Modeling time-bounded prefix Kolmogorov complexity [*]

David W. Juedes
School of EE and CS
Ohio University
Athens, Ohio 45701
U.S.A.

Jack H. Lutz
Department of Computer Science
Iowa State University
Ames, Iowa 50011
U.S.A.

Abstract

In the literature, prefix Kolmogorov complexity is defined either in terms of self-delimiting Turing machines or in terms of partial recursive prefix functions. These notions of prefix Kolmogorov complexity are equivalent because, as Chaitin showed, every partial recursive prefix function can be simulated by a self-delimiting Turing machine. However, the simulation given by Chaitin's construction is not efficient, and so questions regarding the time-bounded equivalence of these notions remained unresolved. Here we closely examine these questions.

As our main result, we show that every partial recursive prefix function can be simulated with polynomial efficiency by a self-delimiting Turing machine if and only if P = NP. Thus, it is unlikely that Chaitin's construction can be used to show the polynomial-time equivalence of these notions of prefix Kolmogorov complexity. Here we further examine the relationships between these notions of time-bounded prefix Kolmogorov complexity.

1 Introduction

The theory of minimal programs has given valuable insight into numerous problems in disciplines ranging from combinatorics to thermodynamics. (See Li and Vitányi [13] for various examples.) The principal tool of this theory is a measure of the complexity of individual finite objects in terms of the size of programs that produce them. This tool is *Kolmogorov complexity.*

For a given finite object (string) x, the Kolmogorov complexity of x is $C(x)$, the length of the shortest program that produces x. While intuitively simple, this rough definition has an obvious shortcoming; the exact Kolmogorov complexity of an individual string is not an intrinsic property of the string itself, but is dependent upon the underlying programming model. To see this, notice that it is easy to construct a programming model where every program produces x, and hence $C(x) = 0$. (In this introduction, programming model and machine are synonymous with Turing machine.) As a crucial, early development, Solomonoff [14], Kolmogorov [9], and Chaitin [4] independently

[*]This work was supported in part by National Science Foundation Grants CCR-9157382 (with matching funds from Rockwell International , Microware Systems Corporation, and Amoco Foundation) and CCR-9610461.

showed that there is an *optimal* Kolmogorov complexity measure, and hence that the Kolmogorov complexity of an individual string is indeed an intrinsic property of that string alone.

Theorem 1.1. (**Optimality Theorem**) There exists an *optimal* machine U such that for every machine M there is a constant c such that

$$C_U(x) \leq C_M(x) + c$$

for every x.

If we restrict attention to optimal machines, then $C(x)$ becomes invariant over the choice of machine.

Theorem 1.2.(**Invariance Theorem**) Let U and U' be optimal machines. There exists a constant c such that

$$|C_U(x) - C_{U'}(x)| \leq c$$

for all x.

Thus $C_U(x)$ is truly an intrinsic property of x. We often remove the dependence on U and let $C(x) = C_U(x)$ for some fixed optimal machine U.

Most straightforward variants of Kolmogorov complexity are also intrinsic properties of strings. For instance, a time-bounded version of Kolmogorov complexity is $C^t(x)$, the length of the shortest program that produces x in less than or equal to t steps. This version of Kolmogorov complexity has an *efficient* optimality theorem.

Theorem 1.3. (**Efficient Optimality Theorem**) There exists a machine U such that for all machines M there exists a constant c such that

$$C_U^{c \cdot t \log t + c}(x) \leq C_M^t(x) + c$$

for all x and t.

The efficient optimality theorem serves the same purpose for time-bounded Kolmogorov complexity as the optimality theorem serves for Kolmogorov complexity; it provides a means to examine the complexity of individual strings without relying upon the architecture of the underlying machine. The efficient optimality theorem is invaluable in practice.

In the present paper, we examine a variant of Kolmogorov complexity for which an efficient optimality theorem is desirable — time-bounded prefix Kolmogorov complexity. Very briefly, the prefix Kolmogorov complexity of a string x is $K(x)$, the length of shortest self-delimiting program that produces x. A time-bounded version of $K(x)$ is defined in the obvious way. This variant of Kolmogorov complexity was defined independently by Levin [10, 11], Gács [6], and Chaitin [5] and is crucial to the formulation of algorithmic probability [10, 6, 12], computational depth [3, 8], and algorithmic randomness in terms of Kolmogorov complexity [5, 15]. Similarly, time-bounded prefix Kolmogorov complexity is crucial to formulations of time-bounded algorithmic probability and computational depth.

In the literature, prefix Kolmogorov complexity is defined in terms of *self-delimiting Turing machines*, or in terms of *partial recursive prefix functions.* These versions of prefix Kolmogorov complexity are intrinsic properties of strings since they satisfy optimality theorems. Indeed, these notions of prefix Kolmogorov complexity are the *same* in the absence of time bounds. However, optimality and equivalence results for these notions of prefix Kolmogorov complexity do not easily translate to the time-bounded case. This requires some explanation.

We say that a set S is *prefix-free* if no element $x \in S$ is a proper prefix of any other element in S. A partial recursive prefix function is a partial recursive function whose domain is prefix-free. In contrast, a self-delimiting Turing machine is a Turing machine with a one-way, read-only input tape. An input x to a self-delimiting Turing machine M is *valid* if M halts with the input head reading the last bit of x. This model of computation ensures that the set of valid inputs is prefix-free.

It is natural to define prefix Kolmogorov complexity either in terms of partial recursive prefix functions or in terms of self-delimiting Turing machines. Since it is easy to construct a universal self-delimiting Turing machine, it is straightforward to prove an optimality theorem for this version of prefix Kolmogorov complexity. The proof of an optimality theorem for the version based on partial recursive prefix functions is more involved.

In an early result, Chaitin [5] shows that each partial recursive prefix function f can be simulated by a self-delimiting Turing machine M_f. We give Chaitin's complete construction in Figure 1; an equivalent construction is given by Li and Vitányi [13, p. 192]. Chaitin's construction proceeds as follows. The machine M_f reads the input one bit at a time. Let x be portion of the input that M_f has read. If there is a proper extension of x in the domain of f, then M_f reads the next bit of the input and appends it to x. If x is in the domain of f, then M_f computes $f(x)$ and halts. It is straightforward to see that the set of valid inputs of M_f is exactly the domain of f. Moreover, M_f produces the same output as f, and hence M_f computes the partial recursive function f.

Chaitin's construction may be used to construct an effective enumeration $T'_1, T'_2, \ldots$ of all partial recursive prefix functions. As done in [13], it is possible to use this enumeration to prove an optimality theorem for prefix Kolmogorov complexity based on partial recursive prefix functions. Notice that since every self-delimiting Turing machine computes a partial recursive prefix function, Chaitin's result also shows that the self-delimiting Turing machines compute *exactly* the partial recursive prefix functions. Moreover, Chaitin's result immediately implies that every universal self-delimiting Turing machine computes a universal partial recursive prefix function. Hence, these notions of prefix Kolmogorov complexity are the same.

While these versions of prefix Kolmogorov complexity are the same in the absence of time bounds, the relationships among these versions of prefix Kolmogorov complexity is much less clear in the presence of time bounds. Adding time bounds to prefix Kolmogorov complexity based on

```
$M_f$
let $f$ be a partial recursive prefix function;
let $S$ be the domain of $f$;
let $x = \lambda$;
for each $s \in S$ do
begin
    while $x$ is a proper prefix of $s$ do
    begin
        read another bit of the input;
        append the bit to the end of $x$;
    end;
    if $x = s$ then compute $f(x)$ and halt with its output;
end;
```

Figure 1: Chaitin's construction

self-delimiting Turing machines changes very little. There exist efficient universal self-delimiting machines, and therefore proving an efficient optimality theorem for prefix Kolmogorov complexity based on self-delimiting Turing machines is straightforward. Adding time bounds to prefix Kolmogorov complexity based on partial recursive prefix functions is more problematic.

A straightforward approach to adding time bounds to prefix Kolmogorov complexity based on partial recursive prefix functions is as follows. Let M be a Turing machine that computes a partial recursive prefix function f. Then, $K^t_M(x)$ is the length of the shortest program y such that $M(x) = y$ in $\leq t$ steps. While this approach appears natural, it is problematic for several reasons. First, it is not known whether there exists a uniform means to efficiently compute every partial recursive prefix function. In particular, it is not known whether there exists an efficient universal partial recursive prefix function. Moreover, it is not known whether this version of time-bounded Kolmogorov complexity satisfies an efficient optimality theorem. Resolving these questions may be difficult.

A closer examination of the proof of the optimality theorem for prefix Kolmogorov complexity reveals the difficulty in the time-bounded case. In order to prove the optimality theorem for the version based on partial recursive prefix functions, we first convert each partial recursive prefix function to a self-delimiting Turing machine using Chaitin's construction. Since the construction given by Chaitin may dramatically increase the running time of the original machine, this approach does not give an efficient optimality theorem. With this in mind, it is natural to pose the following question.

(1) *Can Chaitin's construction be improved to show that every partial recursive prefix function can be efficiently simulated by a self-delimiting Turing machine?*

As our main result, we answer this question in the negative; every partial recursive prefix function can be simulated efficiently by a self-delimiting Turing machine if and only if P = NP. Even though the self-delimiting Turing machines compute exactly the partial recursive prefix functions, they do not do so efficiently. In the proof of our main result, we give a partial recursive prefix function that cannot be computed efficiently by a self-delimiting Turing machine unless P=NP.

Since the self-delimiting Turing machines may not be the most efficient means to compute prefix functions, it is natural to ask whether they are a reasonable basis for *time-bounded* prefix Kolmogorov complexity. The answer to this question is presently "yes" since there is no known alternative; however, the final answer to this question depends heavily on answers to the following open questions.

(2) *Does there exist an efficient universal partial recursive prefix function?*

(3) *Does there exist an efficient optimality theorem for prefix Kolmogorov complexity based on partial recursive prefix functions?*

(4) *If the answer to (3) is yes, then are the notions of prefix Kolmogorov complexity based on self-delimiting Turing machines and partial recursive prefix functions efficiently equivalent?*

This paper is organized as follows. In section 2 we present the preliminary notation and terminology that we use throughout the paper. In section 3, we carefully review the fundamental universality and optimality theorems for time-bounded prefix Kolmogorov complexity, and we bound the difference between the two time-bounded notions of prefix Kolmogorov complexity. In section 4 we examine the connection between the complexity of computing partial recursive prefix functions with self-delimiting Turing machines and the complexity of NP. In Theorem 4.2, we prove that the partial recursive prefix functions can be computed efficiently by self-delimiting Turing machines if and only if P=NP. We show that the complexity of computing partial recursive prefix functions with self-delimiting Turing machines is even more closely connected to the complexity of NP. We show that if Chaitin's construction can be improved even slightly then $\mathrm{NP} \subsetneq \mathrm{E}_2$. Finally, we conclude in section 5 with some observations on time-bounded prefix Kolmogorov complexity.

2 Preliminaries

Here we follow most of the accepted conventions in the subject, such as those found in the books by Balcázar, Díaz, and Gabarró [1, 2], Hopcroft and Ullman [7], and Li and Vitányi [13]. However, we work exclusively with the alphabet $\{0,1\}$. Thus the sets $\{0,1\}^*$, $\{0,1\}^n$, and $\{0,1\}^{\leq n}$ denote the set of all strings, the set of strings of length n, and the set of strings of length $\leq n$, respectively.

Throughout the paper we use a standard string pairing function $\langle , \rangle$ defined by $\langle x,y\rangle = bd(x)01y$, where $bd(x)$ is x with each bit doubled (e.g., $bd(101) = 110011$). Here we assume that $\langle x,y,z\rangle = \langle x,\langle y,z\rangle\rangle$, $\langle x,y,z,w\rangle = \langle x,\langle y,z,w\rangle\rangle$, etc. Note that $|\langle x,y\rangle| = 2|x| + |y| + 2$ for all $x,y \in \{0,1\}^*$.

Given two strings x and y, we write $x \sqsubseteq y$ if x is a *prefix* of y, i.e., if there exists a string z such that $xz = y$. (Note that z may be λ, the empty string.) A set X of strings is *prefix-free* if for every $x,y \in X$, $x \sqsubseteq y$ implies $x = y$. For example, the set $A = \{1^k0 | k \in \mathbf{N}\} = \{0, 10, 110, 1110, \ldots\}$ is prefix-free.

Our model of computation is the Turing machine. If M is a Turing machine, we write $M(y)\downarrow$ if M on input y halts after taking finitely many steps. We write $time_M(y)$ for number of steps that M takes on input y. We write

$$\mathrm{PROG}_M = \{y | M(y)\downarrow\}$$

for the set of valid inputs to M. Similarly, we write $\mathrm{PROG}_M(x) = \{y | M(y) = x\}$ and $\mathrm{PROG}^t_M(x) = \{y | M(y) = x \text{ in } \leq t \text{ steps}\}$.

We are primarily interested in Turing machines whose sets of valid inputs are prefix-free. We specifically use a class of Turing machines where this condition is enforced, namely, the *self-delimiting Turing machines*. A self-delimiting Turing machine has a single input tape, a single output tape, and $k-2$ worktapes. Each of the k-tapes is infinite and only 0's, 1's, and blanks may appear on these tapes. Each of the k tapes has a dedicated scanning head. The scanning head for the input tape is read-only and may not move left. At the start of the computation, this head is scanning a blank cell immediately to the left of the input string.

Self-delimiting Turing machines employ a modified halting criterion. The computation of a self-delimiting Turing machine M on input y is a *success*, and we write $M(y)\downarrow$ if M halts after finitely many steps with the input tape head reading the last (rightmost) bit of y. Otherwise, the computation $M(y)$ is a *failure*, and we write $M(y)\uparrow$. With this criterion, it is clear that PROG_M is prefix-free for every self-delimiting Turing machine M.

If M is a standard Turing machine and PROG_M is prefix-free, then M computes a partial recursive prefix function. Notice that M need not be a self-delimiting Turing machine for this to occur. We say that M computes a *universal partial recursive prefix function* if M computes a partial recursive prefix function and, for every partial recursive prefix function f, there exists a string x_f such that $M(x_f \cdot y) = f(y)$ for every y in the domain of f and $M(x_f \cdot y)$ diverges for every y not in the domain of f.

We now precisely define prefix Kolmogorov complexity.

Definition. Let $t \in \mathbf{N}$ and let M be a standard Turing machine that computes a partial recursive prefix function.

1. The *prefix Kolmogorov complexity of* $x \in \{0,1\}^*$ *relative to* M is

$$\widehat{K}_M(x) = \min\Big(\Big\{|\pi| \;\Big|\; M(\pi)\downarrow \text{ and } M(\pi) = x\Big\} \cup \{\infty\}\Big),$$

   the length of the shortest input for M that produces x. The *prefix Kolmogorov complexity of* $x \in \{0,1\}^*$ is

$$\widehat{K}(x) = \widehat{K}_U(x)$$

   for some machine U that computes a universal partial recursive prefix function.

2. The *t-time-bounded (prefix) Kolmogorov complexity* of x *relative to* M is

$$\widehat{K}^t_M(x) = \min\Big(\Big\{|\pi| \;\Big|\; M(\pi) = x \text{ and } \mathit{time}_M(\pi) \le t\Big\} \cup \{\infty\}\Big).$$

When M or U is a self-delimiting Turing machine, we write K for $\widehat{K}$.

As mentioned in the introduction, defining $\widehat{K}^t(x)$ is problematic since it requires an appropriate function or machine of reference. We address this issue in the following sections.

3 Efficient Universality and Optimality

In this section we review fundamental theorems for time-bounded prefix Kolmogorov complexity and provide precise statements of the results mentioned in the introduction. Propositions 3.1–3.4 below may be found in the text [13].

Proposition 3.1. There exists an *efficient universal* self-delimiting Turing machine. That is, there exist a polynomial p and a self-delimiting Turing machine U that satisfy the following condition. For every self-delimiting Turing machine M, there exist a prefix π_M and a constant c such that $U(\pi_M\pi) = M(\pi)$ and $\mathit{time}_U(\pi_M\pi) \le c \cdot p(\mathit{time}_M(\pi))$ for every program π.

The polynomial p in Proposition 3.1 may be improved to $O(n \log n)$ using standard techniques [7]. This immediately implies an efficient optimality theorem for this version of prefix Kolmogorov complexity.

Proposition 3.2.(Efficient Optimality for K) There exists a self-delimiting Turing machine U such that for every self-delimiting Turing machine M there exists a constant c such that

$$K_U^{c\cdot t\log t+c}(x) \le K^t_M(x) + c$$

for every $x \in \{0,1\}^*$ and $t \in \mathbf{N}$. □

As usual, we now fix a self-delimiting Turing machine U as in Proposition 3.2 and let $K^t(x) = K^t_U(x)$. We may not be able to do the same for $\widehat{K}$ since the existence of an efficient universal partial recursive prefix function is open. However, the existence of a universal partial recursive prefix function is well-known.

Proposition 3.3. There exists a universal partial recursive prefix function. That is, there exists a partial recursive prefix function U such that for every partial recursive prefix function f there exists a string π_f such that $U(\pi_f \cdot y) = f(y)$ for each y in the domain of f and $U(\pi_f \cdot y)$ diverges for every y not in the domain of f.

The existence of a universal partial recursive prefix function immediately implies optimality.

Proposition 3.4. (Optimality for $\widehat{K}$) There exists a partial recursive prefix function U such that for every partial recursive prefix function M there exists a constant c such that

$$\widehat{K}_U(x) \leq \widehat{K}_M(x) + c$$

for every $x \in \{0,1\}^*$.

Constructing an efficient universal partial recursive prefix function appears to be difficult. As we show in section 4, Chaitin's construction will not suffice for this unless P=NP. Moreover, it is easy to see that the set of indices of standard Turing machines that compute partial recursive prefix functions is not recursively enumerable and so a brute force approach to construct a universal machine will not work. However, this does not preclude the possibility that another construction will produce such a machine.

Here we conjecture that, indeed, another construction will not work. In conjectures 3.5 and 3.6 below, we surmise that there does not exist an efficient universal partial recursive prefix function and there does not exist an efficient optimal partial recursive prefix function. The terms "efficient universal partial recursive prefix function" and "efficient optimal partial recursive prefix function" require some explanation since they do not appear explicitly in the conjectures. The term efficient universal partial recurvive prefix function actually refers to a Turing machine that computes a partial recursive prefix function and efficiently simulates every Turing machine (in the standard enumeration) that computes a partial recursive prefix function. The term efficient optimal partial recursive prefix function again refers to a Turing machine U that computes a partial recursive prefix function. In this case, the additional requirement is that time-bounded Kolmogorov complexity with respect to U is within an additive constant of the time-bounded Kolmogorov complexity for any other Turing machine that computes a partial recursive prefix function. Consider the precise conjectures below.

Conjecture 3.5. For every polynomial p, there is no Turing machine U that computes a partial recursive prefix function and satisfies the following requirement: for every Turing machine M that computes a partial recursive prefix function, there exist a string π_M and a constant c such that $U(\pi_M \cdot y) = M(y)$ for each y in the domain of M ($U(\pi_M \cdot y)$ diverges for each y not in the domain of M) and $time_U(\pi_M \cdot y) \leq c \cdot p(time_M(y))$.

Conjecture 3.6. For every polynomial p, there is no Turing machine U that computes a partial recursive prefix function and satisfies the following requirement: for every Turing machine M that computes a partial prefix function, there exists a constant c such that

$$\widehat{K}_U^{c \cdot p(t)}(x) \leq \widehat{K}_M^t(x) + c$$

for each x and $t \in \mathbf{N}$.

As we show in section 4, both Conjecture 3.5 and 3.6 imply that P $\neq$ NP, and so these conjectures are likely to be difficult to directly resolve in the affirmative. However, both conjectures may be refutable without resolving whether P = NP. A natural question is whether there is a complexity-theoretic hypothesis that implies either conjecture.

If conjecture 3.6 is false, then this variant of prefix Kolmogorov complexity is indeed an intrinsic property of strings. In this case, it is natural to ask whether there is a significant difference between K and $\widehat{K}$ in the time-bounded case. Here, we provide some simple bounds on the difference between these two variants of prefix Kolmogorov complexity.

Since every self-delimiting Turing machine computes a partial recursive prefix function, it is easy to modify any partial recursive prefix function to also efficiently simulate every self-delimiting Turing machine. Thus it is reasonable to expect that

$$\widehat{K}_M^{c \cdot t \log t + c}(x) \leq K^t(x) + c \tag{1}$$

for any M that is a potential candidate for an efficient optimal partial recursive prefix function. However, it is possible that there is an M satisfying (1) such that $\widehat{K}_M^t(x)$ is much smaller than $K^t(x)$. Here we bound the difference between K^t and $\widehat{K}_M^t$. (The following theorem was pointed out to the authors by Harry Buhrman.)

Theorem 3.7. For every partial recursive prefix function M there exists a constant c such that

$$K^{c \cdot t (\log t)^2 + c}(x) \leq \widehat{K}_M^t(x) + O(\log |x|)$$

for every $x \in \{0,1\}^*$ and $t \in \mathbf{N}$.

Proof. A straightforward modification of a proof in the text [13, p. 194] provides upper and lower bounds on $K^t(x)$ and $\widehat{K}_M^t(x)$ in terms of $C^t(x)$. In particular, for each partial recursive prefix function M there exists a constant c_1 such that

$$C^{c_1 t \log t + c_1}(x) \leq \widehat{K}_M^t(x) + c_1 \tag{2}$$

for each $x \in \{0,1\}^*$ and $t \in \mathbf{N}$. Similarly, there exists a constant c_2 such that

$$K^{c_2 t \log t + c_2}(x) \leq C^t(x) + 2\log C^t(x) + c_2 \tag{3}$$

for each $x \in \{0,1\}^*$ and $t \in \mathbf{N}$. Combining these inequalities with an appropriate choice of a constant c gives the following.

$$\begin{aligned} K^{c\cdot t(\log t)^2 + c}(x) &\overset{(1)}{\leq} C^{c_1 \cdot t \log t + c_1}(x) + 2\log(C^{c_1\cdot t\log t + c_1}(x)) + c_2 \\ &\overset{(2)}{\leq} \widehat{K}^t_M(x) + 2\log(C_U^{c_1\cdot t\log t + c_1}(x)) + c_2 + c_1 \\ &\leq \widehat{K}^t_M(x) + 2\log(|x|) + c. \end{aligned}$$

□

4 Simulation and NP

As mentioned in the introduction, both Conjecture 3.5 and 3.6 are false if we can efficiently compute every partial recursive prefix function with a self-delimiting Turing machine. In this section we show that this avenue to resolve Conjectures 3.5 and 3.6 will not work unless P=NP. We begin by precisely formulating this hypothesis.

Hypothesis 4.1. **(The efficient simulation hypothesis)** For every partial recursive prefix function M, there exist a self-delimiting Turing machine M' and a *polynomial* p such that $M(x) = M'(x)$ and $time_{M'}(x) \leq p(time_M(x))$ for every $x \in \{0,1\}^*$.

The efficient simulation hypothesis is a statement about the complexity of computing partial recursive prefix functions by self-delimiting Turing machines. As we show in this section, the complexity of computing partial recursive prefix functions by self-delimiting Turing machines is closely tied to the complexity of NP.

Theorem 4.2. **(Main Theorem)** The efficient simulation hypothesis holds if and only if P=NP.

Proof. We begin by showing that the efficient simulation hypothesis is true if we can perform prefix searching efficiently.

Assume that P = NP, and consider the language

$$A = \left\{ \langle x, y, z\rangle \;\middle|\; \exists w \in \{0,1\}^{\leq |z|} \text{ such that } \begin{array}{ll} (i) & y \sqsubseteq w \text{ and} \\ (ii) & M_x(w) \text{ halts in } |z| \text{ steps.} \end{array} \right\}.$$

It is clear that $A \in$ NP. Since P $=$ NP, there exists a fixed polynomial p such that $A \in \mathrm{DTIME}(p)$. We will use a p-time bounded machine for A to do efficient prefix searching.

Let M be a partial recursive prefix function, and let x be the index of this machine in the standard enumeration. (We assume without loss of generality that $time_M(x) \geq |x|$ for all valid inputs x. If this is not true, then M is not a prefix machine.) We now give a self-delimiting Turing M' that efficiently simulates M_x with the help of efficient prefix searching. (See Figure 2.) It is easy

```
$M'$:
begin
          $z = \lambda$;
          $y = \lambda$;
(1)       while $\langle x, y, z \rangle \notin A$ do
                    $z = z0$;
(2)       while TRUE
          begin
          (2.1)     if $M_x(y)$ halts in $|z|$ steps
                    then output $M_x(y)$ and halt;
                    Else
                            move input tape head to the right;
                            $y = y$[next input bit];
          (2.2)     while $\langle x, y, z \rangle \notin A$ do
                            $z = z0$;
          end;
end.
```

Figure 2: The machine M' from Theorem 4.2

to see that M' is a self-delimiting Turing machine since the input tape head for M' moves only to the right. Moreover, M' halts with the input tape head reading the last bit of y if and only if M halts and outputs $M(y) = M'(y)$. To see that M' operates in the required time bound, fix $y \in \{0,1\}^*$, assume that M halts on y, and let $s = time_M(y)$. We will show that $time_{M'}(y) \leq p'(time_M(y))$, where p' is a polynomial satisfying $p'(n) \geq 3 \cdot n \cdot p(c \cdot n)$ for for some $c \in \mathbf{N}$ and all n.

During the operation of M' on y, the string z never has more than s characters. It follows that there is a constant c such that $|\langle x, y, z \rangle| \leq c \cdot s$ for every x, y, and z. With this in mind, it is easy to see that the first while loop requires at most $s \cdot p(c \cdot s)$ steps. Likewise, the second while loop performs at most $|y|$ iterations. Statement (2.1) of the loop consumes at most $|y| \cdot s \leq s \cdot p(c \cdot s)$

steps. Statement (2.2) of the loop consumes at most $s \cdot p(c \cdot s)$ steps. It follows that

$$time_{M'}(y) \leq 3 \cdot s \cdot p(c \cdot s) \leq p'(time_M(y)).$$

Thus M' efficiently simulates M.

We have just shown that the efficient simulation hypothesis is true if P=NP. To show the converse, we show that the efficient simulation hypothesis enables us to decide SAT efficiently.

To begin, construct a partial recursive prefix function M that behaves as follows.

(i) M on input $\langle x, y\rangle$ outputs 0 if x is a boolean formula with n variables and $y \in \{0,1\}^n$ is a satisfying assignment to x.

(ii) M on input $\langle x, \lambda\rangle$ outputs 1 if x is a boolean formula with no satisfying assignments.

(iii) M runs forever on any input that does not satisfy conditions (i) or (ii) above.

By our choice of pairing function, it is clear that the set of valid inputs (programs) to M is a prefix-free set.

The machine M halts on two types of programs: programs of the form $\langle x, \lambda\rangle$, and programs of the form $\langle x, y\rangle$. M runs fastest on strings of the form $\langle x, y\rangle$. In fact, there exists a constant c such that $time_M(\langle x, y\rangle) \leq c \cdot |\langle x, \lambda\rangle|^2$, while $time_M(\langle x, \lambda\rangle) \leq c \cdot 2^{|\langle x, \lambda\rangle|}$. Since $\langle x, \lambda\rangle \sqsubseteq \langle x, y\rangle$ for every y, we can efficiently determine whether $\langle x, \lambda\rangle$ is a valid program if we can efficiently simulate M on all valid programs of the form $\langle x, y\rangle$.

Assume the efficient simulation hypothesis. Fix a self-delimiting Turing machine M' and a polynomial p such that M' simulates M on all valid programs y and $time_{M'}(y) \leq p(time_M(y))$. In this case, we can use M' to solve SAT in polynomial time. Consider the machine M_{SAT} in Figure 3.

It is clear that M_{SAT} runs in polynomial time. Moreover, if x has a satisfying assignment, then there exists a y such that $M(\langle x, y\rangle)$ must accept in $\leq c \cdot |\langle x, \lambda\rangle|^2$ steps. Thus $M'(\langle x, y\rangle)$ must read past $\langle x, \lambda\rangle$ in $p(c \cdot |\langle x, \lambda\rangle|^2)$ steps. In this case, M_{SAT} accepts correctly. Otherwise, $M_{\mathrm{SAT}}(x)$ rejects correctly. It follows that M_{SAT} accepts SAT in polynomial time. Since SAT is NP-complete, it follows that P = NP. □

The complexity of NP is even more closely tied to the complexity of computing partial recursive prefix functions by self-delimiting Turing machines than Theorem 4.2 indicates. If we relax our notion of "efficient" in the efficient simulation hypothesis, then we arrive at other consequences concerning the complexity of NP. Consider the following theorem.

Theorem 4.3. Assume that $t : \mathbf{N} \to \mathbf{N}$ is a nondecreasing, time-constructible function such that for every partial recursive prefix function M there is a self-delimiting Turing machine M' such that for every $x \in \{0,1\}^*$, $M(x) = M'(x)$ and $time_{M'}(x) \leq t(time_M(x))$. Then, NP $\subseteq \bigcup_{c=0}^{\infty}$ DTIME$(t(n^c))$.

```
$M_{\mathrm{SAT}}(x)$:
begin
    if $x$ is not a boolean formula
        reject;
    else
        Simulate $M'(\langle x, \lambda\rangle)$ for $p(c \cdot (|\langle x, \lambda\rangle|^2))$ steps.
        if $M'$ moves past $\langle x, \lambda\rangle$ on the input tape
            accept $x$;
        else
            reject $x$;
end;
```

Figure 3: The machine M_{SAT} from Theorem 4.2

Proof. If we assume the hypothesis, then a straightforward modification of the construction of the machine M_{SAT} from the proof of Theorem 4.2 shows that SAT is decidable in DTIME($t(c \cdot n^2)$). Since SAT is NP-complete, it follows immediately that NP $\subseteq \bigcup_{c=0}^{\infty}$ DTIME($t(n^c)$). □

If t is a *quasipolynomial* function, i.e., $t = O(n^{(\log n)^k})$ for some k, then improving Chaitin's construction to be $t(n)$-efficient implies that

$$\mathrm{NP} \subseteq \mathrm{P}_2 = \{L | L \text{ is decidable in DTIME}(f) \text{ for some quasipolynomial } f\}.$$

If t is a *subexponential* function, i.e., for every $\epsilon > 0$, $t(n) \leq 2^{n^\epsilon}$ for all but finitely many n, then improving Chaitin's construction to be $t(n)$-efficient implies that NP $\subsetneq$ DTIME(2^n).

Since Chaitin's construction gives a simulation where prefix machines are simulated by self-delimiting machines at the expense of an exponential blowup in the running time, it is reasonable to ask if Chaitin's construction can be improved even slightly. Theorem 4.3 shows that even a slight improvement separates NP from E_2.

5 Conclusion

We have shown that self-delimiting Turing machines cannot efficiently simulate all partial recursive prefix functions unless P=NP. Three questions remain open:

(i) Is there an efficient optimality theorem for prefix Kolmogorov complexity based on partial recursive prefix functions?

(ii) Is there an efficient universal partial recursive prefix function?

(iii) Are the two notions of prefix Kolmogorov complexity efficiently equivalent?

In the absence of an affirmative answer to (i), our main theorem is *prima facie* evidence that time-bounded prefix Kolmogorov complexity should be based on self-delimiting Turing machines and not on partial recursive prefix functions.

Acknowledgments The authors thank Harry Buhrman and Giora Slutzki for useful discussions. We especially thank Harry Buhrman for pointing out Theorem 3.7.

References

[1] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, 1988.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II.* Springer-Verlag, Berlin, 1990.

[3] C. H. Bennett. Logical depth and physical complexity. In R. Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pages 227–257. Oxford University Press, Oxford, 1988.

[4] G. J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM*, 16:145–159, 1969.

[5] G. J. Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.

[6] P. Gács. On the symmetry of algorithmic information. *Soviet Mathematics Doklady*, 15:1477–1480, 1974.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.

[8] D. W. Juedes, J. I. Lathrop, and J. H. Lutz. Computational depth and reducibility. *Theoretical Computer Science*, 132:37–70, 1994.

[9] A. N. Kolmogorov. Three approaches to the quantitative definition of 'information'. *Problems of Information Transmission*, 1:1–7, 1965.

[10] L. A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problems of Information Transmission*, 10:206–210, 1974.

[11] L. A. Levin. Various measures of complexity for finite objects (axiomatic description). *Soviet Mathematics Doklady*, 17:522–526, 1976.

[12] L. A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[13] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer–Verlag, New York, second edition, 1997.

[14] R. J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22, 224–254, 1964.

[15] V. V. V'jugin. The algebra of invariant properties of finite sequences. *Problems of Information Transmission*, 18:147–161, 1982.