Turing Machines
Recursive/Recursively Enumerable Languages

Schematic of Turing Machines

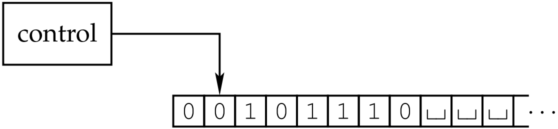
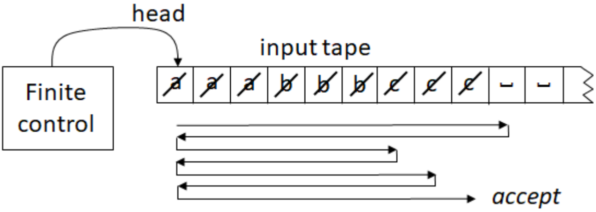


Figure: Schematic of Turing Machines

- A Turing machine has a finite set of control states.
- A Turing machine reads and writes symbols on an infinite tape.
- A Turing machine starts with an input on the left end of the tape.
- A Turing machine moves its read-write head in both directions.
- A Turing machine outputs accept or reject by entering its accepting or rejecting states respectively.
  - A Turing machine need not read all input symbols.
  - A Turing machine may not accept nor reject an input.

Turing Machines

- Consider $B = \{a^k b^k c^k : k \geq 0\}$.
- M_1 = "On input string w:
  1. Scan right until ␣ while checking if input is in $a^* b^* c^*$, reject if not
  2. Return head to left end.
  3. Scan right, crossing off single a, b, and c. (Tape alphabet = $\{a, b, c, \not{a}, \not{b}, \not{c}, ␣\}$)
  4. If the last one of each symbol, accept.
  5. If the last one of some symbol but not others, reject.
  6. If all symbols remain, return to left end and repeat from (3).



(Fig. from M. Sipser's class notes)

Turing Machines – Formal Definition

Definition 1

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

- Q is the finite set of states;
- Σ is the finite input alphabet not containing the blank symbol $\sqcup$;
- Γ is the finite tape alphabet with $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function;
- $q_0 \in Q$ is the start state;
- $q_{\text{accept}} \in Q$ is the accept state; and
- $q_{\text{reject}} \in Q$ is the reject state with $q_{\text{reject}} \neq q_{\text{accept}}$.

- The above definition is for deterministic Turing machines.
- Initially, a Turing machine receives its input $w = w_1 w_2 \cdots w_n \in \Sigma^*$ on the leftmost n cells of the tape.
- Other cells on the tape contain the blank symbol $\sqcup$.

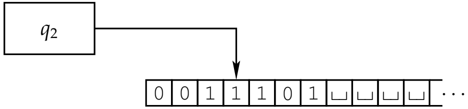
Configurations of Turing Machines

What is a configuration of an automaton?

- Intuitively, a configuration is a snapshot of the automaton's computation, recoding necessary information that determines how the automaton progresses further.
- For FA, a configuration is of the form (q, v) (or abbrev. as qv), where $q \in Q$ and $v \in \Sigma^*$ representing the remainder of the input (i.e., the portion of the input that has not been read).
  - If input string $abcdef$, and the FA in state q after reading c, the configuration is $qdef$.
  - the prefix abc does not affect how the FA behaves in the future.
- For PDA, a configuration is of the form (q, v, s), where $q \in Q$ is the current state, $v \in \Sigma^*$ is the remainder of the input , and $s \in \Gamma^*$ representing the content of the pushdown stack.
- For TMs, a configuration is of the form uqv, where $q \in Q$, $u, v \in \Gamma^*$ such that uv is the content of the tape and TM is reading the first symbol of v.

Computation of Turing Machines

- A configuration of a Turing machine contains its current states, current tape contents, and current head location.
- Let $q \in Q$ and $u, v \in \Gamma$. We write uqv to denote 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.
  - When we say "the current tape contents is uv," we mean an infinite tape contains $uv\sqcup\sqcup\cdots\sqcup\cdots$.
- Consider the configuration $\texttt{001}q_2\texttt{1101}$. The Turing machine
  - is at the state q_2;
  - has the tape contents `0011101`; and
  - has its head location at the second `1` from the left.



Computation of Turing Machines

- Let C_1 and C_2 be configurations. We say C_1 yields C_2 (written as $C_1 \vdash C_2$) if the Turing machine can go from C_1 to C_2 in one step.
- Formally, let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, and $q_i, q_j \in Q$.

$$\begin{aligned} uaq_ibv &\vdash uq_jacv && \text{if } \delta(q_i, b) = (q_j, c, L) \\ q_ibv &\vdash q_jcv && \text{if } \delta(q_i, b) = (q_j, c, L) \\ uaq_ibv &\vdash uacq_jv && \text{if } \delta(q_i, b) = (q_j, c, R) \end{aligned}$$

- Note the 2nd case when the current head location is the leftmost cell of the tape.
  - A Turing machine updates the leftmost cell without moving its head.
- Recall that uaq_i is in fact $uaq_i\sqcup$.
- Can you define the $\vdash$ relation for FA and PDA?
- How many symbols in a configuration change in one step? (Answer: at most 3. See $uaq_ibv \vdash uq_jacv$ and $uaq_ibv \vdash uacq_jv$)

Accept, Reject, and Halting

- The start configuration of M on input w is q_0w.
- An accepting configuration of M is a configuration whose state is q_{accept} (i.e., $uq_{\text{accept}}v$).
- A rejecting configuration of M is a configuration whose state is q_{reject} (i.e., $uq_{\text{reject}}v$).
- Accepting and rejecting configurations are halting configurations and do not yield further configurations.
- A TM has 3 possible outcomes for each input w
  1. Accept w (enter q_{accept})
  2. Reject w (ener q_{reject})
  3. Reject w by looping (running forever)

Recognizable Languages

- For binary relation $\vdash$, let $\vdash^*$ be the reflexive transitive closure of $\vdash$.
- A Turing machine M accepts an input w if there is a sequence of configurations $C_1, C_2, \ldots, C_k$ such that
  - C_1 is the start configuration of M on input w;
  - each $C_i \vdash C_{i+1}$; and
  - C_k is an accepting configuration.
- The language of M or the language recognized by M (written $L(M)$) is thus

$$L(M) = \{w : M \text{ accepts } w\}.$$

or equivalently

$$L(M) = \{w : q_0 w \vdash^* u q_{\text{accept}} v\}.$$

Definition 2

A language is Turing-recognizable or recursively enumerable (abbrev. as *r.e.*) if some Turing machine recognizes it.

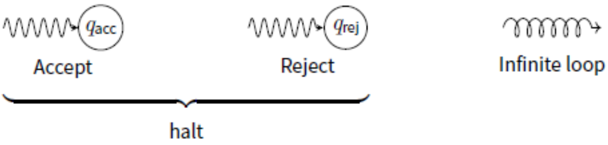
Decidable Languages

- When a Turing machine is processing an input, there are three outcomes: accept, reject, or loop.
  - "Loop" means it never enters a halting configuration.
- A deterministic finite automaton or deterministic pushdown automaton have only two outcomes: accept or reject.
- For a nondeterministic finite automaton or nondeterminsitic pushdown automaton, it can also loop.
  - "Loop" means it does not finish reading the input (ϵ-transitions).
- A Turing machine that halts on all inputs is called a decider.
- When a decider recognizes a language, we say it decides the language.

Definition 3

A language is Turing-decidable (decidable, or recursive) if some Turing machine decides it.

- A is T-recognizable if $A = L(M)$ for some TM M.
- A is T-decidable if $A = L(M)$ for some TM M that halts on all inputs.



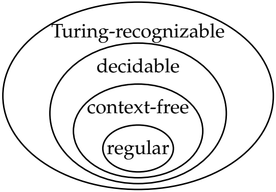
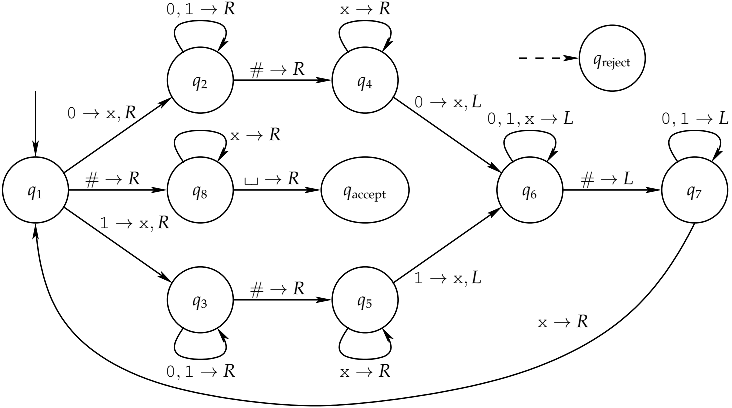


Figure: Relationship among Different Languages

Turing Machines – Example

- We now formally define $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$ which decides $B = \{w\#w : w \in \{\texttt{0}, \texttt{1}\}^*\}$.
- $Q = \{q_1, \ldots, q_{14}, q_{\text{accept}}, q_{\text{reject}}\}$;
- $\Sigma = \{\texttt{0}, \texttt{1}, \#\}$ and $\Gamma = \{\texttt{0}, \texttt{1}, \#, \texttt{x}, \sqcup\}$.



Turing Machines whose Heads can Stay

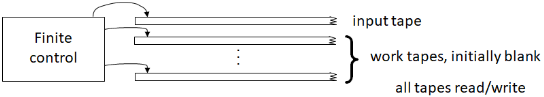
- Recall that the transition function of a Turing machine indicate whether its read-write head moves left or right.
- Consider a new Turing machine whose head can stay (i.e., a stationary move).
- Hence we have $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$.
- Is the new Turing machine more powerful?
- Of course not, we can always simulate S by an R and then an L.

Multitape Turing Machines

- Initially, the input appears on the tape 1.
- If a multitape Turing machine has k tapes, its transition function now becomes

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

- $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, d_1, \ldots, d_k)$ means that if the machine is in state q_i and reads a_i from tape i for $1 \leq i \leq k$, it goes to state q_j, writes b_i to tape i for $1 \leq i \leq k$, and moves the tape head i towards the direction d_i for $1 \leq i \leq k$.
- Are multitape Turing machines more powerful than signel-tape Turing machines?



Multitape Turing Machines

Theorem 4

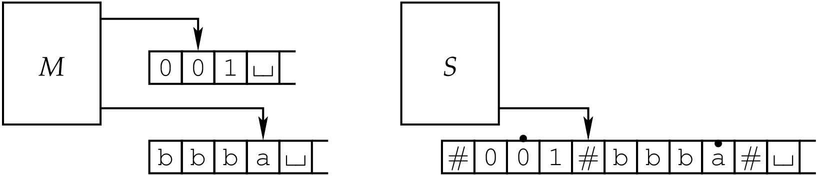
Every multitape Turing machine has an equivalent single-tape Turing machine.

Proof.

We use a special new symbol $\#$ to separate contents of k tapes. Moreover, k marks are used to record locations of the k virtual heads.

$S =$ "On input $w = w_1 w_2 \cdots w_n$:

1. Write w in the correct format: $\#\dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \cdots \#$.
2. Scan the tape and record all symbols under virtual heads. Then update the symbols and virtual heads by the transition function of the k-tape Turing machine.
3. If S moves a virtual head to the right onto a $\#$, S writes a blank symbol and shifts the tape contents from this cell to the rightmost $\#$ one cell to the right. Then S resumes simulation." □



- A "mark" is in fact a different tape symbol.
  - Say the tape alphabet of the multitape TM M is $\{\mathtt{0}, \mathtt{1}, \mathtt{a}, \mathtt{b}, \sqcup\}$.
  - Then S has the tape alphabet $\{\#, \mathtt{0}, \mathtt{1}, \mathtt{a}, \mathtt{b}, \sqcup, \dot{\mathtt{0}}, \dot{\mathtt{1}}, \dot{\mathtt{a}}, \dot{\mathtt{b}}, \dot{\sqcup}\}$.

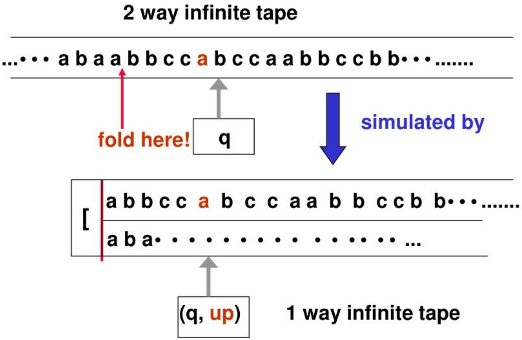
Corollary 5

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

Turing Machines with 2-way Infinite Tape

Theorem 6

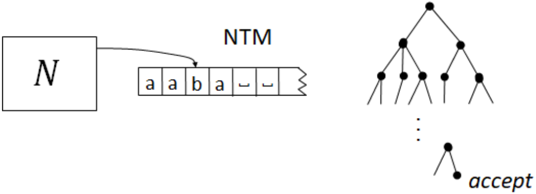
A TM with a 2-way infinite tape can be simulated by one with a 1-way infinite tape.



The new tape alphabet is $\Gamma \times \Gamma$, where Γ is the tape alphabet of the original TM.

Nondeterministic Turing Machines

- A nondeterministic Turing machine has its transition function of type $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$.
  - Equivalently, in some textbooks $\delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$.
  - $\delta(q, a) = \{(q_1, b_1, R), (q_2, b_2, L)\}$ is the same as $(q, a, q_1, b_1, R), (q, a, q_2, b_2, L) \in \delta$.
- Are nondeterministic Turing machines more powerful than deterministic Turing machines?
  - Recall that nondeterminism does not increase the expressive power in finite automata.
  - Yet nondeterminism does increase the expressive power in pushdown automata.



Nondeterministic Turing Machines

Theorem 7

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Proof.

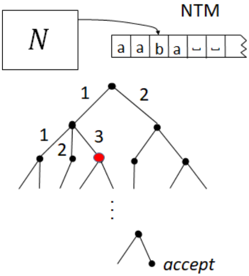
Nondeterministic computation can be seen as a tree. The root is the start configuration. The children of a tree node are all possible configurations yielded by the node. By ordering children of a node, we associate an address with each node. For instance, ϵ is the root; `1` is the first child of the root; `21` is the first child of the second child of the root.
We simulate an NTM N with a 3-tape DTM D. Tape 1 contains the input; tape 2 is the working space; and tape 3 records the address of the current configuration.
Let b be the maximal number of choices allowed in N. Define $\Sigma_b = \{\mathtt{1}, \mathtt{2}, \ldots, \mathtt{b}\}$. We now describe the Turing machine D.

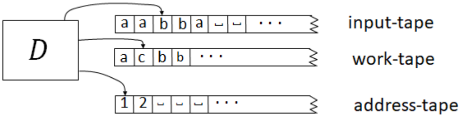
Nondeterministic Turing Machines

Proof.

1. Initially, tape 1 contains the input w; tape 2 and 3 are empty.
2. Copy tape 1 to tape 2.
3. Simulate N from the start state on tape 2 according to the address on tape 3.
   - When compute the next configuration, choose the transition by the next symbol on tape 3.
   - If no more symbol is on tape 3, the choice is invalid, or a rejecting configuration is yielded, go to step 4.
   - If an accepting configuration is yielded, accept the input.
4. Replace the string on tape 3 with the next string lexicographically and go to step 2. □

Nondeterministic Turing Machines





- In the computation tree, the red configuration can be encoded as "13".
- Basically, the simulation is to do a "breadth-first search" of the "possibly" infinite tree. Can we do "depth-first search" instead?

Nondeterministic Turing Machines

Corollary 8

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

- A nondeterministic Turing machine is a decider if all branches halt on all inputs.
- If the NTM N is a decider, a slight modification of the proof makes D always halt. (How?)

Corollary 9

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

Schematic of Enumerators

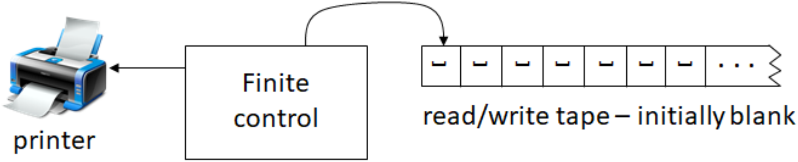


Figure: Schematic of Enumerators

(Fig. from M. Sipser's class notes)

- An enumerator is a Turing machine with a printer.
- An enumerator starts with a blank input tape.
- An enumerator outputs a string by sending it to the printer.
- The language enumerated by an enumerator is the set of strings printed by the enumerator.
  - Since an enumerator may not halt, it may output an infinite number of strings.
  - An enumerator may output the same string several times.

Enumerators for TM Recongnizable/Decidable Languages

Consider the lexicographical order $s_1, s_2, \ldots$ of Σ^*.
E.g., for $\Sigma = \{0, 1\}$, the sequence $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots$.

- L is a TM decidable language $\Leftrightarrow$ an Enumerator E generates L in lexicographical order. E.g. E outputs $\epsilon, 1, 001, 1011, 010011, \ldots$.
  - $\Rightarrow$ E simulates TM M for strings in lexicographical order until halting. If accepts, outputs the string.
  - $\Leftarrow$ On input w, TM M simulates E until (1) E generates w, then accepts; or (2) E generates a string "following" w in lex. order, then rejects.
- L is a TM recognizable language $\Leftrightarrow$ an Enumerator E generates L. E.g. E outputs $010011, (10)^{1000}, \epsilon, 1011, 1, 001, \ldots$.
  - $\Rightarrow$ Note that the set $\{(i, j) \mid i, j \in \mathbb{N}\}$ is countable. When dealing with (i, j), E simulates string s_i for j steps. If accepts, outputs s_i.
    Question: Can't E simulate s_i directly?
  - $\Leftarrow$ On input w, M simulates E. If E outputs w, M accepts.

Enumerators

Theorem 10

A language is Turing-decidable if and only if some enumerator enumerates it in lexicographical order.

Proof.

Let E be an enumerator. Consider the following TM M:
$M =$ "On input w :

1. Run E and compare each generated output string with w.
2. Accept if E ever outputs w; reject if E outputs a w' with $w < w'$"

Conversely, let M be a TM deciding A, and assume that $\Sigma = \{0, 1\}$.
$E =$ "Ignore the input.

1. Repeat for $w = \epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$
   1. Run M on w;
   2. If M accepts w, output s_j;
   3. If M rejects w, exit

□

Enumerators

Theorem 11

A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof.

Let E be an enumerator. Consider the following TM M:

$M =$ "On input w :

1. Run E and compare any output string with w.
2. Accept if E ever outputs w."

Conversely, let M be a TM recognizing A. Consider

$E =$ "Ignore the input.

1. Repeat for $i = 1, 2, \ldots$
   1. Let $s_1, s_2, \ldots, s_i$ be the first i strings in Σ^* (say, lexicographically).
   2. Run M for i steps on each of $s_1, s_2, \ldots, s_i$.
   3. If M accepts s_j for $1 \leq j \leq i$, output s_j.

□

Algorithms

- Let us suppose we lived before the invention of computers.
  - say, circa 300 BC, around the time of Euclid.
- Consider the following problem:
  Given two positive integers a and b, find the largest integer r such that r divides a and r divides b, i.e., finding the greatest common divisor (GCD).
- How do we "find" such an integer?
- Euclid's method is in fact an algorithm.
  - $GCD(A, B) = GCD(B, R)$, where R the remainder of A divided by B.
  - $GCD(35, 30) = GCD(30, 5)$.
- Keep in mind that the concept of algorithms has been in mathematics long before the advent of computer science.

Hilbert's Problems

- Mathematician David Hilbert listed 23 problems in 1900.
  - (#1) Problem of the continuum (Does set A exist where $|\mathbb{N}| < |A| < |\mathbb{R}|$?).
  - (#10) Give an algorithm for solving Diophantine equations.
    - Example: $3x^2 - 2xy - y^2z = 7$; solution: $x = 1, y = 2, z = -2$
    - Goal: devise "a process according to which it can be determined by a finite number of operations," that tests whether a polynomial has an integral root.
- If such an algorithm exists, we just need to invent it.
- What if there is no such algorithm?
  - How can we argue Hilbert's 10th problem has no solution?
- We need a precise definition of algorithms!

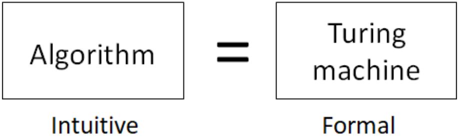
Church-Turing Thesis

- In 1936, two papers came up with definitions of algorithms.
- Alonzo Church used λ-calculus to define algorithms.
  - If you don't know λ-calculus, take Programming Languages.
- Alan Turing used Turing machines to define algorithms.
  - If you don't know TM now, please consider dropping this course.
- It turns out that both definitions are equivalent!
- The connection between the informal concept of algorithms and the formal definitions is called the Church-Turing thesis.

Hilbert's 10th Problem

- In 1970, Yuri Matijasevič showed that Hilbert's 10th problem is not solvable.
  - That is, there is no algorithm for testing whether a polynomial has an integral root.
- Define $D = \{p : p \text{ is a polynomial with an integral root}\}$.
- Consider the following TM:
  $M =$ "The input is a polynomial p over variables $x_1, x_2, \ldots, x_k$
  1. Evaluate p on an enumeration of k-tuple of integers.
  2. If p ever evaluates to 0, accept."
- M recognizes D but does not decide D.



Encodings of Turing Machines

To represent a Turing machine

$$M = (Q, \{0, 1\}, \Gamma, \delta, q_1, \sqcup, F)$$

as a binary string, we must first assign integers to the states, tape symbols, and directions L and R:

- Assume the states are $q_1, q_2, ..., q_r$ for some r. The start state is q_1, and the only accepting state is q_2.
- Assume the tape symbols are $X_1, X_2, ..., X_s$ for some s. Then: $0 = X_1, 1 = X_2$, and $\sqcup = X_3$.
- $L = D_1$ and $R = D_2$.
- Encode the transition rule $\delta(q_i, X_j) = (q_k, X_l, D_m)$ by $0^i 1 0^j 1 0^k 1 0^l 1 0^m$. Note that there are no two consecutive 1s.
- Encode an entire Turing machine by concatenating, in any order, the codes C_i of its transition rules, separated by $11 : C_1 11 C_2 11 \cdots C_{n-1} 11 C_n$.

Example

$M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_1, \sqcup, \{q2\})$ with $\delta(q_1, 1) = (q_3, 0, R)$, $\delta(q_3, 0) = (q_1, 1, R)$, $\delta(q_3, 1) = (q_2, 0, R)$, and $\delta(q_3, \sqcup) = (q_3, 1, L)$.

- Codes for the transition rules:
  $\overbrace{0}^{q_1} \; 1 \; \overbrace{00}^{1=X_2} \; 1 \; \overbrace{000}^{q_3} \; 1 \; \overbrace{0}^{0} \; 1 \; \overbrace{00}^{R=D_2}$ 0001010100100
  00010010010100 0001000100010010
- Code for M: 01001000101001100010101001001100010010010100110001000100010010

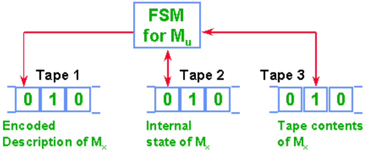
Given a Turing machine M with code w_i , we can now associate an integer to it: M is the ith Turing machine, referred to as M_i. Many integers do no correspond to any Turing machine at all. Examples: 11001 and 001110.
If w_i is not a valid TM code, then we shall take M_i to be the Turing machine (with one state and no transitions) that immediately halts on any input. Hence $L(M_i) = \emptyset$ if w_i is not a valid TM code.

Acceptance Problem for TM's

- Notation: $\langle O_1, O_2, \ldots, O_k \rangle$ encodes objects $O_1, O_2, \ldots, O_k$ as a single string. E.g., $\langle 0011, 10111 \rangle$ can be represented as $0011\#10111$.
- Consider

$$A_{\text{TM}} = \{\langle M, w \rangle : M \text{ is a TM and } M \text{ accepts } w\}$$

- Consider the following TM:
  $U =$ "On input $\langle M, w \rangle$ where M is a TM and w is a string:
  1. Simulate M on the input w.
  2. If M enters its accept state, accept; if M enters its reject state, reject."
- Does U decide A_{TM}? Why not?
- The TM U is called the universal Turing machine.



(Fig. from https://people.csail.mit.edu/devadas/6.004/Lectures/lect13/sld012.htm)

Counting Arguments

- Recall that $|\mathbb{N}| = |\mathbb{Z}| = |\Sigma^*| = \aleph_0$ (Σ is finite).
- Also recall that $|\mathcal{P}(\Sigma^*)| > \aleph_0$.
  - In fact, any subset of Σ^* can be uniquely represented as an infinite string of 0's and 1's. E.g. $\{\epsilon, b, ba, aab, \ldots\} \subseteq \{a, b\}^*$ corresponds to

    $$\overbrace{1}^{\epsilon}\ 0\ \overbrace{1}^{b}\ 00\ \overbrace{1}^{ba}\ 00\ \overbrace{1}^{aab}\ \ldots.$$

    Note that the lex. order of $\{a, b\}^*$ is $\epsilon, a, b, aa, ab, ba, bb, aaa, aab\ldots$

Corollary 12

Some languages are not Turing-recognizable.

Proof.

The set of all Turing machines is countable since each TM M has an encoding $\langle M \rangle$ in Σ^*.
The set of all languages over Σ is $\mathcal{P}(\Sigma^*)$ and hence is uncountable. □

- Can we find a concrete example?

Undecidability of the Acceptance Problem for TM's

Theorem 13

$A_{TM} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ accepts } w\}$ *is not a decidable language.*

Proof.

The proof is by contradiction. Suppose there is a TM H deciding A_{TM}. That is,

$$H(\langle M, w\rangle) = \begin{cases} \text{accept} & \text{if } M \text{ accepts } w \\ \text{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

Consider the following TM:
D = "On input $\langle M\rangle$ where M is a TM:

1. Run H on the input $\langle M, \langle M\rangle\rangle$.
2. If H accepts, reject. If H rejects, accept."

Consider

$$D(\langle D\rangle) = \begin{cases} \text{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \text{reject} & \text{if } D \text{ accepts } \langle D\rangle \end{cases}$$

A contradiction. □

The above proof uses the diagonalization method.

All TM descriptions:

All TMs ⇩	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\ldots$	$\langle D \rangle$
M_1	acc					
M_2		rej				
M_3			acc			
M_4				acc		
$\vdots$						
D						

A Turing-unrecognizable Language

- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language.

Theorem 14

A language is decidable if and only if it is Turing-recognizable and co-Turing-recognizable.

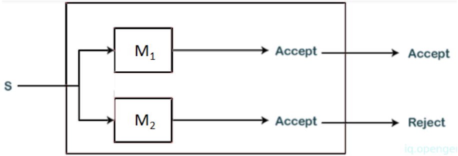
Proof.

If A is decidable, then A and $\overline{A}$ are both recognizable. Since $\overline{\overline{A}} = A$, A is Turing-recognizable and co-Turing-recognizable.
Now suppose A and $\overline{A}$ are Turing-recognizable by M_1 and M_2 respectively. Consider
$M =$ "On input w:

1. Run both M_1 and M_2 on the input w in parallel.
2. If M_1 accepts, accept; if M_2 accepts; reject."

□

How to Run Two Turing Machines in Parallel?



- Suppose $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, \sqcup, F_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, \sqcup, F_2)$.
- M has three tapes. Tape 1 (resp., Tape 2) serves as the work tape of M_1 (resp., M_2), Tape 3 contains the input w.
- $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ where $Q = Q_1 \times Q_2 \times \{1, 2\}$, $q_0 = (q_1, q_2, 1)$, ...
- On input w, M first copies w from tape 3 to both tape 1 and tape 2.
- A run of M is of the form $(q_1, q_2, 1) \to (p_1, q_2, 2) \to (p_1, p_2, 1) \to (-, -, 2)$, which alternates between the executions of M_1 and M_2.
- Can M run M_1 on w, then M_2 on w?

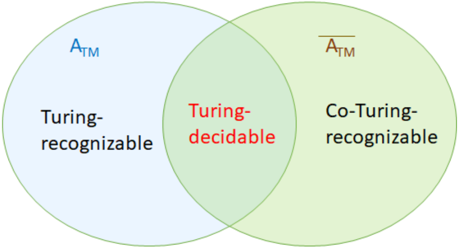
A Turing-unrecognizable Language

Corollary 15

$\overline{A_{TM}}$ is not Turing-recognizable.

Proof.

A_{TM} is Turing-recognizable. If $\overline{A_{TM}}$ is Turing-recognizable, A_{TM} is both Turing-recognizable and co-Turing-recognizable. By Theorem 14, A_{TM} is decidable. A contradiction. □



Turing-recognizable and Decidable Languages

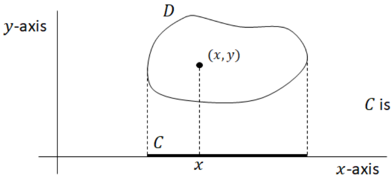
Theorem 16

Language C is Turing-recognizable $\Leftrightarrow$ there is a decidable language D such that $C = \{x \mid \exists y, \langle x, y \rangle \in D, x, y \in \Sigma^*\}$

Proof.

($\Rightarrow$) Let M be a TM accepting C. Define $D = \{\langle x, y \rangle \mid M \text{ accepts } x \text{ in } y \text{ steps}\}$, which is clearly decidable. Furthermore, $x \in L(M) \Leftrightarrow \exists y, \langle x, y \rangle \in D$.

($\Leftarrow$) Let N be a decider for D. Consider TM M, on input x, guesses a y, runs N to check whether $\langle x, y \rangle \in D$; if N accepts, accepts. □



C is a "projection" of D