*Journal of Universal Computer Science, vol. 15, no. 6 (2009), 1365-1380*
*submitted: 4/11/08, accepted: 4/2/09, appeared: 28/3/09 © J.UCS*

# On Finite-time Computability Preserving Conversions[1]

**Hideki Tsuiki**
(Graduate School of Human and Environmental Studies, Kyoto University
Japan
tsuiki@i.h.kyoto-u.ac.jp)

**Shuji Yamada**
(Faculty of Science, Kyoto Sangyo University
Japan
yamada@cc.kyoto-su.ac.jp)

**Abstract:** A finite-time computable function is a partial function from $\Sigma^\omega$ to $\Sigma^\omega$ whose value is constructed by concatenating a finite list with a suffix of 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 $\Sigma^\omega$ are extended sliding block functions.

**Key Words:** Finite-time Computable Functions, Constant-time Computable Functions, Sliding Block Functions, Computable Analysis, Domain Theory.
**Category:** F.1.m, F.4.3, G.2.m

## 1 Introduction

In Type2 theory of effectivity [Weihrauch 2000], 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 $\Sigma^\omega$ sequences and defines the notion of computable functions on $\Sigma^\omega$. One way of implementing infinite sequences in 'real' programming languages is (lazy) infinite lists, and one can write computable functions on $\Sigma^\omega$ with programming languages which support lazy lists, such as Haskell and ML.

In such an implementation, the output infinite list is constructed from the input infinite list. In many cases, the whole output is constructed one by one by applying infinite number of cons operations. However, for some functions, the input and output infinite list share a suffix and therefore the output can be constructed by using a part of the input. That is, the output is constructed by concatenating a finite list and a suffix of the input, both of which are computed from the input. In this case, the output is constructed in a finite time when the input infinite list is given. Therefore, we call such a partial function on $\Sigma^\omega$ a

[1] An extended abstract version of this paper has appeared in [Tsuiki and Yamada 2008].

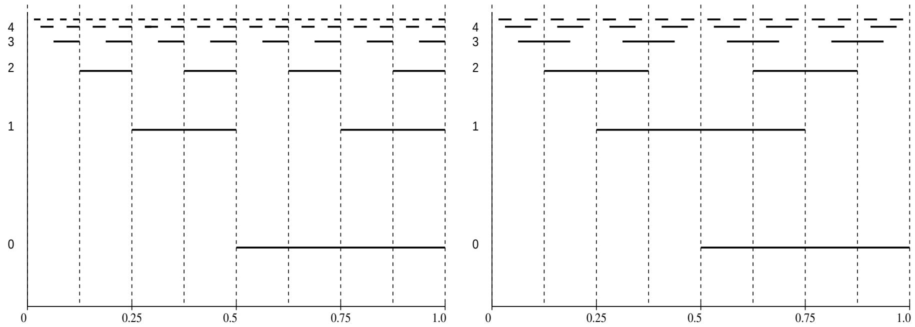


Figure 1: The Binary representation and Gray representation of $[0, 1]$. Here, line segments mean the corresponding digit is 1, and otherwise 0.

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 $\Sigma^\omega$.

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 $\delta_G : \Sigma^\omega \to \mathbb{I}$ [Tsuiki 2002]. 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 $\delta_B$ 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 [Gray 1953].

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 function with respect to binary representation is finite-time computable with respect to Gray representation, but not vice versa. This fact can be explained through the following relation between the two representations

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

Here, $G : \Sigma^\omega \to \Sigma^\omega$ is the Gray-code conversion function defined as $G(x)_n = x_n \oplus x_{n-1}$ with $a \oplus b$ the exclusive-OR of $a$ and $b$, $x_n$ the $n$-th digit of $x$, and

$x_{-1}$ defined to be 0. We can show that if $\varphi$ and $\varphi'$ are two representations of a set and $\varphi'$ is the composition of $\varphi$ with $G$ as (1), then all the finite-time computable functions with respect to $\varphi'$ are finite-time computable with respect to $\varphi$. Therefore, we can consider it a property of the conversion function $G$. We generalize this property and define that a conversion function $\alpha : X \to Y$ for $X, Y \subset \Sigma^\omega$ is finite-time computability preserving if $\alpha$ is computable and $\alpha \circ f \circ \alpha^{-1}$ is finite-time computable for any finite-time computable partial function $f$.

On the other hand, Gray-code conversion is a sliding block function, which is a function whose $n$-th output value is calculated only from a fixed size of "window" around the $n$-th cell of the input[Lind and Marcus 1995]. 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 $\Sigma^\omega$ 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 least 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 $\Sigma^\omega$. 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:**

In this paper, we denote by $\mathbb{N}$ the set of non-negative integers. Let $\Sigma$ be a finite alphabet. For a cardinal number $c \leq \omega$, we denote by $\Sigma^c$ the set of sequences on $\Sigma$ of length $c$. We also denote by $\Sigma^*$ the set of sequences on $\Sigma$ of finite length, and by $\Sigma^\infty$ the set $\Sigma^* \cup \Sigma^\omega$. We denote by $|x|$ the length of a sequence $x$ and denote by $\varepsilon$ the empty sequence.

For a finite or infinite sequence $x$, we denote by $x_n$ the $n$-th element of $x$. For a finite or infinite sequence $x = \{x_i\}_{0 \leq i < c}$ ($c \leq \omega$), we denote by $x\!\uparrow_m$ the sequence $\{x_i\}_{m \leq i < c}$ obtained by discarding the first $m$ digits (elements) of $x$ and denote by $x\!\uparrow^n$ the sequence $\{x_i\}_{0 \leq i < n}$ obtained by extracting the first $n$

digits of $x$. We define $x\uparrow_m$ to be the empty sequence $\varepsilon$ if $|x| \leq m$ and $x\uparrow^n$ to be $x$ if $|x| \leq n$. We denote by $x\uparrow^n_m$ the sequence $(x\uparrow^n)\uparrow_m = \{x_i\}_{m\leq i<n}$. We also denote by $x\uparrow_*$ the set $\{x\uparrow_n \mid n \in \mathbb{N}\}$ and denote by $x\uparrow^*$ the set $\{x\uparrow^n \mid n \in \mathbb{N}\}$. We call an element of $x\uparrow^*$ a prefix of $x$ and an element of $x\uparrow_*$ a suffix of $x$. On $\Sigma^\infty$, 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 $\Sigma^\omega$ to $\Sigma^\omega$, 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 $\Sigma^\omega$ 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 $\Sigma^\omega$ to $\Sigma^*$, and from $\Sigma^\omega$ 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 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.** 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 3.** *A finite-time computable function is a computable function.*

**Proposition 4.** *If the domain of a finite-time computable function is compact, it is a constant-time computable function.*

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

**Definition 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* $\varphi$ if $F$ 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 6.* 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 1.* 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, each of the following (multi-valued) function $F_i : \mathbb{I} \rightrightarrows \mathbb{I}$ $(i = 1, 2, 3)$ is constant time computable with $f_i : \Sigma^\omega \to \Sigma^\omega$ the function which realizes $F_i$.

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

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

3. $F_3(a) = \begin{cases} a + 1/2 & \text{if } 0 \le a \le 1/2, \\ a - 1/2 & \text{if } 1/2 \le a \le 1. \end{cases}$
   $f_3(x) = \mathrm{not}(x_0) \cdot x{\uparrow_1}$.

   Here, $\mathrm{not} : \Sigma \to \Sigma$ is the function defined as $\mathrm{not}(0) = 1$ and $\mathrm{not}(1) = 0$.

*Example 2.* Let $\Sigma = \{0, 1, \text{'.'}\}$, $X = \{x \in \Sigma^\omega \mid x \text{ contains one '.'}\}$, 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,

$$f_4(x) = \begin{cases} x_0 \cdot f_4(x{\uparrow_1}) & \text{if } x_0 = 0, 1, \\ x_1 \cdot . \cdot x{\uparrow_2} & \text{if } x_0 = \text{'.'}. \end{cases}$$

*Example 3.* Let $\Sigma = \{0, 1\}$ and $\delta_G : \Sigma^\omega \to \mathbb{I}$ be the Gray representation defined in Section 1. We can extend it to the representation $\delta'_G :\subseteq \{0, 1, \text{'.'}\}^\omega \to \mathbb{R}^+$ of $\mathbb{R}^+$ as we did for the binary representation in Example 2. As we will show in the next section, all the finite-time computable and constant-time computable functions with respect to $\delta_B$ and $\delta'_B$ belong to the same class of functions with respect to $\delta_G$ and $\delta'_G$, respectively. For example, with respect to $\delta_G$, $F_1$ is realized by $f_1$, $F_2$ is realized by

$$f_5(x) = \begin{cases} x{\uparrow_1} & \text{if } x_0 = 0, \\ \mathrm{not}(x_1) \cdot x{\uparrow_2} & \text{if } x_0 = 1, \end{cases}$$

and $F_3$ is realized by

$$f_6(x) = \mathrm{not}(x_0) \cdot \mathrm{not}(x_1) \cdot x\!\uparrow_2 .$$

Besides, $F_4$ is realized by $f_4$ with respect to $\delta'_G$.

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

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

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

However, $F_6$ and $F_7$ are not finite-time computable with respect to $\delta_B$.

## 3 Finite-time computability preserving conversions

**Definition 7.** Let $X$ and $Y$ be subsets of $\Sigma^\omega$. We say that a bijection $\alpha : X \to Y$ is a *finite-time computability preserving conversion* if $\alpha$ 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 8.** *Let $X$ and $Y$ be subsets of $\Sigma^\omega$, $\alpha : X \to Y$ be a finite-time computability preserving conversion and $\varphi :\subseteq \Sigma^\omega \to A$ be a representation of a set $A$ with $\mathrm{dom}(\varphi) = X$. If a partial multi-valued function $F :\subseteq A \rightrightarrows A$ is finite-time computable with respect to $\varphi$, then it is finite-time computable with respect to $\varphi \circ \alpha^{-1}$.*

*Proof.* If $F$ is realized by a finite-time computable function $f$ with respect to $\varphi$, then $F$ is realized by $\alpha \circ f \circ \alpha^{-1}$ with respect to $\varphi \circ \alpha^{-1}$. □

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 defined in Section 1 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 9.** Let $Y$ be a subset of $\Sigma^\omega$. We say that a function $\alpha : \Sigma^\omega \to Y$ is an *extended sliding block* function if there are an integer $n$ and computable functions $\lambda : \Sigma^\omega \to \Sigma^*$, $\rho : \Sigma^n \to \Sigma^*$, and $s : \Sigma^\omega \to \mathbb{N}$ such that

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

**Proposition 10.** *Let $Y$ be a subset of $\Sigma^\omega$. If $\alpha$ is a computable bijection from $\Sigma^\omega$ to $Y$, then $\alpha^{-1}$ is computable.*

*Proof.* Since $\alpha$ is computable, there is a monotonic computable function $h : \Sigma^* \to \Sigma^*$ which approximates $\alpha$. That is, for $x \in \Sigma^\omega$ and $k \in \mathbb{N}$, $h$ satisfies $h(x\uparrow^k) < \alpha(x)$ and $\lim_{k\to\omega} h(x\uparrow^k) = \alpha(x)$. We define a function $g$ from $\Sigma^*$ to $\Sigma^*$ which approximates $\alpha^{-1}$ as follows. For each $u \in \Sigma^*$, let $U_u$ be the set $\{v \in \Sigma^* \mid h(v) \geq u\}$ and $V_u$ be the set of minimal elements of $U_u$. We define $g(u) \in \Sigma^*$ be the longest common prefix of elements of $V_u$.

Let $u \in \Sigma^*$. Since $\Sigma^\omega$ is compact, there is $k \in \mathbb{N}$ such that if $|v| \geq k$ then $|h(v)| \geq |u|$. Therefore, $V_u \subset \Sigma^{\leq k}$ for $\Sigma^{\leq k} = \cup_{i \leq k} \Sigma^i$. Thus, one can enumerate all the elements of $V_u$ in finite time, and one can compute $g(u)$ from $u$. From the definition, $g$ is a monotonic function from $\Sigma^*$ to $\Sigma^*$. On the other hand, if $\alpha(x) > u$ for $x \in \Sigma^\omega$ and $u \in \Sigma^*$, then $h(x\uparrow^n) > u$ for some $n$ and thus $x\uparrow^n \in U_u$, and we have $g(u) \leq x\uparrow^n$ and thus $x > g(u)$. Therefore, $g(y\uparrow^k) < \alpha^{-1}(y)$ for all $y \in Y$ and $k \in \mathbb{N}$.

Now, we only need to prove that for each $n \in \mathbb{N}$, there is $m \in \mathbb{N}$ such that $|g(y\uparrow^m)| \geq n$ for every $y \in Y$. Since $\Sigma^\omega$ is compact, $Y$ is also compact and $\alpha^{-1}$ is a uniformly continuous function. Therefore, for each $n \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that if $\alpha(x)\uparrow^k = \alpha(x')\uparrow^k$ then $x\uparrow^n = x'\uparrow^n$. Take $m = max(k, max\{|h(u)| \mid u \in \Sigma^n\})$. Then, we have $|g(\alpha(x)\uparrow^m)| \geq n$ for every $x \in \Sigma^\omega$. □

**Proposition 11.** *Any extended sliding block bijection preserves finite-time computability.*

*Proof.* Let $\alpha : x \mapsto \lambda(x) \cdot \prod_{s(x) \leq i < \omega} \rho(x\uparrow_i^{i+n})$ be an extended sliding block bijection. Let $f$ be a finite-time computable partial function, which satisfies $f(x) = g(x) \cdot x\uparrow_{\mu(x)}$ for $x \in \mathrm{dom}(f)$. For $x \in \mathrm{dom}(f)$, we denote $y = \alpha(x)$, $z = f(x)$ and $w = \alpha(z)$. We will show that the function $\alpha \circ f \circ \alpha^{-1}$, which maps $y$ to $w$, is finite-time computable.

Since $z = g(x) \cdot x\uparrow_{\mu(x)}$, we have $x\uparrow_{\mu(x)} = z\uparrow_{|g(x)|}$. Set an integer $k = \max\{0, s(x) - \mu(x), s(z) - |g(x)|\}$, then $x_{k+\mu(x)+i} = z_{k+|g(x)|+i}$ for all $i \geq 0$, and this implies $y\uparrow_{|\lambda(x)\cdot\rho(x\uparrow_{s(x)}^{s(x)+n})\cdot\dots\cdot\rho(x\uparrow_{k+\mu(x)-1}^{k+\mu(x)-1+n})|} = w\uparrow_{|\lambda(z)\cdot\rho(z\uparrow_{s(z)}^{s(z)+n})\cdot\dots\cdot\rho(z\uparrow_{k+|g(x)|-1}^{k+|g(x)|-1+n})|}$. If we set $g'(y) = \lambda(z) \cdot \rho(z\uparrow_{s(z)}^{s(z)+n}) \cdot \dots \cdot \rho(z\uparrow_{k+|g(x)|-1}^{k+|g(x)|-1+n})$, $\mu'(y) = |\lambda(x) \cdot \rho(x\uparrow_{s(x)}^{s(x)+n}) \cdot \dots \cdot \rho(x\uparrow_{k+\mu(x)-1}^{k+\mu(x)-1+n})|$, then we have $w = g'(y) \cdot y\uparrow_{\mu'(y)}$. By Proposition 10, $z = f(\alpha^{-1}(y))$ is computable from $y$ and therefore $g'$ and $\mu'$ are computable functions. □

**Corollary 12.**
*(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 $\Sigma^\omega$

In this section, we show the converse of Proposition 11 as our main theorem.

**Theorem 13.** *If $\alpha : \Sigma^\omega \to Y$ is a finite-time computability preserving conversion, then $\alpha$ is an extended sliding block function.*

To prove this theorem, we need some lemmata. We divide the proof into three parts and show them in the following subsections. Let $\alpha : \Sigma^\omega \to Y$ be a finite-time computability preserving conversion.

### 4.1 A Recursive equation that $\alpha$ must satisfy

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 $\Sigma^\omega$.

**Lemma 14.** *There are an integer $n \geq 0$ and computable functions $g : \Sigma^n \to \Sigma^*$ and $\mu : \Sigma^n \to \mathbb{N}$ such that the following recursive formula holds.*

$$\alpha(x) = g(x{\uparrow}^n) \cdot \alpha(x{\uparrow}_1){\uparrow}_{\mu(x{\uparrow}^n)} \,. \tag{2}$$

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

$$\alpha(a \cdot x) = g_a(\alpha(x)) \cdot \alpha(x){\uparrow}_{\mu_a(\alpha(x))} \,.$$

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

$$g(x) = g_{x_0}(\alpha(x{\uparrow}_1)),$$

$$\mu(x) = \mu_{x_0}(\alpha(x{\uparrow}_1)).$$

Then, $\alpha$ must satisfy

$$\alpha(x) = \alpha(x_0 \cdot x{\uparrow}_1) = g(x) \cdot \alpha(x{\uparrow}_1){\uparrow}_{\mu(x)} \,.$$

Since $\Sigma^\omega$ is compact and $g$ and $\mu$ are computable, there is an integer $n \geq 0$ such that the values of $g(x)$ and $\mu(x)$ are defined by $x{\uparrow}^n$ and we can regard the domains of these functions as $\Sigma^n$. ∎

Equation (2) says that $\alpha(x)$ is composed of the sequence $g(x\uparrow^n)$ and the infinite sequence obtained by removing the first $\mu(x)$ characters from $\alpha(x\uparrow_1)$. Therefore, one way of computing the sequence $\alpha(x)$ would be to compute and output $g(x\uparrow^n)$, calculate the number $\mu(x)$, and start the computation of $\alpha(x\uparrow_1)$ and output it disregarding the first $\mu(x)$ characters. We define a function $\alpha' : \Sigma^\omega \to \Sigma^\infty$ which can be computed with this procedure.

We introduce a new letter $H$ which does not belong to $\Sigma$. 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 $x \in (\Sigma \cup \{H\})^*$,

$$\theta(x) = \begin{cases} x_0 \cdot \theta(x\uparrow_1) & \textit{if } \ x_0 \in \Sigma, \\ \theta(x\uparrow_1)\uparrow_1 & \textit{if } \ x_0 = H, \\ \varepsilon & \textit{if } \ x = \varepsilon. \end{cases}$$

It can be easily checked that $\theta$ is monotonic. That is, for $x \in (\Sigma \cup \{H\})^*$ and an integer $m \geq 0$, there is a integer $m'$ such that, $\theta(x\uparrow^m) = \theta(x)\uparrow^{m'}$. Therefore, we can extend this function to the function $\theta : (\Sigma \cup \{H\})^\infty \to \Sigma^\infty$ as

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

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

We define functions $h_j : \Sigma^{j+n} \to (\Sigma \cup \{H\})^*$ and $h : \Sigma^\omega \to (\Sigma \cup \{H\})^\infty$ for an integer $j \geq 0$ by

$$h_j(x) = \prod_{0 \leq i \leq j} g(x\uparrow_i^{i+n}) \cdot H^{\mu(x\uparrow_i^{i+n})},$$

$$h(x) = \prod_{0 \leq i < \omega} g(x\uparrow_i^{i+n}) \cdot H^{\mu(x\uparrow_i^{i+n})}.$$

We also extend $h_j$ to a function from $\Sigma^\omega$ to $(\Sigma \cup \{H\})^*$ as $h_j(x) = h_j(x\uparrow^{j+n})$ and define functions

$$\alpha'_j = \theta \circ h_j,$$

$$\alpha' = \theta \circ h.$$

By definition, $h_j$ and $h$ satisfy the following formulae.

$$h_{j+1}(x) = g(x\uparrow^n) \cdot H^{\mu(x\uparrow^n)} \cdot h_j(x\uparrow_1),$$

$$h(x) = g(x\uparrow^n) \cdot H^{\mu(x\uparrow^n)} \cdot h(x\uparrow_1).$$

If we apply $\theta$ to these formulae, we have

$$\alpha'_{j+1}(x) = g(x\uparrow^n) \cdot \alpha'_j(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}, \tag{3}$$

$$\alpha'(x) = g(x\!\uparrow^n) \cdot \alpha'(x\!\uparrow_1)\!\uparrow_{\mu(x\uparrow^n)} . \tag{4}$$

The functions $\alpha'_j$ and $\alpha'$ satisfy $\alpha'_0(x) = g(x\!\uparrow^n)$, $\alpha'_j(x) \leq \alpha'_{j+1}(x)$, and $\alpha'(x) = \lim_{m\to\omega} \alpha'_m(x)$. Note that the recursive formula (4) of $\alpha'$ is the same as the one (2) of $\alpha$, though $\alpha : \Sigma^\omega \to \Sigma^\omega$ and $\alpha' : \Sigma^\omega \to \Sigma^\infty$ have different ranges.

**Lemma 15.** *$\alpha'(x) \leq \alpha(x)$ for $x \in \Sigma^\omega$.*

*Proof.* We consider the pointwise order on the set $[\Sigma^\omega \to \Sigma^\infty]$ of functions from $\Sigma^\omega$ to $\Sigma^\infty$. It becomes a domain, that is, a directed complete partial ordered set with a least element. For the details of domain theory, see [Gierz et al. 2003] and [Stoltenberg-Hansen et al. 1994], for example. Let $F$ be the function from $[\Sigma^\omega \to \Sigma^\infty]$ to $[\Sigma^\omega \to \Sigma^\infty]$ defined as

$$F(\beta) = \lambda x.g(x\!\uparrow^n) \cdot \beta(x\!\uparrow_1)\!\uparrow_{\mu(x\uparrow^n)} .$$

Equation (2) and (4) say that $F(\alpha) = \alpha$ and $F(\alpha') = \alpha'$, respectively, and therefore $\alpha$ and $\alpha'$ are fixed-points of $F$. On the other hand, it is easy to see that $F$ is a continuous function, and thus $F$ has a least fixed-point. Let $\alpha'_{-1}(x) = \varepsilon$. Then, $\alpha'_{-1}$ is the least element of $[\Sigma^\omega \to \Sigma^\infty]$ and $\alpha'_0 = F(\alpha'_{-1})$. In addition, we have $\alpha'_{j+1} = F(\alpha'_j)$ $(j \geq 0)$ from (3). Therefore, $\alpha'$ is actually the least fixed-point of $F$, and we have $\alpha' \leq \alpha$. ∎

### 4.2 $\alpha$ is the unique solution of Equation (2)

Note that we still have the possibility that $\alpha$ and $\alpha'$ are different and, for some $x$, $\alpha'(x)$ is a finite sequence and $\alpha(x)$ is an infinite sequence obtained as its extension. Now, we prove that $|\alpha'(x)| = \omega$ for any $x \in \Sigma^\omega$. By Lemma 15, it means that $\alpha = \alpha'$ and therefore $\alpha$ is the unique solution of (2). First, we prepare a small lemma.

**Lemma 16.** *For $x, y, z \in \Sigma^\omega$ and $i, j, k, l \in \mathbb{N}$, if $x\!\uparrow_i = y\!\uparrow_j$ and $y\!\uparrow_k = z\!\uparrow_l$, then $x\!\uparrow_{i+k} = z\!\uparrow_{j+l}$.*

*Proof.* For $m \in \mathbb{N}$, $x_{i+k+m} = y_{j+k+m} = z_{j+l+m}$. ∎

One of the difficulties in handling $h(x)$ is that each digit of the output is determined by some interval $x\!\uparrow_i^{i+n}$ of length $n$ and the intervals overlap for each $i$. Therefore, we pick up indexes $k_1, \ldots, k_n, \ldots$ for which $x\!\uparrow_{k_i}^{k_i+n}$ are the same. Such a sequence $k_1, \ldots, k_n, \ldots$ exists because $|\Sigma^n|$ is finite.

**Lemma 17.** *Let $x \in \Sigma^\omega$ and $0 \leq k < k'$. If $x\!\uparrow_k^{k+n} = x\!\uparrow_{k'}^{k'+n}$, then we have the following.*

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

*(2)* $\sum_{k\le i<k'}(|g(x{\uparrow}_{i+1}^{i+1+n})| - \mu(x{\uparrow}_i^{i+n})) > 0$.

*Proof.* (1) Set

$$u = x{\uparrow}_k^{k'},$$

$$e = |u| = k' - k,$$

$$s = \sum_{0\le i<e} |g(x{\uparrow}_{k+i}^{k+i+n})|,$$

$$t = \sum_{0\le i<e} \mu(x{\uparrow}_{k+i}^{k+i+n}),$$

$$d = t - s,$$

$$q = \lceil n/e \rceil + 1.$$

From the assumption $x{\uparrow}_k^{k+n} = x{\uparrow}_{k'}^{k'+n}$, $x_i = x_{i+e}$ for $k \le i < k+n$. If $e < n$ then $u = x{\uparrow}_k^{k+e} = x{\uparrow}_{k+e}^{k+2e} = x{\uparrow}_{k+2e}^{k+3e} = \cdots = x{\uparrow}_{k+(q-2)e}^{k+(q-1)e}$ and $x{\uparrow}_{k+(q-1)e}^{k+e+n} = u{\uparrow}^{n-(q-2)e}$. If $e \ge n$ then $q = 2$ and $x{\uparrow}_{k+e}^{k+e+n} = u{\uparrow}^n$. Therefore we have $x{\uparrow}_k^{k'+n} = u^q{\uparrow}^{e+n}$.

Consider a word $y = u^{q+1}\cdot w$, where $w$ is an arbitrary word in $\Sigma^\omega$. Then $y{\uparrow}_i^{i+n} = x{\uparrow}_{k+i}^{k+i+n}$ for $0 \le i < e$. From the recursive equation (2), we have

$$\alpha(y{\uparrow}_i){\uparrow}_{|g(y{\uparrow}_i^{i+n})|} = \alpha(y{\uparrow}_{i+1}){\uparrow}_{\mu(y{\uparrow}_i^{i+n})}$$

for $0 \le i < e$. Therefore, by Lemma 16, we have $\alpha(y){\uparrow}_s = \alpha(y{\uparrow}_e){\uparrow}_t$. That is,

$$\alpha(u^{q+1}\cdot w){\uparrow}_s = \alpha(u^q \cdot w){\uparrow}_{s+d}\ .$$

Since this holds for arbitrary $w \in \Sigma^\omega$, by applying it repeatedly, we have

$$\alpha(u^{q+\ell}\cdot w){\uparrow}_s = \alpha(u^q\cdot w){\uparrow}_{s+d\ell}$$

for all positive integer $\ell$ such that $s + d\ell > 0$.

To prove that $d < 0$, we will derive contradictions from assumptions $d > 0$ and $d = 0$.

Assume that $d > 0$. Since $\alpha$ is computable, there is an integer $p$ $(> q)$ such that

$$\alpha(u^p\cdot w){\uparrow}_s^{s+d} = \alpha(u^p\cdot w'){\uparrow}_s^{s+d}$$

for all $w, w' \in \Sigma^\omega$. Then, for all $\ell \ge 0$,

$$\alpha(u^p\cdot w){\uparrow}_{s+d\ell}^{s+d\ell+d} = \alpha(u^{p+\ell}\cdot w){\uparrow}_s^{s+d} = \alpha(u^{p+\ell}\cdot w'){\uparrow}_s^{s+d} = \alpha(u^p\cdot w'){\uparrow}_{s+d\ell}^{s+d\ell+d}\ .$$

Therefore, $\alpha(u^p\cdot w){\uparrow}_s = \alpha(u^p\cdot w'){\uparrow}_s$. This means that $\alpha(u^p\cdot w)$ has only $|\Sigma|^s$ different values for $w \in \Sigma^\omega$. This contradicts that $\alpha$ is an injection.

Assume that $d = 0$. We have

$$\alpha(u^{\ell+q} \cdot w){\uparrow_s} = \alpha(u^q \cdot w){\uparrow_s} \tag{5}$$

for all $\ell \geq 0$. Since $\alpha$ is computable, for any integer $i$ $(\geq s)$, there is an integer $p$ $(> q)$ such that

$$\alpha(u^p \cdot w)_i = \alpha(u^p \cdot w')_i$$

for all $w, w' \in \Sigma^\omega$. Therefore, from (5), we have

$$\alpha(u^q \cdot w)_i = \alpha(u^q \cdot w')_i$$

for any $i \geq s$ and $w, w' \in \Sigma^\omega$. This means that $\alpha(u^q \cdot w)$ has only $|\Sigma|^s$ different values for $w \in \Sigma^\omega$. This contradicts that $\alpha$ is an injection.

(2) Since $g(x{\uparrow_k^{k+n}}) = g(x{\uparrow_{k'}^{k'+n}})$, we have the result from (1).

□

**Lemma 18.** *For any $x \in \Sigma^\omega$,*

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

*Proof.* We can choose infinitely many integers $0 \leq k_1 < k_2 < \cdots < k_j < \cdots$ such that $x{\uparrow_{k_1}^{k_1+n}} = x{\uparrow_{k_2}^{k_2+n}} = \cdots = x{\uparrow_{k_j}^{k_j+n}} = \cdots$. From the previous lemma, $|\alpha'_{k_1}(x)| < |\alpha'_{k_2}(x)| < \cdots$. Then we have $\lim_{i\to\omega} |\alpha'_i(x)| = \omega$. □

Therefore, we have the following.

**Lemma 19.** $\alpha' = \alpha$.

### 4.3 $\alpha$ is an extended sliding block function

Finally, we show that $\alpha'$ is an extended sliding block function. Set $N = |\Sigma|^n$ and $M = \max_{w\in\Sigma^n} \mu(w)$. Since $\alpha'_j(x)$ is monotonic to $j$ and $\lim_{i\to\omega} |\alpha'_i(x)| = \omega$, we have $j_0, j_1, \ldots$ such that $\alpha'_{j_i+1}(x) > \alpha'_{j_i}(x)$. In the following lemma, we consider the case $M > 0$ and show that such $j_i$ appear in every interval of length $MN^2$.

**Lemma 20.** *Suppose that $M > 0$. For any $x \in \Sigma^\omega$ and any integer $j \geq 0$, $|\alpha'_j(x)| < |\alpha'_{j+MN^2}(x)|$.*

*Proof.* Let $j_0$ be the least non-negative integer such that $\alpha'_{j_0}(x) = \alpha'_j(x)$. It is enough to show that $|\alpha'_{j_0}(x)| < |\alpha'_{j_0+MN^2}(x)|$.

For any integer $k \geq j_0$, we denote $\sum_{j_0\leq i<k}(-\mu(x{\uparrow_i^{i+n}}) + |g(x{\uparrow_{i+1}^{i+1+n}})|)$ by $e(k)$. It is easily checked that $|\alpha'_\ell(x)| - |\alpha'_{j_0}(x)| = \max_{j_0\leq k<\ell} e(k)$ for $\ell > j_0$. We shall show that $e(k') > 0$ for some $k'$ $(j_0 < k' \leq j_0 + MN^2)$.

Let $j_0 < j_1 < j_2 < \cdots < j_m \leq j_0 + MN^2$ be the indices at which minimal value of $e$ is updated, namely, $\{j' \mid j_0 \leq j' \leq j_0 + MN^2 \text{and } e(i) >$

$e(j')$ for all $i$ $(j_0 \leq i < j')\} = \{j_0, j_1, j_2, \ldots, j_m\}$. Then $e(j_0) = 0 > e(j_1) > e(j_2) > \cdots > e(j_m) = \min_{j_0 \leq k < j_0 + MN^2} e(k)$. From Lemma 17, $x\uparrow_{j_0}^{j_0+n}, x\uparrow_{j_1}^{j_1+n}, \ldots, x\uparrow_{j_m}^{j_m+n}$ must be mutually distinct. Therefore $m < N$. Since $e(j_\ell) - e(j_{\ell-1}) \geq -M$ for all $\ell = 1, \ldots, m$ and $e(j_0) = 0$, we have $e(j_m) > -MN$.

Let $w \in \Sigma^n$ be the sequence which appears most frequently in the $MN^2 + 1$ sequences $\{x\uparrow_i^{i+n}\}_{j_0 \leq i \leq MN^2}$. Then $w$ appears more than $MN$ times. Let $x\uparrow_k^{k+n}$ be the first appearance of $w$ and $x\uparrow_{k'}^{k'+n}$ be the last appearance of $w$. From Lemma 17, $e(k') > e(k) + MN$. Since $e(k) \geq e(j_m) > -MN$, we have $e(k') > 0$. ∎

**Lemma 21.** *Suppose that $M > 0$. For any $x \in \Sigma^\omega$ and $j \in \mathbb{N}$, $|\alpha'_{j+MN^2}(x\uparrow_1)| > \mu(x\uparrow^n)$.*

*Proof.* By Lemma 20, $|\alpha'_{1+MN^2}(x)| > |\alpha'_1(x)|$ holds. On the other hand, by Equation (3), we have $\alpha'_{1+MN^2}(x) = g(x\uparrow^n) \cdot \alpha'_{MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}$ and $|\alpha'_1(x)| \geq |\alpha'_0(x)| = |g(x\uparrow^n)|$.

Therefore, $|\alpha'_{MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}| > 0$. It means that $|\alpha'_{MN^2}(x\uparrow_1)| > \mu(x\uparrow^n)$. Since $|\alpha'_{j+MN^2}(x\uparrow_1)| \geq |\alpha'_{MN^2}(x\uparrow_1)|$, we have the result. ∎

**Lemma 22.** *Suppose that $M > 0$. For any $x \in \Sigma^\omega$ and $i \in \mathbb{N}$,*

$$\alpha'_{i+1+MN^2}(x)\uparrow_{|\alpha'_{i+MN^2}(x)|} = \alpha'_{1+MN^2}(x\uparrow_i)\uparrow_{|\alpha'_{MN^2}(x\uparrow_i)|}\ .$$

*Proof.* We first prove

$$\alpha'_{j+2+MN^2}(x)\uparrow_{|\alpha'_{j+1+MN^2}(x)|} = \alpha'_{j+1+MN^2}(x\uparrow_1)\uparrow_{|\alpha'_{j+MN^2}(x\uparrow_1)|} \tag{6}$$

for every $j \in \mathbb{N}$. By (3), we have $\alpha'_{j+2+MN^2}(x) = g(x\uparrow^n) \cdot \alpha'_{j+1+MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}$ and $|\alpha'_{j+1+MN^2}(x)| = |g(x\uparrow^n)| + |\alpha'_{j+MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}|$. By Lemma 21, we have $|\alpha'_{j+MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}| > 0$. Therefore,

$$\begin{aligned}
\alpha'_{j+2+MN^2}(x)\uparrow_{|\alpha'_{j+1+MN^2}(x)|} &= (\alpha'_{j+1+MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)})\uparrow_{|\alpha'_{j+MN^2}(x\uparrow_1)\uparrow_{\mu(x\uparrow^n)}|} \\
&= \alpha'_{j+1+MN^2}(x\uparrow_1)\uparrow_{|\alpha'_{j+MN^2}(x\uparrow_1)|}\ .
\end{aligned}$$

Now, we can repeatedly apply Equation (6) to have

$$\begin{aligned}
\alpha'_{i+1+MN^2}(x)\uparrow_{|\alpha'_{i+MN^2}(x)|} &= \alpha'_{i+MN^2}(x\uparrow_1)\uparrow_{|\alpha'_{i-1+MN^2}(x\uparrow_1)|} \\
&= \ldots \\
&= \alpha'_{1+MN^2}(x\uparrow_i)\uparrow_{|\alpha'_{MN^2}(x\uparrow_i)|}\ .
\end{aligned}$$

∎

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

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

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

**Lemma 23.**
$$\alpha(x) = \lambda(x) \cdot \prod_{s(x)\leq i<\omega} \rho(x\uparrow_i^{i+n+1+MN^2}).$$

*Proof.* If $M > 0$, we have $\alpha'_{1+MN^2}(x\uparrow_i)\uparrow_{|\alpha'_{MN^2}(x\uparrow_i)|}= \alpha'_{i+1+MN^2}(x)\uparrow_{|\alpha'_{i+MN^2}(x)|}$ from the previous lemma, Therefore, $\alpha'_{i+MN^2+1}(x) = \alpha'_{i+MN^2}(x) \cdot \rho(x\uparrow_i)$. Then we have, $\alpha(x) = \alpha'_{MN^2}(x) \cdot \prod_{0\leq i} \rho(x\uparrow_i)$.

If $M = 0$, from (2), we have $\alpha(x) = g(x\uparrow^n) \cdot \alpha(x\uparrow_1) = \prod_{0\leq i<\omega} g(x\uparrow_i^{i+n}) = g(x\uparrow^n)\cdot\prod_{0\leq i<\omega} g(x\uparrow_{i+1}^{i+n+1})$. On the other hand, we have $\lambda(x) = \alpha'_0(x) = g(x\uparrow^n)$ and $\rho(x) = \alpha'_1(x)\uparrow_{|\alpha'_0(x)|}= (g(x\uparrow^n) \cdot g(x\uparrow_1^{n+1}))\uparrow_{|g(x\uparrow^n)|}= g(x\uparrow_1^{n+1})$. □

By Proposition 11, Theorem 13, and Lemma 23, we have the following.

**Theorem 24.** *Let $\alpha$ be a conversion from $\Sigma^\omega$ to $Y \subset \Sigma^\omega$. The following are equivalent.*
*(1) $\alpha$ is a finite-time computability preserving conversion.*
*(2) $\alpha$ is an extended sliding block function.*
*(3) $\alpha$ can be expressed as*

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

## 5 Suffix identity preserving conversions

**Definition 25.** We say that two infinite sequences $x$ and $y$ *share a suffix* if there are non-negative integers $m$ and $n$ such that $x\uparrow_m= y\uparrow_n$. 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 $g(x)\cdot x\uparrow_{\mu(x)}$ 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 $\mu$ 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 4.

**Definition 26.** For $X, Y \subset \Sigma^\omega$, we say that a conversion $\alpha$ 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 $\alpha$ determines a function from $\Sigma^\omega/\sim$ to $\Sigma^\omega/\sim$.

**Proposition 27.** *A finite-time computability preserving conversion preserves suffix identities.*

*Proof.* Suppose that $\alpha : X \to Y$ is a finite-time computability preserving conversion and $w\uparrow_n = z\uparrow_m$ for $w, z \in X$. Consider the partial finite-time computable function $f(x) = z\uparrow^m x\uparrow_n$ with the domain $\{w\}$. We have $f(w) = z$, $\mathrm{dom}(f) \subset X$ and $\mathrm{range}(f) \subset X$. Therefore, $h = \alpha \circ f \circ \alpha^{-1}$ is a finite-time computable function and it has the form $g(y) \cdot y\uparrow_{\mu(y)}$. Since $h(\alpha(w)) = \alpha(z)$, $\alpha(z) = g(\alpha(w)) \cdot \alpha(w)\uparrow_{\mu(\alpha(w))}$. For $k = \mu(\alpha(w))$ and $l = |g(\alpha(w))|$, we have $\alpha(w)\uparrow_k = \alpha(z)\uparrow_l$. □

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

*Example 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 $z(\omega) = \prod_{1 \le i} 1^i 0 = 101^2 01^3 0 \cdots 1^{n-1} 01^n 0 \cdots$, 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 $n_k, m_0 \ge 0$. Since $\beta(x\uparrow^n)$ is a prefix of $\beta(x)$ for any $n \in \mathbb{N}$ and $x \in \Sigma^*$, we can extend $\beta$ to $\beta : \Sigma^\omega \to \Sigma^\omega$. Let $Y = \mathrm{range}(\beta)$.

The conversion $\beta : \Sigma^\omega \to Y$ preserves suffix identities. That is, if $x\uparrow_n = y\uparrow_m$, then $\beta(x)\uparrow_k = \beta(z)\uparrow_l$ 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, $\beta$ is not finite-time computability preserving. Consider the finite-time computable function $p_1(x) = 1 \cdot x$. Let $h = \beta \circ p_1 \circ \beta^{-1} : Y \to Y$. It 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 $h(y) = g(y) \cdot y\uparrow_{\mu(y)}$ 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, $\mu$ 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.

**Acknowledgements**

This work was partially supported by Grant-in-Aid for Scientific Research (No.18500013) and Grant-in-Aid for Scientific Research (No.19650029) from Japan Society for the Promotion of Science.

## References

[Gierz et al. 2003] Gierz, G., Hofmann, K. H., Keimel, K., Lawson, J. D., Mislove, M., and Scott, D. S.: "Continuous lattices and domains"; Cambridge University Press, Cambridge (2003).

[Gray 1953] Gray, F.: "Pulse code communications"; U. S. Patent 2632058 (March 1953).

[Lind and Marcus 1995] Lind, D. A., Marcus, B.: "An Introduction to Symbolic Dynamics and Coding"; Cambridge University Press, Cambridge (1995).

[Stoltenberg-Hansen et al. 1994] Stoltenberg-Hansen, V., Lindstrom, I., Griffor, E. R.: "Mathematical theory of domains"; Cambridge University Press (1994).

[Tsuiki 2002] Tsuiki, H.: "Real number computation through gray code embedding"; Theoretical Computer Science, 284,2 (2002) 467-485.

[Tsuiki and Yamada 2008] Tsuiki, H., Yamada, S.: "On Finite-time Computability Preserving Conversions"; in Brattka, V., Dillhage, R., Grubba, T., Klutsch, A. (eds.): "Proceedings of the 5th International Conference on Computability and Complexity in Analysis"; Electronic Notes in Theoretical Computer Science 221 (2008), 299-308.

[Weihrauch 2000] Weihrauch, K.: "Computable analysis, an Introduction"; Springer-Verlag, Berlin (2000).