Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 221 (2008) 299–308
www.elsevier.com/locate/entcs

On Finite-time Computability Preserving Conversions

Hideki Tsuiki[1,2]

Graduate School of Human and Environmental Studies
Kyoto University
606-8501 Kyoto, Japan

Shuji Yamada[3]

Faculty of Science
Kyoto Sangyo University
603-8555 Kyoto, Japan

Abstract

A finite-time computable function is a partial function from Σ^ω to Σ^ω whose value is constructed by applying finite number of list operations 'cons' and 'head' to the argument. A finite-time computability preserving conversion $\alpha : X \to Y$ for $X, Y \subset \Sigma^\omega$ is a bijection which preserves finite-time computability. We show that all the finite-time computability preserving conversions with the domain Σ^ω are extended sliding block functions.

Keywords: Finite-time Computable Functions, Constant-time Computable Functions, Sliding Block Functions, Computable Analysis, Domain Theoryx.

1 Introduction

In Type2 theory of effectivity [6], computability of a function on a space A is defined through a representation $\varphi :\subset \Sigma^\omega \to A$ and a Type2 machine, which inputs and outputs Σ^ω sequences and defines the notion of computable functions on Σ^ω. One way of implementing infinite sequences in 'real' programming languages is (lazy) infinite lists, and one can write computable functions on Σ^ω with programming languages which support lazy lists, such as Haskell and ML.

[1] The first author was partially supported by the Grant-in-Aid for Scientific Research (No.18500013 and No.19650029) from Japan Society for the Promotion of Science.
[2] Email:tsuiki@i.h.kyoto-u.ac.jp
[3] Email:yamada@cc.kyoto-su.ac.jp

doi:10.1016/j.entcs.2008.12.025

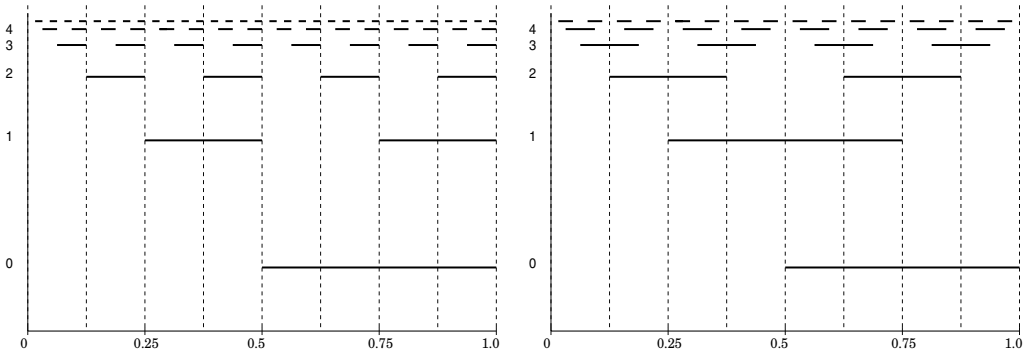

Fig. 1. The Binary representation and Gray representation of $[0,1]$. Here, line segments mean the corresponding digit is 1, and otherwise 0.

In such an implementation, the output infinite list is constructed from the input infinite list. In many cases, the whole output list is constructed with infinite number of cons operations. However, one can also construct the output list by applying finite number of list operations tail and cons to the argument. In other words, the output infinite list is constructed by concatenating a finite list and a suffix of the input, both of which are computed from the input infinite list. In this case, one can construct the output infinite list in a finite time when the input infinite list is given. Therefore, we call such a partial function on Σ^ω a finite-time computable function. When a representation $\varphi :\subset \Sigma^\omega \to A$ is given, we define a finite-time computable (multi-valued partial) function on a set A as one which is realized by a finite-time computable partial function on Σ^ω.

Note that the class of finite-time computable functions depends on the representation to use. For example, on the closed unit interval $\mathbb{I} = [0,1]$ and $\Sigma = \{0,1\}$, we consider, in addition to the binary representation $\delta_B : \Sigma^\omega \to \mathbb{I}$, the Gray representation $\delta_G : \Sigma^\omega \to \mathbb{I}$ [5]. It is the continuous function which satisfies the following,

$$\delta_G(0 \cdot x) = \delta_G(x)/2,$$
$$\delta_G(1 \cdot x) = 1 - \delta_G(x)/2.$$

Note that the binary representation δ_B satisfies

$$\delta_B(0 \cdot x) = \delta_B(x)/2,$$
$$\delta_B(1 \cdot x) = 1/2 + \delta_B(x)/2.$$

Figure 1 shows the binary representation and the Gray representation. Gray representation is based on binary reflected Gray-code, which is a coding of natural numbers different from the ordinary binary code [3].

The two representations have the relation

$$\delta_B = \delta_G \circ G \tag{1}$$

for $G : \Sigma^\omega \to \Sigma^\omega$ the Gray-code conversion function defined as $G(x)_n = x_n \oplus x_{n-1}$. Here, $a \oplus b$ is the exclusive-OR of a and b, x_n is the n-th digit of x, and x_{-1} is defined to be 0. In Section 2, some examples of finite-time computable functions with respect to these representations are given. As we will show, a finite-time computable functions with respect to binary representation is finite-time computable with respect to Gray representation, but not vice versa. More generally, if φ and φ' are two representations of a set and φ' is the composition of φ with G as (1), then all the finite-time computable functions with respect to φ' are finite-time computable with respect to φ. Therefore, we can consider it a property of the conversion function G, and we define that a computable conversion function $\alpha : X \to Y$ for $X, Y \subset \Sigma^\omega$ is finite-time computability preserving if $\alpha \circ f \circ \alpha^{-1}$ is finite-time computable for any finite-time computable partial function f. Gray-code conversion is a special case of a sliding block function [1]. In this paper, we will define a more general class of functions, called extended sliding block functions for the case $\mathrm{X} = \Sigma^\omega$.

With this preparation, we show our main theorem. It says that all the finite-time computability preserving conversions from Σ^ω to Y are extended sliding block functions. We first show that, in this case, a finite-time computability preserving conversion satisfies some recursive equation and it coincides with the minimal fixed-point. Then, we show that it is an extended sliding block function.

One of the properties of a finite-time computability preserving conversion is that it preserves suffix identity in that if two arguments have the same suffix then their values also have the same suffix. However, not all suffix-identity preserving conversions are finite-time computability preserving, and our main theorem does not hold for suffix-identity preserving conversions from Σ^ω. The important thing about finite-time computability is that the shared suffix between x and $f(x)$, which exists from the definition of finite-time computability, should be computable from x.

We introduce finite-time computable functions with some examples in Section 2, and finite-time computability preserving conversions in Section 3. Then, in Section 4, we concentrate on the case $X = \Sigma^\omega$ and show our main theorem. In Section 5, we show relations between finite-time computability preserving and suffix-identity preserving conversions.

Notation and Terminology:

Let Σ be a finite alphabet. For a cardinal number $c \leq \omega$, we denote by Σ^c the set of sequences on Σ of length c. We also denote by Σ^* the set of sequences on Σ of finite length, and by Σ^∞ the set $\Sigma^* \cup \Sigma^\omega$. We denote by $|x|$ the length of a sequence x and denote by ε the empty sequence.

For a finite or infinite sequence $x = \{x_i\}_{0 \leq i < c}$ ($c \leq \omega$), we denote by $x\restriction_m$ the sequence $\{x_i\}_{m \leq i < c}$ obtained by discarding the first m digits (elements) of x and denote by $x\restriction^n$ the sequence $\{x_i\}_{0 \leq i < n}$ obtained by extracting the first n digits of x. We define $x\restriction_m$ to be the empty sequence ε if $|x| \leq m$ and $x\restriction^n$ to be x if $|x| \leq n$. We denote by $x\restriction^n_m$ the sequence $(x\restriction^n)\restriction_m = \{x_i\}_{m \leq i < n}$. We also denote by $x\restriction_*$ the set $\{x\restriction_n \mid n \in \mathbb{N}\}$ and denote by $x\restriction^*$ the set $\{x\restriction^n \mid n \in \mathbb{N}\}$. We call an element of $x\restriction^*$ a prefix of x and an element of $x\restriction_*$ a suffix of x. On Σ^∞, we define the order

relation $x \leq y$ if $x \in y\uparrow^*$ or $x = y$.

For $u \in \Sigma^*$ and $v \in \Sigma^\infty$, we denote by $u \cdot v$ the concatenated sequence of u and v. For a letter $a \in \Sigma$ and a sequence x, we also denote by $a \cdot x$ the sequence obtained by prepending a to x. For finite sequences $x(0), x(1), \ldots, x(n) \in \Sigma^*$, we denote by $\prod_{0 \leq i \leq n} x(i)$ the concatenated sequence $x(0) \cdot x(1) \cdot \cdots \cdot x(n)$.

When f is a partial function from Σ^ω to Σ^ω, we say that f is computable if there is a Type2 machine M which outputs $f(x)$ when $x \in \mathrm{dom}(f)$ is given as the input. It is well known that computable functions on Σ^ω are continuous. Note that we do not require f to be strong, that is, $\mathrm{dom}(f)$ may be a proper subset of $\mathrm{dom}(M)$. We also define computable functions from Σ^ω to Σ^*, and from Σ^ω to $\mathbb{N}$ similarly. For $X, Y \subset \Sigma^\omega$, we call a bijection from X to Y a conversion function.

2 Finite-time Computable Functions

Definition 2.1 We say that a partial function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$ is finite-time computable if there are computable partial functions $g :\subseteq \Sigma^\omega \to \Sigma^*$ and $\mu :\subseteq \Sigma^\omega \to \mathbb{N}$ such that

$$f(x) = g(x) \cdot x\uparrow_{\mu(x)}$$

for $x \in \mathrm{dom}(f)$.

Definition 2.2 We say that a partial function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$ is constant-time computable if f is finite-time computable and there is a $k \in \mathbb{N}$ such that the values of $g(x)$ and $\mu(x)$ in the definition of finite-time computability of f depend only on $x\uparrow^k$ for all $x \in \mathrm{dom}(f)$. That is, $x\uparrow^k = y\uparrow^k$ implies $g(x) = g(y)$ and $\mu(x) = \mu(y)$.

Proposition 2.3 *A finite-time computable function is a computable function.*

Proposition 2.4 *If* $\mathrm{dom}(f)$ *is compact, then any finite-time computable function on* X *is constant-time computable.*

When a representation $\varphi_A :\subseteq \Sigma^\omega \to A$ is given, we can define finite-time computability of a function on A.

Definition 2.5 Let $\varphi : \Sigma^\omega \to A$ be a representation of A. We say that a partial multi-valued function $F :\subseteq A \rightrightarrows A$ is finite-time computable with respect to φ when F is realized by a finite-time computable partial function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$. That is, $\varphi(f(x)) \in F(\varphi(x))$ for every $x \in \Sigma^\omega$.

We also define constant-time computable partial multi-valued function $F : A \rightrightarrows A$, similarly.

Note 1 *We can generally define finite-time computability for multi-valued functions with the domain and the range different. We have this definition because we are only interested in the case they are the same in this paper.*

Example 2.6 Let $\Sigma = \{0, 1\}$, $\mathbb{I} = [0, 1]$ be the closed unit interval, and $\delta_B : \Sigma^\omega \to \mathbb{I}$ be the binary representation of $\mathbb{I}$. That is, $\delta_B(x) = \Sigma_{n=0}^{\infty} x_n / 2^{n+1}$. Then, the following (multi-valued) functions $F_i : \mathbb{I} \rightrightarrows \mathbb{I}$ $(i = 1, 2, 3)$ are constant time computable with $f_i : \Sigma^\omega \to \Sigma^\omega$ the function which realizes F_i $(i = 1, 2, 3)$.

(i) $F_1(a) = a/2$.
$f_1(x) = 0 \cdot x$.

(ii) $F_2(a) = \begin{cases} 2a & \text{if } 0 \le a \le 1/2, \\ 2a-1 & \text{if } 1/2 \le a \le 1. \end{cases}$
$f_2(x) = x\!\uparrow_1$.

(iii) $F_3(a) = \begin{cases} a+1/2 & \text{if } 0 \le a \le 1/2, \\ a-1/2 & \text{if } 1/2 \le a \le 1. \end{cases}$
$f_3(x) = \begin{cases} 1 \cdot x\!\uparrow_1 & \text{if } x_0 = 0, \\ 0 \cdot x\!\uparrow_1 & \text{if } x_0 = 1. \end{cases}$

Example 2.7 Let $\Sigma = \{0, 1, '.'\}$, $X = \{x \in \Sigma^\omega \mid x \text{ contains one '.'}\}$, and $\delta'_B :\subseteq \Sigma^\omega \to \mathbb{R}^+$ be the binary representation of the set $\mathbb{R}^+$ of non-negative real numbers. That is, $\mathrm{dom}(\delta'_B) = X$ and $\delta'_B(x) = \delta_B(x\!\uparrow^k \cdot x\!\uparrow_{k+1})2^k$ for k the index of '.' in x. Then, $F_4 : \mathbb{R}^+ \to \mathbb{R}^+$ defined as $F_4(a) = 2a$ is finite-time computable, with its code the function $f_4 : X \to X$ defined recursively as follows,

$f_4(c \cdot x) = c \cdot f_4(x) \quad (\text{for } c \in \{0,1\})$,
$f_4(. \cdot c \cdot x) = c \cdot . \cdot x \quad (\text{for } c \in \{0,1\})$.

Example 2.8 Let $\Sigma = \{0, 1\}$, $\delta_G : \Sigma^\omega \to \mathbb{I}$ be the Gray representation defined in Section 1. We can extend it to the representation $\delta'_G : X \to \mathbb{R}^+$ of $\mathbb{R}^+$ for X the set defined in Example 2.7, as we did for the binary representation. As we will show in the next section, all the finite-time computable and constant-time computable functions with respect to δ_B and δ'_B belong to the same class of functions with respect to δ_G and δ'_G, respectively. For example, F_1 is realized by f_1 and F_2 is realized by

$$f_5(x) = \begin{cases} x\!\uparrow_1 & \text{if } x_0 = 0, \\ 0 \cdot x\!\uparrow_2 & \text{if } x_0 = 1 \text{ and } x_1 = 1. \\ 1 \cdot x\!\uparrow_2 & \text{if } x_0 = 1 \text{ and } x_1 = 0. \end{cases}$$

In addition to them, there are functions finite-time computable with respect to δ_G but not with respect to δ_B. For example, the following functions F_6 and F_7 are constant-time computable with respect to δ_G with f_2 and f_3 the functions which realize them, respectively.

(i) $F_6(a) = \begin{cases} 2a & \text{if } 0 \le a \le 1/2, \\ 2-2a & \text{if } 1/2 \le a \le 1. \end{cases}$

(ii) $F_7(a) = 1 - a$.

However, F_6 and F_7 are not finite-time computable with respect to δ_B.

3 Finite-time computability preserving conversion

Let the Gray-code conversion $G : \Sigma^\omega \to \Sigma^\omega$ be the function defined as $G(x)_n = x_n \oplus x_{n-1}$. Here, $a \oplus b$ is the exclusive-OR of a and b, and x_{-1} is defined to be 0. Then, the two representations δ_G and δ_B have the relation:

$$\delta_B = \delta_G \circ G.$$

Definition 3.1 Let X and Y be subsets of Σ^ω. We say that a bijection $\alpha : X \to Y$ is a finite-time computability preserving conversion if α is computable and $\alpha \circ f \circ \alpha^{-1}$ is finite-time computable for all finite-time computable partial function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$, whose domain and range are subsets of X.

Proposition 3.2 *Let X and Y be subsets of Σ^ω, $\alpha : X \to Y$ be a finite-time computability preserving conversion and $\varphi :\subseteq \Sigma^\omega \to A$ be a representation of a set A with $\mathrm{dom}(\varphi) = Y$. If a partial multi-valued function $f :\subseteq A \rightrightarrows A$ is finite-time computable with respect to $\varphi \circ \alpha$, then it is finite-time computable with respect to φ.*

For integers $k, m \geq 0$ and a function $B : \Sigma^{k+m+1} \to \Sigma$, we define a sliding block function $\alpha : \Sigma^\omega \to \Sigma^\omega$ as $\alpha(x)_n = B(x_{n-k}, \ldots, x_{n+m})$. Here, we assume that x_n is defined to be some constant for $n < 0$. Gray-code conversion is a sliding block function for $k = 1, m = 0, B : \Sigma^2 \to \Sigma$ the exclusive-OR function, and $x_{-1} = 0$.

More generally, we define an extended sliding block function as follows.

Definition 3.3 Let Y be a subset of Σ^ω. We say that a function $\alpha : \Sigma^\omega \to Y$ is an extended sliding block function if there are an interger n and computable functions $\lambda : \Sigma^\omega \to \Sigma^*$, $\rho : \Sigma^n \to \Sigma^*$, and $s : \Sigma^\omega \to \mathbb{N}$ such that

$$\alpha(x) = \lambda(x) \cdot \rho(x \upharpoonright_{s(x)}^{s(x)+n}) \cdot \rho(x \upharpoonright_{s(x)+1}^{s(x)+1+n}) \cdot \rho(x \upharpoonright_{s(x)+2}^{s(x)+2+n}) \cdot \cdots = \lambda(x) \cdot \prod_{s(x) \leq i < \omega} \rho(x \upharpoonright_i^{i+n}).$$

Proposition 3.4 *Any extended sliding block bijection preserves finite-time computability.*

Corollary 3.5 *(1) A sliding block bijection preserves finite-time computability.*
(2) Gray-code conversion preserves finite-time computability.
(3) Every finite-time computable function with respect to the binary representation is finite-time computable with respect to the Gray representation.

4 Characterization of Finite-time Computability Preserving Conversions from Σ^ω

In this section, we show the converse of Proposition 3.4 as our main theorem.

Theorem 4.1 *If $\alpha : \Sigma^\omega \to Y$ is a finite-time computability preserving conversion, then α is an extended sliding block function.*

To prove this theorem, we need some lemmata. Let $\alpha : \Sigma^\omega \to \Sigma^\omega$ be a finite-time computability preserving conversion.

For a letter $a \in \Sigma$, let $p_a : \Sigma^\omega \to \Sigma^\omega$ be the function which prepends the letter a to sequences, that is, $p_a(x) = a \cdot x$. This prepending function p_a is a typical finite-time computable function on Σ^ω.

Lemma 4.2 *There are an integer n and computable functions $g : \Sigma^n \to \Sigma^*$ and $\mu : \Sigma^n \to \mathbb{N}$ such that the following recursive formula holds.*

$$\alpha(x) = g(x\restriction^n) \cdot \alpha(x\restriction_1)\restriction_{\mu(x\restriction^n)} . \tag{2}$$

Proof. Since $\alpha \circ p_a \circ \alpha^{-1}$ is a finite-time computable function, it must have the form $g_a(x) \cdot x\restriction_{\mu_a(x)}$ for computable functions $g_a : \Sigma^\omega \to \Sigma^*$ and $\mu_a : \Sigma^\omega \to \mathbb{N}$. Thus, we have

$$\alpha(a \cdot x) = g_a(\alpha(x)) \cdot \alpha(x)\restriction_{\mu_a(\alpha(x))} .$$

We define computable functions $g : \Sigma^\omega \to \Sigma^*$ and $\mu : \Sigma^\omega \to \mathbb{N}$ by

$$g(x) = g_{x_0}(\alpha(x\restriction_1)),$$

$$\mu(x) = \mu_{x_0}(\alpha(x\restriction_1)).$$

Then, α must satisfy

$$\alpha(x) = \alpha(x_0 \cdot x\restriction_1) = g(x) \cdot \alpha(x\restriction_1)\restriction_{\mu(x)} .$$

Since Σ^ω is compact and g and μ are computable, there is an integer n such that the values of $g(x)$ and $\mu(x)$ are defined by $x\restriction^n$ and we can regard the domain of these functions as Σ^n. □

Note that (2) is a recursive equation that α must satisfy.

We introduce a new letter H which does not belong to Σ. We call this letter H a "hole" which cancels the succeeding ordinary letter when the following function $\theta : (\Sigma \cup \{H\})^\infty \to \Sigma^\infty$ is applied. We first define $\theta : (\Sigma \cup \{H\})^* \to \Sigma^*$ on finite sequences. For any sequence $x \in (\Sigma \cup \{H\})^*$,

$$\theta(x) = \begin{cases} x_0 \cdot \theta(x\restriction_1) & \text{if } x_0 \in \Sigma, \\ \theta(x\restriction_1)\restriction_1 & \text{if } x_0 = H, \\ \varepsilon & \text{if } x = \varepsilon. \end{cases}$$

It can be easily checked that θ is monotonic. That is, for $x \in (\Sigma \cup \{H\})^*$ and an integer m, there is a integer m' such that, $\theta(x\restriction^m) = \theta(x)\restriction^{m'}$. Therefore, we can extend this function to the function $\theta : (\Sigma \cup \{H\})^\infty \to \Sigma^\infty$ as

$$\theta(x) = \lim_{m \to \omega} \theta(x\restriction^m)$$

for $x \in (\Sigma \cup \{H\})^\omega$.

We define functions $\alpha' : \Sigma^\omega \to (\Sigma \cup \{H\})^\omega$ and $\alpha'_j : \Sigma^{j+n} \to (\Sigma \cup \{H\})^*$ for an integer j by

$$\alpha'(x) = \prod_{0 \le i < \omega} g(x\upharpoonright_i^{i+n}) \cdot H^{\mu(x\upharpoonright_i^{i+n})},$$

$$\alpha'_j(x) = \prod_{0 \le i < j} g(x\upharpoonright_i^{i+n}) \cdot H^{\mu(x\upharpoonright_i^{i+n})}.$$

We also extend α'_j to a function from Σ^ω to $(\Sigma \cup \{H\})^*$ as $\alpha'_j(x) = \alpha'_j(x\upharpoonright^{j+n})$.

Then, from the definition, α' satisfies the following recursive formula

$$\alpha'(x) = g(x\upharpoonright^n) \cdot H^{\mu(x\upharpoonright^n)} \cdot \alpha'(x\upharpoonright_1).$$

If we apply θ to this formula, we have

$$\theta(\alpha'(x)) = g(x\upharpoonright^n) \cdot \theta(\alpha'(x\upharpoonright_1))\upharpoonright_{\mu(x\upharpoonright^n)}.$$

This recursive formula of $\theta \circ \alpha'$ is the same as the one (2) of α. In addition, from the construction of θ and α', $\theta \circ \alpha'$ is the least fixed-point of the recursive equation (2) on the domain $[\Sigma^\omega \to \Sigma^\infty]$, and therefore $\theta(\alpha'(x)) \le \alpha(x)$ for any function $\alpha \in [\Sigma^\omega \to \Sigma^\infty]$ which satisfies the equation (2). For the details of domain theory, see [2] and [4], for example.

Now, we prove that $|\theta(\alpha'(x))| = \omega$ for any $x \in \Sigma^\omega$, and therefore it is the only fixed-point of (2). One of the difficulties in handling $\alpha'(x)$ is that each digit of the output is determined by some interval $x\upharpoonright_i^{i+n}$ of length n and the intervals overlap for each i. Therefore, we pick up indexes $k_1, \ldots, k_n, \ldots$ for which $x\upharpoonright_{k_i}^{k_i+n}$ are the same. Such a sequence $k_1, \ldots, k_n, \ldots$ exists because $|\Sigma^n|$ is finite.

Lemma 4.3 *If $x\upharpoonright_k^{k+n} = x\upharpoonright_{k'}^{k'+n}$ for $x \in \Sigma^\omega$ and integers $k < k'$, then*

$$\sum_{k \le i < k'} (|g(x\upharpoonright_i^{i+n})| - \mu(x\upharpoonright_i^{i+n})) > 0.$$

We skip the proof of this lemma. Note that it uses the fact that α is an injective function.

Lemma 4.4 *For any $x \in \Sigma^\omega$,*

$$\lim_{i \to \omega} |\theta(\alpha'_i(x))| = \omega.$$

Therefore, we have the following.

Lemma 4.5 $\theta \circ \alpha' = \alpha$.

Next, we show that $\theta \circ \alpha'$ is an extended sliding block function. Set $N = |\Sigma|^n$ and $M = \max_{w \in \Sigma^n} \mu(w)$. Since $\theta(\alpha'_j(x))$ is monotonic to j and $\lim_{i \to \omega} |\theta(\alpha'_i(x))| = \omega$, we have $j_0, j_1, \ldots$ such that $\theta(\alpha'_{j_i+1}(x)) > \theta(\alpha'_{j_i}(x))$. In the following lemma, we show that such j_i appear in every interval of length MN^2.

Lemma 4.6 *For any $x \in \Sigma^\omega$ and any integer $j \in \mathbb{N}$, $|\theta(\alpha'_j(x))| < |\theta(\alpha'_{j+MN^2}(x))|$.*

We define functions $\lambda : \Sigma^\omega \to \Sigma^*$, $\rho : \Sigma^{MN^2+n+1} \to \Sigma^*$, and $s : \Sigma^\omega \to \mathbb{N}$ as follows:

$$\lambda(x) = \theta(\alpha'_{MN^2}(x)),$$
$$\rho(x) = \theta(\alpha'_{MN^2+1}(x)) \upharpoonright_{|\theta(\alpha'_{MN^2}(x))|},$$
$$s(x) = 0.$$

We will accomplish the proof of the theorem with the following lemma.

Lemma 4.7

$$\alpha(x) = \lambda(x) \cdot \prod_{s(x) \le i < \omega} \rho(x \upharpoonright_i^{i+MN^2+n+1}).$$

Corollary 4.8 *Every extended sliding block function can be expressed as*

$$\alpha(x) = \lambda(x) \cdot \prod_{0 \le i < \omega} \rho(x \upharpoonright_i^{i+n}).$$

5 Suffix Identity Preserving Conversions

Definition 5.1 We say that two infinite sequences x and y *share a suffix* if there are integers m and n such that $x \upharpoonright_m = y \upharpoonright_n$. We write $x \sim y$ when x and y share a suffix. We say that a partial function $f :\subseteq \Sigma^\omega \to \Sigma^\omega$ is *suffix-identical* if x and $f(x)$ share a suffix for all $x \in \mathrm{dom}(f)$.

From the definition, a partial function f is suffix-identical if and only if $f(x)$ has the form $g(x) \cdot x \upharpoonright_{\mu(x)}$ for partial functions $g :\subseteq \Sigma^\omega \to \Sigma^*$ and $\mu :\subseteq \Sigma^\omega \to \mathbb{N}$. The difference between a suffix-identical function and a finite-time computable function is that g and μ are required to be computable in the latter case.

A finite-time computable function is obviously suffix-identical. However, the converse is not true, as we will see in Example 5.4.

Definition 5.2 For $X, Y \subset \Sigma^\omega$, we say that a conversion α from X to Y *preserves suffix identities* if $\alpha(z)$ and $\alpha(w)$ share a suffix for any $z, w \in X$ which share a suffix.

Since $\sim$ is an equivalence relation, we consider the quotient $\Sigma^\omega/\sim$. A suffix identity preserving conversion α determines a function from $\Sigma^\omega/\sim$ to $\Sigma^\omega/\sim$.

Proposition 5.3 *A finite-time computability preserving conversion preserves suffix identities.*

On the other hand, not all suffix identities preserving conversions are finite-time computability preserving.

Example 5.4 For $n > 0$, we denote $z(n) = \prod_{1 \le i \le n} 1^i 0 = 101^2 01^3 0 \cdots 1^{n-1} 01^n 0$ and $z(\omega) = \prod_{1 \le i} 1^i 0 = 101^2 01^3 0 \cdots 1^{n-1} 01^n 0 \ldots$, and define a function $\beta : \Sigma^* \to \Sigma^*$

by

$$\beta(0^{m_0}1^{n_0}0^{m_1}1^{n_1}\cdots 0^{m_k}1^{n_k}) = 0^{m_0}z(n_0)0^{m_1}z(n_1)\cdots 0^{m_k}z(n_k),$$

where $n_0,\ldots,n_{k-1},m_1,\ldots,m_k > 0$ and $n_k, m_0 \geq 0$. Since $\beta(x\upharpoonright^n)$ is a prefix of $\beta(x)$ for any $n \in \mathbb{N}$ and $x \in \Sigma^*$, we can extend β to $\beta : \Sigma^\omega \to \Sigma^\omega$. Let $Y = \mathrm{range}(\beta)$.

The conversion $\beta : \Sigma^\omega \to Y$ preserves suffix identities. That is, when $x\upharpoonright_n = y\upharpoonright_m$, $\beta(x)\upharpoonright_k = \beta(z)\upharpoonright_l$ for some k and l. Note that we need to study separately the case x contains finite number of 0, in which case $\beta(1^\omega) = z(\omega)$ is a suffix of x. On the other hand, β is not finite-time computability preserving. Consider the function $p_1(x) = 1\cdot x$. It is a finite-time computable function. Let $h = \beta \circ p_1 \circ \beta^{-1} : Y \to Y$. h satisfies the followings.

$$\begin{aligned} h(0\cdot x) &= 1\cdot 0\cdot 0\cdot x,\\ h(z(n)\cdot 0\cdot x) &= z(n+1)\cdot 0\cdot x,\\ h(z(\omega)) &= z(\omega). \end{aligned}$$

One can consider these equations as a procedure to compute h, and thus h is a computable function. One can see that h is a suffix-identical function.

However, h is not a finite-time computable function. Suppose that $h(y) = g(y)\cdot y\upharpoonright_{\mu(y)}$ for $g :\subseteq Y \to \Sigma^*$ and $\mu :\subseteq Y \to \mathbb{N}$ computable functions. We have $\mu(z(n)\cdot 0\cdot x) \geq |z(n)| - n - 1$, and $\lim_{n\to\omega}\mu(z(n)\cdot 0\cdot x) = \omega$. On the other hand, $\lim_{n\to\omega} z(n)\cdot 0\cdot x$ is $z(\omega)$ and $\mu(z(\omega))$ must have a finite value. Therefore, μ is not continuous, and therefore it is not computable.

Since $\mathrm{dom}(\beta) = \Sigma^\omega$, this example also shows that our main theorem does not hold for suffix-identity preserving conversions.

References

[1] Douglas A. Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding.* Cambridge University Press, Cambridge, 1995.

[2] G. Gieez, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove and D. S. Scott. *Continuous lattices and domains.* Cambridge Univ. Press, Cambridge, 2003.

[3] F. Gray. Pulse code communications. U. S. Patent 2632058, March 1953.

[4] Viggo Stoltenberg-Hansen, Ingrid Lindstrom and Edward R. Griffor. *Mathematical theory of domains.* Cambridge University Press, 1994.

[5] Hideki Tsuiki. Real number computation through gray code embedding. *Theoretical Computer Science*, 284(2):467–485, 2002.

[6] Klaus Weihrauch. *Computable analysis, an Introduction.* Springer-Verlag, Berlin, 2000.