On the time complexity of 2-tag systems and small universal Turing machines

Damien Woods*
Boole Centre for Research in Informatics
Department of Mathematics
University College Cork, Ireland.
d.woods@bcri.ucc.ie

Turlough Neary*
TASS, Department of Computer Science
National University of Ireland, Maynooth, Ireland.
tneary@cs.may.ie

Abstract

We show that 2-tag systems efficiently simulate Turing machines. As a corollary we find that the small universal Turing machines of Rogozhin, Minsky and others simulate Turing machines in polynomial time. This is an exponential improvement on the previously known simulation time overhead and improves a forty year old result in the area of small universal Turing machines.

1 Introduction

It has been an open question for forty years as to whether the smallest known universal Turing machines (UTMs) are efficient simulators of Turing machines. This question is intimately related to a problem regarding the computational complexity of 2-tag systems.

Shannon [24] was the first to consider the question of finding the smallest possible UTM, where size is the number of states and symbols. Early attempts [6, 27] gave small UTMs that efficiently (in polynomial time) simulate Turing machines.

In the early 1960s Cocke and Minsky [2] showed that 2-tag systems simulate Turing machines, but in an exponentially slow fashion. Minsky [15] found a small 7-state, 4-symbol UTM that simulates 2-tag systems in polynomial time. So this small UTM simulates Turing machines via the following sequence of simulations

$$\text{Turing machine} \mapsto \text{2-tag system} \mapsto \text{small UTM} \qquad (1)$$

where $A \mapsto B$ denotes that A is simulated by B. Later Rogozhin [23] and others [1, 8, 22] used Minsky's technique of simulation via (1) to find small UTMs for a range of state-symbol pairs (see Figure 1). All of these small UTMs efficiently simulate 2-tag systems. However since the 2-tag simulation of Turing machines is exponentially slow it has remained an open problem as to whether these UTMs can be made to run in polynomial time.

For our result we replace (1) with the following sequence of simulations

$$\text{Turing machine} \mapsto \text{cyclic tag system}$$
$$\mapsto \text{2-tag system} \mapsto \text{small UTM}$$

Specifically, in this paper we show that 2-tag systems efficiently simulate cyclic tag systems. In a recent paper [16] we have shown that cyclic tag systems efficiently simulate Turing machines. Thus the present paper provides an important piece of the puzzle as far as the computational complexity of small UTMs and 2-tag systems are concerned. Our main results states:

Theorem 1 *Given a single tape deterministic Turing machine M that computes in time t then there is a 2-tag system T_M that simulates the computation of M and computes in polynomial time $O(t^4(\log t)^2)$.*

This immediately gives the following interesting result.

Corollary 1 *The small UTMs of Minsky, Rogozhin and others [1, 8, 15, 22, 23] are polynomial time, $O(t^8(\log t)^4)$, simulators of Turing machines.*

Before our result it was entirely plausible that there was an exponential trade-off between UTM program size complexity, and time/space complexity; the smallest UTMs seemed to be exponentially slow. However our result shows there is currently little evidence for such a claim.

Early examples of efficient small UTMs were found by Ikeno and Watanabe [6, 27]. Prior to the present paper the smallest known polynomial time UTMs were to be found in [17]. However these efficient machines are not as small as those of Rogozhin et al., hence the present paper represents a significant size improvement when considering small polynomial time UTMs. This improvement is illustrated in Figure 1.

*The authors are funded by the Irish Research Council for Science, Engineering and Technology.

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

There are numerous other applications of Theorem 1. The technique of simulation via 2-tag systems is at the core of many results in the broad survey by Margenstern [14]; our result exponentially improves the time overheads in many [10, 11, 12, 13, 21] of these constructions. For another example, Levin and Venkatesan [9, 25] used the small 8-state, 5-symbol, polynomial time UTM of Watanabe's [27] to show the average case NP-completeness of a graph colouring problem. Our construction gives polynomial time UTMs that are significantly smaller than Watanabe's and thus improves (lowers) the number of colours in their construction.

In the present paper, the phrase "small UTMs" refers to Turing machines that obey the standard definitions. Recently Cook [3] has found universal machines that are smaller than those discussed in the present paper. Cook's machines simulate the cellular automaton Rule 110, which Cook showed to be universal via an impressive simulation. However Cook's UTMs are generalisations of standard Turing machines: their blank tape consists of an infinitely repeated word to the left and another to the right. Intuitively, this change of definition makes quite a difference, especially since Cook encodes a (possibly universal) program in one of these repeated words. It should be noted that Cook's UTMs are exponentially slow; in a recent paper [16] we have improved their simulation time overhead to polynomial.

1.1 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, Σ denote a finite alphabet, and ϵ denote the empty word. The function $\| \; \| : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N}$ is written as $\|w_1, w_2\|$ and gives the number of occurrences of the word w_2 in w_1. For example $\|1101001, 01\| = 2$. All logs are to the base 2. Throughout the paper a and $\#$ are constant symbols, and $x, x_i \in \{0, 1\}$.

2 2-tag systems

Tag systems where introducted by Post [20], and 2-tag systems were shown to be universal by Cocke and Minsky [2].

Definition 1 (2-tag system) *A tag system consists of a finite alphabet of symbols Σ, a finite set of rules $R : \Sigma \rightarrow \Sigma^*$ and a deletion number $\beta \in \mathbb{N}$, $\beta \geqslant 1$. For a 2-tag system $\beta = 2$.*

The 2-tag systems we consider are deterministic (or equivalently, monogenic [4, 26]). The computation of a 2-tag system acts on a data word $w = \sigma_1\sigma_2 \ldots \sigma_l$. The entire configuration is given by w. In a computation step, the symbols $\sigma_1\sigma_2$ are deleted and if there is a rule for σ_1, i.e. a rule of the form $\sigma_1 \rightarrow \sigma_{l+1} \ldots \sigma_{l+c}$, then the word $\sigma_{l+1} \ldots \sigma_{l+c}$ is appended. We write $w_1 \vdash w_2$ when the data word (configuration) w_2 is obtained from w_1 via a single computation step:

$$\sigma_1\sigma_2\sigma_3 \ldots \sigma_l \vdash \sigma_3 \ldots \sigma_l\sigma_{l+1} \ldots \sigma_{l+c}$$

where $\sigma_1 \rightarrow \sigma_{l+1} \ldots \sigma_{l+c} \in R$. A 2-tag system completes its computation if (i) $|w| < \beta$, or (ii) it enters a repeating sequence of configurations, or (iii) there is no rule for the leftmost symbol. The complexity measures of time and space are defined in the obvious way. Given a word w, we use the term *round* to describe $\lceil |w|/2 \rceil$ computation steps that traverse w exactly once [4, 26].

We often write 2-tag symbols in pairs. The second (even numbered) symbol is dotted to distinguish it from the first. In the sequel we encode binary symbols in the following way, 1 is encoded as $1\dot{1}$ and 0 as $0\dot{0}$. Also a single pair of symbols is distinguished by being 'barred': $\bar{0}\bar{\dot{0}}$ or $\bar{1}\bar{\dot{1}}$. So an example encoding of the word 11010 is $\bar{1}\bar{\dot{1}}1\dot{1}0\dot{0}1\dot{1}0\dot{0}$.

Lemma 1 *Let $w = \bar{x}_0\bar{\dot{x}}_0x_1x_2 \ldots x_l \in \{\bar{0}\bar{\dot{0}}, \bar{1}\bar{\dot{1}}\}\{0, 1\}^*$. Then there is a 2-tag system T that tests whether $|w|$ is odd or even in exactly $\lfloor |w|/2 \rfloor + 1$ timesteps.*

Proof The 2-tag system has 6 rules, $R = \{\bar{x} \rightarrow \underset{1}{\bar{x}}\underset{1}{\bar{\dot{x}}}, x \rightarrow \epsilon, \underset{1}{\bar{x}} \rightarrow \underset{2}{x}\}$ where $x \in \{0, 1\}$. Initially T reads the leftmost symbol $\bar{x}_0$ of w. After one round, if the read symbol is dotted then the output is the single symbol $\underset{1}{\bar{\dot{x}}}$ signifying that $|w|$ is odd. Otherwise the output is $\underset{2}{x}$ signifying that $|w|$ is even. $\blacksquare$

We define the *parity* of a word to be *odd* if the read symbol is undotted and *even* if the read symbol is dotted.

Despite its simplicity, the proof idea of Lemma 1 constitutes one of the main ingredients of Cocke and Minsky's (exponentially slow) 2-tag simulation of Turing machines [2]. The 2-tag system data word encodes an arbitrary TM configuration as two unary numbers. The left side of the tape is encoded as one unary number, the right side as another. Their simulation makes use of repeated tests for oddness and evenness of data word length. Also doubling and halving of data word length is used to read and write to the simulated tape.

3 Cyclic tag systems

Cyclic tag systems were introduced by Cook [3].

Definition 2 (cyclic tag system [3]) *A cyclic tag system $C = \alpha_{p-1}, \alpha_{p-2}, \ldots, \alpha_0$ is a list of binary words $\alpha_m \in \{0, 1\}^*$ called appendants.*

A *configuration* of a cyclic tag system consists of (i) a *marker* that points to a single appendant α_m in C, and (ii) a word $w = x_0x_1 \ldots x_l \in \{0, 1\}^*$. We call w the *data word*.

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

Intuitively the list C is a *program* with the marker pointing to instruction α_m. At the initial configuration the marker points to appendant α_0 and w is the binary input word.

Definition 3 (computation step of a cyclic tag system) *A computation step is deterministic and acts on a configuration in one of two ways:*

- *If $x_0 = 0$ then x_0 is deleted and the marker moves to appendant $\alpha_{(m+1 \mod p)}$.*
- *If $x_0 = 1$ then x_0 is deleted, the word α_m is appended onto the right end of w, and the marker moves to appendant $\alpha_{(m+1 \mod p)}$.*

A cyclic tag system completes its computation if (i) the data word is the empty word, or (ii) it enters a forever repeating sequence of configurations. The complexity measures of time and space are defined in the obvious way.

Example *(cyclic tag system computation)* Let $C = 00, 010, 11$ be a cyclic tag system with input word 011. Below we give the first four steps of the computation. In each configuration C is given on the left with the marked appendant highlighted in bold font.

$$\begin{array}{lll} & \mathbf{00}, 010, 11 \quad 011 & \vdash \quad 00, \mathbf{010}, 11 \quad 11 \\ \vdash & 00, 010, \mathbf{11} \quad 1010 & \vdash \quad \mathbf{00}, 011, 11 \quad 01011 \\ \vdash & 00, \mathbf{011}, 11 \quad 1011 & \vdash \quad \ldots \end{array}$$

We write an arbitrary single step of a cyclic tag system computation as

$$\begin{array}{l} \alpha_0, \ldots, \alpha_{m-1}, \boldsymbol{\alpha_m}, \alpha_{m+1}, \ldots, \alpha_{p-1} \quad x_0 x_1 \ldots x_l \\ \vdash \alpha_0, \ldots, \alpha_m, \boldsymbol{\alpha_{m+1}}, \ldots, \alpha_{p-1} \quad x_1 \ldots x_l x_{l+1} \ldots x_{l+c} \end{array} \qquad (2)$$

where $x \in \{0, 1\}$, and as usual if $x_0 = 0$ then $x_{l+1} \ldots x_{l+c} = \epsilon$, otherwise if $x_0 = 1$ then $x_{l+1} \ldots x_{l+c} = \alpha_m \in \{0, 1\}^c, c \in \mathbb{N}$.

Cook [3] used the universality of cyclic tag systems to show that Rule 110, a binary one-dimensional cellular automaton, is universal. Recently we have improved on Cook's work by showing that cyclic tag systems simulate Turing machines in polynomial time:

Theorem 2 ([16]) *Let M be a single-tape deterministic Turing machine that computes in time t. Then there is a cyclic tag system C_M that simulates the computation of M in time $O(t^3 \log t)$.*

In order to calculate this upper bound we substitute space bounds for time bounds whenever possible in the analysis.

4 2-tag systems efficiently simulate cyclic tag systems

In this section we prove Theorem 1.

4.1 Encoding

Cyclic tag systems use a binary alphabet and program control is determined by the read symbol and the value of the program instruction marker. On the one hand 2-tag systems seem more general than cyclic tag systems as an arbitrary constant (independent of input length) sized alphabet is permitted. On the other hand 2-tag systems seem more restricted in that program control is determined solely by the read symbol.

Because of this restriction we use a large number of symbols in our construction. The number of such symbols is a constant that is independent of input length, but is dependent on our simulation algorithm and the size of the simulated cyclic tag system program. In our encoding we decorate symbols with dots ($\dot{x}, \ddot{x}$), bars ($\bar{x}$) and underindexes ($\underset{j}{x}$). These decorations are used for algorithm control flow.

Definition 4 (2-tag input encoding) *The cyclic tag system input data word $w = x_0 x_1 \ldots x_n \in \{0, 1\}^*$ is encoded as the 2-tag data word*

$$\hat{w} = \underset{1}{\bar{x}}_0 \underset{1}{\dot{\bar{x}}}_0 \ \underset{1}{x}_1 \underset{1}{\dot{x}}_1 \ \underset{1}{x}_2 \underset{1}{\dot{x}}_2 \ldots \underset{1}{x}_n \underset{1}{\dot{x}}_n \ \underset{1}{a}\underset{1}{\dot{a}} \ \underset{1}{a}\underset{1}{\dot{a}} \ldots \underset{1}{a}\underset{1}{\dot{a}}$$

where the number of $a\dot{a}$ pairs in $\hat{w}$ is $\|\hat{w}, a\dot{a}\| = 2^{\lceil \log(n+1) \rceil}$ and the extra whitespace between symbol pairs is for human readability purposes only.

The subword $a\dot{a}\, a\dot{a} \ldots a\dot{a}$ is used as a *counter* and its value $\|\hat{w}, a\dot{a}\|$ is used extensively in our algorithms below.

An arbitrary (not necessarily input) cyclic data word is encoded as in Definition 4 except that the counter is 'embedded' in w. Specifically if $w = x_0 x_1 \ldots x_l$ then

$$\hat{w} = \underset{j}{\bar{x}}_0 \underset{j}{\dot{\bar{x}}}_0 \ \underset{j}{x}_1 \underset{j}{\dot{x}}_1 \ldots \underset{j}{x}_i \underset{j}{\dot{x}}_i \ \underset{j}{a}\underset{j}{\dot{a}} \ldots \underset{j}{a}\underset{j}{\dot{a}} \ \underset{j}{x}_{i+1} \underset{j}{\dot{x}}_{i+1} \ldots \underset{j}{x}_l \underset{j}{\dot{x}}_l \qquad (3)$$

for some $i \in \{0, \ldots, l\}$ and $j \in \mathbb{N}$. As above, the counter value is the next power of 2 strictly greater than l

$$\|\hat{w}, a\dot{a}\| = 2^{\lceil \log(l+1) \rceil} \qquad (4)$$

This encoding is computable in logspace.

4.2 The simulation

We wish to show that there is a 2-tag system that simulates an arbitrary single step of a cyclic tag system computation, as defined in Equation (2). We decompose Equation (2) into three conceptual steps: (i) if $x_0 = 1$ then simulate the rule $x_0 \to \alpha_m$ by appending $\alpha_m = x_{l+1} \ldots x_{l+c}$, (ii) set x_1 to be the new read symbol and delete x_0, (iii) increment the program marker m so that the next appendant is $\alpha_{(m+1 \mod p)}$.

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

We begin by giving a simulation of (ii). Informally speaking, Lemma 2 states that there is a 2-tag system that (efficiently) moves the 'bar' forward by one symbol pair. The main difficulty is in distinguishing $\underset{1}{x}_1\underset{1}{\dot{x}}_1$ from the other unbarred symbol pairs $\underset{2}{x}_2\underset{2}{\dot{x}}_2,\ldots,\underset{l}{x}_l\underset{l}{\dot{x}}_l$.

Lemma 2 *Given a word of the form*

$$\hat{w} = \underset{1}{\bar{x}}_0\underset{1}{\dot{\bar{x}}}_0\ \underset{1}{x}_1\underset{1}{\dot{x}}_1\ldots\underset{1}{x}_i\underset{1}{\dot{x}}_i\ \underset{1}{a}\underset{1}{\dot{a}}\ldots\underset{1}{a}\underset{1}{\dot{a}}\ \underset{1}{x}_{i+1}\underset{1}{\dot{x}}_{i+1}\ldots\underset{1}{x}_l\underset{1}{\dot{x}}_l$$

where $\|\hat{w},a\dot{a}\| = 2^{\lceil\log(l+1)\rceil}$, *then there is a 2-tag system* T *that computes*

$$\underset{6}{\bar{x}}_1\underset{6}{\dot{\bar{x}}}_1\ldots\underset{6}{x}_i\underset{6}{\dot{x}}_i\ \underset{6}{a}\underset{6}{\dot{a}}\ldots\underset{6}{a}\underset{6}{\dot{a}}\ \underset{6}{x}_{i+1}\underset{6}{\dot{x}}_{i+1}\ldots\underset{6}{x}_l\underset{6}{\dot{x}}_l$$

in time $O(l\log l)$.

Proof idea There are 5 stages to the 2-tag algorithm. We let $\hat{w}_0 = \hat{w}$ and let $\hat{w}_k$ denote the output of the k^{th} iteration of the 5 stages.

In Stages 1 to 3 of iteration k we compute $\lceil\|\hat{w}_{k-1},x\dot{x}\|/2\rceil$, by marking every second pair of $x\dot{x}$ symbols (we mark the even numbered pairs). Then in Stages 4 and 5 we halve $\|\hat{w}_{k-1},a\dot{a}\|$ by marking every second pair of $a\dot{a}$ symbols (again we mark the even numbered pairs). We then return to Stage 1 and iterate until $\|\hat{w}_k,a\dot{a}\| = 1$: the counter has an odd value for the first time and we detect this. The number of fully completed iterations, and final value for k, is $k = \log\|\hat{w},a\dot{a}\|$. At this point $x_1\dot{x}_1$ is the only pair of unmarked $x\dot{x}$ symbols in $\hat{w}$, and so $x_1\dot{x}_1$ is isolated (unique) from the other symbol pairs in $\hat{w}$. We delete $\bar{x}_0\dot{\bar{x}}_0$. The uniqueness of $x_1\dot{x}_1$ enables the rule $x_1 \to \bar{x}_1\dot{\bar{x}}_1$ to be executed successfully.

Proof details As usual let $x \in \{0,1\}$. Here we specify 2-tag rules, and take the reader through a single (the first) iteration of these rules.

In Stage 1 we begin with a word of the form

$$\underset{1}{\bar{x}}_0\underset{1}{\dot{\bar{x}}}_0\ \underset{1}{x}_1\underset{1}{\dot{x}}_1\ \underset{1}{x}_2\underset{1}{\dot{x}}_2\ldots\underset{1}{x}_l\underset{1}{\dot{x}}_l\ \underset{1}{a}\underset{1}{\dot{a}}\ \underset{1}{a}\underset{1}{\dot{a}}\ldots\underset{1}{a}\underset{1}{\dot{a}}$$

Stage 1 consists of the rules:

$$\{\underset{1}{\bar{x}} \to \underset{2}{\bar{x}}\underset{2}{\dot{\bar{x}}}, \underset{1}{x} \to \underset{2}{x}\underset{2}{\dot{x}}\underset{2}{\ddot{x}}, \underset{1}{\not{x}} \to \underset{2}{\not{x}}\underset{2}{\dot{\not{x}}}, \underset{1}{a} \to \underset{2}{a}\underset{2}{\dot{a}}, \underset{1}{\not{a}} \to \underset{2}{\not{a}}\underset{2}{\dot{\not{a}}}\}$$

as well as a few more rules that are given below. After one round we have

$$\underset{2}{\bar{x}}_0\underset{2}{\dot{\bar{x}}}_0\ \underset{2}{x}_1\underset{2}{\dot{x}}_1\underset{2}{\ddot{x}}_1\ \underset{2}{x}_2\underset{2}{\dot{x}}_2\underset{2}{\ddot{x}}_2\ldots\underset{2}{x}_l\underset{2}{\dot{x}}_l\underset{2}{\ddot{x}}_l\ \underset{2}{a}\underset{2}{\dot{a}}\ \underset{2}{a}\underset{2}{\dot{a}}\ldots\underset{2}{a}\underset{2}{\dot{a}} \quad (5)$$

The Stage 2 rules are

$$\{\underset{2}{\bar{x}} \to \underset{3}{\bar{x}}\underset{3}{\dot{\bar{x}}}, \underset{2}{x} \to \underset{3}{x}\underset{3}{\dot{x}}, \underset{2}{\dot{x}} \to \underset{3}{\not{x}}\underset{3}{\dot{\not{x}}}, \underset{2}{\ddot{x}} \to \epsilon, \underset{2}{\not{x}} \to \underset{3}{\not{x}}\underset{3}{\dot{\not{x}}}, \underset{2}{\dot{\not{x}}} \to \underset{3}{\not{x}}\underset{3}{\dot{\not{x}}},$$
$$\underset{2}{a} \to \underset{3}{a}\underset{3}{\dot{a}}, \underset{2}{\dot{a}} \to \underset{3}{a}\underset{3}{\dot{a}}, \underset{2}{\not{a}} \to \underset{3}{\not{a}}\underset{3}{\dot{\not{a}}}, \underset{2}{\dot{\not{a}}} \to \underset{3}{\not{a}}\underset{3}{\dot{\not{a}}}\}$$

Continuing from (5), after one round we see that every second (even numbered) pair of $x\dot{x}$ is marked

$$\underset{3}{\bar{x}}_0\underset{3}{\dot{\bar{x}}}_0\ \underset{3}{x}_1\underset{3}{\dot{x}}_1\ \underset{3}{\not{x}}_2\underset{3}{\dot{\not{x}}}_2\ \underset{3}{x}_3\underset{3}{\dot{x}}_3\ \underset{3}{\not{x}}_4\underset{3}{\dot{\not{x}}}_4\ldots\underset{3}{x}_{l-1}\underset{3}{\dot{x}}_{l-1}\ \underset{3}{\not{x}}_l\underset{3}{\dot{\not{x}}}_l\ \underset{3}{a}\underset{3}{\dot{a}}\ \underset{3}{a}\underset{3}{\dot{a}}\ldots\underset{3}{a}\underset{3}{\dot{a}}$$

where (for illustration purposes only) we assume that l is even.

The Stage 3 rules are:

$$\{\underset{3}{\bar{x}} \to \underset{4}{\bar{x}}\underset{4}{\dot{\bar{x}}}, \underset{3}{x} \to \underset{4}{x}\underset{4}{\dot{x}}, \underset{3}{\not{x}} \to \underset{4}{\not{x}}\underset{4}{\dot{\not{x}}}, \underset{3}{a} \to \underset{4}{a}\underset{4}{\dot{a}}, \underset{3}{\not{a}} \to \underset{4}{\not{a}}\underset{4}{\dot{\not{a}}},$$
$$\underset{3}{\dot{\bar{x}}} \to \#\underset{4}{\bar{x}}\underset{4}{\dot{\bar{x}}}, \underset{3}{\dot{x}} \to \underset{4}{x}\underset{4}{\dot{x}}, \underset{3}{\dot{\not{x}}} \to \underset{4}{\not{x}}\underset{4}{\dot{\not{x}}}, \underset{3}{\dot{a}} \to \underset{4}{a}\underset{4}{\dot{a}}, \underset{3}{\dot{\not{a}}} \to \underset{4}{\not{a}}\underset{4}{\dot{\not{a}}}\}$$

We enter Stage 3 by reading either a dotted symbol (there was an odd number of unmarked $x\dot{x}$ pairs in Stage 1) or undotted symbol (there was an even number of unmarked $x\dot{x}$ pairs in Stage 1). Stage 3 begins by checking this (in one step); if the parity is even, that is the 2-tag system is reading dotted symbols, then a # symbol is appended to restore the parity to odd after one round. On completion of Stage 3 we are reading an undotted symbol:

$$\underset{4}{\bar{x}}_0\underset{4}{\dot{\bar{x}}}_0\ \underset{4}{x}_1\underset{4}{\dot{x}}_1\ \underset{4}{\not{x}}_2\underset{4}{\dot{\not{x}}}_2\ \underset{4}{x}_3\underset{4}{\dot{x}}_3\ \underset{4}{\not{x}}_4\underset{4}{\dot{\not{x}}}_4\ldots\underset{4}{x}_{l-1}\underset{4}{\dot{x}}_{l-1}\ \underset{4}{\not{x}}_l\underset{4}{\dot{\not{x}}}_l\ \underset{4}{a}\underset{4}{\dot{a}}\ \underset{4}{a}\underset{4}{\dot{a}}\ldots\underset{4}{a}\underset{4}{\dot{a}}$$

In Stages 4 and 5 of iteration k we halve the value of the counter (we compute $\|\hat{w}_k,a\dot{a}\| = \|\hat{w}_{k-1},a\dot{a}\|/2$), in a similar fashion to Stages 1 to 3. The Stage 4 rules are

$$\{\underset{4}{\bar{x}} \to \underset{5}{\bar{x}}\underset{5}{\dot{\bar{x}}}, \underset{4}{x} \to \underset{5}{x}\underset{5}{\dot{x}}, \underset{4}{\not{x}} \to \underset{5}{\not{x}}\underset{5}{\dot{\not{x}}}, \underset{4}{a} \to \underset{5}{a}\underset{5}{\dot{a}}\underset{5}{\ddot{a}}, \underset{4}{\not{a}} \to \underset{5}{\not{a}}\underset{5}{\dot{\not{a}}}\}$$

Which after one round gives

$$\underset{5}{\bar{x}}_0\underset{5}{\dot{\bar{x}}}_0\ \underset{5}{x}_1\underset{5}{\dot{x}}_1\ \underset{5}{\not{x}}_2\underset{5}{\dot{\not{x}}}_2\ldots\underset{5}{\not{x}}_l\underset{5}{\dot{\not{x}}}_l\ \underset{5}{a}\underset{5}{\dot{a}}\underset{5}{\ddot{a}}\ \underset{5}{a}\underset{5}{\dot{a}}\underset{5}{\ddot{a}}\ldots\underset{5}{a}\underset{5}{\dot{a}}\underset{5}{\ddot{a}}$$

The Stage 5 rules then halve the counter value:

$$\{\underset{5}{\bar{x}} \to \underset{1}{\bar{x}}\underset{1}{\dot{\bar{x}}}, \underset{5}{x} \to \underset{1}{x}\underset{1}{\dot{x}}, \underset{5}{\not{x}} \to \underset{1}{\not{x}}\underset{1}{\dot{\not{x}}}, \underset{5}{\dot{x}} \to \underset{1}{x}\underset{1}{\dot{x}}, \underset{5}{\dot{\not{x}}} \to \underset{1}{\not{x}}\underset{1}{\dot{\not{x}}},$$
$$\underset{5}{a} \to \underset{1}{a}\underset{1}{\dot{a}}, \underset{5}{\dot{a}} \to \underset{1}{\not{a}}\underset{1}{\dot{\not{a}}}, \underset{5}{\ddot{a}} \to \epsilon, \underset{5}{\not{a}} \to \underset{1}{\not{a}}\underset{1}{\dot{\not{a}}}, \underset{5}{\dot{\not{a}}} \to \underset{1}{\not{a}}\underset{1}{\dot{\not{a}}}\}$$

Continuing our computation we get:

$$\underset{1}{\bar{x}}_0\underset{1}{\dot{\bar{x}}}_0\ \underset{1}{x}_1\underset{1}{\dot{x}}_1\ \underset{1}{\not{x}}_2\underset{1}{\dot{\not{x}}}_2\ldots\underset{1}{\not{x}}_l\underset{1}{\dot{\not{x}}}_l\ \underset{1}{a}\underset{1}{\dot{a}}\ \underset{1}{\not{a}}\underset{1}{\dot{\not{a}}}\ \underset{1}{a}\underset{1}{\dot{a}}\ \underset{1}{\not{a}}\underset{1}{\dot{\not{a}}}\ldots\underset{1}{a}\underset{1}{\dot{a}}\ \underset{1}{\not{a}}\underset{1}{\dot{\not{a}}}$$

which switches control back to Stage 1.

Each iteration of the 5 stages halves the counter value. After $\log\|\hat{w},a\dot{a}\|$ iterations the counter has value 1, this causes the output from Stage 4 to be of odd length for the first time. This in turn switches parity to even (dotted symbols) during Stage 5, which is detected at the beginning of Stage 1, by the rules:

$$\{\underset{1}{\dot{\bar{x}}} \to \#, \underset{1}{\dot{x}} \to \underset{6}{\bar{x}}\underset{6}{\dot{\bar{x}}}, \underset{1}{\dot{\not{x}}} \to \underset{6}{x}\underset{6}{\dot{x}}, \underset{1}{\dot{a}} \to \underset{6}{a}\underset{6}{\dot{a}}, \underset{1}{\dot{\not{a}}} \to \underset{6}{a}\underset{6}{\dot{a}}\}$$

The first of these rules deletes x_0 in one step. The second of these rules passes the bar forward by one symbol pair, while 2 of the others unmark the remaining symbols.

$$\underset{1'}{x}_0\underset{1'}{\dot{x}}_0\ \underset{6}{\bar{x}}_1\underset{6}{\dot{\bar{x}}}_1\ \underset{6}{x}_2\underset{6}{\dot{x}}_2\ \underset{6}{x}_3\underset{6}{\dot{x}}_3\ldots\underset{6}{x}_l\underset{6}{\dot{x}}_l\ \underset{6}{a}\underset{6}{\dot{a}}\ \underset{6}{a}\underset{6}{\dot{a}}\ldots\underset{6}{a}\underset{6}{\dot{a}}$$

The # symbol restores the parity to odd so that we read undotted symbols (in subsequent computations in this paper).

If the data word $\hat{w}$ is in the more general form given by Equation (3) then the same proof holds; our rules are such that embedding the counter does not affect parity in a way that would change the algorithm control flow. ∎

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

The following lemma provides much of the mechanics required for simulation of the appending of a cyclic tag system appendant (point (i) from the introductory paragraph of Section 4.2). Simulating the appending is straightforward, the main work is in maintaining the equality in Equation (4).

Lemma 3 *Given a word of the form*

$$\hat{w} = \underset{0}{\bar{1}}\underset{0}{\dot{\bar{1}}}\ \underset{0}{x}_1\underset{0}{\dot{x}}_1 \ldots \underset{0}{x}_i\underset{0}{\dot{x}}_i\ \underset{0}{a}\underset{0}{\dot{a}} \ldots \underset{0}{a}\underset{0}{\dot{a}}\ \underset{0}{x}_{i+1}\underset{0}{\dot{x}}_{i+1} \ldots \underset{0}{x}_l\underset{0}{\dot{x}}_l$$

where $\|\hat{w}, a\dot{a}\| = 2^{\lceil \log(l+1) \rceil}$, *then there is a 2-tag system* T *that computes*

$$\hat{\hat{w}} = \underset{8}{\bar{1}}\underset{8}{\dot{\bar{1}}}\ \underset{8}{x}_1\underset{8}{\dot{x}}_1 \ldots \underset{8}{x}_i\underset{8}{\dot{x}}_i\ \underset{8}{a}\underset{8}{\dot{a}} \ldots \underset{8}{a}\underset{8}{\dot{a}}\ \underset{8}{x}_{i+1}\underset{8}{\dot{x}}_{i+1} \ldots \underset{8}{x}_l\underset{8}{\dot{x}}_l \\ \circ\ \underset{8}{x}_{l+1}\underset{8}{\dot{x}}_{l+1} \ldots \underset{8}{x}_{l+c}\underset{8}{\dot{x}}_{l+c}$$

where $c \leqslant 2^{\lceil \log(l+1) \rceil}$, $\|\hat{\hat{w}}, a\dot{a}\| = 2^{\lceil \log(l+c+1) \rceil}$, *and* $\circ$ *denotes concatenation.* T *completes this computation in time* $O(l \log l)$.

Proof By applying the rule

$$\{\underset{0}{\bar{1}} \rightarrow \underset{0}{x}_{l+1}\underset{0}{\dot{x}}_{l+1} \ldots \underset{0}{x}_{l+c}\underset{0}{\dot{x}}_{l+c}\ \underset{1}{\bar{1}}\underset{1}{\dot{\bar{1}}}\}$$

to $\hat{w}$ we get a word denoted $\hat{w}_0$ that is of a similar form to $\hat{\hat{w}}$ except that $\|\hat{w}_0, a\dot{a}\| = 2^{\lceil \log(l+1) \rceil}$, i.e. the counter has not yet been updated to the correct value of $\|\hat{\hat{w}}, a\dot{a}\| = 2^{\lceil \log(l+c+1) \rceil}$. The remainder of the proof is concerned with updating the counter.

We let $\hat{w}_k$ denote the output of the k^{th} iteration of the 4 stages. The rules for Stages 1 to 4 are of a similar flavour to those used in the proof of Lemma 2, so we omit them in favour of a brief overview.

We begin by computing on $\hat{w}_0$. During Stages 1 and 2 of iteration k, we compute $\|\hat{w}_k, a\dot{a}\| = \|\hat{w}_{k-1}, a\dot{a}\|/2$, by marking every second pair of $a\dot{a}$ symbols (here we mark the even numbered pairs). Then in Stages 3 and 4 we compute

$$\|\hat{w}_k, x\dot{x}\| = \left\lfloor \frac{\|\hat{w}_{k-1}, x\dot{x}\|}{2} \right\rfloor \tag{6}$$

by marking every second pair of $x\dot{x}$ symbols (here we mark the odd numbered pairs). We then return to Stage 1 and iterate until $\|\hat{w}_k, a\dot{a}\| = 1$: the counter now has an odd value (for the first time) and we detect this in Stage 3. The number of fully completed iterations, and final value for k, is $k = \log \|\hat{w}, a\dot{a}\| = \lceil \log(l+1) \rceil$. An additional stage restores the parity (by introducing an extra # symbol and then deleting it after one round) so that we are reading undotted symbols. Then $\hat{w}_k$ is of the form

$$\hat{w}_k = \underset{5}{\bar{1}}\underset{5}{\dot{\bar{1}}}\ \underset{5}{\not{x}}_1\underset{5}{\not{\dot{x}}}_1 \ldots \underset{5}{\not{x}}_i\underset{5}{\not{\dot{x}}}_i\ \underset{5}{a}\underset{5}{\dot{a}}\ \underset{5}{\not{a}}\underset{5}{\not{\dot{a}}} \ldots \underset{5}{\not{a}}\underset{5}{\not{\dot{a}}}\ \underset{5}{\not{x}}_{i+1}\underset{5}{\not{\dot{x}}}_{i+1} \ldots \underset{5}{\not{x}}_l\underset{5}{\not{\dot{x}}}_l \\ \circ\ \underset{5}{\not{x}}_{l+1}\underset{5}{\not{\dot{x}}}_{l+1} \ldots \underset{5}{\not{x}}_{l+c}\underset{5}{\not{\dot{x}}}_{l+c}$$

At this point the number $\|\hat{w}_k, x\dot{x}\|$ of unmarked $x\dot{x}$ pairs satisfies $0 \leqslant \|\hat{w}_k, x\dot{x}\| \leqslant 1$. To see this, note that $\|\hat{w}_0, a\dot{a}\| = 2^{\lceil \log(l+1) \rceil}$ and $c \leqslant 2^{\lceil \log(l+1) \rceil}$. Solving Equation (6) for $k = \lceil \log(l+1) \rceil$ gives 0 iff

$$0 \leqslant \|\hat{w}_0, x\dot{x}\| = l + c < 2^{\lceil \log(l+1) \rceil}$$

and 1 iff

$$2^{\lceil \log(l+1) \rceil} \leqslant \|\hat{w}_0, x\dot{x}\| = l + c < 2^{\lceil \log(l+c+1) \rceil}$$

There are no other possible values for $\|\hat{w}_0, x\dot{x}\|$ thus we have only to check whether $\|\hat{w}_k, x\dot{x}\|$ is 0 or 1. To prepare, in two consecutive rounds we apply the rules

$$\{\underset{5}{\bar{1}} \rightarrow \underset{6}{\bar{1}}\underset{6}{\dot{\bar{1}}}, \underset{5}{x} \rightarrow \underset{6}{x}\underset{6}{\dot{x}}\underset{6}{\ddot{x}}, \underset{5}{\not{x}} \rightarrow \underset{6}{\not{x}}\underset{6}{\not{\dot{x}}}, \underset{5}{a} \rightarrow \underset{6}{a}\underset{6}{\dot{a}}, \underset{5}{\not{a}} \rightarrow \underset{6}{a}\underset{6}{\dot{a}}\}$$

and

$$\{\underset{6}{\bar{1}} \rightarrow \underset{7}{\bar{1}}\underset{7}{\dot{\bar{1}}}, \underset{6}{x} \rightarrow \underset{7}{x}\underset{7}{\dot{x}}, \underset{6}{\ddot{x}} \rightarrow \epsilon, \underset{6}{\not{x}} \rightarrow \underset{7}{x}\underset{7}{\dot{x}}, \underset{6}{\not{\dot{x}}} \rightarrow \underset{7}{x}\underset{7}{\dot{x}}, \\ \underset{6}{a} \rightarrow \underset{7}{a}\underset{7}{\dot{a}}, \underset{6}{\dot{a}} \rightarrow \underset{7}{a}\underset{7}{\dot{a}}, \underset{6}{\not{a}} \rightarrow \underset{7}{a}\underset{7}{\dot{a}}, \underset{6}{\not{\dot{a}}} \rightarrow \underset{7}{a}\underset{7}{\dot{a}}\}$$

These rules have the effect of shifting the parity of the read head to dotted symbols iff $\|\hat{w}_k, x\dot{x}\| = 1$. In addition these rules unmark all marked symbols, as the marks are not needed below.

We have two cases:
Case 1: If $\|\hat{w}_k, x\dot{x}\| = 0$ we do not need to change the counter value in order to satisfy Equation (4). In this case we detect $\|\hat{w}_k, x\dot{x}\| = 0$ by reading the undotted, barred symbol $\underset{7}{\bar{1}}$. To complete the computation we apply the rules

$$\{\underset{7}{\bar{1}} \rightarrow \underset{8}{\bar{1}}\underset{8}{\dot{\bar{1}}}, \underset{7}{x} \rightarrow \underset{8}{x}\underset{8}{\dot{x}}, \underset{7}{a} \rightarrow \underset{8}{a}\underset{8}{\dot{a}}\}$$

Case 2: If $\|\hat{w}_k, x\dot{x}\| = 1$ we double the counter to satisfy Equation (4). In this case we detect $\|\hat{w}_k, x\dot{x}\| = 1$ by reading the dotted, barred symbol $\underset{7}{\dot{\bar{1}}}$. We then restore odd parity and double the counter value, by applying the rules

$$\{\underset{7}{\dot{\bar{1}}} \rightarrow \#\underset{8}{\bar{1}}\underset{8}{\dot{\bar{1}}}, \underset{7}{\dot{x}} \rightarrow \underset{8}{x}\underset{8}{\dot{x}}, \underset{7}{\dot{a}} \rightarrow \underset{8}{a}\underset{8}{\dot{a}}\ \underset{8}{a}\underset{8}{\dot{a}}\}$$

$\blacksquare$

4.3 Proof of main result

Theorem 3 *Given a cyclic tag system* C *that computes in time* $t(n)$ *on input of length* n*, where* n *is at least the length of* C*'s longest appendant, then there is a 2-tag system* T_C *that simulates the computation of* C *and computes in time* $O(t^2(n) \log t(n))$.

Proof Recall, from the beginning of Section 4.2, the decomposition of Equation (2) into the conceptual steps (i), (ii) and (iii).

Lemma 3 provides the algorithm for step (i) for the case that $\bar{x}_0 = \bar{1}$. For the other case of $\bar{x}_0 = \bar{0}$ we skip (i).

Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

Deciding between the two cases is easily implemented by setting the parity to even iff $\bar{x}_0 = \bar{1}$. Also, for Lemma 3 we insist that n is at least the length of C's longest appendant (the constant number of shorter inputs would be padded to this length).

Lemma 2 provides the algorithm for step (ii).

For step (iii) we introduce a new decoration for 2-tag symbols. So far, the number q of distinct 2-tag symbols that we have used is dependent on our algorithm. We now increase this number to pq where, as usual, p is the number of appendants of C. We create a new symbol set by decorating each 2-tag symbol $y \in \{\bar{x}, \dot{\bar{x}}, x, \dot{x}, \ddot{x}, \cancel{x}, \dot{\cancel{x}}, a, \dot{a}, \ddot{a}, \cancel{a}, \dot{\cancel{a}}, \#\}$ with an integer m for all $0 \leqslant m < p$. Using this, our encoding of an arbitrary cyclic tag system configuration is of the form

$$\underset{m}{\underset{j}{\bar{x}}}_0\underset{m}{\underset{j}{\dot{\bar{x}}}}_0\ \underset{m}{\underset{j}{x}}_1\underset{m}{\underset{j}{\dot{x}}}_1 \ldots \underset{m}{\underset{j}{x}}_i\underset{m}{\underset{j}{\dot{x}}}_i\ \underset{m}{\underset{j}{a}}\underset{m}{\underset{j}{\dot{a}}} \ldots \underset{m}{\underset{j}{a}}\underset{m}{\underset{j}{\dot{a}}}\ \underset{m}{\underset{j}{x}}_{i+1}\underset{m}{\underset{j}{\dot{x}}}_{i+1} \ldots \underset{m}{\underset{j}{x}}_l\underset{m}{\underset{j}{\dot{x}}}_l$$

Steps (i) and (ii) are simulated, while ignoring the value of m. (Note that our 2-tag algorithms are easily concatenated by having appropriate j values at the beginning and end of each algorithm.) Then j is given a value that signals the completion of steps (i) and (ii). Then step (iii) (incrementing the program marker) is simulated by rules of the form

$$\{\underset{m}{\underset{j}{\bar{x}}} \rightarrow \underset{m'}{\underset{j}{\bar{x}}}\,\underset{m'}{\underset{j}{\dot{\bar{x}}}},\ \underset{m}{\underset{j}{x}} \rightarrow \underset{m'}{\underset{j}{x}}\,\underset{m'}{\underset{j}{\dot{x}}},\ \underset{m}{\underset{j}{a}} \rightarrow \underset{m'}{\underset{j}{a}}\,\underset{m'}{\underset{j}{\dot{a}}}\}$$

where $m' = (m+1) \mod p$. Applying these rules for a given k takes only one round, or $O(l)$ timesteps.

In the time analysis of the computation of T_C note that for an arbitrary timestep of C we have $l = O(t(n))$. Therefore via Lemmas 3 and 2, and the present proof, T_C simulates a *single* step of C's computation in time $O(t(n)\log t(n))$. ∎

We get the proof of Theorem 1, our main result, by combining the statements of Theorems 2 and 3. If we combine these Theorems directly we get a time bound that is higher than that of Theorem 1. To get our tighter bound a more careful analysis is required where we substitute space bounds for time bounds whenever possible in the analysis.

References

[1] C. Baiocchi. Three small universal Turing machines. In *Machines, Computations, and Universality (MCU)*, volume 2055 of *LNCS*, pages 1–10, Chişinău, Moldova, May 2001. Springer.

[2] J. Cocke and M. Minsky. Universality of tag systems with $P = 2$. *Journal of the ACM*, 11(1):15–20, Jan. 1964.

[3] M. Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[4] S. Cook. The solvability of the derivability problem for one-normal systems. *Journal of the ACM*, 13(2):223–225, Apr. 1966.

[5] G. T. Hermann. The uniform halting problem for generalized one state Turing machines. In *Proceedings of the ninth annual Symposium on Switching and Automata Theory (FOCS)*, pages 368–372, Schenectady, New York, Oct. 1968. IEEE Computer Society Press.

[6] N. Ikeno. A 6-symbol 10-state universal Turing machine. In *Proceedings, Institute of Electrical Communications*, Tokyo, 1958.

[7] M. Kudlek. Small deterministic Turing machines. *Theoretical Computer Science*, 168(2):241–255, 1996.

[8] M. Kudlek and Y. Rogozhin. A universal Turing machine with 3 states and 9 symbols. In *Developments in Language Theory (DLT)*, volume 2295 of *LNCS*, pages 311–318, Vienna, May 2001. Springer.

[9] L. A. Levin and R. Venkatesan. An average case NP-complete graph problem. `arXiv:cs.CC/0112001`, v2, 2002.

[10] K. Lindgren and M. G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.

[11] M. Margenstern. Non-erasing Turing machines: A frontier between a decidable halting problem and universality. In *Fundamentals of Computation Theory (FCT)*, volume 710 of *LNCS*, pages 375–385. Springer, 1993.

[12] M. Margenstern. Non-erasing Turing machines: a new frontier between a decidable halting problem and universality. In *LATIN*, volume 911 of *LNCS*, pages 386–397. Springer, Apr. 1995.

[13] M. Margenstern. Results on the halting problem. In *LMPS'95*, Florence, Aug. 1995.

[14] M. Margenstern. Frontier between decidability and undecidability: a survey. *Theoretical Computer Science*, 231(2):217–251, Jan. 2000.

[15] M. Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, volume 5, pages 229–238, Provelence, 1962. AMS.

[16] T. Neary and D. Woods. P-completeness of cellular automaton Rule 110. In *International Colloquium on Automata Languages and Programming (ICALP)*, volume 4051 of *LNCS*, pages 132–143. Springer, July 2006.

[17] T. Neary and D. Woods. Small fast universal Turing machines. *Theoretical Computer Science*, 2006. Accepted.

[18] L. Pavlotskaya. Solvability of the halting problem for certain classes of Turing machines. *Mathematical Notes (Springer)*, 13(6):537–541, June 1973.

[19] L. Pavlotskaya. Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. *Problemi Kiberneti*, 33:91–118, 1978. (Sufficient conditions for the halting problem decidability of Turing machines) (in Russian).

[20] E. Post. Formal reductions of the combinatorial decision problem. *Amer. Journal of Math.*, 65:197–215, 1943.

[21] R. M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.

[22] R. M. Robinson. Minsky's small universal Turing machine. *International Journal of Mathematics*, 2(5):551–562, 1991.

[23] Y. Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

[24] C. E. Shannon. A universal Turing machine with two internal states. *Automata Studies, Annals of Mathematics Studies*, 34:157–165, 1956.

[25] R. Venkatesan and L. Levin. Random instances of a graph coloring problem are hard. In *Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC)*, pages 217–222, Chicago, Illinois, 1988. ACM.

[26] H. Wang. Tag systems and lag systems. *Math. Annalen*, 152:65–74, 1963.

[27] S. Watanabe. 5-symbol 8-state and 5-symbol 6-state universal Turing machines. *Journal of the ACM*, 8(4):476–483, Oct. 1961.

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE

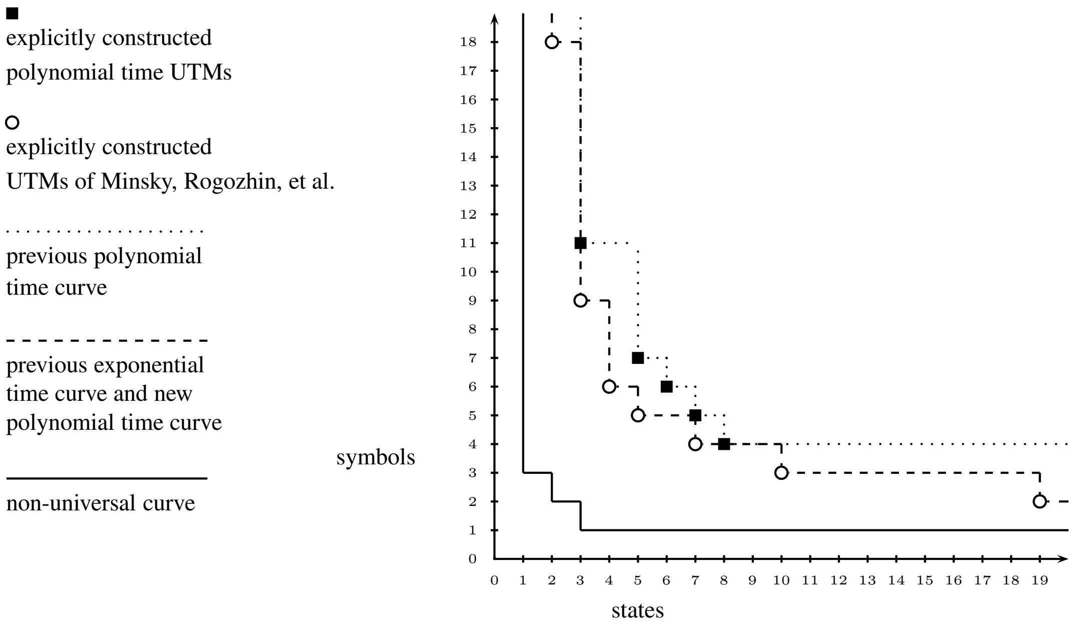

Figure 1. State-symbol plot of small UTMs. The plot shows the polynomial time curve induced by our previous UTMs [17], the exponential time curve of Minsky, Rogozhin and others [1, 8, 15, 23], and the non-universal Turing machine curve for which there are no UTMs [5, 7, 18, 19]. The present paper improves the polynomial time curve so that it coincides with the previous exponential time curve. Our result shows that a polynomial time UTM exists for each state-symbol pair that is on, above, and to the right of the new polynomial time curve.

Proceedings of the 47th Annual IEEE Symposium
on Foundations of Computer Science (FOCS'06)
0-7695-2720-5/06 $20.00 © 2006 IEEE