University of Nevada, Las Vegas Computer Science 456/656 Fall 2026

Answers to Assignment 3: Due Saturday October 19, 2026

1. True **T**/False **F**/Open **O**, meaning that the answer is not known to science at this time.

   (i) **O** $\mathcal{P} = \mathcal{NP}$.

   (ii) **T** If L is $\mathcal{P}$, it must be $\mathcal{NP}$ and also co-$\mathcal{NP}$.

   (iii) **O** The factoring problem for integers, given as binary numerals, is $\mathcal{NP}$-complete.

   (iv) **T** Every language is countable.

   (v) **F** There is an algorithm which decides whether any given context-free grammar generates the Dyck language.

   (vi) **T** For any rational number q, there is an algortihm which computes the n^{th} decimal digit of q for any n.

   (vii) **T** There is an algorithm which computes the n^{th} decimal digit of π for any given n.

   (viii) **F** For any real number x, there is an algortihm which computes the n^{th} decimal digit of x for any given n.

   (ix) **F** There is a program which computes $B(n)$ for any integer n, where B is the busy beaver function. (Look it up!)

   (x) **F** The problem of whether two C++ programs are equivalent is decidable. (Teo programs are defined to be *equivalent* if, given the same input they give the same output, regardless of how long they may run.)

2. Label each set listed below as **C**, countable, or **U**, uncountable. Recall that every C++ program is a string over the ASCII alphabet.

   (i) **C** $\mathbb{Z}$, the integers.

   (ii) **C** $\mathbb{Q}$, the rational numbers.

   (iii) **U** $\mathbb{R}$, the real numbers.

   (iv) **U** Complex numbers.

   (v) **C** Albebraic real numbers. (Look it up.)

   (vi) **C** Recursive real numbers.

   (vii) **U** Gaussian numbers.

   (viii) **U** Binary languges, *i.e.* languages over the binary alphabet $\{0, 1\}$.

   (ix) **C** Recursively enumerable binary languages.

   (x) **C** C++ programs. (**Not** all programs which have ever been written.)

   (xi) **U** Sets of integers.

   (xii) **C** Finite sets of integers.

   (xiii) **U** Finite sets of real numbers..

3. Give a polynomial time reduction of 3-SAT to the independent set problem.

   We define a polynomial time construction which, given any 3-CNF expression E which is the conjunction of k clauses computes a graph $G = R(E)$ which has an independent set of k vertices if and only if E is satisfiable.

   Let $E = C_1 * C_2 * \cdots * C_k$, where the i^{th} clause $C_i = t_{i,1} + t_{i,2} + t_{i,3}$, where each term $t_{i,j}$ is either a variable or the negation of a variable.

   Let $G = (V, E)$ where $V = \{v_{i,j} : 1 \leq i \leq k, j = 1, 2, 3\}$ the set of vertices of G. The set E of edges of G is defined as follows: there is an edge from $v_{i,j}$ to $v_{i',j'}$ if either $i = i'$ or $t_{i,j} * t_{i',j'}$ is a contradiction.

   I did **not** ask you to prove that this construction is a reduction, although it is not difficult.

   In the problems below, if u, v are strings over any alphabet Σ, we say $u < v$ to mean that u is less than v in the canonical order of Σ^*.

4. Prove that every decidable language L can be enumerated in canonical order by some machine.

   Proof. Let Σ be the alphabet of L. Suppose there ia a program D_L which decides L. Let $s_1, s_2, \ldots$ be the canonical enumeration of Σ^*. The following program enumerates L in canonical order:

   for all i from 1 to ∞
     if ($D_L(s_i)$ (that is, $s_i \in L$)
       write s_i

   Every member of L will be written, and the enumeration will be in canonical order because $s_i < s_j$ if $i < j$. □

5. Suppose there is a machine which enumerates a language L in canonical order. Prove that L is decidable.

   Proof. If L is finite, then it is decidable, since a program can search for W in the list of member of L in finite time. Otherwise, let $w_1, \ldots$ be a canonical order enumeration of L. The following program decides whether a given string $w \in \Sigma^*$ is a member of L.

   input w
   for all i from 1 to ∞
     if ($w = w_i$)
       Accept and halt.
     else if ($w < w_i$)
       Reject and halt.

   If $w \in L$, the program will halt because $w = w_i$ for some i. If $w \notin L$, then it will not occur in the enumeration, and eventually a string larger than w will be listed. Since the enumeration is in canonical order, w will never be listed, hence $w \notin L$. □

6. Suppose L is recursively enumerable, that is, there is a machine which enumerates a language L. Prove that there is a machine which accepts L.

Proof. Let $w_1, \ldots$ be an enumeration of L written by some machine. Let P be the following program.

input w
for all i from 1 to ∞
  if ($w = w_i$)
    Accept and halt.

$w \in L$ if and only if $w = w_i$ for some i, and thus P will halt and accept w if and only if $w \in L$. □

7. Let L be a language accepted some machine. Prove that L is recursively enumerable.

Proof. Let Σ be the alphabet of L, let $s_1, \ldots$ be the canonical enumeration of Σ^*. Let A_L be a program which accepts L. Note that any execution of A_L consists of a sequence of steps. For any $w \in L$, let $T(w)$ be the number of steps[1] it takes A_L to accept w. Let P_E be the following program.

for all t from 1 to ∞
  for all i from 1 to t
    if (A_L accepts s_i within its first t steps)
      write s_i

We need to show that P_E enumerates L. If P_E writes s_i then s_i is accepted by A_L and hence is in L. Conversely, suppose $w \in L$, then $w = s_i$ for some i. We need to show that P_E eventually writes w.

Case 1: $i \leq T_w$. Let $t = T_w$. During the t^{th} iteration of the outer loop of P_E, A_L will execute i steps and accept w, hence w will be written.

Case 2: $i > T_w$. During the i^{th} iteration of the outer loop of P_E, A_L will accept w during the last iteration of the inner loop, and hence w will be written.

Thus every member of L is written. Since every string written must be accepted by A_L, no string not in L will be written. □

Note that P_E writes every member of L infinitely many times, but this is allowed by the definition of an enumeration. However, we could modify P_E to avoid duplication.

[1] If L is undecidable T is a non-recursive (uncomputable) function.