Lecture #9: Emulating Computations — and a "Universal" $\mathcal{S}$-Program

What Will Happen During the Lecture

Remember... You Had Homework!

Students were asked to have worked through a set of notes before this first lecture:

- Lecture Notes — "Emulating Computations — and a "Universal" $\mathcal{S}$-Program

Of course, you may attend the lecture presentation if you have not worked through this material ahead of time — but it will not be repeated for you during the lecture presentation.

Problem To Be Solved

In order to describe a "universal" $\mathcal{S}$-program it was necessary to describe a way to represent, or "encode", $\mathcal{S}$-programs as natural numbers. In particular, for each $\mathcal{S}$-program $\mathcal{P}$, a natural number, $\#(\mathcal{P}) \in \mathbb{N}$, which "encodes" $\mathcal{P}$, was defined.

It was noted that if $\mathcal{P}$ is an $\mathcal{S}$-program, and $\mathcal{Q}$ is the $\mathcal{S}$-program obtained by adding the "dummy" statement

$$Y \leftarrow Y$$

as a final statement, then $\#(\mathcal{Q}) = \#(\mathcal{P})$.

Now consider an $\mathcal{S}$-computable partial or total function $f : \mathbb{N}^k \to \mathbb{N}$. Since this is function is $\mathcal{S}$-computable, there exists a $\mathcal{S}$-program, $\mathcal{P}$, the computes f. Since the "dummy statement" $Y \leftarrow Y$ does not change a program's state one can see the the above program $\mathcal{Q}$, obtained from $\mathcal{P}$ by appending a copy of the dummy statement "$Y \leftarrow Y$" at the end, is another $\mathcal{S}$-program that also computes the function f.

Indeed, we can obtain an infinite sequence

$$\mathcal{P}_0 = \mathcal{P}, \mathcal{P}_2, \mathcal{P}_3, \ldots$$

of different $\mathcal{S}$-programs, that each compute f, by setting $\mathcal{P}_i$ to be the program obtained from $\mathcal{P}$ by appending i copies of the dummy statement "$Y \leftarrow Y$" at the end, for every non-negative integer i.

However, $\#(\mathcal{P}_i) = \#(\mathcal{P})$ for every non-negative integer, here,. Since there might also be different programs that compute f, this *does not* answer the following question: Must all the $\mathcal{S}$-programs, that compute a given partial or total function $f : \mathbb{N}^k \to \mathbb{N}$, have the same encoding?

- ***The Padding Lemma*** answers this, along with an even stronger question: It asserts that, for every computable partial or total function $f : \mathbb{N}^k \to \mathbb{N}$, there exist *infinitely many* non-negative integers that are all encodings of $\mathcal{S}$-programs that compute f.
- Indeed, there exists a ***primitive recursive*** function *gen* : $\mathbb{N}^2 \to \mathbb{N}$ which satisfy the following properties.
  - For every non-negative integer in,

    $$\mathit{gen}(n, 0), \mathit{gen}(n, 1), \mathit{gen}(n, 2), \mathit{gen}(n, 3) \ldots$$

    are all encodings of $\mathcal{S}$-programs that compute the same partial or total function.
  - $\mathit{gen}(n, i) < \mathit{gen}(n, i + 1)$ for all $n, i \in \mathbb{N}$ — so that the natural numbers

    $$\mathit{gen}(n, 0), \mathit{gen}(n, 1), \mathit{gen}(n, 2), \mathit{gen}(n, 3) \ldots$$

    are distinct.

During the lecture presentation a simple proof of the above "Padding Lemma", and a description of a function *gen* : $\mathbb{N}^2 \to \mathbb{N}$, with the above properties — along with an explanation why this function is primitive recursive.

The supplemental document for this lecture describes universal Turing machines, describing encodings of Turing machines that compute functions, in the process. If time permits, a "Padding Lemma for Turing Machines" will also be discussed.

Note: If you have time, please try to find a proof of the "Padding Lemma" yourself — perhaps, by considering a *different* way that copies of the dummy statement, "$Y \leftarrow Y$", might be used.