

CS 620 – Theory of Computation – Fall 2009

Instructor: Marc Pomplun

Assignment #2 – Sample Solutions

Question 1:

a) Write down the code of Program \mathcal{P} in the language \mathcal{L} for which $\#(\mathcal{P}) = 1919$.

$$\#(\mathcal{P}) = [\#(I_1), \#(I_2), \dots, \#(I_k)] - 1 \leftrightarrow [\#(I_1), \#(I_2), \dots, \#(I_k)] = \#(\mathcal{P}) + 1$$

$$[\#(I_1), \#(I_2), \dots, \#(I_k)] = 1919 + 1 = 1920 = 2^7 * 3^1 * 5^1$$

$$[\#(I_1), \#(I_2), \#(I_3)] = [7, 1, 1]$$

Instruction 1

$$\langle a, \langle b, c \rangle \rangle = 7$$

$$7 = \langle 3, 0 \rangle = \langle 3, \langle 0, 0 \rangle \rangle$$

$$\#(I_1) = \langle a, \langle b, c \rangle \rangle = \langle 3, \langle 0, 0 \rangle \rangle = [C] Y \leftarrow Y$$

Instructions 2 and 3

$$\langle a, \langle b, c \rangle \rangle = 1$$

$$1 = \langle 1, 0 \rangle = \langle 1, \langle 0, 0 \rangle \rangle$$

$$\#(I_1) = \langle a, \langle b, c \rangle \rangle = \langle 3, \langle 0, 0 \rangle \rangle = [A] Y \leftarrow Y$$

Resulting program:

[C] $Y \leftarrow Y$

[A] $Y \leftarrow Y$

[A] $Y \leftarrow Y$

b) What is the number of the following program?

[B] $X \leftarrow X - 1$

$Y \leftarrow Y + 1$

$Y \leftarrow Y + 1$

IF $X \neq 0$ GOTO B

You do not have to compute the numerical values of expressions such as 3^{27} that would result in huge numbers.

[B] $X \leftarrow X - 1 \leftrightarrow \langle a, \langle b, c \rangle \rangle \leftrightarrow \langle 2, \langle 2, 1 \rangle \rangle = \langle 2, 11 \rangle = 91$
 $Y \leftarrow Y + 1 \leftrightarrow \langle a, \langle b, c \rangle \rangle \leftrightarrow \langle 0, \langle 1, 0 \rangle \rangle = \langle 0, 1 \rangle = 2$
 $Y \leftarrow Y + 1 \leftrightarrow \langle a, \langle b, c \rangle \rangle \leftrightarrow \langle 0, \langle 1, 0 \rangle \rangle = \langle 0, 1 \rangle = 2$
 IF $X \neq 0$ GOTO B $\leftrightarrow \langle a, \langle b, c \rangle \rangle \leftrightarrow \langle 0, \langle 4, 1 \rangle \rangle = \langle 0, 47 \rangle = 94$

$$\#(\mathcal{P}) = 2^{91} * 3^2 * 5^2 * 7^{94} - 1$$

Question 2:

Do you remember how we used the pairing function and the Gödel numbering to associate each program in the language \mathcal{L} with a unique natural number? Then we used this numbering to numerically describe the interpretation of programs in \mathcal{L} . Now it is your task to develop such one-to-one mappings for other things. If you think that a mapping cannot be defined, please give a reason.

a) Define such a mapping for quintuples of natural numbers.

$$(a, b, c, d, e) \leftrightarrow \langle \langle a, b \rangle, \langle c, \langle d, e \rangle \rangle \rangle \quad (\text{for example})$$

b) Define such a mapping for pairs (a, b) , where a is a natural number and b is a letter from the English alphabet.

$$(a, b) \leftrightarrow 26a + b, \text{ where } b \in \{0, \dots, 25\}, \text{ corresponding to the letters } a, \dots, z.$$

c) Define such a mapping for pairs (x, y) , where x and y are letters from the English alphabet.

Such a mapping cannot be computed, because there is only a finite number of such pairs.

d) Define such a mapping for sequences (of any length) of even numbers.

A simple, almost correct mapping can easily be found:

$$(x_1, x_2, \dots, x_n) \leftrightarrow [f(x_1), f(x_2), \dots, f(x_n)] - 1, \text{ where } f(x) = \lfloor x/2 \rfloor$$

The problem with this mapping is that two sequences that only differ in their number of trailing zeroes will be mapped onto the same natural number.

We can fix this problem by encoding the information about the training zeroes into our mapping. Let us name the elements in the sequence as follows:

$$(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m})$$

So that $x_n \neq 0$ and $x_{n+1} = 0, \dots, x_{n+m} = 0$

For example, $(6, 3, 8, 0, 0)$, $n = 3$ and $m = 2$.

If the last (rightmost) number in our sequence is not zero, then $m = 0$.

Then we can define a mapping as follows:

$$(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}) \leftrightarrow [m, f(x_1), f(x_2), \dots, f(x_n)] - 1 \quad \text{with} \quad f(x) = \lfloor x/2 \rfloor$$

This was much trickier than intended, and so any approximate solution will receive full points.

e) Define such a mapping for strings made up of the letters of the English alphabet, including the empty string, that do not end with the letter x.

This question is reminiscent of our encoding of programs that are not allowed to end with an instruction of code 0. So let us do the same here, based on a system similar to the one we use every day – the encoding of numbers as strings of digits, for example:

$$629 = 6 \cdot 10^2 + 2 \cdot 10^1 + 9 \cdot 10^0$$

Here, adding zeroes to the left of the string does not change the associated number, which is the reason why conventionally we do not write any leading zeroes (except for the number 0). In our scheme, we have such a rule for trailing x's, and we should therefore proceed in the reverse order and translate x's into zeroes. Also, since we have 26 different letters, we have to use the basis 26 instead of 10 here.

This is how we translate letters into numbers:

$$f('x') = 0$$

$$f('a') = 1$$

...

$$f('w') = 23$$

$$f('y') = 24$$

$$f('z') = 25$$

Then our mapping is defined by:

$$a_0 a_1 a_2 \dots a_n \leftrightarrow \sum_{i=0}^n [26^i \cdot f(a_i)]$$

Bonus question:

f) Define such a mapping for strings made up of the letters of the English alphabet, including the empty string.

One possible solution is similar to the one we use in (d): Allow training zeroes (x's) and encode their number in the mapping.

Then we have a sequence

$$a_0 a_1 a_2 \dots a_n a_{n+1} a_{n+2} \dots a_{n+m}$$

so that $a_n \neq 0$ and $a_{n+1} = 0, \dots, a_{n+m} = 0$. This allows us to define the following mapping:

$$a_0 a_1 a_2 \dots a_n a_{n+1} a_{n+2} \dots a_{n+m} \leftrightarrow \left\langle m, \sum_{i=0}^n [26^i \cdot f(a_i)] \right\rangle$$

Another solution is to use a different representation of the letters that avoids any zeroes:

$$\begin{aligned} g('a') &= 1 \\ g('b') &= 2 \\ &\dots \\ g('z') &= 26 \end{aligned}$$

Interestingly, this enables us to reuse our previous equation:

$$a_0 a_1 a_2 \dots a_n \leftrightarrow \sum_{i=0}^n [26^i \cdot g(a_i)]$$

This gives us another bijective mapping, and this time, trailing x's are no problem anymore. Strange, but true. We will discuss this scheme in more detail later in the course when we consider strings as inputs and outputs of programs (instead of just using boring numbers).

Question 3:

Let $H(a, b) = a$ if $\Phi(a, b) \downarrow$
 $= \uparrow$ otherwise.

Show that $H(x)$ is partially computable.

We can write a program:

$$Y \leftarrow X_1$$
$$Z \leftarrow \Phi(X_1, X_2)$$

Question 4:

Prove or disprove (you can just give a detailed verbal argument): If there were an upper limit n for the values of all variables in the language \mathcal{L} (so for any variable V , $0 \leq V \leq n$), then $\text{HALT}(x, y)$ would be a computable predicate.

First of all, since we did not get as far in the course as I had thought, this question is more difficult than I wanted it to be, and it is now a bonus question. Second, some of you misunderstood the question: $\text{HALT}(x, y)$ is not affected by the change in \mathcal{L} ; it could be programmed in any language.

At any given point during the execution of a program, the remaining sequence of computational steps completely depends on the current snapshot. This means that if during the computation we encounter the same snapshot more than once, we know that the program is in an infinite loop. Now if the values of variables are restricted, so is the number of different snapshots $\#S$ that can occur:

$$\#S = \#I(n + 1)^{\#V},$$

where $\#I$ is the number of instructions in the program and $\#V$ is the number of variables used by the program. Since there are only $\#S$ different snapshots, we know that if the program has not terminated after at most $\#S$ computational steps, at least one of the snapshots must have occurred more than once, i.e., the program entered an infinite loop. That allows us to calculate $\text{HALT}(x, y)$ for the restricted language by simulating the execution of program y on input x for $\#S$ steps.

An estimate for a number greater than or equal $\#S$ can be obtained from y by a primitive recursive function, because $\#I = \text{Lt}(y)$, and $\#V \leq \#I$.

Therefore, under the given circumstances, $\text{HALT}(x, y)$ becomes a computable function.