1 Enumerable sets

Let $\mathcal{F} : \mathcal{D}_{\mathcal{F}} \subseteq \omega^k \times \Sigma^{*l} \to \omega^n \times \Sigma^{*m}$. We define

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

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 1 *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. $\mathcal{P}(x) \mapsto_{halting} (\overrightarrow{x}, \overrightarrow{\alpha})$.*

1.1 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 $\big((x)_1, \ldots, (x)_n, *^{\leq}((x)_{n+1}), *^{\leq}((x)_m)\big)$. 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).

2 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

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

The pattern of primitive recursion. Primitive recursion consists of defining any function $R \sim (n, m, *)$ with a base case given by f and a recursive case given by g. f will always *lack the recursion parameter*, so if we are making recursion over numbers, it will have one less numeric argument than R; if we are making recursion over letters, it will have one less alphabetic argument than R. On the contrary, g will always a recursion over R in its arguments. Thus, if $R \mapsto \omega$, g will have one numeric argument more than R (the value of R in the recursive step); if $R \mapsto \Sigma$, then g will have one alphabetic argument more than R (same).

2.1 Numeric to numeric

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 1 *Find functions that recursively define $R = \lambda t\left[2^t\right]$*

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 2 *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 3 *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 4 *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.

2.2 Numeric to alphabet

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 5 *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 6 *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.

2.3 Alphabet to numeric

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:

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

Problem 7 *Let* $\Sigma = \{\%, @, ?\}$. *Find* $f, \mathcal{G}$ *s.t.* $R = \lambda \alpha_1 \alpha \, [|\alpha_1| + |\alpha|_{@}]$.

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

$$\begin{aligned} \mathcal{G} : \Sigma &\to \{Suc \circ p_1^{1,2}, p_1^{1,2}\} \\ \% &= p_2^{1,2} \\ ? &= p_2^{1,2} \\ @ &= Suc \circ p_2^{1,2} \end{aligned}$$

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

2.4 Alphabet to alphabet

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

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

Problem 8 *Let* $\Sigma = \{@, ?\}$. *Define* $R = \lambda\alpha_1\alpha[\alpha_1\alpha]$ *recursively.*

Observe that $R \sim (0, 2, *)$. $R(\alpha_1, \epsilon) = \alpha_1 \implies f := \lambda\alpha[\alpha]$. 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:

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

2.5 The point of primitive recursion

Theorem 2 *If* f, g *are* Σ*-computable then* $R(f, g)$ *is too.*

2.6 The primitive recursive set

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

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

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 $\alpha^x = \beta$.

2.7 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.

2.8 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

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

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 3 *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 4 *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 $\mathcal{D}_P = \omega \times \Sigma^* \times (\Sigma^* - \epsilon) \times \Sigma^*$. This set is Σ-p.r. because $\chi^{1,3}_{\mathcal{D}_P} = \neg\lambda\left[\alpha = \beta\right] \circ \left[p^{1,3}_3, C^{1,3}_\epsilon\right]$ 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 5 *A set S is Σ-p.r. if and only if it is the domain of a Σ-p.r. function.*

2.9 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.

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

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 $\lambda\alpha\left[[\alpha]_i\right]$ 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 $[\epsilon]_i = \epsilon$. From this follows that $R = [\alpha]_i \sim (1, 1)$ is difined via primitive recursion by $f = C_\epsilon^{1,0}$ 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 9 *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.*

Complete.

Problem 10 *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 $S \cup \overline{S} = \mathcal{D}_h$.

2.10 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 6 *If f is Σ-p.r. then the functions are Σ-p.r.*

To understand why, let $G = \lambda tx\overrightarrow{x}\overrightarrow{\alpha}\left[\sum_{i=x}^{i=t} f(i,\overrightarrow{x},\overrightarrow{\alpha})\right]$. 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. These can be proven using division by cases and domain restriction.

Problem 11 *Prove that $G = \lambda xx_1\left[\sum_{t=1}^{t=x} Pred(x_1)^t\right]$ is Σ-p.r.*

We know $f = \lambda xt\left[Pred(x)^t\right]$ is Σ-p.r. (trivial to show). Let $\mathcal{G} = \lambda xyx_1\left[\sum_{t=x}^{t=y} f(x_1,t)\right]$. 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 12 *Show that $G = \lambda xy\alpha\left[\prod_{t=y+1}^{t=|\alpha|}(t+|\alpha|)\right]$ is Σ-p.r.*

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

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

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. ■

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. ■

2.11 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 $L \subseteq L_{m+1}, S \subseteq S_0$:

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

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 7 *(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 13 *Let $\Sigma = \{@, !\}$. Show that $S = \{(2^x, @^x, !) : x \in \omega \wedge x$ impar$\}$ is Σ-p.r.*

For clarity, observe that a few elements of S are

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

Let $P_1 = \lambda x y \alpha \left[x = 2^{y+1} \right]$, $P_2 = \lambda x y \alpha [\alpha = @^{y+1}]$. 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

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

is Σ-p.r.

2.12 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 14 *Use the* ***U rule*** *to find a predicate P s.t. $M(P) = \lambda x[\textit{integer part of } \sqrt{x}]$*

.

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 15 *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 8 *If P a predicate that is effectively computable and $\mathcal{D}_P$ is effectively computable, then $M(P)$ is effectively computable.*

3 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 9 *If $f \in R^\Sigma$ then f is Σ-effectively computable.*

Theorem 10 *Not every Σ-recursive function is Σ-p.r. In other words,*

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

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 11 *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 12 *The quotient function, the remainder function, and the ith prime function are Σ-p.r.*

3.1 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 13 *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 16 *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 $M\left(P\left(\alpha,\beta\right)\right) \leq \beta$. In other words, $M(P)$ is bounded by $f = \lambda\alpha[\alpha]$. Then $\lambda\alpha[\sqrt{\alpha}] \in PR^{\Sigma}$.

3.2 Enumerable sets

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

3.3 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.

3.4 Alphabet independence

Theorem 14 *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.

4 Neumann

4.1 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

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

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

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

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

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

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.

4.2 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.

4.3 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 15 *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 $I_j^{\mathcal{P}} = \epsilon$ 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 17 *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 $\{(i, \mathcal{P}) \in \mathbb{N} \times Pro^\Sigma : i \leq n(\mathcal{P})\}$?*

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 18 *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 $\mathcal{P}_1 = \mathcal{P}_2$. ■

4.4 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

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

Recall that

$$\frown\alpha = \begin{cases} [\alpha]_2 \ldots \alpha_{|}\alpha| & |\alpha| \geq 2 \\ \epsilon & \text{otherwise} \end{cases}$$

We define $\omega^{\mathbb{N}} = \{(s_1, s_2, \ldots) : \exists n \in \mathbb{N} : i > n \Rightarrow s_i = 0\}$. 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

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

to denote the state $((x_1, \ldots, x_n, 0, 0, \ldots), (\alpha_1, \ldots, \alpha_m, \epsilon, \epsilon, \ldots))$.

4.5 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 $(i, \overrightarrow{s}, \overrightarrow{\alpha}) \in \omega \times \omega^{\mathbb{N}} \times \Sigma^{*\mathbb{N}}$. These 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). In other words,

4.6 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 19 *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.

4.7 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 $(\overrightarrow{s}, \overrightarrow{\sigma})$.

4.8 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 16 *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 20 *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 21 *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 22 *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

$$
\begin{array}{l}
\quad N_1 \leftarrow N_1 - 1 \\
\quad N_2 \leftarrow N_2 - 1 \\
\quad IFN_2 \neq 0\ GOTO\ L_1 \\
\quad IFN_1 \neq 0\ GOTO\ L_1 \\
\quad GOTO\ L_2 \\
L_1\ GOTO\ L_1 \\
L_2\ SKIP
\end{array}
$$

Problem 23 *Same for $\lambda i\alpha[[\alpha]_i]$ (same alphabet).*

$$
\begin{array}{rl}
 & IF\ N_0 \neq 0\ GOTO\ L_1 \\
 & P_1 \leftarrow \epsilon \\
 & GOTO\ L_{100} \\
L_1 & N_1 \leftarrow N_1 - 1 \\
L_2 & N_1 \leftarrow N_1 - 1 \\
 & P_1 \leftarrow {}^{\curvearrowright}P_1 \\
 & IF\ N_1 \neq 0\ GOTO\ L_2 \\
 & IF\ P_1\ STARTSWITH\ @\ GOTO\ L_2 \\
 & IF\ P_1\ STARTSWITH\ !\ GOTOL_3 \\
 & GOTOL_{100} \\
L_3 & P_1 \leftarrow\ ! \\
L_2 & P_1 \leftarrow @ \\
L_{100} & SKIP
\end{array}
$$

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 $P_1 = \epsilon$ 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 24 *Give a program that computes $s^{\leq}$ where $@ <!$.*

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

$$
\begin{aligned}
s^{\leq}\left((a_n)^m\right) &= (a_1)^{m+1} && m \geq 0 \\
s^{\leq}\left(\alpha a_i (a_n)^m\right) &= \alpha a_{i+1} (a_1)^m && 1 \leq i < n, m \geq 0
\end{aligned}
$$

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

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

4.9 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 17 *Let Σ a finite alphabet. Then if f a Σ-computable function, there is a macro $\left[Z\overline{n+1} \leftarrow f\left(V_1, \ldots, V\overline{n}, W_1, \ldots, W\overline{m}\right)\right]$ 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:

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

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 25 *Let* $\Sigma = \{@, !\}$ *and* $f \sim (0, 1, \#)$ *a* Σ*-computable function. Let* $L = \{\alpha \in \mathcal{D}_f : f(\alpha) = 1\}$*. 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 26 *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.* $\mathcal{D}_{\Psi^{1,0,*}_{\mathcal{P}}} = Im_f$*.*

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

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

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

4.10 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 18 *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*.

5 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 19 *In Godel's paradigm, S is Σ-computable iff it is the domain of a Σ-computable 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.*

6 Paradigm battles

6.1 Neumann triumphs over Godel

Theorem 20 *If h is Σ-recursive then it is Σ-computable.*

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