is not r.e., by Theorem 7.37 we are done.

6. (a) The total number of configurations of the machine is upper bounded by $(n+1)m^n k$, as the product of the total number of possible head positions of the TM, the total number of all the possible different contents of the tape, and the total number of states the machine can be in for each tape content and head position. After this many configurations, some two cofigurations will have repeated, and since there is nothing in the machine configuration distinguishing these two occurrences, the lba will loop forever.

   (b) Yes, K_{lba} is recursive: on an input of size n, merely run the lba for $u(n,m,k)$ steps. If the lba has not halted yet, we are guaranteed it will never do so, and so we abort the computation and reject.

Extra credit problems

1. (Extra Credit Problem from Problem Set 6(I))
   Show that $\{xyyyz \ : \ x, y, z \in (0 \cup 1)^*, |y| \geq 1\}$ is not a DCFL.

   Solution: If the given language is a DCFL, then so is its complement. Let the complement of the given language be denoted by L. We will show that L is not context free.

   One's first instinct is to use the pumping lemma on a sufficiently long string, with $i = 3$, and claim that since the pumped up string has a thrice-repeated substring, we are done. Those students who've read the solutions to the midterm (problem 3.3) know, however, that the key issue is the *existence* of sufficiently long strings which can be pumped up. We will show that there exists an infinitely long string S with the property that it does not have a thrice-repeated substring. We can then get arbitrarily long (finite) strings in L by simply taking any prefix of S, and hence can apply the pumping lemma to pump the prefix out of L.

   (This version of the remainder is due to Sergey Brin)
   We define an infinite string S over $\{1, 2\}$ as below. ($S[1]$ is the first character of the string etc.).

$$S[i] = \begin{cases} S[i/3] & : \text{ if } i \bmod 3 = 0 \\ i \bmod 3 & : \text{ otherwise} \end{cases}$$

   Let $S/3$ denote the subsequence $S[3]S[6]S[9]\cdots$. Observe that $S = S/3$.

   Consider any substring UVW of S where $|U| = |V| = |W| = l$. If ($l \bmod 3 = 0$), then $U = V = W$ only if $S/3$ contains a substring $U'V'W'$ such that $|U'| = |V'| = |W'| = l/3$. Since $S = S/3$, we only need to consider the cases when $l \bmod 3 \neq 0$ and the rest follows by induction.

   Let p be the position in S of the first character of U. Then V will start from $p + l$ and W from $p + 2l$. If $l \bmod 3 \neq 0$, then $p \bmod 3$, ($p + l \bmod 3$) and ($p + 2l \bmod 3$) will all be different. Thus one of $U[1], V[1]$, or $W[1]$ will be 1, and another, 2. Thus some two of U, V and W will disagree on their initial character, thus showing that UVW is not a thrice-repeated substring.

2. (From Problem Set 7) 6.1-2 (e) (page 405)
Prove that $L = \{xy : |x| = |y| \text{ and } y \neq x\}$ is an NCA language.

Solution: (Due to David Rieber)

$$w \in L$$
$$\Leftrightarrow$$
$$w \in (a \cup b)^i a (a \cup b)^j (a \cup b)^i b (a \cup b)^j \bigcup (a \cup b)^i b (a \cup b)^j (a \cup b)^i a (a \cup b)^j \text{ for some } i, j \geq 0$$
$$\Leftrightarrow$$
$$w \in (a \cup b)^i a (a \cup b)^i (a \cup b)^j b (a \cup b)^j \bigcup (a \cup b)^i b (a \cup b)^i (a \cup b)^j a (a \cup b)^j$$
$$\Leftrightarrow$$
$$w \in (a \cup b)^i a (a \cup b)^{i+j} b (a \cup b)^j \bigcup (a \cup b)^i b (a \cup b)^{i+j} a (a \cup b)^j$$

Observe that the last characterization of L allows one to construct a nondeterministic counter for each of the unioned languages, and hence their union. A counter machine for the first of these languagages, for example, would count in i characters, for any i (nondeterminism), scan in an a, count down to 0 and then back up to any (nondeterministic) j, scan in a b, and count down to 0, and, mysteriously enough, this would be just right!

3. 7.6-4 (page 476)
Prove that every infinite r.e. language has an infinite recursive subset.

Solution:

Let M be an enumerator for the given infinite r.e. language, and assume that the strings are over $\{0, 1\}$. Then the language

$$\{w : w \text{ is the } j\text{th largest number output by } M, \text{ for some } j\}$$

is an infinite recursive language contained in the given r.e. language. A recognizer for the language would, given a string w, run M, and do the following:

> If M outputs a number larger than w before outputting (if at all) w, then REJECT. Else ACCEPT.

Since the given language is infinite, note that the first "statement" of the program will not be an infinite computation, as eventually a number larger than w will always be output.

4. Let A be an r.e. language consisting of description of Turing machines, $\{\langle M_1 \rangle, \langle M_2 \rangle, \ldots\}$. Show that there is a recursive language B consisting of descriptions of Turing machines such that every machine described in A has an equivalent one in B (and vice versa).

Solution: Let $A(i)$ be the ith Turing machine output by some enumerator for A. We can make B as follows: for each $A(i)$ in A, add to B the encoding of the Turing machine

$A(i)$, augmented with the number i hardcoded into its encoding in such a way that it does not alter the machine's functionality. A recognizer for B would pick this number i out of the encoding of a candidate Turing machine, run the enumerator for A, compare encodings of $A(i)$ with that of the candidate TM, and check to see if the encodings are the same, except for the augmenting number. Note that this involves only comparing encodings, and not testing for equivalence, and is hence a recursive test.