Part I

Recursivity

Chapter 1

Towards Turing Machines

The whole chapter is highly inspired by Michael Sipser's book: *"Introduction to the Theory of Computation"* [52]. It is a dashing introduction to the notions of Finite Automata, PushDown Automata, Turing Machines.

We also recommend *"Introduction to automata theory, languages, and computation"* by John E. Hopcroft, Rajeev Motwani et Jeffrey D. Ullman [34]; *"Computational complexity"* by Christos H. Papadimitriou [43] and *"A mathematical introduction to logic"* by Herbert B. Enderton [21].

1.1 Deterministic Finite Automata

We will see that any finite automaton can be regarded as a rudimentary Turing machine: a Turing machine that never writes anything and only goes one direction.

Definition 1.1: Deterministic Finite Automaton

A *deterministic finite automaton* (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(1) Q is a finite set called the *states*,

(2) Σ is a finite set called the *alphabet*,

(3) $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function*,

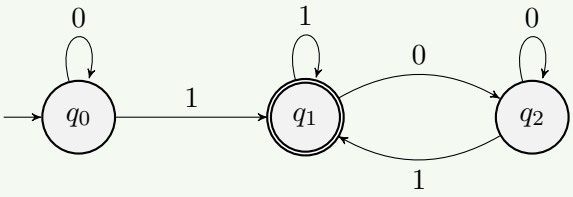
(4) $q_0 \in Q$ is the *initial state*, and

(5) $F \subseteq Q$ is the *set of accepting states*.[a]

[a] Accept states sometimes are called *final states*.

We denote by $\Sigma^{<\omega}$ (or equivalently by Σ^*) the set of finite words on Σ and by ε the empty sequence.

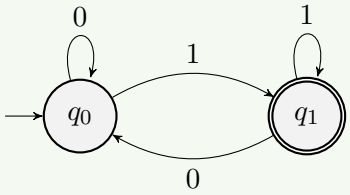
Example 1.1



The DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0, q_1, q_2\}$;
- $\Sigma = \{0, 1\}$;
- q_0 is the *initial state*;
- $F = \{q_1\}$.
- $\begin{aligned} \delta(q_0, 0) &= q_0 \\ \delta(q_0, 1) &= q_1 \\ \delta(q_1, 0) &= q_2 \\ \delta(q_1, 1) &= q_1 \\ \delta(q_2, 0) &= q_2 \\ \delta(q_2, 1) &= q_1 \end{aligned}$

Example 1.2



The DFA $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0, q_1\}$;
- q_0 is the *initial state*;
- $\Sigma = \{0, 1\}$;
- $F = \{q_1\}$.
- $\begin{aligned} \delta(q_0, 0) &= q_0 \\ \delta(q_0, 1) &= q_1 \\ \delta(q_1, 0) &= q_0 \\ \delta(q_1, 1) &= q_1 \end{aligned}$

Definition 1.2

A DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ on an alphabet Σ *accepts* the word $w \in \Sigma^{<\omega}$ if and only if

- either $w = \varepsilon$ (the empty sequence) and $q_0 \in F$
- or $w = \langle a_0, \ldots, a_n \rangle$ with each $a_i \in \Sigma$, and there is a sequence of states $r_0, \ldots, r_{n+1}$ such that:
  - $r_0 = q_0$
  - $\forall i \leqslant n, \delta(r_i, a_i) = r_{i+1}$
  - $r_{n+1} \in F$.

Notation 1.1

Given any DFA $\mathcal{A}$, the language recognized by $\mathcal{A}$ is

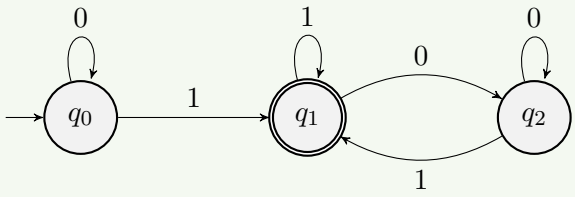
$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^{<\omega} : w \textit{ is accepted by } \mathcal{A}\}.$$

$\mathcal{L}(\mathcal{A})$ denotes the language accepted by $\mathcal{A}$.

Example 1.3

The DFA $\mathcal{A}_1$ below recognizes

$$\mathcal{L}(\mathcal{A}_1) = \{w \in \Sigma^{<\omega} : w \textit{ ends with the letter } 1\}.$$

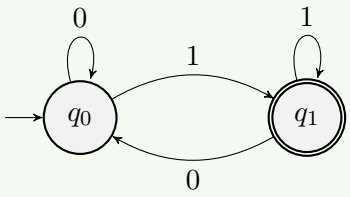


A DFA $\mathcal{A}_1$.

Example 1.4

The DFA $\mathcal{A}_2$ below recognizes

$$\mathcal{L}(\mathcal{A}_2) = \{w \in \Sigma^{<\omega} : w \textit{ ends with the letter } 1\}.$$

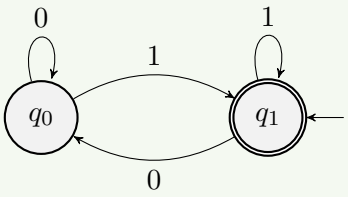


A DFA $\mathcal{A}_2$.

Example 1.5

The DFA $\mathcal{A}$ below recognizes the language

$$\mathcal{L}(\mathcal{A}) = \{\varepsilon\} \cup \{w \in \Sigma^{<\omega} : w \textit{ ends with the letter } 1\}.$$

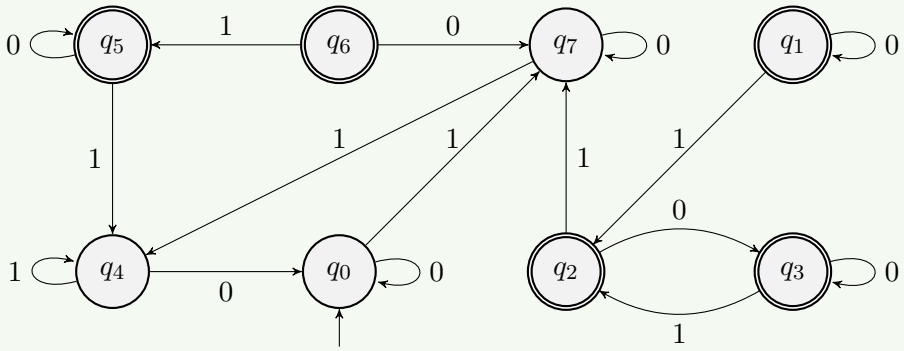


A DFA $\mathcal{A}$.

Example 1.6

The DFA $\mathcal{A}$ below recognizes the language

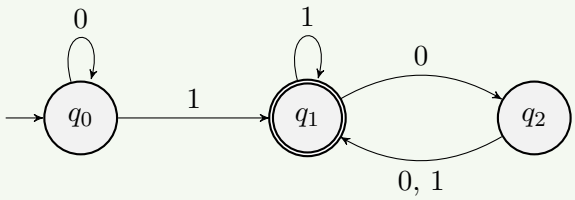
$$\mathcal{L}(\mathcal{A}) = \varnothing.$$



A DFA $\mathcal{A}$.

Example 1.7

What is the language recognized by the DFA below ?



Definition 1.3

Any language recognized by some deterministic finite automata (DFA) is called *regular*.

1.2 Nondeterministic Finite Automata

Given any alphabet Σ, we both assume that $\varepsilon \notin \Sigma$ holds and write Σ_ε for $\Sigma \cup \{\varepsilon\}$.

Definition 2.1

A *nondeterministic finite automaton* (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

(1) Q is a finite set of states,

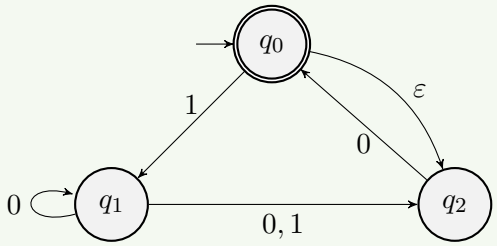
(2) Σ is a finite alphabet,

(3) $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,

(4) $q_0 \in Q$ is the initial state, and

(5) $F \subseteq Q$ is the set of accepting states.

Example 2.1



The NFA $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_0, q_1, q_2\}$;
- $\Sigma = \{0, 1\}$;
- q_0 is the *initial state*;
- $F = \{q_0\}$.

- $\delta(q_0, \varepsilon) = \{q_2\}$
  $\delta(q_0, 0) = \varnothing$
  $\delta(q_0, 1) = \{q_1\}$
  $\delta(q_1, \varepsilon) = \varnothing$
  $\delta(q_1, 0) = \{q_1, q_2\}$
  $\delta(q_1, 1) = \{q_2\}$
  $\delta(q_2, \varepsilon) = \varnothing$
  $\delta(q_2, 0) = \{q_0\}$
  $\delta(q_2, 1) = \varnothing$

Definition 2.2

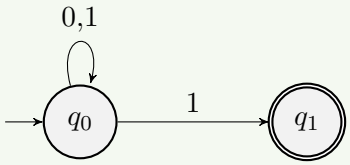
Let $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w \in \Sigma^{<\omega}$. We say that $\mathcal{N}$ *accepts* w if and only if

- either $w = \varepsilon$ the empty sequence and $q_0 \in F$
- or w can be written as $w = \langle a_0, \ldots, a_n \rangle$ with each $a_i \in \Sigma_\varepsilon$, and there is and a sequence of states $r_0, \ldots, r_{n+1}$ such that:
  - $r_0 = q_0$,
  - $\forall i \leqslant n \quad r_{i+1} \in \delta(r_i, a_i)$,
  - $r_{n+1} \in F$.

Example 2.2

The NFA $\mathcal{N}$ below recognizes the language

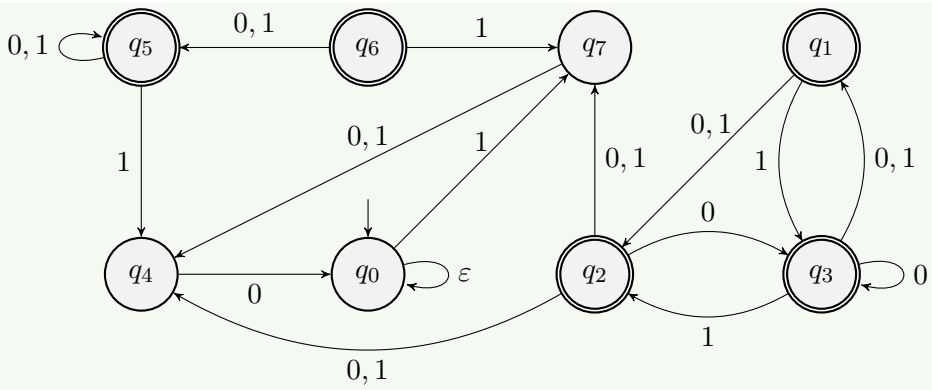
$$\mathcal{L}(\mathcal{N}) = \{w \in \Sigma^{<\omega} : w \textit{ ends with the letter } 1\}.$$



The NFA $\mathcal{N}$

Example 2.3

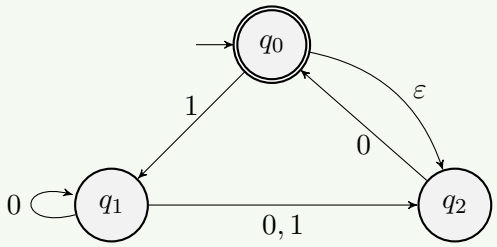
The NFA $\mathcal{N}$ below recognizes the language $\mathcal{L}(\mathcal{N}) = \varnothing$.



An NFA $\mathcal{N}$.

Example 2.4

What is the language recognized by the NFA $\mathcal{N}$ below ?



The NFA $\mathcal{N}$

Proposition 2.1

Every NFA has an equivalent DFA. i.e., given any NFA $\mathcal{N}$ there exists some DFA $\mathcal{D}$ such that

$$\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{D}).$$

Proof of Proposition 2.1:

Given any NFA $\mathcal{N} = \langle Q, \Sigma, \delta, q_0, F \rangle$, we build some DFA $\mathcal{D} = \langle Q', \Sigma, \delta', q_0', F' \rangle$ that recognizes the same language.

(1) $Q' = \mathcal{P}(Q)$

(2) For $S \subseteq Q$ and $a \in \Sigma$ we set

$$\delta'(S, a) = \{q_j \in Q \mid \exists q \in S \;\; q \xrightarrow{\varepsilon^* a \varepsilon^*} q_j\}$$

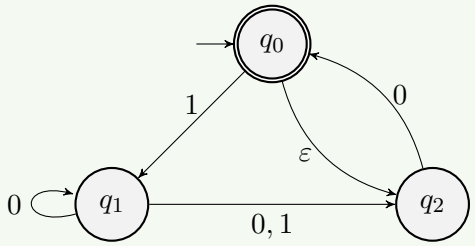
where $q \xrightarrow{\varepsilon^* a \varepsilon^*} q'$ stands for the existence of a path in the graph of $\mathcal{N}$ that goes through exactly one edge labelled with "a", the others being labelled with "ε".

(3) $q'_0 = \{q \in Q \mid q_0 \xrightarrow{\varepsilon^*} q\}$

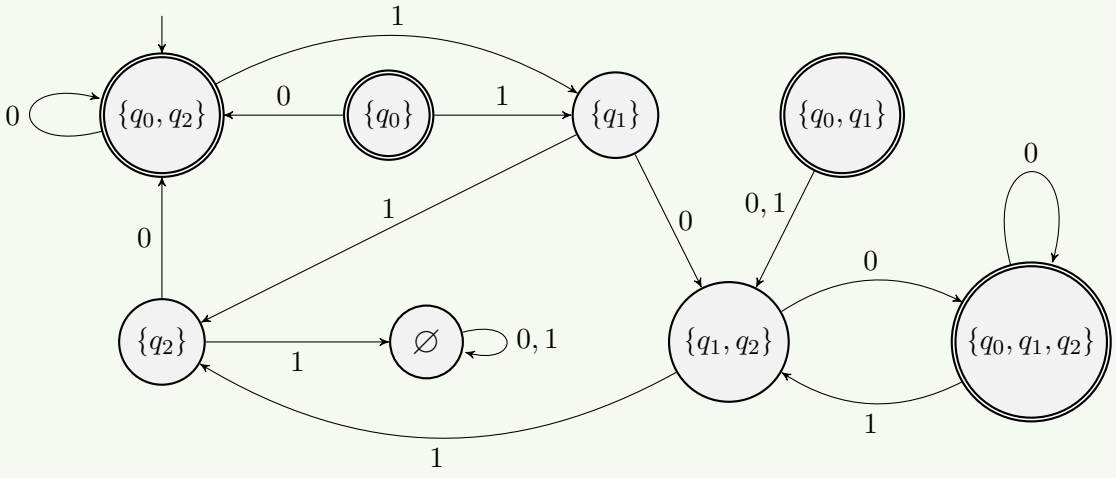
(4) $F' = \{S \subseteq Q \mid S \cap F \neq \varnothing\}$.

□

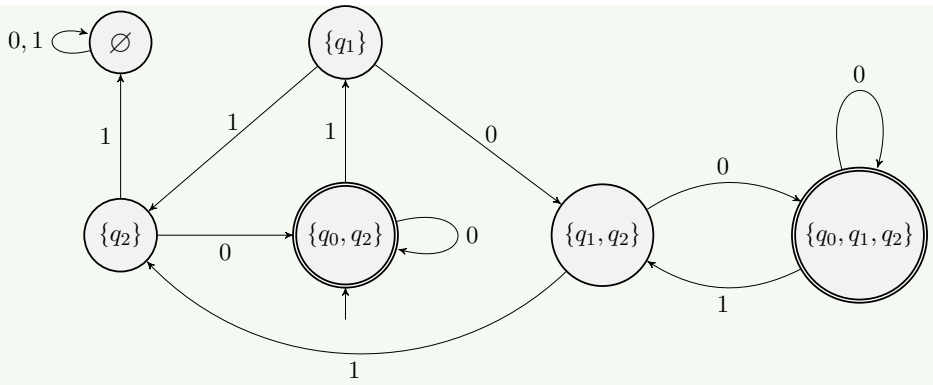
Example 2.5



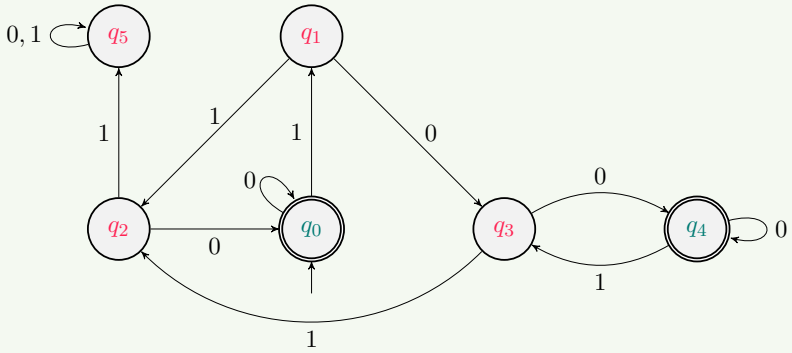
The NFA $\mathcal{N}$



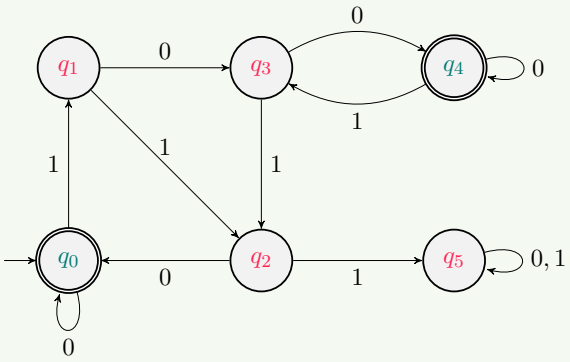
A DFA $\mathcal{A}$ equivalent to the above NFA $\mathcal{N}$.



A DFA $\mathcal{A}'$ equivalent to the above DFA $\mathcal{A}$.



A DFA $\mathcal{A}''$ equivalent to the above DFA $\mathcal{A}'$.



The above DFA $\mathcal{A}''$ presented differently.

Definition 2.3

Let A and B be languages. We define the regular operations *union*, *concatenation*, and *star* as follows.

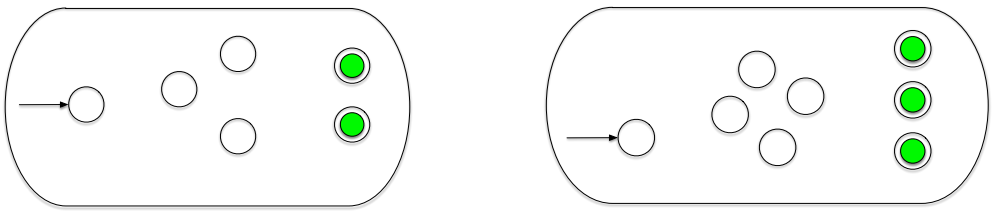
- *Union*: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- *Concatenation*: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- *Star*: $A^* = \{x_1 x_2 \dots x_k \mid k \geqslant 0 \text{ and each } x_i \in A\}$.

Theorem 2.1

Regular languages are closed under union, concatenation and the star operation.

Proof of Theorem 2.1:

Let $\mathcal{N}_1 = (Q_1, \Sigma, \Delta_1, q_1, F_1), \mathcal{N}_2 = (Q_2, \Sigma, \Delta_2, q_2, F_2)$ be two NFAs recognising respectively A_1 and A_2.



$\mathcal{N}_1$ $\mathcal{N}_2$

Union We need an NFA $\mathcal{N}$ such that $\mathcal{N}$ recognizes a string if and only if $\mathcal{N}_1$ or $\mathcal{N}_2$ recognizes it. By working nondeterministically, the automaton $\mathcal{N}$ is allowed to split into two copies: we construct $\mathcal{N}$ in such a way that $\mathcal{N}_1$ and $\mathcal{N}_2$ work in parallel at the same time. We assume $Q_1 \cap Q_2 = \varnothing$ and $q_0 \notin Q_1 \cup Q_2$. Define $\mathcal{N} = (Q, \Sigma, \Delta, q_0, F)$ where

(1) $Q = \{q_0\} \cup Q_1 \cup Q_2$.

(2) $\Delta \subset Q \times \Sigma_\varepsilon \times Q$ is defined by: $(p, s, r) \in \Delta$ if and only if one of the following is true

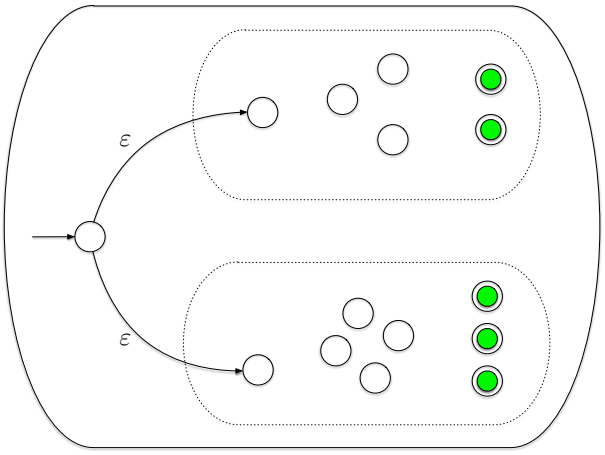
(a) $p = q_0$
$s = \varepsilon$
$r \in \{q_1, q_2\}$

(b) $p, r \in Q_1$
$(p, s, r) \in \Delta_1$

(c) $p, r \in Q_2$
$(p, s, r) \in \Delta_2$.

(3) $F = F_1 \cup F_2$.

The machine splits immediately into two copies of itself, which work exactly as $\mathcal{N}_1$ and $\mathcal{N}_2$. It accepts a string if and only if at least one of the two main copies ends up in an accepting state, i.e., in F_1 or in F_2, i.e., if and only if $\mathcal{N}_1$ or $\mathcal{N}_2$ accept it.



$\mathcal{N}$

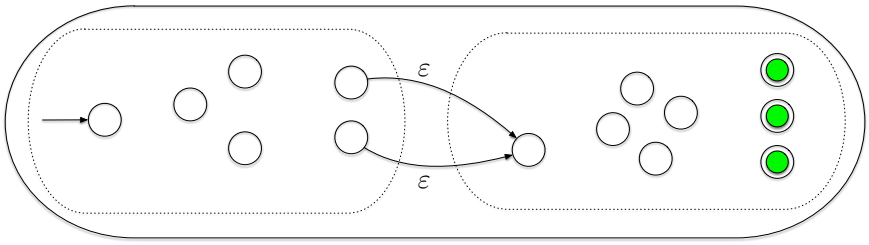
Concatenation Here we need an NFA $\mathcal{N}$ that accepts a word w if and only if w can be broken into two pieces: a prefix and a suffix $w = w_p w_s$ such that w_p is accepted by $\mathcal{N}_1$ and w_s is accepted by $\mathcal{N}_2$. We set q_1 as the initial state and let the machine read the same way $\mathcal{N}_1$ would do. Any time that $\mathcal{N}_1$ finds itself in an accepting state, we want $\mathcal{N}$ to non-deterministically start reading as if it were $\mathcal{N}_2$ but still remaining a copy of itself: so we make it split any time it comes to some final state of $\mathcal{N}_1$. The reason is that we want to be able to check longer sub-strings as well, because it might be the case that the first prefix that is found to be accepted by $\mathcal{N}_1$ corresponds to a suffix that is rejected by $\mathcal{N}_2$, while there is a longer prefix which is also accepted by $\mathcal{N}_1$ that yields a suffix which is this time also accepted by $\mathcal{N}_2$. Formally, we define Δ by: $(p, s, r) \in \Delta$ if and only if one of the following is true

(1) $p, r \in Q_1$
$(p, s, r) \in \Delta_1$

(2) $p, r \in Q_2$
$(p, s, r) \in \Delta_2$

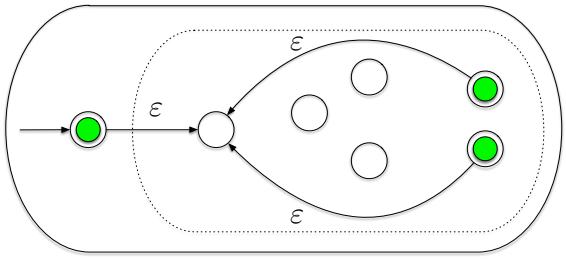
(3) $p \in F_1$
$s = \varepsilon$
$r = q_2$

The third condition guarantees the splitting. Finally, we set the accepting set to be $F = F_2$.



$\mathcal{N}$

Star Here the machine $\mathcal{N}$ should be able to check if a word w can be broken into a finitely many pieces $w = w_1 w_2 \cdots w_k$, each of them being accepted by $\mathcal{N}_1$. So $\mathcal{N}$ has to read w_1 as if it were $\mathcal{N}_1$, and when it finds itself in an accepting state, it needs to start all over again and read w_2 and so on and so forth. The construction is similar to the one of the concatenation, but since A_1^* contains the empty string, we want $\mathcal{N}$ to accept ε. So we just add an initial state q_0 which is also an accepting state, and from where the initial state of $\mathcal{N}_1$ is reached by an ε move.



$\mathcal{N}$

□

1.3 Regular Expressions

Definition 3.1

We say that R is a *regular expression* if R is of one the following form:

(1) a (for some $a \in \Sigma$) (3) $\varnothing$ (5) $R_1 \circ R_2$

(2) ε (4) $R_1 \cup R_2$ (6) ${R_1}^*$

where R_1 and R_2 are regular expressions.

The expression ε represents the language containing a single sequence, namely, the empty sequence, whereas $\varnothing$ represents the language that doesn't contain any sequence. Notice that

(1) $R \circ \varnothing = \varnothing \circ R = \varnothing$ (2) $\varnothing^* = \{\varepsilon\}$.

Definition 3.2

Let R be a regular expression. We define by induction its associated language $L(R)$ as follows:

(1) $L(a) = \{a\}$ (4) $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$

(2) $L(\varepsilon) = \{\varepsilon\}$ (5) $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$

(3) $L(\varnothing) = \varnothing$ (6) $L(R_1^*) = L(R_1)^*$.

Theorem 3.1

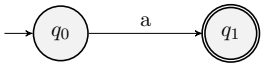
A language L is regular if and only if there exists a regular expression R such that $L = L(R)$.

Proof of Theorem 3.1:

(⇐) Given any any regular expression R, we show, by induction on the length of R, that the language $L(R)$ is recognized by some NFA.

(1) If the length of R is 1:

(a) $L(a) = \{a\}$



(b) $L(\varepsilon) = \{\varepsilon\}$

(c) $L(\varnothing) = \varnothing$

(2) If the length of R is larger than 1, we need to consider the following regular expressions.

(a) $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$

(b) $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$

(c) $L(R_1^*) = L(R_1)^*$.

All three results derive immediately from Theorem 2.1.

(⇒) (1) We go from some n-states DFA to some $n + 2$-states Generalized-NFA:

(a) we add

(A) an initial state "s"

(B) an accepting state "a"

(C) a transition $s \xrightarrow{\varepsilon} q_0$

(D) a transition $q \xrightarrow{\varepsilon} a$ (each accepting state $q \neq a$)

(b) we reduce the set of accepting states to $\{a\}$.

(2) We go from some $k + 1 + 2$-states Generalized-NFA [a] to some $k + 2$-states Generalized-NFA by removing one state from the original automaton: $q_{rip} \notin \{s, a\}$ and for each states $q_{in} \notin \{a, q_{rip}\}$ and $q_{out} \notin \{s, q_{rip}\}$ we set the new transition to be:

$$q_{in} \xrightarrow{R_{in\to rip} \circ (R_{rip\to rip})^* \circ R_{rip\to out} \cup R_{in\to out}} q_{out}$$

where $R_{in\to rip}$, $R_{rip\to rip}$, $R_{rip\to out}$ and $R_{in\to out}$ denote the following transitions:

(a) $q_{in} \xrightarrow{R_{in\to rip}} q_{rip}$

(b) $q_{rip} \xrightarrow{R_{rip\to rip}} q_{rip}$

(c) $q_{rip} \xrightarrow{R_{rip\to out}} q_{out}$

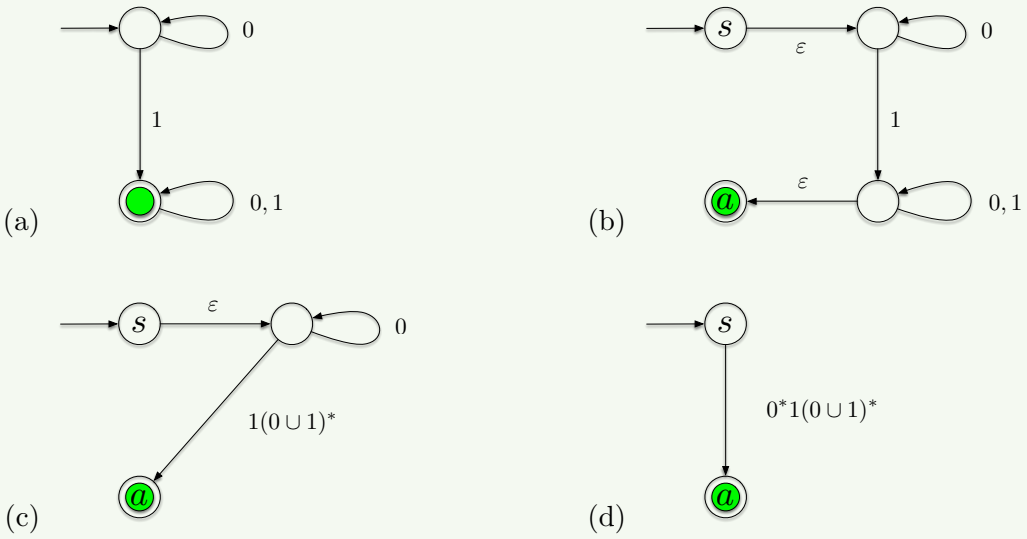
(d) $q_{in} \xrightarrow{R_{in\to out}} q_{out}$.

(3) We end up with a 2-states ("s" and "a") Generalized-NFA with a single transition of the form $s \xrightarrow{R} a$. The regular expression R gives the solution.

□

[a] an NFA whose transitions are labelled with regular expressions.

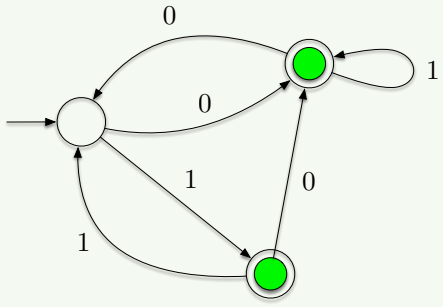
Example 3.1



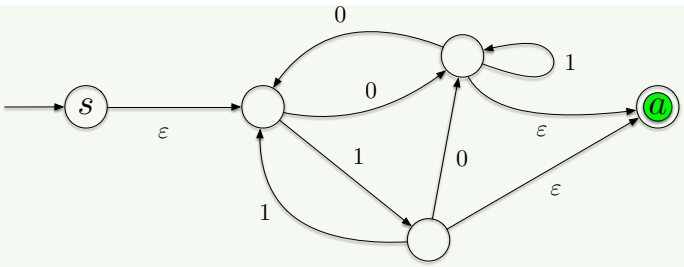
An other example with a more complicated automaton.

Example 3.2

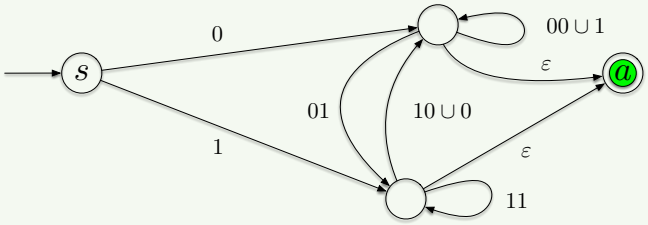
(1)



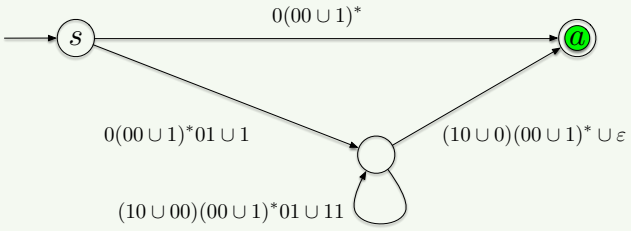
(2)



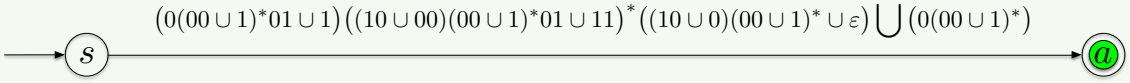
(3)



(4)



(5)



1.4 Non-Regular Languages

Notice that any finite word on Σ can be coded by an integer, so that there are only $\aleph_0$ many regular languages. But there are $2^{\aleph_0}$ many languages for there are as many as the number of subsets of $\mathbb{N}$. Hence most languages are not regular!

Theorem 4.1: Pumping Lemma

If A is a regular language, then there is a number p (the pumping length) where, if s is

any sequence in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

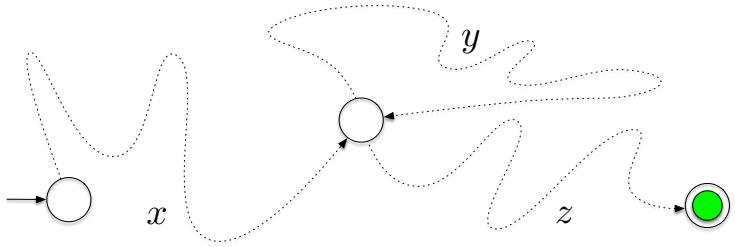
(1) for each $i \geqslant 0$, $xy^iz \in A$,

(2) $|y| > 0$, and

(3) $|xy| \leqslant p$

Proof of Theorem 4.1:

Let $\mathcal{A}$ be any DFA such that $\mathcal{L}(\mathcal{A}) = A$. Set p to be the number of states of $\mathcal{A}$. Let s be accepted by $\mathcal{A}$. Then s may be broken into three pieces: $s = xyz$. Such that the path $q_0 \xrightarrow{x} q$ never visits twice the same state. The path $q \xrightarrow{y} q$ visits twice the state q but none of the others twice. This holds since for every word u of length at least p every path $q' \xrightarrow{u} q$" in $\mathcal{A}$ visits at least twice the same state.



□

Example 4.1

The language $\{0^n1^n \mid n \in \mathbb{N}\}$ is not regular.
By contradiction, assume there exists some DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ which recognizes $\{0^m1^m \mid n \in \mathbb{N}\}$. We consider $p = |Q|$ the number of states of $\mathcal{A}$. the word 0^p1^p is accepted by $\mathcal{A}$. By the previous Pumping Lemma there exist x, y and z such that $0^p1^p = xyz$ and

(1) for each $i \geqslant 0$, $xy^iz \in \{0^n1^n \mid n \in \mathbb{N}\}$,

(2) $|y| > 0$, and

(3) $|xy| \leqslant p$

But since $|xy| \leqslant p$, it turns out that $xy \in 0^*$ and $z \in 0^*1^*$. Therefore, for each integer $i > 1$ we have $xy^i \in 0^*$, hence xy^iz contains too many 0's compared to 1's: a contradiction.

1.5 Pushdown Automata

Definition 5.1

A *pushdown automaton* (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ and F are all finite sets, and

(1) Q is the set of states,

(2) Σ is the input alphabet,

(3) Γ is the stack alphabet,

(4) $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function[a],

(5) $q_0 \in Q$ is the initial state, and

(6) $F \subseteq Q$ is the set of accepting states.

[a] For a deterministic version, replace $\mathcal{P}(Q \times \Gamma_\varepsilon)$ by $Q \times \Gamma_\varepsilon$.

Definition 5.2

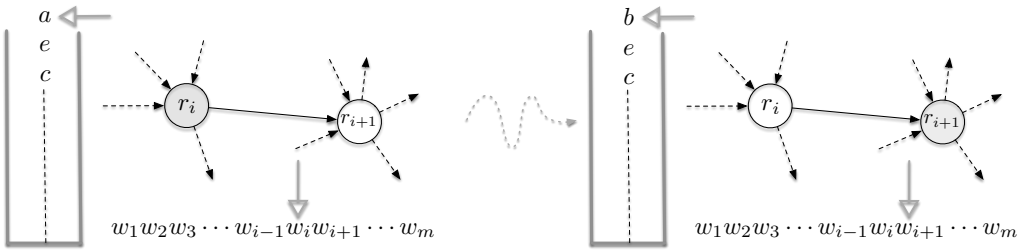
A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows. It accepts input w if w can be written as $w = w_1 w_2 \dots w_m$, where each $w_i \in \Sigma_\varepsilon$ and sequences of states $r_0, r_1, \dots, r_m \in Q$ and sequences $s_0, s_1, \dots, s_m \in \Gamma^{<\omega}$ exist that satisfy the next three conditions. The sequence $(s_i)_{i \leqslant m}$ represent the sequence of stack contents that M goes through on the accepting branch of the computation.

(1) $r_0 = q_0$ and $s_0 = \varepsilon$. This condition testifies that M starts out properly: both in the initial state and with an empty stack.

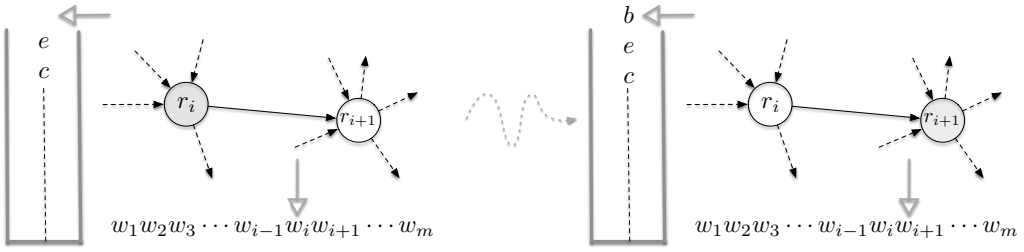
(2) For $i = 0, \dots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^{<\omega}$. This condition states that M moves properly according to the state, stack, and next input symbol.

(3) $r_m \in F$. This condition states that an accepting state occurs right at the end of the reading of the input.

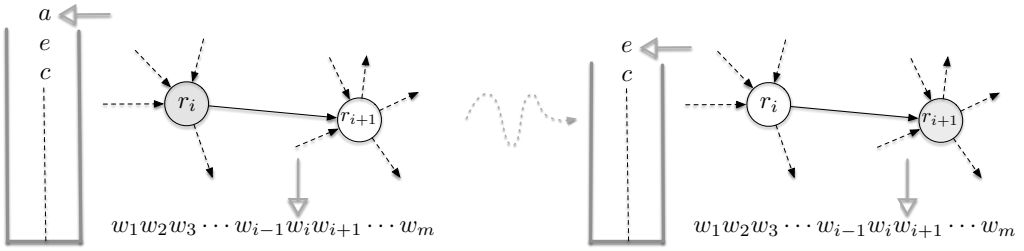
(1) One step of a computation:



(2) The special case where $a = \varepsilon$ and $b \in \Gamma$ (the PDA "pushes" b to the top of the stack)

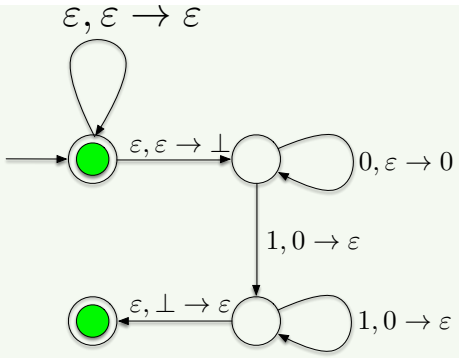


(3) The special case where $a \in \Gamma$ and $b = \varepsilon$ (the PDA "pops off" a from the top of the stack)

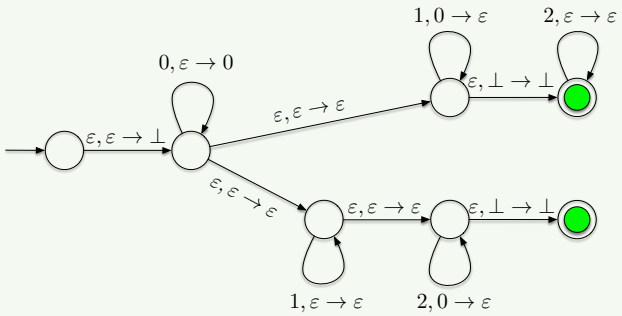


Example 5.1

(1) The language $\{0^m1^m \mid m \in \mathbb{N}\}$ is recognized by the following PDA.



(2) The language $\{0^i 1^j 2^k \mid i, j, k \geqslant 0 \text{ and } i = j \text{ or } i = k\}$ is recognizable by the following PDA, however it is not recognizable by a deterministic PDA.



1.6 Context-Free Grammar

Definition 6.1

A *context-free grammar* is a 4-tuple (V, Σ, R, S), where

(1) V is a finite set whose elements are called *variables*,

(2) Σ is a finite set, disjoint from V. Its elements are called *terminals*,

(3) R is a finite set of *rules*. Each rule is a couple of the form (ξ, u) where $\xi \in V$ and $u \in (V \cup \Sigma)^*$ [a].

(4) $S \in V$ is the initial variable.

[a] In particular, one may have $u = \varepsilon$.

If u, v and w are sequences of variables and terminals, and $A \to w$ is a rule of the grammar, we say that uAv *yields* uwv (written $uAv \Rightarrow uwv$). We write $u \Rightarrow^* v$ if $u = v$ or if a sequence

$u_1, u_2, \ldots, u_k$ exists for $k > 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

The *language generated by the grammar* is $\{w \in \Sigma^{<\omega} \mid S \Rightarrow^* w\}$.

Example 6.1

Consider (V, Σ, R, S) the *context-free grammar* where $V = \{S\}$, $\Sigma = \{0, 1, \sharp\}$ and R is the following set of production rules:

- $S \longrightarrow 0S1$
- $S \longrightarrow \sharp$

This grammar generates the language $\{0^n \sharp 1^n \mid n \in \mathbb{N}\}$.

Example 6.2

Consider (V, Σ, R, S) the *context-free grammar* where $V = \{S\}$, $\Sigma = \{0, 1\}$ and R is the following set of production rules:

- $S \longrightarrow 0S1$
- $S \longrightarrow \varepsilon$

This grammar generates the language $\{0^n 1^n \mid n \in \mathbb{N}\}$.

Example 6.3

Consider (V, Σ, R, S) the *context-free grammar* where $V = \{S, A, B, C, D\}$, $\Sigma = \{0, 1, 2\}$ and R is the following set of production rules:

- $S \longrightarrow AB$
- $S \longrightarrow C$
- $A \longrightarrow 0A1$
- $A \longrightarrow \varepsilon$
- $B \longrightarrow B2$
- $B \longrightarrow \varepsilon$
- $C \longrightarrow 0C2$
- $C \longrightarrow D$
- $D \longrightarrow D1$
- $D \longrightarrow \varepsilon$

which are usually summarized by:

- $S \longrightarrow AB \mid C$
- $A \longrightarrow 0A1 \mid \varepsilon$
- $B \longrightarrow B2 \mid \varepsilon$
- $C \longrightarrow 0C2 \mid D$
- $D \longrightarrow D1 \mid \varepsilon$

This grammar generates the language $\{0^i 1^j 2^k \mid i, j, k \geqslant 0 \text{ and } i = j \text{ or } i = k\}$.

Theorem 6.1

A language is recognized by a PDA if and only if it is context-free.

Proof of Theorem 6.1:

($\Leftarrow$) Get a context-free grammar. First notice that it is equivalent to add strings to the stack all at once or one at a time[a]. The Pushdown P works as follows:

(1) P has four states: the initial state, the final accepting state, the final rejecting state, and a state called "*Loop*"

(2) While in the initial state, P places a marker symbol "$\perp$" followed by the start variable S inside the stack, and goes to the "*Loop*" state. (So that the stack content is now $S\perp$.)

(3) While in the *Loop* state,

(a) If the top stack is a variable A, then P selects *non-deterministically* one of the rules for A and substitutes A by the string on the right hand side of the rule and remains in the *Loop* state.

(b) If the top stack is a terminal symbol a, then P reads the next input symbol from the input and compares it to a.

(A) If they don't match, P enters the final rejecting state (hence this branch of non-deterministic computation is rejected).

(B) If they do match, P pops off the terminal symbol a from the top of the stack, remains in the *Loop* state and starts (3) again.

(c) If the top of stack is "$\perp$", then P enters the final rejecting state. Notice that the only way the input word can be accepted is if no letter (from the input) remains to be read – since there is no transition from this final sate.

($\Rightarrow$) We start from a PDA and construct P an equivalent one such that

(1) P has a single accepting state $q_{acc.}$

(2) It empties its stack before accepting

(3) Each transition either *pushes* a symbol onto the stack or *pops* one off, but does not do both at the same time so that the content of the stack never stays put.

From $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{acc.}\})$ we construct the context-free grammar G.

(1) $V = \{A_{pq} \mid p, q \in Q\}$,

(2) Σ is unchanged,

(3) the start variable is $Aq_0, q_{acc.}$.

(4) The set of rule R is:

(a) For each $p, q, r, s \in Q$, $a \in \Gamma$ and $e, f \in \Sigma_\varepsilon$ if $\delta(p, e, \varepsilon)$ contains (r, a) and $\delta(s, f, a)$ contains (q, ε) put the rule $A_{pq} \to e\ A_{rs} f$ in R.

(b) For each $p, q, r \in Q$ put the rule $A_{pq} \to A_{pr} A_{rq}$ in R.

(c) For each $p \in Q$ put the rule $A_{pp} \to \varepsilon$ in R.

Why is it the case that the language recognized by P is the one derived by G?

($\Rightarrow$) If w is accepted by P, then there exists a computation that accepts it. Notice that by construction, this computation never leaves the stack content still and the automaton ends with an empty stack. So, when something is pushed in the stack, it must be popped off later on. So, the whole computation which goes from q_0 to $q_{acc.}$ determines one derivation.

($\Leftarrow$) Any successful derivation induces an accepting computation.

□

[a] This is an easy exercise to construct from any given PDA that can push finite words to the stack, another one that only pushes letters.

Every regular language is context-free. But many languages are neither regular nor context-free.

Theorem 6.2: Pumping Lemma for Context-Free Languages

If A is a context-free language, then there is a number p (the pumping length) where, if s is any sequence in A of length at least p, then s may be divided into five pieces, $s = vwxyz$, satisfying the following conditions:

(1) for each $i \geqslant 0$, $vw^i xy^i z \in A$,

(2) $|wy| > 0$, and

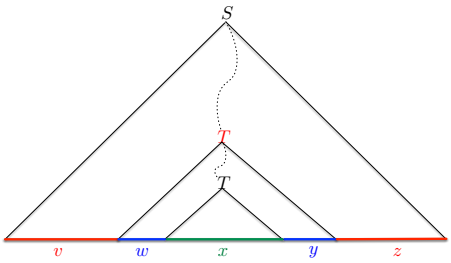
(3) $|wxy| \leqslant p$

Proof of Theorem 6.2:

See Theorem 2.19 in [52]. We first fix a grammar. Then we concentrate on getting a derivation tree[a] large enough so that there is one path – from the root to some leaf – that

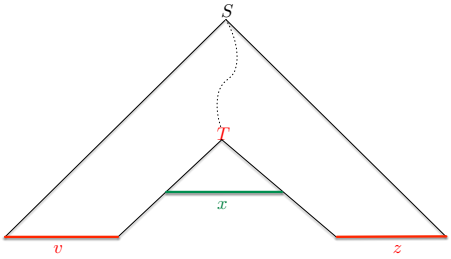
visits twice the same variable T. For this, if k is the number of variables in the grammar, we need a tree of height at least $k+1$. We take m to be the maximum number of symbols in the right hand side of a rule [b], and take $n = \max(2, m)$. Every word of height at least n^{k+1} that is generated by this grammar has a derivation tree with at least one branch whose length is $\geqslant k+1$. We set $p = n^{k+1}$.

Take any word u generated by this grammar such that $|u| \geqslant p$ holds. Consider the smallest – in terms of nodes – derivation tree that produces u, and consider a node T which repeats only once and such that there is no other variable that repeats in the subtree induced by this node. The whole derivation tree is described below:

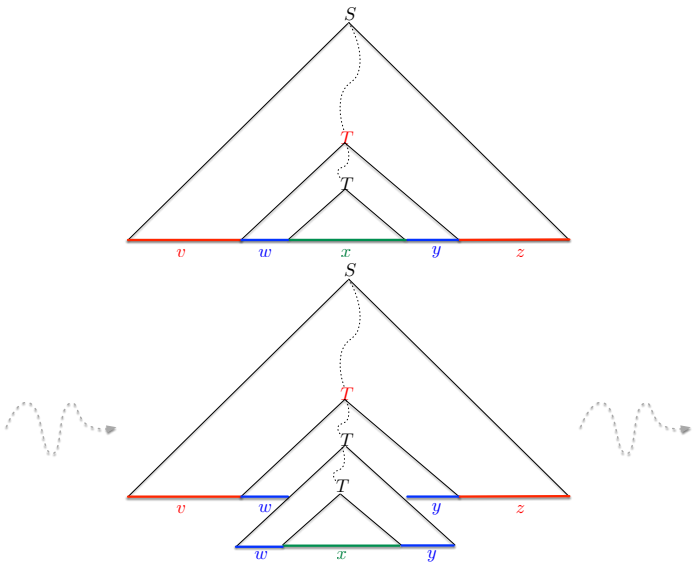


Notice that $|wxy| \leqslant p$ holds, because the subtree induced by T has never twice the same variable (except for T itself which appears only twice). Hence every branch on this subtree has length at most $k+1$, which guarantees that wxy has length at most $p = n^{k+1}$.

Notice also that $|wy| > 0$ because otherwise, we would have $w = y = \varepsilon$. But then the derivation tree below would also produce the same word which would contradict the minimality of the one we chose.



We also clearly have, for each $i \geqslant 0$, $vw^ixy^iz \in A$:



□

[a] notice that in a derivation tree every leaf is a terminal symbol, and very other node is a variable.
[b] k is the maximum number of immediate successors of a node in the derivation tree.

Example 6.4

The following language is not context-free:

$$\{0^n 1^n 2^n \mid n \in \mathbb{N}\}.$$

Towards a contradiction we assume that this language is context-free so that there exists some integer p that verifies the conditions of Theorem 6.2. We consider the word $u = 0^p 1^p 2^p \in A$. By Theorem 6.2, there exist words v, w, x, y, z such that $u = vwxyz$ and

(1) $vw^i xy^i z \in A$ $(\forall i \geqslant 0)$ (2) $|wy| > 0$, and (3) $|wxy| \leqslant p$

Since $|wxy| \leqslant p$ holds, this word cannot contain all three letters 0,1 and 2. We distinguish two different cases:

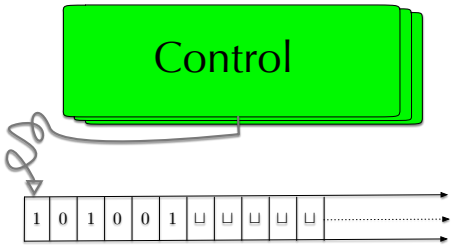
(1) if $wxy \in 0^*1^*$, then $z \in 1^*2^*$. Therefore for each $i > 1$ $vw^i xy^i z$ contains either more 0's than 2's or 1's than 2's.

(2) if $wxy \in 1^*2^*$, then $v \in 0^*1^*$. Therefore for each $i > 1$ $vw^i xy^i z$ contains either more 1's than 0's or 2's than 0's.

Chapter 2

Turing Machines

A Turing Machine (TM) is a general model of computation introduced in 1936 by Alan Turing [60]. 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) Turing machine 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 Turing machine 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 $w_0w_1 = w$. 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_0qw_1 to say that

- the tape content is $w_0w_1 \sqcup \sqcup \sqcup \ldots$
- 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_0w.
An *halting configuration* is

- either an *accepting configuration* of the form $w_0q_{acc.}w_1$,
- or a *rejecting configuration* of the form $w_0q_{rej.}w_1$.

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 $u, v \in \Gamma^{<\omega}$ such that

- either $C = uaq_ibv$, $C' = uq_jacv$ and $\delta(q_i, b) = (q_j, c, L)$,
- or $C = q_ibv$, $C' = q_jcv$ and $\delta(q_i, b) = (q_j, c, L)$,
- or $C = uq_ibv$, $C' = ucq_jv$ and $\delta(q_i, b) = (q_j, c, R)$.

Definition 1.2

A Turing machine accepts input w if there is a sequence of configurations $C_0, \ldots, C_k$ 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 Turing machine $\mathcal{M}$ is the language it recognizes:

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

Example 1.1

A Turing machine that recognizes

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

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

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

If we rename the states :

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

the transition function becomes:

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

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

Definition 1.4

A language L is *Turing recognizable* if there exists a Turing machine $\mathcal{M}$ 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 Turing machine that halts on all inputs.

Definition 1.6

A language is *Turing decidable* iff there exists a *Decider* that recognizes 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 Turing machine is the same as a Turing machine 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

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

Proposition 1.3

Given any Turing machine there exist

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

(2) a multi-tape Turing machine,

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

Proof of Proposition 1.3:

Left as an exercise. □

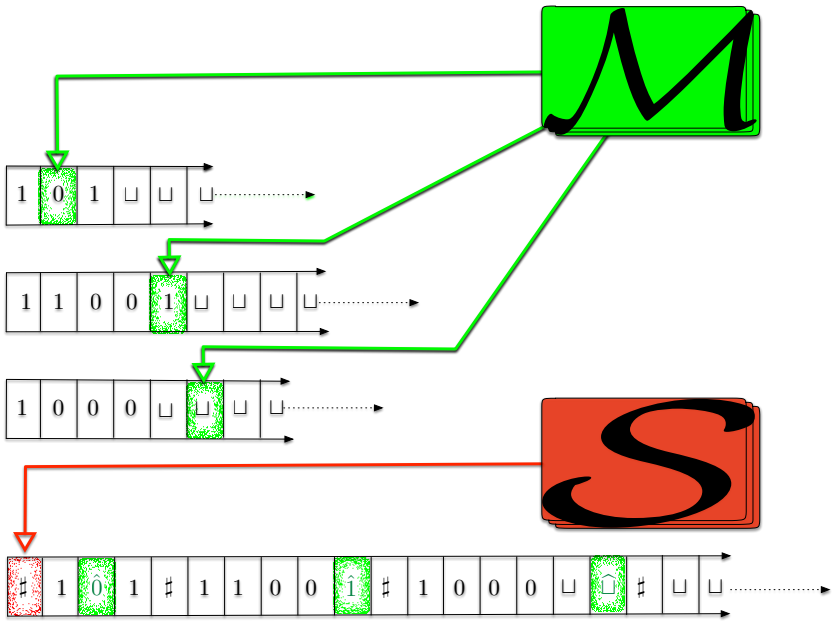
Theorem 1.1

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

Proof of Theorem 1.1:

Let $\mathcal{M}$ be a multi-tape Turing machine. We will describe a Turing machine $\mathcal{S}$ that recognizes 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 ♯, one unit to the right. Then it continues the simulation as before.



□

2.2 Non-Deterministic Turing Machines

Definition 2.1

A non-deterministic Turing machine (NTM) is the same as a deterministic Turing machine 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) Turing machine is a sequence of configurations

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

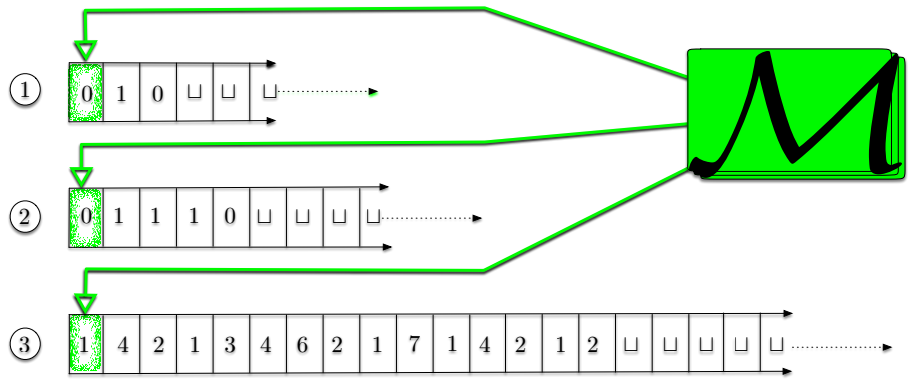
that may be finite or infinite.
It accepts the input if this sequence is finite and the last configuration is an accepting one.
The computation of a non-deterministic Turing machine is no more a sequence of configurations but a tree whose nodes are configurations. This tree may have both infinite and 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 Turing machine that recognizes the same language.

Proof of Theorem 2.1:



We consider a 3-tape (①,② and ③) deterministic Turing machine $\mathcal{M}$ 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:

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

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 $q_0 w$ there are only two non-deterministic choices 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 Turing machine 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 Turing machine if

- either the Turing machine halts and $a_0a_1a_2a_3\ldots\ldots$ is what shows on tape p, or
- the Turing machine 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 Turing machine 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

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

(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

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

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

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

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:

(⇒) 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.)

(⇐) From $\mathcal{E}$ we build a k-tape Turing machine $\mathcal{M}$. 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.

□

Proposition 2.2

For any infinite $L \subseteq \Sigma^{<\omega}$,

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

1. Whose head on the printing tape can stay put.

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

Proof of Proposition 2.2:

Left as an exercise.

□

2.3 The Concept of Algorithm

In 1900, Hilbert gave a list of the main mathematical problems of the time [31, 32]. The 10^{th} 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[2] that the 10^{th} problem of Hilbert is undecidable [40]: 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

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

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

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

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
  - $\mathcal{M}(w) \downarrow^{acc}$ means that $\mathcal{M}$ stops in an accepting configuration, and
  - $\mathcal{M}(w) \downarrow^{rej}$ 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.

We notice the following:

(1) Given any finite alphabet Σ, and any Turing machine $\mathcal{M}$ whose alphabet is Σ, there exists one[3] *Turing computable* coding: $c : \Sigma^{<\omega} \longrightarrow \{0,1,\sqcup\}^{<\omega}$ and a Turing machine $\mathcal{M}_c$ 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 Turing machine that always goes right.

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

(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
- C++
- Common Lisp
- Haskell
- Java
- JavaScript
- Object Pascal
- Perl
- Python
- Lisp
- Pascal

[3] There exist infinitely many such codings.

- R
- Smalltalk
- Prolog
- TEX, etc.

One step further: we go from looking at Turing machines as acceptors (which recognize a language, i.e. a set of input), to machines that compute functions. But since a Turing machine may never halt on a given input, the function it calculates is a partial functios; so, not necessarily defined on the whole domain.

Definition 3.2

Given any two non-empty finite sets A, B, a *partial* function $f : A^{<\omega} \rightharpoonup B^{<\omega}$ 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 $w \in dom(f)$: $\mathcal{M}_f(w) \downarrow^{acc.}$ with the word "$f(w)$" on its tape.

2.4 Universal Turing Machine

If we compare a Turing Machine with a computer, on one hand the Turing machine 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 Turing machine 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 universal Turing machines to obtain:

(1) a language that is *Turing recognizable but not decidable*[4],

(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 Turing machine is of the form:

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

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

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

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

So, that the description of a Turing machine is a finite word over some given finite alphabet. To be precise, the description of such a machine is a finite sequence M over the following finite alphabet:

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

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

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

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 [5]. 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:

[5] this comes from the fact the last letter of a word that defines a Turing machine 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.

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

Proof of Proposition 4.1:

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

(1) on 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 ②;

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

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

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

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

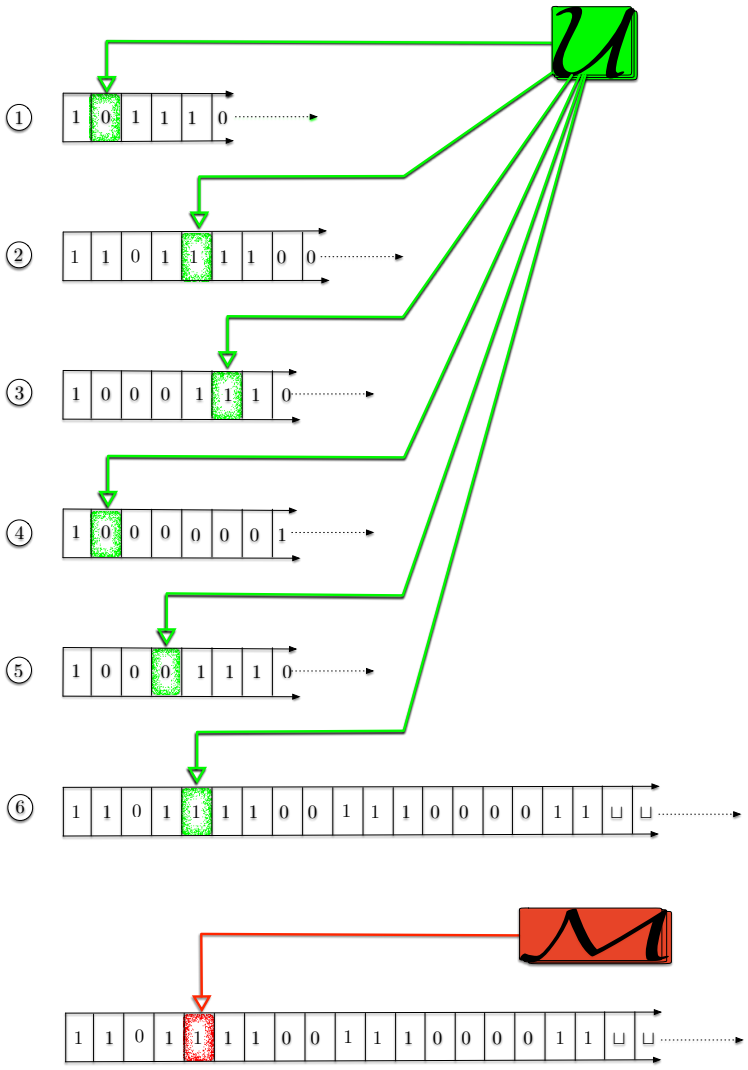
   (a) $\mathcal{U}$ reads a letter – say 0 – on ⑥, and

   (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 $\ulcorner q_1\urcorner$.



□

[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:

By making use of a universal Turing machine, we can easily show that this language is Turing recognizable.

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

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

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:

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

Proof of Proposition 6.1:

By making use of any universal Turing machine, it is easy to check that this language is Turing recognizable. Towards a contradiction we assume it also decidable, hence there exists a *Decider* $\mathcal{D}$ that decides it. From $\mathcal{D}$ we build another decider $\mathcal{D}'$ that decides

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

on input $\ulcorner \mathcal{M} \urcorner w$, $\mathcal{D}'$ runs as $\mathcal{D}$ until right before $\mathcal{D}$ halts.

- if $\mathcal{D}$ halts and rejects, then $\mathcal{D}'$ halts and rejects as well,
- if $\mathcal{D}$ halts and accepts, then $\mathcal{D}'$ runs any universal Turing machine $\mathcal{U}$ on $\ulcorner \mathcal{M} \urcorner w$ again, and then:
  - if $\mathcal{U}$ halts and rejects, $\mathcal{D}'$ halts and rejects, and
  - if $\mathcal{U}$ halts and accepts, $\mathcal{D}'$ halts and accepts.

□

Proposition 6.2

The following language is Turing recognizable but not decidable:

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

Proof of Proposition 6.2:

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

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

would be decidable as well, contradicting Proposition 6.1.
Indeed, towards a contradiction, we assume that there exists some decider $\mathcal{D}$ that decides

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

and build another decider $\mathcal{D}'$ that decides

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

On input $\ulcorner \mathcal{M} \urcorner w$, $\mathcal{D}'$ first computes the code $\ulcorner \mathcal{M}' \urcorner$ of a Turing machine $\mathcal{M}'$ which works as follows on the empty word:

(1) $\mathcal{M}'$ first writes the word w, then moves its head back to the leftmost cell and places itself in the initial state of the machine $\mathcal{D}$.

(2) Then, $\mathcal{M}'$ works exactly as $\mathcal{M}$.

□

Proposition 6.3

For any language L, we have the following equivalence:

$$\text{both } L \text{ and } L^{\complement} \text{ are recursively enumerable} \iff L \text{ is recursive.}$$

Proof of Proposition 6.3:

($\Leftarrow$) is immediate.

($\Rightarrow$) From $\mathcal{M}$ that recognizes L and $\mathcal{M}_{\complement}$ that recognizes $L^{\complement}$, we build a decider $\mathcal{D}$ as follows, on input w repeatedly for $i = 1, 2, 3, \ldots$ it recursively simulates first $\mathcal{M}$ on w for i many steps, and in case $\mathcal{M}$ has not stopped, simulates $\mathcal{M}'_{\complement}$ on w for i many steps, where $\mathcal{M}'_{\complement}$ is the same Turing machine as $\mathcal{M}_{\complement}$ except that $q_{acc.}$ and $q_{rej.}$ are swapped.

□

Corollary 6.1

The following languages are not recursively enumerable:

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

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

(3) $\{0,1\}^* \backslash \{ \ulcorner \mathcal{M} \urcorner \in \{0,1\}^* \mid \mathcal{M}(\varepsilon) \downarrow \}$

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

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

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

Proof of Corollary 6.1:

Immediate consequences of Propositions 5.1, 6.1, 6.2, and 6.3.

□

Proposition 6.4

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 6.4:

Left as an exercise. □

Rice's Theorem 6.1

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 the following language is not decidable.

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

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

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

The same holds when $\mathcal{C}$ is the whole class of Turing-recognizable languages, checking whether the language recognized by a Turing machine belongs to $\mathcal{C}$ in this case is trivial.

Proof of Rice's Theorem:

The assumption that $\mathcal{C}$ is any class of Turing-recognizable languages that is neither the whole class of Turing-recognizable languages nor the empty set yields that there exist one Turing machine

- $\mathcal{M}_{in}$ such that $\mathcal{L}(\mathcal{M}_{in}) \in \mathcal{C}$, and
- $\mathcal{M}_{out}$ such that $\mathcal{L}(\mathcal{M}_{out}) \notin \mathcal{C}$.

Towards a contradiction, we assume that there exists some decider $\mathcal{D}_{\mathcal{L}(\mathcal{M})\in\mathcal{C}}$ which decides membership in $\mathcal{C}$, namely:

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

By making use of $\mathcal{D}_{\mathcal{L}(\mathcal{M})\in\mathcal{C}}$ we build another decider $\mathcal{D}_{\mathcal{M}(\varepsilon)\downarrow}$ that decides the halting problem

on empty tape, namely:

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

For this, we distinguish between $\varnothing \in \mathcal{C}$ and $\varnothing \notin \mathcal{C}$.

(1) If $\varnothing \notin \mathcal{C}$, the decider $\mathcal{D}_{\mathcal{M}(\varepsilon)\downarrow}$ on input $\ulcorner\mathcal{M}\urcorner$ computes $\ulcorner\mathcal{M}'\urcorner$ the code of a machine $\mathcal{M}'$ which works the following way: on input w, $\mathcal{M}'$ stores the input w and runs as $\mathcal{M}$ on the empty tape the following way:

- if $\mathcal{M}(\varepsilon)\uparrow$, then $\mathcal{M}'$ will not stop, so that we have $\mathcal{M}'(w)\uparrow$
- if $\mathcal{M}(\varepsilon)\downarrow$, right before $\mathcal{M}$ reaches an halting configuration, $\mathcal{M}'$ erases the whole working tape, writes the word w back as input, and starts now working exactly as $\mathcal{M}_{in}$ on the input w.

We notice that if $\mathcal{M}(\varepsilon)\downarrow$, then $\mathcal{M}'(w)\overset{\text{acc.}}{\downarrow} \iff \mathcal{M}_{in}(w)\overset{\text{acc.}}{\downarrow}$.

So, we have

- if $\mathcal{M}(\varepsilon)\downarrow$, then $\mathcal{L}(\mathcal{M}') = \mathcal{L}(\mathcal{M}_{in}) \in \mathcal{C}$
- if $\mathcal{M}(\varepsilon)\uparrow$, then $\mathcal{L}(\mathcal{M}') = \varnothing \notin \mathcal{C}$.

Finally, after on the input $\ulcorner\mathcal{M}\urcorner$ the decider $\mathcal{D}_{\mathcal{M}(\varepsilon)\downarrow}$ has computed the code of $\ulcorner\mathcal{M}'\urcorner$, it just runs $\mathcal{D}_{\mathcal{L}(\mathcal{M})\in\mathcal{C}}$ on the input $\ulcorner\mathcal{M}'\urcorner$ to get the result it wants.

(2) If $\varnothing \in \mathcal{C}$, the construction is as above replacing $\mathcal{M}_{in}$ with $\mathcal{M}_{out}$. This time, on input $\mathcal{M}$:

- if $\mathcal{M}(\varepsilon)\downarrow$, then $\mathcal{L}(\mathcal{M}') = \mathcal{L}(\mathcal{M}_{out}) \notin \mathcal{C}$
- if $\mathcal{M}(\varepsilon)\uparrow$, then $\mathcal{L}(\mathcal{M}') = \varnothing \in \mathcal{C}$.

So that the decider $\mathcal{D}_{\mathcal{M}(\varepsilon)\downarrow}$ just runs $\mathcal{D}_{\mathcal{L}(\mathcal{M})\in\mathcal{C}}$ on the input $\ulcorner\mathcal{M}'\urcorner$, swapping the answers yes/no to get the result it wants.

Therefore, in both cases, the halting problem on empty string becomes decidable. A contradiction.

□

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!)

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

such that

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

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*" [52, pp. 227–233]. □

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 usual 2-tape Turing Machine) plus one infinite object so that this Turing machine can have access to an infinite amount of information – something a usual 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 Turing machine $\mathcal{O}$ whose initial configuration is

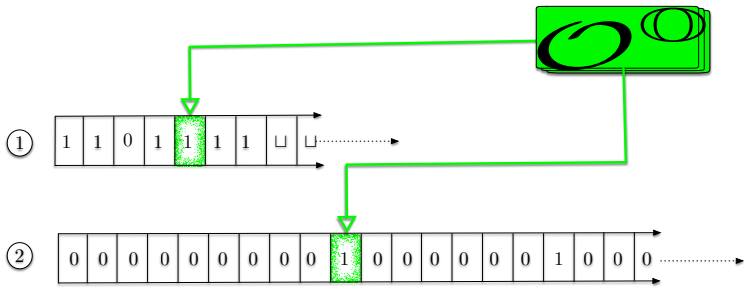
$$\begin{pmatrix} q_0 w & , & q_0 \chi_{\mathbb{O}} \\ ① & & ② \end{pmatrix}$$

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

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



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 integers 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 [6] ever sees its own code onto tape ② (although the code of $\mathcal{O}$ – or the code of an equivalent Turing machine – 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:

$$\mathbb{O} = \{\overline{1^{\ulcorner}\mathcal{M}^{\urcorner}}^{2} \in \mathbb{N} \mid \mathcal{M}(\varepsilon)\downarrow\}^{7}.$$

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 Turing machine $\mathcal{O}^{\mathbb{O}}$ checks whether w is the code of a Turing machine. If it is not the case it rejects right away; otherwise,

[6] we are talking about $\mathcal{O}^{\mathbb{O}}$ and not just $\mathcal{O}$!

(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 $f : \begin{array}{ccc} \{0,1\}^* & \xrightarrow{bij.} & \mathbb{N} \\ w & \longmapsto & \overline{1w}^2 - 1 \end{array}$ 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 $f^{-1}(k)$.

  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

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

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

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

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

So, we have

(1) a coding for the Turing machines:

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

(2) a coding for the words:

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

(3) a coding for the integers:

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

[7] $\overline{w}^2$ stands for the integer n that, once written in base 2, yields the word w

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 need is that $\ulcorner \mathcal{M} \urcorner$ is an integer that codes the Turing machine $\mathcal{M}$, and two different machines are coded with two different integer ($\mathcal{M} \neq \mathcal{M}' \Longrightarrow \ulcorner \mathcal{M} \urcorner \neq \ulcorner \mathcal{M}' \urcorner$). Also that the — coding and deciphering — operations $\ulcorner \mathcal{M} \urcorner \rightsquigarrow \mathcal{M}$ and $\ulcorner \mathcal{M} \urcorner \rightsquigarrow \ulcorner \mathcal{M} \urcorner$ can be performed by Turing machines.

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.

□

Definition 7.2: Turing Reducibility

Given any oracles $\mathbb{O}_A, \mathbb{O}_B \subseteq \mathbb{N}$,
we write

$$\mathbb{O}_A \leqslant_T \mathbb{O}_B$$

and say $\mathbb{O}_A$ is "Turing reducible" to $\mathbb{O}_B$, if there exists an o-c-TM $\mathcal{O}^{\mathbb{O}_B}$ which, starting on the empty tape, computes $\chi_{\mathbb{O}_A}$.

Proposition 7.2

Given any $\mathbb{O}_A, \mathbb{O}_B \subseteq \mathbb{N}$, the following are equivalent:

(1) $\mathbb{O}_A$ is Turing reducible to $\mathbb{O}_B$,

(2) for every o-c-TM $\mathcal{M}$, there exists an o-c-TM $\mathcal{N}$ such that $\mathcal{L}\left(\mathcal{M}^{\mathbb{O}_A}\right) = \mathcal{L}\left(\mathcal{N}^{\mathbb{O}_B}\right)$.
Moreover, in case $\mathcal{M}^{\mathbb{O}_A}$ is an oracle Decider, we can make sure that $\mathcal{N}^{\mathbb{O}_B}$ is an oracle Decider too.

Proof of Proposition 7.2:

Left as an easy exercise.

□

Notation 7.2

Given any $\mathbb{O}_A, \mathbb{O}_B \subseteq \mathbb{N}$, we write

- $\mathbb{O}_A \leqslant_T \mathbb{O}_B$ if $\mathbb{O}_A$ *is Turing reducible to* $\mathbb{O}_B$;
- $\mathbb{O}_A \equiv_T \mathbb{O}_B$ if $\mathbb{O}_A \leqslant_T \mathbb{O}_B$ and $\mathbb{O}_B \leqslant_T \mathbb{O}_A$;
- $\mathbb{O}_A <_T \mathbb{O}_B$ if $\mathbb{O}_A \leqslant_T \mathbb{O}_B$ but $\mathbb{O}_B \not\leqslant_T \mathbb{O}_A$.

Notice that $\equiv_T$ is an equivalence relation:

- $\mathbb{O}_A \equiv_T \mathbb{O}_A$ $\big($since $\mathbb{O}_A \leqslant_T \mathbb{O}_A\big)$;
- $\mathbb{O}_A \equiv_T \mathbb{O}_B \iff \mathbb{O}_B \equiv_T \mathbb{O}_A$;
- $\left.\begin{array}{c}\mathbb{O}_A \equiv_T \mathbb{O}_B \\ \textit{and} \\ \mathbb{O}_B \equiv_T \mathbb{O}_C\end{array}\right\} \Longrightarrow \mathbb{O}_A \equiv_T \mathbb{O}_C$ $\left(\text{since } \left.\begin{array}{c}\mathbb{O}_A \leqslant_T \mathbb{O}_B \\ \textit{and} \\ \mathbb{O}_B \leqslant_T \mathbb{O}_C\end{array}\right\} \Longrightarrow \mathbb{O}_A \leqslant_T \mathbb{O}_C\right)$.

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 $\mathbb{O}_L = \mathbb{O}_{\mathcal{L}_{(\mathbb{O}_L)}}$.

Definition 7.3: Turing Degree

Given any oracle $\mathbb{O}_A \subseteq \mathbb{N}$, the equivalence class

$$[\mathbb{O}_A]_{\equiv_T} = \{\mathbb{O}_B \subseteq \mathbb{N} \mid \mathbb{O}_B \equiv_T \mathbb{O}_A\}$$

is called a *Turing degree*.

The ordering $\leqslant_T$ on oracles induces an ordering $\leqslant$ on the set $\mathbb{TD}$ of all Turing degrees: given any Turing degrees $d, e \in \mathbb{TD}$,

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

or equivalently

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

As usual, we make use of the notation

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

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 $d \in \mathbb{TD}$

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

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

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

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 [28] p. 157 and also [49, 50] – 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) $\Big\{\mathcal{L}_{(\mathbb{O})} \subseteq \{0,1\}^* \mid \mathbb{O} \leqslant_T \varnothing\Big\} = \mathcal{R}ec.$

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

(the inclusion is strict since both $\mathbb{H}^{\complement}_{alt} \equiv_T \mathbb{H}_{alt}$ and $\mathcal{L}_{(\mathbb{H}^{\complement}_{alt})} \notin \mathcal{R}ec.$ hold)

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

Where

- $\mathcal{R}ec.$ is the class of all recursive (i.e. decidable) languages
- $\mathcal{R}.\mathcal{E}.$ is the class of all recursively enumerable (i.e. Turign recognizable) languages

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

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

Proof of Proposition 7.3:

Left as an exercise.

□

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' ("*the jump of* A") which satisfies $A <_T A'$.

Definition 7.4: Jump operator

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

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

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

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

(See page 84 for the definition of $\alpha_2 : \begin{array}{ccc}\mathbb{N}\times\mathbb{N} & \xrightarrow{bij.} & \mathbb{N} \\ (x,y) & \longmapsto & \frac{(x+y)\cdot(x+y+1)}{2}+y.\end{array}$)

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 $A' \not\leqslant_T A$.

($\mathbf{A \leqslant_T A'}$) 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

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

($\mathbf{A' \not\leqslant_T A}$) 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:

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

□

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

Figure 2.1: Diagonal argument: swap 0's and 1's on the diagonal.

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

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 build 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: for this we *swap 0's and 1's on the diagonal:* $0 \rightsquigarrow \mathbf{1}$ and $1 \rightsquigarrow \mathbf{0}$.

Corollary 7.1

The following strict ordering between jumps is satisfied:

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

where

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

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 $\alpha < \beta < \omega_1$.

Chapter 3

Recursive Functions

The whole chapter is highly inspired by René Cori and Daniel Lascar book's book: *"Mathematical Logic, Part 2, Recursion Theory, Gödel Theorems, Set Theory, Model Theory"* [12].

Recursive functions are functions from $\mathbb{N}^p$ to $\mathbb{N}$. We will show that they have a strong relation with the Turing computable ones.
We define the set of recursive functions by induction. For this purpose, for any integer p, we denote by $\mathbb{N}^{(\mathbb{N}^p)}$ the set of all mappings of the form $\mathbb{N}^p \longrightarrow \mathbb{N}$. Notice that $\mathbb{N}^p$ is a notation for the set of all mappings $\{i \in \mathbb{N} \mid i < p\} \longrightarrow \mathbb{N}$. When $p = 0$, the set $\{i \in \mathbb{N} \mid i < p\}$ becomes $\{i \in \mathbb{N} \mid i < 0\} = \varnothing$. Thus the set $\mathbb{N}^0$ only contains one element: the empty function whose graph is $\varnothing$. Therefore the set of all mappings of the form $\mathbb{N}^0 \longrightarrow \mathbb{N}$ contains all mappings that assign one integer to the empty function:

$$\mathbb{N}^0 \longrightarrow \mathbb{N} = \left\{ \begin{array}{lccc} f: & \{\varnothing\} & \longrightarrow & \mathbb{N} \\ & \varnothing & \longrightarrow & n \end{array} \middle| \, n \in \mathbb{N} \right\}.$$

So, as may be expected, mappings in $\mathbb{N}^{(\mathbb{N}^0)}$ are identified with elements of $\mathbb{N}$.

3.1 Primitive Recursive Functions

Definition 1.1

projection: If i is any integer such that $1 \leqslant i \leqslant p$ holds, the i^{th} *projection* π_i^p is the function of $\mathbb{N}^{(\mathbb{N}^p)}$ defined by

$$\pi_i^p(x_1, \ldots, x_p) = x_i.$$

successor: $S \in \mathbb{N}^{\mathbb{N}}$ is the *successor* function[a].

composition: Given $f_1, \dots, f_n \in \mathbb{N}^{(\mathbb{N}^p)}$ and $g \in \mathbb{N}^{(\mathbb{N}^n)}$, the *composition* $h = g(f_1, \dots, f_n) \in \mathbb{N}^{(\mathbb{N}^p)}$ is defined by

$$h(x_1, \dots, x_p) = g\big(f_1(x_1, \dots, x_p), \dots, f_n(x_1, \dots, x_p)\big)$$

We often make use of the notation $\overrightarrow{x}$ for $(x_1, \dots, x_p)$ so that for instance

$$g\big(f_1(\overrightarrow{x}), \dots, f_n(\overrightarrow{x})\big)$$

stands for

$$g\big(f_1(x_1, \dots, x_p), \dots, f_n(x_1, \dots, x_p)\big).$$

recursion: Given $g \in \mathbb{N}^{(\mathbb{N}^p)}$ and $h \in \mathbb{N}^{(\mathbb{N}^{p+2})}$, there exists a unique $f \in \mathbb{N}^{(\mathbb{N}^{p+1})}$ such that for all $\overrightarrow{x} \in \mathbb{N}^p$ and $y \in \mathbb{N}$ satisfies

(1) $f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$

(2) $f(\overrightarrow{x}, y+1) = h\big(\overrightarrow{x}, y, f(\overrightarrow{x}, y)\big)$

We say f is defined by *recursion* on both g (for the initial step) and h (for the successor steps).

[a] $S(n) = n + 1$.

Definition 1.2

The set of *primitive recursive* ($\mathcal{P}rim.\ \mathcal{R}ec.$) functions is the least that

(1) contains:

(a) All constants $\mathbb{N}^p \longrightarrow \mathbb{N}$ (all $\bar{i} \in \mathbb{N}^{(\mathbb{N}^p)}$ s.t. $\bar{i}(\overrightarrow{x}) = i$ – any $i, p \in \mathbb{N}$).

(b) All projections π_i^p (any $p \in \mathbb{N}$, any $1 \leqslant i \leqslant p$)

(c) The successor function $S \in \mathbb{N}^{\mathbb{N}}$.

(2) and is closed under

(a) composition

(b) recursion

We set up these functions in a hierarchy $(R_n)_{n\in\mathbb{N}}$:

(1) R_0 is the set of all functions in (1)(a), (1)(b) and (1)(c).

(2) R_{n+1} is the closure[a] of R_n under (2)(a) and (2)(b).

Clearly

$$\mathcal{P}rim.\ \mathcal{R}ec. = \bigcup_{n\in\mathbb{N}} R_n.$$

[a] $R_{n+1} = R_n \cup \{h \text{ obtained by composition on the basis of functions in } R_n\} \cup \{h \text{ obtained by induction on the basis of functions in } R_n\}$.

Example 1.1

(1) Addition: $(x, y) \longrightarrow x + y$
We have:

$$\begin{cases} x + 0 = x \\ x + (y + 1) = (x + y) + 1. \end{cases} \tag{3.1}$$

Formally:

$$\begin{cases} add(x, 0) = \pi_1^1(x) \\ add\ (x, y + 1) = S\Big(\pi_3^3\big(x, y, add(x, y)\big)\Big) \\ \qquad\qquad\quad = S \circ \pi_3^3\big(x, y, add(x, y)\big) \end{cases} \tag{3.2}$$

(2) Multiplication: $(x, y) \longrightarrow x \cdot y$
We have

$$\begin{cases} x \cdot 0 = 0 \\ x \cdot (y + 1) = x \cdot y + x. \end{cases} \tag{3.3}$$

Formally:

$$\begin{cases} mult(x, 0) = \overline{0}(x) \\ mult(x, y + 1) = add\Big(\pi_3^3\big(x, y, mult(x, y)\big), \pi_1^3\big(x, y, mult(x, y)\big)\Big). \end{cases} \tag{3.4}$$

(3) Exponentiation: $x \longrightarrow n^x$
We have

$$\begin{cases} n^0 = 1 \\ n^{x+1} = n^x \cdot n. \end{cases} \tag{3.5}$$

Formally:

$$\begin{cases} exp_n(0) = 1 \\ exp_n(x + 1) = mult\Big(\pi_2^2\big(x, exp_n(x)\big), \overline{n}\Big). \end{cases} \tag{3.6}$$

(4) Factorial: $x \longrightarrow x!$
We have

$$\begin{cases} 0! = 1 \\ (x + 1)! = x! \cdot (x + 1). \end{cases} \tag{3.7}$$

Formally:

$$\begin{cases} fact(0) = 1 \\ fact(x+1) = mult\Big(\pi_2^2\big(x, fact(x)\big), S\Big(\pi_1^2\big(x, fact(x)\big)\Big)\Big). \end{cases} \tag{3.8}$$

Example 1.2

We define $\dot{-} \in \mathbb{N}^{(\mathbb{N}^2)}$ by

$$\begin{cases} x \dot{-} y &= x - y \quad if \;\; x > y, \\ &= 0 \quad\;\; otherwise. \end{cases}$$

To show $\dot{-} \in \mathbb{N}^{(\mathbb{N}^2)}$ belongs to $\mathcal{P}rim.\ \mathcal{R}ec.$, we first show $x \longrightarrow x \dot{-} 1$ belongs to $\mathcal{P}rim.\ \mathcal{R}ec.$

$$\begin{cases} 0 \dot{-} 1 = 0 \\ (x+1) \dot{-} 1 = x \end{cases} \tag{3.9}$$

Formally:

$$\begin{cases} 0 \dot{-} 1 = \overline{0}(x) \\ (x+1) \dot{-} 1 = \pi_1^2\big(x, x \dot{-} 1\big) \end{cases} \tag{3.10}$$

$$\begin{cases} x \dot{-} 0 = x \\ x \dot{-} (y+1) = \big(x \dot{-} y\big) \dot{-} 1 \end{cases} \tag{3.11}$$

Formally:

$$\begin{cases} x \dot{-} 0 = \pi_1^1(x) \\ x \dot{-} (y+1) = \Big(\pi_3^3\big(x, y, x \dot{-} y\big)\Big) \dot{-} 1 \end{cases} \tag{3.12}$$

Definition 1.3

A set $A \subseteq \mathbb{N}^p$ is *primitive recursive* ($\mathcal{P}rim.\ \mathcal{R}ec.$) if its characteristic function ($\chi_A \in \mathbb{N}^{(\mathbb{N}^p)}$) is primitive recursive.

Example 1.3

(1) The set $\varnothing$ is $\mathcal{P}rim.\ \mathcal{R}ec.$ since $\chi_\varnothing = \overline{0}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$

(2) The set $\mathbb{N}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$ since $\chi_{\mathbb{N}} = \overline{1}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$

(3) The set $<_{\mathbb{N}} = \{(x, y) \mid x < y\}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$ $\chi_{<_{\mathbb{N}}}(x, y) = 1 \dot{-} (1 \dot{-} (y \dot{-} x))$.

On computable and partial functions

Definition 1.4

(1) (dom_f, f) is a *partial function* $\mathbb{N}^p \longrightarrow \mathbb{N}$ if f is a mapping $dom_f \longrightarrow \mathbb{N}$ where $dom_f \subseteq \mathbb{N}^p$.

(2) (dom_f, f) is a *total function* $\mathbb{N}^p \longrightarrow \mathbb{N}$ if $dom_f = \mathbb{N}^p$ holds.

We say that f is undefined on x – or $f(x)$ is undefined – if $x \notin dom_f$.

Notation 1.1

We write $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ for (dom_f, f) is a *partial function* $\mathbb{N}^p \longrightarrow \mathbb{N}$ whose domain is dom_f.

Notice that given any two partial functions $f, g \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$,

$$f = g \iff \begin{cases} dom_f = dom_g \\ \text{and} \\ \forall x \quad f(x) = g(x). \end{cases}$$

Definition 1.5

A partial function $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ is *"Turing Computable"* (TC) if there exists a Turing machine $\mathcal{M}$ such that on input $\overrightarrow{x} = (x_1, \ldots, x_p)$:

(1) if $f(\overrightarrow{x})$ is not defined, then $\mathcal{M}(\overrightarrow{x}) \uparrow$;

(2) if $(\overrightarrow{x}) \in dom_f$, $\mathcal{M}(\overrightarrow{x}) \downarrow$ with $f(\overrightarrow{x})$ written on its tape.

Proposition 1.1

Given any partial function $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$,

$$f \text{ is Turing Computable} \iff \mathcal{G}_f = \left\{ \left(\overrightarrow{x}, f(x) \right) \mid \overrightarrow{x} \in dom_f \right\} \text{ is Turing Recognizable.}$$

	$f \in \mathbb{N}^{(\mathbb{N}^p)}$	$f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$
$\mathcal{G}_f \subseteq \mathbb{N}^{p+1}$	*recursive* $\|\|$ *decidable*	*rec. enum.* $\|\|$ *Turing rec.*

Figure 3.1: Relations between Turing computable functions and their graphs

Proof of Proposition 1.1:

($\Rightarrow$) From $\mathcal{M}$ that computes f it is immediate to build $\mathcal{N}$ that recognizes $\mathcal{G}_f$. On input $(\overrightarrow{x}, y)$ it simulates $\mathcal{M}$ if $\mathcal{M}(\overrightarrow{x}, y) \overset{\text{acc.}}{\downarrow}$ it compares $f(x)$ with y and if $f(x) = y$ it accepts, otherwise it rejects.

($\Leftarrow$) From $\mathcal{N}$ that recognizes $\mathcal{G}_f$, we build $\mathcal{M}$ that computes f as follows, on input $\overrightarrow{x}$ repeatedly for $i = 1, 2, 3, \ldots$ it recursively simulates $\mathcal{N}$ on $(\overrightarrow{x}, 0), (\overrightarrow{x}, 1), (\overrightarrow{x}, 2), \ldots, (\overrightarrow{x}, i)$ for i many steps. If $\mathcal{N}$ accepts $(\overrightarrow{x}, n)$, then $\mathcal{M}$ prints out the value n.

□

Corollary 1.1

Given any function $f : \mathbb{N}^p \longrightarrow \mathbb{N}$,

$$f \textit{ is both total and Turing Computable} \implies \mathcal{G}_f \textit{ is recursive (decidable).}$$

Proof of Corollary 1.1:

Left as an immediate exercise.

□

We notice tha following:
All functions in R_0 are total and Turing computable. By induction on n, it is easy to show that all functions in R_n are also total and Turing computable. Therefore

- All $\mathcal{P}rim.\ \mathcal{R}ec.$ functions are total and Turing computable.

- All graphs of $\mathcal{P}rim.\ \mathcal{R}ec.$ functions are *recursive*

Even though the class of all $\mathcal{P}rim.\ \mathcal{R}ec.$ functions is included in the class of total and turing computable functions, the inverse inclusion does not hold[1].

3.2 Variable Substitution

Proposition 2.1: $\mathcal{P}rim.\ \mathcal{R}ec.$ closed under variable substitution

If $f \in \mathbb{N}^{(\mathbb{N}^p)}$, is $\mathcal{P}rim.\ \mathcal{R}ec.$, then given any $\sigma : \{1, \dots, p\} \longrightarrow \{1, \dots, p\}$, the function $g \in \mathbb{N}^{(\mathbb{N}^p)}$ defined by

$$g(x_1, \dots, x_p) = f(x_{\sigma(1)}, \dots, x_{\sigma(p)})$$

is also $\mathcal{P}rim.\ \mathcal{R}ec.$

Proof of Proposition 2.1:

We have

$$g(x_1, \dots, x_p) = f(x_{\sigma(1)}, \dots, x_{\sigma(p)}) = f\big(\pi^p_{\sigma(1)}(\overrightarrow{x}), \dots, \pi^p_{\sigma(p)}(\overrightarrow{x})\big).$$

□

Proposition 2.2

If $A \subseteq \mathbb{N}^n$ is $\mathcal{P}rim.\ \mathcal{R}ec.$ and $f_1, \dots, f_n \in \mathbb{N}^{(\mathbb{N}^p)}$ are $\mathcal{P}rim.\ \mathcal{R}ec.$ then

$$\{\overrightarrow{x} \in \mathbb{N}^p \mid (f_1(\overrightarrow{x}), \dots, f_n(\overrightarrow{x})) \in A\}$$

is $\mathcal{P}rim.\ \mathcal{R}ec.$

Proof of Proposition 2.2:

Set $B = \{\overrightarrow{x} \in \mathbb{N}^p \mid (f_1(\overrightarrow{x}), \dots, f_n(\overrightarrow{x})) \in A\}$. We have

$$\chi_B(\overrightarrow{x}) = \chi_A\Big(f_1(\overrightarrow{x}), \dots, f_n(\overrightarrow{x})\Big).$$

[1] see exercise on the Ackermann function $A \in \mathbb{N}^{(\mathbb{N}^2)}$ defined by

$$A(m,n) = \begin{cases} n+1 & \text{if} \quad m = 0, \\ A(m-1,1) & \text{if} \quad m > 0 \text{ and } n = 0, \\ A\big(m-1, A(m,n-1)\big) & \text{if} \quad m > 0 \text{ and } n > 0. \end{cases}$$

$A(m,n)$ is fast growing ; for instance $2 \cdot 10^{19728} < A(4,2) < 3 \cdot 10^{19728}$.

□

Example 2.1

If $f, g \in \mathbb{N}^{(\mathbb{N}^p)}$ are $\mathcal{P}rim.\ \mathcal{R}ec.$, then the following sets are also $\mathcal{P}rim.\ \mathcal{R}ec.$:

(1) $\{\overrightarrow{x} \mid f(\overrightarrow{x}) > g(\overrightarrow{x})\}$ (2) $\{\overrightarrow{x} \mid f(\overrightarrow{x}) = g(\overrightarrow{x})\}$ (3) $\{\overrightarrow{x} \mid f(\overrightarrow{x}) < g(\overrightarrow{x})\}$.

Proposition 2.3

If $A, B \subseteq \mathbb{N}^p$ are $\mathcal{P}rim.\ \mathcal{R}ec.$ then $A \cup B$, $A \cap B$, $A \setminus B$, $A \Delta B$ and $\mathbb{N}^p \setminus A$ are all $\mathcal{P}rim.\ \mathcal{R}ec.$

Proof of Proposition 2.3:

$$\begin{aligned}
\chi_{A\cup B} &= 1 \dot{-} \left(1 \dot{-} (\chi_A + \chi_B)\right) \\
\chi_{A\cap B} &= \chi_A \cdot \chi_B \\
\chi_{A\setminus B} &= \chi_A \cdot \left(1 \dot{-} \chi_B\right) \\
\chi_{A\Delta B} &= \left(1 \dot{-} \chi_A \cdot \chi_B\right) \cdot \left(1 \dot{-} \left(\left(1 \dot{-} \chi_A\right) \cdot \left(1 \dot{-} \chi_B\right)\right)\right) \\
\chi_{A^c} &= 1 \dot{-} \chi_A.
\end{aligned}$$

□

Proposition 2.4: Case study

If $f_1, \ldots, f_{n+1} \in \mathbb{N}^{(\mathbb{N}^p)}$ and $A_1, \ldots, A_n \in \mathbb{N}^p$ are all $\mathcal{P}rim.\ \mathcal{R}ec.$, then $g \in \mathbb{N}^{(\mathbb{N}^p)}$ defined by:

$$g(\overrightarrow{x}) = \begin{cases}
f_1(\overrightarrow{x}) & \text{if} \quad \overrightarrow{x} \in A_1 \\
f_2(\overrightarrow{x}) & \text{if} \quad \overrightarrow{x} \in A_2 \setminus A_1 \\
f_3(\overrightarrow{x}) & \text{if} \quad \overrightarrow{x} \in A_3 \setminus (A_1 \cup A_2) \\
\vdots & \vdots \qquad\qquad \vdots \\
f_i(\overrightarrow{x}) & \text{if} \quad \overrightarrow{x} \in A_i \setminus (A_1 \cup A_2 \cup \ldots \cup A_{i-1}) \\
\vdots & \vdots \qquad\qquad \vdots \\
f_{n+1}(\overrightarrow{x}) & \text{if} \quad \overrightarrow{x} \notin (A_1 \cup \ldots \cup A_n)
\end{cases}$$

is also $\mathcal{P}rim.\ \mathcal{R}ec.$

Proof of Proposition 2.4:

$g = f_1 \cdot \chi_{A_1} + f_2 \cdot \chi_{(A_2 \setminus A_1)} + \ldots + f_{n+1} \cdot \chi_{\left(A_1 \cup A_2 \cup \ldots \cup A_n\right)^{\complement}}$. □

Corollary 2.1

$\sup(x_1, \ldots, x_n)$ and $\inf(x_1, \ldots, x_n)$ are $\mathcal{P}rim.\ \mathcal{R}ec.$

Proof of Corollary 2.1:

Left as an exercise. □

Proposition 2.5

$f \in \mathbb{N}^{(\mathbb{N}^{p+1})}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$, then $g, h \in \mathbb{N}^{(\mathbb{N}^{p})}$ below are $\mathcal{P}rim.\ \mathcal{R}ec.$:

$$g(x_1, \ldots, x_p, y) = \sum_{t=0}^{t=y} f(x_1, \ldots, x_p, t),$$

$$h(x_1, \ldots, x_p, y) = \prod_{t=0}^{t=y} f(x_1, \ldots, x_p, t).$$

Proof of Proposition 2.5:

Left as an exercise (both are easily defined by recursion).

□

3.3 Bounded Minimisation and Bounded Quantification

Proposition 3.1: Bounded minimisation

If $A \subseteq \mathbb{N}^{p+1}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$, then $f \in \mathbb{N}^{(\mathbb{N}^{p+1})}$ defined below is also $\mathcal{P}rim.\ \mathcal{R}ec.$:

$$f(\overrightarrow{x}, z) = \begin{cases} 0 \text{ if } \forall t \leqslant z \quad (\overrightarrow{x}, t) \notin A, \\ \text{the least } t \leqslant z \text{ such that } (\overrightarrow{x}, t) \in A \text{ otherwise.} \end{cases}$$

$f(\overrightarrow{x}, z)$ is denoted by $\mu t \leqslant z\ (\overrightarrow{x}, t) \in A$.

Proof of Proposition 3.1:

f is defined by:

$$\begin{cases} f(\overrightarrow{x}, 0) &= 0 \\ f(\overrightarrow{x}, z+1) &= \begin{cases} f(\overrightarrow{x}, z) & \text{if } \sum_{y=0}^{y=z} \chi_A(\overrightarrow{x}, y) \geqslant 1 \\ z+1 & \text{if } \sum_{y=0}^{y=z} \chi_A(\overrightarrow{x}, y) = 0 \text{ and } (\overrightarrow{x}, z+1) \in A \\ 0 & \text{if } \sum_{y=0}^{y=z+1} \chi_A(\overrightarrow{x}, y) = 0. \end{cases} \end{cases}$$

□

Proposition 3.2: *Prim. Rec.* closed under bounded quantification

The set of all *Prim. Rec.* predicates is closed under bounded quantification: i.e., If $A \subseteq \mathbb{N}^{p+1}$ is *Prim. Rec.*, then

- $\{(\overrightarrow{x}, z) : \exists t \leqslant z\ (\overrightarrow{x}, t) \in A\}$
- $\{(\overrightarrow{x}, z) : \forall t \leqslant z\ (\overrightarrow{x}, t) \in A\}$

are both *Prim. Rec.*

Proof of Proposition 3.2:

Set

- $B = \{(\overrightarrow{x}, z) : \exists t \leqslant z\ (\overrightarrow{x}, t) \in A\}$,
- $C = \{(\overrightarrow{x}, z) : \forall t \leqslant z\ (\overrightarrow{x}, t) \in A\}$.

We have

- $\chi_B(\overrightarrow{x}, z) = 1 \dot{-} (1 \dot{-} \sum_{t=0}^{t=z} \chi_A(\overrightarrow{x}, t))$,
- $\chi_C(\overrightarrow{x}, z) = \prod_{t=0}^{t=z} \chi_A(\overrightarrow{x}, t))$.

□

Example 3.1

- $\{2n : n \in \mathbb{N}\}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$ It is defined by recursion

$$\begin{cases} \chi_{(0)} & = & 1 \\ \chi_{(n+1)} & = & 1 \dot{-} \chi_{(n)} \end{cases}$$

- The mapping $\in \mathbb{N}^{(\mathbb{N}^2)}$ $(x,y) \longrightarrow \left[\dfrac{x}{y}\right]$ defined below is $\mathcal{P}rim.\ \mathcal{R}ec.$:

$$\begin{cases} \left[\dfrac{x}{y}\right] & = & 0 \ \textit{if}\ y = 0 \\ & = & \textit{integer part of}\ \dfrac{x}{y}\ \textit{otherwise.} \end{cases}$$

Formally

$$\begin{cases} \left[\dfrac{x}{y}\right] & = & 0 \ \textit{if}\ y = 0 \\ & = & \mu t \leqslant x \quad y \cdot (t+1) > x \ \textit{otherwise.} \end{cases}$$

- $\left\{(x,y) \mid y \ \textit{divides}\ x\right\} \in \mathcal{P}rim.\ \mathcal{R}ec.$:

$$\chi(x,y) = 1 \dot{-} \left(x \dot{-} \left(y \cdot \left[\frac{x}{y}\right]\right)\right).$$

- $\mathit{Prime} = \{x \in \mathbb{N} \mid x\ \textit{is a prime number}\} \in \mathcal{P}rim.\ \mathcal{R}ec.$:

$$x \in \mathit{Prime} \iff \begin{cases} x > 1 \\ \\ \textit{and} \\ \\ \forall y \leqslant x \begin{cases} y = 1 \\ \quad \textit{or} \\ y = x \\ \quad \textit{or} \\ y\ \textit{does not divide}\ x. \end{cases} \end{cases}$$

- $\Pi : \mathbb{N} \longrightarrow \mathbb{N}$ *defined by* $\Pi(n) = n + 1^{th}$ *prime number* $\in \mathcal{P}rim.\ \mathcal{R}ec.$.

$$\begin{cases} \Pi(0) & = & 2 \\ \Pi(n+1) & = & \mu z \leqslant (\Pi(n)! + 1) \quad z > \Pi(n)\ \textit{and}\ z \in\ \mathit{Prime}. \end{cases}$$

3.4 Coding Sequences of Integers

We define $\mathcal{P}rim.\ \mathcal{R}ec.$ functions that allow to treat finite sequences of integers as integers. Every sequence $\langle x_1, \ldots, x_p \rangle$ will be *"coded"* by a single integer $\alpha_p(x_1, \ldots, x_p)$. And from this single integer $\alpha_p(x_1, \ldots, x_p)$ one will be able to recover the elements of the original sequence by having $\mathcal{P}rim.\ \mathcal{R}ec.$ functions β_p^i that satisfy

$$\beta_p^i\Big(\alpha_p(x_1, \ldots, x_p)\Big) = x_i.$$

Proposition 4.1

For every non-zero $p \in \mathbb{N}$ there exists $\mathcal{P}rim.\ \mathcal{R}ec.$ functions $\beta_p^1, \beta_p^2, \ldots, \beta_p^p \in \mathbb{N}^{\mathbb{N}}$ and $\alpha_p \in \mathbb{N}^{(\mathbb{N}^p)}$ such that

$$\begin{cases} \alpha_p : \mathbb{N}^p \xrightarrow{bij.} \mathbb{N} \\ \\ \quad and \\ \\ \alpha_p^{-1}(x) = \big(\beta_p^1(x), \ldots, \beta_p^p(x)\big). \end{cases}$$

Proof of Proposition 4.1:

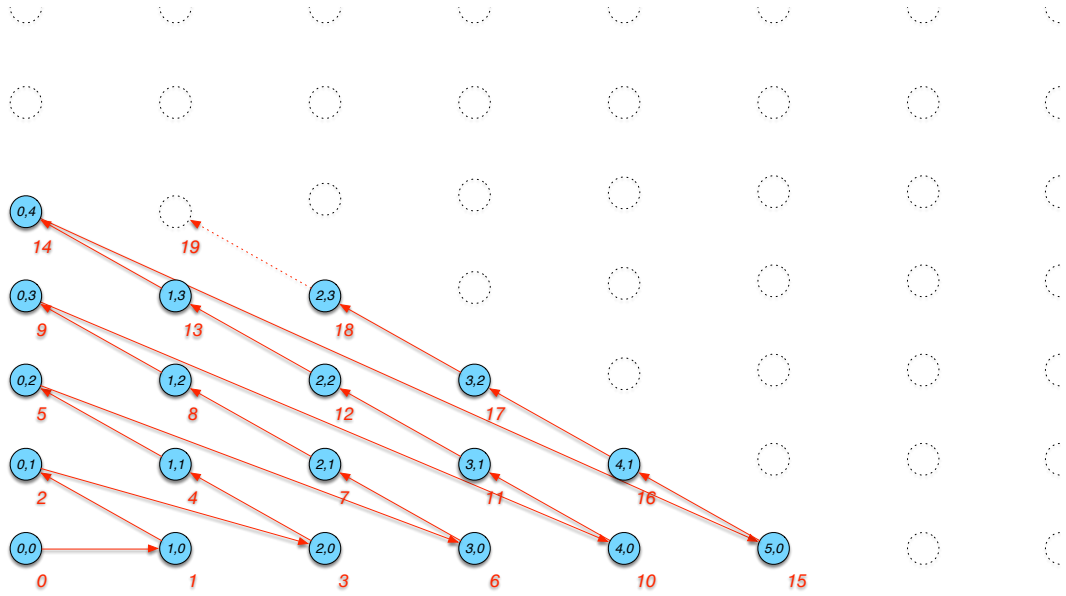
We start by defining $\alpha_1 = \beta_1^1 = id$. Then we move on to

$$\alpha_2(x, y) = \frac{(x+y)\cdot(x+y+1)}{2} + y.$$

This is obtained by looking at the following picture and noticing that

(1) $\alpha_2(x, y) = \alpha_2(x+y, 0) + y$, and

(2) $$\begin{aligned} \alpha_2(x+y, 0) &= 1 + 2 + \cdots + (x+y) \\ &= \tfrac{1}{2}\begin{pmatrix} 1 & & 2 & & & & x+y \\ + & + & + & + & \cdots & + & + \\ x+y & & x+y-1 & & & & 1 \end{pmatrix}. \end{aligned}$$



We have

(1) $\beta_2^1(n) = \mu x \leqslant n \quad \exists t \leqslant n \;\; \alpha_2(x,t) = n$

(2) $\beta_2^2(n) = \mu y \leqslant n \quad \exists t \leqslant n \;\; \alpha_2(t,y) = n.$

Then we define α_{p+1}, β_{p+1}^1, $\beta_{p+1}^2, \ldots\ldots$, β_{p+1}^{p-1}, β_{p+1}^p and β_{p+1}^{p+1} by induction on $p \in \mathbb{N}$:

- $\alpha_{p+1}(x_1, \ldots, x_p, x_{p+1}) = \alpha_p\big(x_1, \ldots, x_{p-1}, \alpha_2(x_p, x_{p+1})\big)$
- $\beta_{p+1}^1 = \beta_p^1$;
- $\beta_{p+1}^2 = \beta_p^2$;

$\vdots$

- $\beta_{p+1}^{p-1} = \beta_p^{p-1}$;
- $\beta_{p+1}^p = \beta_2^1 \circ \beta_p^p$;
- $\beta_{p+1}^{p+1} = \beta_2^2 \circ \beta_p^p$.

□

Example 4.1

A different way of coding sequences of integers:

$$\begin{cases} c(\varepsilon) & = & 1 \\ c(x_0,\ldots,x_p) & = & \Pi\,(0)^{x_0+1}\cdot\Pi\,(1)^{x_1+1}\cdots\Pi\,(p)^{x_p+1}. \end{cases}$$

From $n \in \mathbb{N}\setminus\{0\}$ we recover the sequence $\langle x_0,\ldots,x_p\rangle$ such that $c(x_0,\ldots,x_p)=n$ by considering the $\mathcal{P}rim.\ \mathcal{R}ec.$ function $d \in \mathbb{N}^{(\mathbb{N}^2)}$ which yields the exponents of the prime numbers:

$$d(i,n) = \mu x \leqslant n \quad \Pi\,(i)^{x+1} \textit{ does not divide } n.$$

3.5 Partial Recursive Functions

We recall that

(1) (dom_f, f) is a *partial function* $\mathbb{N}^p \longrightarrow \mathbb{N}$ if f is a mapping $dom_f \longrightarrow \mathbb{N}$ where $dom_f \subseteq \mathbb{N}^p$.

(2) (dom_f, f) is a *total function* $\mathbb{N}^p \longrightarrow \mathbb{N}$ if $dom_f = \mathbb{N}^p$ holds.

We say that f is undefined on x – or $f(x)$ is undefined – if $x \notin dom_f$. We use the notation $f \in \mathbb{N}^{(dom\subseteq\mathbb{N}^p)}$ to signify that (dom_f, f) is a *partial function* $\mathbb{N}^p \longrightarrow \mathbb{N}$ whose domain is dom_f. Notice that for any two partial functions $f, g \in \mathbb{N}^{(dom\subseteq\mathbb{N}^p)}$:

$$f = g \textit{ holds} \iff \begin{cases} dom_f & = & dom_g \\ & \textit{and} & \\ \forall x & f(x) = g(x). \end{cases}$$

Definition 5.1: Composition

Given $f_1,\ldots,f_n \in \mathbb{N}^{(dom\subseteq\mathbb{N}^p)}$ and $g \in \mathbb{N}^{(dom\subseteq\mathbb{N}^n)}$, the *composition* $h = g(f_1,\ldots,f_n) \in \mathbb{N}^{(\mathbb{N}^p)}$ is

$$\begin{cases} h(\overrightarrow{x}) \textit{ is undefined iff} \begin{cases} \overrightarrow{x} \notin \bigcap\limits_{1\leqslant i\leqslant n} dom_{f_i} \\ \textit{or otherwise} \\ \big(f_1(\overrightarrow{x}),\ldots,f_n(\overrightarrow{x})\big) \notin dom_g. \end{cases} \\ h(\overrightarrow{x}) \textit{ is defined otherwise and } h(\overrightarrow{x}) = g\big(f_1(\overrightarrow{x}),\ldots,f_n(\overrightarrow{x})\big). \end{cases}$$

$$= g\big(f_1(x_1,\ldots,x_p),\ldots,f_n(x_1,\ldots,x_p)\big)$$

Definition 5.2: Recursion

Given $g \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ and $h \in \mathbb{N}^{(dom \subseteq \mathbb{N}^{p+2})}$, there exists a unique $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^{p+1})}$ such that for all $\overrightarrow{x} \in \mathbb{N}^p$ and $y \in \mathbb{N}$:

(1)
$$\begin{cases} f(\overrightarrow{x}, 0) \text{ is undefined if } \overrightarrow{x} \notin dom_g \\ \text{and} \\ f(\overrightarrow{x}, 0) \text{ is defined otherwise with } f(\overrightarrow{x}, 0) = g(\overrightarrow{x}). \end{cases}$$

(2)
$$\begin{cases} f(\overrightarrow{x}, y+1) \text{ is undefined if } \begin{cases} (\overrightarrow{x}, y) \notin dom_f \\ \text{or} \\ \big(\overrightarrow{x}, y, f(\overrightarrow{x}, y)\big) \notin dom_h. \end{cases} \\ \text{and} \\ \text{otherwise } f(\overrightarrow{x}, y+1) \text{ is defined and } f(\overrightarrow{x}, y+1) = h\big(\overrightarrow{x}, y, f(\overrightarrow{x}, y)\big). \end{cases}$$

Definition 5.3: Minimization

Given $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^{p+1})}$, we define $g \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ by:

$$g(\overrightarrow{x}) = \mu y \quad f(\overrightarrow{x}, y) = 0.$$

Notice that

$$g(\overrightarrow{x}) = y \iff \begin{cases} \forall z < y \begin{cases} f(\overrightarrow{x}, z) \text{ is defined!} \\ \text{and} \\ f(\overrightarrow{x}, z) > 0 \end{cases} \\ \text{and} \\ f(\overrightarrow{x}, y) = 0. \end{cases}$$

Definition 5.4: Partial recursive functions

The set of *partial recursive* ($\mathcal{P}art.\ \mathcal{R}ec.$) functions is the least that

(1) contains:

(a) All constants $\mathbb{N}^p \longrightarrow \mathbb{N}$ (all $\bar{i} \in \mathbb{N}^{(\mathbb{N}^p)}$ s.t. $\bar{i}(\vec{x}) = i$ – any $i, p \in \mathbb{N}$).

(b) All projections π_i^p (any $p \in \mathbb{N}$, any $1 \leqslant i \leqslant p$)

(c) The successor function $S \in \mathbb{N}^{\mathbb{N}}$.

(2) and is closed under

(a) composition

(b) recursion

(c) minimisation.

Our next goal is to show that a function f is in $\mathcal{P}art.\ \mathcal{R}ec.$ if and only if it is Turing computable. One direction is easy, the other one is more involved. One side effect of our proof will show that every partial recursive function can be obtained by applying the minimisation at most once.

Lemma 5.1

Every partial recursive function is Turing computable.

Proof of Lemma 5.1:

We need to show that given any $p \in \mathbb{N} \setminus \{0\}$ and any $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ there exists some Turing machine $\mathcal{M}$ that computes f. This means that on input $(n_1, \ldots, n_p)$ it stops in configuration $q_{acc.} f(n_1, \ldots, n_p)$ if $(n_1, \ldots, n_p) \in dom_f$, and it never stops otherwise. Of course, we need to fix a certain representation of both integers and finite sequence of integers. For simplicity, let us say that the integers are represented in base-ten and the sequences as $(n_1, \ldots, n_p)$ so that the input alphabet is

$$\Sigma = \Big\{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ (,\ ),\ ,\Big\}.$$

So, for instance a Turing machine computes *Add* if on input word "$(385, 218)$" it returns the word "603".
We do the proof by induction on the number of operation among

- composition
- recursion
- minimisation

that are necessary to obtain $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ on the basis of

- all constants
- all projections
- the successor function.

(1) It is quite obvious that

- if $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ is constant, then there exists some basic Turing machine that computes it.
- Every projection π_i^p (any $1 \leqslant i \leqslant p$) is also trivially computable.
- The successor function is clearly computable as well.

(2) (a) Assume $f_1, \ldots, f_n \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ are computed respectively by $\mathcal{M}_{f_1}, \ldots, \mathcal{M}_{f_n}$ and $g \in \mathbb{N}^{(dom \subseteq \mathbb{N}^n)}$ is computed by $\mathcal{M}_g$. Then $f = g(f_1, \ldots, f_n) \in \mathbb{N}^{(\mathbb{N}^p)}$ is computed by $\mathcal{M}_f$ which works as follows:
on input $\overrightarrow{x} = (n_1, \ldots, n_p)$:

successively for each i $:= \mathbf{1}, \ldots, \mathbf{n}$ the machine $\mathcal{M}_f$ simulates $\mathcal{M}_{f_i}$ on input $\overrightarrow{x}$, if $\mathcal{M}_{f_i}(\overrightarrow{x}) \overset{\text{cc}}{\downarrow}$ with some output m_i it stores m_i.

(In case all simulations of machines $\mathcal{M}_{f_1}, \ldots, \mathcal{M}_{f_n}$ do stop) $\mathcal{M}_f$ finally simulates $\mathcal{M}_g$ on input $(m_1, \ldots, m_n)$.

It is clear that if either

$$\text{(A)}\ \ \overrightarrow{x} \notin \bigcap_{1 \leqslant i \leqslant n} dom_{f_i} \qquad \text{or} \qquad \text{(B)}\ \ \Big(f_1(\overrightarrow{x}), \ldots, f_n(\overrightarrow{x})\Big) \notin dom_g,$$

then $\mathcal{M}_f(\overrightarrow{x}) \uparrow$. In the opposite case, $\mathcal{M}_f(\overrightarrow{x}) \overset{\text{cc}}{\downarrow}$ with the right answer.

(b) Assume $g \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ and $h \in \mathbb{N}^{(dom \subseteq \mathbb{N}^{p+2})}$ are computed respectively by $\mathcal{M}_g$ and $\mathcal{M}_h$. Then f defined by recursion:

(A) $f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$

(B) $f(\overrightarrow{x}, y+1) = h\big(\overrightarrow{x}, y, f(\overrightarrow{x}, y)\big)$

is computed by $\mathcal{M}_f$ which works as follows:

- on input $(\overrightarrow{x}, 0)$ it simply simulates $\mathcal{M}_g$ on input $\overrightarrow{x}$, and
- on input $(\overrightarrow{x}, n+1)$ $\mathcal{M}_f$ first simulates $\mathcal{M}_g$ on input $\overrightarrow{x}$ which gives[a] $f(\overrightarrow{x}, 0)$. Then

  recursively for i $:= \mathbf{0}, \ldots, \mathbf{n}$ $\mathcal{M}_f$ simulates $\mathcal{M}_h$ on $\big(\overrightarrow{x}, i, f(\overrightarrow{x}, i)\big)$ which yields[b] $f(\overrightarrow{x}, i+1)$.

It is clear that $\mathcal{M}_f$ brings the result if and only if every step $f(\overrightarrow{x}, i)$ $(i := 0, \ldots, n+1)$ is defined. Otherwise it simply never stops.

(c) Assume $g \in \mathbb{N}^{(dom \subseteq \mathbb{N}^{p+1})}$ is computed by $\mathcal{M}_g$. We design $\mathcal{M}_f$ that on input $\overrightarrow{x}$ computes

$$\mu y \quad g(\overrightarrow{x}, y) = 0.$$

set i := 0 $\mathcal{M}_f$ simulates $\mathcal{M}_g$ on input $(\overrightarrow{x}, i)$. If $\mathcal{M}_g$ stops and outputs the value of $g(\overrightarrow{x}, i)$,
- if $g(\overrightarrow{x}, i) = 0$ $\mathcal{M}_f$ stops and outputs i,
- if $g(\overrightarrow{x}, i) \neq 0$, $\mathcal{M}_f$ starts over again with $i := i + 1$.

Notice that $\mathcal{M}_f$ stops and outputs $n = \mu y \quad g(\overrightarrow{x}, y) = 0$ if and only if
- $\forall i \leqslant n \quad (\overrightarrow{x}, i) \in dom_g$,
- $\forall i < n \quad g(\overrightarrow{x}, i) > 0$ and
- $g(\overrightarrow{x}, n) = 0$.

□

[a] in case $\mathcal{M}_g(\overrightarrow{x}) \overset{\text{acc.}}{\downarrow}$.

[b] in case $\mathcal{M}_h(\overrightarrow{x}, i, f(\overrightarrow{x}, i)) \overset{\text{acc.}}{\downarrow}$.

Lemma 5.2

Every Turing computable partial function $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^p)}$ is $\mathcal{P}art.\ \mathcal{R}ec.$

Proof of Lemma 5.2:

We show an even stronger result: given any Turing Machine $\mathcal{M}$ and any recursive coding of words on the tape alphabet Γ we show that the partial function $\Sigma^* \longrightarrow \Gamma^*$ which maps $v \in \Sigma^*$ to $w \in \Gamma^*$ if and only if the Turing machine from the initial configuration $q_0 v$ stops in some configuration $w_0 q_{acc.} w_1$ with $w_0 w_1 = w$ is partial recursive in the code. This means the function $f' \in \mathbb{N}^{(dom \subseteq \mathbb{N}^1)}$ defined by

$$\begin{cases} f'\big(code(v)\big) \text{ is undefined if } \mathcal{M}(v) \uparrow \text{ or } \mathcal{M}(v) \overset{\text{rej.}}{\downarrow}; \\ f'\big(code(v)\big) = code(w_0 w_1) \text{ if } \mathcal{M} \overset{\text{acc.}}{\downarrow} \text{ in config. } w_0 q_{acc.} w_1. \end{cases}$$

We first choose a coding of the configurations of $\mathcal{M}$: We assume
- $\Sigma = \{1, \ldots, k-1\}$ and $\Gamma = \{0, \ldots, k-1\}$ with $k > 1$ and necessarily $0 = \sqcup$.
- $Q = \{q_0, \ldots, q_m\}$ with q_0, q_1, q_2 being respectively the initial state, the rejecting state and the accepting state.

The coding of a word $w = a_0 \ldots a_n$ that we choose is

$$\ulcorner a_0 \ldots a_n \urcorner = \sum_{0 \leqslant i \leqslant n} a_i \cdot k^i.$$

This coding is not injective (this will not matter for our purpose) for any two word which differ by the tailing blanks in their prefix will have the same encoding:

$$\ulcorner a_0 \dots a_n \urcorner = \ulcorner a_0 \dots a_n \sqcup \urcorner.$$

But on the other hand, this encoding is surjective.
A configuration $w_0 q_r w_1$ of the Turing machine will be coded by

$$\ulcorner w_0 q_r w_1 \urcorner = \alpha_4(r, \ulcorner w_0 \urcorner, \ulcorner w_1 \urcorner, |w_0|).$$

For instance, the initial configuration of the Turing machine with input w is:

$$\begin{aligned}
\ulcorner q_0 w \urcorner &= \alpha_4(0, 0, \ulcorner w \urcorner, 0) \\
&= \alpha_2\Big(0, \alpha_2\big(0, \alpha_2(\ulcorner w \urcorner, 0)\big)\Big) \\
&= \alpha_2\Big(0, \alpha_2\big(0, \tfrac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}\big)\Big) \\
&= \alpha_2\Big(0, \tfrac{\left(\frac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}\right)\left(\frac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}+1\right)}{2}\Big) \\
&= \frac{\left(\frac{\left(\frac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}\right)\left(\frac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}+1\right)}{2}\right).\left(\frac{\left(\frac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}\right)\left(\frac{\ulcorner w \urcorner(\ulcorner w \urcorner+1)}{2}+1\right)}{2}+1\right)}{2}.
\end{aligned}$$

To say that w is an input word is to say that $w \in \Sigma^*$ (we will identify words of the form $w \in \Sigma^*$ with words of the form $w \sqcup \dots \sqcup$ since our coding will not be able to distinguish them[a] and also because the input word really is the infinite word $w \sqcup \dots \sqcup \dots$).
So we see that a word $w = a_0 \dots a_n \in \Gamma^*$ is an input word if for no $i < n$ we have both $a_i = \sqcup$ and $a_{i+1} \neq \sqcup$. This means that $\ulcorner a_0 \dots a_n \urcorner = a_0 \cdot k^0 + a_1 \cdot k^1 + \dots + a_n \cdot k^n$ satisfies $a_i = 0 \Rightarrow a_{i+1} = 0$ (any $i < n$). With our coding, we recover the coefficient a_i as

$$a_i = \left[\frac{\ulcorner a_0 \dots a_n \urcorner}{k^i}\right] \dot{-} \left(\left[\frac{\ulcorner a_0 \dots a_n \urcorner}{k^{i+1}}\right] \cdot k\right).$$

Therefore the set $Input_{\mathcal{M}}$ of all the codes of input words of $\mathcal{M}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$:

$$\begin{aligned}
\chi_{Input_{\mathcal{M}}}(m) &= 1 \quad \textit{if} \quad \forall i \leqslant m \left(\left[\frac{m}{k^i}\right] \dot{-} \left(\left[\frac{m}{k^{i+1}}\right] \cdot k\right) = 0 \Longrightarrow \left[\frac{m}{k^{i+1}}\right] \dot{-} \left(\left[\frac{m}{k^{i+2}}\right] \cdot k\right) = 0\right) \\
&= 0 \quad \textit{otherwise}
\end{aligned}$$

Since the coding of words that we choose is surjective, it comes that a configuration C is an initial configuration if and only if there exists $m \in Input_{\mathcal{M}}$

$$\ulcorner C \urcorner = \frac{\left(\frac{\left(\frac{m(m+1)}{2}\right)\left(\frac{m(m+1)}{2}+1\right)}{2}\right) \cdot \left(\frac{\left(\frac{m(m+1)}{2}\right)\left(\frac{m(m+1)}{2}+1\right)}{2}+1\right)}{2}.$$

Therefore the set $Init_{\mathcal{M}}$ of all the codes of initial configurations of $\mathcal{M}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$

$$\chi_{Init_{\mathcal{M}}}(c) = \begin{cases} 1 & if \quad \exists m \leqslant c \quad \big(\chi_{Input_{\mathcal{M}}}(m) = 1 \ \wedge \ \alpha_4(0,0,m,0) = c\big), \\ 0 & otherwise. \end{cases}$$

We now describe the transition from a given configuration C to the next configuration C' ($C \Rightarrow C'$), in other words, we analyse the we obtain the code of C' on the basis of both the code of C and the transition function δ.
A transition yields a move of the head either to the right or to the left: $\delta(q_r, a_l) = (q_{r'}, a_{l'}, R)$ or $\delta(q_r, a_l) = (q_{r'}, a_{l'}, L)$. We need to consider differently these two forms, together with differentiating also whether the head can or cannot move to the left when the transition function says so[b].

When $\delta(\mathbf{q_r}, \mathbf{a_l}) = (\mathbf{q_{r'}}, \mathbf{a_{l'}}, \mathbf{R})$, we have $C \Rightarrow C'$ is $w_0 q_r w_1 \Rightarrow w_0' q_{r'} w_1'$ with

(1) $w_0' = w_0 a_{l'}$ (2) $|w_0'| = 1 + |w_0|$ (3) $a_l w_1' = w_1$.

so that

(1) $\ulcorner w_0' \urcorner = \ulcorner w_0 \urcorner + l' \cdot k^{|w_0|} = \beta_4^2(\ulcorner C \urcorner) + l' \cdot k^{\beta_4^4(\ulcorner C \urcorner)}$

(2) $|w_0'| = |w_0| + 1 = \beta_4^4(\ulcorner C \urcorner) + 1$

(3) $a_l w_1' = w_1$ so that $\ulcorner w_1' \urcorner = \left[\dfrac{\beta_4^3(\ulcorner C \urcorner)}{k}\right]$.

So, we obtain:

$$if \begin{cases} \beta_4^1(\ulcorner C \urcorner) = r \\ and \\ \beta_4^3(\ulcorner C \urcorner) \dot{-} \left(k \cdot \left[\dfrac{\beta_4^3(\ulcorner C \urcorner)}{k}\right]\right) = l, \end{cases}$$

then

$$\ulcorner C' \urcorner = \alpha_4\left(r', \ \beta_4^2(\ulcorner C \urcorner) + l' \cdot k^{\beta_4^4(\ulcorner C \urcorner)}, \ \left[\frac{\beta_4^3(\ulcorner C \urcorner)}{k}\right], \ \beta_4^4(\ulcorner C \urcorner) + 1\right).$$

When $\delta(\mathbf{q_r}, \mathbf{a_l}) = (\mathbf{q_{r'}}, \mathbf{a_{l'}}, \mathbf{L})$, there are two different cases depending on whether the head can move to the left or not.

if $\mathbf{w_0} = \varepsilon$, then we have $C \Rightarrow C'$ is $w_0 q_r w_1 \Rightarrow w'_0 q_{r'} w'_1$ with

(1) $w'_0 = \varepsilon$
(2) $|w'_0| = |w_0| = |\varepsilon|$
(3) $w'_1 = a_{l'} v$ and $w_1 = a_l v$ for some word v.

So that we get

(1) $\ulcorner w'_0 \urcorner = \ulcorner \varepsilon \urcorner = 0$
(2) $|w'_0| = |\varepsilon| = 0$

(3) $\ulcorner w'_1 \urcorner = l' + \left[\dfrac{\beta_4^3(\ulcorner C \urcorner)}{k} \right] \cdot k.$

So, all in all we obtain:

$$if \begin{cases} \beta_4^1(\ulcorner C \urcorner) = r \\ and \\ \beta_4^3(\ulcorner C \urcorner) \dot{-} \left(k \cdot \left[\dfrac{\beta_4^3(\ulcorner C \urcorner)}{k} \right] \right) = l, \\ and \\ \beta_4^4(\ulcorner C \urcorner) = 0 \end{cases}$$

then

$$\ulcorner C' \urcorner = \alpha_4 \left(r', \quad 0, \quad l' + \left[\frac{\beta_4^3(\ulcorner C \urcorner)}{k} \right] \cdot k, \quad 0 \right).$$

if $\mathbf{w_0} \neq \varepsilon$, then we have $C \Rightarrow C'$ is $w_0 q_r w_1 \Rightarrow w'_0 q_{r'} w'_1$ with

(1) $w'_0 a_l = w_0$
(2) $|w'_0| = |w_0| - 1$

(3) $w'_1 = a_{l'} w_1.$

so that

(1) $\ulcorner w'_0 \urcorner = \beta_4^2(\ulcorner C \urcorner) \dot{-} \left[\dfrac{\beta_4^2(\ulcorner C \urcorner)}{k^{(\beta_4^4(\ulcorner C \urcorner) \dot{-} 1)}} \right] \cdot k^{(\beta_4^4(\ulcorner C \urcorner) \dot{-} 1)}$

(2) $|w'_0| = \beta_4^4(\ulcorner C \urcorner) \dot{-} 1$

(3) $\ulcorner w'_1 \urcorner = l' + \beta_4^3(\ulcorner C \urcorner) \cdot k.$

So, we have found:

$$\textit{if} \left[\begin{array}{c} \beta_4^1(\ulcorner C \urcorner) = r \\ \textit{and} \\ \beta_4^3(\ulcorner C \urcorner) \dot{-} \left(k \cdot \left[\dfrac{\beta_4^3(\ulcorner C \urcorner)}{k}\right]\right) = l \\ \textit{and} \\ \beta_4^4(\ulcorner C \urcorner) \neq 0 \end{array}\right], \text{ then}$$

$$\ulcorner C' \urcorner = \alpha_4 \left(r', \quad \beta_4^2(\ulcorner C \urcorner) \dot{-} \left[\frac{\beta_4^2(\ulcorner C \urcorner)}{k^{(\beta_4^4(\ulcorner C \urcorner) \dot{-} 1)}}\right] \cdot k^{(\beta_4^4(\ulcorner C \urcorner) \dot{-} 1)}, \quad l' + \beta_4^3(\ulcorner C \urcorner) \cdot k, \quad \beta_4^4(\ulcorner C \urcorner) \dot{-} 1\right).$$

To wrap up everything that we did so far, we recursively define a mapping $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ such that if n codes a word on Σ – i.e., $n = \ulcorner w \urcorner$ for some $w \in \Sigma^*$ – then $f(n,t) = \ulcorner C_{n,t} \urcorner$ where $C_{n,t}$ stands for the configuration that the Turing machine reaches after t-many steps from the initial configuration $q_0 w$.
Since the machine stops if it reaches an accepting or a rejecting configuration, we will simply assume that in any of these two cases the configuration of the Turing machines remains the same: if $C_{n,t}$ is either an accepting or a rejecting configuration, then $C_{n,t+x} = C_{n,t}$ holds for every $x \in \mathbb{N}$.
We set δ_L and δ_R are the two following finite – hence $\mathcal{P}rim.\ \mathcal{R}ec.$ – subsets of $\mathbb{N}^4$:

$$\delta_L = \Big\{(r,l,r',l') \in \{0,\dots m\} \times \{0 \dots k-1\} \times \{0,\dots m\} \times \{0 \dots k-1\} \ \mid \ \delta(r,l) = (r',l',L)\Big\}$$

and

$$\delta_R = \Big\{(r,l,r',l') \in \{0,\dots m\} \times \{0 \dots k-1\} \times \{0,\dots m\} \times \{0 \dots k-1\} \ \mid \ \delta(r,l) = (r',l',R)\Big\}$$

For our convenience we assume that $\Gamma = \Sigma \cup \{\sqcup\}$. This way the mapping $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ we currently construct is total ($f \in \mathbb{N}^{(\mathbb{N}^2)}$). For readability we use the notation $\overline{k}^i$ for $\beta_4^i(k)$ (any $k \in \mathbb{N}$, $i \leqslant 4$).

initial case

$$f(n,0) = \begin{cases} 1 & \text{if} \quad n \notin \mathit{Input}_{\mathcal{M}} \\ \alpha_4(0,0,n,0) & \text{if} \quad n \in \mathit{Input}_{\mathcal{M}} \end{cases}$$

successor case

$$f(n,t+1) =$$

$$\begin{cases} f(n,t) & \text{if} \quad \overline{f(n,t)}^1 = 1 \quad \text{or} \quad \overline{f(n,t)}^1 = 2; \\ \alpha_4\left(r',\ \overline{f(n,t)}^2 + l' \cdot k^{\overline{f(n,t)}^4},\ \left[\frac{\overline{f(n,t)}^3}{k}\right],\ \overline{f(n,t)}^4 + 1\right) & \text{if} \quad \left(\overline{f(n,t)}^1,\ \overline{f(n,t)}^3 \dot{-} \left(k \cdot \left[\frac{\overline{f(n,t)}^3}{k}\right]\right),\ r',\ l'\right) \in \delta_R; \\ \alpha_4\left(r',\ 0,\ l' + \left[\frac{\overline{f(n,t)}^3}{k}\right] \cdot k,\ 0\right) & \text{if} \quad \begin{cases} \left(\overline{f(n,t)}^1,\ \overline{f(n,t)}^3 \dot{-} \left(k \cdot \left[\frac{\overline{f(n,t)}^3}{k}\right]\right),\ r',\ l'\right) \in \delta_L \\ \text{and} \\ \overline{f(n,t)}^4 = 0; \end{cases} \\ \alpha_4\left(r',\ \overline{f(n,t)}^2 \dot{-} \left[\frac{\overline{f(n,t)}^2}{k^{\left(\overline{f(n,t)}^4 \dot{-} 1\right)}}\right] \cdot k^{\left(\overline{f(n,t)}^4 \dot{-} 1\right)},\ l' + \overline{f(n,t)}^3 \cdot k,\ \overline{f(n,t)}^4 \dot{-} 1\right) & \text{if} \quad \begin{cases} \left(\overline{f(n,t)}^1,\ \overline{f(n,t)}^3 \dot{-} \left(k \cdot \left[\frac{\overline{f(n,t)}^3}{k}\right]\right),\ r',\ l'\right) \in \delta_L \\ \text{and} \\ \overline{f(n,t)}^4 \neq 0. \end{cases} \end{cases}$$

Notice that $f \in \mathbb{N}^{(\mathbb{N}^2)}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$ and that $\mathcal{M}(w) \overset{\text{acc}}{\downarrow}$ if and only if there exists some $t \in \mathbb{N}$ such that $\beta_4^1\big(f(\ulcorner w \urcorner, t)\big) = 2$.
Moreover, in this case, we recover the code of the content of the tape w_t from $f(\ulcorner w \urcorner, t)$ by

$$\ulcorner w_t \urcorner \quad = \quad \beta_4^2\big(f(\ulcorner w \urcorner, t)\big) \quad + \quad k^{\beta_4^4\left(f(\ulcorner w \urcorner, t)\right)} \cdot \beta_4^3\big(f(\ulcorner w \urcorner, t)\big).$$

Finally, we only need to fix both (1) a recursive representation of the natural numbers and (2) what it means for a machine to compute a partial function.

(1) we fix our coding of integers: every integer n is coded by the word $\underbrace{11\ldots1}_{n}$. The function

$$\begin{array}{rccc} \ulcorner\ \urcorner : & \mathbb{N} & \longrightarrow & \mathbb{N} \\ & n & \longmapsto & \ulcorner n \urcorner \end{array}$$

such that $\ulcorner n \urcorner = \sum_{i<n} 1 \cdot k^i$ (i.e., $\ulcorner n \urcorner$ is the code of the word $\underbrace{11\ldots1}_{n}$) is $\mathcal{P}rim.\ \mathcal{R}ec.$:

$$\begin{cases} \ulcorner 0 \urcorner & = & 0 \\ \ulcorner n+1 \urcorner & = & \ulcorner n \urcorner + k^n. \end{cases}$$

(2) We only consider Turing machines that on any input words of the form $\overbrace{11\ldots1}^{n}$, provided they reach an accepting configuration, they reach one of the form $\underbrace{11\ldots1}_{n'} q_{acc.}$.

We define the function $f_{\mathcal{M}} \in \mathbb{N}^{(dom \subseteq \mathbb{N}^r)}$ that $\mathcal{M}$ computes by:

- $f_{\mathcal{M}}(n_1, \ldots, n_r)$ is undefined if on input $\overbrace{11\ldots1}^{\alpha_r(n_1,\ldots,n_r)}$ either $\mathcal{M} \uparrow$ or $\mathcal{M} \not\downarrow$[a];
- $f_{\mathcal{M}}(n_1, \ldots, n_r) = n$ if on input $\overbrace{11\ldots1}^{\alpha_r(n_1,\ldots,n_r)}$ $\mathcal{M} \downarrow_{acc.}$[b] in configuration $\underbrace{11\ldots1}_{n} q_{acc.}$.

Then the function $least_{\mathcal{M}} \in \mathbb{N}^{(dom \subseteq \mathbb{N}^r)}$ that picks the minimum number of steps – if any – the machine takes before reaching an accepting configuration when starting from the initial one $q_0 \underbrace{11\ldots\ldots1}_{\alpha_r(n_1,\ldots,n_r)}$ is defined by

$$least_{\mathcal{M}}(n_1, \ldots, n_r) = \mu t \quad \beta_4^1 \circ f\big(\ulcorner \alpha_r(n_1, \ldots, n_r) \urcorner, t\big) = 2.$$

It is undefined if the machine never halts or halts on the rejecting state.

At last, we are ready to provide the desired $f_{\mathcal{M}} \in \mathbb{N}^{(dom \subseteq \mathbb{N}^r)}$. We make use of the fact the position of the head in an accepting configuration indicates precisely the number of 1's there are on the tape:

$$f_{\mathcal{M}}(n_1, \ldots, n_r) = \beta_4^4 \circ f\Big(\ulcorner \alpha_r(n_1, \ldots, n_r) \urcorner, least_{\mathcal{M}}(n_1, \ldots, n_r)\Big).$$

□

[a] remember that this is the reason why we chose 0 for the coding of the blank symbol.

[b] this means whether or not the head is already in position 0 and the transition is of the form $\delta(q_r, a_l) = (q_{r'}, a_{l'}, L)$.

Corollary 5.1

Every partial recursive function $f \in \mathbb{N}^{(\mathbb{N}^r)}$ admits a construction that requires minimisation

at most once.
Moreover, one has $\forall n_1 \dots \forall n_r \forall k \quad \Big(f(n_1, \dots, n_r) = k \longleftrightarrow \exists t \quad F(n_1, \dots, n_r, k, t)\Big)$ where $F \subseteq \mathbb{N}^{r+2}$ is $\mathcal{P}rim.\ \mathcal{R}ec.$.

Proof of Corollary 5.1:

This is an immediate consequence of the whole proof of Lemma 5.2 since we proved that every partial recursive function can be computed by a Turing machine whose function it computes is $f_{\mathcal{M}} \in \mathbb{N}^{(\mathbb{N}^r)}$ defined by

$$f_{\mathcal{M}}(n_1, \dots, n_r) = \beta_4^4 \circ f\Big(\ulcorner \alpha_r(n_1, \dots, n_r) \urcorner, \mu t \quad \beta_4^1 \circ f\big(\ulcorner \alpha_r(n_1, \dots, n_r) \urcorner, t\big) = 2\Big).$$

where all functions $f \in \mathbb{N}^{(\mathbb{N}^2)}$, $\alpha_r \in \mathbb{N}^{(\mathbb{N}^r)}$, $\beta_4^1 \in \mathbb{N}^{\mathbb{N}}$, $\beta_4^4 \in \mathbb{N}^{\mathbb{N}}$, $\ulcorner\ \urcorner \in \mathbb{N}^{\mathbb{N}}$ and the constant $\overline{2} \in \mathbb{N}^{(\mathbb{N}^0)}$ are $\mathcal{P}rim.\ \mathcal{R}ec.$ as well as the equality relation.

□

Theorem 5.1

For every $k > 0$ and every $f \in \mathbb{N}^{(dom \subseteq \mathbb{N}^k)}$ the following are equivalent

- f is $\mathcal{P}art.\ \mathcal{R}ec.$,
- f is Turing computable.

Proof of Theorem 5.1:

Follows immediately from Lemmas 5.1 and 5.2. □