ON PROVING TIME CONSTRUCTIBILITY OF FUNCTIONS

Kojiro KOBAYASHI

Department of Information Science, Tokyo Institute of Technology, Oh-okayama, Meguro-ku, Tokyo, Japan 152

Communicated by R.V. Book
Received March 1984
Revised June 1984

Abstract. We formalize the techniques that have been used to prove time constructibility of functions by means of two theorems. The first theorem gives one sufficient condition for time constructibility of \( f_1(n) + f_2(n) \) and \( f_2(n) \) to imply that \( f_1(n) \). As an application of this theorem, we show that, for a function \( f(n) \) such that \( (\exists \epsilon > 0) (\forall n) f(n) \geq (1 + \epsilon) n \), \( f(n) \) is time constructible if and only if it is computable by a Turing machine within \( O(f(n)) \) steps. The second theorem concerns time constructibility of functions \( f(n) \) for which there are no \( \epsilon > 0 \) such that \( (\forall n) f(n) \geq (1 + \epsilon) n \).

1. Introduction

A time constructible function (tc function, for short) is a function \( f(n) \) such that there is a deterministic multi-tape Turing machine that halts at \( f(n) \) when it is given an input word of length \( n \). Such functions play important roles in the theory of computation. They are interesting by themselves as examples of very 'honest' functions. Hence it is desirable that we have powerful techniques for proving time constructibility of functions.

Many simple functions such as \( n^2 \), \( n^3 \), \( 2^n \), \( n! \) are known to be tc. The proofs are not essentially difficult, but are tedious. In the present paper we formulate many techniques used for these proofs by means of two theorems. Using them, the proofs are considerably simplified.

Let \( \mathcal{F}_1 \) be the set of all functions \( f(n) \) such that \( f(n) \geq n \), and let \( \mathcal{F}_2 \) be the set of all functions \( f(n) \) in \( \mathcal{F}_1 \) such that \( (\exists \epsilon > 0) (\forall n) f(n) \geq (1 + \epsilon) n \). When we try to prove time constructibility of a function \( f(n) \), we usually use completely different techniques depending on whether \( f(n) \) is in \( \mathcal{F}_2 \) or in \( \mathcal{F}_1 - \mathcal{F}_2 \). This is because we can use speed-up of computation for functions in \( \mathcal{F}_2 \) but not for those in \( \mathcal{F}_1 - \mathcal{F}_2 \).

Our first theorem (Theorem 3.1) says that, if both \( f_1(n) + f_2(n) \), \( f_2(n) \) are tc, \( f_1(n) \) is in \( \mathcal{F}_1 \), and

\[
(\exists \epsilon > 0) (\forall n) f_1(n) \geq \epsilon f_2(n) + (1 + \epsilon) n,
\]

then \( f_1(n) \) is also tc. This theorem replaces many tedious arguments using speed-up of computation and simplifies proof of time constructibility of many functions in \( \mathcal{F}_2 \).
From this it also follows that, for a function \( f(n) \) in \( \mathcal{F}_2 \), \( f(n) \) is \( \text{tc} \) if and only if it is computable by a Turing machine within \( O(f(n)) \) steps (the representations of the input and the output may be either unary or binary) (see Corollary 4.3).

Our second theorem (Theorem 5.2) formalizes one technique for proving time constructibility of functions in \( \mathcal{F}_1 - \mathcal{F}_2 \). It reduces the proof of time constructibility of functions of the form \( n + f(n) \) (\( f(n) \) is nondecreasing and unbounded) to efficiently computing \( n_{i+1} - n_i \), where \( n_0, n_1, \ldots \) are the values of \( n \) at which \( f(n) \) increases. Using this theorem, we can easily show that, for example, \( n + \log n \), \( n + \log \log n \), \( n + (\log^* n)^p \) are \( \text{tc} \) (\( p > 0 \) is a rational number).

2. Preliminaries

By a Turing machine we mean a deterministic multi-tape Turing machine [1]. We assume that all the tapes are working tapes (that is, we use no special input or output tapes), and that tapes are infinite in both directions. By \( \mathbb{N} \) we denote the set of all nonnegative integers. We say that a \( d \) variable function \( f(n_1, \ldots, n_d) \) from \( \mathbb{N}^d \) to \( \mathbb{N} \) is \textit{time constructible} (tc for short) if there is a Turing machine with at least \( d \) tapes that stops at time \( f(n_1, \ldots, n_d) \) when it is started with the following configuration at time 0: (1) for each \( i \) (\( 1 \leq i \leq d \)), the content of the \( i \)th tape is \( \ldots BB1^nBB \ldots \) and the head scans the leftmost 1 (if \( n_i > 0 \) (\( B \) denotes the blank symbol and \( 1^n \) denotes the sequence of \( n_i \)'s), and (2) the contents of other tapes, if any, are all blank.

By \( (\forall^\infty n_1, \ldots, n_d) \), we mean that a property \( P \) on \( n_1, \ldots, n_d \) is true except for a finite number of tuples \( (n_1, \ldots, n_d) \) in \( \mathbb{N}^d \). In this case, we say that \( P \) is true for almost all \( n_1, \ldots, n_d \). To simplify description, we write \( n \) instead of \( n_1, \ldots, n_d \), and write \( \max n \) instead of \( \max\{n_1, \ldots, n_d\} \).

We define three classes \( \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_2 \) of functions. \( \mathcal{F}_0 \) is the class of all functions. \( \mathcal{F}_1 \) is the class of all functions \( f(n) \) such that \( f(n) \geq \max n \). \( \mathcal{F}_2 \) is the class of all functions \( f(n) \) in \( \mathcal{F}_1 \) such that \( (\exists \varepsilon > 0) (\forall^\infty n) f(n) \geq (1 + \varepsilon) \max n \). (In this paper, the letters \( \varepsilon, \varepsilon_1, \varepsilon_2, \ldots \) denote real numbers.)

3. One theorem on speed-up of computation

First we show that, in many cases, 'overhead' computation time can be canceled by partial speed-up of computation.

**Theorem 3.1.** If both of \( f_1(n) + f_2(n) \) and \( f_2(n) \) are \( \text{tc} \), \( f_1(n) \in \mathcal{F}_1 \), and

\[
(\exists \varepsilon > 0) (\forall^\infty n) f_1(n) \geq \varepsilon f_2(n) + (1 + \varepsilon) \max n,
\]

then \( f_1(n) \) is \( \text{tc} \).

**Proof.** Let \( M_1, M_2 \) be Turing machines that halt at \( f_1(n) + f_2(n) \) and \( f_2(n) \) respec-
Proving time constructibility of functions

We construct a Turing machine $M(k)$ that depends on an integer parameter $k (>8)$. It performs the following operations.

**Step 1.** $M(k)$ compresses the input $n$ in $1/k$ (that is, $k$ symbols are packed into one tape cell) for the simulation of $M_1$, and compresses the input $n$ in $1/(k-8)$ for the simulation of $M_2$. Moreover, $M(k)$ writes a word of length $r_m/(k-8)^7$ on one tape, where $m = \max n$. The computation time for this step is $m + c$, where $c$ is a constant that does not depend on $k$. We may assume that, at the end of this step, $M(k)$ knows the integer $i_1$ such that $r_m/(k-8)^7 = (m + i_1)/(k-8), 0 \leq i_1 < k-8$.

**Step 2.** $M(k)$ simulates each $k$ steps of $M_1$ and each $k-8$ steps of $M_2$ in 8 steps. ($M(k)$ must see three cells in the packed representation of tapes of $M_1, M_2$ in order to simulate $k$ or $k-8$ steps. Hence eight (or seven) steps are necessary. See, for example, [1, p. 290]). The simulation continues until $M(k)$ detects that $M_2$ halts. The computation time for this step is $8r_{f_2}(n)/(k-8)^7$, and $k^r_{f_2}(n)/(k-8)^7$ steps of $M_1$ are simulated during the step. We may assume that, at the end of this step, $M(k)$ knows the integer $i_2$ such that $r_{f_2}(n)/(k-8)^7 = (f_2(n) + i_2)/(k-8), 0 \leq i_2 < k-8$.

**Step 3.** $M(k)$ simulates each $k$ steps of $M_1$ in eight steps. It repeats this $r_m/(k-8)^7+1$ times. The word of length $r_m/(k-8)^7$ written in Step 1 is used for this purpose. The computation time for this step is $8(r_m/(k-8)^7+1)$, and $k(r_m/(k-8)^7+1)$ steps of $M_1$ are simulated during the step.

**Step 4.** $M(k)$ simulates each step of $M_1$ in one step until $M_1$ halts. The computation time for this step is

$$A = f_1(n) + f_2(n) - k(r_{f_2}(n)/(k-8)^7+r_m/(k-8)^7+1).$$

**Step 5.** $M(k)$ counts $B = i_1 + i_2 + (k-8) - c$ and halts.

The machine $M(k)$ halts at

$$(m + c) + 8r_{f_2}(n)/(k-8)^7 + 8(r_m/(k-8)^7+1) + A + B = f_1(n)$$

under the assumption that $A, B \geq 0$.

If we select a sufficiently large $k$, then we have, for almost all $n$,

$$A = f_1(n) + f_2(n) - k(r_{f_2}(n)/(k-8)^7+r_m/(k-8)^7+1)$$

$$\geq f_1(n) + f_2(n) - \frac{k}{k-8} (f_2(n) + m) - 3k$$

$$\geq (1+\varepsilon)(f_2(n) + m) - \frac{k}{k-8} (f_2(n) + m) - 3k$$

$$= \left(1 + \varepsilon - \frac{k}{k-8}\right) (f_2(n) + m) - 3k$$

$$> 0,$$

$$B = i_1 + i_2 + (k-8) - c > 0.$$  

Hence, there is a $k$ such that $M(k)$ halts at $f_1(n)$ for almost all $n$. 


In Step 1, $M(k)$ knows whether $n$ is one of the finite number of exceptional values at time $\max n \leq f_1(n)$. Hence we can modify $M(k)$ so that it halts at $f_1(n)$ for all $n$. \qed

In the following corollary, $k$ denotes a sequence $k_1, \ldots, k_d$ of nonnegative integers and $kn$ denotes $k_1n_1 + \cdots + k_dn_d$.

**Corollary 3.2.** If $f(n) + kn + c$ is tc $(c \in \mathbb{N})$ and $f(n) \in \mathcal{F}_2$, then $f(n)$ is tc.

**Proof.** We can easily show that $f_1(n) = f(n)$, $f_2(n) = kn + c$ satisfy the conditions of Theorem 3.1. \qed

This corollary is useful because, in many cases, preparatory computations such as copying or compressing the input can be carried out within $kn + c$ steps, and hence exactly in $k'n + c'$ steps.

Next we consider the time constructibility of $Lpf(n)_2$, where $p (>0)$ is a rational number.

**Lemma 3.3.** If $f(n)$ is tc and $p (>0)$ is a rational number, then there is a $c (\in \mathbb{N})$ such that $Lpf(n)_2 + \max n + c$ is tc.

**Proof.** The proof follows by simple speed-up of computation. The term $\max n$ is the time to compress the input. \qed

**Corollary 3.4.** If $f(n)$ is tc and $p, q (>0)$ are rational numbers, then $Lpf(n)_2 + L(1 + q)\max n$ is tc.

**Proof.** By Lemma 3.3,

$$g_1(n) = Lpf(n)_2 + \max n + c_1$$

and

$$g_2(n) = L(1 + q)\max n + \max n + c_2$$

are tc for some $c_1, c_2 (\in \mathbb{N})$, and consequently

$$g_3(n) = g_1(n) + g_2(n) + \max n + c_3$$

is tc for some $c_3 (\in \mathbb{N})$ (max $n + c_3$ is the time to copy input). Moreover,

$$g_4(n) = 3\max n + c_1 + c_2 + c_3$$

is tc. Applying Theorem 3.1 to

$$f_1(n) = g_3(n) - g_4(n) = Lpf(n)_2 + L(1 + q)\max n$$

and

$$f_2(n) = g_4(n)$$

we can show that $f_1(n)$ is tc. \qed
Corollary 3.5. If \( f(n) \) is tc, \( p (>0) \) is a rational number, and \( \lfloor pf(n) \rfloor \in \mathcal{F}_2 \), then \( \lfloor pf(n) \rfloor \) is tc.

Proof. By Lemma 3.3, \( \lfloor pf(n) \rfloor + \max n + c \) is tc for some \( c (\in \mathbb{N}) \). The functions \( f_1(n) = \lfloor pf(n) \rfloor \), \( f_2(n) = \max n + c \) satisfy the conditions of Theorem 3.1. \( \square \)

4. Time constructibility of functions in \( \mathcal{F}_2 \)

For the proof of time constructibility of functions in \( \mathcal{F}_2 \), computation in binary is useful. For \( n \in \mathbb{N} \), let \( \text{un}(n) \) denote its unary representation \( 1^n \) and let \( \text{bin}(n) \) denote its binary representation. We assume that \( \text{bin}(n) \) begins with 1 for \( n \geq 1 \). Note that we can transform \( \text{un}(n) \) into \( \text{bin}(n) \) and \( \text{bin}(n) \) into \( \text{un}(n) \) in \( O(n) \) steps.

In the following, when we say that a function \( f(n) \) is computable by a Turing machine \( M \), we assume that the representation of the input \( n = (n_1, \ldots, n_d) \) and that of the result \( f(n) \) of \( M \) may be either in unary or in binary. We also allow the case where some of \( n_1, \ldots, n_d, f(n) \) are represented in unary and others in binary.

Theorem 4.1. For a function \( f(n) \) in \( \mathcal{F}_2 \), if \( f(n) \) is computable by a Turing machine within \( O(f(n)) \) steps, then \( f(n) \) is tc.

Proof. Let \( M \) be a Turing machine that computes \( f(n) \) within \( O(f(n)) \) steps. We construct another Turing machine \( M' \) that performs the following operations. The input \( n \) to \( M' \) is in unary.

Step 1. If some of the inputs \( n_1, \ldots, n_d \) to \( M \) are to be in binary, transform them from unary to binary. The computation time is at most \( c_1 \max n + c_2 \) (\( c_1, c_2 \in \mathbb{N} \)).

Step 2. Compute \( f(n) \) by simulating \( M \). The computation time is at most \( c_3 f(n) + c_4 \) (\( c_3, c_4 \in \mathbb{N} \)).

Step 3. If the result \( f(n) \) is in binary, transform it into unary. The computation time is at most \( c_5 f(n) + c_6 \) (\( c_5, c_6 \in \mathbb{N} \)).

Step 4. Count the number of 1's in the unary representation of \( f(n) \), and halt. The computation time is exactly \( f(n) \).

Let \( g(n) \) be the time at which Step 3 ends. Then both of \( g(n), f(n) + g(n) \) are tc, and we have, for almost all \( n \),

\[
g(n) \leq c_1 \max n + c_2 + c_3 f(n) + c_4 + c_5 f(n) + c_6
\]

\[
\leq c_7 f(n)
\]

(\( c_7 \in \mathbb{N} \)). We show that \( f_1(n) = f(n) \), \( f_2(n) = g(n) \) satisfy the conditions of Theorem 3.1. The first condition \( f(n) \in \mathcal{F}_1 \) is true by our assumption. As for the second
condition, let $\varepsilon_1, \ldots, \varepsilon_4 (>0)$ be real numbers such that

$$(\forall n \in \mathbb{N}) f(n) \geq (1 + \varepsilon_1) \max n,$$

$$(1 + \varepsilon_1)(1 - \varepsilon_2) > 1,$$

$$(1 + \varepsilon_1)(1 - \varepsilon_2) = 1 + \varepsilon_3,$$

$$\varepsilon_4 = \min\{\varepsilon_2/c_7, \varepsilon_3\}.$$ 

Then we have, for almost all $n$,

$$f(n) = \varepsilon_2 f(n) + (1 - \varepsilon_2)f(n)$$

$$\geq (\varepsilon_2 / c_7) g(n) + (1 - \varepsilon_2)(1 + \varepsilon_1) \max n$$

$$= (\varepsilon_2 / c_7) g(n) + (1 + \varepsilon_3) \max n$$

$$\geq \varepsilon_4 g(n) + (1 + \varepsilon_4) \max n.$$

**Theorem 4.2.** For a function $f(n)$ in $\mathcal{F}_1$, if $f(n)$ is tc, then $f(n)$ is computable by a Turing machine within $O(f(n))$ steps.

**Proof.** The proof is similar to that of Theorem 4.1. The computation time is at most

$$c_1 \max n + c_2 \quad (\text{transformation of } n \text{ from binary to unary, if necessary})$$

$$+ f(n) \quad (\text{computation of } f(n) \text{ in unary})$$

$$+ c_3 f(n) + c_4 \quad (\text{transformation of } f(n) \text{ from unary to binary, if necessary})$$

$$= O(f(n))$$

$(c_1, \ldots, c_4 \in \mathbb{N})$. □

**Corollary 4.3.** For a function in $\mathcal{F}_2$, $f(n)$ is tc if and only if it is computable by a Turing machine within $O(f(n))$ steps.

**Corollary 4.4.** If a function $f(n)$ in $\mathcal{F}_2$ is computable by a Turing machine whose running time is bounded by a polynomial $F(\log n_1, \ldots, \log n_d)$ of the lengths $\log n_1, \ldots, \log n_d$ of the input, then $f(n)$ is tc. (Here we assume that the representations of the input $n$ and the result $f(n)$ are in binary.)

**Proof.** Let $\varepsilon (>0)$ be a real number such that $(\forall n \in \mathbb{N}) f(n) \geq (1 + \varepsilon) \max n$. Using the identity $a_1 a_2 \ldots a_d \leq (a_1 + \cdots + a_d)^d$ repeatedly, we can prove that, for almost all $n$,

$$F(\log n_1, \ldots, \log n_d) \leq (1 + \varepsilon) \max n \leq f(n).$$ □
In [2], time constructible functions are defined to be functions $f(n)$ ($\geq n$) that are computable by Turing machines within $O(f(n))$ steps, assuming that the representation of the input $n$ is in unary and that of the output $f(n)$ is in binary. From Corollary 4.3 we immediately see that, for functions (with one variable) in $F_2$, our definition of time constructibility and that in [2] coincide.

The following are examples. In these examples, $p, q$ are positive rational numbers.

Example 1. $\lfloor n^p \rfloor$ ($p \geq 1$) and $\lfloor n^p \rfloor + \lfloor 1 + q \rfloor n$. The proof follows by Corollary 4.4. Let $p$ be $a/b$ ($a, b$ are positive integers). To compute $\lfloor n^p \rfloor$ in binary, we first compute $n^a$, and then find the largest $m$ such that $m^b \leq n^a$. We determine the $(a/b) \log n$ bits of $m$ from left to right. In the following section, we will show that $\lfloor n^p \rfloor + \lfloor 1 + q \rfloor n$ is tc even when $q = 0$ (Example 8).

Example 2. $\max\{n, \lfloor f(n) \rfloor\}$, where $f(n)$ is a polynomial of $n$ with rational coefficients (by Corollary 4.4). Functions $n^2, n^3, \ldots, \lfloor pn \rfloor$ ($p \geq 1$) are special cases.

Example 3. $n!$ (by Theorem 4.1).

Example 4. $2^n, 2^{2^n}$, and so on (by Theorem 4.1).

Example 5. $\lfloor \log n \rfloor, \lfloor \log \log n \rfloor, \ldots, \lfloor \log^* n \rfloor$. Here, $\log^* n$ means the smallest $k$ such that $1 \geq \log \log \ldots \log n$ ($k$ log's). The proof follows by Corollary 4.4. To compute $\log^* n$, we count how many times we can repeatedly apply the function $\log x$ to $n$ until we obtain a value not greater than $1$.

Example 6. $\lfloor \log n \rfloor + \lfloor 1 + q \rfloor n$, $\lfloor \log n \rfloor + \lfloor 1 + q \rfloor n$, $\ldots$, $\lfloor \log^* n \rfloor + \lfloor 1 + q \rfloor n$ (by Corollary 4.4). In the following section, we will show that these functions are tc even when $q = 0$ (Examples 9-11).

Example 7. $n_1, n_2$. Let $g(n_1, n_2) = \max\{n_1, n_2 - 6, 2n_1, 2n_2\}$. Then $g(n_1, n_2) \in F_2$ and $g(n_1, n_2)$ is tc by Corollary 4.4. Let $M$ be a Turing machine that halts at $g(n_1, n_2)$. We construct another Turing machine $M'$ that halts at $n_1, n_2$. If either $n_1 \leq 3$ or $n_2 \leq 3$, then $M'$ detects it at time $\min\{n_1, n_2\} \leq n_1, n_2$ and halts at time $n_1, n_2$. Otherwise, $M'$ moves the heads to the initial positions and starts simulating $M$ at time $6$. Then $M'$ halts at $6 + g(n_1, n_2) = n_1, n_2$.

The function $\lfloor n \log n \rfloor$ is an example of functions for which, at present, we cannot use our results to show their time constructibility. A natural idea to compute $\lfloor n \log n \rfloor$ in binary would be (i) to compute the value $n^n$, and then (ii) determine the largest $k$ such that $2^k \leq n^n$. Step (i) needs to compute the product of two $2^i \log n$ bit numbers for $i = 0, 1, \ldots, \log n - 1$. At present we do not know how to perform this in $O(n \log n)$ steps. (If we can multiply two $m$ bit numbers in $O(m)$ steps, then this would be possible.)

At present, we do not know whether this function $\lfloor n \log n \rfloor$ is tc or not. The functions $\lfloor n \log \log n \rfloor, \lfloor \log n \rfloor^2$ are other examples. Theorem 4.1 implies that, if we can prove that these functions are not tc, then we will obtain large lower bounds for the computation time of these functions. Therefore, to prove it seems to be not easy. However, a similar function $\lfloor \log n \rfloor^{1/2}$ is tc because $\lfloor \log n \rfloor^{1/2} = \left\lfloor n^{\log \log n / 2} \right\rfloor$ and we can apply Corollary 4.4 (see Example 5).
5. Time constructibility of functions in \( \mathcal{F}_1 - \mathcal{F}_2 \)

When we want to speed up a Turing machine \( M \), we construct another Turing machine \( M' \) that first compresses the input \( n \) in \( 1/k \) and then simulates each \( k \) steps of \( M \) in eight steps. If what \( M \) does depends on all components \( n_1, \ldots, n_d \) of the input \( n \) in an essential way, the running time \( f(n) \) of \( M \) will satisfy \( f(n) \gg \max n \). Then the running time \( g(n) \) of \( M' \) will satisfy \( g(n) > 1 \max n \) (the time to compress \( n \)) + \( 8^{r}f(n)/k \gg (1+8/k) \max n \).

This means that the running time of a Turing machine that was constructed to speed up another Turing machine must be in \( \mathcal{F}_2 \). Hence, direct application of speed-up is not useful in proving time constructibility of functions in \( \mathcal{F}_1 - \mathcal{F}_2 \). In this section we introduce one technique for this purpose.

Lemma 5.1. There is a Turing machine \( M \) and one distinguished state \( s_1 \) of \( M \) such that the following is true for almost all \( n \): if we write \( \text{bin}(n) \) on one tape, set the head at the leftmost symbol of \( \text{bin}(n) \), and start \( M \), then there are times \( t_1, t_2 (> t_1) \) such that (1) \( M \) is not in \( s_1 \) before \( t_1 \), (2) \( M \) is in \( s_1 \) at \( t_1 \), (3) \( M \) halts at \( t_2 \), (4) \( t_1 = O(\log n) \), and (5) \( t_2 - t_1 = n \).

Proof. The movement of \( M \) is as follows.

Step 1. \( M \) searches for the smallest \( k (\in \mathbb{N}) \) such that \(|\text{bin}(n)| \leq 2^k \) (\(|\text{bin}(n)| \) denotes the length of \( \text{bin}(n) \)), writes a word \( y \) of length \( 2^k - |\text{bin}(n)| \) to the right of \( \text{bin}(n) \), determines subwords \( x_1, x_2 \) of \( \text{bin}(n) \) such that \( x_1x_2 = \text{bin}(n) \) and \(|x_2| = k+1 \), moves the head to the rightmost cell of \( x_1x_2y (=\text{bin}(n)y) \), and enters state \( s_1 \). Let \( t_1 \) be the time at which \( M \) enters \( s_1 \).

It is easy to show that \( t_1 = O(|\text{bin}(n)|) = O(\log n) \). Let \( a_1, a_2 (\in \mathbb{N}) \) be values such that \( x_1 = \text{bin}(a_1), x_2 = \text{bin}(a_2) \). Then, for sufficiently large \( n \), we have

\[
2^{k-1} < |x_1x_2| \leq 2^k, \quad n = a_12^{k+1} + a_2, \quad a_1 \geq a_2.
\]

Step 2. \( M \) scans \( x_1x_2y \) repeatedly. While \( M \) scans it from right to left, (i) \( M \) decreases the value represented in the \( x_1 \) part by 1 (this is easy to do while scanning \( x_1 \) from the least significant bit to the most—change all 0's to 1's until the first 1 is encountered; change this to a 0), and (ii) if the value represented in the \( x_2 \) part is not 0, then \( M \) decreases it by 1 and writes a letter 1 on another tape in order to construct \( \text{un}(a_2) \) on the tape. This process is repeated until the value of the \( x_1 \) part becomes 0, in other words, \( a_1 \) times. We may assume that, when this happens, \( \text{un}(a_2) \) is already written because \( a_1 \geq a_2 \) (for sufficiently large \( n \)).

Step 3. \( M \) counts the number of 1's in \( \text{un}(a_2) \), and halts. Let \( t_2 \) be the time at which \( M \) halts.

The computation time \( t_2 - t_1 \) for Steps 2 and 3 is \( a_1 \cdot 2|x_1x_2y| + a_2 = a_1 \cdot 2^{k+1} + a_2 = n \). \( \square \)
Proving time constructibility of functions

With respect to this $M$, we say that $M$ is ready for real time count for $n$ at time $t_1$, and we call the computation from time 0 to $t_1$ the preparation for real time count for $n$.

When $f(n)$ is a nondecreasing unbounded function, by $f^{-1}(m)$ we mean $\min\{n \mid m \leq f(n)\}$. For example, if $f(n) = \sqrt{n}$, then $f^{-1}(m) = m^2$. By definition, $f(n) = m$ if and only if $f^{-1}(m) \leq n < f^{-1}(m+1)$.

Now we are ready to state our second result. In its proof, we essentially try to prove that the composition of two tc functions is tc.

**Theorem 5.2.** Suppose that

1. $f_i(n), \ldots, f_s(n)$ are nondecreasing unbounded functions and $\lim_{m \to \infty} (f_i^{-1}(m + 1) - f_i^{-1}(m)) = \infty$ ($1 \leq i \leq s$),

2. $g_{ij}(m)$ ($1 \leq i \leq s, 1 \leq j \leq k_i$) are functions such that, for each $i$ ($1 \leq i \leq s$), a Turing machine $M_i$ computes $k_i + 1$ values $g_{ij}(m)$ ($1 \leq j \leq k_i$), $f_i^{-1}(m + 1) - f_i^{-1}(m)$ from the $k_i$ values $g_{ij}(m - 1)$ ($1 \leq j \leq k_i$) in $O(f_i^{-1}(m) - f_i^{-1}(m - 1))$ steps, and

3. $h(n_1, \ldots, n_s)$ is a tc function.

Then $n + h(f_1(n), \ldots, f_s(n))$ is tc. In $M_s$, for each $j$ ($1 \leq j \leq k_i$), the representations of $g_{ij}(m - 1)$, $g_{ij}(m)$ may be either in unary or in binary, and similarly for the representation of $f_i^{-1}(m + 1) - f_i^{-1}(m)$.

**Proof.** Let $M$ be a Turing machine that halts at $h(n_1, \ldots, n_s)$. We construct another Turing machine $M'$ that halts at $n + h(f_1(n), \ldots, f_s(n))$. The basic idea is to

- compute $g_{ij}(m), f_i^{-1}(m + 1) - f_i^{-1}(m)$ from $g_{ij}(m - 1)$, and
- perform the preparation for real time count for $f_i^{-1}(m + 1) - f_i^{-1}(m)$ if this value is computed in binary between time $f_i^{-1}(m - 1)$ and $f_i^{-1}(m)$.

This is possible for almost all $m$ because both of the necessary computation times for steps (i) and (ii) are $O(f_i^{-1}(m) - f_i^{-1}(m - 1))$ and we can speed up the computation by representing $g_{ij}(m - 1), g_{ij}(m), f_i^{-1}(m + 1) - f_i^{-1}(m)$ in compressed form. (Note that, if $f_i^{-1}(m + 1) - f_i^{-1}(m)$ is in binary, then the length of its binary representation is $O(f_i^{-1}(m) - f_i^{-1}(m - 1))$ because it has been computed within $O(f_i^{-1}(m) - f_i^{-1}(m - 1))$ steps.)

Then, by counting $f_i^{-1}(m + 1) - f_i^{-1}(m)$ starting at time $f_i^{-1}(m)$, $M'$ can detect the time $f_i^{-1}(m + 1)$. (If $f_i^{-1}(m + 1) - f_i^{-1}(m)$ has been computed in unary, then $M'$ simply counts the number of 1's in it.) In this way, for sufficiently large $m_0$, $M'$ can detect the times $f_i^{-1}(m_0), f_i^{-1}(m_0 + 1), \ldots$ under the assumption that the representations of $g_{ij}(m_0 - 1)$ are available at time $f_i^{-1}(m_0 - 1)$ and that $M'$ can detect the times $f_i^{-1}(m_0 - 1)$ and $f_i^{-1}(m_0)$. This is possible by using finite number of states and sufficiently compressing the representations of $g_{ij}(m)$. $M'$ can also detect times $f_i^{-1}(0), f_i^{-1}(1), \ldots, f_i^{-1}(m_0 - 2)$ by using finite number of states.

Thus, $M'$ can detect times $f_i^{-1}(0), f_i^{-1}(1), f_i^{-1}(2), \ldots$ and consequently we may assume that, at any time $t$ such that $f_i^{-1}(m) \leq t < f_i^{-1}(m + 1)$, $u(m)$ is available on
a tape of $M'$. In other words, we may assume that $\text{un}(f_i(t))$ is available at any time $t$.

Then, for almost all $n$, $M'$ can halt at $n + h(f_1(n), \ldots, f_s(n))$ in the following way.

(i) $M'$ performs what we have explained above, and at the same time scans the input $1^n$ to the right until it finds a blank symbol. This happens at time $n$, and at this moment $\text{un}(f_1(n)), \ldots, \text{un}(f_s(n))$ are available.

(ii) Then $M'$ simulates $M$ that is given these $\text{un}(f_1(n)), \ldots, \text{un}(f_s(n))$.

It is easy to modify $M'$ so that it halts at this time for all $n$. □

In the above theorem, if some $g_{ij}(m)$ are represented in unary in $M_i$, then $g_{ij}(f_i(n))$ may appear in $h$. This is because $\text{un}(g_{ij}(f_i(n)))$ is available at time $n$ for all sufficiently large $n$. Moreover, some $f_i(n)$ may not appear in $h$. For example, if $s = 3$ and $g_{21}(m)$, $g_{35}(m)$ are represented in unary in $M_2$, $M_3$, and $h$ has 4 variables, then $n + h(f_1(n), f_2(n), g_{21}(f_2(n)), g_{35}(f_3(n)))$ is tc.

The following are examples. In these examples, $p$ is a positive rational number.

**Example 8.** $n + \sum \text{un} \log n^p$ \(0 < p < 1\). Let $p$ be $a/b$. We select

$$s = 1, \quad f_1(n) = \text{un} n^p, \quad k_1 = 1, \quad g_{11}(m) = m + 1, \quad h(n_1) = n_1.$$

We can compute the binary representations of $g_{11}(m) = m + 1, f_1^{-1}(m + 1) - f_1^{-1}(m)$ from the binary representation of $g_{11}(m - 1) = m$ in $O(f_1^{-1}(m) - f_1^{-1}(m - 1))$ steps based on the easily provable relations

$$m^{b/a} \leq f_1^{-1}(m) \leq m^{b/a} + 1,$$

$$(m + 1)^{b/a} \leq f_1^{-1}(m + 1) \leq (m + 1)^{b/a} + 1.$$

(Note also that $f_1^{-1}(m) - f_1^{-1}(m - 1) \geq (b/(2a))m^{b/a - 1}$ for almost all $m$.) We have $n + h(f_1(n)) = n + \sum \text{un} \log n^p$.

**Example 9.** $n + \sum \log \log n \cdot \text{un}$. We select $s = 1, f_1(n) = \log \log n \cdot \text{un}$ (and hence $f_1^{-1}(m) = 2^m$), $k_1 = 2, g_{11}(m) = m + 1, g_{12}(m) = m^{p \cdot \text{un}}, h(n_1) = n_1$. We can compute the unary representations of $g_{11}(m) = m + 1, g_{12}(m) = m^{p \cdot \text{un}}$ and the binary representation of $f_1^{-1}(m + 1) - f_1^{-1}(m) = 2^m$ from the unary representation of $g_{11}(m - 1) = m$ in $O(f_1^{-1}(m) - f_1^{-1}(m - 1)) = O(2^{m - 1})$ steps. Hence, $n + h(g_{12}(f_1(n))) = n + \sum \log \log n^p \cdot \text{un}$ is tc.

**Example 10.** $n + \sum \log \log \log n \cdot \text{un}$. We select $s = 1, f_1(n) = \log \log n \cdot \text{un}$ (and hence $f_1^{-1}(m) = 2^{2^m}$), $k_1 = 2, g_{11}(m) = m + 1, g_{12}(m) = m^{p \cdot \text{un}}, h(n_1) = n_1$. The representations of $g_{11}(m)$ and $g_{12}(m)$ are unary and that of $f_1^{-1}(m + 1) - f_1^{-1}(m)$ is binary. We have $n + h(g_{12}(f_1(n))) = n + \sum \log \log \log n^p \cdot \text{un}$.

**Example 11.** $n + \sum (\log^* n \cdot \text{un})$. We select $s = 1, f_1(n) = \log n \cdot \text{un}$ (and hence $f_1^{-1}(m) = 2^m$), $k_1 = 2, g_{11}(m) = m + 1, g_{12}(m) = (\log^* m + 1)^p \cdot \text{un}, h(n_1) = n_1$. The representations of $g_{11}(m)$ and $g_{12}(m)$ are unary and that of $f_1^{-1}(m + 1) - f_1^{-1}(m)$ is binary. We have $n + h(g_{12}(f_1(n))) = n + \sum (\log^*(\log n \cdot \text{un}) + 1)^p \cdot \text{un} = n + \sum (\log^* n)^p \cdot \text{un}$.
Example 12. \( n + \log n \cdot \log \log n \cdot n^{3/2} + \log n^{2/5} \). We select

\[
\begin{align*}
\text{s} &= 3, \quad f_1(n) = \log n, \quad k_1 = 2, \quad g_{11}(m) = m + 1, \quad g_{12}(m) = m^2, \\
f_2(n) &= \log \log n, \quad k_2 = 2, \quad g_{21}(m) = m + 1, \quad g_{22}(m) = \log m^{3/2}, \\
f_3(n) &= n^{2/5}, \quad k_3 = 1, \quad g_{31}(m) = m + 1, \quad h(n_1, n_2, n_3) = n_1 n_2 + n_3.
\end{align*}
\]

The function \( h(n_1, n_2, n_3) \) is tc (see Example 7). The representations of \( g_{12}(m) \) and \( g_{22}(m) \) are unary and those of the remaining \( g_{ij}(m) \) and \( f_{ij}^{-1}(m + 1) - f_{ij}^{-1}(m) \) are binary. We have

\[
n + h(g_{12}(f_1(n)), g_{22}(f_2(n)), f_3(n)) = n + \log n \cdot \log \log n \cdot n^{3/2} + \log n^{2/5}.
\]

References
