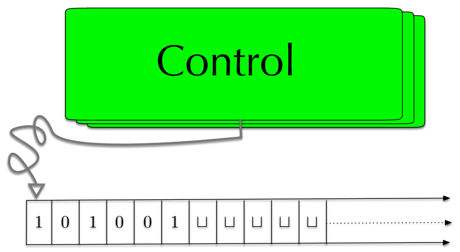
Chapter 2

Turing Machines

A Turing Machine (TM) is a general model of computation introduced in 1936 by Alan Turing [50]. It consist in an infinite tape and a tape head that can read, write and move around. It can both read the content of the tape and write on it. The read-write head can move both to the left and to the right. The tape is infinite. There are special states for rejecting and accepting which both take immediate effect.



With Turing machines as for pushdown automata and finite automata, one has the notion of deterministic machines and non-deterministic ones. We consider deterministic Turing machines first.

2.1 Deterministic Turing Machines

Definition 1.1

A (deterministic) TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc.}, q_{rej.})$ where Q, Σ, Γ are all finite sets and

(1) Q is the set of states,

(2) Σ is the alphabet *not* containing the *blank* symbol, $\sqcup$,

(3) Γ is the tape alphabet which satisfies $\sqcup \in \Gamma$ and $\Sigma \subsetneq \Gamma$

(4) $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function

(5) q_0 is the initial state

(6) $q_{acc.}$ is the accepting state

(7) $q_{rej.}$ is the rejecting state

Clearly $q_{acc.}$ and $q_{rej.}$ must be different states.

Notice that the head cannot move off the left hand end of the tape. If δ says so, it stays put. A *configuration* of a TM is a snapshot: it consists in the actual control state (q), the position of the head and what is written on the tape (w). To indicate the position of the head we consider the word w_0 which is located to the left of the head and slice the tape content w into the $w_0 w_1 = w$. This means that the head is actually positioned on the first letter of w_1. Strictly speaking the content of the tape is an infinite word:

$$w \sqcup \sqcup \sqcup \ldots\ldots \sqcup \sqcup \ldots$$

but we forget about the infinite suffix $\sqcup \sqcup \sqcup \ldots$. We then write $w_0 q w_1$ to say that

- the tape content is $w_0 w_1 \sqcup \sqcup \sqcup \ldots$
- the head is positioned on the first letter of $w_1 \sqcup \sqcup \sqcup \ldots$
- the actual control state is q.

The *initial configuration* on input $w \in \Sigma^{<\omega}$ is $q_0 w$.
An *halting configuration* is

- either an *accepting configuration* of the form $w_0 q_{acc.} w_1$,
- or a *rejecting configuration* of the form $w_0 q_{rej.} w_1$.

Given any two configurations C, C' we write $C \Rightarrow C'$ (for C yields C' in one step) if there exist $a, b, c \in \Gamma$, and $u, v \in \Gamma^*$ such that

- either $C = uaq_i bv$, $C' = uq_j acv$ and $\delta(q_i, b) = (q_j, c, L)$,
- or $C = q_i bv$, $C' = q_j cv$ and $\delta(q_i, b) = (q_j, c, L)$,
- or $C = uaq_i bv$, $C' = uacq_j v$ and $\delta(q_i, b) = (q_j, c, R)$.

Definition 1.2

A TM accepts input w if there is a sequence of configuration $C_0, \dots, C_k$ such that

(1) $C_0 = q_0 w$

(2) C_i yields C_{i+1} (for any $0 \leqslant i < k$)

(3) C_k is an accepting configuration.

Definition 1.3

The set of all words accepted by a TM $\mathcal{M}$ is the language it recognises:

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^{<\omega} \mid \mathcal{M} \textit{ accepts } w\}.$$

Example 1.1

A Turing machine that recognises

$$\{w\overline{w} \mid w \in \{0,1\}^*\}$$

where $\overline{w}$ is the mirror of w (for instance $\overline{001011} = 110100$).

$(Q, \Sigma, \Gamma, \delta, q_0, q_{acc.}, q_{rej.})$ where

(1) $Q = \{q_0, q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, q_{\texttt{write\_0}}, q_{\texttt{write\_1}}, q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, q_{\texttt{step\_right}}, q_{\texttt{acc}}, q_{\texttt{rej}}\}$

(2) $\Sigma = \{0, 1\}$

(3) $\Gamma = \{0, 1, \sqcup\}$

(4) $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is defined by

$(q_0, \sqcup)$	$\longrightarrow$	$q_{acc.}$
$(q_0, 0)$	$\longrightarrow$	$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, \sqcup, R)$
$(q_0, 1)$	$\longrightarrow$	$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, \sqcup, R)$
$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, \sqcup)$	$\longrightarrow$	$(q_{\texttt{write\_0}}, \sqcup, L)$
$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, 0)$	$\longrightarrow$	$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, 0, R)$
$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, 1)$	$\longrightarrow$	$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, 1, R)$
$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, \sqcup)$	$\longrightarrow$	$(q_{\texttt{write\_1}}, \sqcup, L)$
$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, 0)$	$\longrightarrow$	$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, 0, R)$
$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, 1)$	$\longrightarrow$	$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, 1, R)$
$(q_{\texttt{write\_0}}, \sqcup)$	$\longrightarrow$	$q_{rej.}$
$(q_{\texttt{write\_0}}, 0)$	$\longrightarrow$	$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, \sqcup, L)$
$(q_{\texttt{write\_0}}, 1)$	$\longrightarrow$	$q_{rej.}$
$(q_{\texttt{write\_1}}, \sqcup)$	$\longrightarrow$	$q_{rej.}$
$(q_{\texttt{write\_1}}, 0)$	$\longrightarrow$	$q_{rej.}$
$(q_{\texttt{write\_1}}, 1)$	$\longrightarrow$	$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, \sqcup, L)$
$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, \sqcup)$	$\longrightarrow$	$(q_{\texttt{step\_right}}, \sqcup, R)$
$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, 0)$	$\longrightarrow$	$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, 0, L)$
$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, 1)$	$\longrightarrow$	$(q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}, 1, L)$
$(q_{\texttt{step\_right}}, \sqcup)$	$\longrightarrow$	$q_{acc.}$
$(q_{\texttt{step\_right}}, 0)$	$\longrightarrow$	$(q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}, \sqcup, R)$
$(q_{\texttt{step\_right}}, 1)$	$\longrightarrow$	$(q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}, \sqcup, R)$

If we rename the states :

q_0	$\rightsquigarrow$	q_0
$q_{\texttt{remember\_0\_look\_for\_}\sqcup\texttt{\_go\_right}}$	$\rightsquigarrow$	q_1
$q_{\texttt{remember\_1\_look\_for\_}\sqcup\texttt{\_go\_right}}$	$\rightsquigarrow$	q_2
$q_{\texttt{write\_0}}$	$\rightsquigarrow$	q_3
$q_{\texttt{write\_1}}$	$\rightsquigarrow$	q_4
$q_{\texttt{look\_for\_}\sqcup\texttt{\_go\_left}}$	$\rightsquigarrow$	q_5
$q_{\texttt{step\_right}}$	$\rightsquigarrow$	q_6

the transition function becomes:

$(q_0, \sqcup)$	$\longrightarrow$	$q_{acc.}$
$(q_0, 0)$	$\longrightarrow$	$(q_1, \sqcup, R)$
$(q_0, 1)$	$\longrightarrow$	$(q_2, \sqcup, R)$
$(q_1, \sqcup)$	$\longrightarrow$	$(q_3, \sqcup, L)$
$(q_1, 0)$	$\longrightarrow$	$(q_1, 0, R)$
$(q_1, 1)$	$\longrightarrow$	$(q_1, 1, R)$
$(q_2, \sqcup)$	$\longrightarrow$	$(q_4, \sqcup, L)$
$(q_2, 0)$	$\longrightarrow$	$(q_2, 0, R)$
$(q_2, 1)$	$\longrightarrow$	$(q_2, 1, R)$
$(q_3, \sqcup)$	$\longrightarrow$	$q_{rej.}$
$(q_3, 0)$	$\longrightarrow$	$(q_5, \sqcup, L)$

$(q_3, 1)$	$\longrightarrow$	$q_{rej.}$
$(q_4, \sqcup)$	$\longrightarrow$	$q_{rej.}$
$(q_4, 0)$	$\longrightarrow$	$q_{rej.}$
$(q_4, 1)$	$\longrightarrow$	$(q_5, \sqcup, L)$
$(q_5, \sqcup)$	$\longrightarrow$	$(q_6, \sqcup, R)$
$(q_5, 0)$	$\longrightarrow$	$(q_5, 0, L)$
$(q_5, 1)$	$\longrightarrow$	$(q_5, 1, L)$
$(q_6, \sqcup)$	$\longrightarrow$	$q_{acc.}$
$(q_6, 0)$	$\longrightarrow$	$(q_1, \sqcup, R)$
$(q_6, 1)$	$\longrightarrow$	$(q_2, \sqcup, R)$

Definition 1.4

A language L is *Turing recognizable* if there exists a TM $\mathcal{M}$ such that

$$L = \mathcal{L}(\mathcal{M}).$$

Proposition 1.1

Turing Machines with bi-infinite tapes are equivalent to Turing machines.

Proof of Proposition 1.1:

Left as an exercise. □

Proposition 1.2

Pushdown automata with 2 stacks are equivalent to Turing machines.

Proof of Proposition 1.2:

Left as an exercise. □

Definition 1.5

A *Decider* is a TM that halts on all inputs.

Definition 1.6

A language is *Turing decidable* iff there exists a *Decider* that recognises it.

(We will see later that *Turing recognizable* is also called *recursively enumerable* (*r.e.* for short) and *decidable* is also called *recursive*.)

Example 1.2

A *Decider* for $\{a^n b^n c^n \mid n \in \mathbb{N}\}$:

- Scan the input from left to right to be sure that it is a member of $a^*b^*c^*$ and reject if it isn't.
- Return the head to the left and change one c into an x, then one b into x, then one a into x. Go back to the first blank $\sqcup$.

  Repeat again until the tape is only composed of x, in which case accept. Otherwise reject.

Definition 1.7

A k tape TM is the same as a TM except that is composed of k tapes: ①,…,ⓚ, with k independent heads so that the transition function becomes

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

Notice that a configuration of a k-tape Turing machine is of the form

$$\big(\underset{①}{u_1 q v_1} \quad , \quad \underset{②}{u_2 q v_2} \quad , \quad \ldots\ldots \quad , \quad \underset{ⓚ}{u_k q v_k}\big) \; .$$

Proposition 1.3

Given any TM there exist

(1) an equivalent TM with a bi-infinite tape,

(2) a multi-tape TM,

(3) a multi-tape with bi-infinite tapes TM.

Proof of Proposition 1.3:

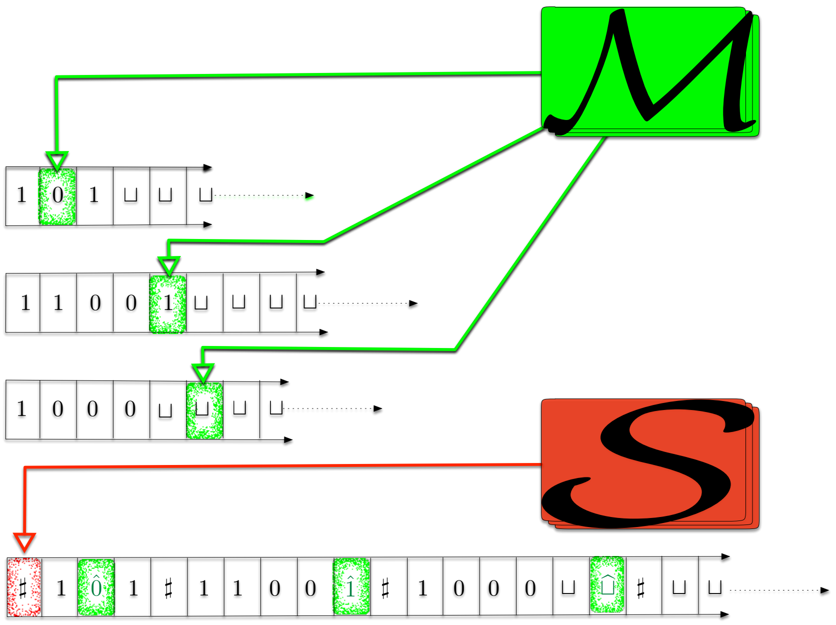
Left as an exercise. □

Theorem 1.1

Every multi-tape TM has an equivalent single tape TM.

Proof of Theorem 1.1:

Let $\mathcal{M}$ be a multi-tape TM. We will describe a TM $\mathcal{S}$ that recognises the same language. Let $(w_1, w_2, \ldots, w_k)$ be the input of $\mathcal{M}$ on its k tapes. The corresponding input of $\mathcal{S}$ will be $\sharp w_1 \sharp w_2 \sharp \ldots \sharp w_k \sharp$, where $\sharp$ does not belong to the alphabet of $\mathcal{M}$. To simulate a single move of $\mathcal{M}$, $\mathcal{S}$ scans its tape from the first $\sharp$ which marks the left-hand end, to the $k+1^{th}$ $\sharp$ (which marks the right-hand end) replacing each letter a right after the $\sharp$ symbol (except for the $k+1^{th}$ one) by $\hat{a}$ to indicate the position of the heads. Then $\mathcal{S}$ makes a second pass to update the tapes according to $\mathcal{M}$'s transition functions. If at any point $\mathcal{S}$ moves one of the virtual heads to the right onto a $\sharp$, this action signifies that $\mathcal{M}$ has moved the corresponding head onto the previously unread blank portion of that tape. So $\mathcal{S}$ writes a blank symbol on this tape cell and shifts the tape contents from this cell until the rightmost $\sharp$, one unit to the right. Then it continues the simulation as before.



$\square$

2.2 Non-Deterministic Turing Machines

Definition 2.1

A non-deterministic TM (NTM) is the same as a deterministic TM except for the transition function which is of the form:

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

The computation of a (deterministic) TM is a sequence of configurations

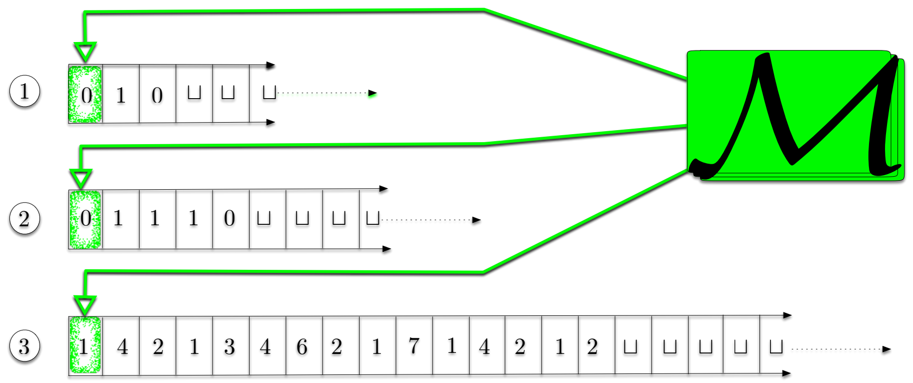
$$C_0 \Longrightarrow C_1 \Longrightarrow \ldots \Longrightarrow C_k \Longrightarrow \ldots$$

that may be finite or infinite.
It accepts the input if this sequence is finite and the last configuration is an accepting one.
The computation of a non-deterministic TM is no more a sequence of configurations but a tree whose nodes are configurations. This tree may have both infinite and finite branches. The machine accepts the input if and only if there exists some branch that is finite and whose leaf is an accepting configuration.

Theorem 2.1

For every NTM there exists a deterministic TM that recognises the same language.

Proof of Theorem 2.1:



We consider a 3-tape (①,② and ③) deterministic TM $\mathcal{M}$ to simulate a NTM $\mathcal{N}$:

- (1) ① is the *input tape*,
  (2) ② is the *simulation tape*, and
  (3) ③ is the *address tape*.
- Initially, ① contains the input w and ② and ③ are empty.
- ① always keeps the input w. So the content of ① is never modified.
- ② simulates $\mathcal{N}$ on one – initial segment of a – branch of its non-deterministic computation tree.
- ③ contains a finite word which corresponds to a succession of non deterministic choices. For instance the word 132 stands for: among the non-deterministic options choose the first one for the first transition, the third one for the second and the second one for the third. This means that we consider $k \in \mathbb{N}$ to be

  $$\max\{Card(\delta(q,\gamma)) \mid q \in Q,\ \gamma \in \Gamma\}$$

  and for each $\mid q \in Q,\ \gamma \in \Gamma$ we fix a total ordering of $\delta(q,\gamma)$.

  Words on ③ all belong to $\{1,2,\ldots,k\}^*$. Moreover, during the running time, the content of ③ changes over and over again until the machine accepts. This series gives rise to an enumeration of the infinite k-ary tree in a breadth-first search. This means it enumerates all words in $\{1,2,\ldots,k\}^*$ along the following well-ordering:

$$u \prec v \iff \begin{cases} |u| < |v| \\ or \\ |u| = |v| \text{ and } u <_{lexic.} v \end{cases}$$

which gives:

$$\varepsilon, 1, 2, \ldots, k, 11, 12, \ldots, 1k, 21, 22, \ldots, 2k, \ldots\ldots, k1, k2, \ldots, kk, 111, 112, \ldots, 11k, \ldots\ldots$$

- At first, $\mathcal{M}$ Copies the content of ① (= the input w) to ②.
- It then uses ② to simulate $\mathcal{N}$ with input w on the branch b of its non-deterministic computation which is lodged on ③. In case the word b does not correspond to a real computation[a] or if the simulation of $\mathcal{N}$ on ② either reaches the rejecting state or does not reach any halting state at all, then $\mathcal{M}$ erases completely ②, replaces b on ③ with its the immediate $\prec$-successor, and starts all over again – by copying ① on ② and simulating $\mathcal{N}$ on ② in accordance with the series of choices recorded on ③.

□

[a]this is the case for instance if from the initial configuration q_0w there are only two control states non-deterministically available, whereas the word on ③ reads $3\ldots$.

Proposition 2.1

- Decidable languages are closed under union, intersection and complementation.
- Turing recognizable languages are closed under union and intersection.

Proof of Proposition 2.1:

Left as an exercise.

□

Definition 2.2

Tape p of a k-tape TM works as a *printer* if its head on tape p only goes right.
We say that on tape p, the infinite word $a_0a_1a_2a_3\ldots\ldots \in \Gamma^\omega$ is printed out by some computation of the TM if

- $\circ$ either the TM halts and $a_0a_1a_2a_3\ldots\ldots$ is what shows on tape p, or
- $\circ$ the TM never halts but for each cell n of tape p, a_n is the letter printed out[a].

[a]This is printed out precisely at step $n+1$ since the head only goes right.

Definition 2.3

An *enumerator* is a 2-tape TM whose second tape works as a *printer*.
Assuming that on the empty input it prints out the infinite word $a_0a_1a_2a_3\ldots\ldots\ldots$ we say that it enumerates the following language

$$\mathcal{L} = \left\{ a_k\ldots a_{k+i} \in \Sigma^* \mid 0 \leqslant i \text{ and } \left[\begin{array}{c} k=0 \text{ and } a_{k+i+1} = \sqcup \\ \text{or} \\ a_{k-1} = a_{k+i+1} = \sqcup \end{array}\right]\right\}$$

(Notice that the alphabet of the printer tape must satisfy $\Sigma \cup \{\varepsilon, \sqcup\} \subseteq \Gamma$ so that it can eventually print out the empty word).

This means that an enumerator prints out words separated by $\sqcup$. When it prints out an ever ending word that contains no $\sqcup$, the result is the same as if it were printing an ever ending sequence of $\sqcup$: the same language would be enumerated. So, for every enumerator $\mathcal{E}$ there exists an equivalent enumerator $\mathcal{E}'$ that enumerate the same language but will always, whenever it writes a letter different from $\sqcup$, write a $\sqcup$ symbol further away.
Such an enumerator would print out something like

$$\sqcup^* w_0 \sqcup \sqcup^* w_1 \sqcup \sqcup^* w_2 \sqcup \sqcup^* \ldots\ldots \sqcup \sqcup^* w_n \sqcup \sqcup^* w_{n+1} \ldots\ldots\ldots$$

when $\{w_i \mid i \in \mathbb{N}\} = \mathcal{L}$ whenever infinitely many words are printed. Or

$$\sqcup^* w_0 \sqcup \sqcup^* w_1 \sqcup \sqcup^* w_2 \sqcup \sqcup^* \ldots\ldots \sqcup \sqcup^* w_n \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \ldots\ldots\ldots$$

when $\{w_i \mid i \leqslant n\} = \mathcal{L}$ is finite. In particular it would print out

$$\sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \ldots\ldots\ldots\ldots\ldots\ldots$$

for the empty language.
Notice that the words that compose $\mathcal{L}$ may come in any order, and they also may be printed out many times or even infinitely often.

Definition 2.4

A language $\mathcal{L}$ is *recursively enumerable* if there is an enumerator that enumerates $\mathcal{L}$.

Theorem 2.2

A language is *Turing Recognizable* if and only if it is *recursively enumerable.*

Proof of Theorem 2.2:

($\Rightarrow$) from $\mathcal{M}$ we build $\mathcal{E}$ that enumerates $\mathcal{L}(\mathcal{M})$.

(1) Repeat the following for $i = 1, 2, 3, \ldots$

(2) Successively for each word $w \in \Sigma^{\leqslant i}$, run $\mathcal{M}$ for i-many steps on w

(3) If any computation accepts, print out the corresponding w.

(This way every word $w \in \mathcal{L}$ will be printed out – even infinitely often – and none others.)

($\Leftarrow$) From $\mathcal{E}$ we build a k-tape TM $\mathcal{M}$. On input w: it runs $\mathcal{E}$ on two of its tape, and some other one it checks every time $\mathcal{E}$ outputs some word v, whether $v = w$ or not and accepts if eventually they are the same.

$\square$

Proposition 2.2

For any <u>infinite</u> $L \subseteq \Sigma^*$,

$$L \textit{ is Turing decidable} \iff \begin{cases} \textit{there exits an enumerator } \mathcal{E} \textit{ which prints out} \\ u_0 \sqcup u_1 \sqcup u_2 \sqcup \ldots\ldots \sqcup u_i \sqcup u_{i+1} \sqcup \ldots\ldots\ldots \\ \textit{such that} \begin{cases} L = \{u_i \mid i \in \mathbb{N}\} \\ \textit{and} \\ i < j \Longrightarrow \begin{cases} |u| < |v| \\ \textit{or} \\ |u| = |v| \textit{ and } u <_{lexic.} v. \end{cases} \end{cases} \end{cases}$$

Notice that this enumerator is required to leave exactly one $\sqcup$ between the successive words that it prints out, so that whenever it prints out two consecutive $\sqcup$, it will forever on only print $\sqcup$ symbols (which means go right indefinitely without modifying the content of the tape).

Proof of Proposition 2.2:

Left as an exercise. $\square$

2.3 The Concept of Algorithm

In 1900, Hilbert gave a list of the main mathematical problems of the time [26, 27]. The 10th one was the following: *given a Diophantine equation with any number of unknown quantities, and with rational integral numerical coefficients, can we derive a process according to which it can be determined in a finite number of operations whether the equation admits a rational integer solution?* This corresponds to the intuitive notion of an algorithm. Proving that such an algorithm does not exist requires a formal definition of the notion of *"algorithm"*. The *"Church-Turing thesis"* states that the informal notion of an algorithm corresponds exactly to the notion of a λ-calculus formula or equivalently to a Turing machine.
In 1970, Yuri Matijasevic proved[1] that the 10th problem of Hilbert is undecidable [33]: assuming that the notation $P(x_1, \ldots, x_n)$ stands for a polynomial with integer coefficients, then there is no decider for

$$\{P(x_1, \ldots, x_n) \mid \exists (a_1, \ldots a_n) \in \mathbb{N}^n \quad P(a_1, \ldots, a_n) = 0\}.$$

Definition 3.1

A *"coding"* is a rule for converting a piece of information into another object. Given any non empty sets E, F, a coding is a *one-to-one* (total) function

$$c : E \xrightarrow{1-1} F.$$

Example 3.1

$E = \{0,1\}^*$, $F = \mathbb{N}$ and $c : E \xrightarrow{1-1} F$ is a coding defined by:

$$c(w) = \overline{1w}^2 \quad (= \textit{the word "}1w\textit{" read in base } 2).$$

[1]this is combined work of Martin Davis, Yuri Matiyasevich, Hilary Putnam and Julia Robinson

Notation 3.1

Given any Turing machine $\mathcal{M}$, we write

- $\mathcal{M}(w)\downarrow$ to say that the machine $\mathcal{M}$ stops on input w
  - $\mathcal{M}(w)\downarrow^{\text{acc.}}$ means that $\mathcal{M}$ stops in an accepting configuration, and
  - $\mathcal{M}(w)\downarrow^{\text{rej.}}$ means that $\mathcal{M}$ stops in a rejecting configuration.
- $\mathcal{M}(w)\uparrow$ to say that the machine $\mathcal{M}$ never stops on input w.

Definition 3.2

Given any two non-empty finite sets A, B, a *partial* function $f : A^* \longrightarrow B^*$ is *"Turing computable"* if and only if there exists a Turing machine $\mathcal{M}_f$ such that

- on input $w \notin dom(f)$: $\mathcal{M}_f(w)\uparrow$, and
- on input $w \in dom(f)$: $\mathcal{M}_f(w)\downarrow^{\text{acc.}}$ with the word "$f(w)$" on its tape.

We notice the following:

(1) Given any finite alphabet Σ, and any TM $\mathcal{M}$ whose alphabet is Σ, there exists a *Turing computable* coding: $c : \Sigma^{<\omega} \longrightarrow \{0,1,\sqcup\}^{<\omega}$ and a TM $\mathcal{M}_c$ with tape alphabet $\{0,1,\sqcup\}$ such that $\mathcal{M}$ accepts w if and only if $\mathcal{M}_c$ accepts $c(w)$.

(2) Every regular language is decidable because a DFA is nothing but a deterministic TM that always goes right.

(3) Every Context-free language is decidable, because any PDA can be easily simulated by some equivalent non-deterministic TM.

(4) We have the following strict inclusions of languages.

$$\textit{Regular} \subsetneq \textit{Context-Free} \subsetneq \underset{\substack{\| \\ \textit{Recursive}}}{\textit{Decidable}} \subsetneq \underset{\substack{\| \\ \textit{Recursively Enumerable}}}{\textit{Turing Recognizable}}.$$

In computer science, a programming language is said be *"Turing complete"* or *"universal"* if it can be used to simulate any single-tape Turing machine. Examples of Turing-complete programming languages include:

∘ Ada	∘ C++	∘ Java	∘ Pascal
∘ C	∘ Common Lisp	∘ Lisp	∘ Prolog, etc.

2.4 Universal Turing Machine

If we compare a Turing Machine with a computer, on one hand the TM seems much better because it can compute for ever without any chance to breakdown and it has an infinitely large storage facility. But on the other hand, a TM seems to be more of a computer with a single software program, whereas a computer can run different programs. A computer resembles more of a Turing machine with finite capacity but, a Turing machine that we can modify by changing its transition function – every program is like a new transition function for the machine.
How are we going to address this issue, since we claimed that a Turing machine is an abstract model of computation ? This answer to this is the *Universal Turing Machine*. It is a machine that can work just like any other machine provided that we feed it with the right *code* of the machine.
We will employ Turing machines to obtain:

(1) a languages that is *Turing recognizable but not decidable*[2],

(2) a language that is not *Turing recognizable.*

From now on, we only consider Turing Machines with fixed alphabets $\Sigma = \{0,1\}, \Gamma = \{0,1,\sqcup\}$. Any such TM is of the form:

$$\mathcal{M} = \langle \{q_0, q_1, \ldots, q_k\}, \{0,1\}, \{0,1,\sqcup\}, \underline{\delta}, q_0, q_{acc.}, q_{rej.} \rangle$$

Where $\underline{\delta}$ is the description of the transition function of $\mathcal{M}$:

$$\underline{\delta} = \{(q_3, 0, q_1, 1, R),\ (q_8, 1, q_4, 0, L),\ (q_3, 0, q_3, 0, L), \ldots\ldots\}$$

The description of such a machine is a finite sequence M over some finite alphabet A:

$$\Big\{ \langle,\ \rangle,\ q,\ {\scriptstyle 0},\ {\scriptstyle 1},\ {\scriptstyle 2},\ {\scriptstyle 3},\ {\scriptstyle 4},\ {\scriptstyle 5},\ {\scriptstyle 6},\ {\scriptstyle 7},\ {\scriptstyle 8},\ {\scriptstyle 9},\ 0,\ 1,\ \sqcup,\ L,\ R,\ \{,\ \},\ (,\ ),\ , \Big\} = A.$$

Since $Card(A) < 2^8$ we can code any letter $l \in A$ by a sequence of *eight* 0's and 1's, i.e we take any 1-1 mapping

$$C : A \longrightarrow \{0,1\}^{[8]}$$

and we define a Turing computable coding

$$c : A^{<\omega} \longrightarrow \{0,1\}^{<\omega}$$

by

$$c(a_0 \ldots a_p) = C(a_0)\hat{}C(a_1)\hat{}C(a_2)\hat{}\ldots\hat{}C(a_p).$$

[2]in other words: a *non-recursive recursively enumerable language.*

We denote by $\ulcorner\mathcal{M}\urcorner$ the code of $\mathcal{M}$, i.e.,

$$\ulcorner\mathcal{M}\urcorner = c(\mathcal{M}).$$

Clearly, the following language is decidable:

$$\{\ulcorner\mathcal{M}\urcorner : \mathcal{M} \textit{ is a TM}\}.$$

Proposition 4.1: Universal Turing Machine

There exists a Turing machine[a] $\mathcal{U}$ such that on each input of the form $vw \in \{0,1\}^*$,

if $v = \ulcorner\mathcal{M}\urcorner$ for some Turing machine[b] $\mathcal{M}$, then $\mathcal{U}$ works as $\mathcal{M}$ on input w.

[a] working on alphabets $\Sigma_{\mathcal{U}} = \{0,1\}$ and $\Gamma_{\mathcal{U}} = \{0,1,\sqcup\}$

[b] also working on alphabets $\Sigma_{\mathcal{U}} = \{0,1\}$ and $\Gamma_{\mathcal{U}} = \{0,1,\sqcup\}$

Notice that for any word $u \in \{0,1\}^*$, if there is a prefix of u which is the code of a Turing Machine, then this prefix is unique [3]. Therefore, in case a word $u \in \{0,1\}^*$ can be decomposed into $u = \ulcorner\mathcal{M}\urcorner w$ for some Turing machine $\mathcal{M}$, this decomposition is then unique.

This means for instance that on any input w:

- $\mathcal{U}(w) \uparrow$ if and only if $\mathcal{M}(w) \uparrow$;
- $\mathcal{U}(w) \downarrow^{\text{rej.}}$ with the word w' on its tape if and only if $\mathcal{M}(w) \downarrow^{\text{rej.}}$ with the word w' on its tape;
- $\mathcal{U}(w') \downarrow^{\text{acc.}}$ with w' on its tape if and only if $\mathcal{M}(w) \downarrow^{\text{acc.}}$ with w' on its tape.

Proof of Proposition 4.1:

We build a 6-tape deterministic Universal Turing machine $\mathcal{U}$.

(1) on $\textcircled{1}$ the input $\ulcorner\mathcal{M}\urcorner w$ is inserted. It will never be modified during the rest of the computation. Then $\mathcal{U}$ copies the code of

  (a) the transition function of $\mathcal{M}$ – $\ulcorner\underline{\delta}\urcorner$ – on $\textcircled{2}$;

  (b) the initial state of $\mathcal{M}$ – $\ulcorner q_0\urcorner$ – on $\textcircled{3}$ [a];

  (c) the accepting state of $\mathcal{M}$ – $\ulcorner q_{acc.}\urcorner$ – on $\textcircled{4}$ [b];

  (d) the rejecting state of $\mathcal{M}$ – $\ulcorner q_{rej.}\urcorner$ – on $\textcircled{5}$ [c].

(2) It then uses $\textcircled{6}$ to simulate $\mathcal{M}$ on input w: for each step of $\mathcal{M}$

  (a) $\mathcal{U}$ reads a letter – say 0 – on $\textcircled{6}$, and

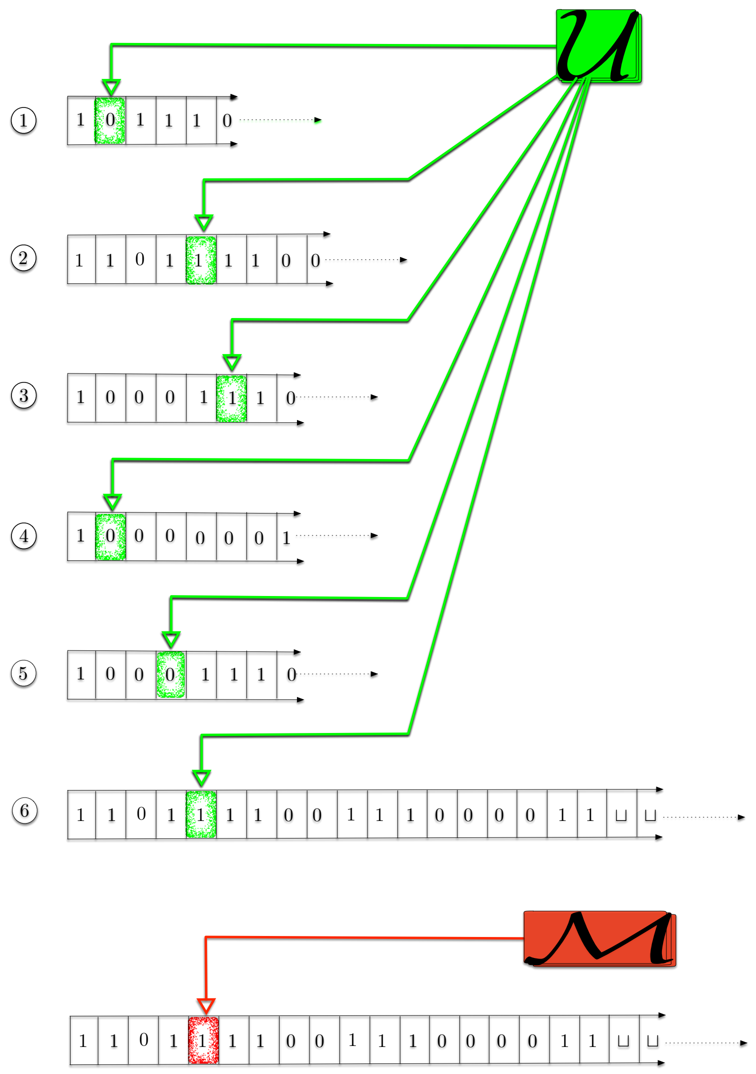
[3] this comes from the fact the last letter of a word that defines a TM is $\rangle$. Therefore, reading u from left to right by blocks of eight 0's or 1's, the first block that corresponds to $\ulcorner\rangle\urcorner$ marks the end of the wanted prefix.

(b) using the code of the actual state – say $\ulcorner q_3 \urcorner$ – on ③, $\mathcal{U}$ looks in ② for the code of the corresponding transition – say $\ulcorner (q_3, 0, q_1, 1, R) \urcorner$ – and then

(c) $\mathcal{U}$ verifies that the code of the new state – here $\ulcorner q_1 \urcorner$ – is different from the content of ④ and ⑤ (otherwise if it corresponds to the content of ④ it means that it is $\ulcorner q_{acc.} \urcorner$, and $\mathcal{U}$ accepts right away, and if it corresponds to the content of ⑤ it means it is $\ulcorner q_{rej.} \urcorner$, in which case $\mathcal{U}$ rejects).

(d) If the new state is different from both $q_{acc.}$ and $q_{rej.}$ – in our example q_1 is different from both $q_{acc.}$ and $q_{rej.}$ – $\mathcal{U}$ replaces on ⑥ the letter it just read with the new one – here it replaces 0 by 1 – and still on tape ⑥ it makes the move indicated – here it goes right – and finally,

(e) $\mathcal{U}$ replaces on ③ the code of the old state by the new one – here it replaces $\ulcorner q_3 \urcorner$ by $\ulcorner q_1 \urcorner$.



$\square$

[a]later on this tape will store the code of the actual state that $\mathcal{M}$ is on.
[b]the content of ④ will never be modified in the future.
[c]the content of ⑤ will never be modified in the future.

2.5 The Halting Problem

Proposition 5.1

The following language is Turing recognizable but not decidable:

$$\{\ulcorner\mathcal{M}\urcorner w \in \{0,1\}^* \mid \mathcal{M} \textit{ is a TM that accepts } w\}.$$

Proof of Proposition 5.1:

Towards a contradiction we assume there exists a *Decider* $\mathcal{D}$ that decides this language. We build a Turing machine $\mathcal{H}$ which works the following way:
on input w

- if $\mathcal{D}$ accepts ww, then $\mathcal{H}$ does not halt.
- if $\mathcal{D}$ rejects ww, then $\mathcal{H}$ accepts.

Notice that

$$\mathcal{H} \textit{ accepts } \ulcorner\mathcal{H}\urcorner \iff \mathcal{D} \textit{ rejects } \ulcorner\mathcal{H}\urcorner\ulcorner\mathcal{H}\urcorner \iff \mathcal{H} \textit{ does not accept } \ulcorner\mathcal{H}\urcorner.$$

Or to say it differently

$$\mathcal{H}(\ulcorner\mathcal{H}\urcorner)\downarrow^{\text{acc.}} \iff \mathcal{D}(\ulcorner\mathcal{H}\urcorner\ulcorner\mathcal{H}\urcorner)\downarrow^{\text{rej.}} \iff \mathcal{H}(\ulcorner\mathcal{H}\urcorner)\uparrow .$$

To see things slightly differently, since the machine $\mathcal{H}$ only stops when it accepts we can reformulate the contradiction in

$$\mathcal{H}(\ulcorner\mathcal{H}\urcorner)\downarrow \iff \mathcal{H}(\ulcorner\mathcal{H}\urcorner)\uparrow .$$

□

2.6 Some Other Undecidable Problems

Proposition 6.1

The following language is Turing recognizable but not decidable:

$$\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{M}(\varepsilon)\downarrow\}.$$

Proof of Proposition 6.1:

The fact this language is Turing recognizable is immediate. It is not recursive because otherwise,

$$\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{M}(\varepsilon)\downarrow\}$$

would also be recursive. Indeed, one can design a program which, given any code $\ulcorner\mathcal{M}\urcorner$ of a TM $\mathcal{M}$, computes the code $\ulcorner\mathcal{M}'\urcorner$ of a TM $\mathcal{M}'$ which on every input, first starts by erasing the entire input, then works exactly as $\mathcal{M}$ on the empty string. □

Corollary 6.1

The following languages are not recursively enumerable:

(1) $\{0,1\}^*\backslash\{\ulcorner\mathcal{M}\urcorner w \in \{0,1\}^* \mid \mathcal{M}(w)\downarrow^{\text{acc.}}\}$

(2) $\{0,1\}^*\backslash\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{M}(\varepsilon)\downarrow\}$

(3) $\{\ulcorner\mathcal{M}\urcorner w \in \{0,1\}^* \mid \mathcal{M}(w)\downarrow^{\text{rej.}} \text{ or } \mathcal{M}(w)\uparrow\}$

(4) $\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{M}(\varepsilon)\uparrow\}$.

Proof of Corollary 6.1:

Left as an exercise. □

Proposition 6.2

The following problem is not decidable:

$$\{\ulcorner\mathcal{M}\urcorner w \in \{0,1\}^* \mid \text{ the computation of } \mathcal{M} \text{ on } w \text{ uses all non-halting states}\}.$$

Proof of Proposition 6.2:

Left as an exercise. □

Rice's Theorem. *If $\mathcal{C}$ is any class of Turing-recognizable languages that is neither the whole class of Turing-recognizable languages nor the empty set, then*

$$\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{L}(\mathcal{M}) \in \mathcal{C}\}.$$

is not decidable:

In other words, if $\mathcal{C}$ is a non-empty proper class of Turing-recognizable languages, then the problem of determining whether the language of a Turing machine belongs to the class is undecidable.

Notice that when $\mathcal{C}$ is the empty set, then this problem is obviously decidable since

$$\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{L}(\mathcal{M}) \in \varnothing\} = \varnothing.$$

Same holds when $\mathcal{C}$ is the whole class of Turing-recognizable languages, checking whether the language recognized by a TM belongs to $\mathcal{C}$ is trivial.

Proof of Rice's Theorem:

We first assume that $\varnothing \notin \mathcal{C}$. Consider any TM $\mathcal{M}_{in}$ such that $\mathcal{L}(\mathcal{M}_{in}) \in \mathcal{C}$. Now, design the following program: given any TM $\mathcal{M}$ build the TM $\mathcal{M}'$ such that on any input w, the machine $\mathcal{M}'$ on w first stores the input w and works exactly as $\mathcal{M}$ on the empty input (i.e., the empty string) until, if ever, it reaches an accepting configuration, in which case, right before this happens, $\mathcal{M}'$ erases the whole working tape and starts now working exactly as $\mathcal{M}_{in}$ on w. One sees that

- if $\mathcal{M}$ accepts the empty string, then $\mathcal{L}(\mathcal{M}') = \mathcal{L}(\mathcal{M}_{in})$,
- if $\mathcal{M}$ does not accept the empty string, then $\mathcal{L}(\mathcal{M}') = \varnothing$.

Therefore, $\mathcal{M}$ accepts the empty string if and only if $\mathcal{L}(\mathcal{M}') \in \mathcal{C}$.

We now assume that $\varnothing \in \mathcal{C}$ and consider any TM $\mathcal{M}_{out} \notin \mathcal{C}$. Given any TM $\mathcal{M}$, we develop a TM $\mathcal{M}'$ as in the previous case just replacing $\mathcal{M}_{in}$ by $\mathcal{M}_{out}$. So, we obtain $\mathcal{M}$ accepts the empty string if and only if $\mathcal{L}(\mathcal{M}') \notin \mathcal{C}$.

Therefore, in any case, the problem of a TM halting or not on the empty string becomes decidable. A contradiction.

$\square$

Corollary 6.2

One cannot decide whether

(1) the language recognized by a TM is also recognized by an automaton;

(2) the language recognized by a TM is not recognized by any automaton;

(3) the language recognized by a TM contains at least one word;

(4) the language recognized by a TM contains at least three words;

(5) the language recognized by a TM contains all finite words.

(6) the language recognized by a TM contains exactly all finite words of length $\leqslant 7$.

(7) the language recognized by a TM contains exactly all finite words of length $\geqslant 7$.

(8) the language recognized by a TM contains infinitely many words.

Proof of Corollary 6.2:

Left as an exercise. □

2.6.1 The Post Correspondence Problem

Imagine you are given a finite set of dominos of the form $P = \left\{ \left[\frac{u_i}{v_i}\right] \mid i \in I \right\}$ where $u_i, v_i \in \Sigma^*$.
For instance:

$$P = \left\{ \left[\frac{aa}{ba}\right], \left[\frac{baa}{ca}\right], \left[\frac{acca}{a}\right], \left[\frac{aaaa}{accc}\right], \left[\frac{a}{ac}\right], \left[\frac{b}{ab}\right], \left[\frac{c}{cc}\right], \left[\frac{bb}{bbbb}\right] \right\}.$$

The question is then whether there exists a non-empty sequence (repetitions of dominos are accepted!)

$$\left[\frac{u_{i_0}}{v_{i_0}}\right] \left[\frac{u_{i_1}}{v_{i_1}}\right] \left[\frac{u_{i_2}}{v_{i_2}}\right] \dots \left[\frac{u_{i_{k-1}}}{v_{i_{k-1}}}\right] \left[\frac{u_{i_k}}{v_{i_k}}\right]$$

such that

$$u_{i_0} \cdot u_{i_0} \cdot \dots \cdot u_{i_{k-1}} \cdot u_{i_k} = v_{i_0} \cdot v_{i_0} \cdot \dots \cdot v_{i_{k-1}} \cdot v_{i_k}.$$

Such a sequence is called a **match**.
For instance:

$$\left[\frac{acca}{a}\right] \left[\frac{c}{cc}\right] \left[\frac{a}{ac}\right] \left[\frac{b}{ab}\right] \text{ is a match since we get } \frac{accacab}{accacab}$$

Post Correspondence Problem.
It is undecidable, given any instance $P = \left\{ \left[\frac{u_i}{v_i}\right] \mid i \in I \right\}$ *of the Post Correspondence Problem, to determine whether there exists a match or not.*

Proof of Post Correspondence Problem:

See exercises sheet or Sipser's "*Introduction to the Theory of Computation*" [43, pages 183–189]. □

The whole idea of the proof consists in reducing the Halting Problem to this one. So that if we were able to decide the Post Correspondence Problem, then we could as well decide the Halting Problem. Since we know that the halting problem is undecidable, this implies that the Post Correspondence Problem is also undecidable.

2.7 Turing Machine with Oracle

A Turing machine with an oracle is one finite object (a Turing machine suitable for any oracle: an almost regular 2-tape Turing Machine) plus one infinite object so that this TM can have access to an infinite amount of information – something a regular Turing machine never does.

Definition 7.1

(1) An oracle is any subset $\mathbb{O} \subseteq \mathbb{N}$.

(2) An *oracle-compatible-Turing machine* (o-c-TM) is a 2-tape Turing machine similar to any 2-tape Turing machine except that it only reads but never writes on tape ②:

$$\mathcal{O} = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc.}, q_{rej.})$$

(3) An oracle-compatible-Turing machine $\mathcal{O}$ *equipped with the oracle* $\mathbb{O}$, on input word $w \in \Sigma^*$ (in short an oracle TM $\mathcal{O}^{\mathbb{O}}$ on word $w \in \Sigma^*$) is nothing but the TM $\mathcal{O}$ whose initial configuration is

$$\Big(\underset{①}{q_0 w} \quad , \quad \underset{②}{q_0 \chi_{\mathbb{O}}}\Big)$$

where $\chi_{\mathbb{O}} \in \{0,1\}^\omega$ is the *infinite word*

$$\chi_{\mathbb{O}}(0)\chi_{\mathbb{O}}(1)\chi_{\mathbb{O}}(2)\ldots\ldots\chi_{\mathbb{O}}(n)\chi_{\mathbb{O}}(n+1)\ldots\ldots\ldots\ldots$$

defined by

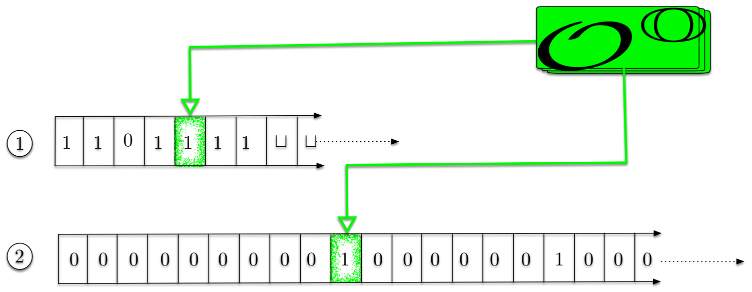
$$\chi_{\mathbb{O}}(n) = \begin{cases} 1 & \textit{if} \quad n \in \mathbb{O} \\ & \textit{and} \\ 0 & \textit{if} \quad n \notin \mathbb{O}. \end{cases}$$

This means that on tape ② the whole characteristic function of the oracle is already available once the machine starts. So that the machine is granted access to all of this ”external” information: it knows which integers belong to $\mathbb{O}$ and which do not. For instance, in case $\mathbb{O}$ is the set of all integers n such that:

(1) n reads " $1\ulcorner\mathcal{M}\urcorner w$ " in the decimal numeral system,

(2) $\mathcal{M}(w)\downarrow$;

then $\mathcal{O}^{\mathbb{O}}$ may be able to decide the Halting Problem. Of course this does not lead to a contradiction since there is no chance that such a Turing machine [4] ever sees its own code onto tape ② (although the code of $\mathcal{O}$ – or the code of an equivalent TM – does show on ②).



Example 7.1

Let $\mathbb{O} \subseteq \mathbb{N}$ be the set of all the codes of Turing machines that halt on the empty input:

$$\mathbb{O} = \{\overline{1\ulcorner\mathcal{M}\urcorner}^2 \in \mathbb{N} \mid \mathcal{M}(\varepsilon)\downarrow\}[5].$$

We describe an oracle-compatible-TM $\mathcal{O}$ that, once equipped with the oracle $\mathbb{O}$, decides the language

$$\{\ulcorner\mathcal{M}\urcorner \in \{0,1\}^* \mid \mathcal{M}(\varepsilon)\downarrow\}.$$

The machine $\mathcal{O}^{\mathbb{O}}$ works this way:

(1) on input $w \in \{0,1\}^*$, the TM $\mathcal{O}^{\mathbb{O}}$ checks whether w is the code of a TM

if it is not the case it rejects right away. Otherwise,

(2) it computes $n = \overline{1\ulcorner\mathcal{M}\urcorner}^2$, then checks on tape ② whether $\chi_{\mathbb{O}}(n) = 1$ – in which case it accepts – or $\chi_{\mathbb{O}}(n) = 0$ – in which case it rejects.

Notation 7.1

- Notice that the mapping $\begin{array}{llll} f: & \{0,1\}^* & \longleftrightarrow & \mathbb{N} \\ & w & \longmapsto & \overline{1w}^2 - 1 \end{array}$ is a bijection.

  For any word w we write $\ulcorner w\urcorner$ for $f(w)$, and for any integer k we write $\ulcorner k\urcorner$ for $f^{-1}(k)$.

[4] we are talking about $\mathcal{O}^{\mathbb{O}}$ and not just $\mathcal{O}$!
[5] $\overline{w}^2$ stands for the integer n that, once written in base 2, yields the word w

For instance $\ulcorner 0010 \urcorner = \overline{10010}^2 - 1 = 18 - 1 = 17$, and $\ulcorner 12 \urcorner = 101$ since $\overline{1101}^2 - 1 = (8+4+1) - 1 = 13 - 1 = 12$.

- Given any language $L \subseteq \{0,1\}^*$, we write $\mathbb{O}_L \subseteq \mathbb{N}$ for the set

$$\mathbb{O}_L = \left\{ \ulcorner w \urcorner \in \mathbb{N} \mid w \in L \right\} = \left\{ k \in \mathbb{N} \mid \ulcorner k \urcorner \in L \right\}.$$

- Given any subset $\mathbb{O} \subseteq \mathbb{N}$, we write $\mathcal{L}_{(\mathbb{O})} \subseteq \{0,1\}^*$ for the language

$$\mathcal{L}_{(\mathbb{O})} = \left\{ w \in \{0,1\}^* \mid \ulcorner w \urcorner \in \mathbb{O} \right\} = \left\{ \ulcorner k \urcorner \in \{0,1\}^* \mid k \in \mathbb{O} \right\}.$$

So $\mathbb{O}_L$ is the oracle associated with the language L, and $\mathcal{L}_{(\mathbb{O})}$ is the language associated with the oracle $\mathbb{O}$.
Notice that the oracle for the empty language is the empty set: $\mathbb{O}_\varnothing = \varnothing$.

So, we have

(1) a coding for the Turing machines:

$$\begin{array}{ccc} \{\langle, \rangle, q, {\scriptstyle 0}, {\scriptstyle 1}, {\scriptstyle 2}, {\scriptstyle 3}, {\scriptstyle 4}, {\scriptstyle 5}, {\scriptstyle 6}, {\scriptstyle 7}, {\scriptstyle 8}, {\scriptstyle 9}, 0, 1, \sqcup, L, R, \{, \}, (,), ,\}^* & \longleftrightarrow & \{0,1\}^* \\ \mathcal{M} & \longmapsto & \ulcorner \mathcal{M} \urcorner \end{array}$$

(2) a coding for the words:

$$\begin{array}{ccc} \{0,1\}^* & \longleftrightarrow & \mathbb{N} \\ w & \longmapsto & \ulcorner w \urcorner \end{array}$$

(3) a coding for the integers:

$$\begin{array}{ccc} \mathbb{N} & \longleftrightarrow & \{0,1\}^* \\ k & \longmapsto & \ulcorner k \urcorner \end{array}$$

We will use the notation $\ulcorner \mathcal{M} \urcorner$ instead of $\ulcorner\ulcorner \mathcal{M} \urcorner\urcorner$ which means we consider first the word in $\{0,1\}^*$ that codes the Turing machine $\mathcal{M}$, then the integer that codes this word. All we mean is that $\ulcorner \mathcal{M} \urcorner$ is an integer that codes the Turing machine $\mathcal{M}$.

Proposition 7.1

Given any recursive language $L \subseteq \{0,1\}^*$, and any oracle Turing machine $\mathcal{O}^{\mathbb{O}_L}$:

- $\mathcal{L}(\mathcal{O}^{\mathbb{O}_L})$ is recursively enumerable, and moreover
- if $\mathcal{O}^{\mathbb{O}_L}$ is an oracle Decider[a], then $\mathcal{L}(\mathcal{O}^{\mathbb{O}_L})$ is recursive.

[a]meaning that $\mathcal{O}^{\mathbb{O}_L}$ halts on every input.

Proof of Proposition 7.1:

Left as an exercise. $\square$

Definition 7.2: Turing Reducibility

Given any $A, B \subseteq \mathbb{N}$,
A is "Turing reducible" to B – denoted $A \leqslant_T B$ – if there exists an o-c-TM $\mathcal{O}^B$ which on empty tape computes χ_A.

Proposition 7.2

Given any $A, B \subseteq \mathbb{N}$, the following are equivalent:

(1) A is Turing reducible to B,

(2) for every o-c-TM $\mathcal{M}$, there exists an o-c-TM $\mathcal{N}$ such that $\mathcal{L}\left(\mathcal{M}^A\right) = \mathcal{L}\left(\mathcal{N}^B\right)$.
Moreover, in case $\mathcal{M}^A$ is an oracle Decider, we may ensure that $\mathcal{N}^B$ be one too.

Proof of Proposition 7.2:

Left as an easy exercise. $\square$

Notation 7.2

Given any $A, B \subseteq \mathbb{N}$, we write

- $A \leqslant_T B$ if A *is Turing reducible to* B;
- $A \equiv_T B$ if $A \leqslant_T B$ and $B \leqslant_T A$;
- $A <_T B$ if $A \leqslant_T B$ but $B \not\leqslant_T A$.

Notice that we have

- $A \leqslant_T A$ (hence $A \equiv_T A$);
- $(A \leqslant_T B \textit{ and } B \leqslant_T C) \Longrightarrow A \leqslant_T C$ (hence $(A \equiv_T B \textit{ and } B \equiv_T C) \Longrightarrow A \equiv_T C$);
- $A \equiv_T B \iff B \equiv_T A$.

So that $\equiv_T$ is an equivalence relation.

Example 7.2

Given any language $L \subseteq \{0,1\}^*$,

(1) $\mathbb{O}_L \equiv_T \mathbb{O}_{L^{\complement}} = \mathbb{N} \setminus \mathbb{O}_L$,

(2) $\varnothing \leqslant_T \mathbb{O}_L$,

(3) L is recursive $\iff \mathbb{O}_L \equiv_T \varnothing$,

(4) L is not recursive $\iff \varnothing <_T \mathbb{O}_L$.

(5) $\mathbb{O}_L \equiv_T \mathbb{O}_{\mathcal{L}(\mathbb{O}_L)}$ holds since we have $\mathbb{O}_L = \mathbb{O}_{\mathcal{L}(\mathbb{O}_L)}$.

An equivalence class – $[A]_{\equiv_T} = \{B \subseteq \mathbb{N} \mid B \equiv_T A\}$ for some $A \subseteq \mathbb{N}$ – is called a *Turing degree.* The ordering on oracles induces an ordering on the set $\mathbb{TD}$ of all Turing degrees: given any $d, e \in \mathbb{TD}$,

$$d \leqslant e \iff A \leqslant_T B \quad \textit{holds for some } A \in d \textit{ and some } B \in e$$

or equivalently

$$d \leqslant e \iff A \leqslant_T B \quad \textit{holds for all } A \in d \textit{ and all } B \in e$$

As usual, the notation

$$d < e \iff d \leqslant e \ \textit{ but } e \nleqslant d$$

Example 7.3

We list a few basic facts about Turing degrees.

(1) Given any $d \in \mathbb{TD}$

$$card(d) = \aleph_0.$$

The reason is that there are countably many Turing machines and always infinitely many oracle that are Turing equivalent: for instance, given any $A \subseteq \mathbb{N}$ and any $k \in \mathbb{N}$ form

$$A_k = \{2n \mid n \in A\} \cup \{2k+1\}$$

We have both $A \equiv_T A_k$ (any $k \in \mathbb{N}$) and $A_k \neq A_l$ (any $k \neq l \in \mathbb{N}$).

(2) Given any set $A \subseteq \mathbb{N}$ the set

$$\{B \subseteq \mathbb{N} \mid B \leqslant_T A\}$$

is countable for the reason that there are only countably many Turing machines.

(3) Given any $d \in \mathbb{TD}$

$$card\big\{e \in \mathbb{TD} \mid e \leqslant d\big\} \leqslant \aleph_0.$$

(4) Given any $d \in \mathbb{TD}$

$$card\big\{e \in \mathbb{TD} \mid d \leqslant e\big\} = 2^{\aleph_0}.$$

To see this, observe that if

$$d = [A]_{\equiv_T}$$

then given any $B \subseteq \mathbb{N}$ the set

$$A \oplus B = \{2n \mid n \in A\} \cup \{2n+1 \mid n \in B\}$$

satisfies

$$A \leqslant_T A \oplus B.$$

Moreover,

$$card\big\{A \oplus B \mid B \subseteq \mathbb{N}\big\} = 2^{\aleph_0}.$$

Since every Turing degree is countable, we obtain

$$card\left(\big\{A \oplus B \mid B \subseteq \mathbb{N}\big\} /_{\equiv_T}\right) = 2^{\aleph_0}$$

which gives the result.

(5) As Sacks showed in 1961 – see [23] p. 157 and also [40, 41] – the ordering $(\mathbb{TD}, \leqslant)$ does not have a familiar shape since every countable partial ordering $(P, \leqslant)$ can be embedded into $(\mathbb{TD}, \leqslant)$.

Proposition 7.3

(1) $\Big\{\mathcal{L}_{(\mathbb{O})} \subseteq \{0,1\}^* \mid \mathbb{O} \leqslant_T \varnothing\Big\}$ is the class $\mathcal{R}ec.$ of all recursive languages.

(2) $\Big\{L \subseteq \{0,1\}^* \mid \mathbb{O}_L \leqslant_T \mathbb{H}_{alt}\Big\} \supsetneq \mathcal{R}.\mathcal{E}.$ (= the class of all r.e. languages)[a].

(3) $\Big\{\mathcal{L}_{(\mathbb{O})} \subseteq \{0,1\}^* \mid \mathbb{O} \leqslant_T \mathbb{H}_{alt}\Big\} \supsetneq \mathcal{R}.\mathcal{E}.$

Where $\mathbb{H}_{alt}$ stands for the set of codes of Turing machines that halt on the empty input:

$$\begin{aligned}\mathbb{H}_{alt} &= \mathbb{O}_{\{\ulcorner\mathcal{M}\urcorner\in\{0,1\}^* \mid \mathcal{M}(\varepsilon)\downarrow\}} \\ &= \{\ulcorner\mathcal{M}\urcorner \in \mathbb{N} \mid \mathcal{M}(\varepsilon)\downarrow\}.\end{aligned}$$

[a] notice that the inclusion is strict since $\mathbb{H}_{alt}^{\complement}$ satisfies both $\mathbb{H}_{alt}^{\complement} \equiv_T \mathbb{H}_{alt}$ and $\mathcal{L}_{(\mathbb{H}_{alt}^{\complement})} \notin \mathcal{R}ec$.

Proof of Proposition 7.3:

Left as an exercise.

$\square$

We now introduce an operation called the "*jump*" which shows that there is no maximum Turing degree, since from any given oracle A it provides us with some oracle A' that satisfies $A <_T A'$.

Definition 7.3: Jump operator

Given any subset $A \subseteq \mathbb{N}$, the *jump* of A (denoted A') is

$$\begin{aligned}A' &= \mathbb{O}_{\{\ulcorner\mathcal{M}\urcorner\in\{0,1\}^* \mid \mathcal{M} \textit{ an o-c-TM}, \mathcal{M}^A(\varepsilon)\downarrow\}} \\ &= \{\ulcorner\mathcal{M}\urcorner \in \mathbb{N} \mid \mathcal{M}^A(\varepsilon)\downarrow\}.\end{aligned}$$

Example 7.4

$$\mathbb{H}_{alt} \equiv_T \varnothing'.$$

Proposition 7.4

For every $A \subseteq \mathbb{N}$ the set

$$A^\dagger = \{\alpha_2(\ulcorner\mathcal{M}\urcorner, \ulcorner w\urcorner) \in \mathbb{N} \mid \mathcal{M}^A(w)\downarrow\}$$

satisfies

$$A' \equiv_T A^\dagger.$$

$$\left(\text{See page 82 for the definition of } \alpha_2 : \begin{array}{ccc} \mathbb{N}\times\mathbb{N} & \xleftrightarrow{bij.} & \mathbb{N} \\ (x,y) & \longmapsto & \frac{(x+y)\cdot(x+y+1)}{2} + y. \end{array}\right)$$

Proof of Proposition 7.4:

Left as an easy exercise. □

Proposition 7.5

For every $A \subseteq \mathbb{N}$,

$$A <_T A'.$$

Proof of Proposition 7.5:

We decompose $A <_T A'$ into first $A \leqslant_T A'$, then $A' \nleqslant_T A$.

($\mathbf{A \leqslant_T A'}$) We need to find an o-c-TM $\mathcal{M}$ that outputs χ_A while being equipped with the oracle A'. To compute $\chi_A(n)$ this machine proceeds as follows: it computes the code $\ulcorner\mathcal{N}_n\urcorner$ of any o-c-TM $\mathcal{N}_n$ that, no matter what its input w is, proceeds as follows when it is equipped with the oracle $\mathbb{O}$:

- if $\chi_{\mathbb{O}}(n) = 1$, then $\mathcal{N}_n(w)\downarrow$;
- if $\chi_{\mathbb{O}}(n) = 0$, then $\mathcal{N}_n(w)\uparrow$.

Then $\mathcal{M}^{A'}$ outputs

$$\chi_A(n) = \chi_{A'}(\ulcorner\mathcal{N}_n\urcorner).$$

($\mathbf{A' \nleqslant_T A}$) Towards a contradiction, we assume that $A' \leqslant_T A$ holds. Since $A^\dagger \equiv_T A'$ we have $A^\dagger \leqslant_T A$ holds as well. So, there exists an o-c-TM $\mathcal{N}$ such that $\mathcal{N}^A$ computes $\chi_{A^\dagger}$.

We build an o-c-TM $\mathcal{H}$ such that $\mathcal{H}^A$ on every input $w \in \{0,1\}^*$:

(1) computes $k = \alpha_2(\ulcorner w\urcorner, \ulcorner w\urcorner)$, then

(2) by making use of $\mathcal{N}$ as a subprogram, computes the value $\chi_{A^\dagger}(k)$, then
   - if $\chi_{A^\dagger}(k) = 0$, then $\mathcal{H}^A(w)\downarrow$;
   - if $\chi_{A^\dagger}(k) = 1$, then $\mathcal{H}^A(w)\uparrow$.

We obtain the following contradiction:

$$\mathcal{H}^A(\ulcorner\mathcal{H}\urcorner)\downarrow \iff \alpha_2(\ulcorner\mathcal{H}\urcorner,\ulcorner\mathcal{H}\urcorner)\notin A^\dagger \iff \mathcal{H}^A(\ulcorner\mathcal{H}\urcorner)\uparrow .$$

$\square$

Below we show a picture that illustrates this diagonal argument that we have just used.

	$\mathcal{M}_0^A$	$\mathcal{M}_1^A$	$\mathcal{M}_2^A$	$\mathcal{M}_3^A$	$\mathcal{M}_4^A$	$\mathcal{M}_5^A$		$\mathcal{M}_n^A$	
$\ulcorner\mathcal{M}_0\urcorner$	0	1	1	0	1	0	…	0	…
$\ulcorner\mathcal{M}_1\urcorner$	1	1	1	0	0	0	…	0	…
$\ulcorner\mathcal{M}_2\urcorner$	1	0	1	0	0	0	…	1	…
$\ulcorner\mathcal{M}_3\urcorner$	0	0	1	0	1	0	…	0	…
$\ulcorner\mathcal{M}_4\urcorner$	0	1	0	1	1	1	…	0	…
$\ulcorner\mathcal{M}_5\urcorner$	1	1	0	0	0	0	…	0	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮	
$\ulcorner\mathcal{M}_n\urcorner$	1	0	0	0	1	1	…	1	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮	

If $(\mathcal{M}_i)_{i\in\mathbb{N}}$ is a enumeration of all the oracle-compatible Turing machines, then we make sure that the machine $\mathcal{H}$ we built is none of them by ensuring that for each $i\in\mathbb{N}$, there exists an input word (its own code $\ulcorner\mathcal{M}_i\urcorner$) such that $\mathcal{H}^A$ has a completely different behaviour than $\mathcal{M}_i^A$ on this word: we *swap 0's and 1's on the diagonal.*

Corollary 7.1

The following strict ordering between jumps is satisfied:

$$\varnothing <_T \varnothing' <_T \varnothing'' <_T \ldots <_T \varnothing^{\overbrace{''\cdots'}^{n}} <_T \varnothing^{\overbrace{''\cdots'}^{n+1}} <_T \ldots <_T \varnothing^{\overbrace{''\cdots}^{\omega}} <_T \varnothing^{\overbrace{''\cdots'}^{\omega+1}} <_T \ldots .$$

where

$$k\in\varnothing^{\overbrace{''\cdots'}^{\omega}} \iff \begin{cases} k=\frac{(n+m)(n+m+1)}{2}+m \\ \textit{and} \\ m\in\varnothing^{\overbrace{''\cdots'}^{n}}. \end{cases}$$

Proof of Corollary 7.1:

Let us use the notations $\varnothing^{(n)}$ for $\varnothing^{\overbrace{\prime\prime\cdots\prime}^{n}}$ and $\varnothing^{(\omega)}$ for $\varnothing^{\overbrace{\prime\prime\cdots}^{\omega}}$.
The only thing one needs to prove is that

$$\varnothing^{(n)} <_T \varnothing^{(\omega)}$$

holds for every integer n.

$\underline{\varnothing^{(\mathbf{n})} \leqslant_{\mathbf{T}} \varnothing^{(\omega)}}$ is almost immediate, since it is straightforward to build an o-c-TM $\mathcal{O}_n$ that outputs $\chi_{\varnothing^{(n)}}$ when it is equipped with the oracle $\varnothing^{(\omega)}$ since

$$\chi_{\varnothing^{(n)}}(m) = \chi_{\varnothing^{(\omega)}}\left(\frac{(n+m)(n+m+1)}{2} + m\right).$$

$\underline{\varnothing^{(\omega)} \nleqslant_{\mathbf{T}} \varnothing^{(\mathbf{n})}}$ it is enough to proceed by contradiction and show that

$$\varnothing^{(\omega)} \leqslant_T \varnothing^{(n)}$$

would imply

$$\varnothing^{(n+1)} \leqslant_T \varnothing^{(n)}.$$

□

Iterating the jump operator into the transfinite

Notice that if for every limit countable ordinal λ we fix some bijection

$$\begin{array}{rccc} f_\lambda: & \mathbb{N} & \longleftrightarrow & \lambda \times \mathbb{N} \\ & k & \longmapsto & (\alpha, m) \end{array}$$

we may then define an uncountable sequence of jumps $\left(\varnothing^{(\alpha)}\right)_{\alpha<\omega_1}$ by ordinal induction:

- $\varnothing^{(0)} = \varnothing$
- $\varnothing^{(\alpha+1)} = \varnothing^{(\alpha)\prime}$
- $\varnothing^{(\lambda)} = \{f_\lambda(\alpha, m) \in \mathbb{N} \mid m \in \varnothing^{(\alpha)}\}$.

It is immediate to see that the sequence $\left(\varnothing^{(\alpha)}\right)_{\alpha<\omega_1}$ is strictly $<_T$-increasing, or in other words

$$\varnothing^{(\alpha)} <_T \varnothing^{(\beta)}$$

holds for every $\alpha < \beta < \omega_1$.