CS517 - Theory of Computation
Instructor: Prof. Mike Rosulek
Problem Set #1 -Due: Friday, Apr 13_{th} at 11:59am

1. Consider the following function $f\colon \mathbb{N} \to \mathcal{P}(\mathbb{N})$:

$$f(x) = \{i | \text{the } 2^i\text{'s place in the binary expansion of } x \text{ is } 1\}$$

Why does this f not contradict Cantor's theorem? Give an *explicit* counterexample to the claim that this f is a bijection.

[ANS]

f is a surjective function since $\forall y \in Y, \exists x \in X$ such that $y = f(x)$, here, $x = \sum\limits_{i \in y} 2^{2^i - 1}$

Multiple x can map to the same $f(x)$, then we have $|f(x)| < |\mathbb{N}| < \mathcal{P}(\mathbb{N})$, so f does not contradict Cantor's theorem.

Counterexample: $f(0) = f(4) = \varnothing$, is f is not a bijection.

2. A Turing printer (TP) is a Turing machine with a work tape, a special "print" tape, and a special "print" state. The TM starts with an empty work tape. Every time it enters the "print" state, we consider the current contents of the print-tape to be "printed." If P is a TP, then we write $L(P)$ to denote the set of strings that P eventually prints.

   (a) Prove that a language L is Turing-recognizable **if and only if** $L = L(P)$ for some TP P.

   [ANS]

   $\Rightarrow$

   First we show that if TM M recognizes a language L, we can construct the following enumerator P for L. Say that $s_1, s_2, s_3, \cdots$ is a list of all possible strings in Σ^*:

   E="Ignore the input.

   1. Repeat the following for $i = 1, 2, 3, \cdots$.
   2. Run M for i steps on each input, $s_1, s_2, s_3, \cdots$.
   3. If any computations accept, print out the corresponding s_j ."

   If M accepts a particular string s, eventually it will appear on the list generated by P. In fact, it will appear on the list infinitely many times because M runs from the beginning on each string for each repetition of step 1. This procedure gives the effect of running M in parallel on all possible input strings.

   $\Leftarrow$

   Now we do the other direction. If we have an enumerator P that enumerates a language L, which is $L = L(P)$, then there must be a TM M recognizes L. The TM M works in the following way:

   M="On input w:

   1. Run P. Every time that P outputs a string, compare it with s.
   2. If s ever appears in the output of P, accept."

   Clearly, M accepts those strings that appear on P's list.

   (b) Prove that a language L is Turing-decidable **if and only if** $L = L(P)$ for some TP P that prints strings in lexicographic order.

   [ANS]

   $\Rightarrow$

   Suppose a language L is Turing-decidable by a TM M. Let $s_1, s_2, s_3, \cdots$ be a lexicographic ordering of the strings in Σ^*. Then we can use M to construct an *enumerator* P as follows:

   P="Ignore the input,

   1. For $i = 1, 2, 3, \cdots$
   2. Run M on s_i
   3. If M accepts, print s_i, if M rejects, move on."

   We know that step 2 is guaranteed to terminate, because M decides (not just recognizes) L. So P can enumerates L, which is $L = L(P)$.

$\Leftarrow$
Suppose a language L is enumerated in lexicographic order by an *enumerator* P, which is $L = L(P)$. If L is finite, then of course it's decidable, so we suppose that L is infinite. A TM M which decides L works as follows:

M="On input w

1. Wait for P to print a string s.
2. If $s = w$, accept
3. If $s > w$ (lexicographically), reject
4. If $s < w$, go back to step 1."

Since P is guaranteed to print strings in lexicographic order, if it prints a string that comes after w in lexicographic order, then we can be sure that it will never print w, and therefore, w is not in L.
It is clear from M's description that M always halts, since M never runs out of strings, and there are only a finite number of strings $\leq w$.

(c) Prove that every infinite Turing-recognizable language L contains an infinite subset that is Turing-decidable.
[ANS]
Let L be an infinite Turing-recognizable language. Then, there exists an enumerator E that enumerates all strings in L (in some order, possibly with repetitions). We construct another enumerator E' that prints a subset of L in lexicographic order:

"Ignore the input.

1. Simulate E. When E prints its first string s_1, print s_1 and let $prev\_s = s_1$.
2. Continue simulating E.
3. When E is ready to print a new string s, check to see if s is longer than $prev\_s$ (this ensures s occurs after $prev\_s$ in lex. order). If so, then print s and let $prev\_s = s$, otherwise do not print s.
4. Go to 2."

It is clear that E' as constructed above only prints strings in L, therefore its language is a subset of L. Since L is infinite, there will always be strings in L longer than the current $prev\_s$, E will eventually print one of these and so will E' (and update $prev\_s$). Therefore, the language of E' is also infinite. Finally, since E' only prints strings in lexicographic order, its language is decidable as proved in (b). Thus, the language of E' is an infinite decidable subset of L.

3. A language C is said to **separate** A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Construct two Turing- recognizable languages A and B so that no Turing-decidable language separates them.
[ANS]
Let $A = \{\langle M\rangle | M(\langle M\rangle) \text{ rejects } \}$ and B $= \{\langle M\rangle | M(\langle M\rangle) \text{ accepts } \}$. They are not Turing separable; suppose that they are; let M_C be the separator. Considering $M_C(\langle M_C\rangle)$ we obtain a contradiction:
$M_C(\langle M_C\rangle)$ rejects $\Rightarrow M_C \in A \subseteq C \Rightarrow M_C \in C \Rightarrow M_C(\langle M_C\rangle)$ accepts.
$M_C(\langle M_C\rangle)$ accepts $\Rightarrow M_C \in B \subseteq \overline{C} \Rightarrow M_C \notin C \Rightarrow M_C(\langle M_C\rangle)$ rejects.