Computability theory

FAMAF - UNC

SLP

Contents

1 Introduction

These are my notes on computability theory. The only definitions that ought to be known first-hand are the following.

Let Σ denote an arbitrary language, $\omega := \mathbb{N} + \{0\}$, and $n, m \geq 0$ fixed elements in ω. Then:

- A Σ-mixed function is a function s.t. $\mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto \varphi$ with either $\varphi = \omega$ or $\varphi = \Sigma^*$.
- A Σ-mixed function is Σ-effectively computable if we can find some algorithmic procedure that, given an input in $\omega^n \times \Sigma^{*m}$, outputs the value of $f(\overrightarrow{x}, \overrightarrow{\alpha})$.
- Any set $S \subseteq \omega^n \times \Sigma^{*m}$ is termed a Σ-mixed set.
- If there is an algorithmic procedure that enumerates S, we say S is Σ-effectively enumerable.
- If there is an algorithmic procedure that decides the belonging to S, we say S is Σ-effectively computable.

Here, the notion of "algorithmic procedure" is non-rigorous. The principal subject of this study are three formalizations of this concept: Turing computability, recursive computability (Godel), and imperative computability (von Neumann).

2 Coding infinite tuples

We define $\omega^{\mathbb{N}} := \{(s_1, s_2, \ldots) : s_i \in \omega\}$ and $\omega^{[\mathbb{N}]} \subseteq \omega^{\mathbb{N}} := \{(s_1, s_2, \ldots) : s_i \in \omega \wedge \exists k \in \omega : i \geq k \Rightarrow s_i = 0\}$.

2.1 The *i*th prime function

We define

$$\begin{aligned} pr : \mathbb{N} &\mapsto \omega \\ n &\mapsto \text{ the } n\text{th prime number} \end{aligned}$$

Theorem 1 *For all $x \in \mathbb{N}$ there is a unique infinituple $\overrightarrow{s} \in \omega^{[\mathbb{N}]}$ s.t.*

$$x = \prod_{i=1}^{\infty} pr(i)^{s_i}$$

The theorem follows trivially from the definition of $\omega^{[\mathbb{N}]}$ and the fundamental theorem of arithmetic.

> **Problem 1** *Prove the previous theorem via complete induction.*
>
> The base case is trivial. Assume the statement holds for all $n \leq k$. The fundamental theorem of arithmetic ensures that $k + 1 = p_1 \cdot \ldots \cdot p_m$ where p_i is prime. Assume the factorization above is ordered (this is, $p_{j+1} > p_j$ for all $j \in [1, m]$). Then $k + 1 = p_m \cdot q$ with $q = p_1 \cdot \ldots \cdot p_{m-1}$.
>
> > *Subproof.* We will prove $k + 1 = p_m \cdot q \Rightarrow q \leq k$. Assume the premise holds and the consequence does not. Since $q > k$ we have $q \cdot x > k + 1$ for all $x > 1$. Then $q \cdot x > k + 1$ for all x that is prime. Then $q \cdot p_m \neq k + 1$ which is a contradiction. Then, if $k + 1 = q \cdot p_m$, we have $q \leq k$. ■
>
> Since $q \leq k$, via inductive hypothesis, q takes the productorial form of the theorem above. Then $k+1 = q \cdot pr(j)$ where $pr(j) = p_m$. Then the theorem holds for all $n \in \mathbb{N}$.

Theorem 2 *If $p, p_1, \ldots, p_m$ are prime ($m \geq 1$) and $p \mid p_1 \ldots p_m$, then $p = p_i$ for some i.*

Proof. Assume $p \mid p_1 \dots p_m$. $\therefore p \leq p_1 \dots p_m$. The uniqueness of the prime factorization tells us that $p_1 \dots p_n$ are factors of a unique integer x.

Assume $p \nmid p_j$ for any $j = 1, \dots, m$. $\therefore p$ is a prime factor of $p_1 \dots p_m$ distinct from p_j for all p_j. $\therefore x = p_1 \dots p_m p \neq x$. $\therefore \bot$

$\therefore p \mid p_i$ for some $i = 1, \dots, m$. ■

We use $\langle s_1, s_2, \dots \rangle$ to denote the number $x = \prod_{n=1}^{\infty} pr(n)^{s_n}$. We use $(x)_i$ to denote s_i in said tuple and (x) to denote the infinituple itself.

Theorem 3 *The functions*

$$\begin{aligned} \mathbb{N} &\mapsto \omega^{[\mathbb{N}]} & \omega^{[\mathbb{N}]} &\mapsto \mathbb{N} \\ x &\mapsto (x) = ((x)_1, (x)_2, \dots) & (s_1, s_2, \dots) &\mapsto \langle s_1, s_2, \dots \rangle \end{aligned}$$

are bijections each the inverse of the other.

The theorem should be intuitive. The function that maps a number x to the infinituple of its prime exponents is the inverse of the function which takes an infinituple and maps it to the product of its prime factors with the corresponding exponents.

Theorem 4

$$(x)_i = \max_t \left(pr(i)^t \mid x \right)$$

Proof. Let $x \in \mathbb{N}$ be s.t. $x = p_1^{s_1} \dots p_m^{s_m}$ with p_j prime and $m_j \neq 0$ for all $j = 1, \dots, m$. Evidently $(x)_i = s_i$.

Let $x \in \mathbb{N}$ and

$$\mathscr{P}_i := \left\{ t \in \omega : pr(i)^t \mid x \right\}$$

Since $\mathscr{P}_i \subseteq \mathbb{N}$ is necessary finite it has a maximum m_i. The fundamental theorem of arithmetic directly gives $x = pr(1)^{m_1} pr(2)^{m_2} \dots$ Then by definition $m_i = (x)_i$. ■

We define

$$\begin{aligned} &Lt : \mathbb{N} \mapsto \omega \\ &\qquad x \mapsto \begin{cases} \max_i \ (x)_i \neq 0 & x \neq 1 \\ 0 & x = 1 \end{cases} \end{aligned}$$

The function returns the index of the maximum prime factor (that is not zero-exponentiated) in the factorization of x. Since, in this factorizations, all prime factors beyond $Lt(x)$ are zero, $Lt(x)$ can be understood as an upper bound of the factorization. This is formalized in the following theorem.

Theorem 5

$$x = \prod_{i=1}^{Lt(x)} pr(i)^{(x)_i}$$

Problem 2 *Prove the previous theorem.*

$x = \prod_{i=1}^{\infty} pr(i)^{(x)_i} \wedge (x) \in \omega^{[\mathbb{N}]} \therefore \exists k \in \omega : i \geq k \Rightarrow (x)_i = 0$ $\therefore x = \prod_{i=1}^{k-1} pr(i)^{(x)_i} \times \prod_{i=k}^{\infty} pr(i)^0 = \prod_{i=1}^{k-1} pr(i)^{(x)_i}$. But $k - 1 := \max_i ((x)_i \neq 0) := Lt(x)$. ■

Prime numbers closely relate to set enumerability. The reason is that the prime decomposition of a number associates to any unique $x \in \mathbb{N}$ an infinite number of inters $(x)_1, (x)_2, \ldots$. Say we want to generate all possible pairs $(x, y, z) \in \omega^3$ given a unique input $x \in \mathbb{N}$. Then, letting $(x, y, z) = ((x)_1, (x)_2, (x)_3)$, we have

$$\begin{aligned}
x = 1 &\mapsto (0, 0, 0) \\
x = 2 &\mapsto (1, 0, 0) \\
x = 3 &\mapsto (0, 1, 0) \\
x = 4 &\mapsto (2, 0, 0) \\
x = 5 &\mapsto (0, 0, 1) \\
x = 6 &\mapsto (1, 1, 0) \\
x = 7 &\mapsto (0, 0, 0) \\
x = 8 &\mapsto (3, 0, 0) \\
x = 9 &\mapsto (0, 2, 0) \\
&\vdots
\end{aligned}$$

It is easy to see that any $(x, y, z) \in \omega^3$ is reached. This generalizes to $\omega^n, n \in \omega$ and to $\omega^n \times \Sigma^{*m}$ via the $*^{\leq}$ function (which we shall study in the following section).

2.2 Orders over Σ

Let Σ an alphabet with n symbols. We want to find a bijection between ω and Σ^* assuming some order $\leq$ over Σ. Let $s^{\leq} : \Sigma^* \mapsto \Sigma^*$ be

$$s^{\leq}((a_n)^m) = (a_1)^{m+1} \qquad m \geq 0$$
$$s^{\leq}(\alpha a_i(a_n)^m) = \alpha a_{i+1}(a_1)^m \qquad 1 \leq i < n, m \geq 0$$

This function enumerates the language ordered Σ. For example, consider $\Sigma = \{@, !\}$ with $@ <!$. Then

$$\begin{aligned} s^{\leq}(\varepsilon) &= s^{\leq}(!^0) = @ \\ s^{\leq}(@) &= s^{\leq}(\varepsilon @ (!)^0) = \varepsilon ! \varepsilon = \ ! \\ &\vdots \end{aligned}$$

Repeated application of this logic outputs the following enumeration:

$$@, !, @@, @!, !@, !!, @@@, @@!, @!@, @!!, !@@, !@!, !!@, !!!, \ldots$$

The reason why $s^{\leq}(\beta)$ enumerates the language is that every β is either of the form $(a_n)^m$ or $\alpha a_i(a_n)^m$. This is, it is either a word with only the last character to a certain exponent, or a word with some subchain before the last character to a certain exponent.

Now we are ready to define a bijection between ω and Σ^*. Let

$$\begin{aligned} *^{\leq} &: \omega \mapsto \Sigma^* \\ x &\mapsto \begin{cases} \varepsilon & x = 0 \\ s^{\leq}\left(*^{\leq}(i)\right) & x = i + 1 \end{cases} \end{aligned}$$

For example, using the same alphabet as before, this function maps

$$\begin{aligned} 0 &\mapsto \varepsilon \\ 1 &\mapsto @ \\ 2 &\mapsto ! \\ 3 &\mapsto @@ \\ 4 &\mapsto @! \\ 5 &\mapsto !@ \\ 6 &\mapsto !! \\ 7 &\mapsto @@@ \\ &\vdots \end{aligned}$$

Now, observe that any $\alpha \in \Sigma^*$ is a concatenation of unique symbols, and that each of this unique symbols is the ith element of Σ^* for some i. We write to express this $\alpha = a_{i_k} \dots a_{i_0}$ where $i_k, i_{k-1}, \dots, i_{k_0} \in \{1, \dots, n\}$. Then we define the inverse of the previous function as follows:

$$\begin{aligned} \#^{\leq} : \Sigma^* &\mapsto \omega \\ \varepsilon &\mapsto 0 \\ a_{i_k} \dots a_{i_0} &\mapsto i_k n^k + \dots + i_0 n^0 \end{aligned}$$

For example, consider $\alpha = @!@ = a_1 a_2 a_1$. Then $\#^{\leq}(\alpha) = 1 \times 2^2 + 2 \times 2^1 + 1 \times 2^0 = 4 + 4 + 1 = 9$. It is easy to verify that $*^{\leq}(9) = @!@$.

Thus, the functions given produce a perfect bijection between numbers and words. Each word can be univocally determined by its numeric position in the language; each number can be univocally determined by a word whose position in the language is that number.

Theorem 6 *Let $n \geq 1$. Then any $x \in \mathbb{N}$ is uniquely written as $x = i_k n^k + i_{k-1} n^{k-1} + \dots + i_0 n_0$ with $k \geq 0, 1 \leq i_j \leq n$ for all j.*

2.3 Extending the order to words

We can extend $\leq$ from Σ onto Σ^* by letting $\alpha \leq \beta$ if and only if $\#^{\leq}(\alpha) \leq \#^{(\leq)}(\beta)$.

3 Enumerable and computable sets

Let $\mathcal{F} : \mathcal{D}_\mathcal{F} \subseteq \omega^k \times \Sigma^{*l} \to \omega^n \times \Sigma^{*m}$. We define $\mathcal{F}_{(i)} = p_i^{n,m} \circ \mathcal{F}$. Then

$$\begin{aligned} \mathcal{F}_{(i)} &: \mathcal{D}_\mathcal{F} \subseteq \omega^k \times \Sigma^{*l} \mapsto \omega && 1 \leq i \leq n \\ \mathcal{F}_{(i)} &: \mathcal{D}_\mathcal{F} \subseteq \omega^k \times \Sigma^{*l} \mapsto \Sigma^* && n+1 \leq i \leq m \end{aligned}$$

We say a set $S \subseteq \omega^n \times \Sigma^{*m}$ is Σ-effectively enumerable if it is empty or there is a function $\mathcal{F} : \omega \to \omega^n \times \Sigma^{*m}$ s.t. $Im_\mathcal{F} = S$ and $\mathcal{F}_{(i)}$ is Σ-computable for all $1 \leq i \leq n+m$.

Theorem 7 *A non-empty set $S \subseteq \omega^n \times \Sigma^{*m}$ is Σ-effectively enumerable if and only if there is an effective procedure $\mathcal{P}$ s.t.*

- *The input space is ω*
- *$\mathcal{P}$ halts for all $x \in \omega$*
- *The output set is S—i.e. whenever $\mathcal{P}$ halts, it outputs an element of S, and for every $(\overrightarrow{x}, \overrightarrow{\alpha}) \in S$ there is some input $x \in \omega$ s.t. $\mathcal{P}(x) \mapsto_{halting} (\overrightarrow{x}, \overrightarrow{\alpha})$.*

Problem 3 *Let $F : \{(x, \alpha) \in \omega \times \Sigma^* : |\alpha|^x \equiv 0 \mod 2\} \mapsto \omega^2 \times \Sigma^*$ be defined as follows:*

$$F(x, \alpha) = (x, x^2 + |\alpha|, \varepsilon)$$

Provide $F_{(1)}, F_{(2)}, F_{(3)}$. What function is $[F_{(1)}, F_{(2)}, F_{(3)}]$?

By definition, $F_{(i)} = p_i^{1,1} \circ F$. Then

$$\begin{aligned} F_{(1)}(x, \alpha) &= x \\ F_{(2)}(x, \alpha) &= x + |\alpha| \\ F_{(3)}(x, \alpha) &= \varepsilon \end{aligned}$$

It is evident that the composition given results in F.

Problem 4 *Prove that $F = \left[F_{(1)}, \ldots, F_{(n+m)}\right]$.*

Let $G := \left[F_{(1)}, \ldots, F_{(n+m)}\right]$.
$\therefore G(\vec{x}, \vec{\alpha}) = \left(p_1^{n,m} \circ F(\vec{x}, \vec{\alpha}), \ldots, p_{n+m} \circ F(\vec{x}, \vec{\alpha})\right) = F(\vec{x}, \vec{\alpha})$ ■

Theorem 8 *Let $S \subseteq \omega^n \times \Sigma^{*m}$. The following statements are equivalent.*

(1) S is Σ-effectively enumerable.

(2) S is the domain of a Σ-effectively computable function f.

(3) S is the image of a function $F : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto S$ s.t. each $F_{(i)}$ is Σ-effectively computable.

That (1) $\Leftrightarrow$ (3) is trivial. (1) $\Leftrightarrow$ (2) can be proven as follows.

> **Proof.** ($\Rightarrow$). Assume S is Σ-effectively enumerable. Then there is a procedure $\mathbb{P}$ s.t. for any $x \in \omega$ the procedure outputs a value of S, and all values of S are mapped. Consider the procedure $\mathbb{P}'$ s.t. given an input $(\vec{x}, \vec{\alpha}) \in S$ it *a.* computes $\mathbb{P}$ with input $x = 0, 1, \ldots$ until $\mathbb{P}$ maps to $(\vec{x}, \vec{\alpha})$ and *b.* then returns x. Evidently $\mathbb{P}'$ computes the inverse of the function computed by $\mathbb{P}$. Observe that $\mathbb{P}'$ will only halt if $(\vec{x}, \vec{\alpha}) \in S$. Then S is the domain of a Σ-effectively computable function.
>
> ($\Leftarrow$) Assume S is the domain of a Σ-effectively computable function. Let $(\vec{w}, \vec{\beta})$ an arbitrary element of S. Consider a procedure $\mathbb{P}$ which does the following with an input $x \in \omega$.
>
> > *(1)* Produce the enumeration of $\omega \times \omega^n \times \Sigma^{*m}$ given by x. This will produce a variable $(t, \vec{x}, \vec{\alpha})$.
> >
> > *(2)* Run t steps of $\mathbb{P}_f$ with input $(\vec{x}, \vec{\alpha})$. If the program terminated, return $(\vec{x}, \vec{\alpha})$. If not, return $(\vec{w}, \vec{\beta})$.
>
> Evidently, $\mathbb{P}$ enumerates S. ■

Since any Σ-effectively computable set is Σ-effectively enumerable, point *(1)* of the previous theorem can be substituted by S is Σ-effectively enumerable—with some loss of generality.

Theorem 9 *Let $f : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto \varphi$ with φ either ω or Σ^*. Let $S \subseteq \mathcal{I}_f$. Then if f is Σ-effectively computable and S is Σ-effectively enumerable, $f^{-1}(S) = \{(\vec{x}, \vec{\alpha}) : f(\vec{x}, \vec{\alpha}) \in S\}$ is Σ-effectively enumerable .*

> **Proof.** Assume f is Σ-effectively computable and $S \subseteq \mathcal{I}_f$ is Σ-effectively enumerable. Let $(\vec{w}, \vec{\beta})$ an arbitrary element of $f^{-1}(S)$. Consider the following effective procedure operating over an input $x \in \omega$:
>
> > *(0)* If $x = 0$ return $(\vec{w}, \vec{\beta})$. Else proceed.
> >
> > *(1)* Enumerate an element of $s \in S$ using input x.
> >
> > *(2)* Use $(\vec{x}, \vec{\alpha}) := ((x)_1, \ldots, (x)_n, *^{\leq}((x)_{n+1}), \ldots, *^{\leq}((x)_m))$ to compute a value of $f(\vec{x}, \vec{\alpha})$.
> >
> > *(3)* If $f(\vec{x}, \vec{\alpha}) = s$ return $(\vec{x}, \vec{\alpha})$. Else return $(\vec{w}, \vec{\beta})$.

Evidently, the procedure enumerates $f^{-1}(S)$. ■

The theorem states that the set of inputs which map to a region of a computable function's range is enumerable if that region is enumerable.

Theorem 10 *If $f : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto \varphi$ with φ either ω or Σ^*, then if $S \subseteq \mathcal{D}_f$ is Σ-effectively enumerable, $f_{|S}$ is Σ-effectively computable.*

3.1 Prime numbers and enumerable sets

Let $\Sigma \neq \emptyset$ be an alphabet with a total order $\leq$. Let $S \subseteq \omega^n \times \Sigma^{*m}$ a Σ-mixed set of arbitrary dimensions. Notice that for any n-tuple $(x_1, \ldots, x_n)$, with $x_i \in \omega$, we can find a corresponding $\varphi \in \mathbb{N}$ s.t.

$$\varphi = 2^{x_1} 3^{x_2} \ldots pr(n)^{x_n}$$

In other words, $(x_1, \ldots, x_n)$ corresponds to the exponents of the n prime factors of a unique natural number. At the same time, the m-tuple $(\alpha_1, \ldots, \alpha_m)$ corresponds to a unique $\psi \in \mathbb{N}$ s.t.

$$\psi = 2^{y_1} 3^{y_2} \ldots pr(m)^{y_m}$$

where $\alpha_j = *^{\leq}(y_j)$. In other words, $(\alpha_1, \ldots, \alpha_m)$ corresponds to a unique natural number whose m prime factors have exponents given by the position of each word in the language.

Both of these relations come from the uniqueness of prime factorizations. They provide a way to enumerate Σ-mixed sets. In particular, if S is Σ-total we enumerate it mapping each $x \in \omega$ to $\big((x)_1, \ldots, (x)_n, *^{\leq}((x)_{n+1}), \ldots, *^{\leq}((x)_m)\big)$. If S is not Σ-total, then one can still enumerate it assuming that it is Σ-computable. Indeed, one maps x to the corresponding $(n+m)$-tuple described above if the tuple is in S, and leaves the procedure undefined (or without halt) otherwise. This can be expressed as follows:

Because Σ-total sets are enumerable (as pointed out above), any Σ-mixed set that is Σ-computable is enumerable (via restriction of the Σ-total enumeration).

Theorem 11 *If $S \subseteq \omega^n \times \Sigma^{*m}$ is Σ-effectively computable, then it is Σ-effectively enumerable.*

> **Problem 5** *Prove that $S = \{(x, @^x) : x \equiv 0 \mod 2\}$ is $\{@\}$-effectively enumerable.*
>
> S is not Σ-total. $\therefore$ We require that it be Σ-effectively computable. It is easy to see that it is: Simply make a procedure that checks if x is even, and that

counts the number of @s. Let $\mathbb{P}_{\text{is in } S}$ denote the procedure which decides the belonging of (x, α) to S.

It is also trivial to observe that $*^{\leq}$ is Σ-effectively computable under any order of Σ, since the function is pretty much algorithmic in nature. Let $\mathbb{P}_{\text{num to word}}$ denote the procedure which given an input $x \in \omega$ computes $*^{\leq}(x)$. Then we define $\mathbb{P}$ the following procedure which takes an input $x \in \omega$:

> *(0)* Let x_1 be the number of times 2 divides x (this is, let it be the exponent of the first prime factor in the decomposition of x). Let x_2 be the number of times 3 divides x_1 (the second prime in the decomposition of x).
> *(1)* Use $\mathbb{P}_{\text{num to word}}$ to compute $*^{\leq}(x_2)$ and store it in α.
> *(3)* Use $\mathbb{P}_{\text{is in } S}$ to determine if $(x_0, \alpha) \in S$. If it is, output (x_0, α). If it is not, go to step three.

Example. Consider $\mathbb{P}(6)$. In *(0)* this maps $x_1 = 1, x_2 = 1$. Since the first word in Σ is @, $6 \mapsto (1, @)$, which is not in S. $\mathbb{P}$ will not halt.

Consider the tuple $(2, @@@@)$. We know there exists some $x \in \omega$ s.t. $\mathbb{P}(x) = (2, @@@@)$ (I use math notation loosely here). Since @@@@ is the fourth word in Σ, x is s.t. $x = \langle 2, 4, (x)_3, (x)_4, \ldots\ldots \rangle$. For example, $2^2 + 3^4 = 85$ or $2^2 + 3^4 + 5^{17} = 762939453210$ will satisfy this.

Problem 6 *Prove the following statement: If $S \subseteq \omega$ and $f : S \mapsto \omega$ is Σ-effectively computable, then*

$$A = \{x \in S : x \text{ is even } \wedge x/2 \in S \wedge f(x) = f(x/2)\}$$

is Σ-effectively enumerable.

Assume f is Σ-effectively computable. $\therefore$ S is Σ-effectively enumerable.

Let $\mathbb{P}_S$ be the procedure that enumerates S. Let $\mathbb{P}_f$ denote the procedure which computes f, $w \in S$ fixed and arbitrary, and $\mathbb{P}$ with input $x \in \omega$ the following procedure:

> *(0)* If $x = 0$ return w and finish.
> *(1)* Use $\mathbb{P}_S$ twice, with inputs $(x)_1, (x)_2$ respectively, to produce elements $s_1, s_2 \in S$.
> *(2)* Check if s_1 is even (this is trivially effectively computable). If it isn't, return w and finish.
> *(3)* Check if $s_2 = s_1/2$. If it isn't, return w and finish.
> *(4)* Use $\mathbb{P}_f$ to compute $f(s_1), f(s_2)$ and compare these values. If they are not equal, return w and finish.
> *(5)* Return s_1.

Evidently, $\mathbb{P}$ enumerates A.

4 Turing

From now on, we will attempt formalizations of three so far informal concepts:

- Σ-effectively computable functions
- Σ-effectively computable sets
- Σ-effectively enumerable sets

The first formalization is given by Turing.

4.1 Turing machine

A Turing machine is a 7-uple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

- Q is a set of states
- $\Gamma \supset \Sigma$ is an alphabet
- Σ is the input alphabet
- $B \in \Gamma - \Sigma$ is a blank symbol
- $\delta : Q \times \Gamma \mapsto \mathcal{P}\,(Q \times \Gamma \times \{L, R, K\})$
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states

Problem 7 *If M a Turing machine then δ is a Σ-mixed function.*

A function is said to be a Σ-mixed function if $\mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m}$ for some $n, m \geq 0$ and $\mathcal{I}_f \subseteq \omega$ or $\mathcal{I}_f \subseteq \Sigma^*$. The δ function satisfies neither of these properties; its domain is a set of states $Q \times \Gamma \nsubseteq \Sigma^{*m}$ and its image is a set of sets.

Problem 8 *If M a Turing machine, $\mathcal{D}_\delta$ is a Σ-mixed set.*

A set S is said to be Σ-mixed iff $S \subseteq \omega^n \times \Sigma^{*m}$ for some $n, m \geq 0$. We have already mentioned that $\mathcal{D}_\delta = Q \times \Gamma \nsubseteq \omega^n \times \Sigma^{*m}$ for any n, m. Then $\mathcal{D}_\delta$ is not Σ-mixed.

Problem 9 *If M a Turing machine, then $\mathcal{I}_\delta$ is Σ-mixed.*

False again.

4.2 Deterministic Turing machine

A Turing machine is said to be deterministic iff $|\delta(p, \sigma)| \leq 1$ for all $p \in Q, \sigma \in \Gamma$.

4.3 Instantaneous descriptions

An instantaneous description is a word of the form $\alpha q \beta$ where $\alpha, \beta \in \Gamma^*$, $[\beta]_{|\beta|} \neq B$ and $q \in Q$. If the instantaneous description is $\alpha_1 \alpha_2 \ldots \alpha_n q \beta_1 \beta_2 \ldots \beta_m BBB \ldots$, we read: *The Turing machine is in state q and it is reading* β_1. We use $\mathbb{D}$ to denote the set of instantaneous descriptions. We define

$$\begin{aligned} St : \mathbb{D} &\mapsto Q \\ d &\mapsto \text{Only symbol of } Q \text{ that is in } d \end{aligned}$$

Problem 10 *Let $d \in \mathbb{D}$ an instantaneous description. Then $Ti(d)$ is a triple.*

False: d is not a triple but a single element of $(\Gamma \cup Q)^*$.

Problem 11 *If $d \in \mathbb{D}$ then $St(d) = d \cap Q$*

False. The operation $d \cap Q$ makes no sense, insofar as d is a symbol. It would be correct to say $St(d) = \{d_1, d_2, \ldots, d_m\} \cap Q$ where d_i are the characters in the word d.

4.4 State transitions

Given $\alpha \in (\Gamma \cup Q)^*$ we define

$$\begin{aligned} \lfloor \varepsilon \rfloor &= \varepsilon \\ \lfloor \alpha\sigma \rfloor &= \alpha\sigma \text{ if } \sigma \neq B \\ \lfloor \alpha B \rfloor &= \lfloor \alpha \rfloor \end{aligned}$$

Thus, $\lfloor \alpha \rfloor$ removes the trailing blank symbols of α (if any).

Given $d_1, d_2 \in \mathbb{D}$ with $d_1 = \alpha p \beta$, we say $d_1 \vdash d_2$ if, given $\alpha \in \Gamma, \alpha, \beta \in \Gamma^*, p, q \in Q$, one of the following three cases hold.

(Case 1) $\alpha \neq \varepsilon$, and

$$\delta\left(p, [\beta B]_1\right) \ni (q, \sigma, L)$$

and

$$d_2 = \left\lfloor \alpha \frown q [\alpha]_{|\alpha|} \sigma \frown \beta \right\rfloor$$

Interpretation. The Turing machine at state p will write σ at its current position, transition to state q, and move to the left.

Example. Let $\Sigma = \{@, \#\}$. Assuming $\delta(p) = \{(q, \#, L)\}$, then the following is an example of *Case 1*.

$$@\ \#\ @\ p\ @\ @\ @\ B\ B\ B \vdash @\ \#\ q\ @\ \#\ @\ @\ B\ B\ \ldots$$

(Case 2)

$$\delta\left(p, [\beta B]_1\right) \ni (q, \sigma, R)$$

and

$$d_2 = \alpha\sigma q^\frown\beta$$

Interpretation. The Turing machine at state p will write σ at its current position, transition to state q, and move to the left.

Example. Let $\Sigma = \{@, \#\}$. Assuming $\delta(p) = \{(q, \#, R)\}$, then the following is an example of *Case 2*.

$$@\ \#\ @\ p\ @\ @\ @\ B\ B\ B \vdash @\ \#\ @\ \#\ q\ @\ @\ B\ B\ \ldots$$

(Case 3)

$$\delta\left(p, [\beta B]_1\right) \ni (q, \sigma, K)$$

and

$$d_2 = \lfloor\alpha q\sigma^\frown\beta\rfloor$$

Interpretation. The Turing machine at state p will write σ at its current position, transition to state q, and stay at the same position.

Example. Let $\Sigma = \{@, \#\}$. Assuming $\delta(p) = \{(q, \#, K)\}$, then the following is an example of *Case 2*.

$$@\ \#\ @\ p\ @\ @\ @\ B\ B\ B \vdash @\ \#\ @\ q\ \#\ @\ @\ B\ B\ \ldots$$

We say $d \vdash^n d'$ if there are $d_1, \ldots, d_{n+1}$ s.t. $d = d_1, d' = d_{n+1}$, and $d_i \vdash d_{i+1}$ for all $i = 1, \ldots, n$. Observe that $d \vdash^0 d'$ if $d = d'$. Finally, we denote $d \vdash^* d'$ iff $(\exists n \in \omega)\ d \vdash^n d'$.

Problem 12 *Determine true or false for the following propositions.*

(1) $d \vdash d$ *for all* $d \in \mathbb{D}$. The proposition is false. It is trivial to find a counterexample.

(2) If $\alpha p \beta \nvdash d$ *for every* $d \in \mathbb{D}$*, then* $\delta(p, [\beta B]_1) = \emptyset$. Assume $\alpha p \beta \nvdash d$ for every $d \in \mathbb{D}$. Assume $\delta(p, [\beta B]_1) \neq \emptyset$. Then there must be some $\{(q, \sigma, D)\}$ with $D \in \{L, R, K\}$ that corresponds to this evaluation of δ. But then there would exist some d, given by the case division above and depending on the value of D, s.t. $\alpha p \beta \vdash d$. But this is a contradiction. The statement is true.

(3) If $(p, \alpha, L) \in \delta(p, a)$ *then* $pa \nvdash d$ *for all* $d \in \mathbb{D}$. This is correct. Remember that for a transition to the left to be defined we require that a substring $\alpha \neq \varepsilon$ precede the Machine's head. (See *Case 1*, requirement $\alpha \neq \varepsilon$.) But here $d_1 = pa$ has an initial segment ε preceding p. Then $pa \neq d$ for any d. The statement is true.

(4) Given $d_1, d_2 \in \mathbb{D}$*, if* $d_1 \vdash d_2$ *then* $|d_1| \leq |d_2| + 1$. It makes no sense to say $|d_1| \leq |d_2| + 1$ insofar as an instantaneous description contains infinitely many symbols B at the end. So the statement, as it is phrased, is false. However, consider the alternative postulate: $d_1 \vdash d_2 \Rightarrow |\lfloor d_1 \rfloor| \leq |\lfloor d_2 \rfloor| + 1$. Two instantaneous description over the same machine always have the same number of symbols. So $|\lfloor d_1 \rfloor| = |\lfloor d_2 \rfloor|$, which makes the statement trivially true.

Problem 13 *Prove that M is deterministic iff for each* $d \in \mathbb{D}$ *there is at most one* $d' \in \mathbb{D}$ *s.t.* $d \vdash d'$.

($\Rightarrow$) Assume M is deterministic. Then for any $d \in \mathbb{D}$ of the form $\alpha q \beta BB \ldots$, we have either $\delta(q) = \{(q', \sigma, D)\}$ or $\delta(q) = \emptyset$. If $\delta(q) = \emptyset$, then (by definition of $\vdash$) there is no instantaneous description d' s.t. $d \vdash d'$. If $\delta(q) = \{(q', \sigma, D)\}$, then two cases are possible. *(1) d* holds the assumptions sustaining the case definition of $\vdash$, in which case the transition is uniquely determined by (q', σ, D). *(2) d* does not hold the assumptions sustaining the case definition of $\vdash$ (e.g. $D = L, \alpha = \varepsilon$), in which case there is by definition no d' s.t. $d \vdash d'$

($\Leftarrow$) This does not hold!! Assume that, for all $d \in \mathbb{D}$, there is at most one d' s.t. $d \vdash d'$. Consider the case where there is no d' s.t. $d \vdash d'$. In the case where $\alpha = \varepsilon$ and

$$\delta(q) = \{(q', \sigma, L), (q'', \sigma', L)\}$$

the machine is non-deterministic without violating the assumption. In other words, that $d \vdash d'$ for at most one d' does not imply that the machine is deterministic.

4.5 Halting and languages

Given $d \in \mathbb{D}$, we say M halts starting from d if there is some $d' \in \mathbb{D}$ s.t.

$$d \vdash^* d'$$
$$d' \nvdash d'' \text{ for all } d'' \in \mathbb{D}$$

We say a word $w \in \Sigma^*$ is accepted by a Turing machine M *by reach of final state* if

$$\exists d \in \mathbb{D} : \lfloor q_0 B w \rfloor \vdash^* d \wedge \ St(d) \in F$$

Observe that we do not require that $d \in St(d)$ satisfies $d \nvdash d'$ for all $d' \in \mathbb{D}$. In other words, a machine may accept a word by reach of final state and not halt.

The language accepted by a Turing machine is

$$\mathcal{L}(M) = \{w \in \Sigma^* : w \text{ is accepted by reach of final state }\}$$

Problem 14 *Let $\Sigma = \{a, b\}$. Find a Turing machine M that accepts the language $\{w \in \Sigma^* : |\omega|_a = 2|\omega|_b\}$*

As examples, here are some words accepted by the language: $\varepsilon, aab, baa, ababaa, \ldots$ Our machine will operate in the following manner.

(1) It will parse the string looking for a b; if a b is found, it replaces it with y, sets the head of the machine to the beginning, and proceeds to parse for two as and replace them with xs. This gives

$$\delta(q_0, \sigma) = \begin{cases} \{(q_1, y, L)\} & \sigma = b \\ \{(q_0, \sigma, R)\} & \sigma = a \\ \{(q_9, \sigma, L)\} & \sigma = B \end{cases}$$
$$\delta(q_1, \sigma) = \begin{cases} \{(q_1, \sigma, L)\} & \sigma \neq \varepsilon \\ \{(q_2, \varepsilon, R)\} & \sigma = \varepsilon \end{cases}$$

where q_1 resets the machine's head to the initial position. Now

$$\delta(q_2, \sigma) = \begin{cases} \{(q_3, x, R)\} & \sigma = a \\ \{(q_2, \sigma, R)\} & \sigma \neq a \end{cases}$$
$$\delta(q_3, \sigma) = \begin{cases} \{(q_4, x, R)\} & \sigma = a \\ \{(q_3, \sigma, R)\} & \sigma \neq a \end{cases}$$

The state q_4 occurs after two as have been found to correspond to a single b. Thus, we must return to q_0 with the machine head reset to its initial position.

$$\delta(q_4, \sigma) = \begin{cases} \{(q_4, \sigma, L)\} & \sigma \neq \varepsilon \\ \{(q_0, \sigma, R)\} & \sigma = \varepsilon \end{cases}$$

The only undefined state is q_9. This is the state reached after, in state q_0, the string is parsed but no b is found. If there is some a left in the string, then the word must not be accepted; but if no a exists, the word must be accepted. Thus, q_9 can parse from right to left looking for as; if ε is reached without as on the way, it should map to a final state. If some a is found, it should never halt.

$$\delta(q_9, \sigma) = \begin{cases} [(q_9, \sigma, L)] & \sigma \neq\in \{a, \varepsilon\} \\ [(q_9, \sigma, K)] & \sigma = a \\ [(q_{\mathcal{F}}, \sigma, K)] & \sigma = \varepsilon \end{cases}$$

where $q_{\mathcal{F}} \in F$ is the only final state. Of course, $\Gamma = \{a, b, x, y, B\}, Q = \{q_0, q_1, q_2, q_3, q_4, q_9, q_{\mathcal{F}}\}$.

We say a word $w \in \Sigma^*$ is accepted by detention (or by a halt) if M halts when starting from $\lfloor q_0 B \rfloor$. We denote

$$\mathcal{H}(M) = \{w \in \Sigma^* : w \text{ is accepted by detention}\}$$

Theorem 12 *Let $L \subseteq \Sigma^*$. Then the following statements are equivalent:*
(1) There is a Turing machine M s.t. $L = L(M)$.
(2) There is a Turing machine M s.t. $L = \mathcal{H}(M)$.

4.6 Σ-Turing computable functions

To represent numbers on the Turing tape, we use the i letter. Thus, we extend the definition of a Turing machine ensuring that $i \in \Gamma - (\{B\} \cup \Sigma)$ is a symbol.

Let $f : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto \Sigma^*$ a Σ-mixed function. We say f is Σ-Turing computable if there is a deterministic Turing machine M s.t.

- If $(\vec{x}, \vec{\alpha}) \in \mathcal{D}_f$, then there is a state $p \in Q$ s.t.

$$\lfloor q_0 \, B \, i^{x_1} \, B \, \ldots \, i^{x_n} \, B \, \alpha_1 \, B \, \ldots \, B \, \alpha_m \rfloor \vdash^* \lfloor pBf(\vec{x}, \vec{\alpha}) \rfloor$$

- If $(\vec{x}, \vec{\alpha}) \in \omega^n \times \Sigma^{*m} - \mathcal{D}_f$, then the machine does not halt starting from

$$\lfloor q_0 \, B \, i^{x_1} \, B \, \ldots \, i^{x_n} \, B \, \alpha_1 \, B \, \ldots \, B \, \alpha_m \rfloor$$

When $f \mapsto \omega$, we analogously say a Turing machine computes f iff

- If $(\vec{x}, \vec{\alpha}) \in \mathcal{D}_f$, then there is a state $p \in Q$ s.t.

$$\lfloor q_0 \, B \, i^{x_1} \, B \, \ldots \, i^{x_n} \, B \, \alpha_1 \, B \, \ldots \, B \, \alpha_m \rfloor \vdash^* \left\lfloor pBi^{f(\vec{x}, \vec{\alpha})} \right\rfloor$$

- If $(\vec{x}, \vec{\alpha}) \in \omega^n \times \Sigma^{*m} - \mathcal{D}_f$, then the machine does not halt starting from

$$\lfloor q_0 \, B \, i^{x_1} \, B \, \ldots \, i^{x_n} \, B \, \alpha_1 \, B \, \ldots \, B \, \alpha_m \rfloor$$

If M and f satisfy the properties above, we say f is computed by M.

Theorem 13 *If $f : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto I \in \{\omega, \Sigma^*\}$ is computed by a Turing machine M, then f is Σ-effectively computable.*

> **Problem 15** *Determine if the following statements are true or false.*
>
> *(1) If M a deterministic turing machine and M computes f, then $\mathcal{L}(M) = \mathcal{D}_f$.*
>
> Since M computes f, M halts for any $(\vec{x}, \vec{\alpha}) \in \mathcal{D}_f$ and does not halt for any $(\vec{x}, \vec{\alpha}) \notin \mathcal{D}_f$. Then, by definition, $\mathcal{H}(M) = \mathcal{D}_f$. This does not imply $\mathcal{L}(M) = \mathcal{D}_f$. However, it does imply that there necessarily exists a Turing machine M' s.t. $\mathcal{L}(M') = \mathcal{D}_f$.
>
> *(2) If f, g are two functions and M is a Turing machine that computes f and g, then $f = g$.* Assume the premise holds. Then $\mathcal{H}(M) = \mathcal{D}_f = \mathcal{D}_g := \mathcal{D}$. Furthermore, for any $(\vec{x}, \vec{\alpha}) \in \mathcal{D}$, the Turing machine halts with an instantaneous description representing $f(\vec{x}, \vec{\alpha})$. But since it computes g, the

instantaneous description also represents $g(\vec{x}, \vec{\alpha})$. This implies $f(\vec{x}, \vec{\alpha}) = g(\vec{x}, \vec{\alpha})$. Then $f = g$. The statement is true.

(3) Let M a Turing machine and assume M computes f with f a Σ-total function. Then M halts from d for whichever $d \in \mathbb{D}$. It is true that, since f is Σ-total, for any $(\vec{x}, \vec{\alpha}) \in \omega^n \times \Sigma^{*m}$, $f(\vec{x}, \vec{\alpha})$ is defined. This does imply that every d of the form $\lfloor q_0 i^{x_1} \dots i^{x_n} \alpha_1 \dots \alpha_m \rfloor$ is accepted (I skipped the possibly infinite B symbols in the description). However, there is no restriction in $\mathbb{D}$ as to the number of unit or alphabetic symbols which an arbitrary $d \in \mathbb{D}$ may contain. Thus,

$$\mathbb{D}' \subseteq \mathbb{D} = \{d \in \mathbb{D} : \lfloor d \rfloor = \lfloor q_0 i^{x_1} \dots i^{x_k} \alpha_1 \dots \alpha_l \rfloor : k \neq n \vee l \notin m\}$$

satisfies $D' \nsubseteq \mathcal{H}(M)$.

5 Godel

Definition 1 *A set $S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m$ is rectangular if $S_i \subseteq \omega, L_i \subseteq \Sigma^*$ for all i.*

Lemma 1 *S is rectangular if and only if $(\overrightarrow{x}, \overrightarrow{\alpha}) \in S \wedge (\overrightarrow{y}, \overrightarrow{\beta}) \in S$ implies $(\overrightarrow{x}, \overrightarrow{\beta}) \in S$.*

Example. The set $\{(0, \#\#), (1, \%\%\%)\}$ is not rectangular ($(1, \#\#), (0, \%\%\%)$ are not in S.) Observe how this set cannot be expressed as a product of subsets of ω and Σ. Thus, the concept of *rectangular set* is equivalent to *a set formed via Cartesian product*.

Notation. If $f : \omega_1 \times \ldots \times \omega_n \times \alpha_1 \times \alpha_m \to \Lambda$ we write $f \sim (n, m, \Lambda)$, and read f is of type n, m to Λ.

Notation. If $f_1, \ldots, f_n$ Σ-mixed functions, then

$$[f_1, \ldots, f_2]\,(\overrightarrow{x}, \overrightarrow{\alpha}) = \left(f_1(\overrightarrow{x}, \overrightarrow{\alpha}), \ldots, f_n(\overrightarrow{x}, \overrightarrow{\alpha})\right)$$

5.1 Primitive recursion

Let R a Σ-mixed function s.t. $R \sim (n, m, \varphi)$, where $\varphi = \omega$ or $\varphi = \Sigma^*$. Primitive recursion consists in recursively defining R with two primitive functions f, which gives the base case, and g, which gives the recursive step. The recursion is done over ω, associating $R(t, \vec{x}, \vec{\alpha})$ with $R(t-1, \vec{x}, \vec{\alpha})$, or over Σ^*, associating $R(\vec{x}, \vec{\alpha}, \alpha a)$ with $R(\vec{x}, \vec{\alpha}, \alpha)$. Since the computation of R may depend on the element which the recursion dismisses (t in the numeric case, a in the alphabetic case), g incorporates this value into its arguments. Thus, we have that:

- If $\varphi = \omega$
  - If the recursion is over ω, $f \sim (n-1, m, \omega), g \sim (n+2, m, \omega)$
  - If the recursion is over Σ^*, $f \sim (n, m-1, \omega), g \sim (n+1, m+1, \omega)$.
- If $\varphi = \Sigma^*$
  - If the recursion is over ω, $f \sim (n-1, m, \Sigma^*), g \sim (n+1, m+1, \Sigma^*)$
  - If the recursion is over Σ^*, $f \sim (n, m-1, \Sigma^*), g \sim (n, m+2, \Sigma^*)$.

We say f, g construct R via primitive recursion and we denote R as $R(f, g)$.

5.2 Numeric to numeric primitive recursion

Let $R \sim (n, m, \#)$. Then functions $f \sim (n-1, m, \#), g \sim (n+1, m, \#)$ recursively define R if and only if

$$\begin{cases} R(0, \overrightarrow{x}, \overrightarrow{\alpha}) & = f(\overrightarrow{x}, \overrightarrow{\alpha}) \\ R(t+1, \overrightarrow{x}, \overrightarrow{\alpha}) & = g\left(R(t, \overrightarrow{x}, \overrightarrow{\alpha}), t, \overrightarrow{x}, \overrightarrow{\alpha}\right) \end{cases}$$

We use the notation $R(f, g)$ to say R is defined by primitive recursion by f and g.

Problem 16 *Find functions that recursively define $R = \lambda t\,[2^t]$*

Since $R\ (1, 0, \#)$ we know $f \sim (0, 0, \#)$ is a constant function and $g \sim (2, 0, \#)$. Since $R(0) = 1$ we know $f = C_1^{0,0}$. Observe that $R(t+1) = R(t) \times 2$. Thus we may let $g = \lambda x[2 \cdot x] \circ p_1^{2,0}$.

Example. $R(2) = \lambda x[2x] \circ p_1^{2,0}\,(R(1), 2) = 2 \times R(1) = 2 \times (2 \times R(0)) = 2 \times 2 \times 1 = 4$.

Problem 17 *Define $R(t) = \lambda t x_1\left[x_1^t\right]$ recursively.*

Since $R \sim (2, 0, \#)$ we know $f \sim (1, 0, \#)$ and $g \sim (3, 0, \#)$. Now, $R(0, x_1) = 1 \implies f = C_1^{1,0}$. Since $R(t+1, x_1) = R(t, x_1) \cdot x_1$ we observe that $g = \lambda xy[xy] \circ \left[p_1^{3,0}, p_3^{3,0}\right]$. Since each $p_k^{3,0} \sim (3, 0, \#)$ we have that g is of the desired type.

Problem 18 *Is it true that $R(\lambda xy[0], p_2^{4,0}) = p_1^{3,0}$?*

$R \sim (2, 0, \#)$; $f \sim (2, 0, \#)$. So f cannot be a primitive constructor of R.

Problem 19 *Determine true or false: If $f : \omega^2 \to \omega$ and $g : \omega^4 \to \omega$, then for each $(x, y) \in \omega^2$ we have*

$$R(f, g)(2, x, y) = g \circ \left(g \circ \left[f \circ \left[p_2^{3,0}, p_2^{3,0}\right], p_1^{3,0}, p_2^{3,0}, p_3^{3,0}\right]\right)(0, x, y).$$

Passing the arguments into the functions this results in

$$\begin{aligned} R(f, g)(2, x, y) &= g \circ (g \circ [f(x, x), 0, x, y]) \\ &= g \circ (g\,(f(x, x), 0, x, y)) \end{aligned}$$

But the expression makes no sense, since $\zeta = g(f(x, x), 0, x, y) \in \omega$ is not a function and hence $g \circ \zeta$ is undefined.

5.3 Numeric to alphabet primitive recursion

Let $R \sim (n, m, \Sigma)$. Then functions $f \sim (n-1, m, \Sigma), g \sim (n, m+1, \Sigma)$ recursively define R if and only if

$$\begin{aligned} R(0, \overrightarrow{x}, \overrightarrow{\alpha}) &= f(\overrightarrow{x}, \overrightarrow{\alpha}) \\ R(t+1, \overrightarrow{x}, \overrightarrow{\alpha}) &= g\left(t, \overrightarrow{x}, \overrightarrow{\alpha}, R(t, \overrightarrow{x}, \overrightarrow{\alpha})\right) \end{aligned}$$

Problem 20 *Let* $\Sigma = \{\%, @, ?\}$. *Define* $R = \lambda t x_1[\%@\%\%\%\%?^t]$ *via primitive recursion.*

Let $f = C^{1,0}_{\%@\%\%\%\%}$ and $g = d_? \circ \left[p^{2,1}_3\right]$. For example, $R(3, x_1) = d_? \circ [R(2, x_1)] = d_? \circ [d_? \circ R[1, x_1]] = d_? \circ \left[d_? \circ \left[d_? \circ \left[C^{1,0}_{\%@\%\%\%\%}\right]\right]\right] = \%@\%\%\%\%???$.

Problem 21 *True or false: If* f, g *are* Σ*-mixed s.t.* $R(f, g) \sim (1+n, m, *)$, *then* $f \sim (n, m, *)$ *and* $g \sim (n, m+1, *)$.

False. The g function must have the same number of numeric arguments than R.

5.4 Alphabet to numeric primitive recursion

If Σ an alphabet, then a Σ-indexed family of functions is a function $\mathcal{G}$ s.t. $D_{\mathcal{G}} = \Sigma$ and for each $a \in D_{\mathcal{G}}$ there is a function $\mathcal{G}(a)$. We write $\mathcal{G}_a$ instead of $\mathcal{G}(a)$.

If $R \sim (n, m, \omega)$ then R can be recursively defined by $f \sim (n, m-1, \omega)$ an indexed family $\mathcal{G}$ s.t. $\mathcal{G}_a \sim (n+1, m, \omega)$ as follows:

$$\begin{cases} R(F, \mathcal{G})(\overrightarrow{x}, \overrightarrow{\alpha}, \varepsilon) = f(\overrightarrow{x}, \overrightarrow{\alpha}) \\ R(f, \mathcal{G})(\overrightarrow{x}, \overrightarrow{a}, \alpha a) = \mathcal{G}_a\left(R(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha), \overrightarrow{x}, \overrightarrow{\alpha}, \alpha\right) \end{cases}$$

Problem 22 *Let* $\Sigma = \{\%, @, ?\}$. *Find* $f, \mathcal{G}$ *s.t.* $R = \lambda\alpha_1\alpha\,[|\alpha_1| + |\alpha|_@]$.

$R \sim (0, 2, \#)$. Since $R(\alpha_1, \varepsilon) = |\alpha_1|$ we let $f := \lambda\alpha = |\alpha|$. Now, $g \sim (1, 2, \#)$ is given by $g := \mathcal{G}$ where

$$\begin{aligned} \mathcal{G} : \Sigma &\to \{Suc \circ p^{1,2}_1, p^{1,2}_1\} \\ \% &= p^{1,2}_2 \\ ? &= p^{1,2}_2 \\ @ &= Suc \circ p^{1,2}_2 \end{aligned}$$

For example, $R(??, @\%?@) = \mathcal{G}_@\,(R(@\%?), ??, @) = 1 + R(??, @\%?)$. This boils down to $1 + R(??, @) = 1 + 1 + R(??, \varepsilon) = 2 + |??| = 2$, the desired output.

5.5 Alphabet to alphabet primitive recursion

If $R \sim (n, m, *)$ then $f \sim (n, m-1, *)$ and $\mathcal{G}$ a Σ-indexed family, with $\mathcal{G}_a \sim (n, m+1, *)$ for all $a \in \Sigma$, define R via primitive recursion if

$$\begin{cases} R(\overrightarrow{x}_n, \overrightarrow{\alpha}_{m-1}, \varepsilon) & = f(\overrightarrow{x}, \overrightarrow{\alpha}) \\ R(\overrightarrow{x_n}, \overrightarrow{\alpha}_{m-1}, \alpha a) & = \mathcal{G}_a\left(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha, R(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha)\right) \end{cases}$$

Problem 23 *Let* $\Sigma = \{@, ?\}$. *Define* $R = \lambda\alpha_1\alpha[\alpha_1\alpha]$ *recursively.*

Observe that $R \sim (0, 2, *)$. $R(\alpha_1, \varepsilon) = \alpha_1 \implies f := \lambda\alpha[\alpha]$. Now, we let $\mathcal{G}_a = d_a \circ p_3^{0,3}$ for all $a \in \Sigma$, and the recursion is complete.

Example. The evaluation for arbitrary inputs looks as follows:

$$\begin{aligned} R(?@?, @?) &= d_? \,(R(?@?, @)) \\ &= d_? \,(d_@ \,(R(?@?, \varepsilon))) \\ &= d_? \,(d_@ \,(?@?)) \\ &= d_? \,(?@?@) \\ &= ?@?@? \end{aligned}$$

5.6 The point of primitive recursion

Theorem 14 *If* f, g *are* Σ*-computable then* $R(f, g)$ *is too.*

5.7 The primitive recursive set

Let Σ a language. We define $PR_0^\Sigma = \left\{Suc, Pred, C_0^{0,0}, C_\varepsilon^{0,0}\right\} \cup \{d_a\} \cup \left\{p_j^{n,m}\right\}$. Observe that every $\mathcal{F} \in PR_0^\Sigma$ is Σ-computable. Then we define

$$PR_{k+1} = PR_k^\Sigma \cup \left\{f \circ [f_1 \dots f_r] : f \text{ and } f_i \in PR_k^{\Sigma\cup}\right\} \cup \left\{R(f, g) : f, g \in PR_k^\Sigma\right\}$$

In other words, PR_k^Σ is the set of all functions that are either compositions of functions in PR_{k-1}^Σ or functions built via primitive recursion by functions in PR_{k-1}^Σ. The total primitive recursive set PR^Σ is defined as $PR^\Sigma = \bigcup_{k\geq 0} PR_k^\Sigma$.

Note. Observe that when we include $R(f,g) : f,g \in PR_k^\Sigma$, we also include the case where $g = \mathcal{G}$ an indexed family of functions.

Observation Due to the previous theorem, we know $\mathcal{F} \in PR \Rightarrow \mathcal{F}$ is Σ-computable.

I provide a list of functions that are in PR^Σ for any Σ.

- Addition, multiplication and factorial
- String concatenation and string length
- All constant functions $C_k^{n,m}$ for any $k, n, m \in \omega$.
- Two-variable exponentiation: $\lambda xy\,[x^y]$.
- Two-variable string exponentiation: $\lambda x\alpha\,[\alpha^x]$.

With $x - y := \max(x - y, 0)$ the list may continue:

- The maximum of two numeric variables
- The predicates $x = y, x \leq y, \alpha = \beta$.
- The predicate x is even.
- The predicate $x = |\alpha|$.
- The predicate $\alpha^x = \beta$.

5.8 Predicates

The $\vee, \wedge$ operators are defined only for predicates of the same type. In other words, $P \circ Q$, where $\circ \in \{\wedge, \vee\}$, is defined only if $P \sim (n, m, \#) \wedge Q \sim (n, m, \#)$. If P, Q are Σ-p.r. then $P \circ Q$ and $\neg P$ also are. Furthermore, P, Q must have the same domains.

5.9 Primitive recursive sets

A Σ-mixed $S \sim (n, m)$ set is primitive recursive if and only if its characteristic function $\chi_S^{\omega^n \times \Sigma^{m*}}$ is p.r. Recall that $\chi_S^{n,m} = \lambda \overrightarrow{x}\,\overrightarrow{\alpha}\,[(\overrightarrow{x}, \overrightarrow{\alpha}) \in S]$.

If S_1, S_2 are Σ-p.r. then their union, intersection and difference are. The proof follows from the fact that

$$\chi_{S_1 \cup S_2} = \left(\chi_{S_1} \vee \chi_{S_2}\right)$$
$$\chi_{S_1 \cap S_2} = \left(\chi_{S_1} \wedge \chi_{S_2}\right)$$
$$\chi_{S_1 - S_2} = \lambda xy[x - y] \circ \left[\chi_{S_1}, \chi_{S_2}\right]$$

The only property here that may not be immediately intuitive is the last one. But observe that $S_1 - S_2 = \{s \in S_1 : s \notin S_2\}$. Now, let $\chi_{S_1}(\overrightarrow{x}, \overrightarrow{\alpha}) = a, \chi_{S_2}(\overrightarrow{x}, \overrightarrow{\alpha}) = b$. Evidently, if the $n+m$-tuple is in S_1 but not in S_2, $a - b = 1$. If the tuple is in both sets, $a - b = 0$. Etc.

Theorem 15 *A rectangular set $S_1 \times \ldots \times S_n \times L_1 \times \ldots L_m$ is Σ-p.r. if and only if each $S_1, \ldots, S_n, L_1, \ldots, L_m$ is Σ-p.r.*

This theorem is important, insofar as it allows us to evaluate whether a Cartesian product is Σ-p.r. only by looking at its set factors. This theorem should follow from the properties of primitive recursive sets mentioned before.

Theorem 16 *If $f \sim (n, m, \Omega)$ is Σ-p.r (not necessarily Σ-total) and S is a Σ-p.r. set, then $f_{|S}$ is Σ-p.r.*

The previous theorem is useful in proving a function is Σ-p.r. For example, let $P = \lambda x\alpha\beta\gamma\left[x = |\gamma| \wedge \alpha = \gamma^{Pred(|\beta|)}\right]$. We cannot use the fact that both predicate functions are Σ-p.r. to conclude that P is Σ-p.r., because $P_1 = \lambda x\alpha[x = |\alpha|]$ and $P_2 = \lambda x\alpha\beta\gamma[\alpha = \gamma^{Pred(|\beta|)}]$ do not have the same domains. Simply observe that β cannot take the value ε in P_2, but it can take in P_1.

However, observe that $\mathcal{D}_P = \omega \times \Sigma^* \times (\Sigma^* - \varepsilon) \times \Sigma^*$. This set is Σ-p.r. because $\chi^{1,3}_{\mathcal{D}_P} = \neg\lambda\,[\alpha = \beta] \circ \left[p^{1,3}_3, C^{1,3}_\varepsilon\right]$ is Σ-p.r. Now, we can safely say that $P = P_{1|\mathcal{D}_P} \wedge P_2$, ensuring with the restriction that both predicates have the same domain. Since $\mathcal{D}_P$ is Σ-p.r. so is $P_{1|\mathcal{D}_P}$, form which readily follows that so is P. ■

Theorem 17 *A set S is Σ-p.r. if and only if it is the domain of a Σ-p.r. function.*

5.10 Case division

If $f_1, \ldots, f_n$ are s.t. $D_{f_j} \cap D_{f_k} = \emptyset$ for $j \neq k$ and $f_j \mapsto \Omega$, then $\mathcal{F} = f_1 \cup \ldots \cup f_n$ is s.t.

$$\mathcal{F} : D_{f_1} \cup \ldots \cup D_{f_n} \to \Omega$$
$$e \to \begin{cases} f_1(e) & e \in D_{f_1} \\ \vdots & \\ f_n(e) & e \in D_{f_n} \end{cases}$$

Under the same constraints, if f_i is Σ-p.r. for all i, then $\mathcal{F}$ is Σ-p.r. This reveals a proving method. Given a function $\mathcal{H}$, we can prove it is Σ-p.r. by proving it is the union of Σ-p.r. functions, under the constraint that the domains of these functions are disjoint.

For example, this can be used to prove that $\lambda\alpha\,[[\alpha]_i]$ is Σ-p.r. Assume a language Σ. Then

$$[\alpha a]_i = \begin{cases} a & i = |\alpha| + 1 \\ [\alpha]_i & \text{otherwise} \end{cases}$$

for any $a \in \Sigma$. The base case is the trivial $[\varepsilon]_i = \varepsilon$. From this follows that $R = [\alpha]_i \sim (1, 1)$ is difined via primitive recursion by $f = C_\varepsilon^{1,0}$ and $\mathcal{G}$ an indexed family where $\mathcal{G}_a$ is of the form above for every a. Evidently f is Σ-p.r.; now we want to prove $\mathcal{G}_a$ is Σ-p.r. for any $a \in \Sigma$.

Observe that the sets $S = \{(i, \alpha, \zeta) : i = |\alpha| + 1\}$ and its complement $\overline{S}$ are disjoint and Σ-p.r. (We skip the proof of this statement.) It follows from the division by cases that

$$\mathcal{G}_a = p_3^{1,2}{}_{|S} \cup C_a^{1,2}{}_{|\overline{S}}$$

is Σ-p.r. Thus, $R = [\alpha]_i$ is Σ-p.r.

Problem 24 *Let $\Sigma = \{@, \$\}$. Let $h : \mathbb{N} \times \Sigma^+ \mapsto \omega$ be x^2 if $x + |\alpha|$ is even, 0 otherwise. Prove that f is Σ-p.r.*

Let $S = \{(n, \alpha) \in \mathbb{N} \times \Sigma^* : n + |\alpha| \equiv 0 \mod 2\}$ and $g := \lambda x\alpha\left[x^2\right]$.

g is Σ-p.r.

$\therefore g_S \cup C_0^{1,1}{}_{\overline{S}} = f$ is Σ-p.r.

Problem 25 *Let h have $\mathcal{D}_h = \{(x, y, \alpha) : x \leq y\}$ and be s.t. $R \mapsto x^2$ if $|\alpha| \leq y$, zero otherwise. Show h is Σ-p.r.*

Let $S := \{(x, y, \alpha) \in \mathcal{D}_h : y \leq |\alpha|\}$. Evidently, $h = f_1 = C_0^{2,3}$ when $|\alpha| > y$ (this is, when the argument is in $\overline{S}$). When the argument is in S, it is $f_2 = \lambda x[x^2] \circ [p_1^{2,1}]$. It is trivial to observe both functions are Σ-p.r. Then $h = f_{1|\overline{S}} \cup f_{2|S}$, where of course $S \cup \overline{S} = \mathcal{D}_h$.

5.11 Summation, product and concatenation

Let $f \sim (n+1, m, \#)$ with domain $\mathcal{D}_f = \omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m$, with $S_i \subseteq \omega, L_i \subseteq \Sigma^*$. Then we define $\sum_{t=x}^{t=y} f(t, \overrightarrow{x}, \overrightarrow{\alpha})$ in the usual way, with the constraint that the sum is 0 if $y > x$. In the same way we deifne $\prod_{t=x}^{t=y} f(t, \overrightarrow{x}, \overrightarrow{\alpha})$ and the concatenation $\subset_{t=x}^{t=y} f(t, \overrightarrow{x}, \overrightarrow{\alpha})$ for the case $I_f \subseteq \Sigma^*$.

The domain of each of these is $\mathcal{D} = \omega \times \omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m$, where the first two ω elements are the x, y domains of the sum.

Theorem 18 *If f is Σ-p.r. then the functions are Σ-p.r.*

To understand why, let $G = \lambda t x \overrightarrow{x}\,\overrightarrow{\alpha} \left[\sum_{i=x}^{i=t} f(i, \overrightarrow{x}, \overrightarrow{\alpha})\right]$. Evidently, $G = \circ \left[p_2^{n+2,m}, p_1^{n+2,m}, p_3^{n+2,m}, \ldots, p_{n+2+m}^{n+2,m}\right]$ and so we only need to prove G is Σ-p.r. Observe that

$$G(0, x, \overrightarrow{x}, \overrightarrow{\alpha}) = \begin{cases} 0 & x > 0 \\ f(0, \overrightarrow{x}, \overrightarrow{\alpha}) & x = 0 \end{cases}$$

$$G(t+1, x, \overrightarrow{x}, \overrightarrow{\alpha}) = \begin{cases} 0 & x > t+1 \\ G(t, x, \overrightarrow{x}, \overrightarrow{\alpha}) + f(t+1, \overrightarrow{x}, \overrightarrow{\alpha}) & \end{cases}$$

Thus, if we let each of these functions be called h, g we have that $G = R(h, g)$. Suffices to show h, g are Σ-p.r. This can be proven using division by cases and domain restriction.

Problem 26 *Prove that $G = \lambda x x_1 \left[\sum_{t=1}^{t=x} Pred(x_1)^t\right]$ is Σ-p.r.*

We know $f = \lambda x t \left[Pred(x)^t\right]$ is Σ-p.r. (trivial to show). Let $\mathcal{G} = \lambda x y x_1 \left[\sum_{t=x}^{t=y} f(x_1, t)\right]$. We know from the last theorem that $\mathcal{G}$ is Σ-p.r. It is evident that $G = \mathcal{G} \circ \left[C_1^{2,0}, p_1^{2,0}, p_2^{2,0}\right]$. Then G is Σ-p.r. ■

Show it to me. Well, $G(x, x_1) = \left(\mathcal{G} \circ \left[C_1^{2,0}, p_1^{2,0}, p_2^{2,0}\right]\right)(x, x_1) = \mathcal{G}(0, x, x_1) = \sum_{t=0}^{t=x} f(x_1, t)$.

Problem 27 *Show that $G = \lambda x y \alpha \left[\prod_{t=y+1}^{t=|\alpha|} (t + |\alpha|)\right]$ is Σ-p.r.*

It is trivial to show $f = \lambda t \alpha \left[t + |\alpha|\right]$ is Σ-p.r. Let

$$\mathcal{G} = \lambda x y \alpha \left[\prod_{t=x}^{t=y} (t + |\alpha|)\right]$$

which is Σ-p.r. Observe that $G(x, y, \alpha) = \mathcal{G}(y + 1, |\alpha|, \alpha)$. Then

$$G = \mathcal{G} \circ \left[Suc \circ p_2^{2,1}, \lambda\alpha[|\alpha|] \circ p_3^{2,1}, p_3^{2,1} \right]$$

Then G is Σ-p.r. ■

Problem 28 *Prove that*

$$\lambda xyz\alpha\beta \left[\underset{t=3}{\overset{t=z+5}{\subset}} \alpha^{Pred(z)\cdot t} \beta^{Pred(Pred(|\alpha|))} \right]$$

is Σ*-p.r.*

Let G denote the function in question. First of all, observe that $\mathcal{D}_G = \omega^2 \times \mathbb{N} \times \Sigma^{*2}$—which means G is not Σ-total. Let us divide our proof by parts.

(1) Let $\mathcal{F} = \lambda xy\alpha\beta \left[\alpha^{Pred(x)\cdot y} \beta^{Pred(Pred(|\alpha|))} \right]$, where evidently $\mathcal{F} \sim (2, 2, *)$ with $x \in \mathbb{N}$. Observe that

$$\begin{aligned}
\mathcal{F}_1 &:= \lambda xy\alpha \left[\alpha^{Pred(x)y} \right] \\
&= \lambda x\alpha \left[\alpha^x \right] \circ \left[\lambda xy \left[xy \right] \circ \left[Pred \circ p_1^{2,1}, p_2^{2,1} \right], p_3^{2,1} \right] \\
\mathcal{F}_2 &:= \lambda\alpha\beta \left[\alpha^{Pred(Pred(|\alpha|))} \right] \\
&= \lambda x\alpha[\alpha^x] \circ \left[p_1^{0,2}, Pred \circ \left[Pred \circ \left[\lambda\alpha[|\alpha|] \circ p_2^{0,2} \right] \right] \right]
\end{aligned}$$

and evidently

$$\begin{aligned}
\mathcal{F} &= \lambda xy\alpha\beta[\mathcal{F}_1(x, y, \alpha)\mathcal{F}_2(\beta, \alpha)] \\
&= \lambda\alpha\beta[\alpha\beta] \circ \left[\mathcal{F}_1 \circ \left[p_1^{2,2}, p_2^{2,2}, p_3^{2,2} \right], \mathcal{F}_2 \circ \left[p_4^{2,2}, p_3^{2,2} \right] \right]
\end{aligned}$$

This proves $\mathcal{F}$ is Σ-p.r.

(2) It is evident that $G = \lambda xyz\alpha\beta \left[\subset_{t=3}^{t=z+5} \mathcal{F}(z, t, \alpha, \beta) \right]$. If we let

$$\mathcal{G} := \lambda xyz\alpha\beta \left[\underset{t=x}{\overset{t=y}{\subset}} \mathcal{F}(z, t, \alpha, \beta) \right]$$

it is evident that $G = \mathcal{G} \circ \left[C_3^{3,2}, \lambda z[z + 5] \circ p_3^{3,2}, p_3^{3,2}, p_4^{3,2}, p_5^{3,2} \right]$. Then G is Σ-p.r. ■

5.12 Predicate quantification

If $P : S_0 \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m$ is a predicate and $S \subseteq S_0$, then

$(\forall t \in S)_{t \leq x} P(t, \overrightarrow{x}, \overrightarrow{\alpha})$ is 1 when $P(t, \overrightarrow{x}, \overrightarrow{\alpha}) = 1$ for all $t \in \{u \in S : u \leq x\}$. The domain of the quantified proposition is $\omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m$, where the first argument (accounted by ω) is the upper bound x. We generalize, where $L \subseteq L_{m+1}, S \subseteq S_0$:

$$
\begin{aligned}
(\forall t \in S)_{t \leq x} P(t, \overrightarrow{x}, \overrightarrow{\alpha}) &: \omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m \to \{0, 1\} \\
(\exists t \in S)_{t \leq x} P(t, \overrightarrow{x}, \overrightarrow{\alpha}) &: \omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m \to \{0, 1\} \\
(\forall \alpha \in L)_{|\alpha| \leq x} P(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha) &: \omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m \to \{0, 1\} \\
(\exists \alpha \in L)_{|\alpha| \leq x} P(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha) &: \omega \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m \to \{0, 1\}
\end{aligned}
$$

It is important to observe that the set over which the quantification is done is a subset of the set from which comes the driving variable t (in the numeric case) or α (in the alphabetic case).

Theorem 19 *(1) If $P : S_0 \times S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m \to \omega$ a predicate Σ-p.r., and $S \subseteq S_0$ is Σ-p.r., then both quantifications over P are Σ-p.r.*

(2) If $P : S_1 \times \ldots \times S_n \times L_1 \times \ldots \times L_m L_{m+1} \to \omega$ a predicate Σ-p.r., and $L \subseteq L_{m+1}$ is Σ-p.r., then both quantifications over P are Σ-p.r.

The theorem above states that the quantification over a Σ-p.r. set of a Σ-p.r. predicate is itself Σ-p.r. Though unbounded quantification does not preserve these properties, in general a bound exists "naturally" for quantifications, which serves to prove that a bounded quantification is Σ-p.r.. Consider the following example.

Example. The predicate $\lambda xy[x \mid y]$ is Σ-p.r, because $P = x_1 x_2 [x_2 = t x_1]$ is Σ-p.r. Since P is Σ-p.r., any **bounded** quantification of it over a Σ-p.r. set is itself Σ-p.r. For example,

$$\lambda x x_1 x_2 \left[(\exists t \in \omega)_{t \leq x} x_2 = t x_1\right]$$

is Σ-p.r. Now, observe that if $x_2 = t x_1$ then it is necessary that $t \leq x_2$. But

$$
\begin{aligned}
&\lambda x_1 x_2 \left[(\exists t \in \omega)_{t \leq x_2} x_2 = t x_1\right] \\
=&\lambda x x_1 x_2 \left[(\exists t \in \omega)_{t \leq x} x_2 = t x_1\right] \circ \left[p_2^{2,0}, p_1^{2,0}, p_2^{2,0}\right]
\end{aligned}
$$

Then the **bounded** quantification, with x_2 as bound, is Σ-p.r.

Problem 29 *Let* $\Sigma = \{@, !\}$. *Show that* $S = \{(2^x, @^x, !) : x \in \omega \wedge x \text{ impar}\}$ *is* Σ*-p.r.*

For clarity, observe that a few elements of S are

$$(2, @, !), (8, @@@, !), (32, @@@@@, !), \ldots$$

Let $P_1 = \lambda xy\alpha \left[x = 2^{y+1}\right], P_2 = \lambda xy\alpha[\alpha = @^{y+1}]$. It is clear that $\mathcal{D}_{P_1} = \mathcal{D}_{P_2}$. It is trivial to prove that both are Σ-p.r. Then $P_1 \wedge P_2$ is Σ-p.r. Then

$$\chi_S^{1,2} = \lambda xy\alpha\beta\left[(\exists k \in \omega)_{k \leq x}\left(P_1(y, k, \alpha) \wedge P_2(y, y, \alpha)\right) \wedge \beta =!\right]$$

is Σ-p.r.

5.13 Minimization of numeric variable

Let P an arbitrary predicate over a numeric variable. If there is some $t \in \omega$ s.t. $P(t, \overrightarrow{x}, \overrightarrow{\alpha})$ holds, we use $\min_t P(t, \overrightarrow{x}, \overrightarrow{\alpha})$ to denote the minimum t that holds. This is **not defined** if there is no tuple $(\overrightarrow{x}, \overrightarrow{\alpha})$ over which the predicate holds. Furthermore, $\min_t P(t, \overrightarrow{x}, \overrightarrow{\alpha}) = \min_i P(i, \overrightarrow{x}, \overrightarrow{\alpha})$; this is, $\min_t$ does not depend on the variable t.

We define

$$M(P) = \lambda \overrightarrow{x}\,\overrightarrow{\alpha}\left[\min_t P(t, \overrightarrow{x}, \overrightarrow{\alpha})\right]$$

We say $M(P)$ is obtained via minimization of the numeric variable from P.

Example. Let $Q : \omega \times \mathbb{N}$ be s.t. $Q(x, y)$ denotes the quotient of $\frac{x}{y}$. This quotient is by definition the maximum element of $\{t \in \omega : ty \leq x\}$. Let $P = \lambda txy\,[ty \leq x]$. Observe that

$$\mathcal{D}_{M(P)} = \left\{(x, y) \in \omega^2 : (\exists t \in \omega)P(t, x, y) = 1\right\}$$

If $(x, y) \in \omega \times \mathbb{N}$, one can show that $\min_t x < ty = Q(x, y) + 1$. Then $M(P) = Suc \circ Q$.

The U rule. If f is a Σ-mixed function with type $(n, m, \#)$ and we want to find a predicat P s.t. $f = M(P)$, it is sometimes useful to design P so

$$f(\overrightarrow{x}, \overrightarrow{\alpha}) = \text{only } t \in \omega \text{ s.t. } P(t, \overrightarrow{x}, \overrightarrow{\alpha})$$

Problem 30 *Use the* ***U rule*** *to find a predicate P s.t.* $M(P) = \lambda x[$*integer part of* $\sqrt{x}]$
.

Let $f(x)$ denote the integer part of $\sqrt{x}$. If $f(x) = y$ then $y^2 \leq x \wedge (y+1)^2 > x$. Then letting $P = \lambda xy \left[x^2 \leq y \wedge (x+1)^2 > y\right]$ ensures that $M(P(x, y)) = f(x)$.

Problem 31 *Find P s.t.* $M(P) = \lambda xy\,[x - y]$.

Since $x - y$ is unique for each pair x, y, $P = \lambda xyz[z = x - y]$. Then $\min_z P(x, y, z) = \lambda xy[x - y]$. For example, $3-5 = 0$ and $\min_z P(3, 5, z) = 0$.

Theorem 20 *If P a predicate that is effectively computable and* $\mathcal{D}_P$ *is effectively computable, then* $M(P)$ *is effectively computable.*

5.14 Recursive function

Now we define $R_0^\Sigma = PR_0^\Sigma$ and

$$\begin{aligned} R_{k+1}^\Sigma =& R_k^\Sigma \\ & \cup \left\{f \circ [f_1, \ldots, f_n] : f_i \in R_k^\Sigma\right\} \\ & \cup \{R(f, g) : f, g \in R_k^\Sigma\} \\ & \cup \left\{M(P) : \text{P is } \Sigma\text{-total} \wedge P \in R_k^\Sigma\right\} \end{aligned}$$

In other words, recursive functions are all primitive recursive functions plus all predicate minimization functions over Σ-total and recursive predicates.

We define $R^\Sigma = \bigcup_{k \geq 0} R_k^\Sigma$.

Theorem 21 *If* $f \in R^\Sigma$ *then* f *is* Σ*-effectively computable.*

Theorem 22 *Not every* Σ*-recursive function is* Σ*-p.r. In other words,*

$$PR^\Sigma \subseteq R^\Sigma \text{ but } PR^\Sigma \neq R^\Sigma$$

It is obvious by definition that if f is Σ-p.r. then it is recursive. But if a function is recursive, it could very well be a minimization predicate over a Σ-total function that is not Σ-p.r. itself! In other words,

$$R^\Sigma - PR^\Sigma = \left\{M(P) : P \text{ is } \Sigma\text{-p.r.} \wedge P \in R^\Sigma \wedge M(P) \text{ is not } \Sigma\text{-p.r.}\right\}$$

In fact, the theorems in previous sections ensured that if P is Σ-p.r. and so is $\mathcal{D}_P$, then $M(P)$ is Σ-effectively computable. Which doesn't entail that it is Σ-p.r.

Theorem 23 *If $P \sim (n{+}1, m, \#)$ is a Σ-p.r. predicate then (1) $M(P)$ is Σ-recursive. If there is a Σ-p.r.function $f \sim (n, m, \#)$ s.t. $M(P)(\overrightarrow{x}, \overrightarrow{\alpha}) = \min_t P(t, \overrightarrow{x}, \overrightarrow{\alpha}) \leq f(\overrightarrow{x}, \overrightarrow{\alpha})$ for all $(\overrightarrow{x}, \overrightarrow{\alpha}) \in \mathcal{D}_{M(P)}$, then $M(P)$ is Σ-p.r.*

The theorem above gives the conditions to say whether $M(P)$ is recursive and whether it is Σ-p.r. It is recursive simply if P is Σ-p.r. And it is Σ-p.r. if $M(P)$ is bounded by some function f for all values in the domain of $M(P)$.

Theorem 24 *The quotient function, the remainder function, and the ith prime function are Σ-p.r.*

5.15 Minimization of alphabetic variable

We define $M^{\leq}(P) = \lambda \overrightarrow{x}\, \overrightarrow{\alpha} \left[\min_{\alpha}^{\leq} P(\overrightarrow{x}, \overrightarrow{\alpha}, \alpha)\right]$, where $\leq$ is some order over the language Σ in question.

Theorem 25 *If P is Σ-p.r. predicate over a string, then the same conditions apply for $M(P)$ to be Σ-p.r. as in the theorem for predicates over numbers.*

> **Problem 32** *Prove that $\lambda\alpha[\sqrt{\alpha}]$ is Σ-p.r.*
>
> Observe that $\lambda\alpha \left[\sqrt{\alpha}\right] = \min_{\alpha} \lambda\alpha\beta[\beta = \alpha\alpha]$. The predicate, which we call P, is trivially Σ-p.r. This means that $\lambda\alpha[\sqrt{\alpha}] \in R^{\Sigma}$.
>
> Let $M(P)$ denote the minimization above. Then $M\,(P\,(\alpha, \beta)) \leq \beta$. In other words, $M(P)$ is bounded by $f = \lambda\alpha[\alpha]$. Then $\lambda\alpha[\sqrt{\alpha}] \in PR^{\Sigma}$.

5.16 Enumerable sets

We say $S \subseteq \omega^n \times \Sigma^{*2}$ is Σ-recursively enumerable if it is empty or there is a function $\mathcal{F} : \omega \to \omega^n \times \Sigma^{*2}$ s.t.

- $Im_{\mathcal{F}} = S$
- $\mathcal{F}_{(i)}$ is Σ-recursive for every $1 \leq 1 \leq n + m$.

Here, Σ-recursive functions model Σ-computable functions.

5.17 Recursive sets

The Godelian model of a Σ-effectively computable set is simple. A set S is Σ-recursive when χ_S is Σ-recursive.

5.18 Alphabet independence

Theorem 26 *Let* Σ, Γ *two alphabets. If* f *is* Σ*-mixed and* Γ*-mixed, then* f *is* Σ*-recursive iff it is* Γ*-recursive. The analogue applies to recursive sets and this extends to primitive recursion.*

The theorem above states that recursiveness or primitive-recursiveness is independent of any given alphabet.

6 Neumann

6.1 The $\mathcal{S}^\Sigma$ language

We provide von Neumann's model of Σ-effectively computable function. We use $Num = \{0, 1, \ldots, 9\}$ a set of *symbols* (not numbers) and define $S : Num^* \mapsto Num^*$ as

$$\begin{aligned} S(\varepsilon) &= 1 \\ S(\alpha 0) &= \alpha 1 \\ S(\alpha 2) &= \alpha 3 \\ &\vdots \\ S(\alpha 9) &= S(\alpha)0 \end{aligned}$$

It is easy to observe that S is a "counting" or "enumerating" function of the alphabet Num. We define

$$\begin{aligned} \text{—} : \omega &\mapsto Num^* \\ \overline{0} &\mapsto \varepsilon \\ \overline{n+1} &\mapsto S(\overline{n}) \end{aligned}$$

In other words, $\overline{n}$ simply denotes the alphabetic symbol of Num that denotes the number n. The whole syntax of the S^Σ language is given by $\Sigma \cup \Sigma_p$, where

$$\Sigma_p = Num \cup \{\leftarrow, +, =, \ .\ , \neq, ^\frown, \varepsilon, N, K, P, L, I, F, G, O, T, B, E, S\}$$

It is important to note that these are *symbols* or *strings*, not values. The ε in Σ_p is not the empty letter, but the symbol that denotes it. The $\overline{+}, -$ signs are not the operations plus and minus, but the same symbols that denote these operations.

6.2 Variables, labels, and instructions

Any word of the form $N\overline{k}$ is a numeric variable; $P\overline{k}$ is an alphabetic variable; $L\overline{k}$ is a label.

The basic instructions in $\mathcal{S}^\Sigma$ make use of these; for a list of the instructions, consult the original source. In general, an instruction of $\mathcal{S}^\Sigma$ is any word of the form αI, where $\alpha \in \{L\overline{n} : n \in \mathbb{N}\}$ and I is a basic instruction. We use Ins^Σ to denote the set of all instructions in $\mathcal{S}^\Sigma$. When $I = L\overline{n}J$ and J a basic instruction, we say $L\overline{n}$ is the label of J.

6.3 Programs in $\mathcal{S}^\Sigma$

A program in $\mathcal{S}^\Sigma$ is any word $I_1 \dots I_n$, with $n \geq 1$, s.t. $I_k \in Ins^\Sigma$ for all $1 \leq k \leq n$ and the following property holds:

GOTO Law: For every $1 \leq i \leq n$, if $GOTOL\overline{m}$ is the end of Ii, then there is some $j, 1 \leq j \leq n$, s.t. I_j has label $L\overline{m}$.

Informally, a program is any chain of instructions satisfying that GOTO instructions map to actual labels in the program.

We use Pro^Σ to denote the set of all programs in $\mathcal{S}^\Sigma$.

Theorem 27 *Let Σ a finite alphabet. Then*

- *If $I_1 \dots I_n = J_1 \dots J_n$, with $I_k, J_k \in Ins^\Sigma$, then $n = m$ and $I_k = J_k$ for all k.*
- *If $\mathcal{P} \in Pro^\Sigma$ then there is a unique set of instructions $I_1 \dots I_n$ s.t. $\mathcal{P} = I_n \dots I_n$.*

The theorem above establishes that any program in Pro^Σ is a *unique* concatenation of instructions. We use $n(\mathcal{P})$ to denote the number of instructions that make up $\mathcal{P} \in Pro^\Sigma$. By convention, if $\mathcal{P} = I_1^\mathcal{P} \dots I_{n(\mathcal{P})}^\mathcal{P}$, then $I_j^\mathcal{P} = \varepsilon$ if $j \notin [1, n(\mathcal{P})]$. In other words, we understand that a program contains infinitely many empty symbols to the right and left (like in Turing machines).

Observation. $n(\alpha)$ and I_j^α are defined only when $\alpha \in Pro^\Sigma, i \in \omega$. This means the domain of $\lambda\alpha[n(\alpha)]$ is $Pro^\Sigma \subseteq \Sigma \cup \Sigma_p$ and that of $\lambda i\alpha[I_i^\alpha]$ is $\omega \times Pro^\Sigma$.

Problem 33 *Is is true that $Ins^\Sigma \cap Pro^\Sigma = \emptyset$? And is it true that $\lambda i\mathcal{P}[I_i^\mathcal{P}]$ has domain $\left\{(i, \mathcal{P}) \in \mathbb{N} \times Pro^\Sigma : i \leq n(\mathcal{P})\right\}$?*

Both statements are false. A single instruction in Ins^Σ can be a program (as long as it is not a GOTO statement to a non-existent label). Furthermore, $\lambda i\mathcal{P}[I_i^\mathcal{P}]$ is defined for $i = 0$ (it maps to ε) and for $i \geq n(\mathcal{P})$ (it also maps to ε).

Problem 34 *Prove: If $\mathcal{P}_1, \mathcal{P}_2 \in Pro^\Sigma$ then $\mathcal{P}_1\mathcal{P}_1 = \mathcal{P}_2\mathcal{P}_2 \Rightarrow \mathcal{P}_1 = \mathcal{P}_2$.*

This follows from the theorem that guarantees that any program $\mathcal{P} \in Pro^\Sigma$ is a *unique* concatenation of instructions. Let $\mathcal{P}_1 = I_1^{\mathcal{P}_1} \dots I_{n(\mathcal{P}_1)}^{\mathcal{P}_1}$ and $\mathcal{P}_2 = I_1^{\mathcal{P}_2} \dots I_{n(\mathcal{P}_2)}^{\mathcal{P}_2}$. Assume $\mathcal{P}_1\mathcal{P}_1 = \mathcal{P}_2\mathcal{P}_2$. Then

$$I_1^{\mathcal{P}_1} \dots I_{n(\mathcal{P}_1)}^{\mathcal{P}_1} I_1^{\mathcal{P}_1} \dots I_{n(\mathcal{P}_1)}^{\mathcal{P}_1} = I_2^{\mathcal{P}_2} \dots I_{n(\mathcal{P}_2)}^{\mathcal{P}_2} I_2^{\mathcal{P}_2} \dots I_{n(\mathcal{P}_2)}^{\mathcal{P}_2}$$

Then, from the last theorem follows that $I_k^{\mathcal{P}_1} = i_k^{\mathcal{P}_2}$. From this follows directly that $\mathcal{P}_1 = \mathcal{P}_2$. ■

6.4 States in programs of $\mathcal{S}^{\Sigma}$

.

We define $Bas : Ins^{\Sigma} \mapsto (\Sigma \cup \Sigma_p)^*$, the program that returns the substring of an instruction corresponding to its basic instruction, as

$$Bas(I) = \begin{cases} J & I = L\overline{k}J \\ I & \text{otherwise} \end{cases}$$

Recall that

$$\sim\alpha = \begin{cases} [\alpha]_2 \dots \alpha_{|\alpha|} & |\alpha| \geq 2 \\ \varepsilon & \text{otherwise} \end{cases}$$

We define $\omega^{\mathbb{N}} = \{(s_1, s_2, \dots) : \exists n \in \mathbb{N} : i > n \Rightarrow s_i = 0\}$. This is, $\omega^{\mathbb{N}}$ denotes the set of infinite tuples that from some index onwards contain only zeroes. Similarly, $\Sigma^{*\mathbb{N}}$ denotes the set of infinite alphabetic tuples that contain only ε from some index onwards.

A **state** is a tuple $(\overrightarrow{s}, \overrightarrow{\sigma}) \in \omega^{\mathbb{N}} \times \Sigma^{*\mathbb{N}}$. If $i \geq i$ we say s_i has the value of the $N\bar{i}$ variable in the state, and σ_i the value of the $P\bar{i}$ variable in the state. Thus, a state is a pair of infinite tuples containing the values of the variables in a program.

We use

$$[[x_1, \dots x_n,\ \alpha_1, \dots, \alpha_m]]$$

to denote the state $((x_1, \dots, x_n, 0, 0, \dots), (\alpha_1, \dots, \alpha_m, \varepsilon, \varepsilon, \dots))$.

6.5 Instantaneous description of a program in $\mathcal{S}^{\Sigma}$

Since a program $\mathcal{P} \in Pro^{\Sigma}$ may contain GOTO instructions, it is not always the case that $I^{\mathcal{P}}_{k+1}$ is executed after $I^{\mathcal{P}}_{k}$. Thus, when running a program, we not only need to consider its state but the specific instruction to be executed. An instantaneous description is a mathematical object which describes all this information.

Formally, an instantaneous description is triple $(i, \overrightarrow{s}, \overrightarrow{\alpha}) \in \omega \times \omega^{\mathbb{N}} \times \Sigma^{*\mathbb{N}}$. This Cartesian product is the set of all possible instantaneous descriptions. The triple reads: The following instruction is $I^{\mathcal{P}}_{i}$ and the current state is $(\overrightarrow{s}, \overrightarrow{\sigma})$. Observe that if $i \notin [1, n(\mathcal{P})]$, then the description reads: We are in state $(\overrightarrow{s}, \overrightarrow{\sigma})$ and we must execute ε (nothing).

We define the successor function

$$S_{\mathcal{P}} : \omega \times \omega^{\mathbb{N}} \times \Sigma^{*\mathbb{N}} \mapsto \omega \times \omega^{\mathbb{N}} \times \Sigma^{*\mathbb{N}}$$

which maps an instantaneous description to the successor instantaneous description (the one after executing the instruction in the first).

6.6 Computation from a given state

Let $\mathcal{P} \in Pro^{\Sigma}$ and a state $(\overrightarrow{s}, \overrightarrow{\sigma})$. The *computation* of $\mathcal{P}$ from $(\overrightarrow{s}, \overrightarrow{\sigma})$ is defined as

$$\left((1, \overrightarrow{\sigma}, \overrightarrow{\sigma}), S_{\mathcal{P}}\left(1, \overrightarrow{s}, \overrightarrow{\sigma}\right), S_{\mathcal{P}}\left(S_{\mathcal{P}}\left(1, \overrightarrow{s}, \overrightarrow{\sigma}\right)\right), \ldots\right)$$

In other words, the *computation* of $\mathcal{P}$ is the infinite tuple whose ith element is the instantaneous description of $\mathcal{P}$ after $i - 1$ instructions have been executed.

We say $S_{\mathcal{P}}\left(\ldots S_{\mathcal{P}}\left(S_{\mathcal{P}}\left(1, \overrightarrow{s}, \overrightarrow{\sigma}\right)\right)\right)$ is the instantaneous description obtained after t steps if the number of times $S_{\mathcal{P}}$ was executed is t.

Problem 35 *Give true or false for the following statements.*

Statement 1: If $S_{\mathcal{P}}(i, \overrightarrow{s}, \overrightarrow{\alpha}) = (i, \overrightarrow{s}, \overrightarrow{\alpha})$ *then* $i \notin [1, n(\mathcal{P})]$. The statement is false. It could be the case that $i \notin [1, n(\mathcal{P})]$, in which case we would say the program halted. However, consider the program

$$L1\ GOTO\ L1$$

Evidently, $S_{\mathcal{P}}(1, \overrightarrow{s}, \overrightarrow{\alpha}) = (1, \overrightarrow{s}, \overrightarrow{\alpha})$, and $1 \leq 1 \leq n(\mathcal{P})$.

Statement 2. Let $\mathcal{P} \in Pro^{\Sigma}$ *and* d *an instantaneous description whose first coordinate is* i. *If* $I_i^{\mathcal{P}} = N_2 \leftarrow N_2 + 1$, *then*

$$S_{\mathcal{P}}(d) = (i + 1, (N_1, Suc(N_2), N_3, \ldots), (P_1, P_2, P_3, \ldots))$$

The statement is true via direct application of the $S_{\mathcal{P}}$ function.

Statement 3. Let $\mathcal{P} \in Pro^{\Sigma}$ *and* $(i, \overrightarrow{s}, \overrightarrow{\sigma})$ *an instantaneous description. If* $Bas(I_i^{\mathcal{P}}) = IF\ P_3\ BEGINS\ a\ GOTO\ L_6$ *and* $[P_3]_1 = a$, *then* $S_{\mathcal{P}}(i, \overrightarrow{s}, \overrightarrow{\sigma}) = (j, \overrightarrow{s}, \overrightarrow{\sigma})$, *where* j *is the least number* l *s.t.* $I_l^{\mathcal{P}}$ *has label* L_6.

Because $[P_3]_1 = a$, the value of $S_{\mathcal{P}}(i, \overrightarrow{s}, \overrightarrow{\sigma})$ must indeed contain the instruction that has label L_6. This instruction is the jth instruction for some j, etc. The statement is true.

6.7 Halting

When the first coordinate of $S_{\mathcal{P}}\left(\ldots S_{\mathcal{P}}\left(S_{\mathcal{P}}\left(1,\overrightarrow{s},\overrightarrow{\sigma}\right)\right)\right)$ with t steps is $n(\mathcal{P})+1$, we say $\mathcal{P}$ *halts after t steps when starting from* $(\overrightarrow{s},\overrightarrow{\sigma})$.

If none of the first coordinates in the computation of $\mathcal{P}$,

$$\left((1,\overrightarrow{\sigma},\overrightarrow{\sigma}),S_{\mathcal{P}}\left(1,\overrightarrow{s},\overrightarrow{\sigma}\right),S_{\mathcal{P}}\left(S_{\mathcal{P}}\left(1,\overrightarrow{s},\overrightarrow{\sigma}\right)\right),\ldots\right)$$

is $n(\mathcal{P})$, we say $\mathcal{P}$ does not halt starting from $(\overrightarrow{s},\overrightarrow{\sigma})$.

6.8 Σ-computable functions

We give the model of a Σ-effectively computable function in the paradigm of von Neumann. Intuitively, f is Σ-computable if there is some $\mathcal{P}\in Pro^{\Sigma}$ that computes it.

Given $\mathcal{P}\in Pro^{\Sigma}$, for every pair $n,m\geq 0$, we define $\Psi_{\mathcal{P}}^{n,m,\#}$ as follows:

$$\mathcal{D}_{\Psi_{\mathcal{P}}^{n,m,\#}}=\left\{(\overrightarrow{x},\overrightarrow{\alpha})\in\omega^n\times\Sigma^{*m}:\mathcal{P}\text{ halts from }[[x_1,\ldots,x_n,\alpha_1,\ldots,\alpha_m]]\right\}$$

$$\Psi_{\mathcal{P}}^{n,m,\#}(\overrightarrow{x},\overrightarrow{\alpha})=\text{Value of }N_1\text{ in halting state from }[[x_1,\ldots,x_n,\alpha_1,\ldots,\alpha_m]]$$

We analogously define $\Psi_{\mathcal{P}}^{n,m,*}$ for the alphabetic case, where the domain is the same and the value is that of P_1 in the halting state.

A Σ-mixed function, not necessarily total, is Σ-computable if there is a program $\mathcal{P}\in Pro^{\Sigma}$ s.t. $f\sim(n,m,\varphi)=\Psi_{\mathcal{P}}^{n,m,\varphi}$, with $\varphi\in\{\#,*\}$. We say f is computed by $\mathcal{P}$.

Theorem 28 *If f is Σ-computable, then it is Σ-effectively computable.*

The previous theorem should be obvious. Any program in $\mathcal{S}^{\Sigma}$ can be translated into an effective procedure with relative simplicity.

> **Problem 36** *Let $\Sigma=\{@,!\}$. Give a program that computes $f:\{0,1,2\}\mapsto\omega$ given by $f(0)=f(1)=0, f(2)=5$.*
>
> Evidently $f\sim(1,0,\#)$ and so we must find some $\mathcal{P}\in Pro^{\Sigma}$ s.t. $\Psi_{\mathcal{P}}^{1,0,\#}(x)=f(x)$. The program must let N_1 hold the value 0 if the starting state is either $[[0]]$ or $[[1]]$, and the value 5 if the starting state is $[[2]]$. In all other cases, it must not halt, to ensure that the domain of $\Psi_{\mathcal{P}}^{1,0,\#}$ is the same as that of f. The desired program is

$$
\begin{array}{ll}
 & N_2 \leftarrow N_1 \\
 & N_2 \leftarrow N_2 - 1 \\
 & IF\ N_2 \neq 0\ GOTO\ L_1 \\
 & GOTO\ L_4 \\
L_1 & N_2 \leftarrow N_2 - 1 \\
 & IF\ N_2 \neq 0\ GOTO\ L_2 \\
 & GOTOL_3 \\
L_2 & GOTO\ L_2 \\
L_3 & N_1 \leftarrow N_1 + 1 \\
 & N_1 \leftarrow N_1 + 1 \\
 & N_1 \leftarrow N_1 + 1 \\
 & GOTO\ L_5 \\
L_4 & N_1 \leftarrow 0 \\
L_5 & SKIP
\end{array}
$$

If $\mathcal{P}$ denotes this program, it is evident that $\mathcal{P}$ only halts for starting states $[[x_1]]$ with $x_1 \in \{0, 1, 2\}$. Thus, the domain of $\Psi_{\mathcal{P}}^{1,0,\#}$ is precisely $\mathcal{D}_f$. It is easy to verify that, more generally, $\Psi_{\mathcal{P}}^{1,0,\#} = f$.

Problem 37 *Using the same alphabet as in the previous problem, find* $\mathcal{P} \in Pro^{\Sigma}$ *that computes* $\lambda xy[x + y]$.

The desired program is

$$
\begin{array}{ll}
L_1 & IF\ N_2 = 0\ GOTO\ L_3 \\
 & N_1 \leftarrow N_1 + 1 \\
 & N_2 \leftarrow N_2 - 1 \\
 & GOTO\ L_1 \\
L_3 & SKIP
\end{array}
$$

Problem 38 *Same for* $C_0^{1,1}{}_{|\{0,1\}\times\Sigma^*}$

Since the domain of the constant function is restricted t o $\{0, 1\} \times \Sigma^*$, we must ensure the program only halts for states $[[x_1, x_2, \alpha]]$ s.t. $x_1, x_2 \in \{0, 1\}$. Thus, the program is

$$
\begin{aligned}
& N_1 \leftarrow N_1 - 1 \\
& N_2 \leftarrow N_2 - 1 \\
& IF N_2 \neq 0 \; GOTO \; L_1 \\
& IF N_1 \neq 0 \; GOTO \; L_1 \\
& GOTO \; L_2 \\
L_1 \; & GOTO \; L_1 \\
L_2 \; & SKIP
\end{aligned}
$$

Problem 39 *Same for $\lambda i \alpha [[\alpha]_i]$ (same alphabet).*

$$
\begin{aligned}
& IF \; N_0 \neq 0 \; GOTO \; L_1 \\
& P_1 \leftarrow \varepsilon \\
& GOTO \; L_{100} \\
L_1 \; & N_1 \leftarrow N_1 - 1 \\
L_2 \; & N_1 \leftarrow N_1 - 1 \\
& P_1 \leftarrow {}^{\frown}P_1 \\
& IF \; N_1 \neq 0 \; GOTO \; L_2 \\
& IF \; P_1 \; STARTSWITH \; @ \; GOTO \; L_2 \\
& IF \; P_1 \; STARTSWITH \; ! \; GOTOL_3 \\
& GOTOL_{100} \\
L_3 \; & P_1 \leftarrow \; ! \\
L_2 \; & P_1 \leftarrow @ \\
L_{100} \; & SKIP
\end{aligned}
$$

Example. Let $\alpha = @!!@@$. Assume we give $[[4, \alpha]]$. Since $4 \neq 0$ we go to L_1 immediately. Here N_1 is set to three. Then N_1 is set to two and P_1 is set to $!!@@$. Since $N_1 \neq 0$, N_1 is now set to 1 and P_1 to $!@@$. Once more, N_1 is now set to 0 and P_1 to $@@$. Since now $N_1 = 0$, we know the starting character of P_1 is the one we looked for. We set P_1 to be its first character (if $P_1 = \varepsilon$ it has no first character and nothings needs to be done, because this means the input $[[x_1, \alpha]]$ had $x_1 > |\alpha|$). The other cases also work.

Problem 40 *Give a program that computes $s^{\leq}$ where $@ <!$.*

Recall that $s^{\leq} : \Sigma^* \mapsto \Sigma^*$ is defined as

$$s^{\leq}((a_n)^m) = (a_1)^{m+1} \qquad m \geq 0$$
$$s^{\leq}(\alpha a_i (a_n)^m) = \alpha a_{i+1}(a_1)^m \qquad 1 \leq i < n, m \geq 0$$

In our case, this functions enumerates the language in question as follows:

$$\varepsilon, @, !, @@, @!, !@, !!, @@@, @@!, @!@, @!!, !@@, !@!, !!@, !!!, \ldots$$

6.9 Macros

A macro is the template of a program that computes a Σ-mixed function. There are two types:

- Those that assign that simulate setting the value of a variable to a function of others;
- Those that use IF statements that direct a program to a label if a predicate function of other variables is true.

A macro is not a program because it does not necessarily hold to **GOTO law**. The formal definition of a macro is hand-wavy and long; check the source. The variables of a macro that are only used within the macro are the *auxiliary variables*. The variables the receive the input (from within some program) are the *official variables*.

Theorem 29 *Let Σ a finite alphabet. Then if f a Σ-computable function, there is a macro $\left[Z\overline{n+1} \leftarrow f\,(V_1, \ldots, V\overline{n}, W_1, \ldots, W\overline{m})\right]$ with $Z \in \{V, W\}$ depending on the value of f.*

Example. The function $\mathcal{F} = \lambda xy[x + y]$ is Σ-computable. Then there is a macro that computes it. Such macro is:

$$\begin{aligned}
&V_4 \leftarrow V_2\\
&V_5 \leftarrow V_3\\
&V_1 \leftarrow V_4\\
A_1\ &IF\ V_5 \neq 0\ GOTO\ A_2\\
&GOTO\ A_3\\
A_2\ &V_5 \leftarrow V_5 - 1\\
&V_1 \leftarrow V_1 + 1\\
&GOTO\ A_1\\
A_3\ &SKIP
\end{aligned}$$

We replace V_1 with that variable where the output is to be stored, V_2, V_3 with the variables the are to be summed, and this performs the sum of two variables. Now, to program $\lambda xy[x \cdot y]$ we can use the following:

$$\begin{array}{ll} L_1 & IF\ N_2 \neq 0\ GOTO\ L_2 \\ & GOTO\ L_3 \\ L_2 & [N_3 \leftarrow \mathcal{F}(N_3, N_1)] \\ & N_2 \leftarrow N_2 - 1 \\ & GOTO\ L_1 \\ L_3 & N_1 \leftarrow N_3 \end{array}$$

Problem 41 *Let* $\Sigma = \{@, !\}$ *and* $f \sim (0, 1, \#)$ *a* Σ*-computable function. Let* $L = \left\{\alpha \in \mathcal{D}_f : f(\alpha) = 1\right\}$. *Using the macro* $[V_1 \leftarrow f(W_1)]$, *give a program* $\mathcal{P} \in Pro^{\Sigma}$ *s.t.* $\mathcal{D}_{\Psi_{\mathcal{P}}^{0,1,\#}} = L$.

$\mathcal{D}_{\Psi_{\mathcal{P}}^{0,1,\#}} = L$ if and only if $\mathcal{P}$ halts only when starting from a state $[[\alpha \in L]]$ Such $\mathcal{P}$ may be

$$\begin{array}{ll} & [N_1 \leftarrow f(P_1)] \\ & IF\ N_1 \neq 0\ GOTO\ L_1 \\ & GOTO\ L_2 \\ L_1 & GOTO\ L_1 \\ L_2 & SKIP \end{array}$$

Incidentally, it is easy to observe that $\Psi_{\mathcal{P}}^{0,1,\#} = f_{|L}$.

Problem 42 *Let* $\Sigma = \{@, !\}$ *and* $f \sim (1, 0, *)$ *a* Σ*-computable function. Using* $[W_1 \leftarrow f(V_1)]$, *give a program* $\mathcal{P} \in Pro^{\Sigma}$ *s.t.* $\mathcal{D}_{\Psi_{\mathcal{P}}^{1,0,*}} = Im_f$.

We require a program $\mathcal{P} \in Pro^{\Sigma}$ s.t. $\mathcal{P}$ halts only from a starting state of the form $[[\alpha \in Im_f]]$. Such a program may be

$$\begin{array}{ll} L_1 & [P_2 \leftarrow f(N_1)] \\ & [IF\ P_1 = P_2\ GOTO\ L_2] \\ & N_1 \leftarrow N_1 + 1 \\ & GOTO\ L_1 \\ L_2 & Skip \end{array}$$

where $[IF\ W_1 = W_2\ GOTO\ A_1]$ is the macro

$$
\begin{array}{lll}
 & W_3 \leftarrow W_1 & \\
 & W_4 \leftarrow W_2 & \\
A_1 & IF\ W_3 BEGINS\ @\ GOTO\ A_2 & \\
 & IF\ W_3 BEGINS\ !\ GOTO\ A_3 & \\
 & A_2 & IF\ W_4\ BEGINS\ @\ GOTO\ A_4 \\
 & GOTO\ A_{1000} & \\
A_3 & IF\ W_4\ BEGINS\ !\ GOTO\ A_4 & \\
A_4 & W_3 \leftarrow {}^\frown W_3 & \\
 & W_4 \leftarrow {}^\frown W_4 & \\
 & GOTO A_5 & \\
A_{1000} & SKIP &
\end{array}
$$

that checks if two *not-empty* strings are equal and jumps to the official label A_5 if the case is true.

6.10 Enumerable sets

A non-empty Σ-mixed set S is Σ-enumerable if and only if there are programs $\mathcal{P}_1, \ldots, \mathcal{P}_{n+m}$ s.t.

$$\mathcal{D}_{\Psi^{n,m,\#}_{\mathcal{P}_1}} = \ldots = \mathcal{D}_{\Psi^{n,m,\#}_{\mathcal{P}_n}} = \omega$$
$$\mathcal{D}_{\Psi^{n,m,*}_{\mathcal{P}_{n+1}}} = \ldots = \mathcal{D}_{\Psi^{n,m,*}_{\mathcal{P}_{n+m}}} = \omega$$

and

$$S = Im\ \left[\Psi^{n,m,\#}_{\mathcal{P}_1}, \ldots, \Psi^{n,m,\#}_{\mathcal{P}_n}, \Psi^{n,m,*}_{\mathcal{P}_{n+1}}, \ldots, \Psi^{n,m,*}_{\mathcal{P}_{n+m}}\right]$$

In other words, for each input $x \in \omega$, the ith program $\mathcal{P}_i$ computes the value of the ith element in a tuple of S. Another way to put this is

Theorem 30 *If S a non-empty Σ-mixed set, then it is equivalent to say:*

(1) S is Σ-enumerable.

(2) There is a $\mathcal{P} \in Pro^\Sigma$ satisfying the following two properties. a. For all $x \in \omega$, $\mathcal{P}$ halts from $[[x]]$ into a state of the form $[[x_1, \ldots, x_n, \alpha_1, \ldots,]]$ when $(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_n) \in S$. b. For any tuple $(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m) \in S$, there is a $x \in \omega$ s.t. $\mathcal{P}$ halts starting from $[[x]]$ in a state of the form $[[x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m]]$

When a program satisfies these properties, we say it *enumerates* S.

6.11 Σ-computable sets

A Σ-mixed set S is said to be Σ-computable if $\chi_S^{\omega^n \times \Sigma^{*m}}$ is Σ-computable. This is, S is Σ-computable if and only if there is a $\mathcal{P} \in Pro^{\Sigma}$ s.t. $\mathcal{P}$ commputes $\chi_S^{\omega^n \times \Sigma^{*m}}$.

Observe that this means that $\mathcal{P}$ halts with $N_1 = 1$ when starting from $[[\overrightarrow{x}, \overrightarrow{\alpha}]]$ if $(\overrightarrow{x}, \overrightarrow{\alpha}) \in S$, and halts with $N_1 = 0$ otherwise. We say $\mathcal{P}$ *decides* the belonging to S.

Observe that if $\chi_S^{\omega^n \times \Sigma^{*m}}$ is Σ-computable, then there is a macro

$$\left[IF\ \chi_S^{\omega^n \times \Sigma^{*m}}\ (V_1 \ldots, V\overline{n}, W_1, \ldots, W\overline{m})\ GOTO\ A_1 \right]$$

We will write this macro as $[IF\ (V_1, \ldots V\overline{n}, W_1, \ldots, W\overline{m}) \in S\ GOTO\ A_1]$. Of course, this macro is only valid when S is a Σ-computable set.

Theorem 31 *In Godel's paradigm, S is Σ-p.r. iff it is the domain of a Σ-p.r. function. This statement does not hold in von Neumman's paradigm. There are sets that are domains of Σ-computable functions that are not Σ-computable themselves.*

7 Paradigm battles

7.1 Neumann triumphs over Godel

Theorem 32 *If h is Σ-recursive then it is Σ-computable.*

A corollary is that every Σ-recursive function has a corresponding macro.

Theorem 33 *If $S_1, S_2 \subseteq \omega^n \times \Sigma^{*m}$ are Σ-enumerable, then $S_1 \cup S_2$ is Σ-enumerable.*

Proof. Via assumption there are $F : \omega \mapsto \omega^n \times \Sigma^{*m}$, $G : \omega \mapsto \omega^n \times \Sigma^{*m}$ s.t. $F_{(i)}, G_{(i)}$ are Σ-computable for every i and $Im_F = S_1$, $Im_G = S_2$. $\therefore$ $F_{(i)}, G_{(i)}$ have associated macros.

λx [x is even] is Σ-p.r. $\therefore$ it is Σ-computable. $\therefore$ λx [x is even] has an associated macro. $\therefore$ There is an IF macro which checks if a variable holds an even number.

λx $[x/2]$ is Σ-p.r. $\therefore$ it is Σ-p.r. $\therefore$ it has an associated macro.

Using the macros above, one can write a program $\mathcal{P} \in Pro^{\Sigma}$ that does the following:

> If the input N_1 is even, set N_1 to its integer quotient by two and then let $N_1 = F_1(N_1), \ldots, N_n = F_n(N_n)$.
> If the input N_1 is odd, set N_1 to its integer quotient by two minus one and then let $N_1 = G_1(N_1), \ldots, N_n = G_n(N_n)$.

It is easy to see that this satisfies the theorem of the **Enumerable sets** section of the von Neumann paradigm.

Theorem 34 *If $S \subseteq \omega^n \times \Sigma^{*m}$ is Σ-computable, then S is enumerable.*

Proof. Assume $S \subseteq \omega^n \times \Sigma^{*m}$ is Σ-computable. $\therefore \underset{S}{}^{n,m}$ is Σ-recursive and has an associated macro.

Recall that $pr : \mathbb{N} \mapsto \omega$ is Σ-p.r. $\therefore$ pr is Σ-recursive and has an associated macro. The same is true of $\leq^*$, since its definition involves only string concatenations and string exponentiations (which are Σ-p.r.) on two mutually exclusive cases.

Let

$$\mathcal{F} : [1, m] \times \mathbb{N} \mapsto \omega \cup \Sigma^*$$
$$(x, i) \mapsto \begin{cases} (x)_i & 1 \leq i \leq n \\ *^{\leq}((x)_i) & n+1 \leq i \leq m \end{cases}$$

The program $\mathcal{P}_i$ with input x, i defined as

$$
\begin{array}{rl}
 & [N_2 \leftarrow pr(N_2)] \\
L_1 & [IF \neg (N_2 \mid N_1)\ GOTO\ L_2] \\
 & N_3 \leftarrow N_3 + 1 \\
 & [N_2 \leftarrow N_2 \times N_2] \\
 & GOTO\ L_1 \\
L_2 & [IF 1\ \leq N_1 \leq m\ GOTO\ L_3] \\
 & [N_1 \leftarrow *^{\leq}(N_3)] \\
 & GOTO\ L_{100} \\
L_3 & N_1 \leftarrow N_3 \\
L_{100} & SKIP
\end{array}
$$

when $n + 1 \leq i \leq m$ computes $\mathcal{F}$. $\therefore$ There is a macro for $\mathcal{F}$.

Let $u = n + m + 1$ and $\mathcal{P}$ the following program:

$$
\begin{array}{rl}
 & N\overline{u} \leftarrow N_1 \\
 & [N_1 \leftarrow \mathcal{F}(N\overline{u}, 1)] \\
 & [N_2 \leftarrow \mathcal{F}(N\overline{u}, 2)] \\
 & \vdots \\
 & [P_{\overline{m}} \leftarrow \mathcal{F}(N\overline{u}, n + m)] \\
 & [IF\ \chi_s^{n,m}\ (N_1, \ldots, N_{\overline{n}}, P_{\overline{m}},)\ \ GOTO\ L_{100}] \\
L_0 & GOTO\ L_0 \\
L_{100} & SKIP
\end{array}
$$

a. For any $x \in \omega$, $\mathcal{P}$ halts at a state with instantaneous description $[[x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_m]]$ with $(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_n) \in S$; *b.* for any $(\vec{x}, \vec{\alpha}) \in S$ there is some $x \in \omega$ s.t. $\mathcal{P}$ halts at $[[(\vec{x}, \vec{\alpha})]]$ when starting from x. $\therefore \mathcal{P}$ enumerates S.

Problem 43 *Prove that, if $S \subseteq \Sigma^*$ is Σ-enumerable, then*

$$T = \{\alpha \in \Sigma^* : \exists \beta \in S : \alpha \text{ is subchain of } \beta\}$$

is Σ-enumerable.

Let $\mathcal{F} : \omega \mapsto \Sigma^*$ denote the function which enumerates S, which is Σ-computable and has a macro. The key to the problem is to observe that, per each $\alpha \in S$, there are $|\alpha| - 1$ subchains of α in T. Thus, for $\mathcal{F}(i)$ we want to enumerate $|\mathcal{F}(i)| - 1$ subchains.

I provide a program that accomplishes this and then illustrate its operation on a finite alphabet. It is easy to verify that the program enumerates T and that this enumeration holds for infinite alphabets as well.

Let $\mathcal{P}$ with input $x \in \omega$ be

$$
\begin{array}{rl}
 & [P_1 \leftarrow \mathcal{F}(N_3)] \\
 & [N_{50} \leftarrow |\mathcal{F}(N_3)|] \\
L_1 & [IF\ N_1 \geq N_{50}\ GOTO\ L_2] \\
 & N_{10} \leftarrow N_{50} - N_1 \\
 & N_{10} \leftarrow N_{10} - 1 \\
L_{11} & [P_1 \leftarrow {}^{\frown}P_1] \\
 & N_{10} \leftarrow N_{10} - 1 \\
 & [IF\ N_{10} = 0\ GOTO\ L_{100}] \\
 & GOTO\ L_{11} \\
L_2 & N_3 \leftarrow N_3 + 1 \\
 & [P_1 \leftarrow \mathcal{F}(N_3)] \\
 & [N_{50} \leftarrow N_{50} + |P_1|] \\
 & GOTO\ L_1 \\
L_{100} & SKIP
\end{array}
$$

*Example.*Let $S = \{abab, bba, bbba, abb\}$. Then the program can be illustrated as follows:

$$
\begin{array}{l}
x = 0 \mid \overbrace{0 \leq 4}^{N_{50} \leq N_1} \mid N_{10} = 3 \mapsto a \\
x = 1 \mid 1 \leq 4 \mid N_{10} = 2 \mapsto ab \\
x = 2 \mid 2 \leq 4 \mid N_{10} = 1 \mapsto aba \\
x = 3 \mid 3 \leq 4 \mid N_{10} = 0 \mapsto aba
\end{array}
$$

When $x = 4$, we have $N_{50} \geq N_1$ and so P_1 becomes bba, N_{50} becomes $4 + 3 = 7$ and

$$
\begin{aligned}
x = 4 \mid \overbrace{4 \leq 7}^{N_{50} \leq N_1} \mid N_{10} = 2 \mapsto b \\
x = 5 \mid 5 \leq 7 \mid N_{10} = 1 \mapsto bb \\
x = 6 \mid 6 \leq 7 \mid N_{10} = 0 \mapsto bb
\end{aligned}
$$

When $x = 7$, the L_2 case is met twice, and so $P_1 = bbba, N_{50} = 4 + 3 + 4 = 11$ and

$$
\begin{aligned}
x = 7 \mid \overbrace{7 \leq 11}^{N_{50} \leq N_1} \mid N_{10} = 3 \mapsto b \\
x = 8 \mid 8 \leq 11 \mid N_{10} = 2 \mapsto bb \\
x = 9 \mid 9 \leq 11 \mid N_{10} = 1 \mapsto bbb \\
x = 10 \mid 10 \leq 11 \mid N_{10} = 0 \mapsto bbb
\end{aligned}
$$

etc. Thus, the program enumerates S and, per each $\alpha \in S$, it enumerates the substrings which make up α. The program is s.t.

$$
\begin{aligned}
&\left.\begin{aligned}
0 &\mapsto \textit{The smallest substring of } \mathcal{F}(0) \\
1 &\mapsto \textit{The next smallest substring of } \mathcal{F}(0) \\
2 &\mapsto \textit{The next smallest substring of } \mathcal{F}(0) \\
3 &\mapsto \textit{The next smallest of } \mathcal{F}(0)
\end{aligned}\right\} 0 \leq x \leq |\mathcal{F}(0)| \\
&\left.\begin{aligned}
4 &\mapsto \textit{The smallest substring of } \mathcal{F}(1) \\
5 &\mapsto \textit{The next smallest } \mathcal{F}(1) \\
6 &\mapsto \textit{The next smallest substring of } \mathcal{F}(1)
\end{aligned}\right\} |\mathcal{F}(0)| \leq x \leq |\mathcal{F}(0)\mathcal{F}(1)| \\
&\vdots
\end{aligned}
$$

7.2 Godel triumphs over Neumann

The following three functions contain all the information relevant to $\mathcal{S}^{\Sigma}$. Assuming $n, m \in \omega$ are fixed,

$$
\begin{aligned}
i^{n,m}\omega \times \omega^n \times \Sigma^{*m} x Pro^{\Sigma} &\mapsto \omega \\
E^{n,m}_{\#}\omega \times \omega^n \times \Sigma^{*m} x Pro^{\Sigma} &\mapsto \omega^{[\mathbb{N}]} \\
E^{n,m}_{*}\omega \times \omega^n \times \Sigma^{*m} x Pro^{\Sigma} &\mapsto \Sigma^{*[\mathbb{N}]}
\end{aligned}
$$

For brevity, let $f^{n,m}(t,\vec{x},\vec{a},\mathcal{P}) = f(t)$ for $f \in \left\{i^{n,m}, E^{n,m}_{\#}, E^{n,m}_{*}\right\}$—this is, let as assume fixed $\vec{x},\vec{\alpha},\mathcal{P}$. Then

$$\left(i^{n,m}(0), E^{n,m}_{\#}(0), E^{n,m}_{*}(0)\right) = \left(1, (x_1,\ldots,x_n,0,\ldots), (\alpha_1,\ldots,\alpha_n,0,\ldots)\right)$$
$$\left(i^{n,m}(t+1), E^{n,m}_{\#}(t+1), E^{n,m}_{*}(t+1)\right) = S_{\mathcal{P}}\left(i^{n,m}(t), E^{n,m}_{\#}(t), E^{n,m}_{*}(t)\right)$$

It is clear that $\left(i^{n,m}(t,\vec{x},\vec{\alpha},\mathcal{P}), E^{n,m}_{\#}(t,\vec{x},\vec{\alpha},\mathcal{P}), E^{n,m}_{*}(t,\vec{x},\vec{\alpha},\mathcal{P})\right)$ is the instantaneous description of $\mathcal{P}$ after t steps starting from $[[\vec{x},\vec{\alpha}]]$.

The E functions are not $(\Sigma\cup\Sigma_p)$-mixed, but if we define $E^{n,m}_{\varphi j}$, with $\varphi \in \{*,\#\}$, as the jth coordinate of $E^{n,m}_{\varphi}$, we find that $E^{n,m}_{\varphi j}$ is $(\Sigma \cup \Sigma_p)$-mixed.

Theorem 35 *The functions $i^{n,m}, E^{n,m}_{\varphi j}$ are $(\Sigma \cup \Sigma_p)$-mixed functions.*

Given $n, m \in \omega$, we define

$$Halt^{n,m} = \lambda t\vec{x}\vec{\alpha}\mathcal{P}\left[i^{n,m}(t,\vec{x},\vec{\alpha},\mathcal{P}) = n(\mathcal{P}) + 1\right]$$

Observe that $\mathcal{D}_{Halt} = \omega \times \omega^n \times \Sigma^{*m} \times Pro^{\Sigma}$.

Theorem 36 *Pro^{Σ} is a $(\Sigma\cup\Sigma_p)$-p.r. set and $n(\mathcal{P}), I^{\mathcal{P}}_j$ are $(\Sigma\cup\Sigma_p)$-p.r. functions.*

Theorem 37 *The Halt function is Σ-p.r.*

Proof. Observe that

$$Halt^{n,m} = \lambda xy\,[x = y] \circ \left[i^{n,m}, \lambda\mathcal{P}\,[n(\mathcal{P})] \circ p^{1+n,m+1}_{n+m+2}\right]$$

We define $T^{n,m} = M(Halt^{n,m})$. Observe that

$$\mathcal{D}^{n,m}_{T} = \{(\vec{x},\vec{\alpha},\mathcal{P}) \in \omega^n \times \Sigma^{*m} : \mathcal{P} \text{ halts when starting from } [[(\vec{x},\vec{\alpha})]]\}$$

Theorem 38 *$T^{n,m}$ is $(\Sigma \cup \Sigma_p)$-recursive but it is not $(\Sigma \cup \Sigma_p)$-p.r.*

The previous theorem follows from the fact that $T^{n,m}$ is a minimization over a Σ-p.r. predicate.

Theorem 39 *If f is a Σ-computable Σ-mixed function, then f is Σ-recursive.*

The previous theorem states that any function computable in von Neumman's paradigm is computable in Godel's paradigm.

Proof. The proof consists in showing that f is $(\Sigma \cup \Sigma_p)$-recursive. Then, due to the alphabet independence of recursion, it is Σ-recursive.

7.3 Interacting programs

Using the results of the previous section, we can construct programs that depend on other programs. This is illustrated with an example. Say $\Sigma\,\{@,+\}$ and $\mathcal{P}_0 \in Pro^{\Sigma}$ is s.t. $0 \in \mathcal{D}_{\Psi^{1,0,\#}_{\mathcal{P}_0}}$ with $\Psi^{1,0,\#}_{\mathcal{P}_0}(0) = 2$. We will show

$$S = \left\{ x \in \mathcal{D}_{\Psi^{1,0,\#}_{\mathcal{P}_0}} : \Psi^{1,0,\#}_{\mathcal{P}_0}(x) \neq 0 \right\}$$

is Σ-enumerable.

The program will do the following. For each input x, it will test whether $\mathcal{P}_0$ halts when starting from $[[(x)_1]]$ in $(x)_2$ steps. If it doesn't it will simply return zero (since $0 \in S$). If it does, it will compute $\mathcal{P}_0$ starting from $[[(x)_1]]$; if the output is not zero, it will return it. If it is zero, it will return 0.

Of course, the program will make use of the $Halt^{1,0}$ function, which is Σ-recursive and thus has an associated macro. It will also use $E^{1,0}_i$ to retrieve the output of $\mathcal{P}_0$ after $(x)_2$ steps. This function is also Σ-p.r. and thus it also has an associated macro. The program is

$$\begin{array}{ll}
 & IF\ N_1 \neq 0\ GOTO\ L_1 \\
 & GOTO\ L_2 \\
L_1 & [N_3 \leftarrow (N_1)_1] \\
 & [N_4 \leftarrow (N_1)_2] \\
 & [IF\ Halt^{1,0}(N_4, N_3, \mathcal{P}_0)\ GOTO\ L_3] \\
 & GOTO\ L_2 \\
L_3 & \left[N_5 \leftarrow E^{1,0}_1(N_4, N_3, \mathcal{P}_0)\right] \\
 & [IF\ N_5 = 0\ GOTO\ L_2] \\
 & N_1 \leftarrow N_5 \\
 & GOTO\ L_4 \\
L_2 & N_1 \leftarrow 0 \\
L_4 & SKIP
\end{array}$$

Thus, a program may use another program to operate.

Theorem 40 *If S a Σ-mixed set then these statements are equivalent:*

(1) S is Σ-enumerable

(2) $S = I_F$ for $F : \mathcal{D}_F \subseteq \omega^n \times \Sigma^{*m} \mapsto S$ s.t. $F_{(i)}$ is Σ-computable for each $i = 1, \ldots, n+m$.

(3) S is the domain of a Σ-computable function.

Proof. (1) $\Rightarrow$ (2) Assume S is Σ-enumerable. Then there are programs $\mathcal{P}_1, \ldots, \mathcal{P}_{n+m}$ s.t. S = Image of $\left[\Psi^{n,m,\#}_{\mathcal{P}_1}, \ldots, \Psi^{n,m,*}_{\mathcal{P}_{n+m}}\right]$. Let $f = \left[\Psi^{n,m,\#}_{\mathcal{P}_1}, \ldots, \Psi^{n,m,*}_{\mathcal{P}_{n+m}}\right]$. ■

(2) $\Rightarrow$ (3) Assume $S = I_F$ with F a function satisfying the specifications of the theorem. Let $\mathcal{P}_i \in Pro^{\Sigma}$ denote the program that computes $F_{(i)}$. It is possible to construct a program that, from an initial state $[\![(\vec{x}, \vec{\alpha})]\!]$, does the following:

(0) Let $N_{n+1} \leftarrow 1$.

(1) Evaluate whether each $\mathcal{P}_i$ with initial state

$$[\![(x_{n+1})_1, \ldots, (x_{n+1})_n, (*^{\leq}(\alpha_{m+1}))_1, \ldots, (*^{\leq}(\alpha_{m+1}))_m]\!]$$

halts in $t = (x_{n+1})_{n+1}$ steps.

(2) If the programs halt, proceed to step *(3)*. Else let $N_1 \leftarrow N_1 + 1$ and return to *(1)*.

(3) Let N_{n+i} store the output of $\mathcal{P}_i$ for $1 \leq i \leq n$, and P_{m+i} store the output of $\mathcal{P}_i$ for $n + 1 \leq i \leq m$.

(4) If $N_{n+i} = N_i \wedge P_{m+i} = P_i$ for all 1ß $\leq n + m$ return 1. Else loop forever.

The reader may implement this program in the $\mathcal{S}^{\Sigma}$ language using the $Halt$ function. The program attempts to (and must eventually succeed at) generating an element of S using the $F_{(1)}, \ldots, F_{(n+m)}$ functions. It then compares whether the values of the input state are equal to the generated element of S. If this is true, then it outputs 1. Otherwise it loops forever. It is evident that the program computes $\left(\lambda \vec{x} \vec{\alpha}\, [1]\right)_{|S}$. ■

(3) $\Rightarrow$ (1) Assume S is the domain of a Σ-computable function f. One can construct a program that does the following: Given a fixed $w \in S$ and a starting state $[\![x]\!]$,

(0) If $N_1 = 0$ set $N_1 \leftarrow w$ and finish.

(1) Check if $\mathcal{P}_f$ halts starting from

$$[\![(N_1)_1, \ldots, (N_1)_n, *^{\leq}((N_1)_{n+1}), \ldots, *^{\leq}((N_1)_{n+m})]\!]$$

in $(N_1)_{n+m+1}$ steps. If it does not, set $N_1 \leftarrow N_1 + 1$ and go to *(1)*; else continue.

(2) Set $N_1, \ldots, N_n, P_1, \ldots, P_m$ so that the final state is

$$[[(N_1)_1, \ldots, (N_1)_n, *^{\leq}((N_1)_{n+1}), \ldots, *^{\leq}((N_1)_{n+m})]]$$

and finish.

This program is easy to implement in $\mathcal{S}^{\Sigma}$. It is evident that it enumerates $\mathcal{D}_f = S$.

Problem 44 *Let* $\Sigma = \{@, +\}$ *and* $f : \mathcal{D}_f \subseteq \Sigma^* \mapsto \omega$ *a* Σ*-computable function s.t.* $f(\varepsilon) = 1$. *Let*

$$L = \left\{\alpha \in D_f : f(\alpha) = 1\right\}$$

Provide a program $\mathcal{Q} \in Pro^{\Sigma}$ *that enumerates L.*

Let $\mathcal{P} \in Pro^{\Sigma}$ be the program that computes f. Our program $\mathcal{Q}$, given a starting state $[[x]]$, may operate as follows:

> *(0)* Check if the input is zero; if it is return ε. (This is important because we will generate strings with the $*^{\leq}$ function, which never maps to ε—without this condition our program would never enumerate ε!) *(1)* Compute $\beta := *^{\leq}((x)_1), (x)_2$.
>
> *(2)* If $\mathcal{P}$ halts in $(x)_2$ steps starting from β, compute it and go to next step; else return ε (since $\varepsilon \in L$).
>
> *(3)* If the output of $\mathcal{P}$ was 1 return β; else return ε.

$$\begin{array}{ll}
 & [IF\ N_1 = 0\ GOTO\ L_2] \\
 & \left[P_1 \leftarrow *^{\leq}((N_1)_1)\right] \\
 & [N_1 \leftarrow (N_1)_2] \\
 & [IF\ Halt^{0,1}(N_1, P_1, \mathcal{P})\ GOTO\ L_1] \\
 & GOTO\ L_2 \\
L_1 & \left[N_2 \leftarrow E^{0,1}_{*1}(N_1, P_1, \mathcal{P})\right] \\
 & [IF\ N_2 = 1\ GOTO\ L_{10}] \\
L_2 & P_1 \leftarrow \epsilon \\
L_{10} & SKIP
\end{array}$$

Problem 45 *Let* $\Sigma = \{@, +\}$ *and* $\mathcal{P}_0 \in Pro^{\Sigma}$. *Show that*

$$S = \left\{(x, \alpha) : \Psi^{1,0,\#}_{\mathcal{P}_0}(x) = \Psi^{0,1,\#}_{\mathcal{P}_0}(\alpha)\right\}$$

is Σ*-enumerable.*

The program may operate as follows. Given an initial state $[[x]]$,

(1) Find $(x)_1, (x)_2$.
(2) Evaluate whether $\mathcal{P}$ halts in $(x)_2$ steps from $[[(x)_1]]$. If it does not, let $x = x + 1$ and return to *(1)*. If it does go to *(3)*.
(3) Find $\alpha = *^{\leq}((x)_3), (x)_4$.
(4) Evaluate whether $\mathcal{P}$ halts in $(x)_4$ steps from $[[\alpha]]$. If it doesn't let $x = x + 1$ and to *(1)*; it it does go to *(5)*.
(5) Check whether the computation of $\mathcal{P}$ from $[[(x)_1]]$ is the same as the computation of $\mathcal{P}$ from $[[\alpha]]$. If it is not, let $x = x+1$ and go to *(1)*, else go to (6).
(6) Set $N_1 \leftarrow (x)_1, P_1 \leftarrow \alpha$.

$$
\begin{array}{ll}
L_0 & [N_{11} \leftarrow (N_1)_1] \\
 & [N_{12} \leftarrow (N_1)_2] \\
 & [IF\ Halt^{1,0}(N_{12}, N_{11}, \mathcal{P})\ GOTO\ L_1] \\
 & N_1 \leftarrow N_1 + 1 \\
 & GOTO\ L_0 \\
L_1 & \left[P_{11} \leftarrow *^{\leq}((N_1)_3)\right] \\
 & [N_{14} \leftarrow (N_1)_4] \\
 & \left[IF\ Halt^{0,1}(N_{14}, P_{11}, \mathcal{P})\ GOTO\ L_2\right] \\
 & N_1 \leftarrow N_1 + 1 \\
 & GOTO\ L_0 \\
L_2 & \left[IF\ E^{1,0}_{\#1}(N_{12}, N_{11}, \mathcal{P}) = E^{0,1}_{\#,1}(N_{14}, P_{11}, \mathcal{P})\ GOTO\ L_3\right] \\
 & N_1 \leftarrow N_1 + 1 \\
 & GOTO\ L_0 \\
L_4 & N_1 \leftarrow N_{11} \\
 & P_1 \leftarrow P_{11}
\end{array}
$$

Problem 46 *If $S \subseteq \omega$ and $f : S \mapsto \omega$ is Σ-computable, then*

$$W = \{x \in S : x \text{ is even } \wedge x/2 \in S \wedge f(x) = f(x/2)\}$$

is Σ-enumerable.

There is some $\mathcal{P}_f \in Pro^{\Sigma}$ that computes f. $\therefore$ S is Σ-enumerable via some $\mathcal{P}_S \in Pro^{\Sigma}$. Let $w \in W$ an arbitrary element.

$$\begin{array}{rl}
 & [IF\ N_1 = 0\ GOTO\ L_{666}] \\
 & [N_2 \leftarrow (N_1)_1] \\
 & [N_3 \leftarrow (N_1)_2] \\
 & [N_4 \leftarrow \mathcal{P}_S(N_2)] \\
 & [N_5 \leftarrow \mathcal{P}_S(N_3)] \\
 & [IF\ N_4 \text{ is odd } GOTO\ L_{666}] \\
 & [IF \neg(N_5 = N_4/2)\ GOTO\ L_{666}] \\
 & [IF\ f(N_5) \neq f(N_4)\ GOTO\ L_{666}] \\
 & N_1 \leftarrow N_2 \\
 & GOTO\ L_1 \\
L_{666} & [N_1 \leftarrow w] \\
L_1 & SKIP
\end{array}$$

The program starting from $[[x]]$ generates two elements of $s_1, s_2 \in S$ using $(x)_1, (x)_2$—unless $x = 0$, where it just outputs w. If s_1 is even, and $s_2 = s_1/2$, and $f(s_1) = f(s_2)$, it outputs s_1. Else it outputs w. It is clear that the program enumerates W.

Problem 47 *Let $\Sigma = \{@, +\}$ and $\mathcal{P}_0 \in Pro^{\Sigma}$. Prove that*

$$W = \{(x, \alpha, \beta) \in \omega \times \Sigma^* \times \Sigma^* : \mathcal{P}_0 \text{ halts from } [[x, \alpha, \beta]]\}$$

is Σ-enumerable.

Observe that $W = \mathcal{D}_f$ where $f = \Psi^{1,2,\#}_{\mathcal{P}_0}$. We have already proven that the domain of a Σ-computable function is Σ-enumerable. Then W is Σ-enumerable.

If the program is still desired, let $(w, \alpha, \beta) \in \mathcal{D}_f$ an arbitrary element of the domain of f. Then

$$
\begin{array}{rl}
 & [IF\ N_1 = 0\ GOTO\ L_{666}] \\
 & [N_2 \leftarrow (N_1)_1] \\
 & [P_1 \leftarrow *^{\leq}((N_1)_2)] \\
 & [P_2 \leftarrow *^{\leq}((N_1)_3)] \\
 & [N_3 \leftarrow (N_1)_4] \\
 & [IF \neg \left(Halt^{1,2}(N_3, N_2, P_1, P_2, \mathcal{P}_f)\right)\ GOTO\ L_{666}] \\
 & N_1 \leftarrow N_2 \\
 & GOTO\ L_1 \\
L_{666} & [N_1 \leftarrow w] \\
 & [P_1 \leftarrow \alpha] \\
 & [P_2 \leftarrow \beta] \\
L_1 & SKIP
\end{array}
$$

enumerates W.

Problem 48 *Let* $\Sigma = \{@, +\}$ *and* $\mathcal{P}_0 \in Pro^{\Sigma}$. *Let*

$$L = \left\{\alpha \in \Sigma^* : (\exists x \in \mathbb{N})\ \Psi^{1,1,\#}_{\mathcal{P}_0}(x^2, \alpha) = \Psi^{0,2,\#}_{\mathcal{P}_0}(\alpha, \alpha)\right\}$$

Provide a program $\mathcal{P} \in Pro^{\Sigma}$ *s.t.* $\mathcal{D}_{\Psi^{1,0,*}_{\mathcal{P}}} = \omega$ *and* $\mathcal{I}_{\Psi^{1,0,*}_{\mathcal{P}}} = L$.

Let $f = \Psi^{1,1,\#}_{\mathcal{P}_0}, g = \Psi^{0,2,\#}_{\mathcal{P}_0}$, which have associated macros by virtue of being Σ-computable. L is the set of strings $\alpha \in \Sigma^*$ s.t. some $x \in \mathbb{N}$ satisfies $f(x^2, \alpha) = g(\alpha, \alpha)$. Given an arbitrary $\varphi \in L$ and a starting state $[[x]]$we can construct $\mathcal{P}$ as follows:

(0) If $N_1 = 0$ return φ and finish.

(1) Let $N_1 \leftarrow (N_1)_1, P_1 \leftarrow *^{\leq}((N_1)_2)$.

(2) Let $N_3 \leftarrow (N_1)_3, N_4 \leftarrow (N_1)_4$.

(3) Evaluate whether $\mathcal{P}_0$ halts with input $[[N_1^2, P_1]]$ in N_3 steps. Evaluate whether $\mathcal{P}_0$ halts with input $[[P_1, P_1]]$ in N_4 steps. If both are true continue, else set $P_1 \leftarrow \varphi$ and finish.

(4) Evaluate whether $f(N_1^2, P_1) = g(P_1, P_1)$. If false, set $P_1 \leftarrow \varphi$ and finish. Else continue.

(5) Finish.

Since $\left((x)_1, *^{\leq}\left((x)_2\right), (x)_3, (x)_4\right)$ over $x = 1, 2, \ldots$ explores all possible tuples $(x, \alpha, t_1, t_2) \in \omega \times \Sigma^* \times \omega \times \omega$, the program enumerates L.

Since every $\mathcal{P} \in Pro^{\Sigma}$ is a word in $\Sigma \cup \Sigma_p$, we can enumerate sets of programs $\mathscr{P} \subseteq (\Sigma \cup \Sigma_p)^*$. The enumeration of a set of programs is no different to the enumeration of any set of words. For example, say we want to enumerate

$$\mathscr{P} = \left\{\mathcal{P} \in Pro^{\Sigma} : \Psi_{\mathcal{P}}^{n,m,\#}(10) = 10\right\}$$

Observe that $SKIP \in (\Sigma \cup \Sigma_p)$ and the program $\mathcal{P} = SKIP \in \mathscr{P}$. Thus, an enumeration starting at $[[x]]$ can proceed as follows:

(1) If $x = 0$ let $P_1 \leftarrow SKIP$ and finish.

(2) Take $\alpha = *^{\leq}\left((x)_1\right), \varphi = (x)_2$.

(3) If $\alpha \in Pro^{\Sigma}$ and $\mathcal{P}$ halts at a state $[[10]]$ when starting from $[[10]]$ and in φ steps, let $P_1 \leftarrow \alpha$. Else let $P_1 \leftarrow SKIP$.

7.4 Church's thesis

Theorem 41 *If a Σ-mixed function f is Σ-Turing computable then it is Σ-recursive.*

Theorem 42 *If a Σ-mixed function is Σ-computable then it is Turing-computable.*

In virtue of the aforementioned theorems, we have that

Theorem 43 *The folllowing statements are equivalent:*

- *f is Σ-computable*
- *f is Σ-recursive*
- *f is Turing-computable*

The theorem above extends to set decidability and enumerability. Church's thesis is the following:

> **Church's thesis.** Every Σ-effectively computable function is Σ-recursive.

A formal proof of this thesis has not been given, but there's consensus around that fact that it is most likely true.

8 Extending Σ-recursion

We now extend the Σ-recursive paradigm with results which are easier to prove in the imperative paradigm.

Theorem 44 (Case division) *Assume $f_i : \mathcal{D}_{f_i} \subseteq \omega^n \times \Sigma^{*m} \mapsto \varphi$, with $\varphi = \omega$ or $\varphi = \Sigma^*$ and $i = 1, 2, \ldots, k$, are Σ-recursive functions. If $\mathcal{D}_{f_i} \neq \mathcal{D}_{f_j}$ for any $i \neq j, i, j \in [1, k]$, then $f_1 \cup \ldots \cup f_k$ is Σ-recursive.*

Proof. Let $\mathcal{P}_1, \ldots, \mathcal{P}_k \in Pro^{\Sigma}$ the programs that compute $f_1, \ldots, f_k$. We define $H_i = \lambda t \vec{x} \vec{\alpha}\, [Halt^{n,m}(t, \vec{x}, \vec{\alpha}, \mathcal{P}_i)]$. Assume $\varphi = \omega$.

Since $Halt^{n,m}$ is $(\Sigma \cup \Sigma_p)$-p.r. it is Σ-p.r. Then it is Σ-computable and has an associated IF macro. Then the (pseudo)program

$$
\begin{array}{rl}
L_0 & :N\overline{n+1} \leftarrow N\overline{n+1} + 1 \\
 & \left[IF\ Halt^{n,m}\left(N\overline{n+1}, N_1, \ldots, N_{\overline{n}}, P_1, \ldots, P_{\overline{m}}, \mathcal{P}_1\right)\ GOTO\ L_2\right] \\
 & \vdots \\
 & \left[IF\ Halt^{n,m}\left(N\overline{n+1}, N_1, \ldots, N_{\overline{n}}, P_1, \ldots, P_{\overline{m}}, \mathcal{P}_k\right)\ GOTO\ L_2\right] \\
L_1 & [N_1 \leftarrow f_1(N_1, \ldots, N\overline{n}, P_1, \ldots, P\overline{m})] \\
 & GOTO\ L\overline{n(\mathcal{P})} \\
L_2 & [N_1 \leftarrow f_2(N_1, \ldots, N\overline{n}, P_1, \ldots, P\overline{m})] \\
 & GOTO\ L\overline{n(\mathcal{P})} \\
 & \vdots \\
L\overline{k} & [N_1 \leftarrow f_k(N_1, \ldots, N\overline{n}, P_1, \ldots, P\overline{m})] \\
L\overline{n(\mathcal{P})} & SKIP
\end{array}
$$

Evidently the (pseudo) program computes $f_1 \cup \ldots \cup f_k$. If $\varphi = \Sigma^*$ the change that is to be done is trivial. ■

Theorem 45 *Let $f : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto \varphi$, with $\varphi = \omega$ or $\varphi = \Sigma^*$, a Σ-recursive function. If $S \subseteq \mathcal{D}_f$ is Σ-recursively enumerable, then $f_{|S}$ is Σ-recursive.*

Proof. We will not provide the explicit program but will describe what it should do.

S is Σ-recursively enumerable. ∴ it is Σ-enumerable. ∴ There are $F_{(1)}, \ldots, F_{(n+m)}$ s.t. $S = \mathcal{I}\left[F_{(1)}, \ldots, F_{(n+m)}\right]$ with $F_{(i)}$ Σ-enumerable for each $i \in [1, n+m]$.
∴ There is a macro for each $F_{(i)}$.

A program that from a starting state $[[\vec{x}, \vec{\alpha}]]$

(1) Generates an element of S using the macros of $F_{(1)}, \ldots, F_{(n+m)}$ from input variable $N\overline{n+1}$

(2) If the coordinates of the generated element correspond to $(\vec{x}, \vec{\alpha})$, compute $f(\vec{x}, \vec{\alpha})$ and return this computation. Else continue.

(3) Let $N\overline{n+1} \leftarrow N\overline{n+1} + 1$ and go back to *(1)*

Evidently, this program computes $f_{|S}$.

Theorem 46 *Let S_1, S_2 be Σ-recursive. Prove that $S_1 \cap S_2, S_1 \cup S_2, S_1 - S_2$ are Σ-recursive.*

Proof. Complete.

Theorem 47 *Let $S \subseteq \omega^n \times \Sigma^{*m}$. If S and $\omega^n \times \Sigma^{*m} - S$ are Σ-recursively enumerable, then S is Σ-recursive.*

Corollary. If S is Σ-recursively enumerable, then it isn't necessarily Σ-recursive.

Proof. $\chi_S^{\omega^n \times \Sigma^{*m}} = C_1^{n,m}{}_{|S} \cup C_0^{n,m}{}_{|\omega^n \times \Sigma^{*m} - S}$ is Σ-recursive and by hypothesis $S, \omega^n \times \Sigma^{*m} - S$ are Σ-recursively enumerable. $\therefore S$ is Σ-recursive.

Problem 49 *Provide a proof of the following theorem in the imperative paradigm.*

Theorem 48 *Let $S \subseteq \omega^n \times \Sigma^{*m}$. The following statements are equivalent.*

(1) S is Σ-recursively enumerable.

(2) S is the domain of a Σ-recursive function f.

(3) S is the image of a function $F : \mathcal{D}_f \subseteq \omega^n \times \Sigma^{*m} \mapsto S$ s.t. each $F_{(i)}$ is Σ-recursive.

Proof. We have proven before this holds for the imperative paradigm. Via paradigm equivalence then this holds for the recursive paradigm.

8.1 The Halting problem

When $\Sigma_p \subseteq \Sigma$, we define

$$AutoHalt^{\Sigma} = \lambda \mathcal{P} \left[(\exists t \in \omega)\, Halt^{0,1}(t, \mathcal{P}, \mathcal{P}) \right]$$

Evidently, $AutoHalt^{\Sigma} \mapsto 1$ with input $\mathcal{P}$ if $\mathcal{P}$ halts when inputted to itself.

Theorem 49 *$AutoHalt^{\Sigma}$ is not Σ-effectively computable. This is, there exists no Σ-effectively computable program that determines if a program halts with itself as input.*

Proof. Assume $AutoHalt$ is Σ-computable. Then we can make $\mathcal{P}$

$$L_1 \quad [IF\ AutoHalt^{\Sigma}(P_1)\ GOTO\ L_1]$$

Let the starting state be $[[\mathcal{P}]]$ itself. Observe that $\mathcal{P}$ halts if $\mathcal{P}$ does not halt, and if it does not halt then it halts. $\perp$ This can be easily extended to effectively computable programs beyond the imperative paradigm.