The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual

Chapter 5, Additional Problem 5.3

1. The statement is false. One intuitive way to see why is to look at the equivalent definition of XOR: it involves intersection and union (under which r.e. sets are closed) and subtraction (or complement), under which r.e. sets are not closed. A simple counterexample is given by $S = \mathbb{N}$ and $T = K$; then the exclusive-or of the two is $\overline{K}$, which is not r.e.

2. The statement is true. If S or T is empty, then so is U, which is thus r.e., as desired. Assume then that S and T are non-empty; then they have total enumeration functions f and g. We claim that

$$h(z) = f(\Pi_1(z)) + g(\Pi_2(z))$$

is an enumeration function for Us, i.e., that $ranh = U$. Assume that $x \in ranh$; then there exists z with $x = h(z) = f(\Pi_1(z)) + g(\Pi_2(z)) = a + b$, where $a \in S$ and $b \in T$, since f enumerates S and g enumerates T; thus $ranh \subseteq U$. Now assume that $x \in U$; then we can write $x = a + b$, where $a \in S$ and $b \in T$; but then there exist y_1 and y_2 such that $f(y_1) = a$ and $g(y_2) = b$, since f enumerates S and g enumerates T; writing $z =< y_1, y_2 >$, we now observe that $h(z) = f(\Pi_1(z)) + g(\Pi_2(z)) = f(y_1) + g(y_2) = a + b = x$, so that $x \in ranh$; thus $U \subseteq ranh$.

3. The statement is false. The sets defined in Exercise 5.19, namely the collection of sets $\{\overline{K_t} \mid t \in \mathbb{N}\}$, give us the desired counterexample. Exercise 5.19 shows that the intersection of all such sets is $\overline{K}$, which is not r.e. and thus also not recursive. All of the $\overline{K_t}$ sets are recursive, however. Thus it remains only to show that we can design a recursive set S such that, for each of the $\overline{K_t}$, S contains at least one program that serves as characteristic function for that $\overline{K_t}$. The following program serves as characteristic function for $\overline{K_t}$:

```
if step(x,x,t) > 0
   then print 0
   else print 1
```

(under the usual assumption about step functions). For each t, we can easily compute the index of this program—in fact, we can write a total recursive function that, given the value of t, outputs the index of this program; moreover, the index grows as t grows. We let S be the set of these indices for all values of t. Given some arbitrary index i, we can decide if it is the index of one of these programs—just generate indices for increasing values of t until either one index equals i or one index exceeds i. Hence S is recursive. Now we have identified a recursive set containing only characteristic functions and such that the intersection of the recursive sets these functions define is non-recursive—our desired counterexample.