Numbering Turing Machines

Turing Machines can be numbered. More specifically: we can *code* Turing Machines as natural numbers in such a way that no two Turing Machines are assigned the same code.

In this section I'll describe one way of doing so.

Two preliminary observations:

1. The only differences between Turing Machines that we'll care about here are differences in their programs. Accordingly, we will only count Turing Machines as different if they instantiate different programs. On the other hand, we will treat Turing Machines as different even if there are differences in their programs that make no difference to their behavior. As long as their programs consist of different lists of symbols, the machines will be counted as distinct.
2. I will simplify the discussion by focusing on **one-symbol** Turing Machines, in which the only symbol allowed on the tape (not counting blanks) is "1". This looks like a substantial assumption, but it's actually not. As you'll be asked to verify in one of the exercises below, every function that can be computed on a many-symbol Turing Machine can also be computed on a one-symbol Turing Machine.

The coding scheme we will discuss in this section proceeds in stages. First, we'll code a Turing machine as a sequence of symbols; then we'll code the sequence of symbols as a sequence of numerals; finally, we'll code the sequence of numerals as a single number:

$$\text{Turing Machine} \rightarrow \text{Sequence of symbols} \rightarrow \text{Sequence of numbers} \rightarrow \text{Unique Number}$$

Let us consider each stage in turn.

Turing Machine $\rightarrow$Sequence of symbols

Since we are counting Turing Machines with the same program as one and the same, we can think of a Turing Machine program as a sequence of command lines.

Each command line is a sequence of symbols.

So all we need to do to encode a Turing Machine as a sequence of symbols is concatenate its commands.

For instance, the Turing Machine with program:

$$\begin{array}{ccccc} 0 & \_ & 1 & * & 0 \\ 0 & 1 & \_ & l & 1 \end{array}$$

gets coded as the sequence of symbols:

$$0 \quad \_ \quad 1 \quad * \quad 0 \quad 0 \quad 1 \quad \_ \quad l \quad 1$$

Sequence of symbols $\rightarrow$ Sequence of numbers

We have managed to code each Turing Machine as a sequence of symbols.

We will now code each of these symbols as a number.

Recall that each command line in a Turing Machine program results from filling out the following parameters:

⟨current state ⟩ ⟨ current symbol ⟩ ⟨new symbol ⟩ ⟨direction ⟩ ⟨new state ⟩

We will employ the following coding scheme:

- ⟨ current state ⟩ and ⟨ new state ⟩ will always be filled in with numerals (which are names of numbers). So we'll code each numeral by the number it represents:

$$\begin{array}{ccc} \text{"0"} & \rightarrow & 0 \\ \text{"1"} & \rightarrow & 1 \\ \vdots & & \vdots \end{array}$$

- ⟨current symbol ⟩ and ⟨ new symbol ⟩ will always be filled in with "1" or "_ ". (Recall that we are working with one-symbol Turing Machines.) So we'll use the following codes:

$$\begin{array}{ccc} \text{"\_"} & \rightarrow & 0 \\ \text{"1"} & \rightarrow & 1 \end{array}$$

- ⟨direction ⟩ is "r", "$*$" or "l". We'll use the following codes:

$$\begin{array}{ccc} \text{"r"} & \rightarrow & 0 \\ \text{"}*\text{"} & \rightarrow & 1 \\ \text{"l"} & \rightarrow & 2 \end{array}$$

So, for instance, the sequence of symbols

$$0 \quad \_ \quad 1 \quad * \quad 0 \quad 0 \quad 1 \quad \_ \quad l \quad 1$$

gets transformed into the sequence of numbers:

$$0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 2 \quad 1$$

Sequence of numbers →Unique number

The final step in our coding scheme is a method for coding each sequence of numbers as a single natural number.

There are many ways of doing so, but the method we will consider here is due to the great Austrian logician and philosopher Kurt G¨odel, about whom you'll be hearing more in the next lecture. It is extremely simple. One simply codes the sequence of numerals ⟨ n_1, n_2, . . ., n_k ⟩ as the number:

$p_1{}^{n_1+1} \cdot p_2{}^{n_2+1} \cdot \ldots \cdot p_k{}^{n_k+1}$

where $p_1 \ldots p_k$ are the first k prime numbers. (The "empty" Turing Machine, which has no command lines, is coded as the number 1.)

For instance, the sequence of numbers ⟨0,0,1,2,0 ⟩ gets coded as:

$2^{0+1} \cdot 3^{0+1} \cdot 5^{1+1} \cdot 7^{2+1} \cdot 11^{0+1} = 2 \cdot 3 \cdot 25 \cdot 343 \cdot 11 = 565,950$

(Why add one to exponents? Because otherwise our coding scheme would be ambiguous. For instance, it would use 1 as a code for each of the following sequences: ⟨0⟩, ⟨0,0⟩, ⟨0,0,0⟩, etc.)

To ensure that this coding scheme gives us what we want, we need to verify that different Turing Machines are always assigned different codes.

Fortunately, this is an immediate consequence of the following:

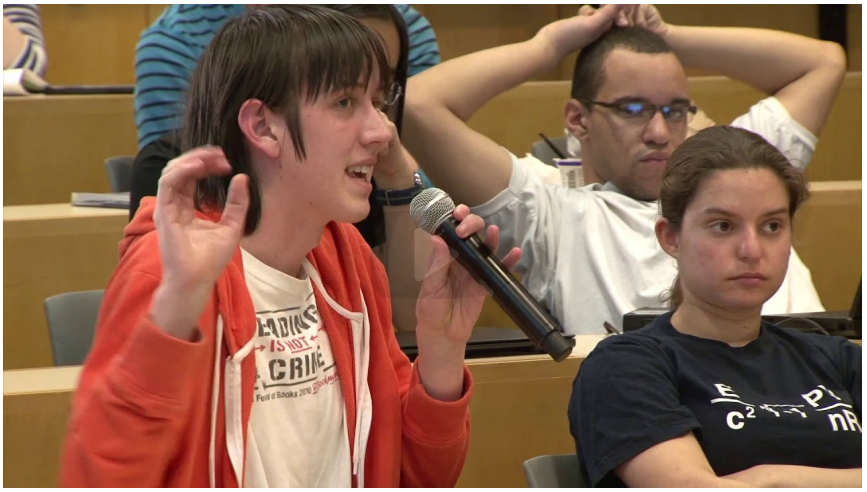
> **Fundamental Theorem of Arithmetic**
> Every positive integer greater than 1 has a unique decomposition into primes.

We have now done what we set out to do: we have identified a coding-scheme which assigns a different number to each one-symbol Turing Machine.

Since the discussion below will presuppose that a fixed coding scheme is in place, I hereby stipulate that it is to be the coding scheme described in this section, with one small qualification. Our scheme so far does not associate a Turing Machine with every number, since not every number codes a valid sequence of command lines. This turns out to be a nuisance, so I will stipulate that any number that doesn't code a valid sequence of command lines is to be treated as a code for the "empty" Turing Machine. The result is a coding scheme on which: (a) every number codes some Turing Machine, and (b) every Turing Machine is coded by some number.

Warning: In the video below, we encode "left" as 1 and "stay" as 2, contrary to the encoding described above. You should use the encoding described above, not the encoding in the video, when you do the exercises.

Video Review: Numbering Turing Machines



0:00 / 0:00 1.50x

Start of transcript. Skip to the end.

Here is one way in which you can think about the numbering.

And this is the method that I use in the lecture notes.

The first observation is that you

can think of each command line of a Turing machine

as just a sequence of five numbers.

Now, it's true that the way

Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

Problem 1

1/1 point (ungraded)

On the coding system above, what number gets assigned to the Turing Machine whose program consists of the following command line?

$$0 \quad 1 \quad \_ \quad * \quad 1$$

533610 ✔ **Answer:** 533610

533610

Explanation

The given command line is first transformed into the sequence "0 1 0 1 1", which gets coded as the following number:

$$2^{0+1} \cdot 3^{1+1} \cdot 5^{0+1} \cdot 7^{1+1} \cdot 11^{1+1} = 2 \cdot 9 \cdot 5 \cdot 49 \cdot 121 = 533,610$$

Submit

Answers are displayed within the problem

Problem 2

1/1 point (ungraded)

Using the coding system above, which of the following programs corresponds to Turing Machine number 97,020?

- $1 \quad 1 \quad 1 \quad * \quad 0$
- $1 \quad 1 \quad \_ \quad * \quad 0$

○ $1 \quad \_ \quad 1 \quad * \quad 0$

✔

Explanation
It is easy to verify that

$$97,020 = 2^{1+1} \cdot 3^{1+1} \cdot 5^{0+1} \cdot 7^{1+1} \cdot 11^{0+1}$$

So 97,020 codes the following sequence of numerals:

$$1 \quad 1 \quad 0 \quad 1 \quad 0$$

which corresponds to the following sequence of symbols:

$$1 \quad 1 \quad \_ \quad * \quad 0$$

Submit

Answers are displayed within the problem

Problem 3

1 point possible (ungraded)
Show that every function that can be computed on an n-symbol Turing Machine can also be computed on a one-symbol Turing Machine (which is only allowed ones and blanks on the tape).

☐ Done

Submit

Problem 4

1/1 point (ungraded)
As it turns out, every function that can be computed on a Turing Machine whose tape is infinite in both directions can also be computed on a Turing Machine whose tape is infinite in only one direction.

Suppose that M^{∞^2} is a Turing Machine that computes function $f(n)$. Sketch a procedure for transforming M^{∞^2}'s into a Turing Machine M^{∞} that computes $f(n)$ while using a tape that is infinite only in one direction.

☑ Done

✔

Explanation

Let me sketch a procedure for building M^{∞} out of M^{∞^2}. The intuitive idea is that M^{∞} shifts the contents of its tape one cell to the right whenever it is at risk of reaching the end of its tape to the left.

Here are some additional details. We will assume, for concreteness, that the tape is infinite towards the right. We also assume, with no loss of generality, that M^{∞^2} is a one-symbol machine. M^{∞}, in contrast, will be a many-symbol machine. But, as we saw in the preceding exercise, it could easily be transformed into a one-symbol machine. When computing $f(n)$, M^{∞} will start out with a tape that is blank except for a sequence of n-ones, with the reader positioned on the left-most one. M^{∞} then proceeds as follows:

Step 1

M^{∞} replaces the first one in the sequence with a special symbol "B" (for "beginning"), adds a one at the end of the sequence (to replace the deleted one at the beginning of the sequence), and adds a special symbol "E" (for "end") to the right of the new one.

Step 2

For each state s of M^{∞^2}, M^{∞}'s program will include additional command lines specifying the behavior of the machine when in state s reading "B" or "E":

- M^{∞}'s command line for s when reading "B" will initiate a subroutine that adds a blank between the "B" and the symbol to its right. This is done by translating the sequence of ones and blanks that is to the right of "B" (along with the "E" at the end) one cell to the right. After doing so, M^{∞} positions its reader on the newly created blank to the right of "B" and goes back to state s.

- M^{∞}'s command line for s when reading "E" replaces the "E" with a blank and adds an "E" to the right of the new blank. After doing so, M^{∞} positions its reader on the newly created blank to the left of "E" and goes back to state s.

Step 3

Whenever M^{∞^2} would execute a command line that would cause it to halt, M^{∞} instead initiates a subroutine that deletes its markers, returns to its former position and halts.

Submit

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Week 9 / Numbering Turing Machines

Add a Post

Show all posts by recent activity

There are no posts in this topic yet.