TASK C

Contents

1.	The Class $\mathcal{SFP}$	1
2.	From $\mathcal{SFP}$ to $\mathcal{POR}$	4
2.1.	Preliminary Notions	5
2.2.	From $\mathcal{SFP}$ to $\mathcal{PTF}_{\mathbb{S}}$	7
2.3.	From $\mathcal{PTF}_{\mathbb{S}}$ to $\mathcal{POR}^-$	7
2.4.	Extractor Function in $\mathcal{POR}$	11
3.	Appendix	13
3.1.	Encodings	13
3.2.	Section 1	22
3.3.	Section 2.1	22
3.4.	Section 2.3	24
3.5.	Section 2.4	26

Definition 1 (Class $\mathcal{PPT}$)**.** *The* class $\mathcal{PPT}$ *is the class of functions, which are computable by a PTM in a polynomial number of steps.*

We show that for each $f \in \mathcal{PPT}$, the probability that $f(x) = y$ is equal to the measure of the set of ω in the corresponding function of $\mathcal{POR}$, returning y on input x, and viceversa. Formally,

Conjecture 1. i. *For each $f \in \mathcal{PPT}$, there is a $g \in \mathcal{POR}$ such that:*

$$\Pr\big(f(x) = y\big) = \mu\big(\{\omega \in \mathbb{O} \mid g(x, \omega) = y\}\big).$$

ii. *For each $g \in \mathcal{POR}$, there is an $f \in \mathcal{PPT}$ such that:*

$$\mu\big(\omega \in \mathbb{O} \mid g(x, \omega) = y\}\big) = \Pr\big(f(x) = y\big).$$

Proof Sketch. First, we introduce an auxiliary class of functions, $\mathcal{SFP}$, which is equivalent to $\mathcal{PPT}$, but based on a source of randomness close to that of $\mathcal{POR}$, Section 1. Then, we actually relates the class $\mathcal{SFP}$ with $\mathcal{POR}$, Section 2, and viceversa, Section **??**. □

1. The Class $\mathcal{SFP}$

In the perspective of Conjecture 1, probabilistic Turing machines (PTM, for short) are not a suitable computational model.

Definition 2 (Probabilistic Turing Machines)**.** *A* probabilistic Turing machine *is a Turing machine with two transition functions: δ_0, δ_1. Given an input x, the PTM chooses at each step to apply, with probability $\frac{1}{2}$, either δ_0 or δ_1. The choice is independent from all the previous ones.*

Indeed, the source of randomness in a PTM is "implicit", as coming from a uniform distribution of probabilities, and the machine outputs distributions of probabilities over strings, (not strings). On the other hand, in $\mathcal{POR}$, randomness is enucleated by the oracle function, Q returning strings (not probability distributions).

To bridge this gap we introduce the *class* $\mathcal{SFP}$ of functions which are computable by stream machines in polynomial-time. This class is defined basing on a better-fitting computational model, namely *stream Turing machines* (STM, for short). In STMs, the source of randomness consists in an "explicit", read-only random tape. More precisely, these machines can be defined as ordinary TMs with $k+1$ tapes, which use one of them as a read-only *oracle*, that is to store an infinite stream of characters to be read from left to right, one character for transition. These machines are a valuable intermediate model, linking $\mathcal{POR}$ and $\mathcal{PPT}$. On the one hand, they use the oracle tape as an *explicit* source or randomness, that can be represented as a function $\eta : \mathbb{N} \longrightarrow \mathbb{B}$, and the behavior of which is consistent with that of the query functions and ωs in $\mathcal{POR}$. On the other hand, STMs share many features with multi-tape TMs.[1] Thus, many well-known results about the latter model still hold Intuitively, the class of functions computed by STM is almost equivalent to that of the ones computable by PTMs.

Definition 3 (Stream Turing Machine)**.** *A* stream Turing machine *is a quadruple* $M := \langle Q, q_0, \Sigma, \delta \rangle$, *where:*

- Q *is a finite set of states ranged over by* $q_1, \dots, q_n$
- $q_0 \in Q$ *is an initial state*
- Σ *is a finite set of characters ranged over by* $c_1, \dots, c_n$
- $\delta : \hat{\Sigma} \times Q \times \hat{\Sigma} \times \mathbb{B} \longrightarrow \hat{\Sigma} \times Q \times \hat{\Sigma} \times \{L, R\}$ *is a transition function describing the new configuration reached by the machine,*

where L *and* R *are two fixed constants, e.g.* $\mathbf{0}$ *and* $\mathbf{1}$, $\hat{\Sigma} = \Sigma \cup \{\circledast\}$ *and* $\circledast$ *represents the* blank character, *such that* $\circledast \notin \Sigma$.

Without loss of generality, we can assume $\Sigma = \{\mathbf{0}, \mathbf{1}\}$, and define the canonical STM as follows:

Definition 4 (Canonical STM)**.** *A* canonical stream machine *is an STM, such that* $\Sigma = \{\mathbf{0}, \mathbf{1}\}$, $L = \mathbf{0}$, *and* $R = \mathbf{1}$.

The *configuration* of a ordinary TM is a tuple which keeps track of the current state and some strings representing the state of the machine tape(s). When dealing with the portion of the oracle tape, which has not been queried yet is represented by means of a function $\eta : \mathbb{N} \longrightarrow \mathbb{B}$.

Definition 5 (Configuration of STM)**.** *The* configuration of an STM *is a quadruple* $\langle \sigma, q, \tau, \eta \rangle$, *where:*

- $\sigma \in \hat{\Sigma}^*$ *is the portion of the work tape on the left of the head*
- $q \in A$ *is the current state of the machine*
- $\tau \in \hat{\Sigma}$ *is the portion of the work tape on the right of the head*
- $\eta \in \mathbb{B}^{\mathbb{N}}$ *is the portion of the oracle tape that has not been read yet.*

At each step, the machine queries a new value on the oracle tape. Shifting on the work tape is naturally defined by pre-fixing and post-fixing characters to strings, so formalized by a shifting operation between the function η and a string σ.

[1]Conversely, we can see a TM as special STM, which basically "ignores" the oracle tape.

Definition 6 (Shifting Operation)**.** *Given $n \in \mathbb{N}$, $\sigma \in \mathbb{S}$, and $\eta : \mathbb{N} \longrightarrow \mathbb{B}$, we define the* shifting of η by σ, *$\sigma\eta(\cdot)$, by induction on the structure of σ:*

$$(\epsilon\eta)(n) := \eta(n)$$

$$(\mathbf{b}\tau)\eta(n) := \begin{cases} \mathbf{b} & \text{if } n = 0 \\ (\tau\eta)(n-1) & \text{otherwise.} \end{cases}$$

The dynamics of STMs is defined as predictable by extending the notion of standard transition function in the natural way.

Definition 7 (STM Transition Function)**.** *Given an STM, $M = \langle Q, q, \Sigma, \delta\rangle$, we define the* partial transition function $\vdash_\delta \hat{\Sigma}^* \times Q \times \hat{\Sigma}^* \times \mathbb{B}^{\mathbb{N}} \longrightarrow \hat{\Sigma}^* \times Q \times \hat{\Sigma}^* \times \mathbb{B}^{\mathbb{N}}$ *between two configurations as:*

$$\begin{aligned} \langle \sigma, q, c\tau, \mathbf{0}\eta\rangle &\Vdash_\delta \langle \sigma c', q', \tau, \eta\rangle && \text{if } \delta(q, c, \mathbf{0}) = \langle q', c', R\rangle \\ \langle \sigma c_0, q, c\tau, \mathbf{0}\eta\rangle &\Vdash_\delta \langle \sigma, q', c_0 c_1' \tau, \eta\rangle && \text{if } \delta(q, c_1, \mathbf{0}) = \langle q', c_1', L\rangle \\ \langle \sigma, q, c\tau, \mathbf{1}\eta\rangle &\Vdash_\delta \langle \sigma c', q', \tau, \eta\rangle && \text{if } \delta(q, c, \mathbf{1}) = \langle q', c', R\rangle \\ \langle \sigma c_0, q, c_1\tau, \mathbf{1}\eta\rangle &\Vdash_\delta \langle \sigma, q', c_0 c_1' \tau, \eta\rangle && \text{if } \delta(q, c_1, \mathbf{1}) = \langle q', c_1', L\rangle. \end{aligned}$$

The configuration reached by the machine after n steps of computation is obtained by composing n times its reachability function, defined below.

Definition 8 (STM Reachability Function)**.** *Given a STM, $M = \langle Q, q_0, \Sigma, \delta\rangle$, $\{|\!\triangleright^n_M\}_n$ is the smallest family of relations such that:*

$$\langle \sigma, q, \tau, \eta\rangle |\!\triangleright^0_M \langle \sigma, q, \tau, \eta\rangle$$

$$\big(\langle \sigma, q, \tau, \eta\rangle |\!\triangleright^n_M \langle \sigma', q', \tau', \eta'\rangle\big) \wedge \big(\langle \sigma', q', \tau', \eta'\rangle \Vdash_\delta \langle \sigma'', q'', \tau'', \eta''\rangle\big) \to \big(\langle \sigma, q, \tau, \eta\rangle |\!\triangleright^{n+1}_M \langle \sigma'', q'', \tau'', \eta''\rangle\big)$$

Without loss of generality, we assume STMs not to use final states: computation is regarded as concluded whenever the current configuration does not define the transition function.[2]

Proposition 1. *For any STM, $M = \langle Q, q_0, \Sigma, \delta\rangle$ and $n \in \mathbb{N}$, $|\!\triangleright^n_M$ is a function.*

Notation 1 (Final Configuration)**.** *Given an STM, $M = \langle Q, q_0, \Sigma, \delta\rangle$, and a configuration $\langle \sigma, q, \tau, \eta\rangle$, we write $\langle \sigma, q, \tau, \eta\rangle \not\Vdash_\delta$ when there are no $\sigma', q', \tau', \eta'$ such that $\langle \sigma, q, \tau, \eta\rangle \Vdash_\delta \langle \sigma', q', \tau', \eta'\rangle$.*

Finally, let us introduce the notion of function computable by (polytime) STMs.

Definition 9 (STM Computation)**.** *Given an STM, $M = \langle Q, q_0, \Sigma, \delta\rangle$, strings $\sigma, \gamma \in \mathbb{S}$, and $\eta : \mathbb{N} \longrightarrow \mathbb{B}$, we say that f_M* computes γ on input σ and oracle tape η, *$f_M(\sigma, \eta) = \gamma$, when there are a number $n \in \mathbb{N}$, astring$\tau \in \mathbb{S}$, $q' \in Q$, and a function $\psi : \mathbb{N} \longrightarrow \mathbb{B}$ such that:*

$$\langle \epsilon, q_0, \sigma, \eta\rangle \ |\!\triangleright^n_M \ \langle \gamma', q', \tau, \psi\rangle \not\Vdash_\delta,$$

γ being the longest suffix of γ' not including $\circledast$.

Definition 10 (Polytime Stream Machine)**.** *A* polytime stream machine *is an STM, $M = \langle \mathbb{Q}, q_0, \Sigma, \delta\rangle$ such that,*

$$\exists p \in \mathit{POLY}. \forall \sigma \in \mathbb{S}, \eta \in \mathbb{B}^{\mathbb{N}}. \exists n \le p(|\sigma|) \big(\langle \epsilon, q_0, \sigma, \eta\rangle |\!\triangleright^n_M \langle \gamma, q', \tau, \psi\rangle \not\Vdash_\delta .$$

We define the class of functions which are computable by polytime STMs.

Definition 11 (The Class $\mathcal{SFP}$)**.**

$$\mathcal{SFP} := \{f \in \mathbb{S} \times \mathbb{B}^{\mathbb{N}} \mid f = f_M\},$$

for some canonical polynomial STM, M.

[2]Indeed, in all these cases, we could imagine to add a final state q_F and a transition to it.

Remark 1. *All definitions given so far concern single-input, single-tape machines only. We could naturally generalize them as multi-input and multi-tape machines in the standard way.*

Finally, the notion of initial prefix of an oracle tape is also crucial as, we will show it sufficient to determine the value of the function, we formally introduce the notion of prefix:

Definition 12 (Prefix of Oracle Tape)**.** *For each function $\eta : \mathbb{N} \longrightarrow \mathbb{B}$, $\sigma \in \mathbb{S}$ and $n, i \in \mathbb{N}$ such that $i < n$,*

$$\eta_n = \sigma \quad \Leftrightarrow \quad \eta(i) = \sigma(i).$$

The following Lemma **??** is proved by induction on n.

Before going on, notice that $\mathcal{POR}$ and $\mathcal{SFP}$ are two inherently different sets:

$$\mathcal{POR} \subseteq \bigcup_{i \in \mathbb{N}} \mathbb{S}^i \times \mathbb{B}^{\mathbb{S}} \longrightarrow \mathbb{S}$$

$$\mathcal{SFP} \subseteq \bigcup_{i \in \mathbb{N}} \mathbb{S}^i \times \mathbb{B}^{\mathbb{N}} \longrightarrow \mathbb{S}$$

Thus, some effort would be required to relate these two classes and, actually, the correspondence obtained in this context would be weaker than the one between Σ^b_1-formulas representable in S^1_3 and $\mathcal{POR}$, from Section **??** and **??**. Differently from formulas of $\mathcal{L}_{\mathbb{PS}}$ and functions of $\mathcal{POR}$, which have access to their source of randomness in almost the same way, we cannot define a precise identity between $\mathcal{SFP}$ and $\mathcal{POR}$. We could relate sets of (oracle) functions, *which are not the same*, by considering their measure, and obtain a weaker (but strong enough) result, providing encodings between the two classes, which preserve measures of the random sources associated to the given input and output.

Conjecture 2. i. *For each $f \in \mathcal{SFP}$, there is a $g \in \mathcal{POR}$ such that for every $x, y \in \mathbb{S}$:*

$$\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid g(x, \omega) = y\}\big).$$

ii. *For each $g \in \mathcal{POR}$, there is an $f \in \mathcal{SFP}$ such that for every $x, y \in \mathbb{S}$:*

$$\mu\big(\{\omega \in \mathbb{O} \mid g(x, \omega) = y\}\big) = \mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}\big).$$

2. From $\mathcal{SFP}$ to $\mathcal{POR}$

In this Section, we address the first of the two claims in Conjecture 2:

Lemma 1. *For any $f \in \mathcal{SFP}$, there is a $g \in \mathcal{POR}$ such that for every $x, y \in \mathbb{S}$,*

$$\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x, \eta) = y\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid g(x, \omega) = y\}\big).$$

When reducing $\mathcal{SFP}$ to $\mathcal{POR}$, we can either build a direct encoding or not. A direct encoding would require an on-line management of the probabilistic choices made by the $\mathcal{POR}$ function on one hand and the $\mathcal{SFP}$ function. This would make the proof cumbersome and would require a consistent amount of effort to show the results. Intuitively, we would need to simulate any function computed by an STM with the oracle tape associated with η by means of a $\mathcal{POR}$-function g, querying a fresh *coordinate* of its oracle ω at each simulate step, in turn emulating the corresponding value written on η. Under a technical viewpoint, providing such apparently-natural correspondence between η and ω (i.e. linking functions in $\mathbb{O}$ and in $\mathbb{B}^{\mathbb{N}}$) is not trivial. For this reason, we prefer to divide the probabilistic concerns by the computational one, ending up with a slighter and clearer proof, but paying the cost of introducing another intermediate formalism.

Proof Sketch. Given an arbitrary $f \in \mathcal{SFP}$ with time bound $p \in \mathsf{POLY}$, we define a function $h : \mathbb{S} \times \mathbb{S} \longrightarrow \mathbb{S}$ such that

$$f(x, \eta) = h\big(x, \eta_{p(|x|)}\big),$$

where h is something very close to an ordinary *polytime* function [3]; in particular, $f \in \mathcal{PTF}_{\mathbb{S}}$. To prove that h is actually polytime we pass through the corresponding machines, as done in Section 2.2. Then, we define a function $h' : \mathbb{S} \times \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ such that, for any $x, y \in \mathbb{S}$ and $\omega \in \mathbb{B}^{\mathbb{S}}$:

$$h'(x, y, \omega) = h(x, y)$$

and we prove that $h' \in \mathcal{POR}^{-}$. Finally, in Section 2.4, we define a function an "extractor" function $e : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S} \in \mathcal{POR}$ to mimic the prefix extractor, namely we define e such that its output have *the same distribution* of all possible η's prefixes, but taking a functional argument with different signature. Proving $e \in \mathcal{POR}$. Since $\mathcal{POR}$ is closed under composition and $\mathcal{POR} \setminus \{Q\} \subseteq \mathcal{POR}$, we get the claim. In particular, taking

$$g(x, \omega) := h'(x, e(x, \omega), \omega)$$

$\square$

We will structure the proof as follows:

- In Section **??**, we will show that for each $f \in \mathcal{SFP}$ with time bound $p \in \mathsf{POLY}$ there is a $\mathcal{PTF}_{\mathbb{S}}$ $h : \mathbb{S} \longrightarrow \mathbb{S}$ such that
$$f(x, \eta) = h\big(x, \eta_{p(|x|)}\big),$$
- In Section **??**, we introduce a subset of $\mathcal{POR} \setminus \{Q\}$: a subset of $\mathcal{POR}$ which is complete with respect to poly-time functions, this entails that there is a function $h' \in \mathcal{POR}$ such that:
$$\forall x, y, \omega . h(x, y) = h'(x, y, \omega)$$
- Then, in Section **??**, we show that $e \in \mathcal{POR}$.
- In Section **??**, we join all the results in order to prove the claim.

2.1. **Preliminary Notions.** The proof of Lemma 1, relies on the introduction of some auxiliary notions. In particular, we need all the necessary instruments to show that the function h introduced in the proof of Lemma 1 is actually poly-time. To do so, we define:

- The class $\mathcal{PTF}_{\mathbb{S}}$, defined in Section 2.1.1.
- A subset of $\mathcal{POR}$ which is expressive enough to capture the class $\mathcal{PTF}_{\mathbb{S}}$, defined in Section 2.1.2.

2.1.1. *(Polytime) Finite Stream Turing Machine.* The Finite Stream Turing Machine are a blending between $\mathcal{SFP}$ machines, introduced in section **??** and ordinary Turing Machines. Basically, they are ordinary Stream Machines, but with the difference that the second tape can contain a finite stream of values.[4]

Definition 13 (Finite Stream Turing Machine)**.** *A* Finite Stream Turing machine *is a quadruple,* $M = \langle Q, q_0, \Sigma, \delta \rangle$*, where:*

- Q *is a finite set of states ranged over by* $q_1, \dots, q_n$
- $q_0 \in Q$ *is the initial state*
- Σ *is a finite set of characters rangeed over by* $c_1, \dots, c_n$
- $\delta : \hat{\Sigma} \times Q \times \hat{\Sigma} \times \hat{\Sigma} \longrightarrow \hat{\Sigma} \times Q \times \hat{\Sigma} \times \{L, R\}$ *is a transition function describing the new configuration reached by the machine.*

[3]The main difference lies in the fact that ordinary poly-time function should take one input only.

[4]Some definitions are omitted but can be found in Appendix [?]

where L and R are two fixed constants, e.g. $\mathbf{0}, \mathbf{1}$, $\hat{\Sigma} = \Sigma \cup \{\circledast\}$ *and* $\circledast$ *represents the blank character such that* $\circledast \notin \Sigma$.

A *canonical Finite Stream Turing Machine* is a FSTM, such that $\Sigma = \{\mathbf{0}, \mathbf{1}\}$, $L = \mathbf{0}$, and $R = \mathbf{1}$. Configuration is defined in the standard way.

Definition 14 (FSTM Transition Function)**.** *Given an STM,* $M = \langle Q, q, \Sigma, \delta\rangle$*, we define the* partial transition function $\vdash_\delta \hat{\Sigma}^* \times Q \times \hat{\Sigma}^* \times \hat{\Sigma}^* \longrightarrow \hat{\Sigma}^* \times Q \times \hat{\Sigma}^*$ *between two configurations as:*

$$\begin{aligned}
\langle \sigma, q, c\tau, d\rangle \vdash_\delta \langle \sigma c', q', \tau\rangle \qquad & \textit{if } \delta(q, c, d) = \langle q', c', R\rangle\\
\langle \sigma c_0, q, c\tau, d\rangle \vdash_\delta \langle \sigma, q', c_0 c_1' \tau\rangle \qquad & \textit{if } \delta(q, c_1, d) = \langle q', c_1', L\rangle\\
\langle \sigma, q, c\tau, d\rangle \vdash_\delta \langle \sigma c', q', \tau\rangle \qquad & \textit{if } \delta(q, c, d) = \langle q', c', R\rangle\\
\langle \sigma c_0, q, c_1\tau, d\rangle \vdash_\delta \langle \sigma, q', c_0 c_1' \tau, d\rangle \qquad & \textit{if } \delta(q, c_1, d) = \langle q', c_1', L\rangle.
\end{aligned}$$

As for STM, the configuration reached by the machine M after n steps of computation is obtained by composing n times its reachability function, denoted by $\triangleright_M^n$.

Definition 15 (Polytime Finite Stream Turing Machine)**.** *A* polytime Finite Stream Turing machine *is a TM,* $M = \langle Q, q_0, \Sigma, \delta\rangle$ *such that,*

$$\exists p \in \mathsf{POLY}.\forall \sigma, \tau \in \mathbb{S}.\exists n \leq p(|\sigma|)\big(\langle \epsilon, q_0, \sigma, \tau\rangle \triangleright_M^n \langle \gamma, q', \sigma', \tau'\rangle \not\vdash_\delta$$

Definition 16 (FSTM Computation)**.** *Given a FSTM,* $M = \langle Q, q_0, \Sigma, \delta\rangle$*, three strings* $\sigma, \tau, \gamma \in \mathbb{S}$*, we say that* f_M *computes* γ *on input* σ, τ $f_M(\sigma, \tau) = \gamma$*, when there are a natural number* $n \in \mathbb{N}$*, a string* $\xi \in \mathbb{S}, q' \in Q$*, such that*

$$\langle \epsilon, q, \sigma, \tau\rangle \triangleright_M^n \langle \gamma', q, \sigma', \tau'\rangle$$

with γ *being the longest suffix of* γ *not including* $\circledast$.

We are now able to define the class $\mathcal{PTF}_\mathbb{S}$, namely the class of functions which are computable by poly-time FSTMs.

Definition 17 (The Class $\mathcal{PTF}_\mathbb{S}$)**.**

$$\mathcal{PTF}_\mathbb{S} := \{f \in \mathbb{S} \times \mathbb{S} \longrightarrow \mathbb{S} \mid f = f_M\}$$

for some poly-time TM M.

The class $\mathcal{PTF}_\mathbb{S}$, basically, is a generalization to k-taped Turing Machines of the class PF.

2.1.2. *The Class* $\mathcal{POR}^-$. Furthermore, we present the class $\mathcal{POR}^-$ which is defined as $\mathcal{POR}$ except for the absence of the query function Q. This class is useful because it will be trivially shown being sound and complete with respect to Ferreira's PTCA [] (Remarks **??** and **??**). This entails that, showing that $\mathcal{POR}^-$ is complete with respect to $\mathcal{PTF}_\mathbb{S}$ function and that $\mathcal{PTF}_\mathbb{S}$ functions are complete with respect to *ordinary poly-time functions* (Lemma 14) yields, as a corollary, the proof that Ferreira's PTCA contains the class of poly-time computable functions. The completeness of $\mathcal{POR}^-$ with respect to FSTM function is proved in Section 2.3.

Definition 18 (The Class $\mathcal{POR}^-$)**.** *The class* $\mathcal{POR}^-$ *is the smallest class of functions* $\mathbb{S}^n \times \mathbb{O} \longrightarrow \mathbb{S}$ *containing:*

- *the empty function* $E(x, \omega) = \epsilon$
- *the projection function* $P_i^n(x_1, \dots, x_n, \omega) = x_i$
- *the word-successor* $S_\mathbf{b}(x, \omega) = x\mathbf{b}$*, for every* $\mathbf{b} \in \mathbb{B}$

- *the conditional function*
$$C(\epsilon, y, z_{\mathbf{0}}, z_{\mathbf{1}}, \omega) = y$$
$$C(x\mathbf{b}, y, z_{\mathbf{0}}, z_{\mathbf{1}}, \omega) = z_{\mathbf{b}}.$$
where $\mathbf{b} \in \mathbb{B}$.

and closed under:

- *composition, such that f is defined from $g, h_1, \dots, h_k$ as:*
$$f(\vec{x}) = g\big(h_1(\vec{x}, \omega), \dots, h_k(\vec{x}, \omega), \omega\big)$$
- *bounded recursion, such that f is defined from g, h_1, h_2 as:*
$$f(\vec{x}, \epsilon, \omega) := g(\vec{x}, \omega)$$
$$f(\vec{x}, y_1\mathbf{0}, \omega) := h_1\big(\vec{x}, y_1, f(\vec{x}, y_1, \omega), \omega\big)|_{t(\vec{x}, y)}$$
$$f(\vec{x}, y_2, \omega\mathbf{1}) := h_2\big(\vec{x}, y, f(\vec{x}, y_2, \omega), \omega\big)|_{t(\vec{x}, y_2)}$$
where t is defined from $\epsilon, \mathbf{0}, \mathbf{1}, \frown, \times$ by explicit definition.

2.2. **From $\mathcal{SFP}$ to $\mathcal{PTF}_{\mathbb{S}}$.** First of all we show that for any $f \in \mathcal{SFP}$, the corresponding $\mathcal{PTF}_{\mathbb{S}}$ function $h : \mathbb{S} \times \mathbb{S} \longrightarrow \mathbb{S}$ can be constructed. The core idea consists in passing through the associated machines. According to Definition 11, there is a polytime STM M, such that $f = f_M$. We want to construct the corresponding *poly-time* FSTM N, which, given as second argument polynomial prefix of η, behaves exactly like M. In particular, N is constructed basing on a two-taped TM N, which simply inherits the transition function of M For each computation, N performs a number of steps which is *exactly the same number of those performed* by M, so also this standard TM must be *polytime.*

Lemma 2. *For each $f \in \mathcal{SFP}$ with time-bound $p \in \mathsf{POLY}$, there is an $h \in \mathcal{PTF}_{\mathbb{S}}$ such that for any $\eta \in \mathbb{B}^{\mathbb{N}}$ and $x, y \in \mathbb{S}$,*
$$f(x, \eta) = h(x, \eta_{p(|x|)}).$$

Proof. Assume that $f \in \mathcal{SFP}$. By Definition 11, there is a polytime STM, $M = \langle Q, q_0, \Sigma, \delta\rangle$, such that $f = f_M$. Let us define a (standard) TM N which, given a polynomial prefix of η, behaves like M. The Definition of N is identical to the definition of M. Formally, for any $k \in \mathbb{N}$ and some $\sigma, \tau, y' \in \mathbb{S}$
$$\langle \epsilon, q_0', x, y\rangle \triangleright^k_{\delta'} \langle \sigma, q, \tau, y'\rangle \quad \Leftrightarrow \quad \langle \epsilon, q_0', x, \eta\rangle |\triangleright^k_{\delta} \langle \sigma, q, \tau, y'\eta\rangle.$$
Moreover, N requires a number of steps which is exactly equal to the number of steps required by M, and thus is in $\mathcal{PTF}_{\mathbb{S}}$, too. We conclude the proof defining $h = f_N$. □

2.3. **From $\mathcal{PTF}_{\mathbb{S}}$ to $\mathcal{POR}^-$.** In this Section we will show that all the function which are in $\mathcal{PTF}_{\mathbb{S}}$ can be represeted in $\mathcal{POR}$, too. This can formalized as follows.

Lemma 3. *For any $f \in \mathcal{PTF}_{\mathbb{S}}$ and $x \in \mathbb{S}$, there is a $g \in \mathcal{POR}^-$, such that $\forall x, y, \omega. f(x, y) = g(x, y, \omega)$.*

Proof Sketch. Any configuration of a Finite Stream Turing Machine can be encoded into strings, thus we can pass the value which is an encoding of the initial machines configuration to a function which emulates the execution of the machine which computes f (M_f) for a polynomial number of steps. When M reaches a final configuration, it is sufficient to extract from the final configuration the longest portion of the primary tape which is on the left and free form $\circledast$ characters. □

In order to prove the lemma above, we did an important technical work, which can be found in Section 3.1 of the Appendix. Now we will show that $\mathcal{POR}^-$ benefits of three important properties:

- There is a function *apply* $\in \mathcal{POR}^-$ which receiving as input the encoding of a FSTM configuration c, and the encoding of a FSTM transition function δ, it computes the encoding configuration obtained applying the function $\vdash_\delta$ on c.
- For each $f \in \mathcal{POR}^-$ and for each $x, y \in \mathbb{S}$ if there is a term $t(x)$ in $\mathcal{L}$ which bounds $f(x, \omega)$ for each ω, then there is a function which applies $|y|$ times f on its own output.
- There is a function *dectape* which extracts the machine's output from any encoded final configuration of a FSTM machine.

2.3.1. *The function apply.* To define *apply*, we show that given the encoding of a finite function (for details, see Corollary 5 in the Appendix), can be *interpreted* by means of a $\mathcal{POR}^-$ function, the *total function simulator*[5]

Definition 19 (Total Function Simulator)**.** *We define the* total function simulator $sim(\cdot, \cdot, \omega)$ *as follows:*

$$sim'(y, x, \epsilon, \omega) := \epsilon$$
$$sim'(y, x, z\mathbf{b}, \omega) := \mathit{if}\big(\pi_2(\pi(y, z\mathbf{b}, \omega), \omega), sim'(y, x, z, \omega), eq(\pi_1(\pi(y, z\mathbf{b}, \omega), \omega), x, \omega), \omega\big)$$

$$sim(x, y, \omega) := sim'\big(y, x, \pi_0(y, \omega), \omega\big).$$

The total function simulator is defined by induction on the number of elements in the encoding of the simulated function is defined (the number of pairs in the function's graph). At step i the unction extracts the i-th projection from the function's graph and compares its first element with the queried value. If they correspond, then it returns the second projection of the pair, otherwise is procedes recursively. The formal proof of the correctness of this function is in the Appendix, Lemma 15. It holds that there is a function *apply* in $\mathcal{POR}$ which, given a transition function δ, simulates the function $\vdash_\delta$ on the encodings of machine's configuration, formally:

Lemma 4. *The function apply is such that for any FSTM M with transition function δ, said x_δ the encoding of δ described in Corollary 5, and $h(c)$ the encoding of a configuration c as described in Corollary 6,*

$$\forall \omega \in \mathbb{O}. \vdash_\delta (c) = d \rightarrow apply(x_\delta, h(c), \omega) = h(d)$$
$$\forall \omega \in \mathbb{O}. (c \not\vdash_\delta) \rightarrow apply(x_\delta, h(c), \omega) = h(c)$$

The formal proof of the statement of Lemma 4, together with the definition of the function *apply* are quite cumbersome, for this reason they are given in the Appendix.

2.3.2. *Power Function.* In this section we show that for each function in $\mathcal{POR}^-$, if it has an output size which is bounded by a term in $\mathcal{L}$, then its n-th power is in $\mathcal{POR}^-$. Thanks to this result, we will be able to compute the polynomial transitive closure of a function in $\mathcal{POR}^-$, in order to compute the final configuration of any FSTM, given its δ function and its input.

Lemma 5. *For each $f : \mathbb{S}^{k+1} \times \mathbb{O} \longrightarrow \mathbb{S} \in \mathcal{POR}$ If there is a term $t \in \mathcal{L}$ such that $\forall x, \vec{z}, \omega. f(x, \vec{z}, \omega)|_t = f(x, \vec{z}, \omega)$ then there is also a function $sa_{f,t} : \mathbb{S}^{k+2} \times \mathbb{O} \longrightarrow \mathbb{S}$ such that*

$$\forall n \in \mathbb{N}. \forall x \in \mathbb{S}, \omega \in \mathbb{O}. sa_{f,t}(x, \underline{n}_\mathbb{N}, \vec{z}, \omega) = \underbrace{f(f(f(x, \vec{z}, \omega), \vec{z}, \omega), \ldots)}_{n \text{ times}}.$$

[5]We call it *total* because, since $\mathcal{POR}^-$ is total, it defines a default value to be returned when the simulated function is not defined on the queried input.

For sake of readability, we will proof this result giving the definition of the $sa_{\cdot,\cdot}$ funtion schema only in the Appendix, the details are in Proof 3.4. The function $sa_{\cdot,\cdot}$ allows us to compute the transitive closure of the function *apply*, but to do so, we must show that the growth of the terms computed by *apply* is under control. This is due to the fact that the only iterative mechanis of $\mathcal{POR}^-$ is the bounded recursion on notation, which requires that, at each step, the function outputs are bounded in size. So, the self-application schema must pass through bounded recursion on notation and then, requires size bounds. For this reason, we show that the term-growth of the size growth of the output of the function *apply* is at mosst constant.

Lemma 6 (*apply* Size growth)**.** *For all $x_c \in \mathbb{S}$ being the encoding of a FSTM configuration and for each encoding of a transition function δ in a string $x_\delta \in \mathbb{S}$ and for each $\omega \in \mathbb{O}$, there is a $k \in \mathbb{N}$ sich that $apply(x_c, x_\delta, \omega)|_{x_c\mathbf{1}^k}$*

The specific value of the size bound expressed in the lemma above is not particularly meaningful, and it is mainly due to the encodings we decided to adopt, but it is important that fixed a FSTM machine we can always find such k. A technical proof of this result is in the Appendix.

2.3.3. *Representing transition in $\mathcal{POR}^-$.* In this Section, we show that it is possible to define a function *dectape* $\in \mathcal{POR}^-$ which takes in input the encoding of a tape and $\omega \in \mathbb{O}$, and returns the longest suffix of the tape on the left of the head without any occurrence of $\circledast$. To do so, we need to do some technical work. First we introduce some auxiliary functions $\in \mathcal{POR}^-$: the difference function *diff* returning the difference between two numbers, the list-projector function π_n taking (the encoding of) a list and a number n as its input and returning the n-th element of the list, and the right-remover function *rrs*.[6] Moreover, $eq : \mathbb{S}^2 \times \mathbb{O} \longrightarrow \mathbb{S}$ is a function in $\mathcal{POR}^-$. While decoding the final configuration of a canonical TM, we need to extract the longest sequence of bits on the immediate left of the head. To do so, we introduce an auxiliary function $\rho : \mathbb{S}^2 \times \mathbb{O} \longrightarrow \mathbb{S}$ that is supposed to take the encoding of a tape as its input and return the y-th right character of the tape. Formally,

$$\rho(x, y, \omega) = \pi\big(x, diff(\pi_0(x, \omega), y, \omega), \omega\big).$$

Then, we define a *decoding* function so that at each step, due to *eq*, it checks whether the character obtained from ρ is the encoding of $\circledast$ defined in the Appendix, i.e. $\mathbf{111}$. If so, it returns ϵ.

Definition 20 (*dectape* Function)**.** *Let $dec : \mathbb{S} \times \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ be an auxiliary function defined as follows:*

$$\begin{aligned} dec(x, \epsilon, \omega) &= \epsilon \\ dec(x, y\mathbf{b}, \omega) &= if\big(dec(x, y, \omega)\rho(x, y\mathbf{b}, \omega), \epsilon, \neg eq(\rho(x, y\mathbf{b}, \omega), \omega), \mathbf{111}, \omega)\big) \end{aligned}$$

with

$$\rho(x, y, \omega) = \pi\big(x, diff(\pi_0(x, \omega), y, \omega), \omega\big).$$

We define the function dectape*: $\mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ as follows:*

$$dectape(x, \omega) = dec\big(x, rrs(\pi_0(x, \omega), \omega), \omega\big)$$

The function *dectape* returns a string which is the longest suffix not including $\circledast$ of the tape encoded with $\ulcorner\cdot\urcorner_{\mathbb{T}}$ and stored in x. By Definition **??**, this is precisely the value computed by the machine. A formal proof of this statement is given in the Appendix, the details are in Lemma 16

[6]Formal definitions are presented in Appendix **??**.

Lemma 7. *The function dectape $\in \mathcal{POR}^-$ is such that if $\underline{\cdot}_{\mathbb{T}}$ is the encoding for tapes above, then for any $\sigma \in \{\mathbf{0}, \mathbf{1}, \circledast\}$ $\omega \in \mathbb{O}$*

$$f(\underline{\sigma}_{\mathbb{T}}, \omega) = \tau$$

and τ is the longest suffix of σ without $\circledast$.

Let us also define a function *size* which allows us to compute the encoding of the size of a string throughout $\underline{\cdot}_{\mathbb{N}}$

Definition 21. *The function size : $\mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ is defined as follows:*

$$\mathit{size}(\epsilon, \omega) = \mathbf{1}$$
$$\mathit{size}(x\mathbf{b}, \omega) = \mathit{size}(x, \omega)\mathbf{1}|_{x\mathbf{11}}.$$

The correctness of the function *size*, with respect to the encoding we are adopting — namely $n \mapsto \mathbf{1}^{n+1}$ — can be shown by induction.

2.3.4. Concluding the Proof. It is now possible to show that, for any $\mathcal{SFP}$-function there is a function in $\mathcal{POR}^-$, which behaves exactly like the former one. The conclusion is the *composition* of three main ingredients:

- Machine steps can be simulated by means of the function *apply*.
- The function *apply* can be self-applied a polynomial numbers of times.
- We can extract the value computed by the machine by its final configuration.

These results allow us to give a proof to Lemma 8.

Lemma 8. *For any $f \in \mathcal{PTF}_{\mathbb{S}}$ and $x \in \mathbb{S}$, there is a $g \in \mathcal{POR}^-$, such that $\forall x, y, \omega. f(x, y) = g(x, y, \omega)$.*

Proof. By definition of $\mathcal{PTF}_{\mathbb{S}}$, there is a FSTM computing f on a machine M with a polynomial time-bound, p, and a transition function δ. Let us consider a function g defined as follows:

$$g(x, y, \omega) = \mathit{dectape}(\pi_1(\mathit{sa}_{\mathit{apply}, t_M}(x_\delta, \langle \underline{\circledast}_{\mathbb{T}}, \underline{0}_{\mathbb{N}}, \underline{x}_{\mathbb{T}}, \underline{y}_{\mathbb{T}} \rangle_{\mathbb{L}}, \underline{p}(\mathit{size}(x, \omega), \omega), \omega), \omega), \omega)$$

where p is the $\mathcal{POR}^-$-function which computes the encoding of the value of the polynomial p.[7] This function is correct as the self-application of *apply* for $p(|x|)$ times returns in the machine configuration, as a consequence of Lemma 4. Finally, *dectape* extracts the longest suffix free from blank characters which is on the left of the head in the encoding of the tape reached at the end of the computation. This is a consequence of the correctness of the projector π_1 with respect to the encoding of lists (for details, see the Appendix) and of the correctness of *dectape* (Lemma 16). As required by Definition **??**, this is exactly $f(x, y)$. □

As a consequence,

Corollary 1. *For each $f \in \mathcal{SFP}$ and polynomial time-bound $p \in$ POLY, there is a function $g \in \mathcal{POR}^-$ such that for any $\eta : \mathbb{N} \longrightarrow \mathbb{B}$, $\omega : \mathbb{N} \longrightarrow \mathbb{B}$ and $x \in \mathbb{S}$,*

$$f(x, \eta) = g\big(x, \eta_{p(|x|)}, \omega\big).$$

Proof. Assume $f \in \mathcal{SFP}$ and $y = \eta_{p(|x|)}$. By Lemma 2, there is a function $h \in \mathcal{PTF}_{\mathbb{S}}$ such that, for any $\eta : \mathbb{N} \longrightarrow \mathbb{B}$ and $x \in \mathbb{S}$,

$$f(x, \eta) = h\big(x, \eta_{p(|x|)}\big).$$

Moreover, due to Lemma 8, there is also a $g \in \mathcal{POR}^-$ such that

$$g(x, y, \omega) = h\big(x, y\big).$$

Then, the desired function is g. □

[7]This function is in $\mathcal{POR}^-$, because we have shown that this class contains all the polynomials.

2.4. **Extractor Function in $\mathcal{POR}$.** Finally, we construct an extractor function $e(x, \omega)$ in $\mathcal{POR}$ we introduced in the sketch of the proof of Lemma 1. This function extracts $|x| + 1$ bits from ω and places them in a and concatenates them in its output. As said, for any polynomial p, $x \in \mathbb{S}$ and $\omega \in \mathbb{O}$, there is $t \in \mathcal{POR}^- \subseteq \mathcal{POR}$, such that $|t(x, \omega)| = p(|x|)$. Then, there is also a $t \in \mathcal{POR}$ such that for any $\vec{x} \in \mathbb{S}$ and $\omega \in \mathbb{O}$, $|t(\vec{x}, \omega)| = p(|\vec{x}|)$. Moreover, it is possible to associate each natural number n to its dyadic representation. Intuitively, for each natural number $n \in \mathbb{N}$, if n_2 is the encoding of n in base 2, the dyadic representation of m is $(m + 1)_2$ without is leftmost bit. The intuition is at the basis of our definition of an "extractor" function in $\mathcal{POR}$.

First, we define a function $bin : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ such that, if x has length k, then for every $\omega \in \mathbb{O}$, $bin(x, \omega)$ is the binary encoding of k.

Definition 22. *Let us define an auxiliary function $binsucc : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$,*

$$\begin{aligned} binsucc(\epsilon, \omega) &= \mathbf{1} \\ binsucc(x\mathbf{0}, \omega) &= x\mathbf{1}|_{x\mathbf{00}} \\ binsucc(x\mathbf{1}, \omega) &= binsucc(x, \omega)\mathbf{0}|_{x\mathbf{00}}. \end{aligned}$$

Then, $bin : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ is defined as:

$$\begin{aligned} bin(\epsilon, \omega) &= \mathbf{0} \\ bin(x\mathbf{b}, \omega) &= binsucc\big(bin(x, \omega), \omega\big)|_{x\mathbf{b}} \end{aligned}$$

On top of the definition of the binary encoding of a number, we define the dyadic encoding of it:

Definition 23. *We call $dy : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ the function in $\mathcal{POR}^-$ such that $\forall n \in \mathbb{N}, \omega \in \mathbb{O}. dy(\underline{n}_{\mathbb{N}}, \omega)$ is the diadic encoding of n. Namely:*

$$dy(x, \omega) := lrs(bin(x, \omega), \omega), \omega$$

In the Appendix (Lemma 17) we prove that for any ω, the function $dy(\underline{n}_{\mathbb{N}}, \omega)$ is a bijection between a a natural number n and a string — its dyadic representation. This result will be crucial proving Lemma 1, because it relates throughout a bijection the set $\mathbb{B}^{\mathbb{N}}$ to $\mathbb{B}^{\mathbb{S}}$. We then define a function $e \in \mathcal{POR}$, which takes a string and an oracle as its input and returns a finite string obtained picking $|x|$ bytes from it, choosing exactly the coordinates of ω corresponding to the dyadic encoding of the first $|x|$ natural numbers.

Definition 24. *Let $e : \mathbb{S} \times \mathbb{O} \longrightarrow \mathbb{S}$ be defined as follows:*

$$\begin{aligned} e(\epsilon, \omega) &= \epsilon \\ e(x\mathbf{b}, \omega) &= e(x, \omega)Q\big(dy(x, \omega), \omega\big)|_{x\mathbf{b}}. \end{aligned}$$

Definition 25. *We define $\sim_{dy}$ as the smallest relation in $\mathbb{B}^{\mathbb{S}} \times \mathbb{B}^{\mathbb{N}}$ such that:*

$$\eta \sim_{dy} \omega \leftrightarrow \forall n \in \mathbb{N}. \eta(n) = \omega(dy(\underline{n}_{\mathbb{N}}, \omega))$$

This relation has some good properties:

Lemma 9. *It holds that:*

- $\forall \eta \in \mathbb{B}^{\mathbb{N}}. \exists! \omega \in \mathbb{B}^{\mathbb{S}}. \eta \sim_{dy} \omega$
- $\forall \omega \in \mathbb{B}^{\mathbb{S}}. \exists! \eta \in \mathbb{B}^{\mathbb{N}}. \eta \sim_{dy} \omega$

Proof. The proofs are very similar, for this we will take in exam only the first. By the fact that dy is a bijectio with respect to its first argument and is constant over its second argument, we obtain the existence of an ω which is in relation with η. Now suppose that there are ω_1, ω_2 both in relation with η being different. It holds then that $\exists \sigma \in \mathbb{S}$ such that $\omega_1(\sigma) \neq \omega_2(\sigma)$. Then, since dy is in $\mathcal{POR}^-$, the value of its last argument does not affect th value of its output, moreover it is a bijection so we get that the claim holds. □

Corollary 2. *The relation $\sim_{dy}$ is a bijection.*

Proof. Consequence of Lemma 9. □

Corollary 3. *For each $f \in \mathcal{SFP}$, there is a $g \in \mathcal{POR}$ such that:*

$$\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x,\eta) = y\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid g(x,\omega) = y\}\big).$$

Proof. From Corollary 1, we know that there is a function $f' \in \mathcal{POR}^-$, and a $p \in \mathsf{POLY}$ such that

$$\forall x, y \in \mathbb{S}.\forall \eta.\forall \omega.y = \eta_{p(x)} \to f(x,\eta) = f'(x,y,\omega) \quad (*)$$

So, by the fact that $\mathcal{POR}^- \subseteq \mathcal{POR}$, $f' \in \mathcal{POR}$, too. Fixed an $\eta \in \{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x,\eta) = y\}$, its image with respect to $\sim_{dy}$ is in

$$\{\omega \in \mathbb{O} \mid f'(x, e(p'(\mathit{size}(x,\omega),\omega),\omega),\omega) = y\}$$

Indeed, by Lemma 18, it holds that $\eta_{p(x)} = e(p(\mathit{size}(x,\omega),\omega)$, where p' is the $\mathcal{POR}^-$ function computing the polynomial p and size is the $\mathcal{POR}^-$, to function computing the encoding of the natural number which corresponds to the size of its first input, so by $(*)$ we have the claim. It also holds that, fixed an $\omega \in \{\omega \in \mathbb{O} \mid f'(x, e(p'(\mathit{size}(x,\omega),\omega),\omega),\omega) = y\}$, then its pre-image with respect to $\sim_{dy}$ is in $\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x,\eta) = y\}$. The proof is analogous to the one we showed above. Now, since $\sim_{dy}$ is a bijection between the two sets, we conclude that

$$\mu\big(\{\eta \in \mathbb{B}^{\mathbb{N}} \mid f(x,\eta) = y\}\big) = \mu\big(\{\omega \in \mathbb{O} \mid g(x, e(p(\mathit{size}(x,\omega),\omega),\omega),\omega) = y\}\big).$$

which concludes the proof. □

3. Folklore becomes true

4. Appendix

4.1. **Encodings.** In the Proof Sketch of Lemma 8, we discussed that we can show the inclusion of $\mathcal{PTF}_{\mathbb{S}}$ in $\mathcal{POR}$ building a function which manipulates the encoding of a machine's configuration. Moreover, the configuration will be manipulated according to the behavior described by δ. For this reason, this section is aimed to show that $\mathcal{POR}^-$ is expressive enough to represent all the data structures which are needed to support the simulation of FSTM in $\mathcal{POR}$. In particular, in the first section we will show how we will represent the data, then in the following section, we will show how we can use $\mathcal{POR}^-$ to define functions which can be used to manipulate data encodings in order to implement complex behaviors such as the simulation of a FSTM.

4.1.1. *Statical encodings.* Before getting into more complex encondings, we start with a toy problem: the encoding of the constant strings. We did not mention such values before, but they are needed, for instance, as a notational support, because they allow us to write string values directly into $\mathcal{POR}^-$ function rather than writing explicitly the functions computing them. It may seem superfluos to show that there is a $\mathcal{POR}^-$ function for all the constant strings, but it will be useful to introduce the same pattern which we will follow later for the definition of more complex encodings.

Definition 26 (Encoding of Constants). *We define the injection* $\underline{\cdot}_{\mathbb{S}} : \mathbb{S} \longrightarrow \mathbb{S}$ *as follows:*

$$\underline{\sigma}_{\mathbb{S}} := \sigma$$

For sake of readability, from now on, the notation $\underline{\cdot}_{\mathbb{S}}$ will be omitted.

Now we should show that this encoding is suitable for $\mathcal{POR}^-$, which means that when we will insert a constant inside the definition of a $\mathcal{POR}^-$ function, it preserves the fact that the overall function is in $\mathcal{POR}^-$. If we show that for each constant there is a function in $\mathcal{POR}^-$ which computes it, then the composition of $\mathcal{POR}^-$ functions with strings is in $\mathcal{POR}^-$ for compositionality.

Remark 2 (Representability of $\mathbb{S}$ in $\mathcal{POR}^-$)**.** $\forall \sigma \in \mathbb{S}.\exists f_\sigma \in \mathcal{POR}.\forall x \in \mathbb{S}.\forall \omega \in \mathbb{O}.f_\sigma(x,\omega) = \underline{\sigma}_{\mathbb{S}}$

Proof. Take as f_σ:

$$f_\epsilon(x,\omega) := E(x,\omega)$$
$$f_{\tau\mathbf{b}}(x,\omega) := S_{\mathbf{b}}(f_\tau(x,\omega),\omega)$$

The claim can be obtained by induction on σ. □

We choose to represent the natural number n as a string composed by $n+1$ **1**s [8], in this way, definitions of arithmetical functions will be simpler.

Definition 27 (Encoding of Natural numbers). *We define the injection* $\underline{\cdot}_{\mathbb{N}} : \mathbb{N} \longrightarrow \mathbb{S}$ *as follows:*

$$\underline{n}_{\mathbb{N}} := \mathbf{1}^{n+1}$$

This encoding is suitable for $\mathcal{POR}^-$, because we can show that each string which is the encoding of a natural number can be represented by a term in $\mathcal{POR}^-$, formally:

Remark 3 (Representability of $\mathbb{N}$ in $\mathcal{POR}^-$)**.** $\forall n \in \mathbb{N}.\exists f_n \in \mathcal{POR}.\forall x,\omega.f_n(x,\omega) = \underline{n}_{\mathbb{N}}$

[8]We took this decision because we do not want ϵ to be neither the encoding of a natural number nor of any other value (indexes, Booleans, tuples, ...). This will allow us to use it as return value in case of errors, if necessary.

Proof. Take as f_n:

$$\begin{aligned} f_0(x,\omega) &:= S_{\mathbf{1}}(E(x,\omega),\omega) \\ f_{n+1}(x,\omega) &:= S_{\mathbf{1}}(f_n(x,\omega),\omega) \end{aligned}$$

The claim can be obtained by induction on n. $\square$

Conventionally, the true Boolean value is always represented with $\mathbf{1}$ and the false value is represented with $\mathbf{0}$. We will follow this convention.

Definition 28 (Encoding of Boolean values)**.** *We define the injection* $\underline{\cdot}_{\mathbb{B}} : \mathbb{B} \longrightarrow \mathbb{S}$ *as follows:*

$$\begin{aligned} \underline{\mathbf{0}}_{\mathbb{B}} &:= \mathbf{0} \\ \underline{\mathbf{1}}_{\mathbb{B}} &:= \mathbf{1} \end{aligned}$$

Remark 4 (Representability of $\mathbb{B}$ in $\mathcal{POR}^-$)**.** $\forall \mathbf{b} \in \mathbb{B}.\exists f_{\mathbf{b}} \in \mathcal{POR}.\forall x,\omega.f_{\mathbf{b}}(x,\omega) = \underline{n}_{\mathbb{B}}$

Proof. Take as f_n:

$$\begin{aligned} f_{\mathbf{0}}(x,\omega) &:= S_{\mathbf{0}}(E(x,\omega),\omega) \\ f_{\mathbf{1}}(x,\omega) &:= S_{\mathbf{1}}(E(x,\omega),\omega) \end{aligned}$$

The claim is correct by definition of the functions $f_{\mathbf{0}}$ and $f_{\mathbf{1}}$. $\square$

In order to encode in $\mathcal{POR}^-$ any polynomial FSTM, we need to represent some complex data structures: we argued that for each $f \in \mathcal{PTF}_{\mathbb{S}}$, there is a function g implementing it. This can be done defining a general schema for g which depends on a representation of the machine transition function δ. In this way, changing only the function which encodes δ, we are be able to decouple the encoding of the machine's mechanics, by the machine's transition function.

Tho encode the information contained in the finite function δ (which can be considered a big but finite table), we need to define some compositional data structures. Basically, functions are sets of pairs. So, for sake of simplicity, we chose to represent all these different data structures as untyped lists. This works because lists are a generalization of sets, tuples and thus of functions, too.

Lists. In order to represent lists, we will adapt an encoding from [**?**, p. 183] which was introduced as a representation of tuples. Before doing so, we would like to formally define the set of lists, in order to clarify which kind of data we will be working on.

Definition 29 (Lists of strings)**.** *We say that l is a list of strings if and only if $l \in \mathbb{L}$, where* $\mathbb{L} := \bigcup_{i\in\mathbb{N}} \mathbb{S}^i$.

Before presenting the encoding, we need to introduce two auxiliary functions, the *doubling* function $\mathcal{D}$ and the *halving* function $\mathcal{H}$, such epithets are due to the fact that they respectively double and halve the length of their input. The function $\mathcal{D}$ interleaves the bits in its input with $\mathbf{1}$s and $\mathcal{H}$ removes those characters.

The $\mathcal{D}$ function can be seen as the mapping of a string σ through an encoding $\mathbb{B} \longrightarrow \mathbb{B}^2$, and $\mathcal{H}$ is the left-inverse of $\mathcal{D}$. In this way we can represent lists as sequences of characters $c \in \mathbb{B}^2$. Two of those four characters are used to encode the values in $\mathbb{B}$ which compose the strings in the list, while one of the two remaining characters (we chose $\mathbf{00}$) is used as separator. This way we will be able to embed multiple strings in a single sequence of bits encoding a list and to extract the values contained in it.

Definition 30 (List Element Encoding and Decoding Functions)**.** *Encoding and decoding functions are defined as follows:*

$$\mathcal{D}(\sigma\mathbf{0}) := \mathcal{D}(\sigma)\mathbf{10}$$
$$\mathcal{D}(\sigma\mathbf{1}) := \mathcal{D}(\sigma)\mathbf{11}$$

$$\mathcal{H}(\sigma\mathbf{10}) := \mathcal{H}(\sigma)\mathbf{0}$$
$$\mathcal{H}(\sigma\mathbf{11}) := \mathcal{H}(\sigma)\mathbf{1}.$$

We can finally define the encoding of lists.

Definition 31 (List Encoding)**.** *The family of list encoding injections* $\langle\cdot,\dots,\cdot\rangle^n_{\mathbb{L}} : \mathbb{S}^n \longrightarrow \mathbb{S}$ *is defined as the family of functions described below:*

$$\langle\cdot,\dots,\cdot\rangle^n_{\mathbb{L}} := \mathbf{00}\mathcal{D}(x_n)\mathbf{00}\dots\mathbf{00}\mathcal{D}(x_1)\mathbf{00}\mathcal{D}(x_0)\mathbf{00}\mathcal{D}(\underline{n}_{\mathbb{N}})\mathbf{00}$$

Following the pattern which we used for the previous encodings we should show that all the values which encode tuples can be represented by functions in $\mathcal{POR}^-$. To do so, we need to show that $\mathcal{D}$ and $\mathcal{D}$ have got their natural corresponednt in $\mathcal{POR}^-$, together with the fact that the function which computes the concatenation is in $\mathcal{POR}^-$, too.

Definition 32 (String Concatenation function)**.**

$$concat(x, \epsilon, \omega) := x$$
$$concat(x, y\mathbf{b}, \omega) := S_{\mathbf{b}}(concat(x, y, \omega), \omega)|_{xy\mathbf{b}}$$

Lemma 10 (String concatenation is in $\mathcal{POR}^-$)**.** *Formally:* $concat \in \mathcal{POR} \wedge \forall x_1, x_2 \in \mathbb{S}.\forall\omega \in \mathbb{O}.concat(x_1, x_2, \omega) = x_1x_2$

Proof. Such function is defined by bounded recursion, so the result can be proven by induction on x_2. □

Lemma 11 ($\mathcal{D}$ is in $\mathcal{POR}^-$)**.** *There is a function doub in* $\mathcal{POR}^-$ *such that* $\forall x \in \mathbb{S}, \forall\omega \in \mathbb{O}.doub(x, \omega) = \mathcal{D}(x)$

Proof. It can be shown that the doubling function, *doub*, can be defined by bounded recursion:

$$doub(\epsilon, \omega) := \epsilon$$
$$doub(y\mathbf{1}, \omega) := S_{\mathbf{1}}(S_{\mathbf{1}}(doub(y, \omega), \omega))|_{(\mathbf{11})\times(y\mathbf{1})}$$
$$doub(y\mathbf{0}, \omega) := S_{\mathbf{0}}(S_{\mathbf{1}}(doub(y, \omega), \omega))|_{(\mathbf{11})\times(y\mathbf{1})}.$$

□

Remark 5. *All the constant lists can be represented by functions in* $\mathcal{POR}^-$*. Namely:* $\forall l \in \mathbb{L}.\exists f_l \in \mathcal{POR}.\forall x \in Ss, \omega \in \mathbb{O}.f_l(x, \omega) = \langle l\rangle^n_{\mathbb{L}}$.

Proof. This comes form the fact that $\langle l\rangle^n_{\mathbb{L}}$ can be defined by composition of functions which are in $\mathcal{POR}^-$, namely constant functions (e.g. $f_{\mathbf{00}}$), *concat*, f_n, D_f. □

We can also state representability results dealing with functions and sets, instead of lists. We will not state a similar result for tuples directly because, according to Definition 29, tuples are *exactly* specific cases of lists. Concerning sets and functions, basically we can represents sets as specific lists with at most one copy per element, and finite functions from $\mathbb{S}$ to $\mathbb{S}$ (but not only) with their graph using lists as generalized couples and sets.

Definition 33 (Finite Set Encoding)**.** *The family of set encoding injections* $\underline{\cdot}_{\mathcal{P}_{fin}(\mathbb{S})} : \mathcal{P}_{fin}(\mathbb{S}) \longrightarrow \mathbb{S}$ *is defined as the family of function described below:*

$$\underline{\{x_1, \dots, x_n\}}_{\mathcal{P}_{fin}(\mathbb{S})} := \langle x_1, \dots, x_n \rangle^n_{\mathbb{L}}$$

where $x_1, \dots, x_n$ *are taken in ascending order according to the value of the natural number encoded by* $\mathbf{1}x_i$*. This definition il well-founded because sets do not contain repetitions.*

Definition 34 (Function Encoding)**.** *The family of Finite Function Encoding injections* $\underline{\cdot}_{\mathbb{S}^\mathbb{S}} : \mathbb{S}^\mathbb{S} \longrightarrow \mathbb{S}$ *is defined as the family of functions described below:*

$$\underline{\{\langle x_1, y_1\rangle, \dots, \langle x_n, y_n\rangle\}}_{\mathbb{S}^\mathbb{S}} := \langle \langle x_1, y_1\rangle^n_{\mathbb{L}}, \dots, \langle x_n, y_n\rangle^n_{\mathbb{L}}\rangle^n_{\mathbb{L}}$$

Corollary 4. *Each finite function from* $\mathbb{S}$ *to* $\mathbb{S}$ *and each set included in* $\mathbb{S}$ *can be represented by functions in* $\mathcal{POR}^-$*. Namely:*

$$\forall g \in \mathcal{P}_{fin}(\mathbb{S} \times \mathbb{S}).\exists f_g \in \mathcal{POR}.\forall x \in \mathbb{S}, \omega \in \mathbb{O}.f_g(x, \omega) = \underline{g}_{\mathbb{S}^\mathbb{S}}$$
$$\forall s \in \mathcal{P}_{fin}(\mathbb{S}).\exists f_s \in \mathcal{POR}.\forall x \in \mathbb{S}, \omega \in \mathbb{O}.f_s(x, \omega) = \underline{s}_{P_{fin}(\mathbb{S})}$$

Proof. This claim is a direct consequence of Definitions 34 and 33 and of Remark 5. □

Another result which we need to state is that the graph of any function $\delta : \mathbb{Q} \times \hat{\Sigma} \times \{\mathbf{0}, \mathbf{1}\} \longrightarrow \mathbb{Q} \times \hat{\Sigma} \times \{L, R\}$ can be represented by a term in $\mathbb{S}$. Formally:

Corollary 5 (Representability of δ)**.** *There is an injection* $f : (\mathbb{Q} \times \hat{\Sigma} \times \{\mathbf{0}, \mathbf{1}\} \longrightarrow \mathbb{Q} \times \hat{\Sigma} \times \{L, R\}) \longrightarrow \mathbb{S}$*.*

Proof. We have suggested that the constants L and R can be represented by means of $\mathbf{0}$ and $\mathbf{1}$ respectively, this describes an injection $i_1 : \{L, R\} \longrightarrow \mathbb{S}$. Similarly, we have suggested that we can be represented through their indexes. This identifies an injection $i_2 : \mathbb{Q} \longrightarrow \mathbb{N}$, we have also shown that there exists a natural injection $\underline{\cdot}_{\mathbb{N}} : \mathbb{N} \longrightarrow \mathbb{S}$; so, the composition of i_2 and $\underline{\cdot}_{\mathbb{N}}$ is an injection $\mathbb{Q} \longrightarrow \mathbb{S}$. Since $\hat{\Sigma}$ is a finite set, it is countable too, so there is a bijection $i_3 : \hat{\Sigma} \longrightarrow \mathbb{N}$, which composed with $\underline{\cdot}_{\mathbb{N}}$ yields another injection $\hat{\Sigma} \longrightarrow \mathbb{S}$. These observations, together with Remark 5 prove that there is an injection $i_4 : \Big(\mathbb{Q} \times \hat{\Sigma} \times \{\mathbf{0}, \mathbf{1}\}\Big) \longrightarrow \mathbb{S}$ and another injection $i_5 : \Big(\mathbb{Q} \times \hat{\Sigma} \times \{L, R\}\Big) \longrightarrow \mathbb{S}$ Finally, the composition of these injection with the one described by Corollary 4 yields the desired injection. □

With this result we have shown that we can encode all the information contained in a SFTM transition function in a string. Together with the previous result, we need to state that even all the possible machine configuration can be encoded by means of strings. Namely, that there exists an injection from an exhaustive representation of a Finite Stream Turing Machine's configuration and the set of strings because we want to simulate a Stream Machine M recurring to its configuration. As we argued before, we do not want a complete description of the machine's configuration due to the fact that a configuration contains a total and generic function $\mathbb{N} \longrightarrow \mathbb{S}$.

Corollary 6 ((Partial) Representability of Machine's Configurations)**.** *There is an injection* $f : (\hat{\Sigma}^* \times \mathbb{Q} \times \hat{\Sigma}^*) \longrightarrow \mathbb{S}$*.*

Proof. As for the proof of Corollary 5, we observe that there is an injection $i_2 : \mathbb{Q} \longrightarrow \mathbb{N}$. Since $\hat{\Sigma}$ is a finite set, it is countable too, so there is a bijection $i_3 : \hat{\Sigma} \longrightarrow \mathbb{N}$, which composed with $\underline{\cdot}_{\mathbb{N}}$ yields another injection $\hat{\Sigma} \longrightarrow \mathbb{S}$. There is another injection $\underline{\cdot}_{\mathbb{T}} : \hat{\Sigma}^* \longrightarrow \mathbb{L}$ defined as follows:

$$\underline{\cdot}_{\mathbb{T}}(\epsilon) := \langle \underline{i_3(\circledast)}_{\mathbb{N}} \rangle^n_{\mathbb{L}}$$
$$\underline{\cdot}_{\mathbb{T}}(c_0 c_1 \dots c_n) := \langle \underline{i_3(c_n)}_{\mathbb{N}}, \dots, \underline{i_3(c_1)}_{\mathbb{N}}, \underline{i_3(c_0)}_{\mathbb{N}} \rangle^n_{\mathbb{L}}$$

In other words, for each $\sigma \in \hat{\Sigma}^*$, it suffices to take the list which contains in position k the mapping of the k-th element of σ through $\underline{\cdot}_{\mathbb{T}}$, and as encoding of the empty tape a list with a single instance of the blank character. We took this decision for a technical reason: the rightmost and the leftmost characters in an empty tape exist and are $\circledast$, this will turn out to be useful in the definition of the machine's simulation in $\mathcal{POR}^-$. We also know that there is an injection $\langle\cdot\rangle^n_{\mathbb{L}} : \mathbb{S}^{|\sigma|} \longrightarrow \mathbb{S}$. The composition of the aforementioned injections with $\langle\cdot\rangle^n_{\mathbb{L}}$ yields the claimed one. □

We are interested in capturing $\mathcal{SFP}$ so, before going further, we want to instantiate encodings described in Corollaries 6 and 5 for the class of the Canonical Stream Machines.

Definition 35 (Encodings for Canonical Stream Machines)**.** *The Encodings for Canonical Stream Machines are defined from the one described in Corollaries 6 and 5, choosing in particular:*

- $q_i \mapsto i$ *for* i_2.
- $L \mapsto \mathbf{0}, R \mapsto \mathbf{1}$.
- *For* i_3: $\mathbf{0} \mapsto 0, \mathbf{1} \mapsto 1, \circledast \mapsto 2$

4.1.2. *Dynamic encoding.* In this Section we will introduce some manipulating functions in $\mathcal{POR}^-$ for the data encodings defined in the previous Section. The functions introduced in this section, we can easily define complex program behaviors, such as the implementation in $\mathcal{POR}$ of any FSTM.
Boolean Algebra. Since we choose to use $\mathbf{0}$ and $\mathbf{1}$ to represent boolean values, it's natural to represent predicates with functions $\mathbb{S}^n \times \mathbb{O} \longrightarrow \mathbb{B}$. Given two values, x_1 and x_2, we can define a function that returns x_1 if a condition y is met, x_2 if such condition is false, and ϵ otherwise. Such function behaves as an `if`-expression.

Definition 36 (`if`)**.** *The* `if` *expression is defined as follows:*

$$\texttt{if}(x_2, x_1, y, \omega) := C(y, \epsilon, x_1, x_2, \omega)$$

Remark 6. `if` $\in \mathcal{POR}^-$

Proof. Trivial: the definition of the function is given recurring to C. □

From now on, we will omit similar proofs. Thanks to the `if` function we can now also easily define some basic connectives of classical Propositional Logic.

Definition 37 (Logical Connectives)**.** *Conjunction, disjunction and negation are defined as follows:*

$$\begin{aligned}(P_1 \wedge P_2)(\vec{x}, \omega) &:= \texttt{if}(P_2(\vec{x}, \omega), \mathbf{0}, P_1(\vec{x}, \omega), \omega)\\ (P_1 \vee P_2)(\vec{x}, \omega) &:= \texttt{if}(\mathbf{1}, P_2(\vec{x}, \omega), P_1(\vec{x}, \omega), \omega)\\ (\neg P)(\vec{x}, \omega) &:= \texttt{if}(\mathbf{0}, \mathbf{1}, P(\vec{x}, \omega), \omega).\end{aligned}$$

Now we define some predicates which are helpful in the development of the list mainpulators. In particular, in order to show that *halv* $\in \mathcal{POR}^-$, it will be useful to show that there is a predicate *odd* $\in \mathcal{POR}^-$ which is $\mathbf{1}$ if and only if the length of the remaining part of such value is even or odd, depending on the value of this predicate, proceeding by iteration it will be possible to define whether to keep or remove a certain bit.

Definition 38. *Basic logical predicates in $\mathcal{POR}^-$ are defined by the functions below:*

$$\begin{aligned} odd(\epsilon,\omega) &:= \mathbf{0}\\ odd(y\mathbf{b},\omega) &:= \neg\big(even(y)\big)|_{\mathbf{0}}\\ even(x,\omega) &:= \neg\big(odd(x,\omega)\big)\\ eq(x,y,\omega) &:= S(x,y,\omega)\wedge S(y,x,\omega)\end{aligned}$$

Remark 7. *Observe that the odd and even predicates work as their opposites for the encoding of natural numbers:*

$$\forall\sigma\in\mathbb{S}.\forall\mathbf{b}\in\mathbb{B}.odd(\sigma)\leftrightarrow even(\sigma b)$$
$$\forall n\in\mathbb{N}.odd(\underline{n}_{\mathbb{N}})\leftrightarrow n \textit{ is even.}$$

Finally, encoding more advanced data manipulators, such as list manipulators, it will often be the case to test the rightmost or leftmost bit of a string. In order to do so, we need to define two other predicates:

Definition 39. *Given a string, predicates*

- *res, i.e. Right Most Element String*
- *les, i.e. Left Most Element String*

are defined as follows:

$$\begin{aligned} res(\epsilon,\omega) &:= \epsilon\\ res(y\mathbf{0},\omega) &:= \mathbf{0}|_{\mathbf{1}}\\ res(y\mathbf{1},\omega) &:= \mathbf{1}|_{\mathbf{1}}\end{aligned}$$

$$\begin{aligned} les(\epsilon,\omega) &:= \epsilon\\ les(y\mathbf{0},\omega) &:= \mathtt{if}\big(\mathbf{0}, les(y,\omega), C(y,\epsilon,\omega),\omega\big)|_{\mathbf{1}}\\ les(y\mathbf{1},\omega) &:= \mathtt{if}\big(\mathbf{1}, les(y,\omega), S(y,\epsilon,\omega),\omega\big)|_{\mathbf{1}}.\end{aligned}$$

Remark 8.

$$\forall\sigma\in\mathbb{S}.res(\sigma\mathbf{1})=\mathbf{1}\wedge res(\sigma\mathbf{1})=\mathbf{1}$$
$$\forall\sigma\in\mathbb{S}.les(\mathbf{1}\sigma)=\mathbf{1}\wedge res(\mathbf{0}\sigma)=\mathbf{0}.$$

Strings. The Stream Machines' reachability function is defined on top of configurations, in particular, the strings which represent the tape are obtained from the strings in the initial configuration, applying some of the following transformations:

- Appending on the left or on the right.
- Deleting a character on the left or on the right.

Definition 40 (String Appenders and Removers)**.** *We define string appenders*

- *ras, i.e.* Right Appender for Strings
- *las, i.e.* Left Appender for Strings
- *rrs, i.e.* Right Remover for Strings
- *lrs, i.e.* Left Remover for Strings

as follows:

$$\begin{aligned}
\mathit{reverse}(\epsilon,\omega) &:= E(x,\omega)\\
\mathit{reverse}(x\mathbf{b},\omega) &:= \mathit{concat}(\mathbf{b},\mathit{reverse}(x,\omega),\omega)\\
\mathit{ras}(x,y,\omega) &:= \mathit{concat}(x,y,\omega)\\
\mathit{las}(x,y,\omega) &:= \mathit{reverse}(\mathit{ras}(\mathit{reverse}(x,\omega),\omega),\omega)\\
\mathit{rrs}(\epsilon,\omega) &:= E(x,\omega)\\
\mathit{rrs}(x\mathbf{b},\omega) &:= x|_x\\
\mathit{lrs}(\epsilon,\omega) &:= \mathit{reverse}(\mathit{rrs}(\mathit{reverse}(x,\omega),\omega),\omega)
\end{aligned}$$

Natural Numbers. The execution of a FSTM in $\mathcal{POR}^-$ can modelled with the self application of the machine's transition function to itself a polynomial number of times. The only iteration construct of $\mathcal{POR}^-$ is bounded recursion, so, to iterate a function a polynomial number of step, we must be able to compute a string whose size is polynomial in the size of the input. For this reason, we defined the encodings of numbers in a way that their size is linear in the size of the encoded number. This will allow us to use them directly as iteration parameters for the bounded recursion schema. Finally, we said that the time bound of a SFTM is polynomial, so to iterate a function a polynomial number of steps, we could simply compute reduce the task to the computation of polynomial values in our encoding of naturals.

Definition 41 (Successor)**.** *We define a successor function $S : \mathbb{S}_{\mathbb{N}} \longrightarrow \mathbb{S}_{\mathbb{N}}$ that computes the successor of a number:*

$$\begin{aligned}
S(\epsilon,\omega) &:= \epsilon\\
S(y\mathbf{b},\omega) &:= S_{\mathbf{1}}(y,\omega)|_{y\mathbf{1}}
\end{aligned}$$

Definition 42 (Predecessor)**.** *If $y \in \mathbb{S}_{\mathbb{N}}$ is the encoding of a number, the pd function calculates its predecessor y simply removing its last digit (if present):*

$$\begin{aligned}
\mathit{pd}(\epsilon,\omega) &:= \epsilon\\
\mathit{pd}(y\mathbf{b},\omega) &:= y|_y
\end{aligned}$$

Remark 9. $\forall\sigma, c_1, c_2.\mathit{pd}(\mathit{pd}(\sigma c_1 c_2)) = \sigma$.

Proof. The claim is a trivial consequence of Definition 42. □

Definition 43 (Sum)**.** *The sum of two numbers sum is implemented using the operation of concatenation concat introduced in Definition 32 as:*

$$\mathit{sum}(x,y,\omega) := \mathit{pd}(\mathit{concat}(x,y,\omega),\omega)$$

.

The $\mathcal{POR}^-$-encoding of the difference between two numbers is cumbersome because it recurs on the definition of the *sa* functor. Intuitively, it applies *rrs* n times to x, where $\underline{n}_{\mathbb{N}}$ is y.

Definition 44 (Difference)**.** *The function diff, which computes the difference between two natural numbers, is defined as follows:*

$$\mathit{diff}(x,y,\omega) := \mathit{sa}_{\mathit{rrs},x}(x,y,\omega)$$

Remark 10. *The function diff is in $\mathcal{POR}^-$ and follows its intended semantics.*

Proof. Lemma 5 states that the function $\mathit{sa}_{\mathit{rrs},x}$ is in $\mathcal{POR}^-$ and that $\mathit{sa}_{\mathit{rrs},x}(x,\underline{n}_{\mathbb{N}},\omega) = \mathit{rrs}^n(x,\omega)$ this is sufficient to prove the claim, in accoradance with the definition of *rrs* (Definition 40). □

In order to multiply two values x, y, we can remove their last digit – so that their size is equal to the number that they encode – concatenate x to itself y-times and return the successor of the number we get.

Definition 45 (Multiplication)**.** *Multiplication between two natural numbers, mult, is defined as follows:*

$$\begin{aligned} \mathit{mult}^*(x, \epsilon) &:= \epsilon \\ \mathit{mult}^*(x, y\mathbf{b}) &:= \big(\mathit{sum}(\mathit{mult}(x, y, \omega)x, \omega)\big)|_{x\times y\mathbf{1}} \\ \mathit{mult}(x, y, \omega) &:= S\big(\mathit{mult}^*(\mathit{pd}(x, \omega), \mathit{pd}(y, \omega))\big). \end{aligned}$$

With this encoding the computation of an exponential function would require an exponential size for the representation of the output, but our iteration is bounded by a term $\mathcal{L}$ and the size of the number in our encoding is linear in its value[9]. Nevertheless, we can show how to compute an exponentiation, and so how to represent monomials and polynomials.

Definition 46 (Exponentiation)**.** *Given a $k \in \mathbb{N}$, the function computing the k-th exponentiation of such value is defined as follows:*

$$\begin{aligned} @0(x, \omega) &:= \underline{1}_{\mathbb{N}} \\ @(n+1)(x, \omega) &:= \mathit{mult}(@n(x), x) \end{aligned}$$

Notice that all these functions, which are clearly in $\mathcal{POR}$, behave as desired. Indeed the following claims are easily verifiable:

- For any $n, m \in \mathbb{N}$ and $\omega \in \mathbb{O}$, $\mathit{succ}(\underline{n}_{\mathbb{N}}, \omega) = \underline{n+1}_{\mathbb{N}}$
- For any $n \in \mathbb{N}^+$, $\mathit{pd}(\underline{n}_{\mathbb{N}}, \omega) = \underline{n-1}_{\mathbb{N}}$
- For any $n, m \in \mathbb{N}$, $\mathit{sum}(\underline{n}_{\mathbb{N}}, \underline{m}_{\mathbb{N}}, \omega) = \underline{n+m}_{\mathbb{N}}$
- For any $n, m \in \mathbb{N}$, such that $n \geq m$, $\mathit{diff}(\underline{n}_{\mathbb{N}}, \underline{m}_{\mathbb{N}}, \omega) = \underline{n-m}_{\mathbb{N}}$.
- For any $n, m \in \mathbb{N}$, $\mathit{mult}(\underline{n}_{\mathbb{N}}, \underline{m}_{\mathbb{N}}, \omega) = \underline{n \cdot m}_{\mathbb{N}}$
- For any $n, m \in \mathbb{N}$, $@m(\underline{n}_{\mathbb{N}}, \omega) = \underline{n^m}_{\mathbb{N}}$.

Lists. In the previous Section, we reduced all the composed data structures to the case of lists. For this reason, it comes natural to develop a tool-set of functions which can be used to manipulate lists, which will be used as building-blocks to define the function simulator. We start showing that the list member representation can be decoded in $\mathcal{POR}^-$, namely that $\mathit{halv} \in \mathcal{POR}^-$.

Lemma 12. *There is a function halv in $\mathcal{POR}^-$ such that $\forall x \in \mathbb{S}.\forall \omega \in \mathbb{O}.\mathit{halv}(x, \omega) = \mathcal{H}(x)$*

Proof. It can be shown giving the following definition:

$$\begin{aligned} \mathit{halv}(\epsilon, \omega) &:= \epsilon \\ \mathit{halv}(y\mathbf{0}, \omega) &:= \mathit{concat}\big(\mathit{halv}(y, \omega), \mathtt{if}(\mathbf{0}, \epsilon, \mathit{odd}(y, \omega), \omega)\big)|_{y\mathbf{0}} \\ \mathit{halv}(y\mathbf{1}, \omega) &:= \mathit{concat}\big(\mathit{halv}(y, \omega), \mathtt{if}(\mathbf{1}, \epsilon, \mathit{odd}(y, \omega), \omega)\big)|_{y\mathbf{0}}. \end{aligned}$$

□

Moreover, we ought show that it's the left-inverse of *doub*, as stated by the following lemma.

Lemma 13 (Left-Inverse of *doub*)**.** $\forall \sigma \in \mathbb{S}, \omega \in \mathbb{O}.\mathit{halv}\big(\mathit{doub}(\sigma, \omega), \omega\big) = \sigma$.

Proof. The proof is by (right) induction on σ.

[9]We prove such result in Lemma **??**

ϵ. The thesis comes from a trivial rewriting of the two functions' bodies.

τc. The thesis is

$$\begin{aligned} halv\big(doub(\tau c, \omega), \omega\big) &= \tau c \\ halv\big(doub(\tau, \omega)\mathbf{1}c, \omega\big) &= \tau c. \end{aligned}$$

By induction on σ we can also prove that $\forall \sigma. odd\big(doub(\sigma)\big) = \mathbf{0}$. So we can simplify our claim as follows:

$$\begin{aligned} halv\big(doub(\tau, \omega)\mathbf{1}c, \omega\big) &= \tau c \\ halv\big(doub(\tau, \omega)\mathbf{1}, \omega\big)c &= \tau c \\ halv\big(doub(\tau, \omega)\mathbf{1}, \omega\big) &= \tau. \end{aligned}$$

We argued that $\forall \sigma. odd\big(doub(\sigma)\big) = \mathbf{0}$. So, we can state the claim as:

$$\begin{aligned} halv\big(doub(\tau, \omega)\mathbf{1}, \omega\big) &= \tau \\ halv\big(doub(\tau, \omega), \omega\big) &= \tau \end{aligned}$$

which is IH.

□

Thanks to the previous result, we can leverage *halv* to define list appenders, extractors and removers.

Definition 47 (List Extractors, Appenders and Removers)**.** *We define lists' manipulator*

- *rel, i.e.* Right Appender for Lists
- *lel, i.e.* Left Appender for Lists
- *ral, i.e.* Right Appender for Lists
- *lal, i.e.* Left Appender for Lists
- *rrl, i.e.* Right Remover for Lists
- *lrl, i.e.* Left Remover for Lists

and the auxiliary functions

- *rmsep, i.e. the functions which removes an encoded character.*
- *rel', i.e. the functions which returns the rightmost element of a list (size included).*
- *srrl, i.e. the right semi-remover, which given a string shaped as $\sigma\mathbf{00}\tau\mathbf{00}$ returns $\sigma\mathbf{00}$ where σ is as long as possible. This function is dual with respect to rel.*

Starting from the auxiliary functions, we define them as:

$$\begin{aligned} rmsep(x, \omega) &:= rrs(rrs(x, \omega), \omega) \\ rel'(\epsilon, \omega) &:= E(x, \omega) \\ rel'(y\mathbf{0}, \omega) &:= \mathtt{if}(\epsilon, concat(rel'(y, \omega), \mathbf{0}), (\neg res(y, \omega)) \wedge odd(y, \omega), \omega)|_{y\mathbf{0}} \\ rel'(y\mathbf{1}, \omega) &:= concat(rel'(y, \omega), \mathbf{1}, \omega))|_{y\mathbf{0}} \\ srrl'(\epsilon, \omega) &:= E(x, \omega) \\ srrl'(y\mathbf{0}, \omega) &:= \mathtt{if}(y\mathbf{0}, srrl'(y, \omega), \neg(res(y, \omega)) \wedge odd(y, \omega), \omega)|_{y\mathbf{0}} \\ srrl'(y\mathbf{1}, \omega) &:= srrl'(y, \omega)|_{y\mathbf{0}} \\ srrl(x, \omega) &:= srrl'(rmsep(x, \omega), \omega) \end{aligned}$$

$$\begin{aligned}
rel(t,\omega) &:= halv(rel'(rmsep(t),\omega),\omega)\\
lel(x,\omega) &:= reverse(rel(reverse(x,\omega),\omega),\omega)\\
ral(x,y,\omega) &:= srrl(x,\omega)\mathbf{00}doub(y,\omega)\mathbf{00}doub(rel(x,\omega)\mathbf{1},\omega)\mathbf{00}\\
lal(x,y,\omega) &:= \mathbf{00}doub(y,\omega)srrl(y,\omega)doub(rel(x,\omega)\mathbf{1},\omega)\mathbf{00}\\
rrl(x,\omega) &:= srrl(srrl(x,\omega),\omega)doub(rrs(rel(x,\omega),\omega)\omega)\mathbf{00}\\
lrl(x,\omega) &:= srrl(reverse(srrl(reverse(x,\omega),\omega),\omega)doub(rrs(rel(x,\omega),\omega)\omega)\mathbf{00}
\end{aligned}$$

Remark 11 (Correctness of lists operators)**.** *Lists operators of Definition 47 follow their intended specification.*

Proof. Respectively:

rmsep Trivial, comes from the definition od *rrs*.

rel′ Suppose that $y = \{\mathbf{10},\mathbf{11}\}^*\mathbf{00}\{\mathbf{10},\mathbf{11}\}^*$. We will not take in account the case E because we are not interested in it: this function will always be invoked with non-empty inputs. So we go by cases on the input assuming that it has the shape described above. If the input is $y\mathbf{0}$, we continue by cases on the guard of the `if` function: $\neg(res(y,\omega))$ entails that $y = y'\mathbf{0}$, so the input of the function is $y'\mathbf{00}$ finding two consecutive $\mathbf{0}$s means that we reached the separator, indeed the output is the same of *rel′* is its input. If the guard is not true, the $\mathbf{0}$ at the end of the input is the second character of an encoded $\mathbf{0}$ in the rightmost value; even if the input is $y\mathbf{1}$, the current character is the encoding of a value, so we get the claim by induction.

srrl′ We have already argued that this function is the dual of the *rel′* function. The proof is a slight variation of the previous.

srrl Trivial by the proof of the correctness of *srrl′*

rel The claim is a consequence of the correctness of *rel′* and of Lemma 13.

lel The claim comes from the fact that the definition of *rel* checks that a sequence of two characters is found and that the current character is in even position (the remaining part of the encoding has odd length) and that

- By induction on σ we can show that $\forall\sigma.doub(\sigma)$ has odd length.
- If σ is the encoding of a list, the separators $\mathbf{00}$ are always followed by an even number of characters. This can be show by induction on the number of elements of the list.

ral, *lal* The correctness comes from the definition of lists' encoding and from the correctness of all the functions which appear in the definitions of *ral* and *lal*.

rrl The correctness comes from the definition of lists' encoding and from the correctness of all the functions composed in the definition.

lrl The correctness comes from the definition of lists' encoding and from the fact that *srrl*, as *rel* does, is safe with respect to the reversing of a string.

□

4.2. Section 1.

Proof of Proposition 1. The proof is by induction on n.

$n = 0$. Then, $\triangleright^n_M$ is the identity function.

$n+1$. By IH $\triangleright^n_M$ is a function. Since $\vdash_\delta$ is a function, $\triangleright^{n+1}_M = \triangleright^n_M \circ \vdash_\delta$ is a function too. □

4.3. Section 2.1.

Defining Finite Stream Turing Machine.

Definition 48 (FSTM Configuration)**.** *The* configuration of a FSTM *is a 4-tuple* $\langle \sigma, q, \tau, \xi \rangle$*, where*

- $\sigma \in \hat{\Sigma}^*$ *is the portion of the work tape on the left of the head*
- $q \in Q$ *is the current state of the machine*
- $\tau \in \hat{\Sigma}^*$ *is the portion of the work tape on the right of the head.*
- $\sigma \in \hat{\Sigma}^*$ *is the portion of the secondary tape on the right of the head.*

Now, it is possible to define the TM's transition function.

Definition 49 (FSTM Reachability Function)**.** *Given a TM,* $M = \langle Q, q_0, \Sigma, \delta \rangle$*,* $\{\triangleright^n_M\}_n$ *is the smallest family of relations such that:*

$$\langle \sigma, q, \tau, \xi \rangle \triangleright^0_M \langle \sigma, q, \tau, \xi \rangle$$
$$(\langle \sigma, q, \tau, \xi \rangle \triangleright^n_M \langle \sigma', q', \tau', \xi' \rangle) \wedge (\langle \sigma', q', \tau', \xi'' \rangle \vdash_\delta \langle \sigma'', q'', \tau'', \xi'' \rangle) \rightarrow (\langle \sigma, q, \tau, \xi \rangle \triangleright^{n+1}_M \langle \sigma'', q'', \tau'', \xi'' \rangle)$$

Without loss of generality, we assume TMs not to use final states: computation is regarded as concluded whenever the current configuration does not define the transition function.[10]

Proposition 2. *For any FSTM,* $M\langle Q, q_0, \Sigma, \delta \rangle$ *and* $n \in \mathbb{N}$*,* $\triangleright^n_M$ *is a function.*

Notation 2 (Final Configuration)**.** *Given a FSTM,* $M = \langle Q, q_0, \Sigma, \delta \rangle$*, and a configuration* $\langle \sigma, q, \tau, \xi \rangle$*, we write* $\langle \sigma, q, \tau, \xi \rangle \not\vdash_\delta$ *when there are no* σ', q', τ', ξ' *such that* $\langle \sigma, q, \tau, \xi \rangle \vdash_\delta \langle \sigma', q', \tau', \xi' \rangle$*.*

Lemma 14. *Each for each ordinary poly-time computable function* f *there is a function in* $\mathcal{PTF}_\mathbb{S}$ g *such that:*

$$\forall x_1, x_2. f(x_1) = g(x_1, x_2)$$

Proof. Given the (canonical) Turing Machines which computes f, $M_f = \langle Q, q_0, \Sigma, \delta \rangle$, we can define the Finite State Turing Machine which computes g as $N = \langle Q, q_0, \Sigma, \Delta(\delta) \rangle$ where the functional Δ basically extends all the transitions of the original machine in order to match each of the characters on the secondary tape. Formally

$$\Delta(\delta) := \bigcup \{\{\langle p, c_r, \mathbf{0}, q, c_w, d \rangle, \langle p, c_r, \mathbf{1}, q, c_w, d \rangle, \langle p, c_r, \circledast, q, c_w, d \rangle | \langle p, c_r, q, c_w, d \rangle \in \delta\}$$

It can be shown by induction on the steps of the ordinary Turing Machine that the Finite Stream Turing Machine N (restricted to its work tape), behaves exactly as the Turing Machine M. □

The Class $\mathcal{POR}^-$. Following [?] and [?], we introduce the class $\mathcal{PF}_\mathbb{S}$ which will be proved to precisely characterize string-functions computable in polynomial time.

Definition 50 (The Class $\mathcal{PF}_\mathbb{S}$)**.** *The class* $\mathcal{PF}_\mathbb{S} : \mathbb{S}^n \longrightarrow \mathbb{S}$ *is the smallest class containing initial functions:*

- $E_\mathcal{D}(x) = \emptyset$
- $P^i_\mathcal{D}(x_1, \ldots, x_n) = x_i$*, for* $1 \leq i \leq n$
- $C^\mathbf{0}_\mathcal{D}(x) = x\mathbf{0}$ *and* $C^\mathbf{1}_\mathcal{D}(x) = x\mathbf{1}$
- $Q_\mathcal{D}(x, y) = \mathbf{1}$ *iff* $x \subseteq \mathbf{0}$*,* $Q_\mathcal{D}(x, y) = \mathbf{0}$ *otherwise*

[10]Indeed, in all these cases, we could imagine to add a final state q_F and a transition to it.

and closed under composition and bounded recursion, where a function f is defined by bounded recursion from g, h_0, h_1 as:

$$\begin{aligned} f(x_1,\dots,x_n,\epsilon) &= g(x_1,\dots,x_n) \\ f(x_1,\dots,x_n,y\mathbf{0}) &= h_0(x_1,\dots,x_n,y,f(x_1,\dots,x_n,y))|_{t(x_1,\dots,x_n,y)} \\ f(x_1,\dots,x_n,y\mathbf{1}) &= h_1(x_1,\dots,x_n,y,f(x_1,\dots,x_n,y))|_{t(x_1,\dots,x_n,y)}. \end{aligned}$$

with t term of $\mathcal{L}$.

Remark 12. $\forall f \in \mathcal{PF}_{\mathbb{S}}.\exists g_f \in \mathcal{POR}^-.\forall \omega, \vec{x}.f(\vec{x}) = g_f(\vec{x},\omega)$

Proof. By induction on the proof that a function is in Ferreira's $\mathcal{PF}_{\mathbb{S}}$.

- If the function is $E_{\mathcal{D}}$, E respects the property.
- If the function is $P^n_{j\ \mathcal{D}}$, the correspondent $\mathcal{POR}$ projector respects the claim.
- If the function is $C_{\mathbf{b}\mathcal{D}}$, $S_{\mathbf{b}}$ verifies the claim.
- $Q_{\mathcal{D}}$ can be encoded by means of bounded recursion on notation, as shown in **??**
- The composition and recursion on notation cases follow by inductive hypothesis.

□

Remark 13. $\forall g \in \mathcal{POR}^-.\exists f_g \in \mathcal{PF}_{\mathbb{S}}.\forall \omega, \vec{x}.g(\vec{x},\omega) = f_g(\vec{x})$

Proof. By induction on the proof that $g \in \mathcal{POR}$

- If the function is E, $E_{\mathcal{D}}$ is a witness for the existential.
- If the function is P^n_j , the correspondent $\mathcal{PF}_{\mathbb{S}}$ projector has the claimed property.
- If the function is $S_{\mathbf{b}}$, $C_{\mathbf{b}\mathcal{D}}$ verifies the claim.
- C can be encoded by means of bounded recursion on notation, taking:

$$\begin{aligned} C(\epsilon, z_\epsilon, z_1, z_2, \omega) &:= z_\epsilon \\ C(x\mathbf{b}, z_\epsilon, z_1, z_2, \omega) &:= x\mathbf{b}|_{x\mathbf{1}} \end{aligned}$$

- The composition and recursion on notation cases follow by inductive hypothesis.

□

4.4. Section 2.3.

Lemma 15. *If γ is a function $\mathbb{S} \longrightarrow \mathbb{S}$ such that $|\gamma| = n$, define $x_\gamma = \langle\langle x_1, y_1\rangle^n_{\mathbb{L}}, \dots, \langle x_n, y_n\rangle^n_{\mathbb{L}}\rangle^n_{\mathbb{L}}$, then $sim(x, x_\gamma, \omega) = \gamma(x)$, otherwise $sim(x, x_\gamma, \omega) = \epsilon$.*

Proof. Suppose x' is in γ's domain. It means that there is a tuple $t = \langle x', \overline{y}\rangle^n_{\mathbb{L}}$ in x_γ, so there exists $k \in \mathbb{N}$ such that $t = \pi_k(x_\gamma, \omega)$. Thus we can prove that if the third argument of sim' is the number of elements in t, then $\forall k \leq m < |t|.sim'(x_\gamma, x, \underline{m}_{\mathbb{N}}, \omega) = \overline{y}$; this result can be proven by induction on m. Then, the correctness of sim comes as consequence because it is an instance of $simulate'$ with a value which is greater or equal to any possible value of k, namely the number of pairs in x_γ. Now suppose that x' is not in γ's domain, then there will not be any k such that $\pi_k(x_\gamma) = \langle x', \overline{y}\rangle^n_{\mathbb{L}}$, so sim' will return ϵ. □

Definition 51 (Apply)**.** *The function $apply(\cdot,\cdot,\omega) \in \mathcal{POR}^-$ is defined as follows:*

$$apply(x, y, \omega) := \texttt{if}\big(x, f_{aux1}(x, y), Neg(eq(\xi(x, y, \omega), \epsilon, \omega), \omega), \omega\big)$$

where:

$$apply(x, y, \omega) = \texttt{if}\big(x, f_{aux1}(x, y), \neg(eq(\xi(x, y, \omega), \omega)\mathbf{0}, \omega\big)$$

$$\begin{aligned}
f_{a1}(x, y, \omega) = \langle &\chi(rrl(\pi_1(x, \omega), \omega), \omega), \pi_3(\xi(x, y, \omega), \omega), \\
&lal(lal(lrl(\pi_3(x, \omega), \omega), \pi_1(\xi(x, y, \omega)), \omega), rel(\pi_1(x, \omega), \omega), \omega) \\
&lel(\pi_4(x, \omega), \omega)\rangle_{\mathbb{L}} \\
f_{a2}(x, y, \omega) = \langle &ral(rrl(\pi_1(x, \omega), \omega), \pi_1(\xi(x, y, \omega), \omega), \omega), \pi_3(\xi(x, y, \omega), \omega), \\
&\chi(lrl(\pi_3(x, \omega), \omega), \omega), lel(\pi_4(x, \omega), \omega)\rangle_{\mathbb{L}}
\end{aligned}$$

$$\begin{aligned}
\chi(x, \omega) &= \texttt{if}\big(\underline{\circledast}_{\mathbb{T}}, x, eq(\pi_0(x, \omega), \underline{0}_{\mathbb{N}}, \omega), \omega\big) \\
\xi(x, y, \omega) &= sim\big(y, \langle \pi_2(x, \omega), lel(\chi(\pi_3(x, \omega), \omega), \omega), lel(\chi(\pi_4(x_c, \omega), \omega), \omega)\big)
\end{aligned}$$

Proof. It should not be too much of a problem to see that the function χ return a tape which contains an instance of the blank character if and only if the tape passed as argument is empty, otherwise it returns its agument. Thanks to this observation, we prove that ξ is defined as the invoking of *sim* on the tuple which contains:

(1) the current state $\pi_2(x_c, \omega)$
(2) current character $lel(\chi(\pi_3(x_c, \omega), \omega), \omega)$
(3) the first character on the second tape oracle bit $lel(\chi(\pi_3(x_c, \omega), \omega), \omega)$[11]

to *sim* which returns either the encoding of the image of the tuple through δ (if defined), or ϵ as stated by Lemma 15. Suppose that the result of ξ is ϵ, then the claim is trivially true, since the function returns exactly its input, but with the counter increased by one, as required by the claim. Otherwise, suppose that ξ returns a value different from ϵ. It means that δ is defined on its input, so, the guard of the inner `if` expression checks whether the result describes a right or left movement of the head and, depending on the response, behaves differently. Observe that the constant L is represented by $\mathbf{0}$ as stated in Definition 35. We will show only the case in which the head moves right, for the left movement the proof is analogous. The configuration is made up as follows:

(1) The left portion of the tape tape $ral(rrl(\pi_1(x_c, \omega), \omega), \pi_1(\chi(x_c, x_\delta, \omega), \omega))$ is obtained removing the current character by means of $rrl(\pi_1(x_c, \omega), \omega)$, appending the character which is written by the head $\pi_1(\chi(x_c, x_\delta, \omega), \omega)$ by means of *ral*.
(2) The current state is obtained projecting the third element of χ.
(3) The right prortion tape loses its leftmost element, becoming $\underline{\circledast}_{\mathbb{T}}$ if empty, as described by $\rho(lrl(\pi_3(x_c, \omega), \omega), \omega)$.
(4) the second tape loses its leftmost character.

$\square$

Proof of Lemma 5. Given $f \in \mathcal{POR}^-$ and $t \in \mathcal{L}_{\mathbb{PW}}$, let $sa_{f,t}$ be defined as follows:

$$\begin{aligned}
sa'_{f,t}(x, \epsilon, \vec{z}, \omega) &:= x \\
sa'_{f,t}(x, y\mathbf{b}, \vec{z}, \omega) &:= f\big(sa'_{f,t}(x, y, \omega), \vec{z}, \omega\big)|_t
\end{aligned}$$

$$sa_{f,t}(x, y, \vec{z}, \omega) := sa'_{f,t}(x, rrs(y, \omega), \vec{z}, \omega).$$

11

The function sa is correct as $\underline{n}_{\mathbb{N}}$ has size $n+1$. We prove that if $|y| = n$, then $sa'_{f,t}(x, y, \vec{z}, \omega) = f(f(f(\vec{x}, \vec{z}, \omega), \vec{z}, \omega)\dots)$, nested n times, by induction on n.

- If $n = 0$, $sa_{f,t}$ reduces to $sa'_{f,t}$ with argument ϵ, so the result is x.
- $n + 1$. By applying IH and the definition of $sa'_{f,t}$.

The correctness of sa comes as a consequence of the correctness of the function rrs (which removes the rightmosr digit of a string) and of the definition of sa.

□

Proof of Lemma 6. At each step, the function *apply* manipulates three values:

- The portion of the tape on the left of the head.
- The portion of the tape on the right of the head.
- the current state

The manipulation of the tapes is done through shifting of characters and rewriting. An ordinary shifting operation does not change the whole size of the enoded configuration, while the rewriting can cause such phenomenon, due to the different sizes of the encoded characters, but fixed a machine, the maximum size of its characters is fixed. Say that value $k_1 \in \mathbb{N}$. Replacing a character on the encoded tape with it takes less than $2 \cdot k_1$ bits. Even if a tape contains an infinite sequence of $\circledast$ and the machine moves in that direction, the overall result is the appending of a character on the other portion of the tape this takes exactly $2 \cdot k_1$ bits, plus $\mathbf{1}^2$ additional bits to store the new size of the portion of the tape which receives an additional bit. Then, we must take in account the difference in size due to the new state, but since the numbers of state in a machne is fixed, there is a constant k_2 which bounds the size of any state, so rewriting a state takes less than $2 \cdot k_2$ additional bits. Finally, we must take in account the representation of the secondary tape: at each step, we strip off a cell from its representation since the head moves always to the right, so it causes no term growth. The value of k in the claim is $2 \cdot (1 + k_1 + k_2)$.

□

Lemma 16. *There's a function $f : \mathbb{S} \longrightarrow \mathbb{S}$ in $\mathcal{POR}$ such that if $\underline{\cdot}_{\mathbb{T}}$ is the encoding for tapes proposed in Corollary 35, then it holds that $\forall \sigma \in \{\mathbf{0}, \mathbf{1}, \circledast\}^*, \omega. f(\underline{\sigma}_{\mathbb{T}}, \omega) = \tau$ and τ is the longest suffix of σ without $\circledast$.*

Proof of Lemma 16. Let us consider $dectape(x, \omega)$. If x is the encoding of a tape through $\underline{\cdot}_{\mathbb{T}}$, $x = \underline{n}_{\mathbb{N}}$ and $n \leq |x|$, so $\rho(x, y, \omega)$ returns the n-th projection of x from the right. This is a consequence of the correctness of the function used in the definition of ρ, which has already been shown. Before, we ought prove that *dectape'* is a generalization of *dectape*, for which $y = \underline{n}_{\mathbb{N}}$, then $\underline{\sigma}_{\mathbb{T}} = x$ then $dectape'(x, y, \omega)$ is the longest substring of σ without $\circledast$, which ends in the n-th character of σ from the left. The proof is by induction on n.

- Case 0. Intuitively ρ checks the right-most character of the encoding for σ. Moreover, the encoding of any encoded $\sigma \in \hat{\Sigma}^*$ contains at least one character. If the character is $\circledast$, its encoding will be $\mathbf{111}$, so the condition in the guard of $f_i f$ is verified and the whole expression reduces to ϵ. Otherwise, the function returns the decoding of the remaining part, which reduces to ϵ, and the decoded value of the character in exam.
- Case $n + 1$. The claim follows from IH on the recursive call and the condition above concerning the value of the projected character.
  We conclude the proof by joining the consideration above and the fact that the definition of *dectape* is defined as an instance of *dectape'* with x equal to the size of σ. So, the substring returned by *dectape'* is actually a suffix.

□

4.5. **Section 2.4.**

Lemma 17. *The function $dy(\underline{n}_{\mathbb{N}}, \omega)$ is bijective.*

Proof. By the definition of the function, we know that it is in $\mathcal{POR}^-$, so the function is constant with respect to its second parameter. Moreover, It is clearly an injection, because different numbers have different encodings and $\forall n > 0 \in \mathbb{N}.\forall \omega \in \mathbb{O}.bin(n, \omega)$ has $\mathbf{1}$ as leftmost bit (it can be shown by induction on n, leveraging the definition of bin). So we just need to show that it is surjective. We know that dy is computed removing a bit which is always $\mathbf{1}$. This entails that taken a string $\sigma \in \mathbb{S}$, it is the image of the natural number n such that, the binary encoding of $n + 1$ is $\mathbf{1}\sigma$. This number always exist. □

Lemma 18.

$$\eta \sim_{dy} \omega \to \forall n \in \mathbb{N}.\eta_n = e(\underline{n}_{\mathbb{N}}, \omega)$$

Proof. By contraposition: suppose that the consequence does not hold:

$$\eta_n \neq e(\underline{n}_{\mathbb{N}}, \omega)$$

By definition of e, this means that there is an $i \in \mathbb{N}$ such that $\eta(i) \neq \omega(dy(\underline{i}_{\mathbb{N}}, \omega))$, which is a contradiction. □