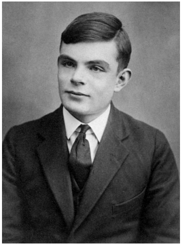
Lec 13. Turing Machine

Eunjung Kim

Turing Machine, Alan Turing 1936



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes

Model of Computation

Exercutor(machine) constituents:

- an alphabet Γ; it reads and writes a symbol in Γ,
- a finite set of states to perceive its status ("where am I?"),
- a memory as an infinite tape from which to read and write.
- a gadget (called a header) to read from and write on the tape.

Basic operation:

- read one symbol from the tape,
- update its internal state,
- move the header (only in one fixed direction, or both direction, or neither) on tape,
- write(change) a symbol on the tape.

Turing Machine, briefly

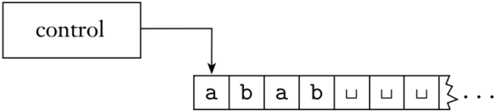


Figure 3.1, Sipser 2012.

- Read-write tape infinite to the right, which initially contains the input string and the rest filled with blank symbol ␣.
- Head on one cell in the tape, which moves left or right by one cell at a time.
- At the beginning, only the input string in the tape and the head is on the first cell.
- Each transition (a.k.a. move) of TM M does the following, depending on the current state.
  1. read one symbol from the current cell
  2. update its state,
  3. write a symbol at the current cell (where head is on),
  4. move the head left or right.

Formal definition of TM

A TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q a finite set called the states,
- Σ a finite set called the input alphabet,
- Γ a finite set called the (tape) alphabet, with $\Sigma \cup \{\sqcup\} \subseteq \Gamma$,
- δ a function from $Q \times \Gamma$ to $Q \times \Sigma \times \{L, R\}$ called a transition function,
- $q_0 \in Q$ the start state,
- $q_{accept} \in Q$ the accept state.
- $q_{reject} \in Q$ the reject state, with $q_{reject} \neq q_{accept}$.

FORMAL DEFINITION OF COMPUTATION

CONFIGURATION OF TM

A configuration of TM is a triple consisting of

1. a state q,
2. tape contents,
3. head location.

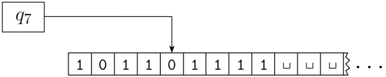


Figure 3.4, Sipser 2012.

- We represent a configuration as as $u\ q\ aw$, where q is the current state, $uaw \in \Sigma^*$ is the tape contents (except for right-infinite blanks), and q is written in the middle of the tape contents uaw, right before the head location.
- Sometimes, we write it $(q, u\underline{a}w)$ where q is the current state and the underbar below a indicates the head location.

Formal definition of computation

Yield

We say that a configuration C_1 yields a configuration C_2 if the Turing machine can go from C_1 and C_2 in a single step. That is,

$$ua\ q\ bv \text{ yields } u\ q\ acv$$

if $\delta(q, b) = (q', c, L)$ and

$$ua\ q\ bv \text{ yields } uac\ q\ v$$

if $\delta(q, b) = (q', c, R)$ and

Formal definition of computation

- A sequence $C_1, C_2, \ldots, C_\ell$ of configurations is a computation history of TM M if each C_i yields C_{i+1} for all $i = 1, \ldots, \ell - 1$.
- We write $C_1 \rightsquigarrow_M^* C_2$ for two configurations C_1, C_2 if there is a computation history which begins with C_1 and ends with C_2 (possibly $C_1 = C_2$).

Formal definition of computation

- A sequence $C_1, C_2, \ldots, C_\ell$ of configurations is a computation history of TM M if each C_i yields C_{i+1} for all $i = 1, \ldots, \ell - 1$.
- We write $C_1 \rightsquigarrow_M^* C_2$ for two configurations C_1, C_2 if there is a computation history which begins with C_1 and ends with C_2 (possibly $C_1 = C_2$).
- The start (initial) configuration on an input string w is $q_0\ w$
- A configuration $w'\ q\ w''$ is an accepting/rejecting/halting configuration if q equals the accept/reject/accept or reject state.

Formal definition of computation

The language of a Turing Machine

TM accepts an input string $w \in \Sigma^*$ if there is a computation history which starts with the start configuration $q_0\ w$ on w and ends with an accepting configuration.

The set of strings in Σ^* accepted by TM M is called the language of M, or the language recognized by M, and denoted as $L(M)$.

Recognizable & Decidable

Turing-recognizable; recursively enumerable

- M recognizes a language $A \subseteq \Sigma^*$ if $A = L(M)$.
  ($\rightsquigarrow$ M is not guaranteed to halt on all $w \in \Sigma^* \setminus A$. That is, M may loop on some input.)
- A language A is recursively enumerable if there is a Turing machine recognizing it.

Turing-decidable; recursive

- M decides a language $L \subseteq \Sigma^*$ if $A = L(M)$ AND M halts on every input $w \in \Sigma^*$.
- A language M is recursive if there is a Turing machine deciding it.

Example of TM: sameness

Consider the language $A = \{w\#w : w \in \{0,1\}^*\}$.

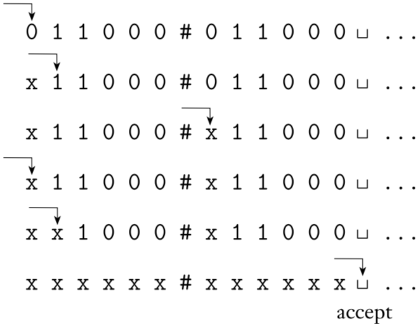


Figure 3.2, Sipser 2012.

Example of TM: sameness

Consider the language $A = \{w\#w : w \in \{0,1\}^*\}$.

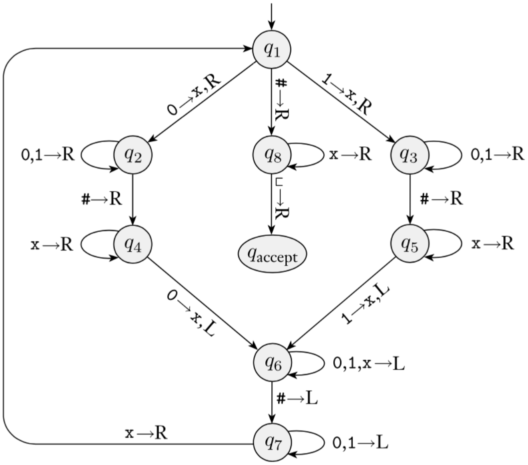


Figure 3.10, Sipser 2012.

Example of TM: power of 2

Consider the language $A = \{0^{2^n} : n \geq 0\}$.

Key observation: $2^n = 2^{n-1} \times 2$ for $n \geq 1$, otherwise $2^n = 1$.

The Turing Machine will cross off every other 0 (replace it by x), and accept when there is a single 0 left.

Example of TM: power of 2

Consider the language $A = \{0^{2^n} : n \geq 0\}$.

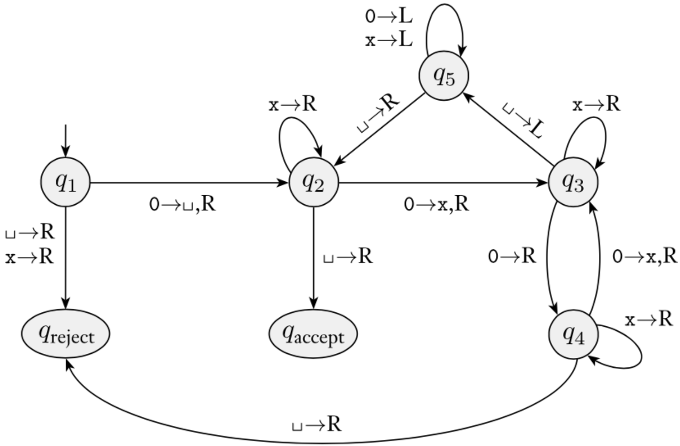


Figure 3.8, Sipser 2012.

Example of TM: multiplication

Consider the language $A = \{a^i b^j c^k : k = i \times j \geq 1\}$.

Approach:

- First check if the input w is $a^i b^j c^k$ for some $i, j, k \geq 1$ (DFA suffices).
- How would you test $w \in A$ if you were a Turing machine? You only see your current state and a symbol at the current head, but also knows that the input is of the form $a^i b^j c^k$.

Example of TM: multiplication

Consider the language $A = \{a^i b^j c^k : k = i \times j \geq 1\}$.

Approach:

- First check if the input w is $a^i b^j c^k$ for some $i, j, k \geq 1$ (DFA suffices).
- How would you test $w \in A$ if you were a Turing machine? You only see your current state and a symbol at the current head, but also knows that the input is of the form $a^i b^j c^k$.
- Idea: c^k can be written as b^j concatenated i times if and only if $k = ij$.
  1. cross off one a at a time, then for each of such cross off:
  2. zig-zag between the b-part and c-part, cross off one b and one c (always the leftmost one alive).
  3. If we're short of c, then reject.
  4. when all b's are crossed off, restore all b's.
  5. If all a's are crossed off, check if there is any c alive. If so, reject. Otherwise accept.