THEORY OF COMPUTATION
Turing machines - 19

Prof. Dan A. Simovici

UMB

Alan Mathison Turing (23 June 1912–7 June 1954) was an English mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist.
Turing was highly influential in the development of theoretical computer science, providing a formalisation of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general purpose computer.
During the Second World War, Turing worked for the Government Code and Cypher School at Bletchley Park, Britain's codebreaking centre.

Emile Post (11 February 1897–21 April, 1954) was a Polish born American mathematician and logician. He is best known for his work in the field that eventually became known as computability theory.
In 1936, Post developed (independently of Alan Turing) a mathematical model of computation that was essentially equivalent to the Turing machine model. This model is sometimes called Post's machine or a Post-Turing machine. Post's rewrite technique is now ubiquitous in programming language specification and design, and so with Church's lambda calculus is a salient influence of classical modern logic on practical computing.

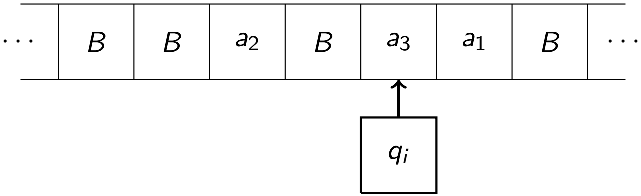
A *Turing machine* (TM) consists of a tape and a memory device that is capable of various internal states.
The current internal state plus the symbol on the square currently scanned determine two things:

1. the next state, and
2. printing a symbol on the current cell, or moving one square at the right or left.

Use of symbols in TMs:

- $q_1, q_2, \ldots$ represent states;
- $s_0, s_1, s_2, \ldots$ are symbols that appear on the tape, where s_0 is the blank.

- All but a finite number of cells contain B. The content of the tape is shown by exhibiting a finite portion of the tape containing the non-blank symbols.
- At any given moment only one tape symbol is being scanned by a head. The fact that the machine is in state q_i is indicated by an arrow and the symbol q_i as shown in the next slide.
- The head can move one square to the left or to the right of the square that is currently scanned.



This is indicated by writing

$$\begin{array}{cccc} a_2 & B & a_3 & a_1 \\ & & \uparrow & \\ & & q_i & \end{array}$$

A quadruple is an expression of one of the following forms that consist of four symbols:

- $q_i s_j s_k q_\ell$ (in state q_i scanning symbol s_j the device will print s_k and go into state q_ℓ);
- $q_i s_j R q_\ell$ (in state q_i scanning symbol s_j the device will move one square to the right and go into state q_ℓ);
- $q_i s_j L q_\ell$ (in state q_i scanning symbol s_j the device will move one square to the left and go into state q_ℓ).

Definition

A deterministic Turing machine is a finite set of quadruples, no two of which begin the same two symbols q_i and s_j. If this condition is not satisfied we have a non-deterministic Turing machine.

Unless stated otherwise, we will use deterministic Turing machines, referred to as Turing machines.
Deterministic Turing machines are capable of only one action at any given moment: writing a new symbol on the tape or moving the head left or right.

Workings of a TM:

- a TM always begins in the state q_1;
- a TM will halt if it is in the state q_i scanning s_j and there is no quadruple of the machine that begins with $q_i s_j$;
- a TM $\mathcal{M}$ computes a function $f(x_1, \ldots, x_m)$ in the same way as a Post-Turing program computes a function.

Definition

A TM $\mathcal{M}$ computes strictly a function on A^* if:

- the alphabet of $\mathcal{M}$ is a subset of A;
- the initial configuration is $\underset{\substack{\uparrow \\ q_0}}{B}x$;
- whenever $\mathcal{M}$ halts, the final configuration is $\underset{\substack{\uparrow \\ q_i}}{B}y$ where $y = f(x)$ contains no blanks.

Example

With $s_0 = B$ and $s_1 = 1$ consider the TM with alphabet $\{1\}$:

$$q_1 \; B \; R \; q_2$$
$$q_2 \; 1 \; R \; q_2$$
$$q_2 \; B \; 1 \; q_3$$
$$q_3 \; 1 \; R \; q_3$$
$$q_3 \; B \; 1 \; q_1$$

Example cont'd

The same machine can be presented in tabular form:

Symbol	*State*		
	q_1	q_2	q_3
B	Rq_2	$1q_3$	$1q_1$
1		Rq_2	Rq_3

In this TM we have the computation:

$$\underset{\substack{\uparrow \\ q_1}}{B}111 \qquad B\,\underset{\substack{\uparrow \\ q_2}}{1}\,11 \qquad \cdots \qquad B111\,\underset{\substack{\uparrow \\ q_2}}{B}$$

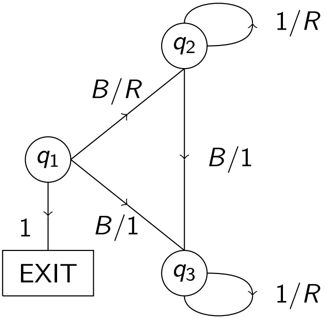
$$B111\,\underset{\substack{\uparrow \\ q_3}}{1} \qquad B1111\,\underset{\substack{\uparrow \\ q_3}}{B} \qquad B1111\,\underset{\substack{\uparrow \\ q_1}}{1}$$

Note that

- the computation halts because there is no quadruple beginning with $q_1 1$;
- the TM computes (but not strictly the function $f(x) = x + 2$ using the base 1 notation.

The steps of the computation are known as configurations.

An alternative representation of a TM is a state transition diagram.



	State		
Symbol	q_1	q_2	q_3
B	Rq_2	$1q_3$	$1q_1$
1		Rq_2	Rq_3

Theorem

Any partial function that can be computed by a Post-Turing program can be computed by a TM using the same alphabet.

Proof

Let $\mathcal{P}$ be a Post-Turing program that consists of instructions $I_1, \ldots, I_K$ and let $s_0, s_1, \ldots, s_n$ be a list that includes all symbols mentioned in $\mathcal{P}$.
The proof consists in constructing a TM $\mathcal{M}$ that simulates $\mathcal{P}$.
$\mathcal{M}$ is in the state q_i when $\mathcal{P}$ is about to execute instruction I_i.

- if I_i is PRINT s_k we place in $\mathcal{M}$ all of the quadruples $q_i\ s_j\ s_k\ q_{i+1}$ for $0 \leqslant j \leqslant n$;
- if I_i is RIGHT we place in $\mathcal{M}$ all of the quadruples $q_i\ s_j\ R\ q_{i+1}$ for $0 \leqslant j \leqslant n$;
- if I_i is LEFT we place in $\mathcal{M}$ all of the quadruples $q_i\ s_j\ L\ q_{i+1}$ for $0 \leqslant j \leqslant n$;

- if I_i is IF s_k GOTO L let m be the least number such that I_m is labeled L if there is an instruction in $\mathcal{P}$ labeled L; otherwise let $m = K + 1$; place in $\mathcal{M}$ the quadruple:

$$q_i \; s_k \; s_k \; q_m$$

as well as the quadruples:

$$q_i \; s_j \; s_j \; q_{i+1}$$

for $j \in \{0, 1, \ldots, n\} - \{k\}$.

The actions of $\mathcal{M}$ correspond to the instructions of $\mathcal{P}$, which concludes the proof.

Theorem

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

A Special Case: Let $A = \{1\}$. If $f(x_1, \dots, x_m)$ is a partially computable function on $\mathbb{N}$, there is a TM that computes f using only the symbols B and 1. The initial configuration corresponds to inputs $x_1, \dots, x_m$ is

$$\underset{\substack{\uparrow \\ q_1}}{B}\, 1^{x_1}\ B\ \cdots\ B\ 1^{x_m}$$

and the final configuration when $f(x_1, \dots, x_m)\downarrow$ is

$$\underset{\substack{\uparrow \\ q_{K+1}}}{B}\quad 1^{f(x_1,\dots,x_m)}$$

We consider now Turing machines specified by quintuples instead of quadruples.

Definition

A quintuple Turing machine $\mathcal{M}$ consists of a finite set of quintuples that have one of two forms:

$$q_i \; s_j \; s_k \; R \; q_\ell$$
$$q_i \; s_j \; s_k \; L \; q_\ell,$$

such that no two quintuples begin with the same pair $q_i \; s_j$.

The first (second) quintuple means that when the machine is in state q_i scanning s_j it will print s_k and then move one square to the right (left) and go into the state q_ℓ.

Theorem

Any partial function that can be computed by a TM can be computed by a quintuple machine using the same alphabet.

Proof

Let $\mathcal{M}$ be a TM with states $q_1, \ldots, q_K$ and alphabet $\{s_1, \ldots, s_n\}$. We construct a quintuple TM $\overline{\mathcal{M}}$ to simulate $\mathcal{M}$. The states of $\overline{\mathcal{M}}$ are $q_1, \ldots, q_K, q_{K+1}, \ldots, q_{2K}$.

- For each quadruple of $\mathcal{M}$ of the form

$$q_i \; s_j \; R \; q_\ell$$

  we place in $\overline{\mathcal{M}}$ the quintuple

$$q_i \; s_j \; s_j \; R \; q_\ell.$$

- Similarly, for each quadruple of $\mathcal{M}$ of the form

$$q_i \; s_j \; L \; q_\ell$$

  we place in $\overline{\mathcal{M}}$ the quintuple

$$q_i \; s_j \; s_j \; L \; q_\ell$$

Proof cont'd

For each quadruple

$$q_i \; s_j \; s_k \; q_\ell$$

in $\mathcal{M}$ we place in $\overline{\mathcal{M}}$ all quintuples of the form

$$q_i \; s_j \; s_k \; R \; q_{K+\ell}.$$

Finally, we place in $\overline{\mathcal{M}}$ all quintuples of the form

$$q_{K+\ell} \; s_j \; s_j \; L \; q_\ell.$$

- Quadruples requiring motion are simulated easily by quintuples.
- However, a quadruple that requires a print requires using a quintuple which causes a motion after the print has taken place. The final list of quintuples undoes the effect of the unwanted motion. The extra states $q_{K+1}, \ldots, q_{2K}$ serve to remember that we have gone a square too far to the left.

Theorem

Any partial function that can be computed by a quintuple TM can be computed by a Post-Turing program using the same alphabet.

Proof:

Let $\mathcal{M}$ be a quintuple TM 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 label $q_i s_j$ a label B_{ij}. Each label A_i is to be placed next to the first instruction in:

$$\begin{array}{ll} [A_i] & \text{IF } s_0 \text{ GOTO } B_{i0} \\ & \text{IF } s_1 \text{ GOTO } B_{i1} \\ & \vdots \\ & \text{IF } s_n \text{ GOTO } B_{in} \end{array}$$

If $\mathcal{M}$ contains the quintuple

$$q_i\ s_j\ s_k\ R\ q_\ell$$

we introduce the block

$$\begin{array}{ll} [B_{ij}] & \text{PRINT}\, s_k \\ & \text{RIGHT} \\ & \ \text{GOTO } A_\ell \end{array}$$

Similarly, if $\mathcal{M}$ contains the quintuple

$$q_i\ s_j\ s_k\ L\ q_\ell$$

we introduce the block

$$\begin{array}{ll} [B_{ij}] & \text{PRINT}\, s_k \\ & \text{LEFT} \\ & \ \text{GOTO } A_\ell \end{array}$$

Finally, of there is no quintuple in $\mathcal{M}$ beginning with $q_i s_j$ we introduce the block

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

Concatenating all the above blocks results in a Post-Turing program that simulates $\mathcal{M}$. The order of the blocks is irrelevant except that the block labeled A_1 must begin the program. The full program is listed on the next slide.

$$
\begin{array}{ll}
[A_1] & \text{IF } s_0 \text{ GOTO } B_{10} \\
 & \text{IF } s_1 \text{ GOTO } B_{11} \\
 & \vdots \\
 & \text{IF } s_n \text{ GOTO } B_{1n} \\
 & \vdots \\
[A_K] & \text{IF } s_0 \text{ GOTO } B_{K0} \\
 & \text{IF } s_1 \text{ GOTO } B_{K1} \\
 & \vdots \\
 & \text{IF } s_n \text{ GOTO } B_{Kn} \\
[B_{i_1 j_1}] & \text{PRINT} s_{k_1} \\
 & \text{LEFT} \\
 & \text{ GOTO } A_{\ell_1} \\
 & \vdots \\
[B_{i_2 j_2}] & \text{PRINT} s_{k_2} \\
 & \text{LEFT} \\
 & \text{ GOTO } A_{\ell_2} \\
 & \vdots
\end{array}
$$

Corollary

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.*