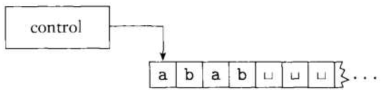
TURING MACHINES

20 February 2024 11:35 PM

The Turing machine model uses an infinite tape as its unlimited memory. It has a tape head that can read and write symbols and move around on the tape.

Initially the tape contains only the input string and is blank everywhere else. If the machine needs to store information, it may write this information on the tape. To read the information that it has written, the machine can move its head back over it. The machine continues computing until it decides to produce an output. The outputs *accept* and *reject* are obtained by entering designated accepting and rejecting states. If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.



, δ takes the form: $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$

machine

is in a certain state q and the head is over a tape square containing a symbol a, and if $\delta(q, a) = (r, b, L)$, the machine writes the symbol b replacing the a, and goes to state r. The third component is either L or R and indicates whether the head moves to the left or right after writing. In this case the L indicates a move to the left.

DEFINITION 3.3

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*** ␣,
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$.

by the transition function. If M ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates L. The computation continues until it M goes on forever.

As a Turing machine 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 Turing machine. Configurations often are represented in a special way. For a state q and two strings u and v over the tape alphabet Γ we write $u\,q\,v$ for the configuration where the current state is q, the current tape contents is uv, and the current head location is the first symbol of v. The tape contains only blanks following the last symbol of v. For example, $1011q_701111$ represents the configuration when the tape is 101101111, the current state is q_7, and the head is currently on the second 0. The following figure depicts a Turing machine with that configuration.

chine computes. Say that configuration C_1 ***yields*** configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step. We define this notion formally as follows.

Suppose that we have a, b, and c in Γ, as well as u and v in Γ^* and states q_i and q_j. In that case $ua\,q_i\,bv$ and $u\,q_j\,acv$ are two configurations. Say that

$$ua\,q_i\,bv \quad \text{yields} \quad u\,q_j\,acv$$

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

$$ua\,q_i\,bv \quad \text{yields} \quad uac\,q_j\,v$$

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

The ***start configuration*** of M on input w is the configuration $q_0\,w$, which indicates that the machine is in the start state q_0 with its head at the leftmost position on the tape. In an ***accepting configuration*** the state of the configuration is q_{accept}. In a ***rejecting configuration*** the state of the configuration is q_{reject}. Accepting and rejecting configurations are ***halting configurations*** and do not yield further configurations. Because the machine is defined to halt when in the states q_{accept} and q_{reject}, we equivalently could have defined the transition function to have the more complicated form $\delta\colon Q' \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$, where Q' is Q without q_{accept} and q_{reject}. A Turing machine M ***accepts*** input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where

1. C_1 is the start configuration of M on input w,
2. each C_i yields C_{i+1}, and
3. C_k is an accepting configuration.

The collection of strings that M accepts is ***the language of M***, or ***the language recognized by M***, denoted $L(M)$.

DEFINITION 3.5

Call a language ***Turing-recognizable*** if some Turing machine recognizes it.[1]

A Turing machine M can fail to accept an input by entering the q_{reject} state and rejecting, or by looping. Sometimes distinguishing a machine that is looping from one that is merely taking a long time is difficult. For this reason we prefer Turing machines that halt on all inputs; such machines never loop. These machines are called ***deciders*** because they always make a decision to accept or reject. A decider that recognizes some language also is said to ***decide*** that language.

DEFINITION 3.6

Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.[2]

For describing a turing machine, one can just use a higher definition instead of defining the whole 7-tuple.
Describing the high level lang is in a way describing the 7-tuple

HIGH LEVEL DESCRIPTION

FORMAL DESCRIPTION

EXAMPLE 3.7

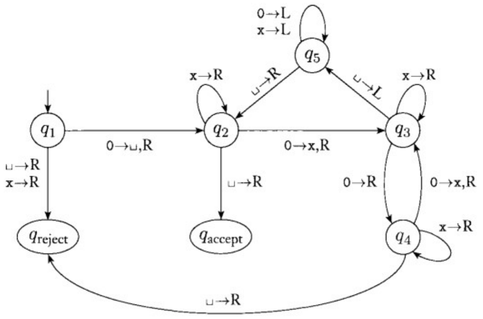
Here we describe a Turing machine (TM) M_2 that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language consisting of all strings of 0s whose length is a power of 2.

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."

achine

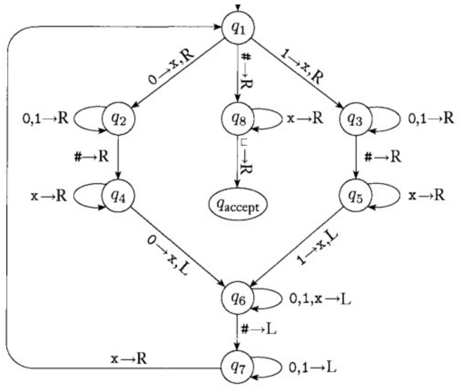
- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0\}$, and
- $\Gamma = \{0, \text{x}, \sqcup\}$.
- We describe δ with a state diagram (see Figure 3.8).
- The start, accept, and reject states are q_1, q_{accept}, and q_{reject}.



In this state diagram, the label $0\rightarrow\sqcup$,R appears on the transition from q_1 to q_2. This label signifies that, when in state q_1 with the head reading 0, the machine goes to state q_2, writes $\sqcup$, and moves the head to the right. In other words, $\delta(q_1,0) = (q_2,\sqcup,\text{R})$. For clarity we use the shorthand $0\rightarrow$R in the transition from q_3 to q_4, to mean that the machine moves to the right when reading 0 in state q_3 but doesn't alter the tape, so $\delta(q_3,0) = (q_4,0,\text{R})$.

guage $B = \{w\#w \mid w \in \{0,1\}^*\}$.

- $Q = \{q_1, \ldots, q_{14}, q_{\text{accept}}, q_{\text{reject}}\}$,
- $\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,\text{x},\sqcup\}$.



EXAMPLE 3.11

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

M_3 = "On input string w:

1. Scan the input from left to right to determine whether it is a member of $a^+b^+c^+$ and *reject* if it isn't. → can make such checks
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*."

EXAMPLE 3.12

Here, a TM M_4 is solving what is called the ***element distinctness problem***. It is given a list of strings over $\{0,1\}$ separated by #s and its job is to accept if all the strings are different. The language is

$$E = \{\#x_1\#x_2\# \cdots \#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}.$$

Machine M_4 works by comparing x_1 with x_2 through x_l, then by comparing x_2 with x_3 through x_l, and so on. An informal description of the TM M_4 deciding this language follows.

M_4 = "On input w:

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only x_1 was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to Stage 3."

3.2 VARIANTS OF TURING MACHINES

invariance to certain changes in the definition ***robustness***. Both finite automata and pushdown automata are somewhat robust models, but Turing machines have an astonishing degree of robustness.

The transition function would then have the form $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R, S\}$. Might this feature allow Turing machines to recognize additional languages, thus adding to the power of the model? Of course not, because we can convert any TM with the "stay put" feature to one that does not have it. We do so by replacing each stay put transition with two transitions, one that moves to the right and the second back to the left.

This small example contains the key to showing the equivalence of TM vari-
ants. To show that two models are equivalent we simply need to show that we can simulate one by the other.

MULTITAPE TURING MACHINES

A ***multitape Turing machine*** is like an ordinary Turing machine with several tapes. Each tape has its own head for reading and writing. Initially the input

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

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

means that, if the machine is in state q_i and 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.

Multitape Turing machines appear to be more powerful than ordinary Turing machines, but we can show that they are equivalent in power. Recall that two machines are equivalent if they recognize the same language.

THEOREM 3.13

Every multitape Turing machine has an equivalent single-tape Turing machine.

COROLLARY 3.15

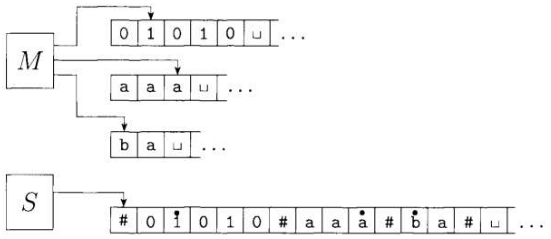
A language is Turing-recognizable if and only if some multitape Turing machine recognizes it.

THEOREM 3.13

Every multitape Turing machine has an equivalent single-tape Turing machine.

PROOF We show how to convert a multitape TM M to an equivalent single-tape TM S. The key idea is to show how to simulate M with S.

Say that M has k tapes. Then S simulates the effect of k tapes by storing their information on its single tape. It uses the new symbol # as a delimiter to separate the contents of the different tapes. 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 on that tape would be. Think of these as "virtual" tapes and heads. As before, the "dotted" tape symbols are simply new symbols that have been added to the tape alphabet. The following figure illustrates how one tape can be used to represent three tapes.



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

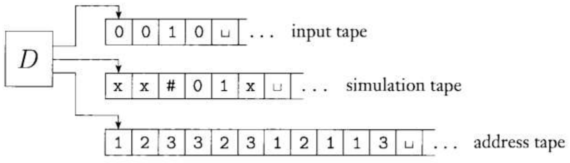
1. First S puts its tape into the format that represents all k tapes of M. The formatted tape contains
$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#$$
2. To simulate a single move, S scans its tape from the first #, which marks the left-hand end, to the $(k+1)$st #, which marks 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."

NONDETERMINISTIC TURING MACHINES

$$\delta\colon Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L},\mathrm{R}\}).$$

idea is to have D explore the tree by using depth-first search. The depth-first search strategy goes all the way down one branch before backing up to explore other branches. If D were to explore the tree in this manner, D could go forever down one infinite branch and miss an accepting configuration on some other branch. Hence we design D to explore the tree by using breadth first search instead. This strategy explores all branches to the same depth before going on to explore any branch to the next depth. This method guarantees that D will visit every node in the tree until it encounters an accepting configuration.

PROOF The simulating deterministic TM D has three tapes. By Theorem 3.13 this arrangement is equivalent to having a single tape. The machine D uses its three tapes in a particular way, as illustrated in the following figure. Tape 1 always contains the input string and is never altered. Tape 2 maintains a copy of N's tape on some branch of its nondeterministic computation. Tape 3 keeps track of D's location in N's nondeterministic computation tree.


D
0 0 1 0 ␣ ... input tape
x x # 0 1 x ␣ ... simulation tape
1 2 3 3 2 3 1 2 1 1 3 ␣ ... address tape

THEOREM 3.16

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

COROLLARY 3.18

A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.

COROLLARY 3.19

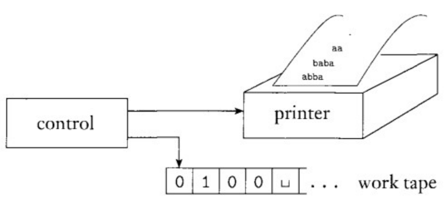
A language is decidable if and only if some nondeterministic Turing machine decides it.

Let's first consider the data representation on tape 3. Every node in the tree can have at most b children, where b is the size of the largest set of possible choices given by N's transition function. To every node in the tree we assign an address that is a string over the alphabet $\Sigma_b = \{1, 2, \ldots, b\}$. 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 that node's 1st child. Each symbol in the string tells us which choice to make next when simulating a step in one branch in N's nondeterministic computation. Sometimes a symbol may not correspond to any choice if too few choices are available for a configuration. In that case the address is invalid and doesn't correspond to any node. Tape 3 contains a string over Σ_b. It represents the branch of N's computation from the root to the node addressed by that string, unless the address is invalid. The empty string is the address of the root of the tree. Now we are ready to describe D.

1. Initially tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2.
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 lexicographically next string. Simulate the next branch of N's computation by going to stage 2.

ENUMERATORS

As we mentioned earlier, some people use the term *recursively enumerable language* for Turing-recognizable language. That term originates from a type of Turing machine variant called an ***enumerator***. Loosely defined, an enumerator is a Turing machine with an attached printer. The Turing machine can use that printer as an output device to print strings. Every time the Turing machine wants to add a string to the list, it sends the string to the printer. Exercise 3.4 asks you to give a formal definition of an enumerator. The following figure depicts a schematic of this model.



An enumerator E starts with a blank input tape. If the enumerator doesn't halt, it may print an infinite list of strings. The language enumerated by E is the collection of all the strings that it eventually prints out. Moreover, E may generate the strings of the language in any order, possibly with repetitions.

THEOREM 3.21

A language is Turing-recognizable if and only if 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 ever appears in the output of E, accept."
```

Clearly, M accepts those strings that appear on E's list.

Now we do the other direction. If 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, ...
    2.     Run M for i steps on each input, s_1, s_2, ..., 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 generated 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.

3.3 THE DEFINITION OF ALGORITHM

Informally speaking, an *algorithm* is a collection of simple instructions for carrying out some task. Commonplace in everyday life, algorithms sometimes are

The definition came in the 1936 papers of Alonzo Church and Alan Turing. Church used a notational system called the λ-calculus to define algorithms. Turing did it with his "machines." These two definitions were shown to be equivalent. This connection between the informal notion of algorithm and the precise definition has come to be called the ***Church–Turing thesis***.

such algorithms? Students commonly ask this question, especially when preparing solutions to exercises and problems. Let's entertain three possibilities. The first is the *formal description* that spells out in full the Turing machine's states, transition function, and so on. It is the lowest, most detailed, level of description. The second is a higher level of description, called the *implementation description*, in which we use English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape. At this level we do not give details of states or transition function. Third is the *high-level description*, wherein we use English prose to describe an algorithm, ignoring the implementation details. At this level we do not need to mention how the machine manages its tape or head.

We now set up a format and notation for describing Turing machines. The input to a Turing machine is always a string. If we want to provide an object other than a string as input, we must first represent that object as a string. Strings can easily represent polynomials, graphs, grammars, automata, and any combination of those objects. A Turing machine may be programmed to decode the representation so that it can be interpreted in the way we intend. Our notation for the encoding of an object O into its representation as a string is $\langle O\rangle$. If we have several objects $O_1, O_2, \ldots, O_k$, we denote their encoding into a single string $\langle O_1, O_2, \ldots, O_k\rangle$. The encoding itself can be done in many reasonable ways. It doesn't matter which one we pick because a Turing machine can always translate one such encoding into another.

In our format, we describe Turing machine algorithms with an indented segment of text within quotes. We break the algorithm into stages, each usually involving many individual steps of the Turing machine's computation. We indicate the block structure of the algorithm with further indentation. The first line of the algorithm describes the input to the machine. If the input description is simply w, the input is taken to be a string. If the input description is the encoding of an object as in $\langle A\rangle$, the Turing machine first implicitly tests whether the input properly encodes an object of the desired form and rejects it if it doesn't.

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}.$$

Here is a TM M_1 that recognizes D_1:

```
M_1 = "The input is a polynomial p over the variable x.
    1. Evaluate p with x set successively to the values 0, 1, −1, 2, −2,
       3, −3, . . . If at any point the polynomial evaluates to 0, accept."
```

Both M_1 and $\bar{M}$ are recognizers but not deciders.

Let A be the language consisting of all strings representing undirected graphs that are connected. Recall that a graph is ***connected*** if every node can be reached from every other node by traveling along the edges of the graph. We write

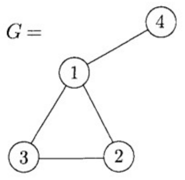
$$A = \{\langle G\rangle \mid G \text{ is a connected undirected graph}\}.$$

The following is a high-level description of a TM M that decides A.

```
M = "On input ⟨G⟩, the encoding of a graph G:
    1. Select the first node of G and mark it.
    2. Repeat the following stage until no new nodes are marked:
    3.     For each node in G, mark it if it is attached by an edge to a
           node that is already marked.
    4. Scan all the nodes of G to determine whether they all are
       marked. If they are, accept; otherwise, reject."
```

First, we must understand how $\langle G\rangle$ encodes the graph G as a string. Consider an encoding that is a list of the nodes of G followed by a list of the edges of G.

$G =$



$\langle G\rangle =$

`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

input is the proper encoding of some graph. To do so, M scans the tape to be sure that there are two lists and that they are in the proper form. The first list should be a list of distinct decimal numbers, and the second should be a list of pairs of decimal numbers. Then M checks several things. First, the node list should contain no repetitions, and second, every node appearing on the edge list should also appear on the node list. For the first, we can use the procedure given

3.15 Show that the collection of decidable languages is closed under the operation of

$^{\mathrm{A}}$**a.** union.
b. concatenation.
c. star.
d. complementation.
e. intersection.

3.16 Show that the collection of Turing-recognizable languages is closed under the operation of

$^{\mathrm{A}}$**a.** union.
b. concatenation.
c. star.
d. intersection.