Turing Machines

Now let us look at a sample computation:

The previous Turing machine computes (but not strictly) the function $f(x) = x + 2$, where we are using unary (base 1) notation.

The steps of the computation, given by

- the state of the machine,
- the string of symbols on the tape, and
- the square on the tape currently being scanned

are called \textit{configurations}.

Another Turing Machine Example

\textit{Alphabet A = \{a, b\}}
Another Turing Machine Example

May 9, 2019
Another Turing Machine Example

May 9, 2019  Theory of Computation  Lecture 24: Turing Machines IV

13

Another Turing Machine Example

May 9, 2019  Theory of Computation  Lecture 24: Turing Machines IV

14

Another Turing Machine Example

May 9, 2019  Theory of Computation  Lecture 24: Turing Machines IV

15

Another Turing Machine Example

May 9, 2019  Theory of Computation  Lecture 24: Turing Machines IV

16

Another Turing Machine Example

May 9, 2019  Theory of Computation  Lecture 24: Turing Machines IV

17

Another Turing Machine Example

May 9, 2019  Theory of Computation  Lecture 24: Turing Machines IV

18
Another Turing Machine Example

May 9, 2019

Theory of Computation
Lecture 24: Turing Machines IV
Another Turing Machine Example

May 9, 2019

Turing Machines

Theorem 1.1:
Any partial function that can be computed by a Post-Turing program can be computed by a Turing machine using the same alphabet.

Proof:
Let $P$ be a given Post-Turing program consisting of the instructions $I_1, \ldots, I_k$.

Let $s_0, s_1, \ldots, s_n$ be a list that includes all of the symbols mentioned in $P$.

We will construct a Turing machine $M$ that simulates $P$.

Our idea is that $M$ will be in state $q_i$ precisely when $P$ is about to execute instruction $I_i$.

So if $I_i$ is PRINT $s_k$, then we place in $M$ all of the following quadruples:

$q_i \quad s_j \quad s_k \quad q_{i+1}$, $j = 0, 1, \ldots, n$.

If $I_i$ is RIGHT, then we place in $M$ all of the following quadruples:

$q_i \quad s_j \quad R \quad q_{i+1}$, $j = 0, 1, \ldots, n$.

If $I_i$ is LEFT, then we place in $M$ all of the following quadruples:

$q_i \quad s_j \quad L \quad q_{i+1}$, $j = 0, 1, \ldots, n$.

Obviously, the actions of $M$ correspond precisely to the instructions of $P$, so our proof is complete.
Turing Machines

Do you remember Corollary 6.4 of Chapter 5?
"Every partially computable function is computed strictly by some Post-Turing program that uses only the symbols $s_0$ and $s_1$.

Given this corollary and the proof we just completed, we can derive the following:

**Theorem 1.2:** Let $f$ be an $m$-ary partially computable function on $A^*$ for a given alphabet $A$. Then there is a Turing machine $\mathcal{M}$ that computes $f$ strictly.

It is interesting to see the application of Theorem 1.2 to the case $A = \{1\}$.

Thus, if $f(x_1, \ldots, x_m)$ is any partially computable function on $N$, then there is a Turing machine that computes $f$ using only the symbols $B$ and 1.

The initial configuration corresponding to inputs $x_1, \ldots, x_m$ is

$B \ 1[x_1] \ B \ldots \ B \ 1[x_m]$

If $f(x_1, \ldots, x_m)$ is

$B \ 1[f(x_1, \ldots, x_m)]$

and the final configuration if $f(x_1, \ldots, x_m)$ is

$B \ 1[f(x_1, \ldots, x_m)]$

is

$\mathcal{M} \quad q_{K+1}$

Now we will take a look at Turing machines that consist of quintuples instead of quadruples.

There are two kinds of quintuples:

1. $q_i \ s_j \ s_k \ R \ q_l$
2. $q_i \ s_j \ s_k \ L \ q_l$

So you see that in either case the head prints the symbol $s_k$, which of course can be the same as the current symbol $s_j$.

Then a quintuple of type 1 makes the head move one step to the right, while type 2 makes it move one step to the left.

A quintuple Turing machine is a finite set of quintuples, no two of which begin with the same pair $q_i, s_j$.

**Theorem 1.3:** Any partial function that can be computed by a Turing machine can be computed by a quintuple Turing machine using the same alphabet.

**Proof:** Let $\mathcal{M}$ be a Turing machine with states $q_1, \ldots, q_K$ and alphabet $\{s_1, \ldots, s_n\}$.

We construct a quintuple Turing machine $\mathcal{M}'$ to simulate $\mathcal{M}$.

The states of $\mathcal{M}'$ are $q_{1,1}, \ldots, q_{2K}$.

So for each quadruple of $\mathcal{M}$ of the form $q_i \ s_j \ s_k \ R \ q_l$, we place in $\mathcal{M}'$ the corresponding quintuple $q_i \ s_j \ s_k \ R \ q_l$.

Similarly, for each quadruple of $\mathcal{M}$ of the form $q_i \ s_j \ L \ q_l$, we place in $\mathcal{M}'$ the corresponding quintuple $q_i \ s_j \ s_k \ L \ q_l$.

The third type of quadruple, $q_i \ s_j \ s_k \ R \ q_l$, is harder to simulate, because the quintuples always demand a move to either the right or the left.

Finally, we place in $\mathcal{M}'$ all quintuples of the form $q_{K+i} \ s_j \ s_k \ L \ q_l$, $i = 1, \ldots, K$; $j = 0, \ldots, n$.

The extra states $q_{K+1}, \ldots, q_{2K}$ are used to "remember" that we have gone one square too far to the right.
Turing Machines

**Theorem 1.4:** Any partial function that can be computed by a quintuple Turing machine can be computed by a Post-Turing program using the same alphabet.

This theorem, together with Theorems 1.1, 1.3, and 1.4 in this chapter, gives us the following corollary:

**Corollary 1.5:** For a given partial function f, the following are equivalent:
1. f can be computed by a Post-Turing program,
2. f can be computed by a Turing machine,
3. f can be computed by a quintuple Turing machine.

**Proof of Theorem 1.4:**
Let $M$ be a given quintuple Turing machine with states $q_1, \ldots, q_K$ and alphabet $\{s_1, \ldots, s_n\}$. We associate with each state $q_i$ a label $A_i$ and with each pair $q_i, s_j$ a label $B_{ij}$.

Each label $A_i$ is to be placed next to the first instruction in the following filter:

$[A_i] \text{ IF } s_0 \text{ GOTO } B_{i0}$
$\quad \text{ IF } s_1 \text{ GOTO } B_{i1}$
$\quad \vdots$
$\quad \text{ IF } s_n \text{ GOTO } B_{in}$

If $M$ contains the quintuple $q_i \ s_j \ s_k \ R \ q_l$, then we introduce the following block of instructions:

$[B_{ij}] \text{ PRINT } s_k$
$\quad \text{ RIGHT}$
$\quad \text{ GOTO } A_i$

Similarly, for the quintuple $q_i \ s_j \ s_k \ L \ q_l$ we introduce

$[B_{ij}] \text{ PRINT } s_k$
$\quad \text{ LEFT}$
$\quad \text{ GOTO } A_i$

Finally, if there is no quintuple in $M$ beginning with $q_i, s_j$, then we introduce the following block:

$[B_{ij}] \text{ GOTO } E$

Now we can easily construct a Post-Turing program that simulates $M$ simply by putting all of these blocks and filters one under the other. The order of placement only matters for the filter labeled $A_i$, which must begin the program.

The next slide will show the entire simulation program.

**A Universal Turing Machine**

Do you remember the **universal program** from Section 4.3?

It computes the output of another program when it is given that program’s number and the input to that program.

We defined $\phi(x, z)$ to be the unary partial function computed by the program with number $z$ for the input $x$. Now let $M$ be a Turing machine that computes this function with alphabet $\{1\}$.

Then let $g(x)$ be any partially computable function of one variable, and let $z_g$ be the number of some program in the language $\mathcal{L}$ that computes $g$. 
If we begin with the configuration
\[ B \times B z_0 \]
\[ \uparrow \]
\[ q_1 \]
where \( x \) and \( z_0 \) are written in unary notation, that is, as blocks of ones, then \( M \) will compute \( \Phi(x, z_0) = g(x) \).
(While unary notation is not very practical, it is intuitive and avoids the consideration of different bases and converting representations between them.) Thus, \( M \) can be used to compute any partially computable function of one variable.

A Universal Turing Machine

The universal Turing Machine was Turing's original theoretical construction of a programmable computer.
It provides a model of an all-purpose computer, in which data and programs are stored together in a single memory.
In the year 1936, this achievement showed that such an all-purpose computer would be possible.
It led to the anticipation of the modern digital computer.

Overview of Simulations

Our simulations show that the computational power of all five languages is identical.
In other words, these languages can compute exactly the same functions.
This result supports Church’s Thesis – any function that can be computed in principle can be computed by a Turing machine.

THE END