Computability*

Xiaofeng Gao

Department of Computer Science and Engineering
Shanghai Jiao Tong University, P. R. China

CSC101-Introduction to Computer Science

*This lecture note is arranged according to Prof. John Hopcroft's Introduction to Computer Science course at SJTU.

Outline

Cardinality of Sets

For finite sets, it is easy to compare their cardinality.

- E.g., $|\{1, 2, 3\}| > |\{a, b\}|$.

For infinite set, we need to match them pairwise.

- If A can map 'one-to-one' into B, then $A \preceq B$.
- If A can map 'onto' into B, then $A \succeq B$.
- If A can map 'one-to-one' and 'onto' into B, then $A \approx B$.

Schröder-Bernstein Theorem: $A \preceq B$, $A \succeq B \Rightarrow A \approx B$.

Proof. It is done with mirrors.

We are given one-to-one functions $f : A \to B$ and $g : B \to A$. Define C_n by recursion, using the formulas

$$C_0 = A - ran(g) \qquad \text{and} \qquad C_{n^+} = g[f[C_n]].$$

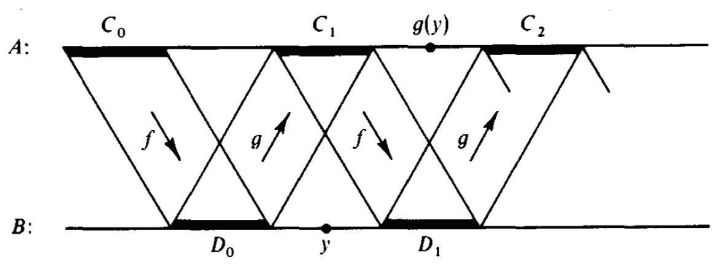
Thus C_0 is the troublesome part that keeps g from being an one-to-one correspondence between B and A. We bounce it back and forth, obtaining $C_1, C_2, \cdots$. The function showing that $A \approx B$ is the function h: $A \to B$ defined by

$$h(x) = \begin{cases} f(x) & \text{if } x \in C_n \text{ for some } n, \\ g^{-1}(x) & \text{otherwise.} \end{cases}$$

Schröder-Bernstein Theorem (Cont.)

Note that in the second case ($x \in A$ but $x \notin C_n$ for any n) it follows that $x \notin C_0$ and hence $x \in ran(g)$.

Thus $g^{-1}(x)$ makes sense in this case.



Schröder-Bernstein Theorem (Cont.)

Does it work? We must verify that h is one-to-one and has range B.

Define $D_n = f[C_n]$, so that $C_{n^+} = g[D_n]$.

To show that h is one-to-one, consider distinct x and x' in A.

Since both f and g^{-1} are one-to-one, the only possible problem arises when, say, $x \in C_m$ and $x' \notin \cup_{n\in\omega} C_n$.

In this case, $h(x) = f(x) \in D_m$, whereas $h(x) = g^{-1}(x) \notin D_m$, lest $x' \in C_{m^+}$. So $h(x) \neq h(x')$.

Schröder-Bernstein Theorem (Cont.)

Finally we must check that $ran(h)$ exhausts B.

Certainly each $D_n \subseteq ran(h)$, because $D_n = h[C_n]$. Consider then a point y in $B - \cup_{n\in\omega} D_n$.

Where is $g(y)$? Certainly $g(y) \notin C_0$. Also $g(y) \notin C_{n^+}$, because $C_{n^+} = g[D_n]$, $y \notin D_n$, and g is one-to-one.

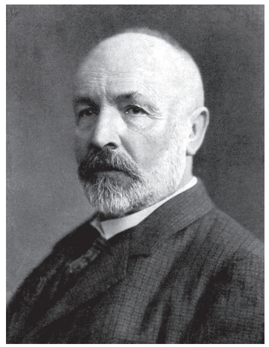
So $g(y) \notin C_n$ for any n. Therefore $h(g(y)) = g^{-1}(g(y)) = y$. This shows that $y \in ran(h)$, thereby proving it. $\square$

Cantor's Diagonal Argument

In set theory, Cantor's diagonal argument, also called the diagonalisation argument, the diagonal slash argument or the diagonal method, was published in 1891 by Georg Cantor.

It was proposed as a mathematical proof for uncountable sets.

It demonstrates a powerful and general technique that has been used in a wide range of proofs.



Georg Cantor
1845-1918

Cantor's Diagonal Method

Assumption: If $\{s_1, s_2, \cdots, s_n, \cdots\}$ is any enumeration of elements from T, then there is always an element s of T which corresponds to no s_n in the enumeration.

Diagonal Method: Construct the sequence s by choosing the 1^{st} digit as complementary to the 1^{st} digit of s_1 , the 2^{nd} digit as complementary to the 2^{nd} digit of s_2, and generally for every n, the n^{th} digit as complementary to the n^{th} digit of s_n.

By construction, s differs from each s_n, since their n^{th} digits differ (highlighted in the example). Hence, s cannot occur in the enumeration.

$$\begin{array}{ll}
s_1 & = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,\ldots \\
s_2 & = 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,\ldots \\
s_3 & = 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,\ldots \\
s_4 & = 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,\ldots \\
s_5 & = 1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,1\,\ldots \\
s_6 & = 0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\,\ldots \\
s_7 & = 1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,\ldots \\
s_8 & = 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,\ldots \\
s_9 & = 1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,\ldots \\
s_{10} & = 1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\,\ldots \\
s_{11} & = 1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\,\ldots \\
\vdots & \quad \vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\vdots\,\ddots
\end{array}$$

$$s = 1\,0\,1\,1\,1\,0\,1\,0\,0\,1\,1\,\ldots$$

Cantor Diagonal Method

Based on this theorem, Cantor then uses a proof by contradiction to show that:

The set T is uncountable.

Proof. He assumes for contradiction that T was countable. Then all its elements could be written as an enumeration $s_1, s_2, \cdots, s_n, \cdots$. Applying the previous theorem to this enumeration would produce a sequence s not belonging to the enumeration.

However, s was an element of T and should therefore be in the enumeration. This contradicts the original assumption, so T must be uncountable.

What is Effective Procedure

- Methods for addition, multiplication $\cdots$
  - ▷ Given n, finding the nth prime number.
  - ▷ Differentiating a polynomial.
  - ▷ Finding the highest common factor of two numbers $HCF(x, y) \to$ Euclidean algorithm
  - ▷ Given two numbers x, y, deciding whether x is a multiple of y.
- Their implementation requires no ingenuity, intelligence, inventiveness.

Intuitive Definition

An *algorithm* or *effective procedure* is a mechanical rule, or automatic method, or programme for performing some mathematical operations.

Blackbox: input $\longrightarrow \blacksquare \longrightarrow$ output

What is "effective procedure"?

An Example: Consider the function $g(n)$ defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive } 7'\text{s} \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

Question: Is $g(n)$ effective?

▷ The answer is unknown $\neq$ the answer is negative.

Other Examples:

- *Theorem Proving* is in general not effective/algorithmic.
- *Proof Verification* is effective/algorithmic.

Algorithm and Computable Function

Algorithm: An algorithm is a procedure that consists of a finite set of *instructions* which, given an *input* from some set of possible inputs, enables us to obtain an *output* through a systematic execution of the instructions that *terminates* in a finite number of steps.

Computable Function: When an algorithm or effective procedure is used to calculate the value of a numerical function then the function in question is effectively calculable (or algorithmically computable, effectively computable, computable).

Partial and Total Function

- n-ary function: $f(x_1, \cdots, x_n), f : \mathbb{N}^n \to \mathbb{N}$.
- Partial function: $dom(f)$ is not necessarily the whole $\mathbb{N}^n$. (In our class function means partial function)
- Total function: $dom(f) = \mathbb{N}^n$.

The definition of unary function is similar.

Computable Functions

Theorem: The computable functions are countable (enumerable).

Proof: By Gödel Coding technique (will not covered here).

Thus, we can enumerate all computable functions as a sequence $\mathscr{C} = \{\phi_0, \phi_1, \cdots\}$.

Uncomputable Function

Theorem. There is a total unary function that is not computable.

Proof. Suppose $\phi_0, \phi_1, \phi_2, \ldots$ is an enumeration of $\mathscr{C}_1$. Define

$$f(n) = \begin{cases} \phi_n(n) + 1, & \text{if } \phi_n(n) \text{ is defined,} \\ 0, & \text{if } \phi_n(n) \text{ is undefined.} \end{cases}$$

The function $f(n)$ is not computable.

Example of uncomputable function

Consider again the construction of f to construct a total uncomputable function. Complete details of the functions $\phi_0, \phi_1, \cdots$ can be represented by the following infinite table:

	0	1	2	3	4
ϕ_0	$\phi_0(0)$	$\phi_0(1)$	$\phi_0(2)$	$\phi_0(3)$	...
ϕ_1	$\phi_1(0)$	$\phi_1(1)$	$\phi_1(2)$	$\phi_1(3)$	...
ϕ_2	$\phi_2(0)$	$\phi_2(1)$	$\phi_2(2)$	$\phi_2(3)$	...
ϕ_3	$\phi_3(0)$	$\phi_3(1)$	$\phi_3(2)$	$\phi_3(3)$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Diagonal Method

We suppose that in this table the word ‘undefined’ is written whenever $\phi_n(m)$ is not defined.

The function f was constructed by taking the diagonal entries on the table $\phi_0(0), \phi_1(1), \phi_2(2), \cdots$ and systematically changing them, obtaining $f(0), f(1), \cdots$ such that $f(n)$ differs from $\phi_n(n)$, for each n.

Note that there was considerable freedom in choosing the value of $f(n)$ (just differ from $\phi_n(n)$). Thus

$$g(n) = \begin{cases} \phi_n(n) + 27^n & \text{if } \phi_n(n) \text{ is defined,} \\ n^2 & \text{if } \phi_n(n) \text{ is undefined,} \end{cases}$$

is another non-computable total function.

Halting Problem

Now we define a function:
halt(*computer program*, *input to computer program*)

$$halt(prog, x) = \begin{cases} yes & \text{if } prog \text{ halts on input } x, \\ no & \text{if } prog \text{ does not halt on input } x. \end{cases}$$

Computable function

Theorem: *halt* is uncomputable.

Proof. Assume *halt* is computable, then we can compute the uncomputable $f(i)$ (mentioned above) as follow.

First compute $\phi(i) = halt(prog, i)$,

$$\begin{cases} \text{If } halt(prog, i) = no, & \text{output } 0, \\ \text{If } halt(prog, i) = yes, & \text{simulate program with } i \text{ and add 1 to answer,} \end{cases}$$

So it is impossible. Thus *halt* is uncomputable.

Basic Concepts

Let $\Sigma = \{a_1, \ldots, a_k\}$ be the set of symbols, called alphabet.

A string (word) from Σ is a sequence $a_{i_1}, \cdots, a_{i_n}$ of symbols from Σ.

Σ^* is the set of all words/strings from Σ. (Kleene Star)

ε is the empty string, that has no symbols.

Regular Expression (Type-3 Language)

Production Rules:
- ◦ $A \to a$;
- ◦ $A \to aB$;
- ◦ $A \to \varepsilon$;

Type-3 grammar can be constructed based on Nondeterministic Finite Automata (NFA).

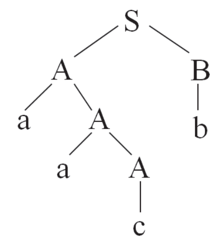
Context-Free Language (Type-2 Language)

Definition:
◦ T is alphabet subset (terminals)
◦ N is arbitrary string set (non-terminals, S is starting symbol);
◦ $Q = \{A \to \beta, \beta \in (N \cup T)^+ \cup \{\Lambda\}\}$ (production set);

Example: $T = \{a, b, c\}$; $N = \{S, A, B\}$,
$Q = \{S \to AB;\ B \to b;\ A \to aA|c\}$

Derivation:
$S \Rightarrow AB \Rightarrow aAB \Rightarrow aAb \Rightarrow aaAb \Rightarrow aacb$

Parse Tree:
◦ A tree representing a derivation;
◦ All internal nodes are non-terminals;
◦ All leave nodes are terminals;
◦ Build a tree following derivation.



Context-Sensitive Language (Type-1 Language)

Definition:
∘ T alphabet subset (terminals)
∘ N arbitrary string set (non-terminals, S is starting symbol);
∘ $Q = \{\alpha A \beta \to \alpha \gamma \beta\}$

- ▷ Replace A by γ only if found in the context of α and β;
- ▷ Left side does not have to be a single non-terminal;
- ▷ $\alpha, \beta \in (N \cup T)^*$;
- ▷ $\gamma \in (N \cup T)^* - \Lambda$.

Also Q includes all possible rules in type-2 grammar.

Corresponds to recursive language

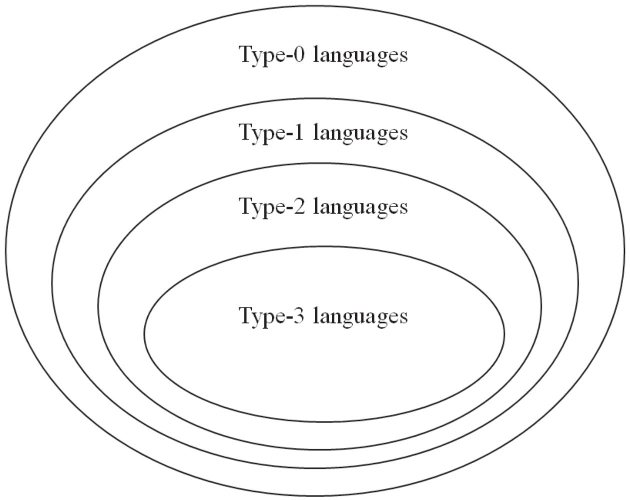
Recursively Enumerable Language (Type-0 Language)

Production Rules:
∘ Include all possible forms for the rules Type-3 to Type-1;
∘ Allow rules of the form: $\alpha \to \beta$

- ▷ $\alpha \in (N \cup T)^* N (N \cup T)^*$; (At least one non-terminal)
- ▷ $\beta \in (N \cup T)^*$.

Type-0 language includes all languages that are recognizable by Tuning machine.

Chomsky Schützenberger Hierarchy

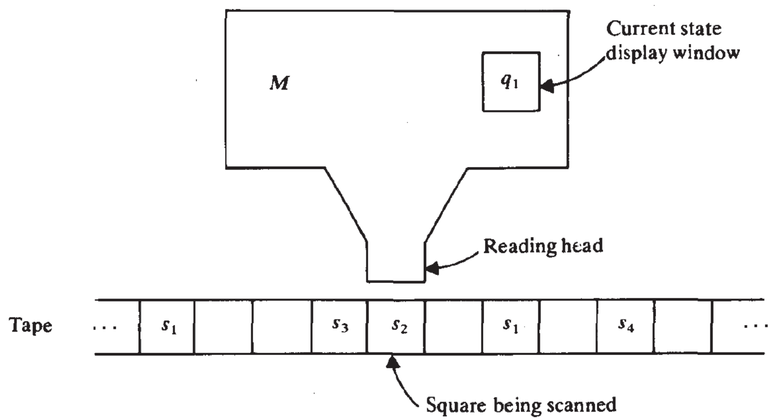


One-Tape Turing Machine

A Turing machine has five components:

1. A finite set $\{s_1, \ldots, s_n\} \cup \{\triangleright, \sharp, \triangleleft\} \cup \{\square\}$ of symbols.
2. A tape consists of an infinite number of cells, each cell may store a symbol.
$$\cdots \square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square \cdots$$
3. A reading head that scans and writes on the cells.
4. A finite set $\{q_S, q_1, \ldots, q_m, q_H\}$ of states.
5. A finite set of instructions (specification).

One-Tape Turing Machine



Turing Machines, Turing 1936

The input data

$$\rhd s_1^1 \dots s_{i_1}^1 \Box \dots \Box s_1^k \dots s_{i_k}^k \lhd \Box \cdots$$

The reading head may write a symbol, move left, move right.

An instruction is of the form:

$$\langle q_i, s_j \rangle \to \langle q_l, s_k, L \rangle,$$

which means when reads s_j with state q_i, the machine will turn to state q_l, replace s_j with s_k, and turn one cell to the left.

The direction can be L, R, or S, meaning move to left, right, or stay at the current position.

An Example

Given a Turing machine M with the alphabet $\{0, 1\} \cup \{\rhd, \Box, \lhd\}$.

$$\begin{aligned}
\langle q_S, \rhd \rangle &\to \langle q_1, \rhd, R \rangle \\
\langle q_1, 0 \rangle &\to \langle q_1, 0, R \rangle \\
\langle q_1, 1 \rangle &\to \langle q_2, 0, S \rangle \\
\langle q_2, 0 \rangle &\to \langle q_2, 0, R \rangle \\
\langle q_2, 1 \rangle &\to \langle q_1, 1, R \rangle \\
\langle q_1, \lhd \rangle &\to \langle q_3, \lhd, L \rangle \\
\langle q_2, \lhd \rangle &\to \langle q_3, \lhd, L \rangle \\
\langle q_3, 0 \rangle &\to \langle q_3, 0, L \rangle \\
\langle q_3, 1 \rangle &\to \langle q_3, 1, L \rangle \\
\langle q_3, \rhd \rangle &\to \langle q_H, \rhd, R \rangle
\end{aligned}$$

▷	1	1	0	1	◁
$\widehat{q_S}$					

▷	**1**	1	0	1	◁
	$\widehat{q_1}$				

▷	**0**	1	0	1	◁
	$\widehat{q_2}$				

▷	0	**1**	0	1	◁
		$\widehat{q_2}$			

▷	0	1	**0**	1	◁
			$\widehat{q_1}$		

▷	0	1	0	**1**	◁
				$\widehat{q_1}$	

▷	0	1	0	**0**	◁
				$\widehat{q_2}$	

▷	0	1	0	0	**◁**
					$\widehat{q_2}$

▷	0	1	0	**0**	◁
				$\widehat{q_3}$	

▷	0	1	**0**	0	◁
			$\widehat{q_3}$		

▷	0	1	0	0	◁
$\widehat{q_3}$					

▷	**0**	1	0	0	◁
	$\widehat{q_H}$				

Multi-Tape Turing Machine

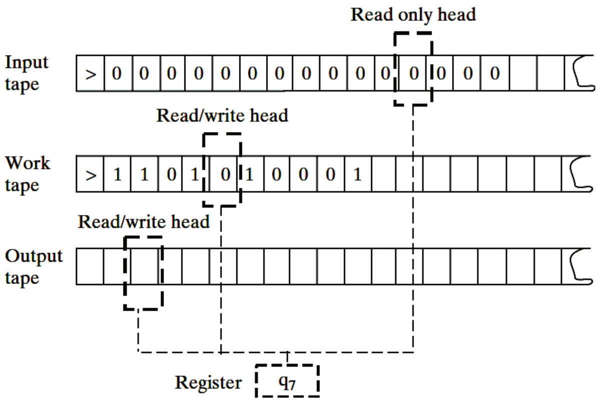
A multi-tape TM is described by a tuple (Γ, Q, δ) containing

- A finite set Γ called alphabet, of symbols. It contains a blank symbol $\Box$, a start symbol $\rhd$, and the digits 0 and 1.
- A finite set Q of states. It contains a start state q_{start} and a halting state q_{halt}.
- A transition function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times L, S, R^k$, describing the rules of each computation step.

Example: A 2-Tape TM will have transition function (also named as specification) like follows:

$$\begin{aligned}
\langle q_s, \rhd, \rhd \rangle &\quad\to\quad \langle q_1, \rhd, R, R \rangle \\
\langle q_1, 0, 1 \rangle &\quad\to\quad \langle q_2, 0, S, L \rangle
\end{aligned}$$

Computation and Configuration



Computation, configuration, initial/final configuration

A 3-Tape TM for the Palindrome Problem

A palindrome is a word that reads the same both forwards and backwards. For instance:

ada, anna, madam, and nitalarbralatin.

Requirement: Give the specification of M with $k = 3$ to recognize palindromes on symbol set $\{0, 1, \triangleright, \triangleleft, \Box\}$.

Preparation

To recognize palindrome we need to check the input string, output 1 if the string is a palindrome, and 0 otherwise.

Initially the input string is located on the first tape like "$\triangleright$ 0110001 $\triangleleft \Box\Box\Box \cdots$", strings on all other tapes are "$\triangleright\Box\Box\Box \cdots$".

The head on each tape points the first symbol "$\triangleright$" as the starting state, with state mark q_S.

In the final state q_F, the output of the k^{th} tape should be "$\triangleright$ 1 $\triangleleft \Box$" if the input is a palindrome, and "$\triangleright$ 0 $\triangleleft \Box$" otherwise.

A 3-Tape TM for the Palindrome Problem

$Q = \{q_s, q_h, q_c, q_l, q_t, q_r\}$; $\Gamma = \{\Box, \triangleright, \triangleleft, 0, 1\}$; two work tapes.

Start State:

$\langle q_s, \triangleright, \triangleright, \triangleright\rangle \to \langle q_c, \triangleright, \triangleright, R, R, R\rangle$

Begin to copy:

$\langle q_c, 0, \Box, \Box\rangle \to \langle q_c, 0, \Box, R, R, S\rangle$
$\langle q_c, 1, \Box, \Box\rangle \to \langle q_c, 1, \Box, R, R, S\rangle$
$\langle q_c, \triangleleft, \Box, \Box\rangle \to \langle q_l, \Box, \Box, L, S, S\rangle$

Return back to the leftmost:

$\langle q_l, 0, \Box, \Box\rangle \to \langle q_l, \Box, \Box, L, S, S\rangle$
$\langle q_l, 1, \Box, \Box\rangle \to \langle q_l, \Box, \Box, L, S, S\rangle$
$\langle q_l, \triangleright, \Box, \Box\rangle \to \langle q_t, \Box, \Box, R, L, S\rangle$

Begin to compare:

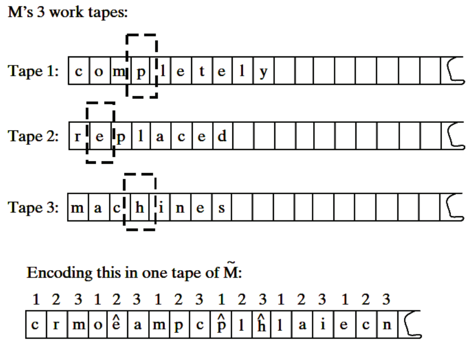
$\langle q_t, \triangleleft, \triangleright, \Box\rangle \to \langle q_r, \triangleright, 1, S, S, R\rangle$
$\langle q_t, 0, 1, \Box\rangle \to \langle q_r, 1, 0, S, S, R\rangle$
$\langle q_t, 1, 0, \Box\rangle \to \langle q_r, 0, 0, S, S, R\rangle$
$\langle q_t, 0, 0, \Box\rangle \to \langle q_t, 0, \Box, R, L, S\rangle$
$\langle q_t, 1, 1, \Box\rangle \to \langle q_t, 1, \Box, R, L, S\rangle$

Ready to terminate:

$\langle q_r, \triangleleft, \triangleright, \Box\rangle \to \langle q_h, \triangleright, \triangleleft, S, S, S\rangle$
$\langle q_r, 0, 1, \Box\rangle \to \langle q_h, 1, \triangleleft, S, S, S\rangle$
$\langle q_r, 1, 0, \Box\rangle \to \langle q_r, 0, \triangleleft, S, S, S\rangle$

Single-Tape vs. Multi-Tape

- The basic idea is to interleave k tapes into one tape.
- The first $n+1$ cells are reserved for the input.



- Every symbol a of M is turned into two symbols $a, \hat{a}$ in $\widetilde{M}$, with $\hat{a}$ used to indicate head position.

Cardinality
Cantor Diagonal Method
Halting Problem and Language
Turing Machine
One-Tape Turing Machine
Multi-Tape Turing Machine
Computing Capacity for Difference Turing Machines

Single-Tape vs. Multi-Tape

The outline of the algorithm:

The machine $\widetilde{M}$ places $\triangleright$ after the input string and then starts copying the input bits to the imaginary input tape. During this process whenever an input symbol is copied it is overwritten by $\triangleright$.

$\widetilde{M}$ marks the $n+2$-cell, …, the $n+k$-cell to indicate the initial head positions.

$\widetilde{M}$ Sweeps $kT(n)$ cells from the $(n+1)$-th cell to right, recording in the register the k symbols marked with the hat $\hat{\_}$.

$\widetilde{M}$ Sweeps $kT(n)$ cells from right to left to update using the transitions of M. Whenever it comes across a symbol with hat, it moves right k cells, and then moves left to update.

Cardinality
Cantor Diagonal Method
Halting Problem and Language
Turing Machine
One-Tape Turing Machine
Multi-Tape Turing Machine
Computing Capacity for Difference Turing Machines

Unidirectional Tape vs. Bidirectional Tape

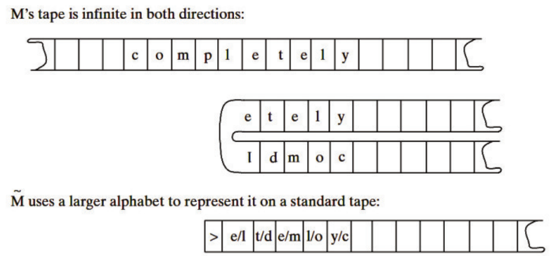
Define a bidirectional Turing Machine to be a TM whose tapes are infinite in both directions.

Fact: If $f : \{0,1\}^* \to \{0,1\}^*$ is computable in time $T(n)$ by a bidirectional TM M, then it is computable in time $4T(n)$ by a TM $\widetilde{M}$ with one-directional tape.

Cardinality
Cantor Diagonal Method
Halting Problem and Language
Turing Machine
One-Tape Turing Machine
Multi-Tape Turing Machine
Computing Capacity for Difference Turing Machines

Unidirectional Tape vs. Bidirectional Tape

- The idea is that $\widetilde{M}$ makes use of the alphabet $\Gamma \times \Gamma$.



- Every state q of M is turned into $\bar{q}$ and $\underline{q}$.

Unidirectional Tape vs. Bidirectional Tape

Let H range over $\{L, S, R\}$ and let $-H$ be defined by

$$-H = \begin{cases} R, & \text{if} \quad H = L, \\ S, & \text{if} \quad H = S, \\ L, & \text{if} \quad H = R. \end{cases}$$

$\widetilde{M}$ contains the following transitions:

$\langle \overline{q}, (\triangleright, \triangleright) \rangle \to \langle \underline{q}, (\triangleright, \triangleright), R \rangle$
$\langle \underline{q}, (\triangleright, \triangleright) \rangle \to \langle \overline{q}, (\triangleright, \triangleright), R \rangle$

$\langle \overline{q}, (a, b) \rangle \to \langle \overline{q'}, (a', b), H \rangle$ if $\langle q, a \rangle \to \langle q', a', H \rangle$
$\langle \underline{q}, (a, b) \rangle \to \langle \underline{q'}, (a, b'), -H \rangle$ if $\langle q, b \rangle \to \langle q', b', H \rangle$