TURING MACHINES

Definition of Turing machine

Turing machines are believed to be the most powerful generalization of dfa/nfa automata that is possible. Modern computers are, in principle, reducable to Turing machines. The definition of the Turing machine is done in 3 steps:

- 1 We define the machine itself
- 2 We define the "tape", i.e. the input/output device used by the machine
- 3 We define the process by which the machine converts its input into output.

Def: A Turing machine M is defined as $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where:

a) Q is a finite set of internal states
b) Σ is a finite set, the input alphabet
c) Γ is a finite set, the tape alphabet
d) δ is a transition function $\delta: Q \times \Gamma \to [Q \times \Gamma \times \{L, R\}] \cup \{H\}$ where L, R, H are fixed symbols.
e) $q_0 \in Q$ is an initial state.
f) $B \in \Gamma$ is the blank symbol
g) $F \subseteq Q$ is a set of final states

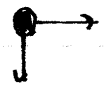
such that $\Sigma \subseteq \Gamma - \{B\}$ and $\forall q \in F: \forall a \in \Gamma: \delta(q, a) = H$

notation : $Tur(\Sigma)$ will denote the set of all Turing machines that can be defined on some input alphabet Σ.

Remarks

a) The symbols R, L are instructions to the attached tape device (to be defined below) that instruct the header reading the tape to move right or left. The symbol H corresponds to a command to halt the computation.

b) It is assumed that if the machine finds itself in a final state $q \in F$, then the transition mapping δ will halt the computation. However it is not necessary to reach a final state for the computation to halt.



The tape device

Attached to a Turing machine is an input/output device that we will call "tape". Informally, we envision the tape as follows:

a) The tape is a one-dimensional storage device of symbols from Γ with infinite length in either direction.

b) The Turing machine itself is envisioned as a header that points to some symbol somewhere on the tape. The header also has an internal state $q \in Q$.

c) If $q \in Q$ is the state of the Turing machine and $a \in \Gamma$ the symbol on the tape currently under the

machine the

1) $\delta(q,a) = (p,b,L)$ means that the machine will replace a with b on the tape, transition from state q to p, then move the machine left.
2) $\delta(q,a) = (p,B,R)$ means that the machine will replace a with B on the tape, transition from state q to p, then move the machine right
3) $\delta(q,a) = H$ means that the machine terminates the algorithm

Def: Let $M = (Q,\Sigma,\Gamma,\delta,q_0,B,F) \in Tur(\Sigma)$ be a Turing machine

a) A string $u = XqaY$ with $X,Y \in \Gamma^*$ and $q \in Q$ and $a \in \Gamma$ is a <u>configuration</u> of the Turing machine where the tape content is XaY and the header is pointing at the character a while in internal state q.

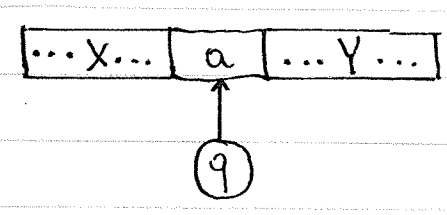
b) A string $u = XaY$ with $X,Y \in \Gamma^*$ and $a \in \Gamma$ is a configuration of the Turing machine, where the tape content is XaY and the machine is in a <u>halted</u> state (i.e. the header is unmounted from the tape)

c) The set of all possible configurations is:

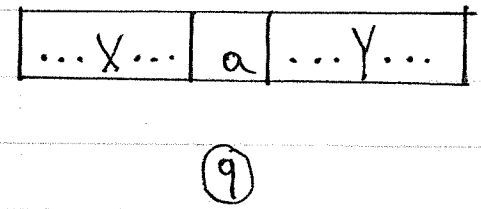
$$config(M) = \{XqaY, XaY \mid X,Y \in \Gamma^* \wedge q \in Q \wedge a \in \Gamma\}$$
$$= (\Gamma^* Q \Gamma^+) \cup (\Gamma^+)$$

Remark: The configurations described above can be visually represented as follows:

$\$ = XqaY$



$\$ = XaY$



The operation of the Turing machine is defined formally via a deterministic transition function $\Delta : config(M) \to config(M)$ as follows:

Def: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \in Tur(\Sigma)$ be a Turing machine. We define:

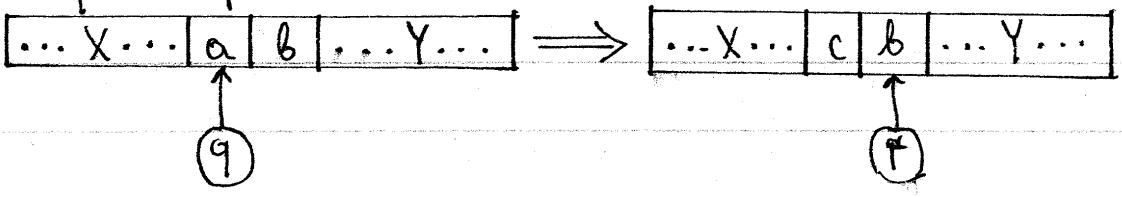
the deterministic <u>configuration transition function</u> $\Delta : config(M) \to config(M)$ as follows:

$$\forall X, Y \in \Gamma^* : \forall q \in Q : \forall a, b \in \Gamma :$$
$$: \begin{cases} \delta(q,a) = (p,c,R) \Rightarrow \Delta(XqabY) = XcpbY \\ \delta(q,a) = (p,c,L) \Rightarrow \Delta(XaqbY) = XpacY \end{cases}$$
$$\forall X \in \Gamma^* : \forall q \in Q : \forall a \in \Gamma :$$
$$: \begin{cases} \delta(q,a) = (p,c,R) \Rightarrow \Delta(Xqa) = XcpB \\ \delta(q,a) = (p,c,L) \Rightarrow \Delta(qaX) = pBcX \end{cases}$$
$$\forall X, Y \in \Gamma^* : \forall q \in Q : \forall a \in \Gamma : \delta(q,a) = H \Rightarrow \Delta(XqaY) = XaY$$
$$\forall P \in \Gamma^+ : \Delta(P) = P$$

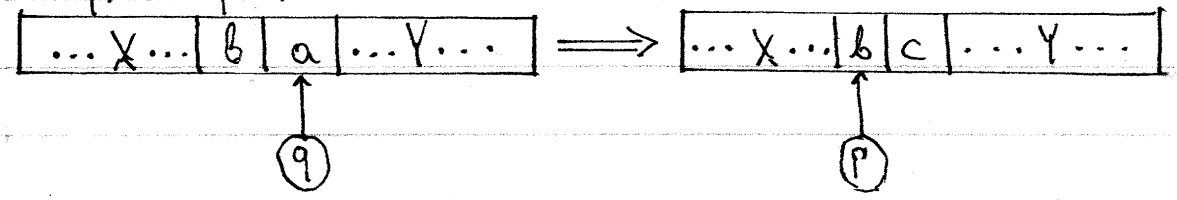
Remark: Note that we distinguish between configurations where the Turing machine is scanning the beginning or end of the tape string vs configurations where the Turing machine is scanning the interior of the tape string. In the first case it becomes necessary to utilize the "blank" character B. Once the machine is halted, the configuration transition function Δ merely returns the content of the tape.

Remark: A graphical representation of the transitions accounted by the previous definition is given below:

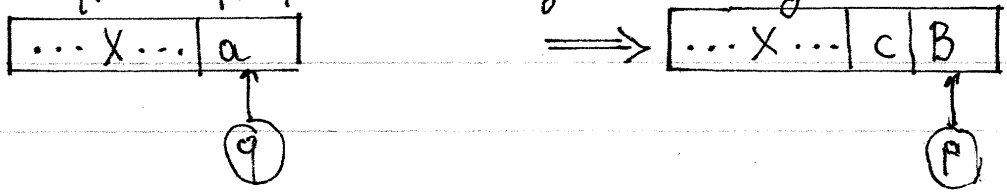
$\delta(q,a) = (p,c,R)$ — interior case

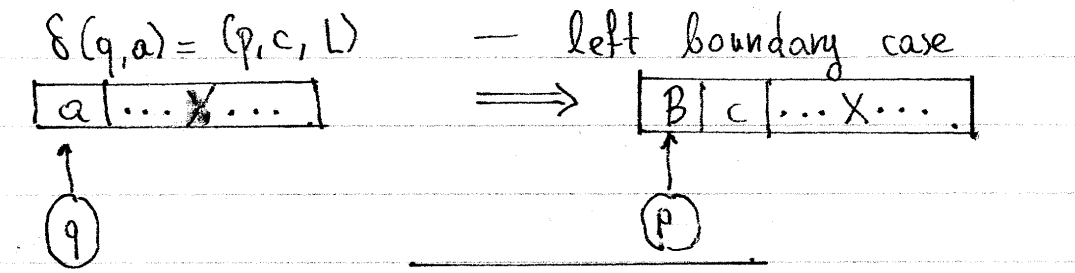


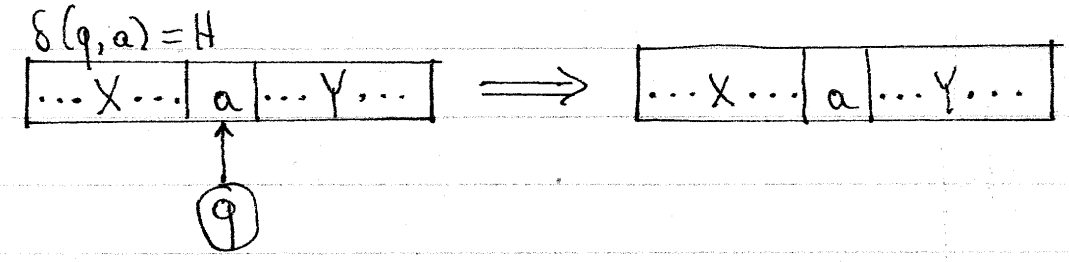
$\delta(q,a) = (p,c,L)$ — interior case



$\delta(q,a) = (p,c,R)$ — right boundary







Configuration transitions and halting states

Turing machines are deterministic. Consequently, given an initial configuration, all subsequent configurations are predetermined. Eventually, the machine may reach a halting state. It is also possible that the machine never reaches a halting state or may even find itself in an infinite loop. We give the relevant definitions below:

Def: Let $M \in Tur(\Sigma)$ be a Turing machine with configuration transition function $\Delta : config(M) \to config(M)$. Let $n \in \mathbb{N}^*$. We define the n-step configuration transition function $\Delta_n : config(M) \to config(M)$ recursively as:

$\forall \$ \in config(M): \Delta_1(\$) = \Delta(\$)$

$\forall n \in \mathbb{N} - \{0,1\}: \forall \$ \in config(M): \Delta_n(\$) = \Delta(\Delta_{n-1}(\$))$

Note that a halted configuration $\$ \in \text{config}(M)$ will satisfy $\$ \in \Gamma^+$, whereas a non-halted configuration will satisfy $\$ \in \Gamma^* Q \Gamma^+$. We may therefore give the following definitions:

Def: Let $M \in \text{Tur}(\Sigma)$ be a Turing machine and let $\$_1, \$_2 \in \text{config}(M)$ be two machine configurations. We say that

$$\$_1 \vdash \$_2 \iff \Delta(\$_1) = \$_2$$
$$\$_1 \vdash^* \$_2 \iff \exists n \in \mathbb{N}^* : \Delta^n(\$_1) = \$_2$$
$$\$_1 \vdash^* \infty \iff \forall n \in \mathbb{N}^* : \Delta^n(\$_1) \notin \Gamma^*$$
$$\$_1 \vdash^* \text{loop} \iff \exists n, m \in \mathbb{N}^* : (n \neq m \wedge \Delta^n(\$_1) = \Delta^m(\$_1))$$

interpretation

a) $\$_1 \vdash \$_2$ means that the machine will transition from configuration $\$_1$ to configuration $\$_2$ in one step.

b) $\$_1 \vdash^* \$_2$ means that the machine will transition from configuration $\$_1$ to configuration $\$_2$ in a finite number of steps.

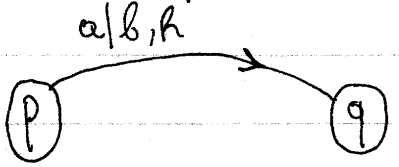
c) $\$_1 \vdash^* \infty$ means that once initialized with the configuration $\$_1$, the machine will never reach a halt state in subsequent steps.

d) $\$_1 \vdash^* \text{loop}$ means that once initialized with the configuration $\$_1$, the machine will enter an infinite loop where it cycles through a finite sequence of configurations infinite times.

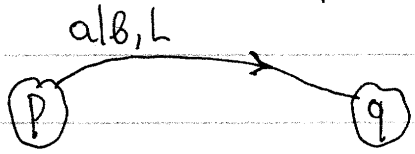
Graph representation of Turing machines

Turing machines can be represented as directed graphs according to the following conventions:

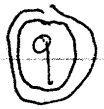
1) For every transition rule $\delta(p,a) = (q,b,R)$ we have the representation



Likewise, for the transition rule $\delta(p,a) = (q,b,L)$ we have the representation



2) Final internal states are denoted via a double oval as:



and according to the Turing machine definition, they have no outgoing arrows

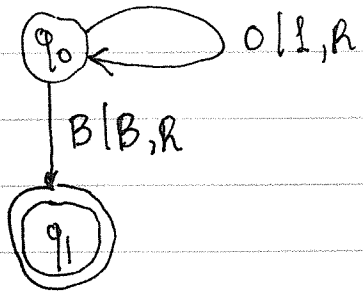
3) Transitions of the form $\delta(p,a) = H$ are not shown graphically. The absense of a needed outgoing arrow indicates that the machine will halt.

EXAMPLE

Consider a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ with $Q = \{q_0, q_1\}$ and $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, B\}$ and $F = \{q_1\}$ and transition rules

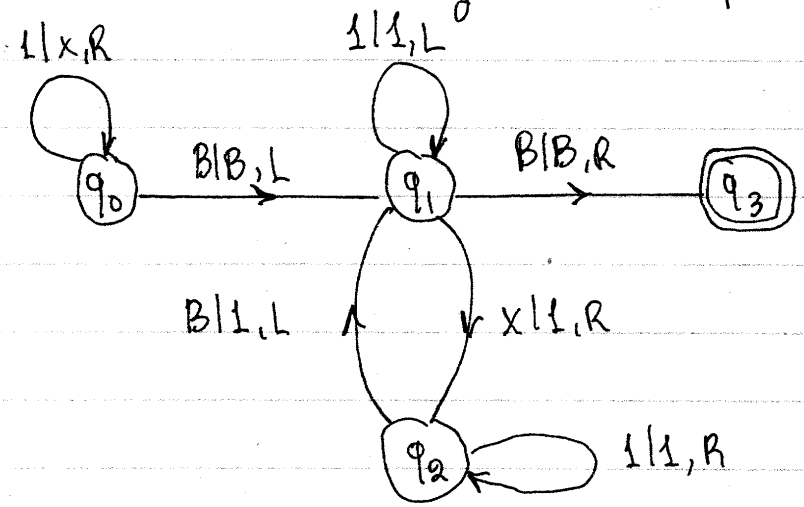
$$\delta(q_0, 0) = (q_0, 1, R)$$
$$\delta(q_0, B) = (q_1, B, R)$$

The corresponding graphical representation is



EXERCISES

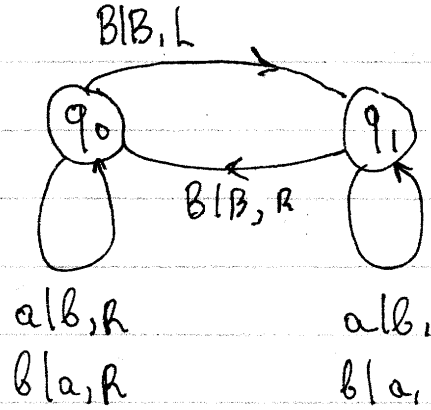
(1) Consider the Turing machine represented by:



a) Give the corresponding set theoretic definition of this machine.

b) Consider the following initial configurations:
i) $\$ = q_0 11$ ii) $\$ = q_0 111$ iii) $\$ = q_0 1111$
Trace the calculation of the Turing machine from these initial configurations until it halts.

(2) Consider a Turing machine with graphical representation:



a) Give a set theoretic definition of this machine

b) Trace the first 20 steps of the machine configuration from initial state $\xi = q_0 aaba$

c) Explain why this machine never halts for any initial configuration $\xi = q_0 u$ with $u \in \{a,b\}^*$ any string.

(3) Let $M \in Tur(\Sigma)$ be a Turing machine with configuration transition function $\Delta_n : config(M) \to config(M)$ with $n \in \mathbb{N}^*$. Show that

a) $\forall n,m \in \mathbb{N}^* : \Delta_n(\Delta_m(\xi)) = \Delta_{n+m}(\xi)$

b) $\forall \xi_1, \xi_2 \in config(M) : (\xi_1 \vdash^* \xi_2 \wedge \xi_2 \vdash^* \xi_3 \Rightarrow \xi_1 \vdash^* \xi_3)$

c) $\forall \xi \in config(M) : (\xi \vdash^* loop \Rightarrow \xi \vdash^* \infty)$

Turing machines as language acceptors

Turing machines can be used as language acceptors, analogously with dfas and nfas.

Def: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing machine $M \in Tur(\Sigma)$. The language accepted by M is:
$\mathcal{L}(M) = \{u \in \Sigma^* \mid \exists n \in \mathbb{N}^* : \exists q \in F : \exists x, y \in \Gamma^* : \Delta^n(q_0 u) = x q y\}$.

interpretation: We begin with a candidate string $u \in \Sigma^*$, and place the Turing machine on the leftmost character of u with q_0 as the initial internal state of the Turing machine. The string u is accepted if and only if after n steps the Turing computation terminates with the Turing machine in a final state. When $u \notin \mathcal{L}(M)$, the Turing machine could terminate in a non-final internal state or never terminate.

EXAMPLE

Given the alphabet $\Sigma = \{a, b\}$, implement a Turing machine that accepts the language

$$L = \{a^n b^n \mid n \in \mathbb{N}^+\}$$

Solution

▸ Strategy: Starting at leftmost a, we mark it by replacing it with x. Then we move to leftmost b and mark it with y. We move back left and look for the leftmost a and repeat. Eventually, when we cannot find any leftmost a, we should not be able to find a leftmost b either. To illustrate this process, the corresponding string modifications will look like:

$$aaabbb \rightarrow xaabbb \rightarrow xaaybb \rightarrow xxaybb \rightarrow$$
$$\rightarrow xxayyb \rightarrow xxxyyb \rightarrow xxxyyy.$$

▸ Implementation

•1 We use the following tape alphabet $\Gamma = \{a, b, x, y, B\}$.

•2 The following transitions replace the leftmost a with x and search for leftmost b by moving the machine right.

$\delta(q_0, a) = (q_1, x, R)$ // If 1st character is a, replace with x, move right. Go to state q_1 to find the leftmost b

// Skip all a, y characters while searching for the
// leftmost B

$\delta(q_1, a) = (q_1, a, R)$

$\delta(q_1, y) = (q_1, y, R)$

// When we find the leftmost B, mark it as y and
// go to state q_2 to search for next leftmost a

$\delta(q_1, B) = (q_2, y, L)$

•3 The following transitions implement searching for the next "a" by moving the machine left.

// Moving left, skip all "y" and "a" characters

$\delta(q_2, a) = (q_2, a, L)$

$\delta(q_2, y) = (q_2, y, L)$

// When we find the first x, move right to return to the leftmost "a". Go to state q_0 in order to repeat the whole process

$\delta(q_2, x) = (q_0, x, R)$

// It is assumed that after the above transition is executed the machine is scanning the "a" character. However, if we have exhausted all "a", then it will be scanning a "y" character instead. In that case, we enter a new mode q_3 and move right to verify that we have exhausted all "b" characters.

$\delta(q_0, y) = (q_3, y, R)$

$\delta(q_3, y) = (q_3, y, R)$

// Encountering a blank means that we exhausted all "b" characters so we go to final state q_4 and halt

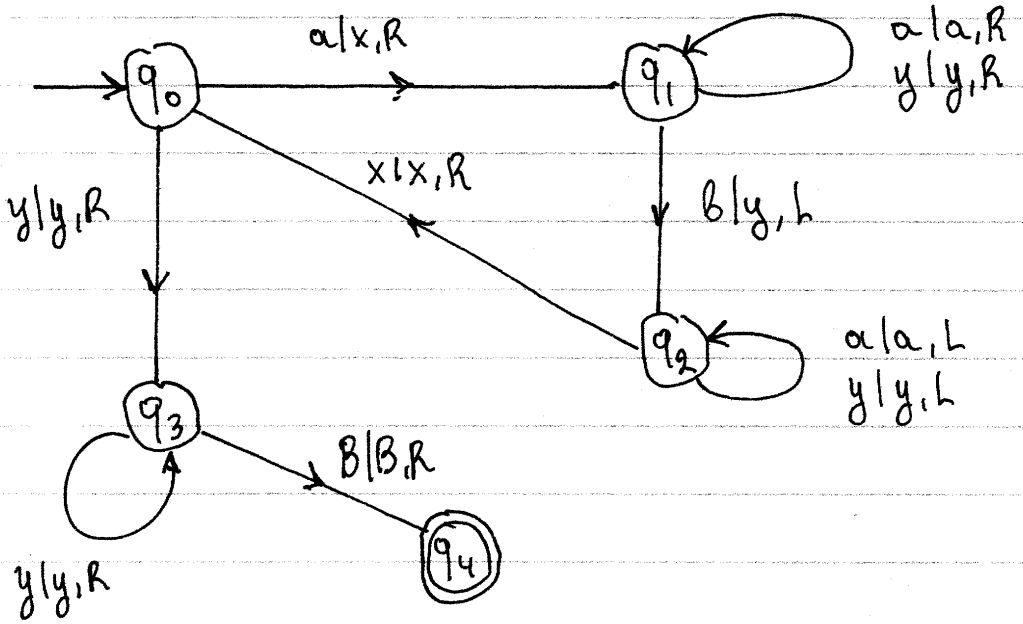
$\delta(q_3, B) = (q_4, B, R)$

- It follows that

  $Q = \{q_0, q_1, q_2, q_3, q_4\}$

  $F = \{q_4\}$

  with q_4 being the halting state.
- A graphical representation of this machine is as follows:



Remarks

a) Note that q_3 has missing outgoing edges for a, b, x. If such characters are encountered, the machine halts in q_3 and since $q_3 \notin F$ the overall string is rejected.

b) q_1 and q_2 have two loops, but for convenience we show them as one loop with 2 labels. Implicitly each label corresponds with a different loop.

c) In describing a Turing machine, just like with any other programming language, you should COMMENT YOUR CODE.

EXERCISES

④ Design and implement Turing machines that accept the following languages. Use both commented source code explicitly defining the Turing machine via set theory and also the corresponding graph representation.

a) $L = \{a, a^2b, ab^2\}$

b) $L = \{ab^n \mid n \in \mathbb{N}^*\}$

c) $L = \{w \in \{a,b\}^* \mid \exists k \in \mathbb{N}^* : |w| = 2k\}$

d) $L = \{w \in \{a,b\}^* \mid \exists k \in \mathbb{N}^* : |w| = 3k+1\}$

e) $L = \{a^n b^n c^n \mid n \in \mathbb{N}^*\}$

f) $L = \{a^n b^{2n} \mid n \in \mathbb{N}^*\}$

g) $L = \{a^n b^m a^{n+m} \mid n, m \in \mathbb{N}^*\}$

h) $L = \{u \in \{a,b\}^* \mid n_a(u) = n_b(u)\}$

Recursively enumerable and recursive languages

Def: Let Σ be an alphabet and $L \subseteq \Sigma^*$ a language. We say that

L **recursively enumerable** $\Leftrightarrow \exists M \in Tur(\Sigma) : \mathcal{L}(M) = L$

The problem with this definition is that when a string $u \notin \mathcal{L}(M)$, it may throw the Turing machine into a computation that never terminates. This motivates the following stricter definition:

Def: Let Σ be an alphabet and let $L \subseteq \Sigma^*$ be a language. We say that

L recursive $\Leftrightarrow \exists M \in Tur(\Sigma) : \begin{cases} \mathcal{L}(M) = L \\ \forall u \in \Sigma^+ : \overline{q_0 u \vdash \infty} \end{cases}$

In a recursive language we make the demand that the Turing machine M that can accept the language L should halt in a finite number of steps when it is given non-empty strings that do not belong to L. Thus the machine M will be able to tell us, for all $u \in \Sigma^+$, whether or not they belong to L with a finite number of steps.

Remarks

a) It is obvious that

$\forall L \in \mathcal{P}(\Sigma^*)$: (L recursive $\Rightarrow$ L recursively enumerable)

However there is a counterexample to the converse statement.

b) We will show below that there are languages in $\mathcal{P}(\Sigma^*)$ that are not recursively enumerable. This means that the Turing machine is not powerful enough to account for all languages in $\mathcal{P}(\Sigma^*)$.

c) On the other hand, according to the Church-Turing hypothesis, there are no possible modifications that can be made to the Turing machine definition to create a more powerful machine that can accept all recursively enumerable languages in addition to languages that are not recursively enumerable

d) The following modifications fail to make the machine more powerful:

i) Using a two-dimensional (or multi-dimensional) tape

ii) Adding a <u>stay</u> S operation, in addition to the <u>right</u> R and <u>left</u> L operations.

iii) Adding multiple tape devices, one-dimensional or multi-dimensional.

iv) Making the Turing machine non-deterministic (analogously to nfas vs. dfas)

or any combination of the above.

Limitations of Turing machines

- It takes a finite sequence of symbols to define every Turing machine. These symbols can be encoded as a binary string $u \in \{0,1\}^*$. We can thus construct a bijection from $Tur(\Sigma)$ to $\{0,1\}^*$ and conclude that

$$Tur(\Sigma) \sim \{0,1\}^* \qquad (1)$$

- We also know that

$$\{0,1\}^* = \bigcup_{n \in \mathbb{N}} \{0,1\}^n \Rightarrow \{0,1\}^* \text{ countably infinite}$$

$$\Rightarrow \{0,1\}^* \sim \mathbb{N} \qquad (2)$$

because $\{0,1\}^*$ is a countable union of finite sets.

- Likewise, for any finite alphabet Σ, we can show that

$$\Sigma^* \sim \mathbb{N} \qquad (3)$$

- From the above statements we can now prove the existence of languages that are not recursively enumerable.

> **Thm:** Given a finite alphabet Σ
> $\exists L \in \mathcal{P}(\Sigma^*) : L$ not recursively enumerable

Proof

To show a contradiction, we assume the negation of the claim that:

$$\forall L \in \mathcal{P}(\Sigma^*) : L \text{ recursively enumerable}$$
$$\Rightarrow \forall L \in \mathcal{P}(\Sigma^*) : \exists M \in Tur(\Sigma) : L = \mathcal{L}(M)$$

This statement allows us to define a mapping $f: \mathcal{P}(\Sigma^*) \to Tur(\Sigma)$ such that for every language $L \in \mathcal{P}(\Sigma^*)$, $f(L)$ is a Turing machine such that $\mathcal{L}(f(L)) = L$. It is easy to show that f is one-to-one (i.e. the same machine cannot accept two different languages simultaneously). It follows that:

$$\begin{aligned}
\Sigma^* &< \mathcal{P}(\Sigma^*) && [\text{Cantor's theorem}] \\
&\leq Tur(\Sigma) && [f \text{ one-to-one}] \\
&\sim \{0,1\}^* && [\text{Eq. (1)}] \\
&\sim \mathbb{N} && [\text{Eq. (2)}] \\
&\sim \Sigma^* && [\text{Eq (3)}]
\end{aligned}$$

$$\Rightarrow \Sigma^* < \Sigma^* \Rightarrow \Sigma^* \not\sim \Sigma^*$$

which is a contradiction, since $\Sigma^* = \Sigma^* \Rightarrow \Sigma^* \sim \Sigma^*$.
It follows that

$\exists L \in \mathcal{P}(\Sigma^*) : L$ not recursively enumerable. □

This theorem establishes the existence of languages that are not recursively enumerable and exposes the underlying problem: the set $\mathcal{P}(\Sigma^*)$ of all possible languages is uncountable whereas the set $Tur(\Sigma)$ of all possible Turing machines is countable. As a result, we cannot construct a distinct Turing machine for every language in $\mathcal{P}(\Sigma^*)$. Another way to show this is to construct an example of a specific language that is not recursively enumerable. This can be done as follows:

Construction: Let $\Sigma = \{x\}$ and consider the set $Tur(\Sigma)$ of all Turing machines. Since $Tur(\Sigma) \sim \mathbb{N}$, let $M_0, M_1, M_2, \ldots$ be an enumeration of all possible Turing machines. We define the language

$$L = \{ x^a \mid a \in \mathbb{N} \wedge x^a \notin \mathcal{L}(M_a) \}$$

Then L is NOT recursively enumerable.

Proof

To show that L is not recursively enumerable, we assume that L is recursively enumerable in order to derive a contradiction. Then:

$$L \text{ recursively enumerable} \Rightarrow \exists M \in Tur(\Sigma) : L = \mathcal{L}(M)$$
$$\Rightarrow \exists a \in \mathbb{N} : L = \mathcal{L}(M_a)$$

Choose a $b \in \mathbb{N}$ such that $L = \mathcal{L}(M_b)$. We distinguish between the following cases:

Case 1: Assume that $x^b \in L$. Then:

$$x^b \in L \Rightarrow x^b \in \{ x^a \mid a \in \mathbb{N} \wedge x^a \notin \mathcal{L}(M_a) \} \quad [\text{Def of } L]$$
$$\Rightarrow b \in \mathbb{N} \wedge x^b \notin \mathcal{L}(M_b)$$
$$\Rightarrow x^b \notin \mathcal{L}(M_b)$$
$$\Rightarrow x^b \notin L \quad [\text{via } L = \mathcal{L}(M_b)]$$

which is a contradiction, therefore case 1 does not materialize.

Case 2: Assume that $x^b \notin L$. Then:

$$b \in \mathbb{N} \wedge x^b \notin L \Rightarrow b \in \mathbb{N} \wedge x^b \notin \mathcal{L}(M_b) \quad [\text{via } L = \mathcal{L}(M_b)]$$
$$\Rightarrow x^b \in \{ x^a \mid a \in \mathbb{N} \wedge x^a \notin \mathcal{L}(M_a) \}$$
$$\rightarrow x^b \in L$$

which is a contradiction. We conclude that case 2 does not materialize.
From the above argument, it follows that L is NOT recursively enumerable □

EXERCISES

⑤ Let $\Sigma = \{a, b\}$ be an alphabet and consider the set

$A = \{ L \in \mathcal{P}(\Sigma^*) \mid L \text{ not recursively enumerable} \}$

Show that A is uncountable.

⑥ Let $\Sigma = \{a, b\}$ and consider two languages $L_1, L_2 \in \mathcal{P}(\Sigma^*)$.
Show that:

a) $\left\{ \begin{array}{l} L_1 \text{ recursively enumerable} \\ L_2 \text{ recursively enumerable} \end{array} \right. \Rightarrow L_1 \cap L_2$ recursively enumerable.

b) $\left\{ \begin{array}{l} L_1 \text{ recursive} \\ L_2 \text{ recursive} \end{array} \right. \Rightarrow L_1 \cap L_2$ recursive.