Assignment #4

Sample Solutions

Question 1: Expanding the Language $L$ and Causing Trouble

After having worked for some time with the language $L$, we start feeling annoyed about the fact that, if we do not want to use macros, there are only three different types of instruction (or four if you count the rather impractical $V \leftarrow V$ instruction type). So we decide to add the instruction type $V \leftarrow 0$ (zero) to our language $L$ and thereby create a new language $L'$. Unfortunately, now we realize that we have to write new universal programs $U'_n$ for the new language $L'$ as well, and that implies that we also have to devise a new system of enumerating programs in $L'$ (associating every program with a unique number and every number with a unique program).

Well, anyway, we decided to do this, and now we will stick with this decision. So please

(a) write down the differences of your new enumerating scheme compared to the one for the original language $L$, and

(b) write down the entire $L'$ code for the new universal programs $U'_n$ that can execute the code of any $L'$ program. Of course you can use macros, and you can reuse most of the code that we wrote for programming $U_n$ (see slides and textbook).

(a)

$$\#(I) = <a, <b, c>>,$$

where

$b = 0$ means instruction type $V \leftarrow V$

$b = 1$ means instruction type $V \leftarrow V + 1$

$b = 2$ means instruction type $V \leftarrow V - 1$

$b = 3$ means instruction type $V \leftarrow 0$  // our new instruction type!

$b > 3$ means instruction type IF $V \neq 0$ GOTO L, where $b = #(L) + 3$

Everything else remains the same.
Here is the new universal program with the new or modified parts in bold font:

\[
\begin{align*}
Z & \leftarrow X_{n+1} + 1 \\
S & \leftarrow \prod_{i=1}^{n} (p_{2i})^{X_i} \\
K & \leftarrow 1 \\
\text{[C]} & \quad \text{IF } K = \text{LT}(Z) + 1 \lor K = 0 \text{ GOTO F} \\
& \quad U \leftarrow r((Z)_K) \\
& \quad P \leftarrow p_{r(U)+1} \\
& \quad \text{IF } l(U) = 0 \text{ GOTO N} \\
& \quad \text{IF } l(U) = 1 \text{ GOTO A} \\
& \quad \text{IF } \sim(P \mid S) \text{ GOTO N} \\
& \quad \text{IF } l(U) = 2 \text{ GOTO M} \\
& \quad \text{IF } l(U) = 3 \text{ GOTO Z} \\
& \quad K \leftarrow \min_{\text{def}(Z)} [l((Z)_K) + 3 = l(U)] \\
& \quad \text{GOTO C} \\
\text{[Z]} & \quad S \leftarrow \lfloor S/P \rfloor \\
& \quad \text{IF } P \mid S \text{ GOTO Z} \\
& \quad \text{GOTO N} \\
\text{[M]} & \quad S \leftarrow \lfloor S/P \rfloor \\
& \quad \text{GOTO N} \\
\text{[A]} & \quad S \leftarrow S - P \\
\text{[N]} & \quad K \leftarrow K + 1 \\
& \quad \text{GOTO C} \\
\text{[F]} & \quad Y \leftarrow (S)_1
\end{align*}
\]

**Question 2: Prove It!**

Let \( A, B \) be sets. Prove or disprove:

(a) For all sets \( A \) and \( B \), if \( A \) and \( B \) are both r.e., then \( A \cap B \) is also r.e.

Since \( A \) and \( B \) are r.e., there must be partially computable functions \( g(x) \) and \( h(x) \) such that:

\[
A = \{ x \in \mathbb{N} \mid g(x) \downarrow \}
\]

\[
B = \{ x \in \mathbb{N} \mid h(x) \downarrow \}
\]

This means that we can write the following program that computes the function \( f(x) \):

\[
\begin{align*}
X_1 & \leftarrow g(x) \\
X_1 & \leftarrow h(x)
\end{align*}
\]

This function is defined on precisely those inputs on which both \( g(x) \) and \( h(x) \) are defined.

Therefore, we have:

\[
A \cap B = \{ x \in \mathbb{N} \mid f(x) \downarrow \}
\]
This shows that $A \cap B$ is r.e.

(b) For all sets $A$ and $B$, if $A$ and $B$ are both r.e., then $A \cup B$ is also r.e.

This is similar to (a), but we now need a function $f(x)$ that is defined on those values on which either $g(x)$ or $h(x)$ is defined. So we cannot just call $g(x)$ and then $h(x)$, because the program for $g(x)$ may never terminate, and then we would never get to test whether the program computing $h(x)$ terminates. So we have to use the dovetailing technique we discussed in class. Let $g_0$ and $h_0$ be the numbers of the programs computing $g(x)$ and $h(x)$, respectively. Then we can write the following program computing $f(x)$:


[A]

$X_2 \leftarrow \text{STP}(X_1, g_0, X_3)$

IF $X_2$ GOTO E

$X_2 \leftarrow \text{STP}(X_1, h_0, X_3)$

IF $X_2$ GOTO E

$X_3 \leftarrow X_3 + 1$

GOTO A

This program will terminate on an input $x$ if and only if either $g(x)$ or $h(x)$ is defined on $x$ (or both are defined on $x$). Therefore:

$A \cup B = \{x \in \mathbb{N} \mid f(x) \downarrow\}$

This shows that $A \cup B$ is r.e.

(c) If $A \cup B$ is r.e., then $A$ and $B$ are both r.e.

The statement is false. Assume that $A = \mathbb{N}$, $B = \neg K$. Then $A \cup B = \mathbb{N}$ is r.e., but $B$ is not r.e.

(d) If $A \supset B$ and $B$ is r.e., then $A$ is also r.e.

The statement is false. Assume that $A = \neg K$, $B = \emptyset$. Then $A \supset B$ and $B$ is r.e., but $A$ is not r.e.
Let $B = \{ f(n) | n \in \mathbb{N} \}$, where $f$ is a strictly increasing computable function (i.e., $f(n + 1) > f(n)$ for all $n$). Prove that $B$ is a recursive set.

$B$ is a recursive set if and only if there is a computable characteristic function (predicate) $P_B(x)$ such that

$$B = \{ x \in \mathbb{N} | P_B(x) \}$$

We know the function $f(n)$, and we know that $B$ is simply the range of $f$. So the idea is to write a program $P$ that computes $P_B(x)$ by going through all outputs of $f$, i.e., $f(0)$, $f(1)$, $f(2)$, $f(3)$, and so one, and once any of these outputs equals $x$, then the output $P_B(x) = 1$.

If $f(n)$ could just be any function, then we would have a problem: We would not know when to stop looking through the sequence and determine that none of the values in the sequence equals $x$, i.e., that the output $P_B(x) = 0$. We could then show that $B$ is r.e., but not that it is recursive.

Fortunately, we know that $f$ is strictly increasing. So when we go through the sequence $f(0)$, $f(1)$, $f(2)$, $f(3)$, … without finding $x$ until we get a value that is greater than $x$, then we know that $x$ is not in $B$. This is because $f$ is strictly increasing, and therefore, each value in the sequence is greater than the previous one, meaning that once we get a value greater than $x$, we will never get any smaller values again, and so if we have not found $x$ in the sequence yet, we know that $x$ is not contained in the sequence at all.

So we can indeed write a program $P$ that always terminates and computes $P_B(x)$:

[A] $Z_1 \leftarrow f(Z_2)$
   IF $Z_1 = X_1$ GOTO B
   IF $Z_1 > X_1$ GOTO E
   $Z_2 \leftarrow Z_2 + 1$
   GOTO A  

[B] $Y \leftarrow Y + 1$

Therefore, $B$ is a recursive set.