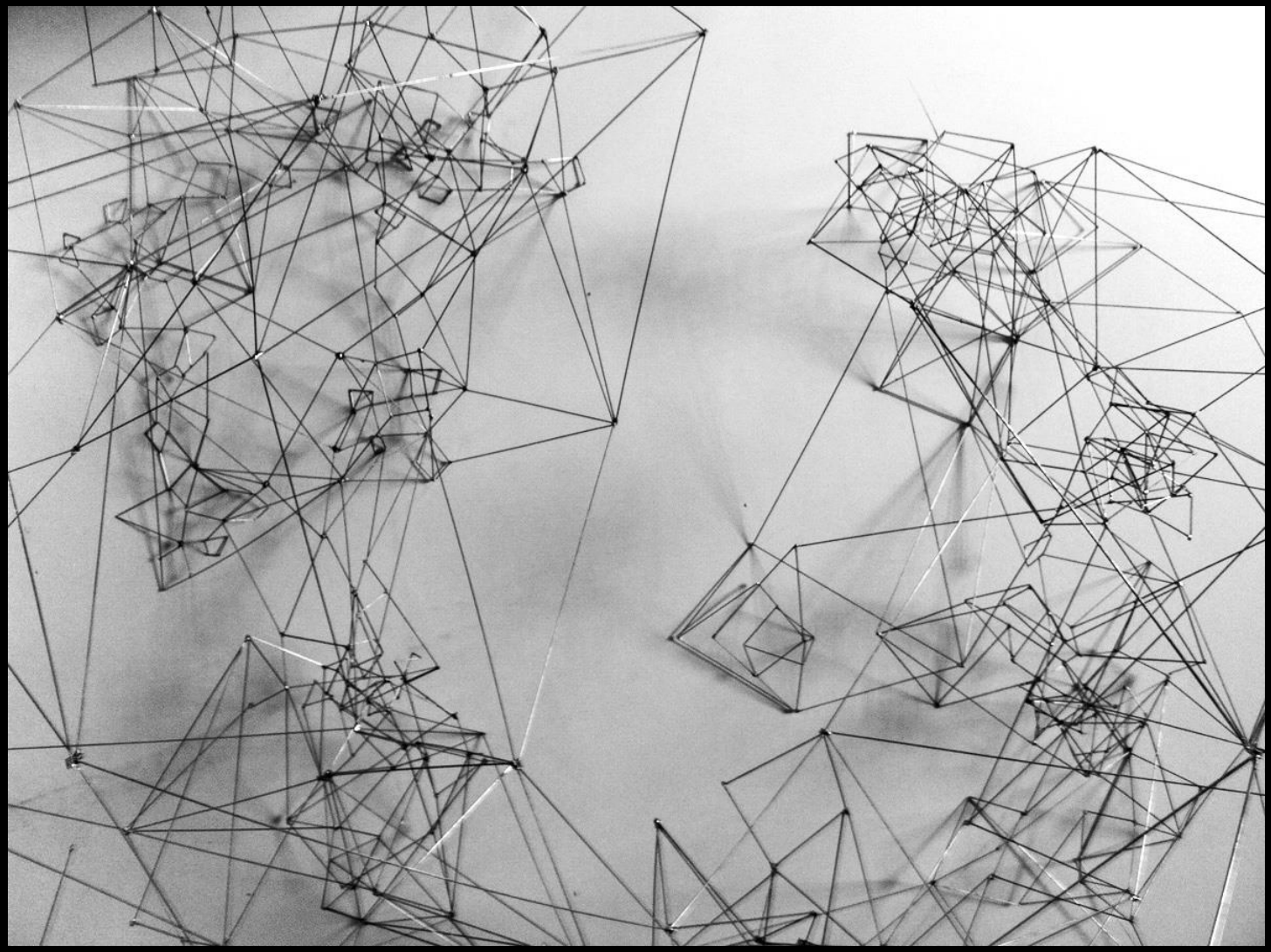


The Church-Turing Thesis

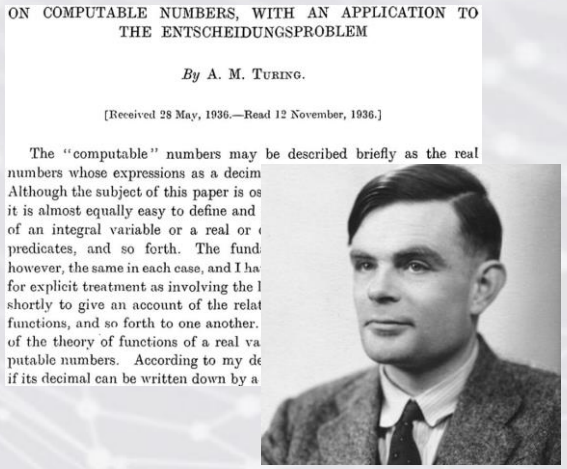
Contents

- Turing Machines
- Variants of Turing Machines

Turing Machines

- Recalling the computational models we have encountered thus far:
  - (*) FA (very limited amount of memory)
  - (*) PDA (unlimited memory but limited to LIFO processing)

- In both cases these models were too restrictive to serve as models for general computation.


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 decim
Although the subject of this paper is os
it is almost equally easy to define and
of an integral variable or a real or
predicates, and so forth. The fund
however, the same in each case, and I ha
for explicit treatment as involving the l
shortly to give an account of the relat
functions, and so forth to one another.
of the theory of functions of a real va
putable numbers. According to my de
if its decimal can be written down by a

- Turing's 1936 paper "On Computable Numbers, With an Application to the Entscheidungsproblem" introduced (concurrent to Church) a general-purpose computational model – the **Turing Machine** (TM) – equivalent in power to modern day (even quantum) computational models.

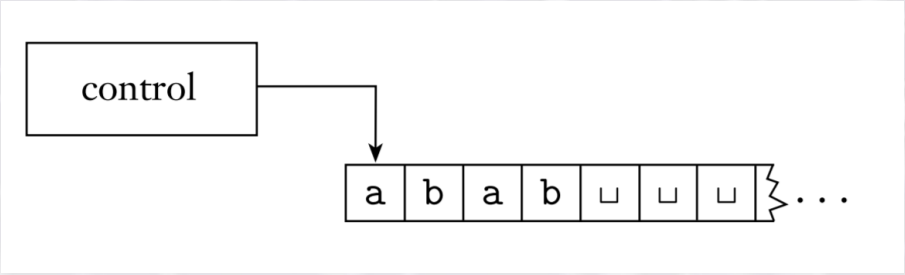
- Importantly, Turing showed that even these general-purpose devices cannot solve certain problems – that is, these problems are beyond the theoretical limits of computation.

Turing Machines

• TMs use an **infinite tape** for memory; a TM has a **tape head** that can *read/write* symbols and move *left/right* along the tape.

• Initially, the tape contains only the input string and is blank everywhere else; the machine continues computing until it produces an output.

• The outputs **accept** and **reject** are obtained by entering designated *accepting/rejecting* state. If the TM doesn't enter an accepting/rejecting state it will go on forever, without halting.



Turing Machines

• Here is a summary of the differences between FA and TMs:

(1) A TM can both read/write with respect to its tape

(2) The tape head can move both left/right

(3) The tape is infinite

(4) The special states for rejecting and accepting take effect immediately

• Let's consider TM M_1 for testing membership in the language: $B = \{w\#w | w \in \{0,1\}^*\}$.

Basic Idea for M_1:

(1) Zig-zag across the tape to corresponding positions on either side of #; if same symbol is found, cross them off; otherwise if not or no # is found **reject**.

(2) When all symbols left of the # have been crossed off, check to see if there are any remaining symbols to right of #; if symbols remain **reject**; otherwise, **accept**.

Turing Machines

• Let's consider TM M_1 for testing membership in the language: $B = \{w\#w | w \in \{0,1\}^*\}$.

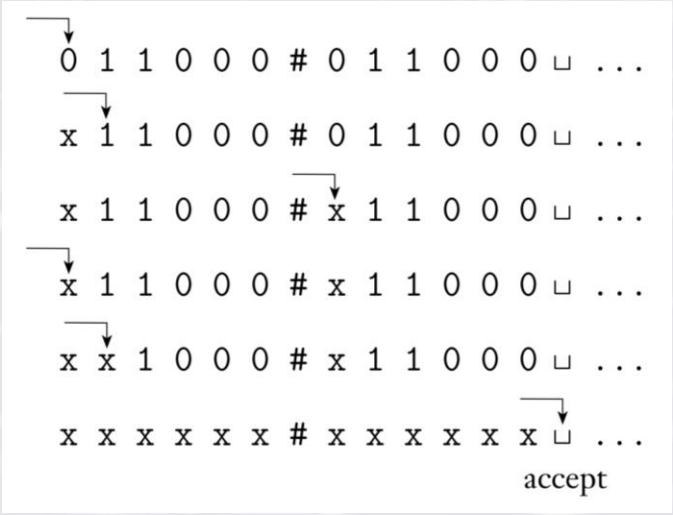
.

Basic Idea for M_1:

(1) Zig-zag across the tape to corresponding positions on either side of #; if same symbol is found, cross them off; otherwise if not or no # is found **reject**.

(2) When all symbols left of the # have been crossed off, check to see if there are any remaining symbols to right of #; if symbols remain **reject**; otherwise, **accept**.

Here is an example run for the input 011000#011000 on M_1.



Turing Machines

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

A few notes:

- M receives its input $w = w_1 \ldots w_n \in \Sigma^*$ on the leftmost n squares of the tape, and the rest of the tape is blank. Note that Σ does not contain the blank symbol.
- If M every tries to move its head to the left of the left-hand end of the tape, the head stays in the same place.
- As a TM computes, changes occur in the current state, the current tape contents and the current head location. A setting of these three items is called a configuration of the TM.

Turing Machines

• We say that configuration C_1 **yields configuration** C_2 if the TM can legally go from C_1 to C_2 in a single step.

• Concretely, suppose we have a, b, and c in Σ, as well as u and v in Γand states q_i and q_j. In that case, $ua\ q_i\ bv$ and $u\ q_j\ acv$ are two configurations. Say that:

$$ua\ q_i\ bv \textbf{ yields } u\ q_j\ acv$$

if in the transition function $\delta(q_i, b) = (q_j, c, L)$. That handles the case where the TM moves leftward. For rightward move, we say that:

$$ua\ q_i\ bv \textbf{ yields } uac\ q_j\ v$$

if $\delta(q_i, b) = (q_j, c, R)$.

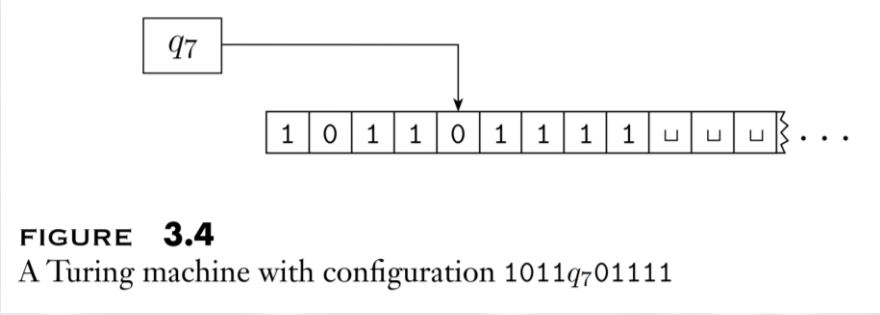


FIGURE **3.4**
A Turing machine with configuration $1011q_701111$

Turing Machines

• The **start configuration** of M on input w is the configuration $q_0\ w$; in an **accepting configuration,** the state of the configuration is q_{accept}; in a **rejecting configuration**, the state configuration is q_{reject}.

• Accepting and rejecting configurations are called **halting configurations** and do not yield further configurations.

• A TM **accepts input** w if a sequence of configurations C_1, C_2 , …, C_k, where:

(1) C_1 is the start configuration of M on input w

(2) Each C_i yields C_{i+1}

(3) C_k is an accepting configuration

Turing Machines

• The collection of strings that M accepts is **the language of M**, denoted **L(M)**.

A language is called **Turing-recognizable** if some TM recognizes it.

• Note that when a TM is run on an input, *three outcomes are possible*: **accept**, **reject** or **loop** (indefinitely).

A TM can **fail to accept** an input by entering the q_{reject} state or by looping.
A *decider* is a TM that halts on all inputs (i.e. it always makes a "decision").

A language is **Turing-decidable** or simply **decidable** if some TM decides it.

Turing Machines

• The collection of strings that M accepts it **the language of M**, denoted **L(M)**.

A language is called **Turing-recognizable** if some TM recognizes it.

• Note that when a TM is run on an input, *three outcomes are possible*: **accept**, **reject** or **loop** (indefinitely).

A TM can **fail to accept** an input by entering the q_{reject} state or by looping.
A *decider* is a TM that halts on all inputs (i.e. it always makes a "decision").

A language is **Turing-decidable** or simply **decidable** if some TM decides it.

(*) Observe that **every decidable language is Turing-recognizable**; however, there exist Turing-recognizable languages that are undecidable.

Turing Machines

• For simplicity, we typically avoid formal (and thus often tedious) definitions of TMs; instead, we often prefer a high-level description.

Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

M_2 = "On input string w:

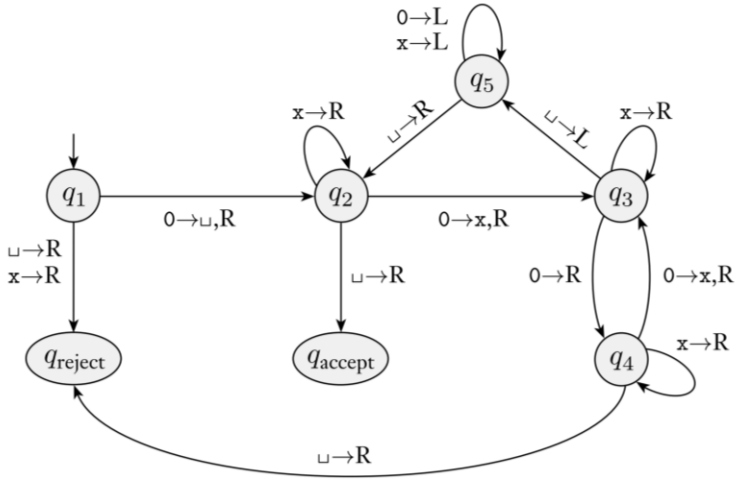
1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1."

Turing Machines

Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

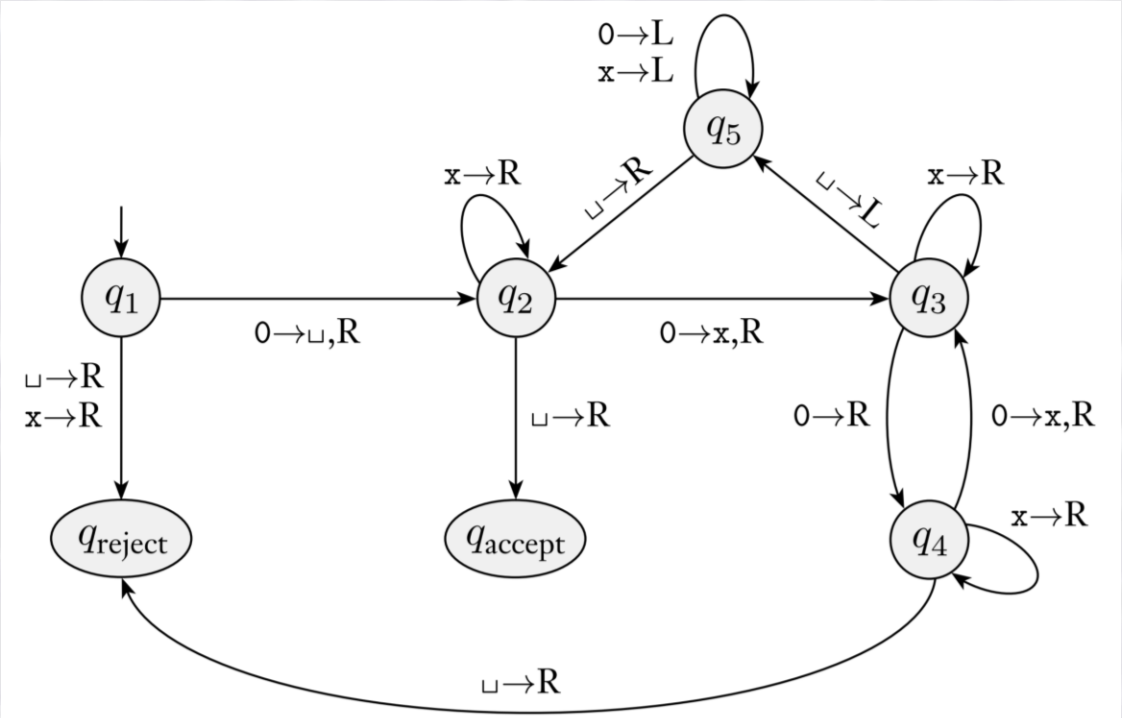
M_2 = "On input string w:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1."



- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$,
- $\Sigma = \{0\}$, and
- $\Gamma = \{0, x, ␣\}$.
- We describe δ with a state diagram (see Figure 3.8).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.

Turing Machines

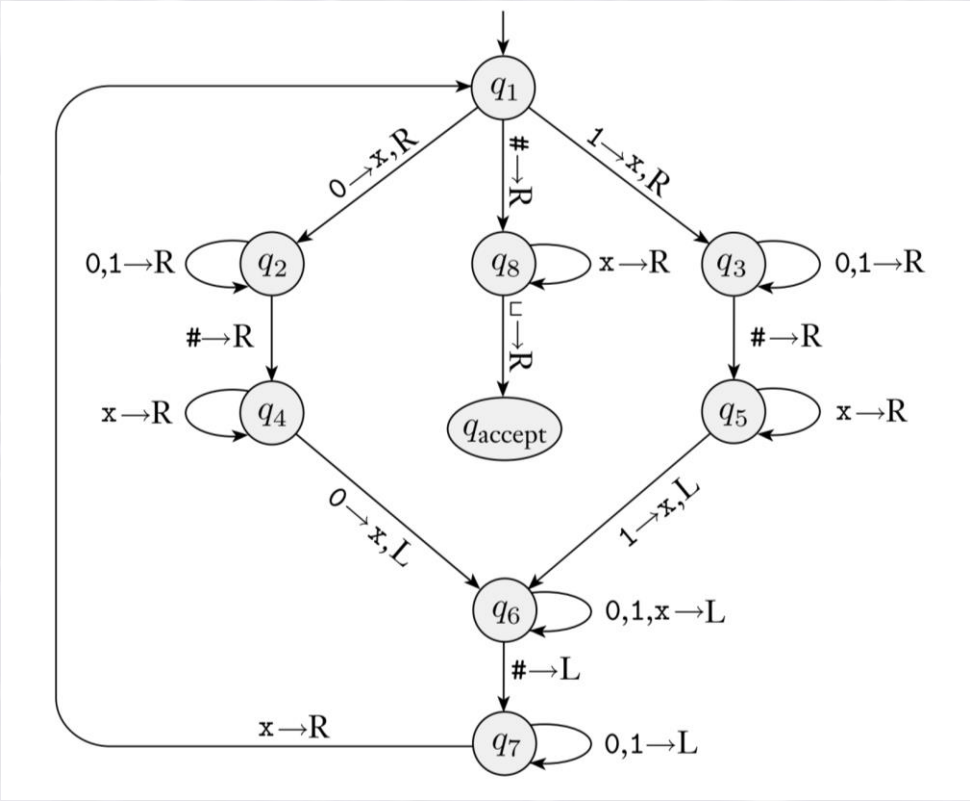


$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$\sqcup q_5 x x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

Turing Machines

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described (page 167) for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

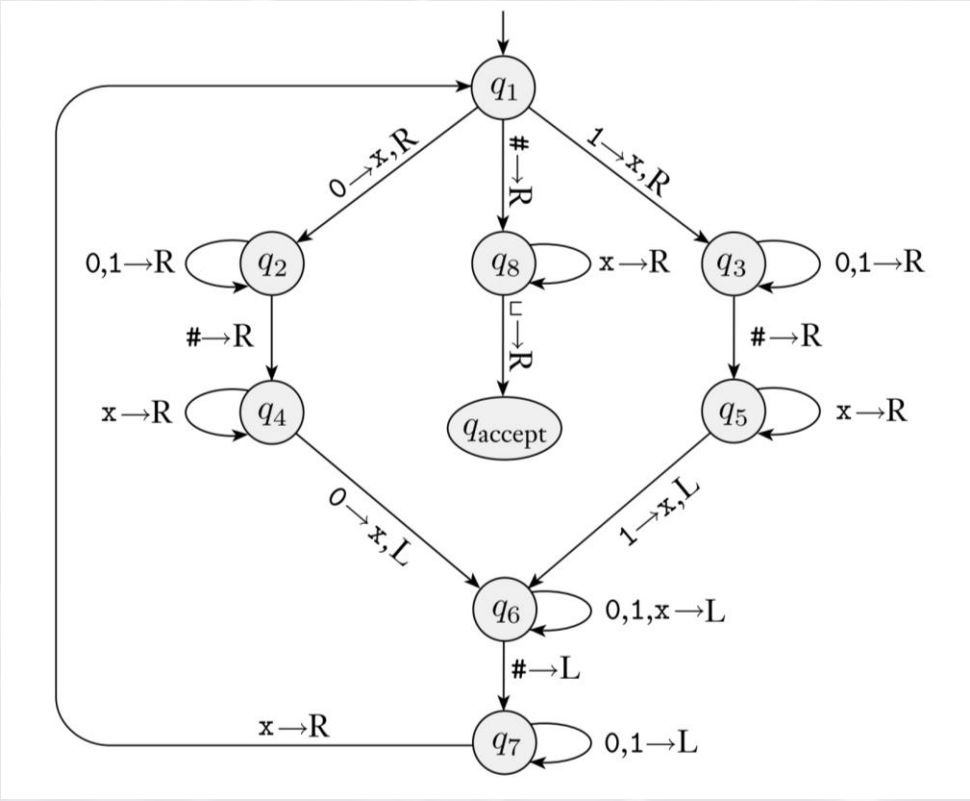
- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.



Turing Machines

The following is a formal description of $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, the Turing machine that we informally described (page 167) for deciding the language $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.
- We describe δ with a state diagram (see the following figure).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.



$q_1 101\#101\beta$

$Xq_3 01\#101\beta$

$X0q_3 1\#101\beta$

$X01q_3 \#101\beta$

$X01\#q_5 101\beta$

$X01q_6 \#X01\beta$

$\vdots$

$q_7 X01\#X01\beta$

$Xq_1 01\#X01\beta$

$\vdots$

$Xq_7 X1\#XX1\beta$

$XXq_1 1\#XX1\beta$

$\vdots$

$XXq_7 X\#XXX\beta$

$XXXq_1 \#XXX\beta$

$XXX\#q_8 XXX\beta$

$\vdots$

$XXX\#XXX\beta q_{accept}$

Turing Machines

Here, a TM M_3 is doing some elementary arithmetic. It decides the language $C = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$.

M_3 = "On input string w:

1. Scan the input from left to right to determine whether it is a member of $\mathtt{a}^+\mathtt{b}^+\mathtt{c}^+$ and *reject* if it isn't.
2. Return the head to the left-hand end of the tape.
3. Cross off an a and scan to the right until a b occurs. Shuttle between the b's and the c's, crossing off one of each until all b's are gone. If all c's have been crossed off and some b's remain, *reject*.
4. Restore the crossed off b's and repeat stage 3 if there is another a to cross off. If all a's have been crossed off, determine whether all c's also have been crossed off. If yes, *accept*; otherwise, *reject*."

Variants of Turing Machines

• Variants of TMs include TMs with multiple tapes (and tape heads) and non-deterministic machines.

• Remarkably, **these variants all possess the same computational power as the original TM** we described (meaning that they recognize the same languages); in this way we say that TMs are **robust**, in this sense that they are invariant to these modifications.

Variants of Turing Machines

• A **multitape TM** is identical to the original TM with the addition of several tapes. Conventionally, the input appears on tape 1 and the other tapes start out blank; each tape has its own head for reading and writing.

The transition function is altered to accommodate multiple tapes as follows:

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k,$$

where k is the number of tapes (L: left, R: right, S: stay put); the expression:

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$$

means that if the machine is in state q_i the heads *1* through *k* are reading symbols a_1 through a_k, the machine goes to state q_j, writes symbols b_1 through b_k, and directs each head to move left or right, or to stay put, as specified.

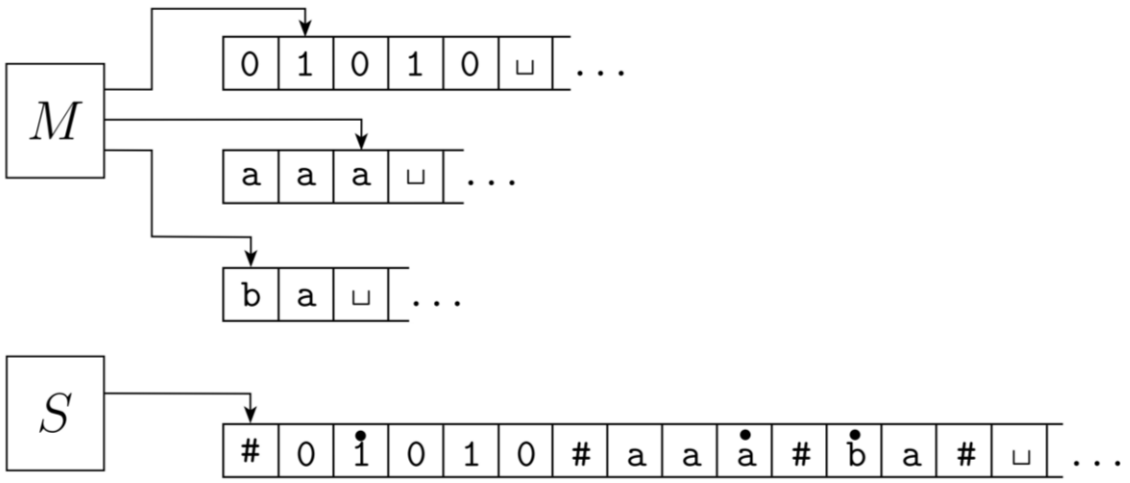
Variants of Turing Machines

Theorem. Every multitape TM has an equivalent single-tape TM, i.e. multitape TMs are equivalent in power to single-tape TMs.

Proof. We describe how to convert a multitape TM *M* to an equivalent single-tape TM *S*.

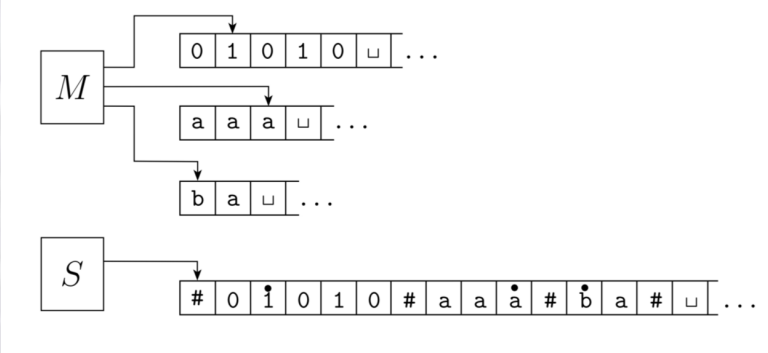
Say that *M* has *k* tapes. Then ***S* simulates the effect of *k* tapes** by storing their information on a single tape. ***S* uses the new symbol # as a delimiter**.

In addition to the contents of these tapes, *S* must keep track of the locations of the heads. It does so by writing a tape symbol with a dot above it to mark the place where the head of the tape would be.



Variants of Turing Machines

Theorem. Every multitape TM has an equivalent single-tape TM, i.e. multitape TMs are equivalent in power to single-tape TMs.



S = "On input $w = w_1 \cdots w_n$:

1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains

$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#.$$

2. To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the $(k+1)$st #, which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M's transition function dictates.
3. If at any point S moves one of the virtual heads to the right onto a #, this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before."

Variants of Turing Machines

Corollary. A language is Turing-recognizable *iff* some multitape TM recognizes it.

Non-deterministic Turing Machines

• Recall that with **non-determinism**, the transition function outputs a set (of possible states, symbols, etc.). The computation graph of a non-deterministic TM is a tree whose branches correspond to different possibilities for the machine.

• The transition function for a non-deterministic TM has the form:

$$\delta: Q \times \Gamma^k \longrightarrow P(Q \times \Gamma \times \{L, R\})$$

Variants of Turing Machines

Theorem. Every non−deterministic TM has an equivalent deterministic TM.

Proof Idea: We can simulate any non-deterministic TM *N* with a deterministic TM *D*. The idea is to have *D* try all possible branches of *N*'s non-deterministic computation.

• If *D* ever finds and accept state on one of these branches, *D* accepts. Otherwise, *D*'s simulation will not terminate.

• Consider N's computation on an input w as a tree; each branch is a non-deterministic computation, and each node is a configuration of N; the root of the tree is the start configuration.

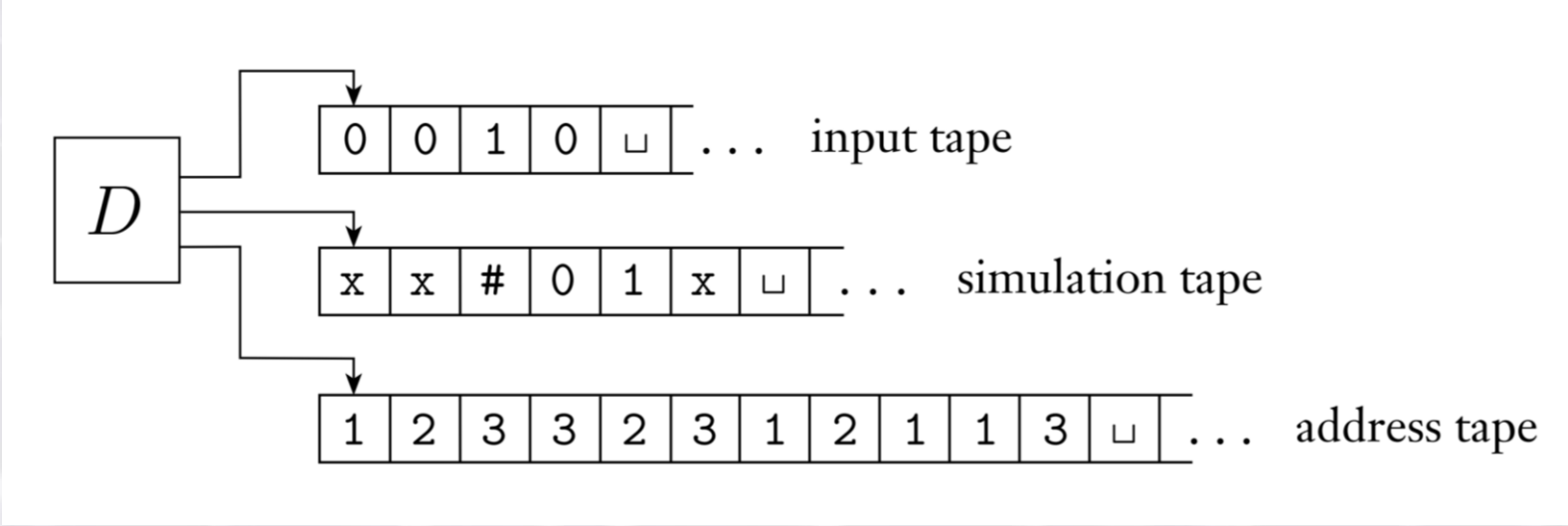
In summary, **we have *D* simulate *N* by performing a BFS** (*breadth-first search*) over the computation tree of *N* (note that using DFS is a bad idea here – why?).

Variants of Turing Machines

Theorem. Every non−deterministic TM has an equivalent deterministic TM.

Proof. The simulating deterministic TM *D* has three tapes (which is computationally equivalent to one tape by the preceding theorem).

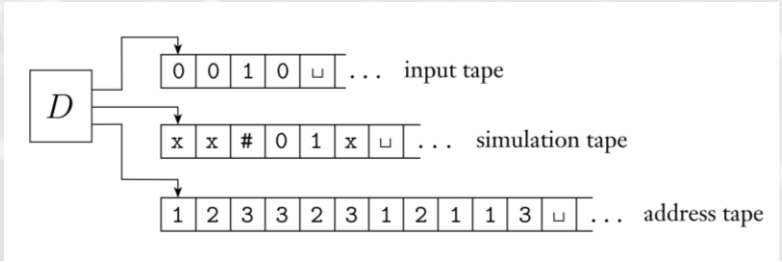
Tape 1 contains the input string (these tape contents are never altered); **Tape 2** maintains a copy of N's tape on some branch of its non-deterministic computation; **Tape 3** keeps track of D's location in N's non-deterministic computation tree.



Variants of Turing Machines

Theorem. Every non−deterministic TM has an equivalent deterministic TM.

Say that every node in the tree has at most b children (so b is the "branching factor" of the tree); to every node in the tree we assign an address that is a string over the alphabet $\Gamma_b = \{1,2,\dots,b\}$. For example, we assign the address 231 to the node we arrive at by starting at the root, going to its 2nd child, going to that node's 3rd child, and finally going to the node's first child. In this fashion, Tape 3 contains a string over Γ_b.



1. Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ϵ.
3. Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each step of N, consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, *accept* the input.
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of N's computation by going to stage 2.

Variants of Turing Machines

Corollary. A language is Turing−recognizable *iff* some non−deterministic TM recognizes it.

• We can modify the previous proof so that if *N* always halts on all branches of its computation, *D* will always halt. We call a non-deterministic TM a **decider** if all branches halt on all inputs.

Corollary. A language is decidable *iff* some non−determinstic TM decides it.

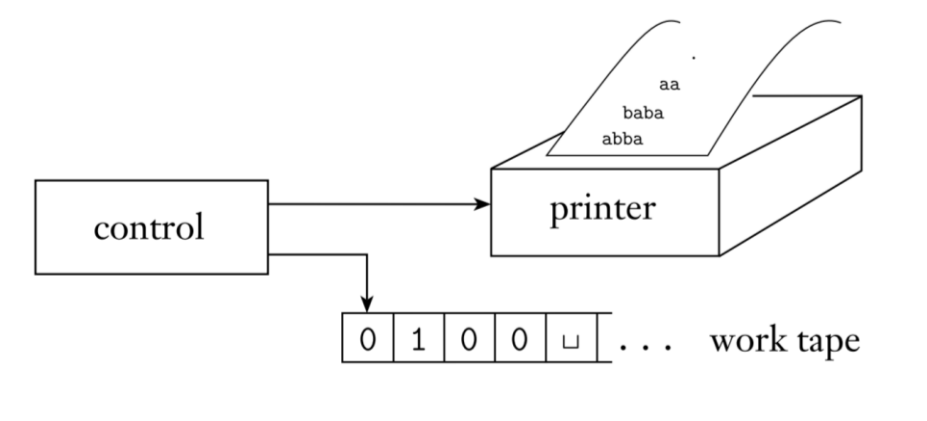
Variants of Turing Machines

Enumerators:

• Note that the term **recursively enumerable language** is used synonymously (in other sources) for a *Turing-recognizable language*.

• The term originates from a type of TM variant called an **enumerator**. Put informally, an enumerator is a TM with an attached "printer." The TM can use its printer as an output device to print strings.

• An enumerator *E* starts with a blank input on its work tape. The language enumerated by *E* is the collection of all the strings that it eventually prints out. Note that *E* may generate the strings of the language in any order (and possibly with repetition).



Variants of Turing Machines

Theorem. A language is Turing−recognizable *iff* some enumerator enumerates it.

Proof. First we show that if we have an enumerator E that enumerates a language A, a TM M recognizes A.

The TM M works in the following way:

M: on input w

(1) Run E. Every time that E outputs a string, compare it with w.
(2) If w every appears in the output of E, **accept**.

Conversely, if a TM M recognizes a language A, we can construct the following enumerator E for A. Say that $s_1, s_2, s_3, \ldots$ is a list of all possible strings in Σ^*.

E: ignore the input

(1) Repeat the following for $i = 1, 2, 3, \ldots$
(2) Run M for i steps on each input, $s_1, s_2, \ldots, s_i$.
(3) If any computations accept, print out the corresponding s_j.

If M accepts a particular string s, eventually it will appear on the list generate by E. In fact, it will appear on the list infinitely many times because M runs from the beginning on each string for each repetition of step 1. This procedure gives the effect of running M in parallel on all possible input strings.

Variants of Turing Machines

• Many other models of general-purpose computation have been proposed (e.g. Church's lambda-calculus, among others). All share the essential features of TMs, namely: unrestricted access to unlimited memory. Remarkably, **all models with that feature turn out to be equivalent in power** (under some reasonable requirements, e.g. finite compute time).

• Recall that prior to the work of Church and Turing, the scientific community was wanting of a formal definition of an algorithm.

Variants of Turing Machines

• In particular, several of **Hilbert's 23 millennium problems** (1900) alluded to "processes" which yield a solution in a finite number of steps. Famously, Hilbert's *10th Problem* asks for the derivation of such a process to determine whether a polynomial has an integral root (he presumed that such a procedure exists; in 1970 this was proven impossible).

The **Church-Turing Thesis** asserts the equivalence of the intuitive notion of an algorithm (i.e. a sequence of "pencil and paper operations") with Turing machine algorithms.

Church-Turing Thesis.
Intuitive notion of algorithms $\leftrightarrow$ Turing Machine algorithms.

Fin