Small universal Turing machines

Turlough Neary

NUI MAYNOOTH
Ollscoil na hÉireann Má Nuad

A thesis submitted for the
degree of Doctor of Philosophy

Department of Computer Science
National University of Ireland, Maynooth

Supervisors: Dr. Damien Woods and Dr. J. Paul Gibson
External Examiner: Prof. Maurice Margenstern
Internal Examiner: Dr. James Power
Department Head: Dr. Adam Winstanley

October 2008

Acknowledgements

My supervisor Damien Woods deserves a special thank you. His help and guidance went far beyond the role of supervisor. He was always enthusiastic, and generous with his time. This work would not have happened without him. I would also like to thank my supervisor Paul Gibson for his advice and support.

Thanks to the staff and postgraduates in the computer science department at NUI Maynooth for their support and friendship over the last few years. In particular, I would like to mention Niall Murphy he has always been ready to help whenever he could and would often lighten the mood in dark times with some rousing Gilbert and Sullivan.

I thank the following people for their interesting discussions and/or advice: Maurice Margenstern, Yurii Rogozhin, Manfred Kudlek, Matthew Cook, Liesbeth De Mol, Fred Lunnon, Ronan Reilly, James Power, and Tom Naughton. I would also like to thank Tony Seda for his support and allowing me the time to complete my work.

I would like to thank my parents Ann and Donal, my brother Fiachra, and my sisters Sarah and Rebecca for their help and support over the years. Finally, I would like to dedicate this work to the three most important people in my life Astrid, Lelah, and Benjamin.

Abstract

Numerous results for simple computationally universal systems are presented, with a particular focus on small universal Turing machines. These results are towards finding the simplest universal systems. We add a new aspect to this area by examining trade-offs between the simplicity of universal systems and their time/space computational complexity.

Improving on the earliest results we give the smallest known universal Turing machines that simulate Turing machines in $O(t^2)$ time. They are also the smallest known machines where direct simulation of Turing machines is the technique used to establish their universality. This result gives a new algorithm for small universal Turing machines.

We show that the problem of predicting t steps of the 1D cellular automaton Rule 110 is P-complete. As a corollary we find that the small weakly universal Turing machines of Cook and others run in polynomial time, an exponential improvement on their previously known simulation time overhead. These results are achieved by improving the cyclic tag system simulation time of Turing machines from exponential to polynomial.

A new form of tag system which we call a bi-tag system is introduced. We prove that bi-tag systems are universal by showing they efficiently simulate Turing machines. We also show that 2-tag systems efficiently simulate Turing machines in polynomial time. 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 on a forty-year old result.

We present new small polynomial time universal Turing machines with state-symbol pairs of $(5, 5)$, $(6, 4)$, $(9, 3)$ and $(15, 2)$. These machines simulate bi-tag systems and are the smallest known universal Turing machines with 5, 4, 3 and 2-symbols, respectively. The 5-symbol machine uses the same number of instructions (22) as the current smallest known universal Turing machine (Rogozhin's 6-symbol machine).

We give the smallest known weakly universal Turing machines. These machines have state-symbol pairs of $(6, 2)$, $(3, 3)$ and $(2, 4)$. The 3-state and 2-state machines are very close to the minimum possible size for weakly universal machines with 3 and 2 states, respectively.

Contents

List of Figures

1

Introduction

Since the advent of the Church-Turing thesis there has been much work to simplify computationally universal systems. Now, seventy years on, the size of the simplest universal systems is quite amazing. In this thesis we examine some of the most intuitively simple universal models of computation including Turing machines [Tur37, Min67, HU79], tag systems [Pos43, Min67] and cellular automata [vN66]. We improve the state of the art in many of these simple models by giving even simpler models. Furthermore, moving in a new direction we examine possible trade-offs between the simplicity of a model and its time/space resource efficiency.

The problem of finding simple universal systems is in itself an interesting one and also has a number of applications. Perhaps the most obvious is in finding boundaries between universality and non-universality. Another application is that giving increasingly simple computationally universal systems in many cases simplifies the problem of emulating universal systems. This simplifies proving universality results for other computational systems. It also simplifies the problem of proving various questions about the behaviour of a dynamical system are undecidable. For example a dynamical system emulates a universal system then that dynamical system must have some undecidable properties.

Giving universal systems that are time efficient[1] also has important applications. For instance simulating an existing efficient universal system in polynomial time is one way to prove that another computational model is efficient. Another application would be determining whether a computational model may be predicted exponentially faster (in parallel) than explicit step-by-step simulation. If a computational model simulates Turing machines in polynomial time then such exponentially fast prediction is not possible unless $\mathsf{P} = \mathsf{NC}$.

1.1 History of simple universal models

In Turing's paper [Tur37] he defines the machine model that is now known as the Turing machine. It has become widely accepted that the Turing machine model captures the notion of algorithm. In his paper, Turing also gives an instance of his model, a universal Turing machine, that simulates the behaviour of any Turing machine when given a description (suitable encoding) of the machine and its input. This reduces the problem of simulating all Turing machines to the problem of simulating any universal Turing machine.

Independently of Turing, Emil Post [Pos36] gave a machine similar to that of Turing. The basic operations employed by these two types of machines are essentially the same. In [Pos36], Post gives no details of how his machines would solve specific problems or encode them as input to his machines. Despite this, Post hypothesised that his machine would be proved equivalent to Church's λ-calculus and thus, by inference, Turing machines. Unlike Turing's machines, Post's machines[2] used only 2 symbols and so, in the wake of the Church-Turing thesis, Post's hypothesis could be construed as an early conjecture that 2-symbol Turing machines are universal.

Years later, Moore [Moo52] noted that 2-symbol machines were universal as any Turing machine could be converted into a 2-symbol machine by encoding the symbols in binary. In the same paper Moore used this observation to give a universal 3-tape machine with 2 symbols and 15 states. Moore's machine uses only 57 instructions, each instruction being a sextuple that either moves one of its tape heads or prints a single symbol to one of its tapes. This result has been largely ignored in the literature despite being the first small universal Turing machine.

In the seminal paper on small universal Turing machines, it was proved by Shannon [Sha56] that both 2-state and 2-symbol universal Turing machines existed. Shannon's paper ends with the sentence: "An interesting unsolved problem is to find the minimum state-symbol product for a universal Turing machine." This sparked a vigorous competition between Minsky and Watanabe to see who could come up with the smallest universal Turing machine [Min60a, Wat60, Wat61, Min62a, Min67, Wat72]. The game was now afoot!

[1] Here we say a system is time efficient if it simulates Turing machines in polynomial time.

[2] In later papers when authors [Fis65, AF67] refer to Post machines they mean a (Turing) machine whose instructions are defined by quadruples instead of quintuples and any finite tape alphabet. This follows from a later paper by Post [Pos47] where he adopts quadruples in his "formulation of a Turing machine." Davis [Dav58] also adopts this quadruples formalism but does not refer to such machines as Post machines.

states	symbols	state-symbol product	author
m	2	2m	Shannon [Sha56]
2	n	2n	Shannon [Sha56]
12	6	72	Takahashi [Tak58] (mentioned in [Wat61])
10	6	60	Ikeno [Ike58] (also appears in [Min60a])
17	3	51	Watanabe [Wat60] (mentioned in [Wat61])
8	6	48	Watanabe [Wat60] (mentioned in [Min62a])
25	2	50	Minsky [Min62b]
7	6	42	Minsky [Min60a]
8	5	40	Watanabe [Wat61]
9	4	36	Alan Tritter (mentioned in [Min62a])
6	5	30	Watanabe [Wat61]†
25	2	50	Minsky [Min62b]
6	6	36	Minsky [Min62a]
7	4	28	Minsky [Min62a, Min62b]
9	3	27	Goto (mentioned in [Wat72])
7	3	21	Watanabe (mentioned in [Wat72, Noz69])†
5	4	20	Watanabe [Wat72]†

Table 1.1.1: Table of small semi-weakly and standard universal Turing machines up until 1972. Semi-weakly universal machines are denoted by †.

1.1.1 Standard universal Turing machines

The Turing machine model that we choose as standard has a single one-dimensional tape, one tape head, and is deterministic [HU79]. Some of the earliest small standard universal machines are given in Table 1.1.1. One particular machine worthy of note is the 7-state 4-symbol universal Turing machine of Minsky [Min62a, Min67]. Minsky's machine simulates Turing machines via 2-tag systems, which were proved universal by Cocke and Minsky [CM64]. The technique of simulating 2-tag systems, pioneered by Minsky, was extended by Rogozhin [Rog79, Rog82] to give the (then) smallest known universal Turing machines for a number of state-symbol pairs. These 2-tag simulators were subsequently reduced in size by Rogozhin [Rog92, Rog93, Rog96, Rog98], Kudlek and Rogozhin [KR02], and Baiocchi [Bai01]. The smallest 2-tag simulators are plotted as hollow circles in Figure 1.1.1. These machines induce a universal curve.

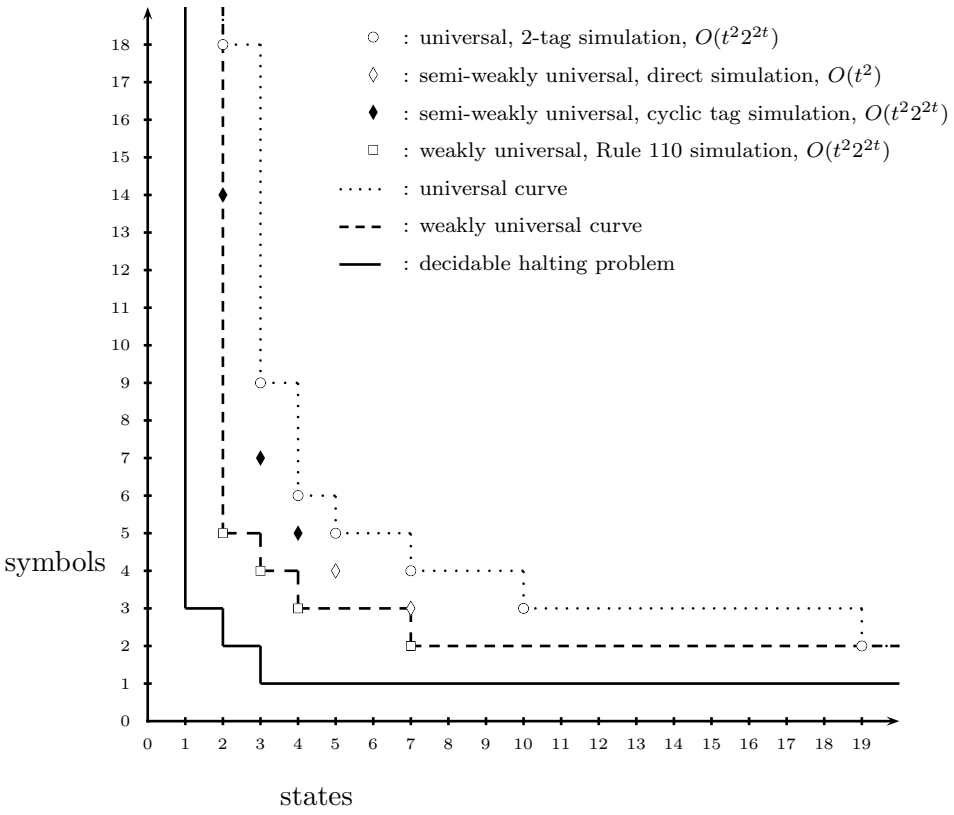


Figure 1.1.1: State-symbol plot of small universal Turing machines, excluding the work presented in this thesis. The simulation technique is given for each group of machines. Also, we give the simulation time overheads in terms of simulating any single tape, deterministic Turing machine that runs in time t.

1.1.2 Weakly and semi-weakly universal Turing machines

Over the years, small universal machines were given for a number of variants on the standard Turing machine model. By generalising the model we often find smaller universal programs. One variation on the standard Turing machine is to allow an infinitely repeated word to one side of its input, and on the other side a (standard) infinitely repeated blank symbol. We call such a machine semi-weak. In 1961 Watanabe [Wat61] gave a semi-weakly universal Turing machine with 6 states and 5 symbols. Watanabe improved on his earlier machine to give 5-state, 4-symbol and 7-state, 3-symbol semi-weakly universal machines [Wat72]. These semi-weak machines are plotted as hollow diamonds in Figure 1.1.1.

Recently, Woods and Neary [WN07b, WNb] have given semi-weakly universal machines with state-symbol pairs of $(2, 14)$, $(3, 7)$, and $(4, 5)$ that

states	symbols	tape dimension	tapes	author
15	2	1	3	Moore [Moo52]
1	2	1	4	Hooper [Hoo63, Hoo69]†
2	3	1	2	Hooper [Hoo63, Hoo69]
8	4	2	1	Wagner [Wag73]
2	7	2	1	Ottmann [Ott75a]†
10	2	2	1	Ottmann [Ott75b, KBO77]†
6	3	2	1	Ottmann [Ott75b, KBO77]†
4	4	2	1	Ottmann [Ott75b, KBO77]†
2	6	2	1	Kleine-Büning & Ottmann [KBO77]†
1	7	3	1	Kleine-Büning & Ottmann [KBO77]†
2	5	2	1	Kleine-Büning & Ottmann [KBO77]†
2	3	2	1	Kleine-Büning & Ottmann [KBO77]†
4	5	2	1	Kleine-Büning & Ottmann [KBO77]
3	6	2	1	Kleine-Büning & Ottmann [KBO77]
10	2	2	1	Kleine-Büning [KB77]
2	5	2	1	Kleine-Büning [KB77]
2	4	2	1	Priese [Pri79]
2	2	2	2	Priese [Pri79]
4	7	1	1	Pavlotskaya [Pav96]‡
2	5	1	1	Margenstern & Pavlotskaya [MP95b]‡
2	3	1	1	Margenstern & Pavlotskaya [MP03]‡

Table 1.1.2: Table of small non-standard universal Turing machines, excluding semi-weak machines. Weakly universal machines are denoted by †. Turing machines that are universal when coupled with a finite automaton are denoted by ‡.

simulate cyclic tag systems. These semi-weak machines are plotted as solid diamonds in Figure 1.1.1.

A further generalisation on the standard model is to allow the blank portion of the Turing machine's tape to have an infinitely repeated word to the left, and another to the right. We refer to such universal machines as weakly universal Turing machines. Cook and Eppstein [Coo04], and Wolfram [Wol02] recently gave weakly universal Turing machines, smaller than Watanabe's semi-weak machines, that simulate the universal cellular automaton Rule 110. These machines have state-symbol pairs of $(7, 2)$, $(4, 3)$, $(3, 4)$, and $(2, 5)$ and are plotted as hollow squares in Figure 1.1.1. (Note that David Eppstein constructed the $(7, 2)$ machine to be found in [Coo04].)

1.1.3 Other non-standard universal Turing machines

Weakness has not been the only generalisation on the standard model in the search for small universal Turing machines. We give some notable examples here, others are to be found in Table 1.1.2. Hooper [Hoo63, Hoo69] gave a universal machine with 2 states, 3 symbols, and 2 tapes, and another with 1 state, 2 symbols and 4 tapes. One of the tapes in Hooper's 4-tape machine is circular and contains the simulated program. His machine would also operate correctly if this circular tape is replaced with a semi-weak tape. Thus Hooper's 4-tape machine could be considered semi-weak. Priese [Pri79] gave a 2-state, 4-symbol machine with a 2-dimensional tape, and a 2-state, 2-symbol machine with a pair of 2-dimensional tapes. Margenstern and Pavlotskaya [MP95b, MP03] gave a 2-state, 3-symbol Turing machine that uses only 5 instructions and is universal when coupled with a finite automaton.

1.1.4 Universal Turing machines with restrictions

If we restrict the standard Turing machine model the problem of finding machines with small state-symbol products becomes more difficult. Non-erasing Turing machines are a restriction of Turing machines that are permitted to overwrite blank symbols only. Moore [Moo52] mentions that Shannon had proved that non-erasing Turing machines simulate Turing machines, however this result was never published. Shortly after, Shannon proved 2-symbol Turing machines universal, Wang [Wan57] proved 2-symbol non-erasing Turing machines universal. Later, Minsky [Min61] proved the same result as Wang using a different technique. Minsky proved 2-tape non-writing Turing machines were universal and showed 2-symbol non-erasing Turing machines simulate these non-writing machines. More recently, Margenstern [Mar92, Mar93, Mar94, Mar95a, Mar95b, Mar01] has constructed a number of small non-erasing universal machines with further restrictions.

Fischer [Fis65] gives universality results for Turing machines that use restricted forms of transition rules. Fischer gives results for variations on the quadruple formulation (see Footnote 1.1). In one result he proves 3-state Post machines universal.

1.1.5 Universal tag systems

Post [Pos43] proved that a restriction of his canonical systems, called normal systems, are universal. Post also wondered if tag systems, a restriction of normal systems, had an unsolvable prediction problem. Minsky [Min60b, Min61] settled this problem when he proved that tag systems with deletion number 6 (called 6-tag systems) simulate Turing machines and hence

are universal. Later, Cocke and Minsky [Min62b, CM63, CM64] proved that 2-tag systems are universal by showing that they simulate Turing machines. Their technique used productions (appendants) of length 4 or less. Wang [Wan63] further improved on this result by showing that 2-tag systems with productions of length 3 or less also simulate Turing machines. Wang also proved the universality of lag systems, a variation of tag systems. Recently, cyclic tag systems were proved universal [Coo04, Wol02]. Kudlek and Rogozhin [KR01b, KR01a] introduced another tag like system called a circular Post machine. The operation of a circular Post machine is also similar to that of a Turing machine with a circular tape and a tape head that only moves one direction. Small universal circular Post machines have been given by Kudlek, Rogozhin and Alhazov [KR01b, KR01a, KR03, AKR02].

1.1.6 Simple universal cellular automata

Since cellular automata [vN66] were first proved universal there have been a number of incremental steps towards giving simpler universal cellular automata. Here we consider a cellular automata to be universal if it is Turing universal. Below we give results only for the most common types of cellular automata, those with one-dimensional nearest neighbour and two-dimensional von Neumann (5 neighbours) and Moore neighbourhoods (8 neighbours).

Codd [Cod68] gave a universal cellular automaton with von Neumann neighbourhood and 8 states on a blank background. Banks [Ban70] reduced the number of states needed for universality to 3 for a blank background and 2 for a periodic background. Conway [BCG82] proved that with a Moore neighbourhood it is possible to have universality with only 2 states on a blank background. Smith [Smi71], gave a one-dimensional nearest neighbour cellular automaton with 18 states that is universal on a blank background. Albert and Culik [AC87] reduced the number of states sufficient for universality, on a blank background, to 14. Lindgren and Nordahl [LN90] further reduced the number of states needed for universality to 9 on a blank background and 7 on a periodic background. Recently, Cook [Coo04] proved that Rule 110, a one-dimensional nearest neighbour cellular automaton with 2 states, is universal on a periodic background (a sketch of Cook's proof also appears in [Wol02]). There have been many other forms of simple cellular automata given. Both Albert and Culik [AC87], and Lindgren and Nordahl [LN90], have given one-dimensional universal cellular automata with neighbourhoods greater than three. Some other examples of simple universal cellular automata have been given for reversible cellular automata [Tof77, MH89, MF05], majority voting cellular automata [Moo97a], and cellular automata in the hyperbolic plane [HM03, IIM07, Mar06].

In the literature the stronger notion of intrinsic universality [Oll02] is

also used. An intrinsically universal cellular automaton simulates other cellular automata in linear time using a constant number of cells to encode a single cell. Ollinger [Oll02] gave a one dimensional nearest neighbour cellular automaton that is intrinsically universal and has only 6 states. Later Richard [Ric06] improved this result by giving a 4-state intrinsically universal cellular automaton.

1.1.7 Other simple universal systems

Many biologically inspired computational models have also been simplified to give simple universal models. Some examples of these are neural networks [MP43, Sie98], H systems [Hea87, PRS98] (also called splicing systems), P systems [Pău00, Pău02] (also called membrane systems), and spiking neural P systems [IPY06]. Neural networks have been around since the 1940s and more recently a number of different authors have given increasingly simple universal neural networks [Ind95, KCG94, KS96, Pol87, SM99, SS91, SS95]. In 1987, Head systems were born. Some results from the area of simple universal Head systems are to be found in [FMKY00, HM05, MR02]. Membrane computing has received much attention since Păun introduced this model of computation. Some results relating to small universal membrane systems can be found in [AFR06, AR06, CVMVV07, FO06, NPRP06, RV06]. Spiking neural P systems [IPY06] are a very new model inspired by a fusion of spiking neural networks and P systems. These systems have already given rise to a number of number of small spiking neural P systems [PP07, ZZP, Nea08a, Nea08b].

Minsky [Min67] proved that register machines with only 2 registers are universal. Later focusing on a different parameter, Korec [Kor96] proved that between 14 and 32 instructions are sufficient for universality depending on the type of instructions allowed. Morita [Mor96] has proved that reversible registers machines with 2 registers are universal. Benett [Ben73] has shown that 3-tape reversible Turing machines are universal. Morita [MSG89] improved on this result proving that reversible Turing machines with 1 tape and 2 symbols are universal. More recently Morita and Yamaguchi [MY07] gave a universal reversible Turing machine with 1 tape, 17 states, and 5 symbols.

The very earliest proofs of universality, the negative solution to the Entscheidungsproblem, and the first problems proved undecidable are to be found in Davis's [Dav65] book. Minsky's [Min67] book contains a number of early results on simple computationally universal models. More recently Margenstern [Mar00] gave a survey on the subject that catalogues many interesting results. Also, Delvenne et al. [DKB04, **?**] gives universality results for dynamical systems. There are a multitude of other simple universal models to be found in the literature but we will stop here.

1.2 Decidability and lower bounds

The pursuit to find the simplest universal models must also involve the search for lower bounds. To simplify our point we will take the example of Shannon's problem of finding the minimal state-symbol product for a universal Turing machine. For Shannon's problem, lower bounds involve finding the largest state-symbol product which, in some sense, is non-universal. One approach is to settle the decidability of the halting problem. However, we will see that this approach is not suitable for all models. We give an overview of decidability results for some of the models from Section 1.1.

Shannon [Sha56] claimed that 1-state Turing machines are non-universal. Fischer [Fis65] and Nozaki [Noz69] both note that Shannon's definition of universal Turing machine is too strict and so his proof is not sufficiently general. On page 281 of his book, Minsky [Min67] mentions that he and Bobrow proved that the halting problem is decidable for Turing machines with 2 states and 2 symbols "by a tedious reduction to thirty-odd cases (unpublishable)." It is currently known that the halting problem is decidable for machines with the following state-symbol pairs $(2, 2)$ [DK89, Kud96, Pav73], $(3, 2)$ [Pav78], $(2, 3)$ (claimed by Pavlotskaya [Pav73]), $(1, n)$ [Her66a, Her66b, Her68c, Noz69, Fis65] and $(n, 1)$ (trivial), where $n \geqslant 1$. These results induce the decidable halting problem curve in Figure 1.1.1. Also, these decidability results imply that a universal Turing machine, that simulates any Turing machine M and halts if and only if M halts, is not possible for these state-symbol pairs. Hence these results give lower bounds on the size of universal machines of this type. While it is trivial to prove that the halting problem is decidable for weak machines with a halting state and state-symbol pairs of the form $(n, 1)$, it is not known whether the other decidability results given above generalise to weak Turing machines that have a halting state. More recently, Pavlotskaya [Pav02] has shown that the halting problem is decidable for machines with less than 5 instructions.

Nozaki [Noz69] claims the non-universality of Turing machines with state-symbol pairs of $(2, 2)$ and $(3, 2)$. Kryukov [Kry71] claimed that the state-symbol pair $(1, n)$ is non-universal and used a computer to solve the $(3, 2)$ case. However, in the English translation versions of these papers insufficient details are given to reconstruct these proofs. More details of the technique used by Kryukov is available in [Kry67].

Herman [Her66a, Her68a, Her68c] proved the halting problem decidable for a number of Turing machine variants with 1-state, including Turing machines with a single 2-dimensional tape. Wagner [Wag73] generalised Kryukov's [Kry67] non-universality result for 2-state, 2-symbol machines to Turing machines with n-dimensional tapes. Aandrea and Fischer [AF67]

proved the decidability of the halting problem for 2-state Post machines (see footnote 1.1). Fischer [Fis65] gives universality results for Turing machines that use restricted forms of transition rules. Fischer gives non-universality [Fis65] results for variations on the quadruple formulation (see Footnote 1.1).

Margenstern [Mar95a, Mar97b, Mar97a] introduced a useful notion, that of decidability criterion, which we now define. Let f be a positive integer-valued function defined on a set of Turing machines M_1 such that, there is an integer d where for all machines with $f < d$, the halting problem is decidable and for each $f \geqslant d$ a universal machine exists. We say that f has frontier value of d for M_1. Also, d may be described as a boundary between universality and non-universality in the following sense. A decidability criterion f implies that a universal Turing machine, that simulates any Turing machine M and halts if and only if M halts, is not possible in M_1 for $f < d$.

Recall from Section 1.1.3 that Margenstern and Pavlotskaya gave a universal (Turing machine, finite automaton) pair where the Turing machine uses only 5 instructions. Margenstern and Pavlotskaya [MP03] also show that the halting problem is decidable for all such pairs if the Turing machine has 4 instructions. Their results give a frontier value of 5 instructions for the Turing machine in such pairs. Hence they have given the smallest possible Turing machine that is universal in this sense. Note that here we are considering the notion of non-universality given in the final sentence of the previous paragraph.

The following decidability results of Margenstern and Pavlotskaya are for 2-symbol Turing machines. One decidability criterion they use is the number of colours. The number of colours of a machine is the number of distinct triples (σ, D, δ), where some transition rule in the machine has read symbol σ, move direction D, and write symbol δ. Pavlotskaya [Pav73, Pav75] established a frontier value of 3 colours for Turing machines. Margenstern [Mar93] established a frontier value of 5 colours for non-erasing Turing machines. Let l be the number of left instructions and r be the number of right instructions in a machine. The minimum of l and r is called the laterality number of a machine. Margenstern and Pavlotskaya [Pav73, MP95a] established a frontier value of 2 for the laterality number of Turing machines. Margenstern [Mar95a, Mar97b] established a frontier value of 3 for the laterality number of non-erasing Turing machines. The above results involved the construction of a number of different universal Turing machines. Margenstern gives a 125-state, 2-symbol non-erasing machine that uses only 5 colours [Mar93], a 218-state, 2-symbol non-erasing machine that uses only 3 left-move instructions [Mar95a], a 59-state, 2-symbol standard machine that uses only 6 left-move instructions [Mar01], and a 190-state, 2-symbol machine that uses only 3 left-move instructions [Mar01].

There are also decidability results given for other models. Wang [Wan63] showed that the reachability problem, and hence the halting problem, is de-

cidable for 1-tag systems. Stephen Cook [Coo66] proved that the reachability problem, and hence the halting problem, is decidable for non-deterministic 1-tag systems. Post [Pos65] mentioned that he solved the reachability problem for 2-tag systems with productions of length strictly less than 3; however he did not publish this result. Recently, De Mol [De Mol07] proved the reachability problem decidable for this class of tag systems.

A number of small Turing machines have been given for other interesting problems. Many of these machines lie between the current universality curve, and the current decidable halting problem curve. Margenstern [Mar98, Mar00] gives machines that simulate iterations of the Collatz function ($3x + 1$ problem) with state-symbol pairs of $(11, 2)$, $(5, 3)$, $(4, 4)$, $(3, 6)$ and $(2, 10)$. Later, Baiocchi [Bai98] reduced the size of some of these machines to give Turing machines with state-symbol pairs of $(10, 2)$, $(5, 3)$, $(4, 4)$, $(3, 5)$ and $(2, 8)$. Michel [Mic04, Mic93] has shown that there are Turing machines that simulate iterations of Collatz-like functions with state-symbol pairs of $(2, 4)$, $(3, 3)$, and $(5, 2)$. Kudlek [Kud96] has given a 4-state, 4-symbol machine that accepts a context-sensitive language. These machines would seem to suggest that it will be difficult to improve on the current decidable halting problem curve in Figure 1.1.1.

An interesting problem, introduced by Tibor Rado [Rad62], is the Busy Beaver problem. The problem is as follows. Let $T_{x,2}$ be the set of all binary Turing machines with x states. For a given x determine the maximum number of non-blank symbols on the tape of any machine in $T_{x,2}$ when it halts, having started on a blank tape (sometimes the maximum time before halting is also considered). To date the problem has been solved for the following values Rado [Rad62] $x = 2$, Lin and Rado [LR65] $x = 3$ and Brady [Bra83] $x = 4$. The best results known for $x = 5$ and $x = 6$ are given by Marxen[3] and Buntrock [MB90] and halt in 47,176,870 timesteps and 3×10^{1730} timesteps, respectively. Michel [Mic93, Mic04] gives results for a generalisation of the problem by allowing more than 2 tape symbols. He has given results for the following state-symbols pairs $(2, 3)$, $(3, 3)$ and $(2, 4)$. Lafitte and Papazian [LP07] proved that the 2-state, 3-symbol result given by Michel's machine solves the problem for this class of machines. They also gave an analysis of some other classes.

There are many other decidability results many of which are to be found in Margenstern's [Mar00] survey. Also, Delvenne [**?**] gives decidability results for dynamical systems.

1.3 Thesis outline

The results presented in this thesis are concerned with simple universal models of computation with a particular focus on small universal Turing machines. Many of our results may be thought of as a continuation of the work mentioned in the previous sections. However, a new aspect is added to the study of simple universal systems as we also focus on the resource usage, such as the time and space complexity [Pap95], of such models. The efficiency of many existing simple universal systems is improved and new efficient systems are given. Many of our results are to be found in Figure 1.3.1. Improvements in the simulation time of previously constructed machines may seen by comparing Figures 1.1.1 and 1.3.1.

Chapter 2 gives time/space complexity analysis of previous simple universal models. It also contains important definitions and discussions, as well as technical information regarding the notation used in the thesis.

Notice from Figure 1.1.1 that when work began on this thesis all of the smallest universal Turing machines were exponentially slow. The results in Chapter 3 were aimed at giving small *polynomial* time universal Turing machines. Let t be the running time of any deterministic single tape Turing machine M. Then, the main result in Chapter 3 states that there exists deterministic $O(t^2)$ polynomial time universal Turing machines with state-symbol pairs of $(3, 11)$, $(5, 7)$, $(6, 6)$, $(7, 5)$ and $(8, 4)$. To date, these are the smallest known standard machines that simulate Turing machines in $O(t^2)$ time. They are also the smallest known standard machines where direct simulation of Turing machines is the technique used to establish their universality. Each of these machines is plotted as a solid circle in Figure 1.3.1. This result established a polynomial time universal curve and gave a new simulation algorithm for universal Turing machines.

Our search to find simple models led us to cyclic tag systems and Rule 110. Cook [Coo04] proved that Rule 110 is universal. In Chapter 4 we prove that Rule 110 is an efficient simulator of Turing machines, an exponential improvement on the previous time overhead. Rule 110 simulates Turing machines via cyclic tag systems. We improve the cyclic tag system simulation time of Turing machines from exponential to polynomial. As a corollary, we find that the prediction problem for Rule 110 is P-complete. This is the simplest cellular automaton proved to be P-complete. As another corollary, all of the small weak and semi-weak Turing machines in Figure 1.3.1 now simulate in polynomial time, an exponential improvement.

In Chapter 5 our attention turns to tag systems. 2-tag systems are a type of Post system that were originally proved to simulate Turing machines, by Cocke and Minsky. Their simulation is exponentially slow. Here we give a proof that 2-tag systems simulate Turing machines efficiently in polynomial

[3] See webpage: http://www.drb.insel.de/~heiner/BB/

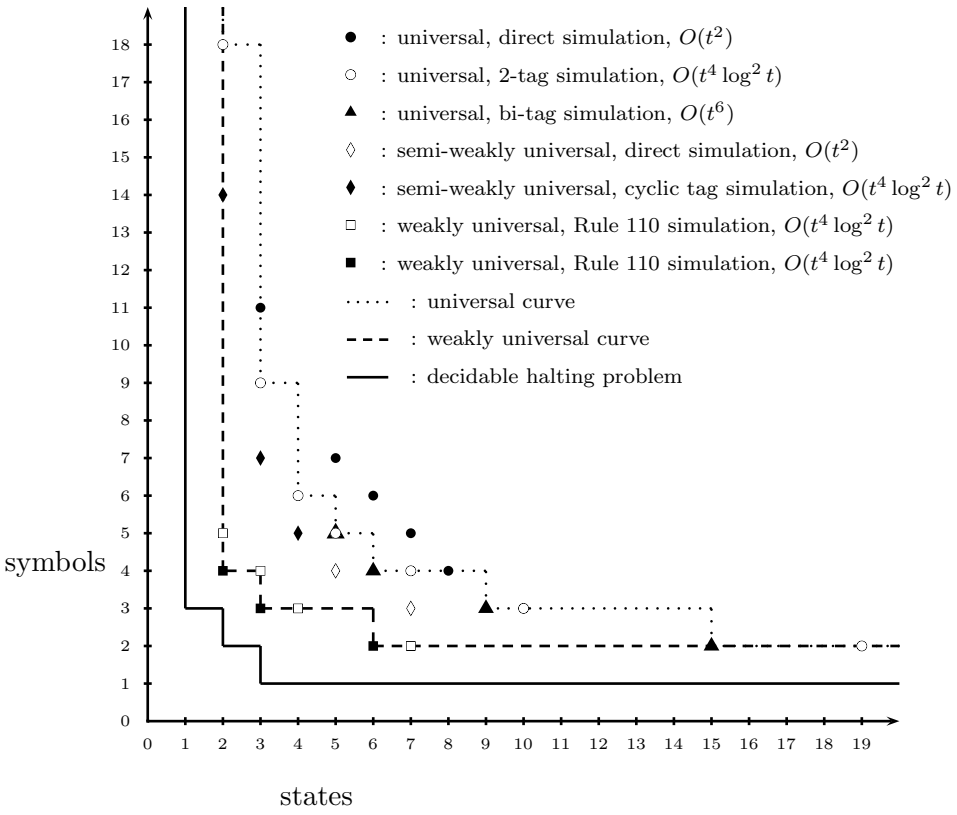


Figure 1.3.1: State-symbol plot of small universal Turing machines, including the work presented in this thesis. The (improved) simulation time and simulation technique is given for each group of machines. Each of our new universal Turing machines is represented by a solid shape.

time. As an immediate corollary, all of the small universal Turing machines that simulate 2-tag systems (see Figure 1.3.1) now simulate in polynomial time, an exponential improvement. This result also improves the efficiency of a number of other models that simulate 2-tag systems. In addition, we give a new variant on tag systems called bi-tag systems, and prove that they are efficient simulators of Turing machines.

In Chapter 6 we present small polynomial time universal Turing machines with state-symbol pairs of $(5, 5)$, $(6, 4)$, $(9, 3)$ and $(15, 2)$. These machines simulate bi-tag systems and are plotted as triangles in Figure 1.3.1. They are the smallest known standard universal Turing machines with 5, 4, 3, and 2 symbols, respectively. Our 5-symbol machine uses the same number of instructions (22) as the smallest known universal Turing machine by Rogozhin.

In Chapter 7 we give small weakly universal machines with state-symbol pairs of $(6, 2)$, $(3, 3)$ and $(2, 4)$. These machines improve on the size of

the Rule 110 simulators given by Cook and Eppstein [Coo04], and Wolfram [Wol02]. The machines we present here also simulate Rule 110 and are the smallest known weakly universal Turing machines. Our machines are plotted as solid squares in Figure 1.3.1.

Finally, in Chapter 8 we discuss our results and possible future work. Much of the work that we present and survey here has previously been published, and may be found in [Nea06, NW06c, NW06a, NW, NW07a, WN06c, WN06a, WN07a, WNa].

2

Preliminaries

We begin by giving definitions that are used throughout the thesis. We then discuss definitions of universal Turing machine and simulation that are given by a number of different authors. Next we establish notational conventions used in the thesis. Finally, we give a brief time/space complexity analysis of previous small universal Turing machines.

2.1 Definitions

The Turing machine model we choose has a single one-dimensional tape, one tape head, and is deterministic [HU79]. We choose this particular model as standard because it is common throughout the small Turing machine literature.

Definition 2.1.1 (Turing machine) *A Turing machine is a tuple* $M = (Q, \Sigma, b, f, q_1, H)$. *Here* Q *and* Σ *are the finite sets of states and tape symbols respectively. The blank symbol is* $b \in \Sigma$, $q_1 \in Q$ *is the start state, and* $H \subseteq Q$ *is the set of halt states. The transition function* $f : Q \times \Sigma \to \Sigma \times \{L, R\} \times Q$ *is defined for all* $q \in Q - H$. *If* $q \in H$ *then the function* f *is undefined on at least one element of* $q \times \Sigma$.

We write f as a list of transition rules. Each transition rule is a quintuple $t = (q_i, \sigma_1, \sigma_2, D, q_j)$, with initial state q_i, read symbol σ_1, write symbol σ_2, move direction D and next state q_j. The Turing machine definition we use here is standard in the literature. However, it is often common to have a set of input symbols (not including the blank symbol) that is a proper subset of the tape alphabet. This would not be suitable for our purposes.

Definition 2.1.2 (Weak Turing machine) *A weak Turing machine is a tuple* $M = (Q, \Sigma, r, l, f, q_1, H)$. *Here* $r \in \Sigma^*$ *and* $l \in \Sigma^*$ *are fixed constant length words called the right blank word and the left blank word, respectively. Definition 2.1.1 defines the remaining elements of the tuple* M.

Another related model is the *semi-weak* Turing machine. A semi-weak Turing machine has an infinitely repeated word to one side of its input, and on the other side a (standard) infinitely repeated blank symbol. Semi-weak

machines are a generalisation on the standard model and a restriction of the weak model.

Definition 2.1.3 (Turing machine configuration) *A configuration of a Turing machine consists of the current state q_i, the read symbol σ_0 and its tape contents $\ldots \sigma_{-3}\, \sigma_{-2}\, \sigma_{-1}\, \underline{\sigma_0}\, \sigma_1\, \sigma_2\, \sigma_3 \ldots$.*

Example 2.1.1 (Turing machine configuration) In the thesis we represent Turing machine configurations as follows:

$$\boldsymbol{q_i},\ \ldots \sigma_{-3}\, \sigma_{-2}\, \sigma_{-1}\, \underline{\sigma_0}\, \sigma_1\, \sigma_2\, \sigma_3 \ldots$$

where q_i is the current state, $\ldots \sigma_{-3}\, \sigma_{-2}\, \sigma_{-1}\, \sigma_0\, \sigma_1\, \sigma_2\, \sigma_3 \ldots$ is the tape contents and the tape head location given by an underline is under the read symbol σ_0.

Definition 2.1.4 (Turing machine computation step) *If the current state is q_i and the read symbol is σ_1 then the transition rule $(q_i, \sigma_1, \sigma_2, D, q_j)$ is applied to the configuration in the following way. The symbol under the tape head σ_1 is replaced with σ_2, q_j becomes the new current state, and if $D = L$ the tape head moves one cell to the left and if $D = R$ the tape head moves one cell to the right.*

A computation step is deterministic. In the sequel we write $c_1 \vdash c_2$ if a configuration c_2 is obtained from c_1 via a single computation step.

Example 2.1.2 (Turing machine computation step) We show the execution of the transition rule $(q_i, \sigma_0, \sigma_x, R, q_j)$.

$$\begin{aligned} &\boldsymbol{q_i},\ \ldots \sigma_{-3}\, \sigma_{-2}\, \sigma_{-1}\, \underline{\sigma_0}\, \sigma_1\, \sigma_2\, \sigma_3 \ldots \\ \vdash\ &\boldsymbol{q_j},\ \ldots \sigma_{-3}\, \sigma_{-2}\, \sigma_{-1}\, \sigma_x\, \underline{\sigma_1}\, \sigma_2\, \sigma_3 \ldots \end{aligned}$$

Definition 2.1.5 (Turing machine computation) *A computation is a finite sequence of configurations $c_1, c_2, \ldots, c_t$ of a Turing machine M that ends in a terminal configuration c_t where $\forall i,\ c_i \vdash c_{i+1}$. Also, we write $M(c_1) = c_t$.*

Definition 2.1.6 (Turing machine halting configuration) *A halting configuration is a terminal configuration where no transition rule is defined for the current state $q \in H$ and the read symbol.*

Definition 2.1.7 (Universal Turing machine) *A Turing machine U is universal if there exists recursive functions h and f such that*

$$M(c) = h(U(f(g(M, c)))) \tag{2.1.1}$$

where M is any Turing machine, c is a configuration of M, $g(M, c)$ is the Gödel number of the ordered pair (M, c), f maps $g(M, c)$ to a configuration of U, and h maps a terminal configuration of U to a terminal configuration of M.

The function f is injective and the function h is total and surjective. Also, the encoding function f and decoding function h are both recursive. The latter requirement, which is standard in the literature, ensures that the universality lies in the machine that we claim is universal and not in the encoding or decoding functions.

With the exception of some minor details, Definition 2.1.7 is equivalent to Priese's [Pri79] definition of universal Turing machine. It is also equivalent to Davis's [Dav57] definition if the terminal configuration is always assumed to be a halting configuration. Throughout the thesis it is assumed that the terminal configuration is a halting configuration unless explicitly stated otherwise.

In the definition of universal Turing machine it is also standard in the literature [Dav57, Noz69, Pri79] to use a Gödel numbering of Turing machines and Turing machine configurations as the domain of the encoding function. This ensures that: (1) there is a standard representation of (Turing machine, Turing machine configuration) pairs, and (2) the domain of f is the set of all (Turing machine, Turing machine configuration) pairs. In practice when constructing a universal Turing machine the domain of the encoding function is usually the table of behaviour (e.g. the set of transition rules) of a Turing machine and a Turing machine configuration. It is sufficient that the set of (Turing machines, Turing machine configuration) pairs that define the domain of our encoding function admit a Gödel numbering of all such pairs.

2.2 Some notes on universal Turing machines

We discuss definitions of universal Turing machine and then take a brief look at some previous definitions of universal Turing machine. Following this we consider the problem of defining the notion of simulation.

2.2.1 Previous definitions of universal Turing machine

Davis [Dav56] gave the following definition of universal Turing machine. A Turing machine is universal if the set of configurations which lead to a halting configuration is recursively enumerable complete. A set C is recursively

enumerable complete if it is recursively enumerable and for every recursively enumerable set R there is a recursive function f such that if $x \in R$, $f(x) \in C$. Later, Davis [Dav57] gave a more restricted definition which also required that the output of the machine being simulated be retrievable from the halting configuration of the universal machine via a recursive function. Davis's earlier definition requires only that the universal machine halts if and only if the simulated machine halts. Davis also proves that machines which obey his later definition also obey his earlier definition and that the converse of this is false.

Priese [Pri79] gives a definition of universal Turing machine that differs from Davis's definition in the following respect. Priese does not require the universal machine to halt. Priese's universal machine computes until a configuration at time t containing the encoded output is reached, such that a recursive function may be applied to any configuration after time $t-1$ to retrieve the output. Note that many different configurations may encode the same output.

Nozaki [Noz69] discusses the notion of universal Turing machine and gives a definition of universal Turing machine. Like Priese, Nozaki's definition does not require his universal machine to halt when the machine it simulates halts. Also, Nozaki's definition involves emulating the sequence of configurations of the machine being simulated. The definitions of Davis and Priese do not have this restriction. We discuss some of the implications of this restriction in the next section.

Finally, we note that to date all of the small universal Turing machines constructed are Nozaki-universal and Priese-universal. However, some of these machines are not Davis-universal as they do not halt.

2.2.2 The simulation of one abstract machine by another

The problem of defining simulation of one abstract machine by another has been discussed by a number of different authors [Fis65, Her68b, vEB90]. To quote van Emde Boas [vEB90]: "it is hard to define simulation as a mathematical object and still remain sufficiently general."

Fischer [Fis65] gives a definition of "simulation of one machine by another." where the following property holds. If machine M' simulates machine M then for each computation in M defined by the sequence of configurations $c_1, c_2, c_3 \ldots$ there is a computation $f(c'_{g(1)}), \ldots, f(c'_{g(2)}), \ldots, f(c'_{g(3)}), \ldots$ in M' such that the computation of M' halts if and only if the computation of M halts. Note $\forall i, g(i) > g(i+1)$. We are interested in the fact that every configuration c_i of the machine M being simulated is encoded by a unique configuration $c'_{g(i)}$ in M'. This requirement may exclude some simulations which may be considered reasonable. Take the example given by Herman [Her68b] where more than one computation step of M is simulated by a single computation step of M'. If this is the case then there

exists configurations in the computation of M that are not uniquely encoded in the computation of M'. In her definition of universal Turing machine, Nozaki [Noz69] uses a notion of simulation that is similar to Fischer's and so has the same problem of excluding reasonable simulations from her definition.

Herman accepted Fischer's invitation to improve his definition. Herman's definition is concerned only with the input-output relationship of the simulated machine. If machine M' simulates machine M then $M(c_1) = h(M'(f(c_1)))$ where f and h are the encoding and decoding functions respectively. The definitions of universal Turing machine given by Davis and Priese use this notion of simulation and so they avoid the problem of restricting the simulation technique in the manner of Fischer and Nozaki.

We have seen that the problem of defining the term simulation is simplified if we consider only the input-output relationship of the simulated machine. However, it is useful in our analysis to consider the sequence of computation steps taken by the simulator and the machine being simulated. While the notion of simulation in Definition 2.1.7 is not concerned with these steps we often talk about simulations in these terms. For instance in the sequel we often speak of the simulator simulating a transition rule of the machine being simulated. In our explanations we also use the terms simulated tape head, simulated read symbol, simulated current state ,etc. of the Turing machine being simulated. While in Definition 2.1.7 these objects are not defined; they are always well-defined in any of the proofs we give.

Another idea that is useful in our analysis of simulators is that of simulation technique. For instance model A is proved universal through simulation of Turing machines and model B is proved universal through simulation of model A. We say that model A simulates Turing machines directly and model B simulates Turing machines via model A. In terms of Definition 2.1.7 the notions of "simulates directly" and "simulate via model A" are redundant. However, we permit this abuse of the term "simulate" as we view the "simulation technique" as being important in our analysis of universal Turing machines.

2.3 Notational conventions

Throughout the thesis we adopt the following notational conventions. M denotes a deterministic Turing Machine with a single bi-infinite tape and a single tape head [HU79]. We let t denote the running time of M. U denotes a universal Turing machine and $U_{m,n}$ denotes some specific universal Turing machine with m states and n symbols. The encoding of M as a word is denoted $\langle M \rangle$. Analogously the encodings of state q and tape symbol σ are denoted $\langle q \rangle$ and $\langle \sigma \rangle$, respectively. For convenience we often call the word $\langle q \rangle$ a *state* of $\langle M \rangle$.

In the sequel we write $c_1 \vdash c_2$ if a configuration c_2 is obtained from c_1 via a single computation step. We let $c_1 \vdash^s c_2$ denote a sequence of s computation steps and let $c_1 \vdash^* c_2$ denote 0 or more computation steps.

In regular expressions $\cup$, *, ϵ and parentheses have their usual meanings [HU79]. Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, then Σ^* is the set of all words of length $\geqslant 0$ over the alphabet $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$. The length of the word $w \in \Sigma^*$ is denoted by $|w|$.

We use big-Oh notation in the analysis of time and space resource usage [Pap95]. Let $f : \mathbb{N} \mapsto \mathbb{N}$ and $g : \mathbb{N} \mapsto \mathbb{N}$ we write $g(n) = O(f(n))$ when there exists $b, n_0 \in \mathbb{N}$ such that for all $n > n_0$, $g(n) \leqslant bf(n)$.

2.4 Complexity analysis of previous simulations

We give some terminology that we use in our analysis of the time/space complexity of simulators. Let M be any Turing machine and let t be the running time of M. B is a polynomial time simulator of Turing machines if (1) $\exists k \in \mathbb{N}$ such that $\forall M : t_B = O(t^k)$ where t_B is the running time of B and (2) its encoding and decoding functions are logspace computable. We say that B simulates M in polynomial time $O(t^k)$. The terms exponential time simulator and linear time simulator may be defined analogously using appropriate encoding and decoding functions. Definitions of time and space simulation overheads are to be found in Boas [vEB90].

With the exception of Watanabe's Turing machines, the results given in this thesis give improvements on simulation times of all the simple models given in this section.

We give a time/space complexity analysis of Cocke and Minsky's [CM64] simulation of a single tape deterministic Turing machine M by a 2-tag system T_M. The tape contents of M has a maximum length of $O(t)$. This is encoded as a 2-tag system dataword of length $O(2^t)$. Thus $O(2^t)$ space is sufficient to simulate M. Each simulated timestep of M takes time $O(2^t)$. Hence $O(t2^t)$ time is sufficient to simulate the computation of M.

The universal machines of Minsky and Rogozhin et al. [Min62a, Rog96, KR02, Bai01] in Figure 1.1.1 simulate 2-tag systems with a quadratic polynomial overhead in time. Hence $O(t^2 2^{2t})$ time is sufficient to simulate the computation of M.

Cyclic tag systems simulate 2-tag systems in linear time [Coo04, Wol02]. Hence $O(t2^t)$ time is sufficient for cyclic tag systems to simulate the computation of M. Rule 110 simulates cyclic tag system in linear time [Coo04]. Hence $O(t2^t)$ time is sufficient for Rule 110 to simulate the computation of M. The weakly universal Turing machines of Cook and Eppstein [Coo04], and Wolfram [Wol02] in Figure 1.1.1 simulate Rule 110 with a quadratic increase in time. Hence $O(t^2 2^{2t})$ time is sufficient for these machines to simulate the computation of M.

The semi-weakly universal Turing machines of Woods and Neary [WN07b, WNb] in Figure 1.1.1 simulate cyclic tag system with a quadratic polynomial increase in time. Hence $O(t^2 2^{2t})$ time is sufficient to simulate the computation of M, via the above simulations.

The semi-weakly universal Turing machines of Watanabe simulate Turing machines directly with a quadratic polynomial increase in time. Hence $O(t^2)$ time is sufficient to simulate the computation of M.

The results we give in the thesis are for deterministic single tape Turing machines. The multitape Turing machine model is more usually used in complexity analysis. Some of our results also assume that the simulated machine has a binary tape alphabet. The simulation times we give do not change greatly when we consider deterministic multitape machines with larger alphabets. For example, let M' be a deterministic multitape Turing machine with more than two symbols. M' is converted to a two symbol, single tape Turing machine M. The number of states in M is only a constant times greater than the number of states and symbols of M', also M is at worst $O(t^2)$ polynomially slower than M' [Pap95].

3

Small $O(t^2)$ time universal Turing machines

3.1 Introduction

In this chapter we present deterministic $O(t^2)$ polynomial time universal Turing machines with state-symbol pairs of $(3, 11)$, $(5, 7)$, $(6, 6)$, $(7, 5)$ and $(8, 4)$. These are the smallest known machines that simulate Turing machines in $O(t^2)$ time. Each of the machines is plotted as a solid circle in Figure 3.1.1. The $O(t^2)$ polynomial time curve that is induced by these machines is also given in Figure 3.1.1.

Initially small universal Turing machines were constructed that directly simulated Turing machines [Ike58, Wat61]. Subsequently, the technique of indirect simulation via 2-tag systems was applied by Minsky [Min62a]. Due to their unary encoding of Turing machine tape contents, 2-tag systems were exponentially slow simulators of Turing machines [CM64]. Hence the small universal Turing machines of Minsky, Rogozhin, Kudlek and Baiocchi all suffered from a $O(t^2 2^{2t})$ exponential time overhead [Min62a, Rog96, KR02, Bai01]. In Chapter 5 we show that 2-tag systems simulate Turing machines efficiently in polynomial time. From this result it follows that the universal Turing machines of Minsky and Rogozhin et al. simulate in $O(t^4 \log^2 t)$ time. The smallest of these 2-tag simulators are plotted as hollow circles in Figure 3.1.1. The $O(t^4 \log^2 t)$ time curve induced by these machines is also plotted in Figure 3.1.1.

The small universal Turing machines presented in Chapter 6 are plotted as solid triangles in Figure 3.1.1. These machines simulate Turing machines via bi-tag systems in $O(t^6)$ time. Here we are particularly interested in the comparison between our $O(t^2)$ machines and other standard small universal machines. Thus only standard machines (see Section 1.1.1) appear in Figure 3.1.1.

Prior to the work in this chapter the smallest known polynomial time universal Turing machine was constructed by Watanabe [Wat61] in 1961 and has 8 states and 5 symbols. Our machines are significantly smaller and represent a new algorithm for small universal Turing machines. It should also be noted that they are the smallest known machines where direct simulation

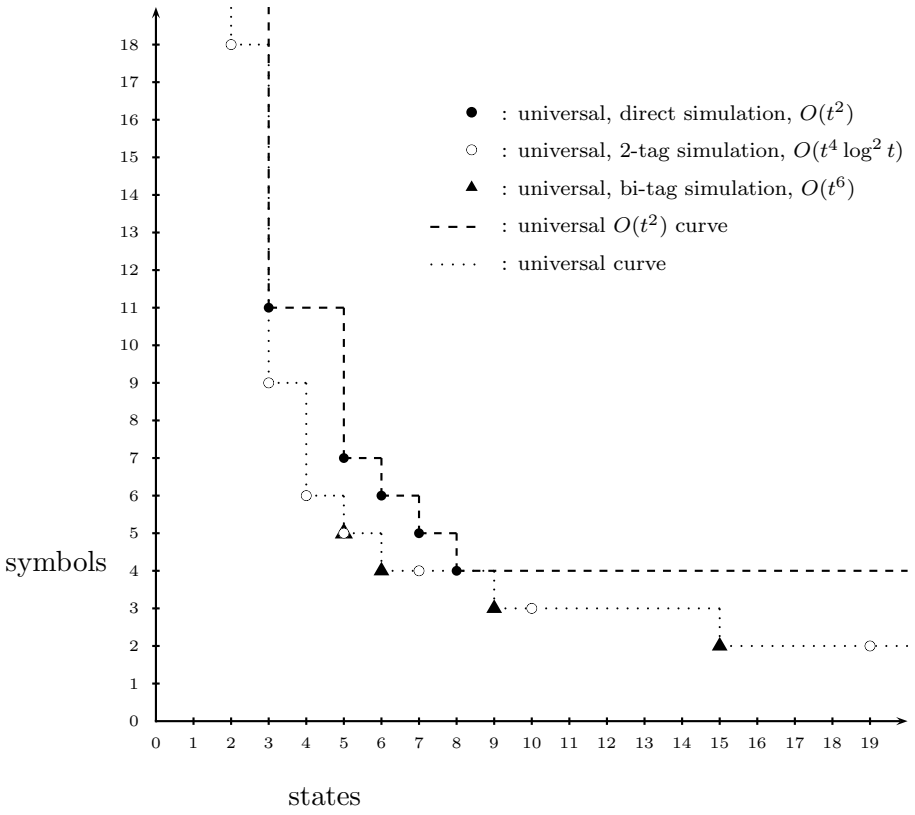


Figure 3.1.1: State-symbol plot of small universal Turing machines. Each new $O(t^2)$ machine is plotted as a solid circle.

of Turing machines is the technique used to establish their universality.

In Section 3.2 we give some definitions used to encode input to our universal Turing machines and an overview of the simulation algorithm. In Section 3.3 we give a 3-state, 11-symbol machine. We explain its input encoding and computation in some detail. Section 3.4 contains a proof of correctness which proves that this universal Turing machine simulates Turing machines in $O(t^2)$ polynomial time. In Section 3.5 our algorithm is extended to universal Turing machines with a number of other state-symbol pairs and finally a conclusion is given.

The results we present in this chapter we previously given in [NW05b, NW06c].

3.2 Preliminaries

At the beginning of this section we establish some formal conventions. We then introduce some general encodings that each of our five machines adhere

to. We also give an overview of our simulation algorithm. Each universal Turing machine uses a variation on this algorithm. The Turing machines we consider in this chapter have a single *one-way* infinite tape but otherwise adhere to Definition 2.1.1.

3.2.1 Input encodings for universal Turing machines

Without loss of generality, any simulated Turing machine M has the following restrictions: (i) M's tape alphabet is $\Sigma = \{0, 1\}$ and 0 is the blank symbol, (ii) for all $q_i \in Q$, i satisfies $1 \leqslant i \leqslant |Q|$, (iii) f is always defined, (iv) M's start state is q_1, (v) M has exactly one halt state $q_{|Q|}$ and its transition rules are of the form $(q_{|Q|}, 0, 0, L, q_{|Q|})$ and $(q_{|Q|}, 1, 1, L, q_{|Q|})$. Point (v) is a well-known halting technique that places the tape head at the beginning of the output. The following definitions encode M.

Definition 3.2.1 (Encoding of M's tape symbols) *The binary tape symbols 0 and 1 of M are encoded as the words $\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$ and $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$.*

Each of our five universal Turing machines has the symbols $\overleftarrow{a}$, $\overleftarrow{b}$ and λ as part of its tape alphabet. The symbols $\overleftarrow{a}$ and $\overleftarrow{b}$ are typically used to encode M's tape contents while λ is usually used as a marker symbol.

Definition 3.2.2 (Encoding of M's initial configuration) *The encoding of an initial configuration of M is of the form*

$$\langle M \rangle \langle q_1 \rangle \langle w \rangle \overleftarrow{a}^{\omega}$$

where $\langle q_1 \rangle$ is the start state of $\langle M \rangle$, $\langle w \rangle \in \{\overleftarrow{a}\,\overleftarrow{a}, \overleftarrow{b}\,\overleftarrow{a}\}^*$ is the encoding of the input w to M that is given by Definition 3.2.1, $\overleftarrow{a}^{\omega} = \overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{a}\ldots$, and $\langle M \rangle$ is the encoding of M:

$$\langle M \rangle = \lambda \mathcal{P}(f, q_{|Q|}) \lambda \mathcal{P}(f, q_{|Q|-1}) \lambda \ldots \lambda \mathcal{P}(f, q_2) \lambda \mathcal{P}(f, q_1) \lambda E \tag{3.2.1}$$

where the function $\mathcal{P}$ is defined below in Equation (3.2.2)*, and the word $E \in \{\epsilon, e, \overleftarrow{a}, \lambda\overleftarrow{b}\,\lambda\overleftarrow{a}, \overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,\lambda\overleftarrow{a}\}$ specifies the* ending.

The initial position of U's tape head is at the leftmost symbol of $\langle q_1 \rangle$.

In the previous definition the encoding of M is placed to the left of its encoded input. The initial position of M's simulated tape head is indicated by the word $\langle q_1 \rangle$ and is immediately to the left of the leftmost encoded input symbol. The remainder of the infinite tape of U contains the blank symbol $\overleftarrow{a}$. The ending E varies over the five universal Turing machines that we present.

The encoding of M's transition rules is defined using the function $\mathcal{P}$ that specifies the relative *positions* of encoded transition rules for a given state q_i.

$$\mathcal{P}(f, q_i) = \mathcal{E}(t_{i,1}) \lambda \mathcal{E}(t_{i,0}) \lambda \mathcal{E}(t_{i,0}) \lambda \mathcal{E}(t_{i,1}) \lambda \mathcal{E}'(f, t_{i,0}) \tag{3.2.2}$$

In Equation 3.2.2 $t_{i,1}$ and $t_{i,0}$ denote the unique transition rules for the state-symbol pairs $(q_i, 1)$ and $(q_i, 0)$, respectively. This notation is used for transition rules throughout this chapter. The encoding functions $\mathcal{E}$ and $\mathcal{E}'$ map transition rules to words called ETRs (encoded transition rules). There is a unique pair of $\mathcal{E}$ and $\mathcal{E}'$ functions for each of our five universal Turing machines. Given what we have so far, we need only to give $\mathcal{E}$, $\mathcal{E}'$ and $\langle q_1 \rangle$ to completely define the input to our universal Turing machines. These functions are given before each universal Turing machine.

3.2.2 Universal Turing machine algorithm overview

In order to distinguish the current state q_x of a simulated Turing machine M, the earliest small universal Turing machines [Ike58, Wat61] maintained a list of all states with a marker at q_x. A change in M's current state is simulated by moving the marker to another location in the list of states. The most significant difference between these earlier universal Turing machines and our algorithm is that we store the encoded current state of M on M's simulated tape at the location of M's tape head. Thus the encoded current state also records the current location of M's tape head during the simulation. This point is illustrated in Figure 3.2.3.

The problem of constructing a universal Turing machine can be divided into the following basic steps. The universal Turing machine (1) reads the encoded current state and (2) reads the encoded read symbol. Next the universal Turing machine (3) prints the encoded write symbol, (4) moves the simulated tape head and (5) establishes the new encoded current state. Due to the location of the encoded current state and the encodings that we use for our universal Turing machines, the sets {(1), (2)} and {(3), (4), (5)} each become a single process. Steps (1) and (2) are combined such that a single set of transition rules read both the encoded current state and the encoded read symbol. Steps (3), (4) and (5) have been similarly combined. Combining these steps has reduced the number of transition rules needed by our universal Turing machines.

Here we give a brief description of the simulation algorithm. The encoded current state of M is positioned at the simulated tape head location of M. Using a unary indexing method, our universal Turing machine U locates the next ETR (encoded transition rule) to execute (see Figure 3.2.1). The next ETR is indexed (pointed to) by the number of $\overleftarrow{b}$ symbols contained in the encoded current state and read symbol. If the number of $\overleftarrow{b}$ symbols in the encoded current state and encoded read symbol is i then the number of λ markers between the encoded current state and the next ETR to be executed is $i - 1$. To locate the next ETR, U simply neutralises the rightmost λ (i.e. replaces λ with some other symbol) for each $\overleftarrow{b}$ in the encoded current state and read symbol, until there is only one $\overleftarrow{b}$ remaining.

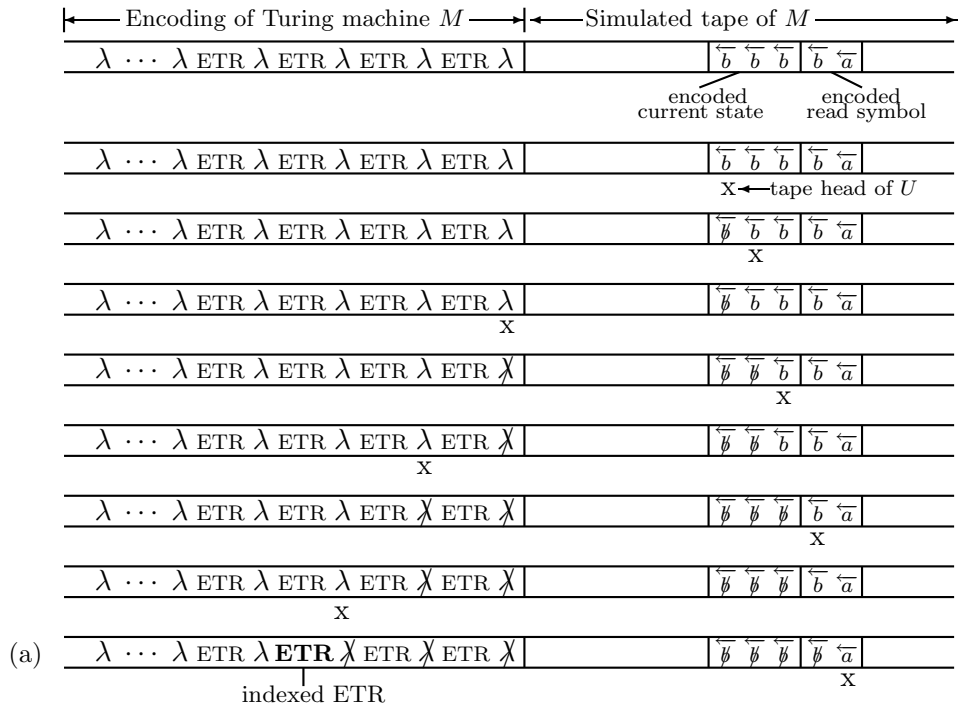


Figure 3.2.1: Indexing of an encoded transition rule (ETR) during simulation of a transition rule of M. The ETR to be executed is indexed by reading the encoded current state and read symbol, and marking off λ symbols in the encoding of M.

This indexed ETR is printed over the encoded current state and read symbol (see Figure 3.2.2). This printing completes the execution of the ETR and establishes the new encoded current state, encoded write symbol and simulated tape head move. Figure 3.2.3(b) represents the tape contents of U after an ETR of $\langle M \rangle$ is indexed. Figures 3.2.3(cR) and 3.2.3(cL) represent the two possibilities for U's tape contents after an ETR is printed. To give more details we present the algorithm as four cycles.

Cycle 1 (Index next ETR)

In this cycle U reads the encoded current state and encoded read symbol and neutralises markers to index the next ETR (see Figure 3.2.1). Initially U's tape head scans to the right until it reads a $\overleftarrow{b}$. This $\overleftarrow{b}$ is replaced with some other symbol. U's tape head then scans left to neutralise a λ marker. This process is repeated until U reads the subword $\overleftarrow{b}\,\overleftarrow{a}$ while scanning

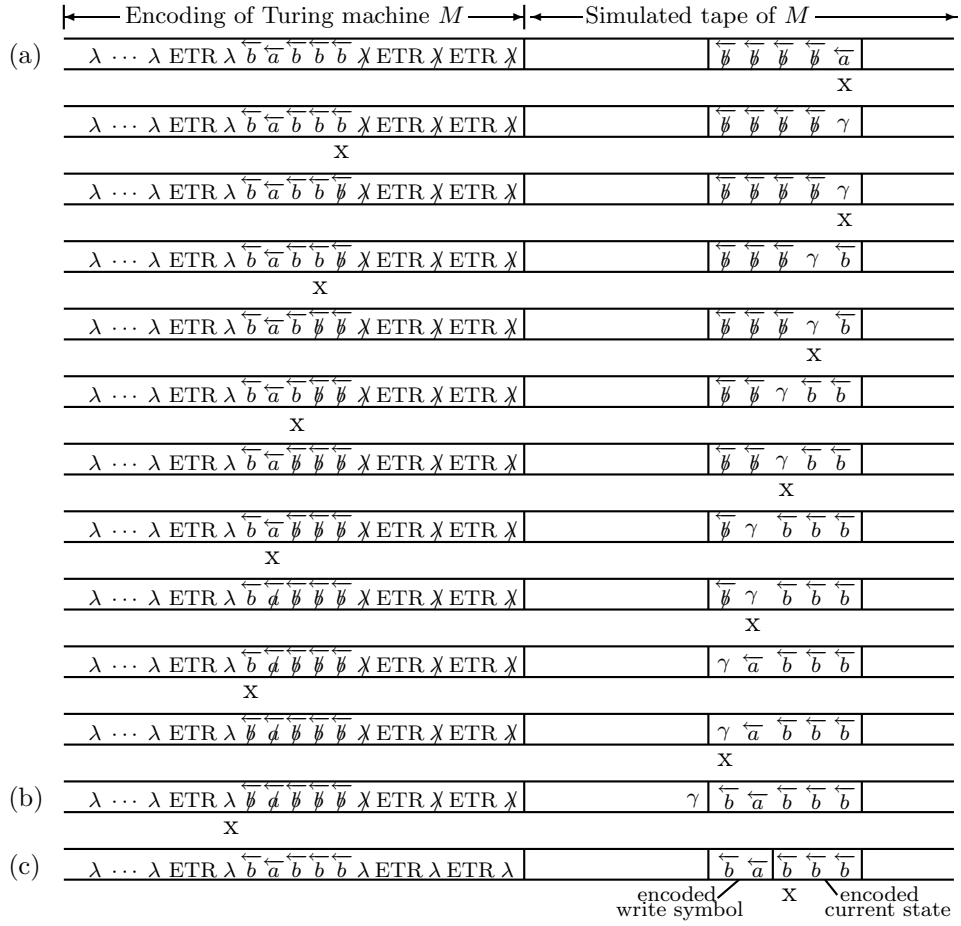


Figure 3.2.2: Printing of an encoded transition rule (ETR) during simulation of a transition rule of M. The ETR indexed in configuration (a) of Figure 3.2.1 is printed over the previous encoded current state and read symbol to complete the transition rule simulation.

right. This signals that the encoded current state and encoded read symbol have been read. Cycle 1 is now complete and Cycle 2 begins.

Cycle 2 (Print ETR)

Cycle 2 copies an ETR to M's simulated tape head location (see Figure 3.2.2). U scans left and records the next symbol of the ETR to be printed. U then scans right and prints the next symbol of the ETR at a location specified by a marker. The location of this marker is initially set at the end of Cycle 1 and its location is updated after the printing of each symbol of the ETR. This process is repeated until the end of the ETR is

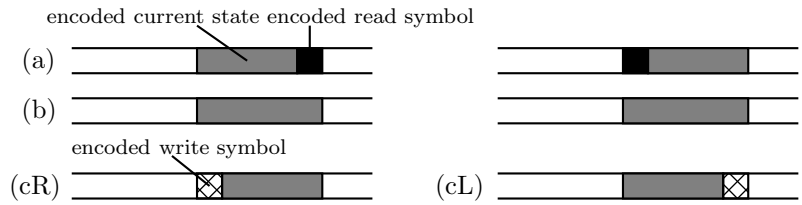


Figure 3.2.3: Right and left moving transition rule simulations. The encoded current state marks the location of M's simulated tape head. (a) Encoded configurations before beginning each transition rule simulation. (b) Intermediate configurations immediately after the encoded read symbol and encoded current state have been read. (cR) Configuration immediately after the simulated right move. (cL) Configuration immediately after the simulated left move.

detected causing U to enter Cycle 3. The end of the ETR is detected by U encountering the marker or neutralised marker that separates ETRs.

Cycle 3 (Restore tape)

Cycle 3 restores M's encoded table of behaviour after an ETR has been indexed and printed (see configurations (b) and (c) in Figure 3.2.2). U scans right restoring $\langle M \rangle$ to its initial value. This Cycle ends when U encounters the marker which was used in Cycle 2 to specify the position of the next symbol of the ETR to be printed. U then enters Cycle 4.

Cycle 4 (Choose read or write symbol)

This cycle either (i) begins the indexing of an ETR or (ii) completes the execution of an ETR. More precisely: (i) if U is immediately after simulating a left move then this cycle reads the encoded read symbol to the left of the encoded current state, (ii) if U is simulating a right move then this cycle prints the encoded write symbol to the left of the encoded current state. On completion of either case Cycle 1 is entered.

3.3 Construction of $U_{3,11}$

Our first machine has 3 states and 11 symbols and is denoted $U_{3,11}$. As usual we let M be a Turing machine that is simulated by $U_{3,11}$.

Definition 3.3.1 (Encoding of start state of $\langle M \rangle$ for $U_{3,11}$) *The start state of $\langle M \rangle$ is $\langle q_1 \rangle = \overleftarrow{a}^{5|Q|}\overleftarrow{b}^{2}$.*

Recall that $\langle M \rangle$ is the encoding of M and is defined via the functions $\mathcal{E}$ and $\mathcal{E}'$. These encoding functions map to words over the alphabet of $U_{3,11}$, as defined in Equations (3.3.1) and (3.3.2). We denote the words defined by $\mathcal{E}$ and $\mathcal{E}'$ with the acronyms ETR and ETR$'$, respectively.

Recall that t_{i,σ_1} denotes the unique transition rule $q_i, \sigma_1, \sigma_2, D, q_y$ in M with initial state q_i and read symbol σ_1. Also, $t^{R,i} = (q_x, \sigma_1, \sigma_2, R, q_i)$ and $t^{L,i} = (q_x, \sigma_1, \sigma_2, L, q_i)$; we write $\exists t^{R,i}$ to mean that there exists a transition rule which moves right and has q_i as its next state (there are zero or more such transition rules).

Let $t = (q_x, \sigma_1, \sigma_2, D, q_y)$ be a fixed transition rule in M, then t is encoded via Equation (3.2.2) using the function $\mathcal{E}$ on its own, or in conjunction with $\mathcal{E}'$, where

$$\mathcal{E}(t) = \begin{cases} e^{a(t)}h^{b(t)} & \text{if } D = R, \sigma_2 = 0 \\ he^{a(t)}h^{b(t)} & \text{if } D = R, \sigma_2 = 1 \\ e^{a(t)-1}h^{b(t)}eee & \text{if } D = L, \sigma_2 = 0 \\ e^{a(t)-1}h^{b(t)}ehe & \text{if } D = L, \sigma_2 = 1 \end{cases} \tag{3.3.1}$$

and

$$\mathcal{E}'(f,t) = \begin{cases} e^{a(t^{R,x})-3}h^{b(t^{R,x})+2} & \text{if } \exists t^{R,x}, q_x \neq q_1 \\ \epsilon & \text{if } \nexists t^{R,x}, q_x \neq q_1 \\ e^{5|Q|-3}h^4 & \text{if } q_x = q_1 \end{cases} \tag{3.3.2}$$

where as before $t^{R,x}$ is any right moving transition rule such that $t^{R,x} \vdash t$, the functions $a(\cdot)$ and $b(\cdot)$ are defined by Equations (3.3.3) and (3.3.4), e and h are tape symbols, and ϵ is the empty word.

$$a(t) = 5|Q| + 2 - b(t) \tag{3.3.3}$$

$$b(t) = 2 + \sum_{j=1}^{y} g(t,j) \tag{3.3.4}$$

where $g(\cdot)$ is given by

$$g(t,j) = \begin{cases} 5 & \text{if } j < y \\ 3 & \text{if } D = L, j = y \\ 0 & \text{if } D = R, j = y \end{cases} \tag{3.3.5}$$

Definition 3.3.2 (Encoding of M's current state for $U_{3,11}$) *The encoding of M's current state is of the form $\overleftarrow{a}^{*}\overleftarrow{b}^{2}\overleftarrow{b}^{*}\{\overleftarrow{a} \cup \epsilon\}$ and is of length $5|Q| + 2$.*

ETR	transition rule	$t^{R,x}$ for $\mathcal{E}'$	$b(t)$	$a(t)$	$\mathcal{E}'$ or $\mathcal{E}$
$\mathcal{E}'(f, t_{1,0})$	$q_1, 0, 1, R, q_2$	$q_1, 1, 0, R, q_1$	$2 + 0 = 2$	15	$e^{12}h^4$
$\mathcal{E}(t_{1,0})$	$q_1, 0, 1, R, q_2$		$2 + 5 + 0 = 7$	10	$he^{10}h^7$
$\mathcal{E}(t_{1,1})$	$q_1, 1, 0, R, q_1$		$2 + 0 = 2$	15	$e^{15}h^2$
$\mathcal{E}'(f, t_{2,0})$	$q_2, 0, 0, L, q_2$	$q_1, 0, 1, R, q_2$	$2 + 5 + 0 = 7$	10	e^7h^9
$\mathcal{E}(t_{2,0})$	$q_2, 0, 0, L, q_2$		$2 + 5 + 3 = 10$	7	$e^6h^{10}eee$
$\mathcal{E}(t_{2,1})$	$q_2, 1, 1, L, q_3$		$2 + 5 + 5 + 3 = 15$	2	$eh^{15}ehe$
$\mathcal{E}'(f, t_{3,0})$	$q_3, 0, 0, L, q_3$	null	null	null	ϵ
$\mathcal{E}(t_{3,0})$	$q_3, 0, 0, L, q_3$		$2 + 5 + 5 + 3 = 15$	2	$eh^{15}eee$
$\mathcal{E}(t_{3,1})$	$q_3, 1, 1, L, q_3$		$2 + 5 + 5 + 3 = 15$	2	$eh^{15}ehe$

Table 3.3.1: Values for the $a(\cdot)$ and $b(\cdot)$ functions, and for each ETR of $\langle M_1 \rangle$ in Example 3.3.1 .

The value of the ending E, from Equation (3.2.1), for $U_{3,11}$ is $E = e$.

Example 3.3.1 (Encoding of M_1 for $U_{3,11}$) Let Turing machine $M_1 = (\{q_1, q_2, q_3\}, \{0, 1\}, 0, f, q_1, \{q_3\})$ where $f = \{(q_1, 0, 1, R, q_2),\ (q_1, 1, 0, R, q_1),\ (q_2, 0, 0, L, q_2),\ (q_2, 1, 1, L, q_3),\ (q_3, 0, 0, L, q_3),\ (q_3, 1, 1, L, q_3)\}$. Using Equation (3.2.1), M_1 is encoded as:

$$\langle M_1 \rangle = \lambda \mathcal{P}(f, q_3)\lambda \mathcal{P}(f, q_2)\lambda \mathcal{P}(f, q_1)\lambda e$$

From Definition 3.3.1 the start state of $\langle M_1 \rangle$ is $\overleftarrow{a}^{15}\overleftarrow{b}^{2}$. Substituting the appropriate values from Equation (3.2.2) gives

$$\begin{aligned}\langle M_1 \rangle =& \lambda \mathcal{E}(t_{3,1})\lambda \mathcal{E}(t_{3,0})\lambda \mathcal{E}(t_{3,0})\lambda \mathcal{E}(t_{3,1})\lambda \mathcal{E}'(f, t_{3,0}) \\ & \lambda \mathcal{E}(t_{2,1})\lambda \mathcal{E}(t_{2,0})\lambda \mathcal{E}(t_{2,0})\lambda \mathcal{E}(t_{2,1})\lambda \mathcal{E}'(f, t_{2,0}) \\ & \lambda \mathcal{E}(t_{1,1})\lambda \mathcal{E}(t_{1,0})\lambda \mathcal{E}(t_{1,0})\lambda \mathcal{E}(t_{1,1})\lambda \mathcal{E}'(f, t_{1,0})\lambda e\end{aligned}$$

Rewriting $\langle M_1 \rangle$ using Equations (3.3.1) and (3.3.2) and the values given in Table 3.3.1 gives the word

$$\begin{aligned}\langle M_1 \rangle =& \lambda eh^{15}ehe\lambda eh^{15}eee\lambda eh^{15}eee\lambda eh^{15}ehe\lambda\epsilon\lambda eh^{15}ehe\lambda e^6h^{10}eee\lambda \\ & e^6h^{10}eee\lambda eh^{15}ehe\lambda e^7h^9\lambda e^{15}h^2\lambda he^{10}h^7\lambda he^{10}h^7\lambda e^{15}h^2\lambda e^{12}h^4\lambda e\end{aligned} \tag{3.3.6}$$

To aid understanding, note that a key property of $\mathcal{P}$ from Equation (3.2.2) is that it creates five ETRs in $\langle M \rangle$ for each state in M. Hence five ETRs encode two transition rules. This apparent redundancy is due to the algorithm used by our universal Turing machines. When executing an ETR, the algorithm makes use of the *direction of the previous tape head movement* of M. The leftmost ETR given by Equation (3.2.2) simulates execution of transition rule $t_{i,1}$ following a simulated left move. The second ETR from

the left simulates execution of transition rule $t_{i,0}$ following a simulated left move. The rightmost ETR and the centre ETR are both used to simulate execution of transition rule $t_{i,0}$ following a simulated right move. Finally the second ETR from the right simulates execution of transition rule $t_{i,1}$ following a simulated right move.

In our simulation, the number of $\overleftarrow{b}$ symbols in the encoded current state is used as a unary index to locate the *next* ETR to be executed. The function $b(\cdot)$ defined by Equation (3.3.4) gives the number of h symbols in an ETR. The number of h symbols in the ETR being executed defines the number of $\overleftarrow{b}$ symbols in the *next* encoded current state $\langle q_y \rangle$. The word $\mathcal{P}(f, q_y)$ gives the ETRs that encode the transition rules for state q_y. Hence the next ETR to be indexed is a subword of $\mathcal{P}(f, q_y)$ and $b(\cdot)$ is a summation dependant on all encoded states $\langle q_j \rangle$ such that $j \leqslant y$. The function g defined by Equation (3.3.5) is used by $b(\cdot)$ to calculate the number of ETRs in each $\langle q_j \rangle$. The first case of g corresponds exactly to the number of ETRs given in $\mathcal{P}$ (Equation (3.2.2)). The final two cases of g define whether the encoded current state points to the rightmost ETR ($g = 0$) in the list of ETRs for a state, or to the fourth from the right ($g = 3$).

It is important to note that the input and output encodings for our universal Turing machines are efficiently (logspace) computable. This is an important requirement for universal Turing machines that simulate Turing machines in polynomial time. Recall that a logspace transducer [Sip97] is a Turing machine that has a read-only input tape, a work tape, and a write-only output tape, where only the space used by the work tape is considered. Definition 3.2.2 gives the encoding of an initial configuration of M. The transducer that computes this input encoding to $U_{3,11}$ takes M and w as input, where M is explicitly given as a word in some straightforward manner.

Lemma 3.3.1 *Given Turing machine M as a word, and its input w, then there exists a logspace transducer that computes the input $\langle M \rangle \langle q_1 \rangle \langle w \rangle$ to $U_{3,11}$.*

Proof. The input to $U_{3,11}$ is given by Definition 3.2.2. Space of $O(\log |M|)$ is sufficient to compute $\langle M \rangle$ and $\langle q_1 \rangle$ via Equations (3.2.1) to (3.3.5). Constant space is sufficient to compute $\langle w \rangle$ via Definition 3.2.1. □

We state the lemma for $U_{3,11}$. However all five universal Turing machines in this chapter have logspace computable input encodings. The decoding of the output from $U_{3,11}$, and our four other universal Turing machines, is computed by a linear time, constant space transducer via Definition 3.2.1.

3.3.1 $U_{3,11}$ and its computation

Definition 3.3.3 ($U_{3,11}$) *Let Turing machine* $U_{3,11} = (\{u_1, u_2, u_3\}, \{\overleftarrow{a}, \overleftarrow{b}, e, h, \overrightarrow{e}, \overrightarrow{h}, \overleftarrow{e}, \overleftarrow{h}, \lambda, \delta, \gamma\}, \overleftarrow{a}, f, u_1, \{u_3\})$ *where* f *is given by Table 3.3.2.*

	u_1	u_2	u_3
$\overleftarrow{a}$	$\overleftarrow{e}, R, u_1$	γ, L, u_2	$\overleftarrow{a}, L, u_3$
$\overleftarrow{b}$	$\overleftarrow{e}, R, u_2$	$\overleftarrow{b}, L, u_1$	e, R, u_1
e	$\overrightarrow{e}, L, u_1$	$\overleftarrow{e}, R, u_1$	e, R, u_1
h	$\overrightarrow{h}, L, u_1$	$\overleftarrow{h}, R, u_3$	$\overleftarrow{a}, L, u_1$
$\overrightarrow{e}$	$\overleftarrow{e}, R, u_1$	e, R, u_2	$\overleftarrow{e}, R, u_3$
$\overrightarrow{h}$	$\overleftarrow{h}, R, u_1$	h, R, u_2	$\overleftarrow{h}, R, u_3$
$\overleftarrow{e}$	$\overrightarrow{e}, L, u_1$	$\overrightarrow{e}, L, u_2$	γ, L, u_2
$\overleftarrow{h}$	$\overrightarrow{h}, L, u_1$	$\overrightarrow{h}, L, u_2$	$\overleftarrow{a}, L, u_3$
λ	δ, R, u_1	λ, R, u_2	δ, R, u_3
δ	λ, L, u_1	λ, L, u_2	
γ	$\overleftarrow{a}, L, u_3$	$\overleftarrow{h}, R, u_3$	$\overleftarrow{b}, L, u_3$

Table 3.3.2: Table of behaviour for $U_{3,11}$.

We give an example of $U_{3,11}$ simulating a transition rule of M_1 from Example 3.3.1. This simulation is of the first step in M_1's computation for a specific input. The example is presented as the 4 cycles given in Section 3.2.2. In the below configurations the current state of $U_{3,11}$ is highlighted in bold font, to the left of $U_{3,11}$'s tape contents. M_1's encoded read and write symbols are also highlighted in bold font. The position of $U_{3,11}$'s tape head is given by an underline. In the sequel we use the term *overlined region*.

Definition 3.3.4 (Overlined region) *The overlined region exactly spans the encoded current state (has length* $5|Q| + 2$*), except on completion of reading an encoded read symbol (has length* $5|Q| + 4$*) until the next encoded current state is established.*

Example 3.3.2 ($U_{3,11}$**'s simulation of the right moving transition rule** $t_{1,1} = (q_1, 1, 0, R, q_1)$ **from Turing machine** M_1**)** The start state of $U_{3,11}$ is u_1 and tape head of $U_{3,11}$ is over the leftmost symbol of $\langle q_1 \rangle$ (as in Definition 3.2.2). In this example M_1's input is 101 (encoded via $\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$ and $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$). $\langle M_1 \rangle$ is in start state $\langle q_1 \rangle$ with encoded read symbol $\langle 1 \rangle$. Thus the initial configuration of U is:

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^3\lambda e\overline{\underline{\overleftarrow{a}}\,\overleftarrow{a}^{14}\overleftarrow{b}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

Cycle 1 (Index next ETR)

	u_1
$\overleftarrow{a}$	$\overleftarrow{e}, R, u_1$
$\overleftarrow{b}$	$\overleftarrow{e}, R, u_2$
$\overrightarrow{e}$	$\overleftarrow{e}, R, u_1$
$\overrightarrow{h}$	$\overleftarrow{h}, R, u_1$
λ	δ, R, u_1

	u_2
$\overleftarrow{a}$	γ, L, u_2
$\overleftarrow{b}$	$\overleftarrow{b}, L, u_1$

	u_1
e	$\overrightarrow{e}, L, u_1$
h	$\overrightarrow{h}, L, u_1$
$\overleftarrow{e}$	$\overrightarrow{e}, L, u_1$
$\overleftarrow{h}$	$\overrightarrow{h}, L, u_1$
λ	δ, R, u_1
δ	λ, L, u_1

Table 3.3.3: Sub-tables for Cycle 1 of $U_{3,11}$.

In Cycle 1 the leftmost table (above) reads the encoded current state. The rightmost table scans left and neutralises markers to index the next ETR. The middle table decides when the cycle is complete. $U_{3,11}$ scans the encoded current state from left to right in state u_1; each $\overleftarrow{b}$ is replaced with an $\overleftarrow{e}$ and $U_{3,11}$ then enters state u_2 to see if it is finished reading the encoded current state and encoded read symbol. $U_{3,11}$ is simulating transition rule $t_{1,1}$ which is encoded by $\mathcal{E}(t_{1,1})$. Hence we have replaced the shorthand notation $\mathcal{E}$ with the word $e^{15}h^2$ defined by $\mathcal{E}(t_{1,1})$. The word $e^{15}h^2$ appears in the location defined by Equation (3.3.6). After the initial configuration we have:

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{\overleftarrow{e}\,\underline{\overleftarrow{a}}\,\overleftarrow{a}^{14}\overleftarrow{b}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}}^{\omega}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{\overleftarrow{e}\,\overleftarrow{e}^{14}\underline{\overleftarrow{b}}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{\overleftarrow{e}\,\overleftarrow{e}^{14}\overleftarrow{e}\,\underline{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}}^{\omega}$$

The leftmost $\overleftarrow{b}$ is replaced with an $\overleftarrow{e}$. $U_{3,11}$ then moves right to test if it is finished reading the encoded current state. If not, $U_{3,11}$ reads another $\overleftarrow{b}$, then scans left in state u_1 and neutralises the rightmost λ marker:

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\underline{\lambda}\,\overrightarrow{e}\,\overrightarrow{e}^{15}\overrightarrow{e}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\delta\underline{\overrightarrow{e}}\,\overrightarrow{e}^{15}\overrightarrow{e}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

Having neutralised a λ marker, $U_{3,11}$ scans right in state u_1 searching for the next $\overleftarrow{b}$.

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\delta\overleftarrow{e}\,\overleftarrow{e}^{15}\overleftarrow{e}\,\underline{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

The neutralisation process is repeated until the end of this cycle. Thus the number of $\overleftarrow{b}$ symbols index the next ETR to be executed. $U_{3,11}$ is finished reading the encoded current state and read symbol when $U_{3,11}$ reads a $\overleftarrow{b}$ in state u_1, moves right to test for the end of the encoded current state and encoded read symbol, and reads an $\overleftarrow{a}$ in state u_2. In the configurations below when all the e and h symbols in an $\mathcal{E}'$ or an $\mathcal{E}$ are replaced with $\overleftarrow{e}$ and $\overleftarrow{h}$ symbols the resulting word is denoted $\overleftarrow{\mathcal{E}'}$ or $\overleftarrow{\mathcal{E}}$ respectively. Similarly when all the e and h symbols in an $\mathcal{E}'$ or an $\mathcal{E}$ are replaced with $\overrightarrow{e}$ and $\overrightarrow{h}$ symbols the resulting word is denoted $\overrightarrow{\mathcal{E}'}$ or $\overrightarrow{\mathcal{E}}$ respectively.

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\overleftarrow{e}^{15}\overleftarrow{e}\,\overleftarrow{e}\,\underline{\boldsymbol{\overleftarrow{b}}}\,\boldsymbol{\overleftarrow{a}}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\overleftarrow{e}^{15}\overleftarrow{e}\,\overleftarrow{e}\,\boldsymbol{\overleftarrow{e}}\,\underline{\boldsymbol{\overleftarrow{a}}}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\overleftarrow{e}^{15}\overleftarrow{e}\,\overleftarrow{e}\,\underline{\overleftarrow{e}}\,\gamma}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega} \qquad \text{(I)}$$

In configuration (I) above $U_{3,11}$ has entered Cycle 2. Also, the overlined region is now extended to include the encoded read symbol as this has been read and thus recorded in the same manner as the encoded current state.

Cycle 2 (Print ETR)

	u_2
$\overleftarrow{a}$	γ, L, u_2
e	$\overleftarrow{e}, R, u_1$
h	$\overleftarrow{h}, R, u_3$
$\overleftarrow{e}$	$\overrightarrow{e}, L, u_2$
$\overleftarrow{h}$	$\overrightarrow{h}, L, u_2$
λ	λ, R, u_2
δ	λ, L, u_2

	u_1
$\overrightarrow{e}$	$\overleftarrow{e}, R, u_1$
$\overrightarrow{h}$	$\overleftarrow{h}, R, u_1$
λ	δ, R, u_1
γ	$\overleftarrow{a}, L, u_3$

	u_3
$\overrightarrow{e}$	$\overleftarrow{e}, R, u_3$
$\overrightarrow{h}$	$\overleftarrow{h}, R, u_3$
$\overleftarrow{e}$	γ, L, u_2
λ	δ, R, u_3
γ	$\overleftarrow{b}, L, u_3$

Table 3.3.4: Sub-tables for Cycle 2 of $U_{3,11}$.

This cycle copies an ETR to $\langle M\rangle$'s tape head position. The leftmost table scans left and records the next symbol of the ETR to be printed. The two right tables scan right and print the appropriate symbol. In the configurations below, $U_{3,11}$ scans left until a h is read. Then $U_{3,11}$ moves right and records this h by entering u_3.

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}hh\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\overleftarrow{e}^{17}\underline{\overleftarrow{e}}\,\gamma}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h\underline{h}\lambda\overrightarrow{\mathcal{E}'}\lambda\overrightarrow{e}\,\overline{\overrightarrow{e}^{17}\overrightarrow{e}\,\gamma}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h\overleftarrow{h}\,\underline{\lambda}\overrightarrow{\mathcal{E}'}\lambda\overrightarrow{e}\,\overline{\overrightarrow{e}^{17}\overrightarrow{e}\,\gamma}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$U_{3,11}$ now scans right until it reads a γ and prints the recorded symbol.

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h\overleftarrow{h}\,\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\overleftarrow{e}^{17}\overleftarrow{e}\,\underline{\gamma}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h\overleftarrow{h}\,\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\overleftarrow{e}^{17}\underline{\overleftarrow{e}}\,\overleftarrow{b}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h\overleftarrow{h}\,\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overline{\underline{\overleftarrow{e}}^{17}\gamma\,\overleftarrow{b}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

This printing process is iterated until $U_{3,11}$ is finished printing the ETR. The completion of this process occurs on reading a λ in state u_2.

$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\underline{\lambda}\overrightarrow{e}^{15}\overrightarrow{h}^2\lambda\overrightarrow{\mathcal{E}'}\lambda\overrightarrow{e}\,\overline{\overrightarrow{e}\gamma\overleftarrow{a}^{15}\overleftarrow{b}^{2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda\underline{\overrightarrow{e}}^{15}\overrightarrow{h}^2\lambda\overrightarrow{\mathcal{E}'}\lambda\overrightarrow{e}\,\overline{\overrightarrow{e}\gamma\overleftarrow{a}^{15}\overleftarrow{b}^{2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

Cycle 3 (Restore tape)

	u_2
$\overrightarrow{e}$	e, R, u_2
$\overrightarrow{h}$	h, R, u_2
λ	λ, R, u_2
γ	$\overleftarrow{h}\,, R, u_3$

Table 3.3.5: Cycle 3 of $U_{3,11}$.

This table restores M's simulated tape and encoded table of behaviour. This cycle is entered from Cycle 2 (Print ETR). In Cycle 3, $U_{3,11}$ moves right restoring each $\overrightarrow{e}$ to e and each $\overrightarrow{h}$ to h. This continues until $U_{3,11}$ reads γ, sending $U_{3,11}$'s control to u_3. Thus the configuration:

$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{e\underline{\gamma}\overleftarrow{a}\,\overleftarrow{a}^{14}\overleftarrow{b}^{2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

becomes:

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda ee\overline{\overleftarrow{h}\,\underline{\overleftarrow{a}}\,\overleftarrow{a}^{14}\overleftarrow{b}^{2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

Cycle 4 (Choose read or write symbol)

This cycle either (i) begins the indexing of an ETR or (ii) completes the execution of an ETR. More precisely: (i) if $U_{3,11}$ is immediately after simulating a left move then this cycle reads the encoded read symbol to the

	u_1	u_3
$\overleftarrow{a}$	$\overleftarrow{e}, R, u_1$	$\overleftarrow{a}, L, u_3$
$\overleftarrow{b}$		e, R, u_1
e		e, R, u_1
h		$\overleftarrow{a}, L, u_1$
$\overleftarrow{h}$		$\overleftarrow{a}, L, u_3$
λ		δ, R, u_3
δ		

Table 3.3.6: Cycle 4 of $U_{3,11}$.

left of the encoded current state, (ii) if $U_{3,11}$ is simulating a right move then this cycle prints the encoded write symbol to the left of the encoded current state. Case (ii) follows:

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{e\underline{\overleftarrow{h}}\,\overleftarrow{a}\,\overleftarrow{a}^{14}\overleftarrow{b}^{\,2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{e\underline{e}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{a}^{14}\overleftarrow{b}^{\,2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\overline{e\underline{\overleftarrow{a}}\,\overleftarrow{a}\,\overleftarrow{a}^{14}\overleftarrow{b}^{\,2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^3\lambda e^{15}h^2\lambda\mathcal{E}'\lambda e\boldsymbol{e}\overleftarrow{\boldsymbol{e}}\,\overline{\underline{\overleftarrow{a}}\,\overleftarrow{a}^{14}\overleftarrow{b}^{\,2}}\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega} \qquad \text{(II)}$$

In configuration (II) above we have shortened the overlined region; the two symbols $e\overleftarrow{e}$ to the left of M_1's encoded current state encode the write symbol 0.

The example simulation of transition rule $t_{1,1} = (q_1, 1, 0, R, q_1)$ is now complete. As $U_{3,11}$ simulates M_1 the encoded tape contents to the left of the simulated tape head is encoded as e and h symbols (i.e. $\langle 0\rangle = ee$ and $\langle 1\rangle = he$). The contents to the right is encoded as $\overleftarrow{a}$ and $\overleftarrow{b}$ symbols (as in Definition 3.2.1). This is not a problem as $U_{3,11}$ simulates halting by moving the simulated tape head to the left end of the tape. As a result the entire encoded tape contents of the Turing machine are to the right of the tape head and so are encoded by $\overleftarrow{a}$ and $\overleftarrow{b}$ symbols.

In configuration (II) above the encoded write symbol $\langle 0\rangle$ is written as the word $e\overleftarrow{e}$. This word will become ee after the next ETR has executed. The new encoded current state satisfies Definition 3.3.2. M_1's simulated tape head (the new encoded current state) is configured so that $U_{3,11}$ reads the next encoded read symbol to the right when searching for the next ETR. The $\overleftarrow{a}$ that signals the end of the encoded current state is provided by the next encoded read symbol $\langle 0\rangle$.

Remark 3.3.1 If the first read symbol of Example 3.3.2 is changed from a $\langle 1\rangle$ to a $\langle 0\rangle$, then one less $\overleftarrow{b}$ is read when indexing the next ETR. This

indexes the rightmost (rather than the second from the right) ETR.

3.4 Proof of correctness of $U_{3,11}$

In this section we prove that $U_{3,11}$ correctly simulates a number of the possible types of transition rules. We then extend these cases to all cases thus proving the correctness of $U_{3,11}$'s computation.

Lemma 3.4.1 *Given a valid initial configuration of $U_{3,11}$, the encoded start state indexes the ETR defined by $\mathcal{E}(t_{1,1})$ if M's read symbol is 1 and $\mathcal{E}'(f, t_{1,0})$ if M's read symbol is 0.*

Proof. The encoded start state contains exactly two $\overleftarrow{b}$ symbols. From Example 3.3.2 when $U_{3,11}$ reads a $\langle 1 \rangle$ in state $\langle q_1 \rangle$ it neutralises two λ markers thus locating the second ETR from the right. By Definition 3.2.2 and Equation (3.2.2) this ETR is defined by $\mathcal{E}(t_{1,1})$. From Remark 3.3.1 and Example 3.3.2 when $U_{3,11}$ reads a $\langle 0 \rangle$ in state $\langle q_1 \rangle$ it neutralises one λ, thus indexing the rightmost ETR defined by $\mathcal{E}'(f, t_{1,0})$. □

Example 3.4.1 ($U_{3,11}$'s simulation of the right moving transition rule $t_{1,0} = (q_1, 0, 1, R, q_2)$ from M_1) In this example $U_{3,11}$ is reading a $\langle 0 \rangle$ *after* a right move. The right move was given by the simulation of $t_{1,1} = (q_1, 1, 0, R, q_1)$ in Example 3.3.2. This unique case involves two steps, executing an ETR′ and then an ETR. The execution of an ETR′ is represented by parts (a) and (b) of Figure 3.4.1 and the execution of the subsequent ETR is represented by parts (c) and (d) of Figure 3.4.1.

We take the last configuration of Example 3.3.2, with the encoded read symbol $\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$ to the right of the encoded current state. Substituting the appropriate ETR′ $e^{12}h^4$ from Equation (3.3.6) gives:

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\lambda ee\overleftarrow{e}\,\overline{\underline{\overleftarrow{a}}\,15\,\overleftarrow{b}\,\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}\,\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\delta\overleftarrow{e}\,\overleftarrow{e}\,\overleftarrow{e}\,\overline{\overleftarrow{e}\,15\,\overleftarrow{e}\,\underline{\overleftarrow{e}}\,\gamma\boldsymbol{\overleftarrow{a}}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

In the configuration immediately above we have reached the end of Cycle 1 (Index next ETR). One λ has been replaced with one δ, thus indexing the ETR′ $e^{12}h^4$. The $\overleftarrow{b}\,\overleftarrow{a}$ that signaled the end of Cycle 1 was provided by the rightmost $\overleftarrow{b}$ of the encoded current state and the leftmost $\overleftarrow{a}$ of the encoded read symbol. Thus, only the leftmost $\overleftarrow{a}$ of $\langle 0 \rangle = \boldsymbol{\overleftarrow{a}\,\overleftarrow{a}}$ was read and this is sufficient to distinguish $\langle 0 \rangle$ from $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$. However, the overlined region does not cover the entire encoded read symbol which is why an ETR′

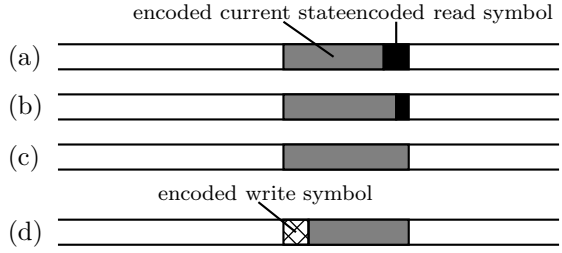


Figure 3.4.1: Right move simulation (special case). The encoded current state marks the location of M's simulated tape head. The configurations given in (a), (b) and (c) represent the reading of the encoded current state and an encoded 0 following a right move. (a) Encoded configuration before beginning the transition rule simulation. (b) Intermediate configuration after the encoded current state and first symbol of the encoded read symbol have been read. (c) Intermediate configuration immediately after the encoded read symbol has been read. (d) Configuration immediately after the simulated right move.

executes before an ETR in this unique case. Skipping to the end of Cycle 2 (Print ETR) gives:

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\underline{\lambda}\overrightarrow{e}^{12}\overrightarrow{h}^4\lambda\overrightarrow{e}\overrightarrow{e}\overrightarrow{e}\overrightarrow{e}\overline{\gamma\overleftarrow{a}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\lambda eeee\overline{\underline{\gamma}\overleftarrow{a}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\lambda eeee\overline{\overleftarrow{h}\underline{\overleftarrow{a}}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\lambda eeee\overline{\underline{\overleftarrow{h}}\overleftarrow{a}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\lambda eeee\overline{\underline{e}\overleftarrow{a}\overleftarrow{a}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^4\lambda e^{12}h^4\lambda eeee\overline{\underline{\overleftarrow{a}}\overleftarrow{a}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

At this point $U_{3,11}$ has executed the ETR′. $U_{3,11}$ now executes the ETR that represents the second step of the simulation of transition rule $t_{1,0}$. This ETR is defined by $\mathcal{E}(t_{1,0})$. Substituting the ETR $he^{10}h^7$ from Equation (3.3.6) into the configuration immediately above gives:

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\lambda he^{10}h^7\lambda\mathcal{E}\lambda\mathcal{E}'\lambda eeee\overline{\underline{\overleftarrow{a}}\overleftarrow{a}\overleftarrow{a}^{11}\overleftarrow{b}^4\boldsymbol{\overleftarrow{a}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

We now skip to the end of Cycle 1 (Index next ETR) giving:

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\lambda he^{10}h^7\delta\overleftarrow{\mathcal{E}}\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\overleftarrow{e}\overleftarrow{e}\overline{\overleftarrow{e}^{18}\underline{\boldsymbol{\overleftarrow{a}}}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\lambda he^{10}h^7\delta\overleftarrow{\mathcal{E}}\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\overleftarrow{e}\overleftarrow{e}\overline{\underline{\overleftarrow{e}}^{18}}\gamma\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega} \qquad \text{(III)}$$

The ETR is indexed by neutralising three λ markers. The second part of the $\langle 0 \rangle$ is read during this process. We now skip to the end of Cycle 3 (Restore tape) and illustrate a $\langle 1 \rangle$ being written to the left of the encoded current state:

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\underline{\lambda}\overrightarrow{h}\overrightarrow{e}^{10}\overrightarrow{h}^{7}\lambda\overrightarrow{\mathcal{E}}\lambda\overrightarrow{\mathcal{E}'}\lambda\overrightarrow{e}\overrightarrow{e}\overrightarrow{e}\overline{\gamma\overleftarrow{b}\overleftarrow{a}^{10}\overleftarrow{b}^{7}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\lambda he^{10}h^{7}\lambda\mathcal{E}\lambda\mathcal{E}'\lambda eee\overline{\underline{\gamma}\overleftarrow{b}\overleftarrow{a}^{10}\overleftarrow{b}^{7}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\lambda he^{10}h^{7}\lambda\mathcal{E}\lambda\mathcal{E}'\lambda eee\overline{\overleftarrow{h}\underline{\overleftarrow{b}}\overleftarrow{a}^{10}\overleftarrow{b}^{7}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2(\lambda\mathcal{E})^2\lambda he^{10}h^{7}\lambda\mathcal{E}\lambda\mathcal{E}'\lambda eee\overleftarrow{\boldsymbol{h}}\boldsymbol{e}\overline{\underline{\overleftarrow{a}}^{10}\overleftarrow{b}^{7}}\overleftarrow{b}\overleftarrow{a}\overleftarrow{a}^{\omega}$$

In the configuration immediately above the write symbol is positioned to the left of the new encoded current state. Recall that to the left of the simulated tape head the symbol 1 is encoded as he. The $\overleftarrow{h}$ becomes h after execution of the next ETR. The new encoded current state satisfies Definition 3.3.2 and the simulation of transition rule $t_{1,0} = (q_1, 0, 1, R, q_2)$ is complete.

Lemma 3.4.2 *Given a valid configuration of $U_{3,11}$, the encoded current state $\langle q_x \rangle$ and encoded read symbol $\langle \sigma_1 \rangle$ index the ETR $\mathcal{E}(t_{x,\sigma_1})$.*

Proof. $\langle M \rangle$ is a list of ETRs, five ETRs for each state (pair of transition rules) in M. The number of $\overleftarrow{b}$ symbols in the encoded current state $\langle q_x \rangle$ and encoded read symbol, is used to index the next ETR to be executed. If the number of $\overleftarrow{b}$ symbols is l then the $l-1^{\text{th}}$ ETR from the right is indexed. In the encoding, the function $b(\cdot)$ determines the number of $\overleftarrow{b}$ symbols in the next encoded current state. The function $b(\cdot)$ is defined as a summation over $g(\cdot)$ in Equation (3.3.4) for j, where $1 \leqslant j \leqslant x$.

From Equation (3.3.5), for each $j < x$, the function $g(\cdot)$ always has value 5, hence there are at least $5(x-1)$ juxtaposed $\overleftarrow{b}$ symbols in $\langle q_x \rangle$. The state q_x is encoded using five ETRs. When $j = x$, then $g = 0$ or $g = 3$; giving a total number of $\overleftarrow{b}$ symbols that point to the 1^{st} or 4^{th} of these five ETRs respectively.

Any encoded current state $\langle q_x \rangle$, was established by execution of an ETR r. The ETR r encodes move direction D_r and next state q_x. The location of the ETR that is indexed by $\langle q_x \rangle$ is dependent on the move direction D_r of r. When $D_r = L$ and $j = x$ then $g(\cdot) = 3$; when this 3 is added to $5(x-1)$ this indexes the 4^{th} ETR (from the right) of the ETRs for q_x. Using this value of $5(x-1)+3$ we get Cases A and B below. For clarity at this point note that D_r is the move direction of the ETR r that *established* $\langle q_x \rangle$ and $\langle \sigma_1 \rangle$ is the read symbol that is read with $\langle q_x \rangle$ to index the *next* ETR $\mathcal{E}(t_{x,\sigma_1})$.
Case A: $(D_r = L, \sigma_1 = 0)$. $\langle 0 \rangle = \overleftarrow{a}\overleftarrow{a}$ adds no extra $\overleftarrow{b}$ symbols to the number of $\overleftarrow{b}$ symbols provided by $\langle q_x \rangle$, thus the number of $\overleftarrow{b}$ symbols is given by $g(\cdot)$ alone and indexes the 4^{th} ETR (from the right). By Equation (3.2.2) this is $\mathcal{E}(t_{x,0})$.

Case B: $(D_r = L, \sigma_1 = 1)$. $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$ adds one extra $\overleftarrow{b}$ to the number of $\overleftarrow{b}$ symbols provided by $\langle q_x \rangle$, thus indexing the 5^{th} ETR (from the right). By Equation (3.2.2) this is $\mathcal{E}(t_{x,1})$.

When $D_r = R$ and $j = x$ then $g(\cdot) = 0$. Adding this 0 to $5(x-1)$ we get Cases C and D.

Case C: $(D_r = R, \sigma_1 = 1)$. $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$ adds one extra $\overleftarrow{b}$ to the number of $\overleftarrow{b}$ symbols provided by $\langle q_x \rangle$, thus indexing the 2^{nd} ETR (from right). By Equation (3.2.2) this is $\mathcal{E}(t_{x,1})$.

Case D: $(D_r = R, \sigma_1 = 0)$. Case D is a unique case in which $U_{3,11}$ simulates a transition rule t with read symbol 0, immediately after a right moving transition rule $t^{R,x}$ (i.e. $t^{R,x} \vdash t$). In such a case t is encoded as 2 ETRs using $\mathcal{E}$ and $\mathcal{E}'$. The encoded read symbol $\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$ adds no extra $\overleftarrow{b}$ symbols thus indexing the rightmost ETR, which is an ETR′. This ETR′ is given by the function $\mathcal{E}'$ and establishes an *intermediate* encoded current state $\langle q_x \rangle'$ that indexes another ETR that in turn completes the simulation of t. This other ETR is positioned 2 ETRs to the left of the ETR′. Hence in Equation (3.3.2), $t^{R,x}$ is passed to $b(\cdot)$ as a parameter (instead of t) and $\mathcal{E}'$ adds 2 extra $\overleftarrow{b}$ symbols to index the ETR 2 places to the left of the ETR′. By Equation (3.2.2) this is $\mathcal{E}(t_{x,0})$. □

Examples 3.3.2 and 3.4.1 give simulations of right moving transition rules with the later case covering the special case of reading a 0 after a right move. Example 3.4.2 gives the simulation of a left moving transition rule.

Example 3.4.2 ($U_{3,11}$'s simulation of the left moving transition rule $t_{2,1} = (q_2, 1, 1, L, q_3)$ from M_1) We take the last configuration of Example 3.4.1, with $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$ to the right of the encoded current state. Substituting the appropriate ETR from Equation (3.3.6) gives:

$$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\lambda\mathcal{E}'(\lambda\mathcal{E})^4\lambda\mathcal{E}'\lambda eee\overleftarrow{h}\,e\underline{\overleftarrow{a}}^{10}\overleftarrow{b}^{\,7}\boldsymbol{\overleftarrow{b}\,\overleftarrow{a}}\,\overleftarrow{a}^{\omega}$$

We now skip to the end of Cycle 1 (Index next ETR) giving:

$$\boldsymbol{u_2}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\delta\overleftarrow{\mathcal{E}'}(\delta\overleftarrow{\mathcal{E}})^4\delta\overleftarrow{\mathcal{E}'}\delta\overleftarrow{e}\,\overleftarrow{e}\,\overleftarrow{e}\,\overleftarrow{h}\,\overleftarrow{e}\,\underline{\overleftarrow{e}}^{18}\gamma\overleftarrow{a}^{\omega} \qquad \text{(IV)}$$

Notice that the ETR is indexed by neutralising seven λ markers. Note also that the $\langle 1 \rangle$ is read during this process. Next the ETR $eh^{15}ehe$ is printed and we skip to the end of Cycle 2 (Print ETR):

$$\boldsymbol{u_2}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\underline{\lambda}\,\overrightarrow{e}\,\overrightarrow{h}^{15}\overrightarrow{e}\,\overrightarrow{h}\,\overrightarrow{e}\lambda\overrightarrow{\mathcal{E}'}(\lambda\overrightarrow{\mathcal{E}})^4\lambda\overrightarrow{\mathcal{E}'}\lambda\,\overrightarrow{e}\,\overrightarrow{e}\,\overrightarrow{e}\,\overrightarrow{h}\,\gamma\overleftarrow{a}\,\overleftarrow{b}^{\,15}\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

Skipping to the end of Cycle 3 (Restore tape) gives:

$$\boldsymbol{u_2}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\lambda\mathcal{E}'(\lambda\mathcal{E})^4\lambda\mathcal{E}'\lambda eeeh\underline{\gamma}\overleftarrow{a}\,\overleftarrow{b}^{\,15}\overleftarrow{a}\,\boldsymbol{\overleftarrow{b}\,\overleftarrow{a}}\,\overleftarrow{a}^{\omega} \qquad \text{(V)}$$

In configuration (V) above the correct write symbol ($\langle 1\rangle = \overleftarrow{b}\,\overleftarrow{a}$) has been placed to the right of the encoded current state. The new encoded current state satisfies Definition 3.3.2 and the simulation of transition rule $t_{2,1} = (q_2, 1, 1, L, q_3)$ is complete.

Remark 3.4.1 We show how $U_{3,11}$ reads an encoded read symbol *following a left move*. In this case the encoded read symbol is to the left of the encoded current state. Immediately after configuration (V) of Example 3.4.2 we would get:

$$\boldsymbol{u_3}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\lambda\mathcal{E}'(\lambda\mathcal{E})^4\lambda\mathcal{E}'\lambda eee\boldsymbol{\overleftarrow{h}}\,\overline{\underline{\overleftarrow{a}}\,\overleftarrow{b}\,15\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\lambda\mathcal{E}'(\lambda\mathcal{E})^4\lambda\mathcal{E}'\lambda eee\boldsymbol{\underline{\overleftarrow{h}}}\,\overline{\overleftarrow{a}\,\overleftarrow{b}\,15\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

$$\boldsymbol{u_3}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\lambda\mathcal{E}'(\lambda\mathcal{E})^4\lambda\mathcal{E}'\lambda eee\boldsymbol{\underline{h}\,\overleftarrow{a}}\,\overline{\overleftarrow{a}\,\overleftarrow{b}\,15\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega} \qquad \text{(VI)}$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda eh^{15}ehe\lambda\mathcal{E}'(\lambda\mathcal{E})^4\lambda\mathcal{E}'\lambda ee\underline{e}\,\overline{\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,15\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega} \qquad \text{(VII)}$$

In configuration (VII) above the overlined region is extended as the encoded read symbol has been read. $U_{3,11}$ has begun to index the next ETR and is moving to the left to neutralise a λ. The rightmost symbol of the encoded read symbol (which for left moves is always e) was previously overwritten with a γ and this was eventually replaced with a $\overleftarrow{h}$. Only the leftmost symbol of the encoded read symbol must be recorded. If the read symbol was a $\langle 0\rangle = ee$ then $U_{3,11}$'s tape head would have read an e instead of a h in configuration (VI) above, sending $U_{3,11}$'s tape head right instead of left. This would result in one less λ being neutralised. This process records the difference in the encoded read symbols ee and he.

Continuing from configuration (VII) immediately after the next ETR has been indexed, we have the following configuration:

$$\boldsymbol{u_2}, \lambda eh^{15}ehe(\delta\overleftarrow{\mathcal{E}})^3\delta\overleftarrow{\mathcal{E}'}((\delta\overleftarrow{\mathcal{E}})^4\delta\overleftarrow{\mathcal{E}'})^2\delta\overleftarrow{e}\,\overleftarrow{e}\,\overleftarrow{e}\,\overline{\underline{\overleftarrow{e}}^{18}\gamma}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega} \qquad \text{(VIII)}$$

Lemma 3.4.3 *Given a valid configuration with $\langle q_{|Q|}\rangle$ as the encoded current state then $U_{3,11}$ halts.*

Proof. Recall from Section 3.2.1 that for all M the transition rules for the halt state $q_{|Q|}$ are left moving and have $q_{|Q|}$ as the next state. Thus when $\langle q_{|Q|}\rangle$ is the encoded current state $U_{3,11}$ simulates repeated left moves. These left moves continue until the left end of M's simulated tape is reached. When the simulated tape head is attempting to move left at the left end of

the simulated tape then immediately after Cycle 3 the following computation occurs:

$\boldsymbol{u_2}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\underline{\gamma}\overline{\overleftarrow{a}\,\overleftarrow{a}^*\overleftarrow{b}^{\,2}\overleftarrow{b}^*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\overleftarrow{h}\overline{\underline{a}\,\overleftarrow{a}^*\overleftarrow{b}^{\,2}\overleftarrow{b}^*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\underline{\overleftarrow{h}}\overline{\overleftarrow{a}\,\overleftarrow{a}^*\overleftarrow{b}^{\,2}\overleftarrow{b}^*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\underline{\lambda}\overleftarrow{a}\overline{\overleftarrow{a}\,\overleftarrow{a}^*\overleftarrow{b}^{\,2}\overleftarrow{b}^*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\delta\underline{\overleftarrow{a}}\overline{\overleftarrow{a}\,\overleftarrow{a}^*\overleftarrow{b}^{\,2}\overleftarrow{b}^*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$

$\boldsymbol{u_3}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\underline{\delta}\overleftarrow{a}\overline{\overleftarrow{a}\,\overleftarrow{a}^*\overleftarrow{b}^{\,2}\overleftarrow{b}^*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$

There is no transition rule for the state-symbol pair (u_3, δ) in $U_{3,11}$ so the simulation halts. □

Lemma 3.4.4 *Given a valid initial configuration of $U_{3,11}$, then immediately after the first ETR of a computation is indexed, the overlined region is of the form $\overleftarrow{e}^{5|Q|+3}\gamma$.*

Proof. From Definition 3.3.1 the encoded start state $\langle q_1\rangle$ in an initial configuration is of the form $\overleftarrow{a}^{5|Q|}\overleftarrow{b}^{\,2}$. Example 3.3.2 gives the case of reading a $\langle 1\rangle$ in the encoded start state $\langle q_1\rangle$. In this case $\langle q_1\rangle$ and $\langle 1\rangle$ have both been read, that is the $\overleftarrow{b}$ and $\overleftarrow{a}$ symbols of $\langle q_1\rangle$ and $\langle 1\rangle$ have been replaced with $\overleftarrow{e}$ symbols and the rightmost $\overleftarrow{a}$ is replaced with a γ. This gives an overlined region of $\overleftarrow{e}^{5|Q|+3}\gamma$.

The other case is reading a $\langle 0\rangle$ in state $\langle q_1\rangle$. By Definition 3.2.2 the $\langle 0\rangle$ is located immediately to the right of $\langle q_1\rangle$. When reading a $\langle 0\rangle$ on the right there are two steps; executing an ETR′ and then an ETR. From Lemma 3.4.1 we know that the ETR′ $\mathcal{E}'(f, t_{1,0})$ is indexed. Example 3.4.1 executes an ETR′ and indexes the subsequent ETR, while in encoded state $\langle q_1\rangle$. At this point the entire $\langle 0\rangle$ has been read. Thus the $\overleftarrow{b}$ and $\overleftarrow{a}$ symbols of $\langle q_1\rangle$ and $\langle 0\rangle$ have been replaced with $\overleftarrow{e}$ symbols and the rightmost $\overleftarrow{a}$ is replaced with a γ. This gives an overlined region of $\overleftarrow{e}^{5|Q|+3}\gamma$. □

Lemma 3.4.5 *$U_{3,11}$ simulates any transition rule of any deterministic Turing machine M.*

Proof. The proof is by induction on the form of the overlined region. The base case is given by Lemma 3.4.4; after the first ETR is indexed then the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$.

We will show that immediately after any ETR is indexed, the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$.

Assume that the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$ immediately after indexing an ETR ξ_1 in the simulation of timestep i of M's computation. Let ξ_2 be the ETR that is executed immediately after ξ_1. We now show that the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$ immediately after indexing ξ_2 in the simulation of timestep $i+1$.

The four cases of ETRs are defined by Equation (3.3.1). In Examples 3.3.2, 3.4.1 and 3.4.2, three of these cases are shown to execute correctly on an overlined region of the form $\overleftarrow{e}^{5|Q|+3}\gamma$. We use Example 3.4.2 to verify the remaining case (left move, write 0) by substitution of the ETR defined by Case 4 of Equation (3.3.1) with the ETR defined by Case 3 of Equation (3.3.1). The examples generalise to arbitrary transition rules.

Case 1 of Equation (3.3.1)**:** Examples 3.3.2 and 3.4.1 verify Case 1. In configuration (I) (in simulation of timestep i) the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$ and the ETR ξ_1 that is indexed is defined by Case 1 of Equation (3.3.1). In configuration (III) (in simulation of timestep $i+1$) the next ETR ξ_2 has been indexed and the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$.

Case 2 of Equation (3.3.1)**:** Examples 3.4.1 and 3.4.2 verify Case 2. In configuration (III) (in simulation of timestep i) the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$ and the ETR ξ_1 that is indexed is defined by Case 2 of Equation (3.3.1). In configuration (IV) (in simulation of timestep $i+1$) the next ETR ξ_2 has been indexed and the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$.

Case 4 of Equation (3.3.1)**:** Example 3.4.2 and configuration (VIII) verify Case 4. In configuration (IV) (in simulation of timestep i) the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$ and the ETR ξ_1 that is indexed is defined by Case 4 of Equation (3.3.1). In configuration (VIII) (in simulation of timestep $i+1$) the next ETR ξ_2 has been indexed and the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$.

Case 3 of Equation (3.3.1)**:** Case 4 also verifies Case 3 by substitution of the ETR defined by Case 4 of Equation (3.3.1) with the ETR defined by Case 3.

We have shown that the overlined region is $\overleftarrow{e}^{5|Q|+3}\gamma$ immediately after any ETR is indexed. From Examples 3.3.2, 3.4.1 and 3.4.2, each ETR executes on an overlined region of $\overleftarrow{e}^{5|Q|+3}\gamma$ establishing the correct simulated tape head location, encoded write symbol, and an encoded current state that satisfies Definition 3.3.2. By Lemmata 3.4.1 and 3.4.2 the encoded current state indexes the correct ETR. Due to the relative lengths of the encoded current state and overlined region the above mentioned examples generalise to any transition rule of any Turing machine M. □

Theorem 3.4.1 *Let M be a single tape deterministic Turing machine with $|Q|$ states that runs in time t. Then $U_{3,11}$ simulates any Turing machine M in space $O(t + |Q|^2)$ and time $O(|Q|t^2 + |Q|^3 t)$.*

Proof. By Lemma 3.4.5 $U_{3,11}$ simulates any transition rule. Thus given a valid encoding of M's initial configuration (Definition 3.2.2), $U_{3,11}$ simulates the sequence of transition rules in M's computation. From Lemma 3.4.3 when $U_{3,11}$ simulates the halting state of M, $U_{3,11}$'s tape head returns to the left end of M's encoded output and halts. The encoded output is easily decoded via Definition 3.2.1.

(Space). At time t the space used by M is bounded by t. Simulator $U_{3,11}$ uses space $O(t + |Q|^2)$, where $O(|Q|^2)$ space is used to store M as the word $\langle M \rangle$ and $O(t)$ space is used to store M's encoded tape after t simulated steps.

(Time). Simulating a transition rule involves 4 cycles. (1) Index an ETR by neutralising $O(|Q|)$ of the λ markers: $O(|Q|t + |Q|^3)$ steps. (2) Copy an ETR of length $O(|Q|)$ from $\langle M \rangle$ to the encoded current state location: $O(|Q|t+|Q|^3)$ steps. (3) Restore $U_{3,11}$'s tape contents: $O(t+|Q|^2)$ steps. (4) Complete execution of ETR: a small constant number of steps. Thus $U_{3,11}$ uses $O(|Q|t+|Q|^3)$ time to simulate a single step of M, and $O(|Q|t^2+|Q|^3 t)$ time to simulate the entire computation of M. □

3.5 Polynomial time Curve

In this section we further extend our result by finding small polynomial time universal Turing machines with other state-symbol pairs. Thus we establish a $O(t^2)$ polynomial time curve.

All universal Turing machines in this chapter use the same basic algorithm as $U_{3,11}$. The proof of correctness given for $U_{3,11}$ can be applied to the remaining machines in a straightforward way, so we do not restate it. The encoding of the input and operation of these universal Turing machines is the same as $U_{3,11}$ unless noted otherwise. Each universal Turing machine makes use of specially tailored $\mathcal{E}$ and $\mathcal{E}'$ functions.

3.5.1 Construction of $U_{6,6}$

For $U_{6,6}$ the start state of $\langle M \rangle$ is encoded as $\langle q_1 \rangle = \overleftarrow{a}^{5|Q|}\overleftarrow{b}^{2}$. The encoding of the current state is of the form $\overleftarrow{a}^{*}\overleftarrow{b}^{2}\overleftarrow{b}^{*}\{\overleftarrow{a} \cup \epsilon\}$ and is of length $5|Q|+2$.

Let $t = (q_x, \sigma_1, \sigma_2, D, q_y)$ be a fixed transition rule in M, then t is encoded via $\mathcal{P}$ using the function $\mathcal{E}$ on its own, or in conjunction with $\mathcal{E}'$,

where

$$\mathcal{E}(t) = \begin{cases} \overleftarrow{b}\, b(t) \overleftarrow{a}\, a(t)-3 \overleftarrow{b} & \text{if } D = R, \sigma_2 = 0 \\ \overleftarrow{b}\, b(t) \overleftarrow{a}\, a(t)+1 & \text{if } D = R, \sigma_2 = 1 \\ \overleftarrow{a}\, \overleftarrow{a}\, \overleftarrow{a}\, \overleftarrow{b}\, b(t) \overleftarrow{a}\, a(t)-2 \overleftarrow{b} & \text{if } D = L, \sigma_2 = 0 \\ \overleftarrow{a}\, \overleftarrow{b}\, \overleftarrow{a}\, \overleftarrow{b}\, b(t) \overleftarrow{a}\, a(t)-2 \overleftarrow{b} & \text{if } D = L, \sigma_2 = 1 \end{cases} \tag{3.5.1}$$

and

$$\mathcal{E}'(f,t) = \begin{cases} \overleftarrow{b}\, b(t^{R,x})+2 \overleftarrow{a}\, a(t^{R,x})-5 \overleftarrow{b} & \text{if } \exists t^{R,x}, q_x \neq q_1 \\ \epsilon & \text{if } \nexists t^{R,x}, q_x \neq q_1 \\ \overleftarrow{b}\, 4 \overleftarrow{a}\, 5|Q|-5 \overleftarrow{b} & \text{if } q_x = q_1 \end{cases} \tag{3.5.2}$$

where as before $t^{R,x}$ is any right moving transition rule such that $t^{R,x} \vdash t$.

The value of the ending E, from Equation (3.2.1), for $U_{6,6}$ is $E = \overleftarrow{a}$.

Example 3.5.1 (Encoding of Turing machine M_2 for $U_{6,6}$) Let Turing machine $M_2 = (\{q_1, q_2\}, \{0,1\}, 0, f, q_1, \{q_2\})$ where the function f is defined by $(q_1, 0, 0, R, q_1)$, $(q_1, 1, 1, R, q_2)$, $(q_2, 0, 0, L, q_2)$ and $(q_2, 1, 1, L, q_2)$. M_2 is encoded as: $\langle M_2 \rangle = \lambda\mathcal{P}(f, q_2)\lambda\mathcal{P}(f, q_1)\lambda E$. Substituting the appropriate values from Equation (3.2.2) gives

$$\begin{aligned} \langle M_2 \rangle = & \lambda\mathcal{E}(t_{2,1})\lambda\mathcal{E}(t_{2,0})\lambda\mathcal{E}(t_{2,0})\lambda\mathcal{E}(t_{2,1})\lambda\mathcal{E}'(f, t_{2,0}) \\ & \lambda\mathcal{E}(t_{1,1})\lambda\mathcal{E}(t_{1,0})\lambda\mathcal{E}(t_{1,0})\lambda\mathcal{E}(t_{1,1})\lambda\mathcal{E}'(f, t_{1,0})\lambda E \end{aligned}$$

Rewriting this using Equations (3.5.1) and (3.5.2) gives

$$\begin{aligned} \langle M_2 \rangle = & \lambda \overleftarrow{a}\, \overleftarrow{b}\, \overleftarrow{a}\, \overleftarrow{b}^{11} \lambda \overleftarrow{a}\, \overleftarrow{a}\, \overleftarrow{a}\, \overleftarrow{b}^{11} \lambda \overleftarrow{a}\, \overleftarrow{a}\, \overleftarrow{a}\, \overleftarrow{b}^{11} \lambda \overleftarrow{a}\, \overleftarrow{b}\, \overleftarrow{a}\, \overleftarrow{b}^{11} \\ & \lambda \overleftarrow{b}^{10} \lambda \overleftarrow{b}^{7} \overleftarrow{a}^{6} \lambda \overleftarrow{b}^{2} \overleftarrow{a}^{7} \overleftarrow{b}\, \lambda \overleftarrow{b}^{2} \overleftarrow{a}^{7} \overleftarrow{b}\, \lambda \overleftarrow{b}^{7} \overleftarrow{a}^{6} \lambda \overleftarrow{b}^{4} \overleftarrow{a}^{5} \overleftarrow{b}\, \lambda \overleftarrow{a} \end{aligned} \tag{3.5.3}$$

Definition 3.5.1 ($U_{6,6}$) *Let Turing machine* $U_{6,6} = (\{u_1, u_2, u_3, u_4, u_5, u_6\}, \{\overleftarrow{a}, \overleftarrow{b}, \overrightarrow{a}, \overrightarrow{b}, \lambda, \delta\}, \overleftarrow{a}, f, u_1, \{u_3, u_5, u_6\})$ *where* f *is given by Table 3.5.7.*

	u_1	u_2	u_3	u_4	u_5	u_6
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$	λ, L, u_4	$\overrightarrow{a}, L, u_3$	$\overrightarrow{a}, L, u_4$	$\overleftarrow{a}, L, u_1$	
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$	$\overleftarrow{b}, L, u_3$	$\overrightarrow{b}, L, u_3$	$\overrightarrow{b}, L, u_4$	$\overleftarrow{a}, L, u_3$	
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_1$	$\overleftarrow{a}, R, u_2$		$\overleftarrow{a}, R, u_5$	$\overrightarrow{a}, R, u_2$	$\overleftarrow{a}, R, u_6$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$	$\overleftarrow{b}, R, u_2$	$\overleftarrow{a}, L, u_5$	$\overleftarrow{b}, R, u_5$	$\overrightarrow{b}, R, u_1$	$\overleftarrow{b}, R, u_6$
λ	$\overleftarrow{b}, L, u_2$	$\overleftarrow{a}, L, u_2$	δ, R, u_1	λ, R, u_5		$\overrightarrow{b}, R, u_5$
δ	δ, R, u_1	δ, R, u_2	δ, L, u_3	δ, L, u_4	λ, R, u_6	λ, R, u_6

Table 3.5.7: Table of behaviour for $U_{6,6}$.

Remark 3.5.1 There are some minor differences between the operation of $U_{6,6}$ and $U_{3,11}$. The order of symbols in ETRs of $U_{6,6}$ is reversed when compared with ETRs of $U_{3,11}$, assuming $\overleftarrow{a} = e$ and $\overleftarrow{b} = h$. To see this, note the difference between Equations (3.3.1) and (3.5.1). When printing an ETR, $U_{6,6}$ reverses the order so that encoded current states are of the same form as those in $U_{3,11}$. Also M's encoded tape symbols to the left and right of the simulated tape head use the *same* encodings ($\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$ and $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$). This is not the case for $U_{3,11}$.

We give an example of $U_{6,6}$ simulating a transition rule of M_2 from Example 3.5.1. As usual the example is separated into 4 cycles.

Example 3.5.2 ($U_{6,6}$**'s simulation of the right moving transition rule** $t_{1,1} = (q_1, 1, 1, R, q_2)$ **from Turing machine** M_2**)** The start state of $U_{6,6}$ is u_1 and the tape head of $U_{6,6}$ is over the leftmost symbol of $\langle q_1 \rangle$ (as in Definition 3.2.2). The input to M_2 is 11 (encoded via $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$). Thus the initial configuration is:

$$\boldsymbol{u_1}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^2\lambda\overleftarrow{a}\,\underline{\overleftarrow{a}}^{10}\overleftarrow{b}^{\,2}\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$$

Cycle 1 (Index next ETR)

In Cycle 1 the left table (see Table 3.5.8) reads the encoded current state. The right table neutralises λ markers to index the next ETR. The neutralisation is done in the usual way; each $\overleftarrow{b}$ in the encoded current state causes a λ to be replaced with a δ. The middle table decides when the cycle is complete. In state u_1 the tape head scans from left to right; each $\overleftarrow{b}$ in the encoded current state is replaced with an $\overleftarrow{a}$ and $U_{6,6}$ then enters state u_3 via u_2.

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$
δ	δ, R, u_1

	u_2
$\overleftarrow{a}$	λ, L, u_4
$\overleftarrow{b}$	$\overleftarrow{b}, L, u_3$

	u_3
$\overleftarrow{a}$	$\overrightarrow{a}, L, u_3$
$\overleftarrow{b}$	$\overrightarrow{b}, L, u_3$
λ	δ, R, u_1
δ	δ, L, u_3

Table 3.5.8: Sub-tables for Cycle 1 of $U_{6,6}$.

We have replaced the shorthand notation $\mathcal{E}$ with the word $\overleftarrow{b}^7\overleftarrow{a}^6$ defined by $\mathcal{E}(t_{1,1})$. The word $\overleftarrow{b}^7\overleftarrow{a}^6$ appears in the location defined by Equation (3.5.3). After the initial configuration we have:

$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\overleftarrow{a}^{10}\underline{\overleftarrow{b}}\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_2}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\overleftarrow{a}^{10}\overleftarrow{a}\,\underline{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_3}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\overleftarrow{a}^{10}\underline{\overleftarrow{a}}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_3}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\underline{\lambda}\,\overrightarrow{a}\overrightarrow{a}^{10}\overrightarrow{a}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\delta\underline{\overrightarrow{a}}\,\overrightarrow{a}^{10}\overrightarrow{a}\,\overleftarrow{b}\,\boldsymbol{\overleftarrow{b}}\,\boldsymbol{\overleftarrow{a}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

The neutralisation process continues until $U_{6,6}$ reads the final $\overleftarrow{b}$, moves right to test for the end of the encoded current state and read symbol in u_2, and then reads an $\overleftarrow{a}$. When this occurs $U_{6,6}$ is finished reading the encoded current state and read symbol. Skipping to the end of this cycle gives:

$\boldsymbol{u_4}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\delta\mathcal{E}'\delta\overleftarrow{a}\,\overleftarrow{a}^{10}\overleftarrow{a}\,\overleftarrow{a}\,\underline{\overleftarrow{a}}\,\lambda\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^{\omega}$

$U_{6,6}$ has neutralised two λ markers to index the next ETR.

	u_4
$\overleftarrow{a}$	$\overrightarrow{a}, L, u_4$
$\overleftarrow{b}$	$\overrightarrow{b}, L, u_4$
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_5$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_5$
λ	λ, R, u_5
δ	δ, L, u_4

	u_5
$\overrightarrow{a}$	$\overrightarrow{a}, R, u_2$
$\overrightarrow{b}$	$\overrightarrow{b}, R, u_1$
δ	λ, R, u_6

	u_1
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$
λ	$\overleftarrow{b}, L, u_2$
δ	δ, R, u_1

	u_2
$\overleftarrow{a}$	λ, L, u_4
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_2$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_2$
λ	$\overleftarrow{a}, L, u_2$
δ	δ, R, u_2

Table 3.5.9: Sub-tables for Cycle 2 of $U_{6,6}$.

Cycle 2 (Print ETR)

This cycle copies an ETR to M's simulated tape head position. The leftmost table scans left and locates the next symbol of the ETR to be printed. The second table from the left records the symbol to be printed or ends the cycle. The rightmost two tables scan right and print the appropriate symbol. In the configurations below, $U_{6,6}$ scans left until a λ is read. Then $U_{6,6}$ moves right and records the symbol read by entering state u_1 or u_2. In the configurations below when all the $\overleftarrow{b}$ and $\overleftarrow{a}$ symbols in an $\mathcal{E}'$ are replaced with $\overrightarrow{b}$ and $\overrightarrow{a}$ symbols then the resulting word is denoted $\overrightarrow{\mathcal{E}'}$.

$\boldsymbol{u_4}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\underline{\lambda}\,\overrightarrow{b}\,\overrightarrow{b}\,\overrightarrow{b}^5\overrightarrow{a}^6\delta\overrightarrow{\mathcal{E}'}\delta\overrightarrow{a}\,\overrightarrow{\overrightarrow{a}^{12}\overrightarrow{a}\lambda\overleftarrow{b}}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_5}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\underline{\overrightarrow{b}}\,\overrightarrow{b}\,\overrightarrow{b}^5\overrightarrow{a}^6\delta\overrightarrow{\mathcal{E}'}\delta\overrightarrow{a}\,\overrightarrow{\overrightarrow{a}^{12}\overrightarrow{a}\lambda\overleftarrow{b}}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\,\overrightarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}^5\overleftarrow{a}^6\delta\mathcal{E}'\delta\overleftarrow{a}\,\overleftarrow{\overleftarrow{a}^{12}\overleftarrow{a}\underline{\lambda}\overleftarrow{b}}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_2}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\,\overrightarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}^5\overleftarrow{a}^6\delta\mathcal{E}'\delta\overleftarrow{a}\,\overleftarrow{\overleftarrow{a}^{12}\underline{\overleftarrow{a}}\,\overleftarrow{b}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_4}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\underline{\overrightarrow{b}}\,\overrightarrow{b}\,\overrightarrow{b}^5\overrightarrow{a}^6\delta\overrightarrow{\mathcal{E}'}\delta\overrightarrow{a}\,\overrightarrow{\overrightarrow{a}^{12}\lambda\overleftarrow{b}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_5}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\,\overleftarrow{b}\,\underline{\overrightarrow{b}}\,\overrightarrow{b}^5\overrightarrow{a}^6\delta\overrightarrow{\mathcal{E}'}\delta\overrightarrow{a}\,\overrightarrow{\overrightarrow{a}^{12}\lambda\overleftarrow{b}}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

On the first pass $U_{6,6}$ located the symbol to be printed by using λ as a marker. On subsequent passes $U_{6,6}$ locates the symbol to be printed by locating an $\overrightarrow{a}$ or $\overrightarrow{b}$. This printing process is iterated until $U_{6,6}$ is finished printing the ETR. The completion of this process occurs on reading a δ in state u_5 which switches control to u_6:

$\boldsymbol{u_4}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^5\underline{\overrightarrow{a}}\delta\overrightarrow{\mathcal{E}'}\delta\overrightarrow{a}\,\overrightarrow{\lambda\overleftarrow{a}^6\overleftarrow{b}^7}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_5}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^5\overleftarrow{a}\,\underline{\delta}\overrightarrow{\mathcal{E}'}\delta\overrightarrow{a}\,\overrightarrow{\lambda\overleftarrow{a}^6\overleftarrow{b}^7}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_6}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^5\overleftarrow{a}\,\lambda\underline{\overrightarrow{\mathcal{E}'}}\delta\overrightarrow{a}\,\overrightarrow{\lambda\overleftarrow{a}^6\overleftarrow{b}^7}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

Cycle 3 (Restore tape)

This table restores M's simulated tape and encoded table of behaviour. $U_{6,6}$ moves right restoring each $\overrightarrow{a}$ to $\overleftarrow{a}$, each $\overrightarrow{b}$ to $\overleftarrow{b}$, and each δ to λ. This continues until $U_{6,6}$ reads λ, sending control to state u_5.

$\boldsymbol{u_6}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\,\overleftarrow{\underline{\lambda}\overleftarrow{a}\,\overleftarrow{a}^5\overleftarrow{b}^7}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

$\boldsymbol{u_5}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\,\overrightarrow{b}\,\overleftarrow{\underline{a}\,\overleftarrow{a}^5\overleftarrow{b}^7}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$

	u_6
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_6$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_6$
λ	$\overrightarrow{b}, R, u_5$
δ	λ, R, u_6

Table 3.5.10: Cycle 3 of $U_{6,6}$.

Cycle 4 (Choose read or write symbol)

	u_5
$\overleftarrow{a}$	$\overleftarrow{a}, L, u_1$
$\overleftarrow{b}$	$\overleftarrow{a}, L, u_3$
λ	

	u_3
$\overrightarrow{b}$	$\overleftarrow{a}, L, u_5$

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$

Table 3.5.11: Sub-tables for Cycle 4 of $U_{6,6}$.

This cycle either (i) begins the indexing of an ETR or (ii) completes the execution of an ETR. More precisely: (i) if $U_{6,6}$ is immediately after simulating a left move then this cycle reads the encoded read symbol to the left of the encoded current state, (ii) if $U_{6,6}$ is simulating a right move then this cycle prints the encoded write symbol to the left of the encoded current state. Case (ii) follows:

$$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\,\overline{\overrightarrow{b}\overleftarrow{a}\,\overleftarrow{a}^5\overleftarrow{b}^7\overleftarrow{b}}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$$

$$\boldsymbol{u_1}, (\lambda\mathcal{E})^4\lambda\mathcal{E}'(\lambda\mathcal{E})^3\lambda\overleftarrow{b}^7\overleftarrow{a}^6\lambda\mathcal{E}'\lambda\overleftarrow{a}\,\overleftarrow{\boldsymbol{b}}\,\underline{\overleftarrow{\boldsymbol{a}}}\,\overline{\overleftarrow{a}^5\overleftarrow{b}^7\overleftarrow{b}}\,\overleftarrow{a}\,\overleftarrow{a}^\omega$$

In the configuration immediately above we have shortened the overlined section; the two symbols to the left of $\langle M_2\rangle$'s encoded current state encode the write symbol 1.

The example simulation of transition rule $t_{1,1} = (q_1, 1, 1, R, q_2)$ is now complete. The correct encoded write symbol $\langle 1\rangle = \overleftarrow{b}\,\overleftarrow{a}$ has been written and the new encoded current state is of the correct form. M_2's simulated tape head (the new encoded current state) is configured so $U_{6,6}$ reads the next encoded read symbol to the right when searching for the next ETR.

Left moving transition rules are simulated in a similar fashion to the right moving transition rule given above, except in this case the write symbol is written on the right hand side of the encoded current state as shown in Figure 3.2.3 (cL). After the left move M_2's simulated tape head (encoded current state) is configured to read the encoded tape symbol to its left when searching for the next ETR.

The halting case for $U_{6,6}$ is similar to the halting case for $U_{3,11}$. When $U_{6,6}$ encounters the state symbol pair (u_5, λ), for which there is no transition rule, the computation halts. This occurs during Cycle 4 when $U_{6,6}$ attempts to simulate a left move at the left end of the simulated tape.

3.5.2 Construction of $U_{5,7}$

For $U_{5,7}$ the start state of $\langle M \rangle$ is encoded as $\langle q_1 \rangle = \overleftarrow{a}^{5|Q|}\overleftarrow{b}^{4}$. The encoding of M's current state is of the form $\overleftarrow{a}^{*}\overleftarrow{b}^{4}\overleftarrow{b}^{*}\{\overleftarrow{a} \cup \epsilon\}$ and is of length $5|Q|+4$.

Let $t = (q_x, \sigma_1, \sigma_2, D, q_y)$ be a fixed transition rule in M, then t is encoded via $\mathcal{P}$ using the function $\mathcal{E}$ on its own, or in conjunction with $\mathcal{E}'$, where

$$\mathcal{E}(t) = \begin{cases} \overleftarrow{b}^{b(t)+2}\overleftarrow{a}^{a(t)+1} & \text{if } D = R, \sigma_2 = 0 \\ \overleftarrow{b}^{b(t)+2}\overleftarrow{a}^{a(t)}\overleftarrow{b} & \text{if } D = R, \sigma_2 = 1 \\ \overleftarrow{a}\overleftarrow{a}\overleftarrow{a}\overleftarrow{b}^{b(t)+2}\overleftarrow{a}^{a(t)-1} & \text{if } D = L, \sigma_2 = 0 \\ \overleftarrow{a}\overleftarrow{b}\overleftarrow{a}\overleftarrow{b}^{b(t)+2}\overleftarrow{a}^{a(t)-1} & \text{if } D = L, \sigma_2 = 1 \end{cases} \tag{3.5.4}$$

and

$$\mathcal{E}'(f,t) = \begin{cases} \overleftarrow{b}^{b(t^{R,x})+4}\overleftarrow{a}^{a(t^{R,x})-2} & \text{if } \exists t^{R,x}, q_x \neq q_1 \\ \epsilon & \text{if } \nexists t^{R,x}, q_x \neq q_1 \\ \overleftarrow{b}^{6}\overleftarrow{a}^{5|Q|-2} & \text{if } q_x = q_1 \end{cases} \tag{3.5.5}$$

where as before $t^{R,x}$ is any right moving transition rule such that $t^{R,x} \vdash t$.

The value of the ending E, from Equation (3.2.1), for $U_{5,7}$ is $E = \lambda\overleftarrow{b}\lambda\overleftarrow{a}$.

Definition 3.5.2 ($U_{5,7}$) *Let Turing machine* $U_{5,7} = (\{u_1, u_2, u_3, u_4, u_5\}, \{\overleftarrow{a}, \overleftarrow{b}, \overrightarrow{a}, \overrightarrow{b}, \lambda, \overleftarrow{\lambda}, \overrightarrow{\lambda}\}, \overleftarrow{a}, f, u_1, \{u_4, u_5\})$ *where* f *is given by the following table.*

	u_1	u_2	u_3	u_4	u_5
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$	λ, L, u_4	$\overrightarrow{a}, L, u_3$	$\overrightarrow{a}, L, u_4$	$\overleftarrow{a}, L, u_5$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$	$\overleftarrow{b}, L, u_3$	$\overrightarrow{b}, L, u_3$	$\overrightarrow{b}, L, u_4$	$\overleftarrow{a}, R, u_1$
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_1$	$\overleftarrow{a}, R, u_2$	$\overrightarrow{a}, R, u_1$	$\overleftarrow{a}, R, u_3$	$\overleftarrow{a}, R, u_5$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$	$\overleftarrow{b}, R, u_2$	$\overrightarrow{b}, R, u_2$	$\overleftarrow{b}, R, u_3$	$\overleftarrow{b}, R, u_5$
λ	$\overleftarrow{a}, L, u_2$	$\overleftarrow{b}, L, u_2$	$\overleftarrow{\lambda}, R, u_1$	λ, R, u_3	$\overleftarrow{\lambda}, L, u_1$
$\overleftarrow{\lambda}$	$\overleftarrow{b}, R, u_5$	$\overleftarrow{a}, L, u_3$	$\overrightarrow{\lambda}, L, u_3$	$\overrightarrow{\lambda}, L, u_4$	
$\overrightarrow{\lambda}$	$\overleftarrow{\lambda}, R, u_1$	$\overleftarrow{\lambda}, R, u_2$	λ, R, u_5		λ, R, u_5

Table 3.5.12: Table of behaviour for $U_{5,7}$.

Remark 3.5.2 There are some minor differences between the operation of $U_{5,7}$ and $U_{3,11}$. The order of symbols in ETRs of $U_{5,7}$ is reversed when compared with ETRs of $U_{3,11}$, assuming $\overleftarrow{a} = e$ and $\overleftarrow{b} = h$. To see this, note the difference between Equations (3.3.1) and (3.5.4). When printing an ETR, $U_{5,7}$ reverses the order so that encoded current states are of the same form as those in $U_{3,11}$. Also M's encoded tape symbols to the left and right of the simulated tape head use the *same* encodings ($\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$ and $\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$). This is not the case for $U_{3,11}$.

We give a brief overview of the computation of $U_{5,7}$.

Cycle 1 (Index next ETR)

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$
$\overrightarrow{\lambda}$	$\overleftarrow{\lambda}, R, u_1$

	u_2
$\overleftarrow{a}$	λ, L, u_4
$\overleftarrow{b}$	$\overleftarrow{b}, L, u_3$

	u_3
$\overleftarrow{a}$	$\overrightarrow{a}, L, u_3$
$\overleftarrow{b}$	$\overrightarrow{b}, L, u_3$
λ	$\overleftarrow{\lambda}, R, u_1$
$\overleftarrow{\lambda}$	$\overrightarrow{\lambda}, L, u_3$

Table 3.5.13: Sub-tables for Cycle 1 of $U_{5,7}$.

In Cycle 1 the leftmost table (above) reads the encoded current state. The rightmost table neutralises λ markers by replacing them with $\overleftarrow{\lambda}$ or $\overrightarrow{\lambda}$ to index the next ETR. The middle table decides when the cycle is complete. Each $\overleftarrow{b}$ in the encoded current state is replaced with $\overleftarrow{a}$ and then $U_{5,7}$ enters state u_3 via u_2.

Cycle 2 (Print ETR)

	u_4
$\overleftarrow{a}$	$\overrightarrow{a}, L, u_4$
$\overleftarrow{b}$	$\overrightarrow{b}, L, u_4$
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_3$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_3$
λ	λ, R, u_3
$\overleftarrow{\lambda}$	$\overrightarrow{\lambda}, L, u_4$

	u_3
$\overrightarrow{a}$	$\overrightarrow{a}, R, u_1$
$\overrightarrow{b}$	$\overrightarrow{b}, R, u_2$
$\overrightarrow{\lambda}$	λ, R, u_5

	u_2
$\overleftarrow{a}$	λ, L, u_4
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_2$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_2$
λ	$\overleftarrow{b}, L, u_2$
$\overrightarrow{\lambda}$	$\overleftarrow{\lambda}, R, u_2$

	u_1
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_1$
λ	$\overleftarrow{a}, L, u_2$
$\overrightarrow{\lambda}$	$\overleftarrow{\lambda}, R, u_1$

Table 3.5.14: Sub-tables for Cycle 2 of $U_{5,7}$.

This cycle copies an ETR to M's simulated tape head position. The leftmost table scans left and locates the next symbol of the ETR to be printed. The second table from the left records the symbol to be printed or ends the cycle. The rightmost two tables scan right and print the appropriate symbol.

Cycle 3 (Restore tape)

	u_5
$\overrightarrow{a}$	$\overleftarrow{a}, R, u_5$
$\overrightarrow{b}$	$\overleftarrow{b}, R, u_5$
λ	$\overleftarrow{\lambda}, L, u_1$
$\overrightarrow{\lambda}$	λ, R, u_5

Table 3.5.15: Cycle 3 of $U_{5,7}$.

Table 3.5.15 restores M's simulated tape and encoded table of behaviour. $U_{5,7}$ moves right restoring each $\overrightarrow{a}$ to $\overleftarrow{a}$, each $\overrightarrow{b}$ to $\overleftarrow{b}$, and each $\overrightarrow{\lambda}$ to λ. This continues until $U_{5,7}$ reads λ, sending $U_{5,7}$'s control into u_1.

Cycle 4 (Choose read or write symbol)

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$
$\overleftarrow{\lambda}$	$\overleftarrow{b}, R, u_5$

	u_2
$\overleftarrow{\lambda}$	$\overleftarrow{a}, L, u_3$

	u_5
$\overleftarrow{a}$	$\overleftarrow{a}, L, u_5$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_1$

Table 3.5.16: Sub-tables for Cycle 4 of $U_{5,7}$.

This cycle either (i) begins the indexing of an ETR or (ii) completes the execution of an ETR. More precisely: (i) if $U_{5,7}$ is immediately after simulating a left move then this cycle reads the encoded read symbol to the left of the encoded current state, (ii) if $U_{5,7}$ is simulating a right move then this cycle prints the encoded write symbol to the left of the encoded current state. The halting case for $U_{5,7}$ is more complex than the previous universal

Turing machines. If the simulated tape head is attempting to move left at the left end of the simulated tape then at the end of Cycle 3 $U_{5,7}$ has the following configuration:

$$\boldsymbol{u_5}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\lambda\overleftarrow{b}\,\lambda\underline{\lambda}\overline{\overleftarrow{a}\,\overleftarrow{a}*\overleftarrow{b}^{\,4}\overleftarrow{b}*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^{\omega}$$

The computation continues through 13 configurations before the halting configuration given below is reached.

$$\boldsymbol{u_5}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\lambda\overleftarrow{a}\,\overleftarrow{b}\,\underline{\overleftarrow{\lambda}}\overline{\overleftarrow{a}\,\overleftarrow{a}*\overleftarrow{b}^{\,4}\overleftarrow{b}*\overleftarrow{a}}(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^{\omega}$$

There is no transition rule for the state-symbol pair $(u_5, \overleftarrow{\lambda})$ in $U_{5,7}$ so the simulation halts.

3.5.3 Construction of $U_{7,5}$

For $U_{7,5}$ the start state of $\langle M\rangle$ is encoded as $\langle q_1\rangle = \overleftarrow{a}^{5|Q|+1}\overleftarrow{b}^{\,3}$. The encoding of M's current state is of the form $\overleftarrow{a}^*\overleftarrow{b}^{\,3}\overleftarrow{b}^*\{\overleftarrow{a}\cup\epsilon\}$ and is of length $5|Q|+4$.

Let $t = (q_x, \sigma_1, \sigma_2, D, q_y)$ be a fixed transition rule in M, then t is encoded via $\mathcal{P}$ using the function $\mathcal{E}$ on its own, or in conjunction with $\mathcal{E}'$, where

$$\mathcal{E}(t) = \begin{cases} \overleftarrow{b}^{\,b(t)+1}(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)+1}\overleftarrow{b} & \text{if } D = R, \sigma_2 = 0 \\ \overleftarrow{b}^{\,b(t)+1}(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)-1}\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{b} & \text{if } D = R, \sigma_2 = 1 \\ (\overleftarrow{a}\,\overleftarrow{b}\,)^3\overleftarrow{b}^{\,b(t)+1}(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)-1}\overleftarrow{b} & \text{if } D = L, \sigma_2 = 0 \\ \overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{b}^{\,b(t)+1}(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)-1}\overleftarrow{b} & \text{if } D = L, \sigma_2 = 1 \end{cases} \tag{3.5.6}$$

and

$$\mathcal{E}'(f,t) = \begin{cases} \overleftarrow{b}^{\,b(t^{R,x})+3}(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t^{R,x})-2}\overleftarrow{b} & \text{if } \exists t^{R,x}, q_x \neq q_1 \\ \epsilon & \text{if } \nexists t^{R,x}, q_x \neq q_1 \\ \overleftarrow{b}^{\,5}(\overleftarrow{a}\,\overleftarrow{b}\,)^{5|Q|-2}\overleftarrow{b} & \text{if } q_x = q_1 \end{cases} \tag{3.5.7}$$

where as before $t^{R,x}$ is any right moving transition rule such that $t^{R,x} \vdash t$.

The value of the ending E, from Equation (3.2.1), for $U_{7,5}$ is $E = \overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,\lambda\overleftarrow{a}$.

Definition 3.5.3 ($U_{7,5}$) *Let Turing machine* $U_{7,5} = (\{u_1, u_2, u_3, u_4, u_5, u_6, u_7\}, \{\overleftarrow{a}, \overleftarrow{b}, \lambda, \delta, \gamma\}, \overleftarrow{a}, f, u_1, \{u_2, u_5\})$ *where the function* f *is given by Table 3.5.17.*

	u_1	u_2	u_3	u_4
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$	γ, L, u_4	$\overleftarrow{a}, L, u_3$	$\overleftarrow{a}, L, u_4$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$	$\overleftarrow{b}, L, u_3$	λ, L, u_3	λ, L, u_4
λ	$\overleftarrow{b}, R, u_1$	γ, R, u_1	δ, R, u_1	λ, R, u_5
δ	δ, R, u_1		δ, L, u_3	δ, L, u_4
γ	$\overleftarrow{a}, L, u_2$	$\overleftarrow{b}, R, u_6$	$\overleftarrow{a}, R, u_5$	$\overleftarrow{b}, R, u_5$

	u_5	u_6	u_7
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_2$	$\overleftarrow{a}, R, u_6$	$\overleftarrow{a}, R, u_7$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_3$	$\overleftarrow{a}, L, u_7$	$\overleftarrow{a}, R, u_1$
λ	γ, R, u_6	$\overleftarrow{b}, R, u_6$	$\overleftarrow{b}, R, u_7$
δ	λ, R, u_7	δ, R, u_6	λ, R, u_7
γ		$\overleftarrow{b}, L, u_2$	γ, L, u_5

Table 3.5.17: Table of behaviour for $U_{7,5}$.

Remark 3.5.3 There are some minor differences between the operation of $U_{7,5}$ and $U_{3,11}$. The order of symbols in ETRs of $U_{7,5}$ is reversed when compared with ETRs of $U_{3,11}$, assuming $\overleftarrow{a}\,\overleftarrow{b} = e$ and $\overleftarrow{b} = h$. To see this, note the difference between Equations (3.3.1) and (3.5.6). When printing an ETR, $U_{7,5}$ reverses the order so that encoded current states are of the same form as those in $U_{3,11}$. Also, M's encoded tape symbols to the left and right of the simulated tape head use the *same* encodings ($\langle 0\rangle = \overleftarrow{a}\,\overleftarrow{a}$ and $\langle 1\rangle = \overleftarrow{b}\,\overleftarrow{a}$). This is not the case for $U_{3,11}$.

We give a brief overview of $U_{7,5}$'s computation.

Cycle 1 (Index next ETR)

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$
λ	$\overleftarrow{b}, R, u_1$
δ	δ, R, u_1

	u_2
$\overleftarrow{a}$	γ, L, u_4
$\overleftarrow{b}$	$\overleftarrow{b}, L, u_3$

	u_3
$\overleftarrow{a}$	$\overleftarrow{a}, L, u_3$
$\overleftarrow{b}$	λ, L, u_3
λ	δ, R, u_1
δ	δ, L, u_3

Table 3.5.18: Sub-tables for Cycle 1 of $U_{7,5}$.

In Cycle 1 the left table (above) reads the encoded current state. The right table neutralises λ markers by replacing them with δ symbols to index the next ETR. The middle table decides when the cycle is complete. Each $\overleftarrow{b}$ in the encoded current state is replaced with $\overleftarrow{a}$ and $U_{7,5}$ then enters state u_3 via u_2.

Cycle 2 (Print ETR)

	u_2	u_4
$\overleftarrow{a}$	γ, L, u_4	$\overleftarrow{a}, L, u_4$
$\overleftarrow{b}$		λ, L, u_4
λ		λ, R, u_5
δ		δ, L, u_4
γ		$\overleftarrow{b}, R, u_5$

	u_5
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_2$
λ	γ, R, u_6
δ	λ, R, u_7

	u_6
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_6$
λ	$\overleftarrow{b}, R, u_6$
δ	δ, R, u_6
γ	$\overleftarrow{b}, L, u_2$

	u_1	u_2
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$	
λ	$\overleftarrow{b}, R, u_1$	γ, R, u_1
δ	δ, R, u_1	
γ	$\overleftarrow{a}, L, u_2$	

Table 3.5.19: Sub-tables for Cycle 2 of $U_{7,5}$.

This cycle copies an ETR to M's simulated tape head position. The top left table scans left and locates the next symbol of the ETR to be printed. The top right table records the symbol to be printed or ends the cycle. The bottom two tables scan right and print the appropriate symbol.

Cycle 3 (Restore tape)

This table restores M's simulated tape and encoded table of behaviour. $U_{7,5}$ moves right restoring each λ to $\overleftarrow{b}$, and each δ to λ. This continues until $U_{7,5}$ reads γ, sending $U_{7,5}$'s control to u_5.

	u_7
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_7$
λ	$\overleftarrow{b}, R, u_7$
δ	λ, R, u_7
γ	γ, L, u_5

Table 3.5.20: Cycle 3 of $U_{7,5}$.

Cycle 4 (Choose read or write symbol)

	u_5
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_2$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_3$

	u_2	u_3
γ	$\overleftarrow{b}, R, u_6$	$\overleftarrow{a}, R, u_5$

	u_6
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_6$
$\overleftarrow{b}$	$\overleftarrow{a}, L, u_7$

	u_7
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_7$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_1$

Table 3.5.21: Sub-tables for Cycle 4 of $U_{7,5}$.

This cycle either (i) begins the indexing of an ETR or (ii) completes the execution of an ETR. More precisely: (i) if $U_{7,5}$ is immediately after simulating a left move then this cycle reads the encoded read symbol to the left of the encoded current state, (ii) if $U_{7,5}$ is simulating a right move then this cycle prints the encoded write symbol to the left of the encoded current state.

The halting case for $U_{7,5}$ is more complex than the first two universal Turing machines in this chapter. When the simulated tape head is attempting to move left at the left end of the simulated tape then the last configuration of Cycle 3 for $U_{7,5}$ has the following form:

$$\boldsymbol{u_7}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,\lambda\underline{\gamma}\,\overleftarrow{\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a}^*\,\overleftarrow{b}^3\,\overleftarrow{b}^*\overleftarrow{a}}\,(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$$

The computation continues through 42 configurations before the halting configuration given below is reached.

$$\boldsymbol{u_5}, (\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}\lambda\mathcal{E}')^*\lambda\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,\underline{\gamma}\,\overleftarrow{\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}^*\,\overleftarrow{b}^3\,\overleftarrow{b}^*\overleftarrow{a}}\,(\overleftarrow{b}\,\overleftarrow{a}\cup\overleftarrow{a}\,\overleftarrow{a})^*\overleftarrow{a}^\omega$$

There is no transition rule for the state-symbol pair (u_5, γ) in $U_{7,5}$ so the simulation halts.

3.5.4 Construction of $U_{8,4}$

For $U_{8,4}$ the start state of $\langle M\rangle$ is encoded as $\langle q_1\rangle = \overleftarrow{a}^{5|Q|}\overleftarrow{b}^2$. The encoding of M's current state is of the form $\overleftarrow{a}^*\overleftarrow{b}^2\overleftarrow{b}^*\{\overleftarrow{a}\cup\epsilon\}$ and is of length $5|Q|+2$.

Let $t = (q_x, \sigma_1, \sigma_2, D, q_y)$ be a fixed transition rule in M, then t is encoded via $\mathcal{P}$ using the function $\mathcal{E}$ on its own, or in conjunction with $\mathcal{E}'$, where

$$\mathcal{E}(t) = \begin{cases} \overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)}\,\overleftarrow{b}^{\,2(b(t))}\,\overleftarrow{a}\,\overleftarrow{a} & \text{if } D = R, \sigma_2 = 0 \\ \overleftarrow{a}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)-1}\,\overleftarrow{b}^{\,2(b(t))}\,\overleftarrow{a}\,\overleftarrow{a} & \text{if } D = R, \sigma_2 = 1 \\ \overleftarrow{a}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)-1}\,\overleftarrow{b}^{\,2(b(t))}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{3}\,\overleftarrow{a}\,\overleftarrow{a} & \text{if } D = L, \sigma_2 = 0 \\ \overleftarrow{a}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t)-1}\,\overleftarrow{b}^{\,2(b(t))}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{b}\,\overleftarrow{a}\,\overleftarrow{a} & \text{if } D = L, \sigma_2 = 1 \end{cases} \tag{3.5.8}$$

and

$$\mathcal{E}'(f,t) = \begin{cases} \overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{a(t^{R,x})-3}\,\overleftarrow{b}^{\,2(b(t^{R,x})+2)}\,\overleftarrow{a}\,\overleftarrow{a} & \text{if } \exists t^{R,x}, q_x \neq q_1 \\ \overleftarrow{a} & \text{if } \nexists t^{R,x}, q_x \neq q_1 \\ \overleftarrow{b}\,\overleftarrow{b}\,\overleftarrow{a}\,(\overleftarrow{a}\,\overleftarrow{b}\,)^{5|Q|-3}\,\overleftarrow{b}^{\,8}\,\overleftarrow{a}\,\overleftarrow{a} & \text{if } q_x = q_1 \end{cases} \tag{3.5.9}$$

where as before $t^{R,x}$ is any right moving transition rule such that $t^{R,x} \vdash t$.

The value of the ending E, from Equation (3.2.1), for $U_{8,4}$ is $E = \epsilon$.

Definition 3.5.4 ($U_{8,4}$) *Let Turing machine* $U_{8,4} = (\{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}, \{\overleftarrow{a}, \overleftarrow{b}, \lambda, \delta\}, \overleftarrow{a}, f, u_1, \{u_2\})$ *where* f *is given by the Table 3.5.22.*

	u_1	u_2	u_3	u_4
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$	λ, L, u_4	$\overleftarrow{a}, L, u_3$	$\overleftarrow{a}, L, u_4$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$	$\overleftarrow{b}, L, u_3$	δ, L, u_3	δ, L, u_5
λ	$\overleftarrow{b}, L, u_2$		δ, R, u_1	λ, R, u_6
δ	δ, R, u_1		δ, L, u_3	δ, L, u_4
	u_5	u_6	u_7	u_8
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_5$	$\overleftarrow{a}, R, u_7$	$\overleftarrow{a}, R, u_6$	$\overleftarrow{a}, R, u_6$
$\overleftarrow{b}$	δ, R, u_1	$\overleftarrow{a}, L, u_7$	$\overleftarrow{a}, R, u_1$	$\overleftarrow{a}, L, u_3$
λ	$\overleftarrow{a}, L, u_2$	$\overleftarrow{b}, R, u_6$	$\overleftarrow{a}, R, u_1$	$\overleftarrow{a}, L, u_8$
δ	δ, R, u_5	$\overleftarrow{b}, R, u_8$	λ, R, u_6	$\overleftarrow{b}, R, u_6$

Table 3.5.22: Table of behaviour for $U_{8,4}$.

We give a brief overview of $U_{8,4}$'s computation. The tape contents are given by the same symbols ($\langle 1 \rangle = \overleftarrow{b}\,\overleftarrow{a}$ and $\langle 0 \rangle = \overleftarrow{a}\,\overleftarrow{a}$) to the left and right of the simulated Turing machines tape head.

Cycle 1 (Index next ETR)

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_2$
δ	δ, R, u_1

	u_2
$\overleftarrow{a}$	λ, L, u_4
$\overleftarrow{b}$	$\overleftarrow{b}, L, u_3$

	u_3
$\overleftarrow{a}$	$\overleftarrow{a}, L, u_3$
$\overleftarrow{b}$	δ, L, u_3
λ	δ, R, u_1
δ	δ, L, u_3

Table 3.5.23: Sub-tables for Cycle 1 of $U_{8,4}$.

In Cycle 1 the leftmost table (above) reads the encoded current state. The rightmost table neutralises markers to index the next ETR. The middle table decides when the cycle is complete. In state u_1 the tape head scans from left to right; each $\overleftarrow{b}$ in the encoded current state is replaced with $\overleftarrow{a}$ and $U_{8,4}$ then enters state u_3 via u_2.

Cycle 2 (Print ETR)

	u_2	u_4
$\overleftarrow{a}$	λ, L, u_4	$\overleftarrow{a}, L, u_4$
$\overleftarrow{b}$		δ, L, u_5
λ		λ, R, u_6
δ		δ, L, u_4

	u_5
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_5$
$\overleftarrow{b}$	δ, R, u_1
λ	$\overleftarrow{a}, L, u_2$
δ	δ, R, u_5

	u_1
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_1$
λ	$\overleftarrow{b}, L, u_2$
δ	δ, R, u_1

Table 3.5.24: Sub-tables for Cycle 2 of $U_{8,4}$.

Before we explain this cycle we mention why ETRs for $U_{8,4}$ are longer than ETRs for the other universal Turing machines (e.g. compare Equations (3.5.8) and (3.3.1)). In $U_{8,4}$'s ETRs there are multiple copies of the subwords $\overleftarrow{a}\,\overleftarrow{b}$ and $\overleftarrow{b}\,\overleftarrow{b}$. During the Print ETR cycle, the subword $\overleftarrow{a}\,\overleftarrow{b}$ will cause an $\overleftarrow{a}$ to be printed and the subword $\overleftarrow{b}\,\overleftarrow{b}$ will cause a $\overleftarrow{b}$ to be printed. During this cycle the next symbol to be printed is the symbol to the left of the rightmost $\overleftarrow{b}$ in the ETR. The rightmost $\overleftarrow{b}$ of the subwords $\overleftarrow{a}\,\overleftarrow{b}$ and $\overleftarrow{b}\,\overleftarrow{b}$ is simply a marker and the symbol directly to its left is the symbol that is to be printed. Extra $\overleftarrow{a}$ symbols appear in $U_{8,4}$'s ETRs that do not result in symbols being printed during the print ETR cycle. These extra $\overleftarrow{a}$ symbols are added to allow the restore tape cycle to execute correctly.

This cycle copies an ETR to M's simulated tape head position. The leftmost table scans left and locates the next symbol of the ETR to be

printed or ends the cycle. The middle table records the symbol to be printed. If an $\overleftarrow{a}$ is to be printed the middle table also scans right and prints an $\overleftarrow{a}$. If a $\overleftarrow{b}$ is to be printed the rightmost table scans right and prints a $\overleftarrow{b}$.

Cycle 3 (Restore tape)

	u_6
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_7$
λ	$\overleftarrow{b}, R, u_6$
δ	$\overleftarrow{b}, R, u_8$

	u_7
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_6$
λ	$\overleftarrow{a}, R, u_1$
δ	λ, R, u_6

	u_8
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_6$
λ	$\overleftarrow{a}, L, u_8$
δ	$\overleftarrow{b}, R, u_6$

Table 3.5.25: Sub-tables for Cycle 3 of $U_{8,4}$.

These tables restore M's simulated tape and encoded table of behaviour. $U_{8,4}$'s tape head scans right restoring δ symbols to $\overleftarrow{b}$ and λ symbols. Recall that during Cycle 1 (Index next ETR) λ symbols are replaced with δ symbols in order to index the next ETR. Note also that during Cycle 1, as $U_{8,4}$ scans left, it also replaces each $\overleftarrow{b}$ with δ. As mentioned earlier there are extra $\overleftarrow{a}$ symbols in each ETR that do not effect what is printed to the overlined region. The reason for these extra $\overleftarrow{a}$ symbols is to ensure that $U_{8,4}$ can distinguish which δ symbols to restore to λ symbols and which δ symbols to restore to $\overleftarrow{b}$ symbols. The extra $\overleftarrow{a}$ symbols ensure that $U_{8,4}$ will be in state u_7 if δ should be restored to λ and in u_6 or u_8 if δ should be restored to $\overleftarrow{b}$. This cycle ends when $U_{8,4}$ reads λ.

Cycle 4 (Choose read or write symbol)

	u_6
$\overleftarrow{a}$	$\overleftarrow{a}, R, u_7$
$\overleftarrow{b}$	$\overleftarrow{a}, L, u_7$

	u_7
$\overleftarrow{b}$	$\overleftarrow{a}, R, u_1$

	u_8
$\overleftarrow{b}$	$\overleftarrow{a}, L, u_3$

Table 3.5.26: Sub-tables for Cycle 4 of $U_{8,4}$.

This cycle either (i) begins the indexing of an ETR or (ii) completes the execution of an ETR. More precisely: (i) if $U_{8,4}$ is immediately after simulating a left move then this cycle reads the encoded read symbol to the left of the encoded current state, (ii) if $U_{8,4}$ is simulating a right move then this cycle prints the encoded write symbol to the left of the encoded current state.

Remark 3.5.4 Halting case $U_{8,4}$. Recall that all our universal Turing machines (in this chapter) simulate halting by attempting to simulate a left move at the left end of the simulated tape. This is also true for $U_{8,4}$. However, the halting case for $U_{8,4}$ differs slightly from the halting case for $U_{3,11}$. $U_{3,11}$ halts during Cycle 4 (Choose read or write symbol). $U_{8,4}$ halts in the configuration immediately after printing the last symbol of the left moving ETR at the end of Cycle 2 (Print ETR). When $U_{8,4}$ prints the last symbol of the ETR it enters state u_2 and its tape head moves left. The tape head is now over the tape cell which contains the leftmost symbol of $\langle M \rangle$. The leftmost symbol of $\langle M \rangle$ is λ but during Cycle 2 this tape cell contains a δ (a neutralised λ). There is no transition rule for the state-symbol pair (u_2, δ) in $U_{8,4}$ so the simulation halts.

3.6 Conclusion and future work

We have improved the state of the art in small efficient universal Turing machines giving the smallest machines that simulate Turing machines in $O(t^2)$ time. They are also the smallest known machines where direct simulation of Turing machines is the technique used to establish their universality. The most recent small universal Turing machines simulate Turing machines via 2-tag systems, in $O(t^4 \log^2 t)$ time [Bai01, KR02, Rog96, Rog98]. Before the advent of Minsky's universal Turing machine [Min62a], the smallest universal Turing machines used the technique of direct simulation of Turing machines [Ike58, Wat61]. One problem in the construction of these universal Turing machines was the addressing of states, that is locating the next encoded state during the simulation of a transition rule. Some approaches to solving this problem are briefly discussed in Section 3.1 of Minsky's paper [Min62a]. A major advantage of our algorithm is the fact that the encoded current state is located at the simulated tape head position. This technique simplifies the addressing of states.

As future work it would be of interest to use our algorithm to construct universal Turing machines with state-symbol pairs of $(2, n)$ and $(n, 2)$. This would give a more complete polynomial $O(t^2)$ time curve. Also, our 6-state, 6-symbol universal Turing machine uses only 32 of 36 available transition rules and so it seems possible that it could be improved to a 6-state, 5-symbol machine or a 5-state, 6-symbol machine.

What about small universal Turing machines with less than polynomial time complexity? For example, consider the construction of a linear time universal Turing machine. Our universal Turing machines stores the encoded current state at the simulated tape head location. Suppose the *entire* encoded table of behaviour is stored at this location. Simulating a transition rule merely involves scanning through the encoded table of behaviour,

it is not necessary to scan the entire simulated tape contents. The idea is straightforward, however we expect that a significant number of transition rules would be required to move the *entire* encoded table of behaviour each time a tape head movement is simulated. Trying to construct *small* linear time universal Turing machines using this idea could be difficult, and we leave this as an open problem.

4

The P-completeness of cellular automaton Rule 110

4.1 Introduction

In this chapter we solve an open problem regarding the computational complexity of Rule 110 which is one of the most intuitively simple cellular automata. We show that Rule 110 is efficiently universal and that its prediction problem is P-complete. Rule 110 is a nearest neighbour, one-dimensional, binary cellular automaton [Wol83]. It is composed of a sequence of cells $\ldots p_{-1}p_0p_1\ldots$ where each cell has a binary state $p_i \in \{0,1\}$. At timestep $s+1$, the value $p_{i,s+1} = F(p_{i-1,s}, p_{i,s}, p_{i+1,s})$ of the cell at position i is given by the synchronous local update function F

$$\begin{aligned} F(0,0,0) &= 0 & F(1,0,0) &= 0 \\ F(0,0,1) &= 1 & F(1,0,1) &= 1 \\ F(0,1,0) &= 1 & F(1,1,0) &= 1 \\ F(0,1,1) &= 1 & F(1,1,1) &= 0 \end{aligned}$$

The problem of Rule 110 prediction is defined as follows.

Definition 4.1.1 (Rule 110 prediction**)** *Given an initial Rule 110 configuration, a cell index i and a natural number t written in unary. Is cell p_i in state 1 at time t?*

This problem is in P as a Turing machine simulates the cellular automaton in $O(t^2)$ steps by repeatedly traversing from left to right. From Cook's [Coo04] result one infers a NC [GHR95] lower bound on the problem. Cook showed that Rule 110 simulates Turing machines via the following sequence of simulations

$$\text{Turing machine} \mapsto \text{2-tag system} \mapsto \text{cyclic tag system} \mapsto \text{Rule 110} \quad (4.1.1)$$

where $A \mapsto B$ denotes that A is simulated by B. The universality of 2-tag systems [CM64] is well-known. Cook supplied the latter two simulations (a sketch of Cook's proof is also to be found in Wolfram's book [Wol02]). Each

of these simulations runs in polynomial time (that is, B runs in a number of steps that is polynomial in the number of A's steps) with the exception of the exponentially slow 2-tag system simulation of Turing machines [CM64]. This slowdown is due to the 2-tag system's unary encoding of Turing machine tape contents. Thus from Equation (4.1.1), using the result in [CM64], Rule 110 is an exponentially slow simulator of Turing machines and so it has remained open as to whether RULE 110 PREDICTION is P-complete.

In this work we eliminate the 2-tag system to give the following chain of simulations

$$\text{Turing machine} \mapsto \text{cyclic tag system} \mapsto \text{Rule 110} \tag{4.1.2}$$

Each simulation runs in polynomial time and the reduction from Turing machine to Rule 110 is computable by a logspace Turing machine. Thus our work shows that Rule 110 simulates Turing machines efficiently, giving the following result.

Theorem 4.1.2 RULE 110 PREDICTION is logspace complete for P.

Rule 110 is a very simple (2-state, nearest neighbour, one dimensional) cellular automaton and Cook and Eppstein [Coo04], and Wolfram [Wol02] gave four weakly universal Turing machines that simulate its computation. Their size given as (number of states, number of symbols), are respectively $(2, 5)$, $(3, 4)$, $(4, 3)$ and $(7, 2)$. Previously, these machines were exponentially slow simulators of Turing machines. A corollary of our work is that these small weakly universal machines are now polynomial time simulators of Turing machines. The results we present in this chapter improves on the time overhead of the version presented in [NW06a, NW06b].

The prediction problem for a number of classes of cellular automata has been shown to be P-complete. However, Rule 110 is the simplest so far, in the sense that previous P-completeness results have been shown for more general cellular automata (e.g. more states, neighbours or dimensions). For example, prediction of cellular automata of dimension $d \geqslant 1$ with an arbitrary number of states is known to be P-complete [GHR95]. Ollinger's [Oll02] result shows that prediction for one-dimensional nearest neighbour cellular automata is P-complete for six states. Richard's [Ric06] result reduces this to 4 states. Moore [Moo97a] shows that prediction of binary majority voting cellular automata is P-complete for dimension $d \geqslant 3$. On the other hand, the prediction problem for a variety of linear and quasilinear cellular automata is in NC [Moo97b, Moo98]. The question of whether RULE 110 PREDICTION is P-complete has been asked, either directly or indirectly, in a number of previous works (for example [Moo97b, Moo98, Aar02]).

An interesting decidability result has been given for the reachability problem in binary cellular automata. The reachability problem is as follows: will a given cellular automata configuration evolve from another given

configuration? Codd [Cod68] has shown that the reachability problem is decidable for 2-dimensional binary cellular automata with von Neumann neighbourhood on a blank background. Sutner [Sut03] notes that the proof of universality for Rule 110 depends on a periodic background and that the reachability problem is decidable for Rule 110 on a blank background following Codd's result.

4.2 Cyclic tag systems

Cyclic tag systems were used by Cook [Coo04] to show that Rule 110 is universal.

Definition 4.2.1 (cyclic tag system) *A cyclic tag system C is a list of binary words $\alpha_0, \ldots, \alpha_{p-1}$ called appendants where $\alpha_m \in \{0,1\}^*$.*

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 $w = w_0 \ldots w_{|w|-1} \in \{0,1\}^*$. We call w the *dataword*. Intuitively the list C is a *program* with the marker pointing to instruction α_m. In the initial configuration the marker points to appendant α_0 and w is the binary input word.

Definition 4.2.2 (computation step of a cyclic tag system) *A computation step is deterministic and acts on a configuration in one of two ways:*

- *If $w_0 = 0$ then w_0 is deleted and the marker moves to appendant $\alpha_{(m+1 \mod p)}$.*
- *If $w_0 = 1$ then w_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 dataword is the empty word or (ii) it enters a repeating sequence of configurations. The complexity measures of time and space are defined in the obvious way.

Example 4.2.1 *(cyclic tag system computation)* Let $C = 001, 01, 11$ 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.

$$\begin{array}{llllllll} & \mathbf{001}, 01, 11 & 011 & \vdash & 001, \mathbf{01}, 11 & 11 & \vdash & 001, 01, \mathbf{11} \quad 101 \\ \vdash & \mathbf{001}, 01, 11 & 0111 & \vdash & 001, \mathbf{01}, 11 & 111 & \vdash & \ldots \end{array}$$

4.3 Cyclic tag systems efficiently simulate Turing machines

In an earlier version of this work [NW06a] the time efficiency of cyclic tag systems is proved by replacing the 2-tag system in Equation (4.1.1) with a *clockwise Turing machine* to give the following chain of simulations

$$\begin{aligned} &\text{Turing machine} \mapsto \text{clockwise Turing machine} \\ &\mapsto \text{cyclic tag system} \mapsto \text{Rule 110} \end{aligned} \tag{4.3.1}$$

From the sequence of simulations in Equation (4.3.1) we get Theorem 4.3.1.

Theorem 4.3.1 ([NW06a]) *Let M be a deterministic binary Turing machine with a single tape that computes in time t. Then there is a cyclic tag system that simulates the computation of M in time $O(t^3 \log t)$.*

While Theorem 4.3.1 is not explicitly given in [NW06a] the result may be obtained by a relatively straightforward analysis of Lemmata 1 and 2 in [NW06a]. This is similar to the analysis used in Chapter 5 to get the time bound in Theorem 5.2.6.

In this work the clockwise Turing machine in Equation (4.3.1) is eliminated to give the chain of simulations in Equation (4.1.2). This gives us the improved simulation time given in Theorem 4.3.2. Much of the proof of Theorem 4.1.2 is given by Theorem 4.3.2.

In order to simplify this proof we state the result for Turing machines that have a binary tape alphabet $\Sigma = \{a, b\}$. The proof runs for a number of pages.

Theorem 4.3.2 *Let M be a binary Turing machine with $|Q|$ states that runs in time t. Then there is a cyclic tag system C_M that simulates the computation of M in time $O(|Q|t^2 \log t)$.*

Proof. Let $M = (Q, \{a, b\}, b, f, q_1, q_{|Q|})$ where $Q = \{q_1, \ldots, q_{|Q|}\}$ are the states, $\{a, b\}$ is the binary alphabet, f is the transition function, b is the blank symbol and $q_1, q_{|Q|} \in Q$ are the initial and final states, respectively. In the sequel $\sigma_j \in \{a, b\}$. The bulk of the proof is concerned with simulating a single (but arbitrary) left moving transition rule of M in time $O(|Q|t \log t)$. The simulation of a right moving transition rule was given previously in [NW06a] and is similar to a restriction of the method for simulating a left moving transition rule so we do not give it here.

Encoding

We define the cyclic tag system (program) to be of the following form $C_M = \alpha_0, \ldots, \alpha_{2z-1}$ where $z = 60|Q| + 121$. Given a configuration of

M (consisting of current state $q_i \in Q$, read symbol σ_j, and tape contents $w_M = \sigma_1 \dots \sigma_j \sigma_{j+1} \dots \sigma_{s-2} \in \{a, b\}^*$) we encode this as a configuration of C_M as follows:

$$\boldsymbol{\alpha_0}, \dots, \alpha_{2z-1} \quad \langle 1, q_{i,L} \rangle \langle \sigma_{j+1} \rangle \dots \langle \sigma_{s-2} \rangle \langle \sigma_b \rangle \mu^{s'} \langle \sigma_b \rangle \langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_j \rangle \tag{4.3.2}$$

Here the two infinite sequences of bank symbols on each side of the tape contents are each encoded by $\langle \sigma_b \rangle$. The encoded current state is $\langle 1, q_{i,L} \rangle$ and the encoded read symbol $\langle \sigma_j \rangle$ is the rightmost encoded tape symbol. Also, $\mu = 0^7 10^{z-8}$ and

$$s' = 2^{\lceil \log_2 s \rceil} \tag{4.3.3}$$

are used for a 'tape length' counter. The values of appendants $\alpha_0, \dots, \alpha_{2z-1}$ are given during the proof below. States q_i and tape symbols $\{a, b\}$ of M, and $\langle \sigma_b \rangle$ are encoded as:

$$\begin{aligned} \langle 1, q_{i,L} \rangle &= 0^{60i+50} 10^{2z-60i-51} \\ \langle a \rangle &= 010^{2z-2} \\ \langle b \rangle &= 0^2 10^{2z-3} \\ \langle \sigma_b \rangle &= 0^3 10^{2z-4} \end{aligned}$$

Our simulation algorithm consists of a number of stages. In a cyclic tag system configuration the current stage x of our algorithm is identifiable by the notation $\langle x, q_{i,L} \rangle$.

How to read the tables

We define the cyclic tag system C_M via a number of tables that specify encoded objects (e.g. encoded symbols, states) in the dataword and the appendants they map to. Each table row gives an "encoded object" followed by the "encoded object length". The "initial marker index" gives the location of the program marker immediately before the encoded object is read. Each encoded object indexes an appendant α_y, where y is specified by the "index y of appendant" column and α_y is specified by the "appendant α_y" column.

To aid the reader we carefully describe the initial steps in the simulation of a transition rule. We encode a configuration that is arbitrary except for its tape length (which is 2). Initially the marker is pointing at appendant α_0 and the dataword is $\langle 1, q_{i,L} \rangle \langle \sigma_2 \rangle \langle \sigma_b \rangle \mu\mu\mu\mu \langle \sigma_b \rangle \langle \sigma_1 \rangle \in \{0,1\}^{14z}$. The leftmost $2z$ symbols in the dataword encode the current state q_i. From Table 4.3.1 this is $\langle 1, q_{i,L} \rangle = 0^{60i+50} 10^{2z-60i-51}$. The computation begins by deleting the $60i + 50$ leftmost 0 symbols while moving the marker rightwards through the appendants, one step for each deletion. The leftmost data symbol is now 1, this is deleted and causes the appendant α_{60i+50} to be appended onto the rightmost end of the dataword. From Table 4.3.1 we see that this

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$\langle 1, q_{i,L}\rangle = 0^{60i+50}10^{2z-60i-51}$	$2z$	0	$60i+50$	$\langle\, 1', q_{i,L}\rangle$
$\langle 1, q_{i,L}\rangle = 0^{60i+50}10^{2z-60i-51}$	$2z$	z	$z+60i+50$	$0^z\langle 1', q_{i,L,s<s'}\rangle$
$\langle 1, q_{i,L,s<s'}\rangle = 0^{60i+55}10^{2z-60i-56}$	$2z$	0	$60i+55$	$\langle 1', q_{i,L,s<s'}\rangle$
$\langle 1, q_{i,L,s<s'}\rangle = 0^{60i+55}10^{2z-60i-56}$	$2z$	z	$z+60i+55$	$0^z\langle 1', q_{i,L,s<s'}\rangle$
$\langle a\rangle = 010^{2z-2}$	$2z$	0	1	$\langle a\rangle$
$\langle a\rangle = 010^{2z-2}$	$2z$	z	$z+1$	$\langle a\rangle$
$\langle b\rangle = 0^210^{2z-3}$	$2z$	0	2	$\langle b\rangle$
$\langle b\rangle = 0^210^{2z-3}$	$2z$	z	$z+2$	$\langle b\rangle$
$\langle \sigma_b\rangle = 0^310^{2z-4}$	$2z$	0	3	$\langle \sigma_b\rangle$
$\langle \sigma_b\rangle = 0^310^{2z-4}$	$2z$	z	$z+3$	$\langle \sigma_b\rangle$
$\langle \not{a}\rangle = 0^410^{2z-5}$	$2z$	0	4	$\langle \not{a}\rangle$
$\langle \not{a}\rangle = 0^410^{2z-5}$	$2z$	z	$z+4$	$\langle \not{a}\rangle$
$\langle \not{b}\rangle = 0^510^{2z-6}$	$2z$	0	5	$\langle \not{b}\rangle$
$\langle \not{b}\rangle = 0^510^{2z-6}$	$2z$	z	$z+5$	$\langle \not{b}\rangle$
$\langle \not{\sigma}_b\,\rangle = 0^610^{2z-7}$	$2z$	0	6	$\langle \not{\sigma}_b\,\rangle$
$\langle \not{\sigma}_b\,\rangle = 0^610^{2z-7}$	$2z$	z	$z+6$	$\langle \not{\sigma}_b\,\rangle$
$\mu = 0^710^{z-8}$	z	0	7	$\not{\mu}$
$\mu = 0^710^{z-8}$	z	z	$z+7$	μ'
$\mu' = 0^810^{2z-9}$	$2z$	0	8	μ'
$\mu' = 0^810^{2z-9}$	$2z$	z	$z+8$	μ'
$\not{\mu} = 0^910^{2z-10}$	$2z$	0	9	$\not{\mu}$
$\not{\mu} = 0^910^{2z-10}$	$2z$	z	$z+9$	$\not{\mu}$

Table 4.3.1: (Stage 1. Halve counter). Every odd numbered $\mu = 0^710^{z-8}$ is marked off by being changed to $\not{\mu} = 0^910^{2z-10}$.

is $\alpha_{60i+50} = \langle 1', q_{i,L}\rangle$. Then $2z - 60i - 51$ contiguous 0 symbols are deleted while moving the marker one step for each deletion. Since $|\langle 1, q_{i,L}\rangle| = 2z$ and there are exactly $2z$ appendants in C_M, the marker is once again positioned at α_0. We write these $2z$ steps as

$$\begin{aligned} & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} \quad \langle 1, q_{i,L}\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle\mu\mu\mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle \\ \vdash^{2z} \; & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} \quad \langle\sigma_2\rangle\langle\sigma_b\rangle\mu\mu\mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 1', q_{i,L}\rangle \end{aligned}$$

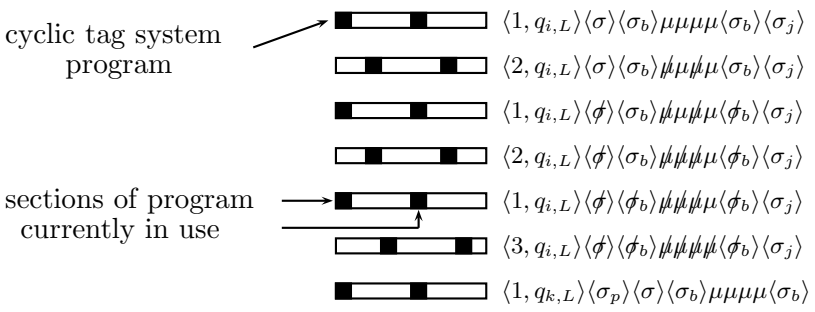


Figure 4.3.1: Cyclic tag system simulation of transition rule $(q_i, \sigma_j, \sigma_p, L, q_k)$. The cyclic tag system program is illustrated on the left. In the dataword the encoded current state $\langle x, q_{i,L}\rangle$ directs the control flow by determining the sections of the cyclic tag system program that are used in algorithm Stage x.

Algorithm overview

Our cyclic tag system algorithm has three stages. Stages 1 and 2 isolate the encoded read symbol $\langle\sigma_1\rangle$ of M which is located immediately to the left[1] of the encoded current state $\langle 1, q_{i,L}\rangle$. These stages make use of the tape-length counter specified by Equations (4.3.2) and (4.3.3). In Stage 1 every odd numbered μ is marked and then in Stage 2 every even number $\langle\sigma\rangle$ is marked. This process is iterated until all μ objects are marked ($1 + \log_2 s'$ iterations). The first six configurations of Figures 4.3.1 and 4.3.2 illustrate this process. The encoded read symbol is now isolated as it is the only unmarked encoded tape symbol. The computation then enters Stage 3 which uses the encoded current state and (isolated) encoded read symbol to index an appendant that encodes the write symbol and next state. Finally the counter is doubled if necessary to maintain the equality in Equation (4.3.3). The last two configurations of Figure 4.3.2 illustrate this process.

Stage 1. Halve counter

The counter value is specified by Equation (4.3.3) as the number of μ (or later, μ') objects. This value is halved by marking half of the μ objects (changing μ to $\not\mu$) using Table 4.3.1. In this table we see that $|\mu| = z$ so exactly two μ objects are read for a single traversal of the marker through all $2z$ appendants. Every odd numbered μ indexes $\not\mu$ and every even numbered μ indexes μ'. The encoded state $\langle 1, q_{i,L}\rangle$ indexes $\langle 1', q_{i,L}\rangle$ or $\langle 1', q_{i,L,s<s'}\rangle$,

[1]The manner in which a cyclic tag system reads its dataword is circular, hence the rightmost object in a dataword can be considered to be immediately to the left of the leftmost object. To see this, note that $\langle 1, q_{i,L}\rangle$ is the leftmost object in the first configuration and $\langle 1', q_{i,L}\rangle$ is the rightmost object in the second configuration.

which sends control to Table 4.3.2.

We continue the above simulation (we later generalise to an arbitrary number of tape symbols).

$$\begin{aligned}
& \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle\sigma_2\rangle\langle\sigma_b\rangle\mu\mu\mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 1', q_{i,L}\rangle \\
\vdash^{2z} \ & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle\sigma_b\rangle\mu\mu\mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 1', q_{i,L}\rangle\langle\sigma_2\rangle \\
\vdash^{2z} \ & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \mu\mu\mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 1', q_{i,L}\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle \\
\vdash^{z} \ & \alpha_0, \ldots, \boldsymbol{\alpha_z}, \ldots, \alpha_{2z-1} && \mu\mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 1', q_{i,L}\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle\not\mu \\
\vdash^{z} \ & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \mu\mu\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 1', q_{i,L}\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle\not\mu\mu' \\
\vdash^{6z} \ & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle 1', q_{i,L}\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma_b\rangle\langle\sigma_1\rangle
\end{aligned}$$

The algorithm tests if the counter is 0 by checking if exactly one unmarked μ was read. If so $\langle 3, q_{i,L}\rangle$ or $\langle 3, q_{i,L,s<s'}\rangle$ is appended and we enter Stage 3. Otherwise $\langle 2, q_{i,L}\rangle$ or $\langle 2, q_{i,L,s<s'}\rangle$ is appended and we enter Stage 2. Table 4.3.2 simulates this 'if' statement.

As we continue our simulation we note from Table 4.3.2 that the word $\langle 1', q_{i,L}\rangle$ is of length $2z + 10$. Hence the marker is at appendant α_{10} after $\langle 1', q_{i,L}\rangle$ is read. The computation proceeds as follows:

$$\begin{aligned}
& \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle 1', q_{i,L}\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma_b\rangle\langle\sigma_1\rangle \\
\vdash^{2z+10} \ & \alpha_0, \ldots, \boldsymbol{\alpha_{10}}, \ldots, \alpha_{2z-1} && \langle\sigma_2\rangle\langle\sigma_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma_b\rangle\langle\sigma_1\rangle\langle 2, q_{i,L}\rangle \\
\vdash^{16z} \ & \alpha_0, \ldots, \boldsymbol{\alpha_{10}}, \ldots, \alpha_{2z-1} && \langle 2, q_{i,L}\rangle\langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle
\end{aligned}$$

Immediately above is the first configuration of Stage 2.

Stage 2. Mark half of the encoded tape symbols

There are three rather large tables for Stage 2. To simplify the proof for the reader we give summary tables. If an encoded object points to an appendant with it is own value no table entry is given.

The ultimate aim of this stage is to isolate the encoded read symbol. Each iteration of this stage uses Table 4.3.5 to mark off every even numbered encoded tape symbol $\langle\sigma\rangle$. We must ensure that the encoded read symbol $\langle\sigma'_1\rangle$ immediately to the left of $\langle 2, q_{i,L}\rangle$ is at an odd numbered position so it is not marked off during the execution of Table 4.3.5. Tables 4.3.3 and 4.3.4 test if the encoded read symbol is at an even numbered position and if so the dummy symbol $\langle\sigma'_d\rangle$ is appended to place the encoded read symbol at an odd numbered position.

Note from Table 4.3.3 that encoded state $\langle 2, q_{i,L}\rangle$ or $\langle 2, q_{i,L,s<s'}\rangle$ is of length $2z+10$. Hence the marker is at appendant α_{20} after reading $\langle 2, q_{i,L}\rangle$. When the initial marker index is moved to α_{20} this causes each encoded object to index a copy of itself (e.g. μ' indexes μ'). Thus the computation

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$\langle 1', q_{i,L}\rangle = 0^{60i+51}10^{2z-60i-42}$	$2z+10$	0	$60i+51$	$\langle 2, q_{i,L}\rangle$
$\langle 1', q_{i,L}\rangle = 0^{60i+51}10^{2z-60i-42}$	$2z+10$	z	$z+60i+51$	$\langle 3, q_{i,L}\rangle$
$\langle 1', q_{i,L,s<s'}\rangle = 0^{60i+56}10^{2z-60i-47}$	$2z+10$	0	$60i+56$	$\langle 2, q_{i,L,s<s'}\rangle$
$\langle 1', q_{i,L,s<s'}\rangle = 0^{60i+56}10^{2z-60i-47}$	$2z+10$	z	$z+60i+56$	$\langle 3, q_{i,L,s<s'}\rangle$
$\langle a\rangle = 010^{2z-2}$	$2z$	10	11	$\langle a'\rangle$
$\langle a\rangle = 010^{2z-2}$	$2z$	$z+10$	$z+11$	$\langle a\rangle$
$\langle b\rangle = 0^2 10^{2z-3}$	$2z$	10	12	$\langle b'\rangle$
$\langle b\rangle = 0^2 10^{2z-3}$	$2z$	$z+10$	$z+12$	$\langle b\rangle$
$\langle \sigma_b\rangle = 0^3 10^{2z-4}$	$2z$	10	13	$\langle \sigma'_b\rangle$
$\langle \sigma_b\rangle = 0^3 10^{2z-4}$	$2z$	$z+10$	$z+13$	$\langle \sigma_b\rangle$
$\langle \not{a}\rangle = 0^4 10^{2z-5}$	$2z$	10	14	$\langle \not{a}\rangle$
$\langle \not{a}\rangle = 0^4 10^{2z-5}$	$2z$	$z+10$	$z+14$	$\langle \not{a}\rangle$
$\langle \not{b}\rangle = 0^5 10^{2z-6}$	$2z$	10	15	$\langle \not{b}\rangle$
$\langle \not{b}\rangle = 0^5 10^{2z-6}$	$2z$	$z+10$	$z+15$	$\langle \not{b}\rangle$
$\langle \not{\sigma}_b\rangle = 0^6 10^{2z-7}$	$2z$	10	16	$\langle \not{\sigma}_b\rangle$
$\langle \not{\sigma}_b\rangle = 0^6 10^{2z-7}$	$2z$	$z+10$	$z+16$	$\langle \not{\sigma}_b\rangle$
$\mu' = 0^8 10^{2z-9}$	$2z$	10	18	μ'
$\mu' = 0^8 10^{2z-9}$	$2z$	$z+10$	$z+18$	μ'
$\not{\mu} = 0^9 10^{2z-10}$	$2z$	10	19	$\not{\mu}$
$\not{\mu} = 0^9 10^{2z-10}$	$2z$	$z+10$	$z+19$	$\not{\mu}$

Table 4.3.2: (Stage 1. Check counter value). Here $\langle 1', q_{i,L}\rangle$ or $\langle 1', q_{i,L,s<s'}\rangle$ is used to check if the counter is 0.

proceeds as follows.

$$\begin{aligned} & \alpha_0, \ldots, \boldsymbol{\alpha_{10}}, \ldots, \alpha_{2z-1} \quad \langle 2, q_{i,L}\rangle\langle \sigma'_2\rangle\langle \sigma'_b\rangle\not{\mu}\mu'\not{\mu}\mu'\langle \sigma'_b\rangle\langle \sigma'_1\rangle \\ \vdash^{2z+10} \ & \alpha_0, \ldots, \boldsymbol{\alpha_{20}}, \ldots, \alpha_{2z-1} \quad \langle \sigma'_2\rangle\langle \sigma'_b\rangle\not{\mu}\mu'\not{\mu}\mu'\langle \sigma'_b\rangle\langle \sigma'_1\rangle\langle 2', q_{i,L}\rangle \\ \vdash^{12z} \ & \alpha_0, \ldots, \boldsymbol{\alpha_{20}}, \ldots, \alpha_{2z-1} \quad \langle 2', q_{i,L}\rangle\langle \sigma'_2\rangle\langle \sigma'_b\rangle\not{\mu}\mu'\not{\mu}\mu'\langle \sigma'_b\rangle\langle \sigma'_1\rangle \end{aligned}$$

Each encoded tape symbol $\langle \sigma'\rangle$ is of length z. If an odd number of $\langle \sigma'\rangle$ objects are read during execution of Table 4.3.3 then the initial marker will be at $20+z$ and rows 2 or 4 of Table 4.3.4 will execute. If an even number of encoded tape symbols $\langle \sigma'\rangle$ are read then the initial marker will be at 20 and rows 1 or 3 of Table 4.3.4 will execute appending the dummy object $\langle \sigma'_d\rangle$. In our simulation an even number of $\langle \sigma'\rangle$ symbols were read during the execution of Table 4.3.3 hence the initial marker index is at appendant α_{20}. The encoded state $\langle 2', q_{i,L}\rangle$ moves the initial marker index to appendant α_{30}. This causes each encoded object to index a copy of itself (e.g. μ' indexes μ').

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$\langle 2, q_{i,L}\rangle = 0^{60i+42}10^{2z-60i-33}$	$2z+10$	10	$60i+52$	$\langle 2', q_{i,L}\rangle$
$\langle 2, q_{i,L,s<s'}\rangle = 0^{60i+47}10^{2z-60i-38}$	$2z+10$	10	$60i+57$	$\langle 2', q_{i,L,s<s'}\rangle$

Table 4.3.3: (Stage 2. Read σ' symbols.) Each encoded object omitted from this summary table indexes a copy of itself (e.g. μ' indexes μ').

Thus continuing our simulation gives:

$$\begin{aligned}
& \alpha_0, \ldots, \boldsymbol{\alpha_{20}}, \ldots, \alpha_{2z-1} && \langle 2', q_{i,L}\rangle\langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle \\
\vdash^{2z+10}\ & \alpha_0, \ldots, \boldsymbol{\alpha_{30}}, \ldots, \alpha_{2z-1} && \langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma_d\rangle\langle 2'', q_{i,L}\rangle\langle\sigma'_d\rangle \\
\vdash^{14z}\ & \alpha_0, \ldots, \boldsymbol{\alpha_{30}}, \ldots, \alpha_{2z-1} && \langle 2'', q_{i,L}\rangle\langle\sigma'_d\rangle\langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma'_d\rangle
\end{aligned}$$

In a manner similar to that of Table 4.3.1, in which every odd numbered μ was marked off, Table 4.3.5 marks off even numbered $\langle\sigma'\rangle$. To see this note $|\langle a'\rangle| = |\langle b'\rangle| = |\langle\sigma'_b\rangle| = |\langle\sigma'_d\rangle| = z$. Thus every even numbered $\langle\sigma'\rangle$ indexes $\langle\not\phi\rangle$ and every odd numbered $\langle\sigma'\rangle$ indexes $\langle\sigma\rangle$. Continuing our simulation gives:

$$\begin{aligned}
& \alpha_0, \ldots, \boldsymbol{\alpha_{30}}, \ldots, \alpha_{2z-1} && \langle 2'', q_{i,L}\rangle\langle\sigma'_d\rangle\langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma'_d\rangle \\
\vdash^{2z+10}\ & \alpha_0, \ldots, \boldsymbol{\alpha_{40}}, \ldots, \alpha_{2z-1} && \langle\sigma'_d\rangle\langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma'_d\rangle 0^{2z-40}\langle 1, q_{i,L}\rangle \\
\vdash^{z}\ & \alpha_0, \ldots, \boldsymbol{\alpha_{z+40}}, \ldots, \alpha_{2z-1} && \langle\sigma'_2\rangle\langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma'_d\rangle 0^{2z-40}\langle 1, q_{i,L}\rangle \\
\vdash^{z}\ & \alpha_0, \ldots, \boldsymbol{\alpha_{40}}, \ldots, \alpha_{2z-1} && \langle\sigma'_b\rangle\not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma'_d\rangle 0^{2z-40}\langle 1, q_{i,L}\rangle\langle\not\phi_2\rangle \\
\vdash^{z}\ & \alpha_0, \ldots, \boldsymbol{\alpha_{z+40}}, \ldots, \alpha_{2z-1} && \not\mu\mu'\not\mu\mu'\langle\sigma'_b\rangle\langle\sigma'_1\rangle\langle\sigma'_d\rangle 0^{2z-40}\langle 1, q_{i,L}\rangle\langle\not\phi_2\rangle\langle\sigma_b\rangle \\
\vdash^{13z-40}\ & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle 1, q_{i,L}\rangle\langle\not\phi_2\rangle\langle\sigma_b\rangle\not\mu\mu\not\mu\mu\langle\not\phi_b\rangle\langle\sigma_1\rangle
\end{aligned}$$

The simulation continues as follows iterating Stages 1 and 2:

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$\langle 2', q_{i,L}\rangle = 0^{60i+33}10^{2z-60i-24}$	$2z+10$	20	$60i+53$	$\langle\sigma_d\rangle\langle 2'', q_{i,L}\rangle\langle\sigma'_d\rangle$
$\langle 2', q_{i,L}\rangle = 0^{60i+33}10^{2z-60i-24}$	$2z+10$	$z+20$	$z+60i+53$	$\langle 2'', q_{i,L}\rangle$
$\langle 2', q_{i,L,s<s'}\rangle = 0^{60i+38}10^{2z-60i-29}$	$2z+10$	20	$60i+58$	$\langle\sigma_d\rangle\langle 2'', q_{i,L,s<s'}\rangle\langle\sigma'_d\rangle$
$\langle 2', q_{i,L,s<s'}\rangle = 0^{60i+38}10^{2z-60i-29}$	$2z+10$	$z+20$	$z+60i+58$	$\langle 2'', q_{i,L,s<s'}\rangle$
$\langle\sigma_d\rangle = 0^{13}10^{2z-14}$	$2z$	30	43	$\langle\sigma'_d\rangle$

Table 4.3.4: (Stage 2. Is the number of σ' symbols odd or even.) Each encoded object omitted from this summary table indexes a copy of itself (e.g. μ' indexes μ').

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$\langle 2'', q_{i,L}\rangle = 0^{60i+24}10^{2z-60i-15}$	$2z+10$	30	$60i+54$	$0^{2z-40}\langle 1, q_{i,L}\rangle$
$\langle 2'', q_{i,L,s<s'}\rangle = 0^{60i+29}10^{2z-60i-20}$	$2z+10$	30	$60i+59$	$0^{2z-40}\langle 1, q_{i,L,s<s'}\rangle$
$\mu' = 0^8 10^{2z-9}$	$2z$	40	48	μ
$\mu' = 0^8 10^{2z-9}$	$2z$	$z+40$	$z+48$	μ
$\langle a'\rangle = 0^{10}10^{z-11}$	z	40	50	$\langle a\rangle$
$\langle a'\rangle = 0^{10}10^{z-11}$	z	$z+40$	$z+50$	$\langle \not\phi\rangle$
$\langle b'\rangle = 0^{11}10^{z-12}$	z	40	51	$\langle b\rangle$
$\langle b'\rangle = 0^{11}10^{z-12}$	z	$z+40$	$z+51$	$\langle \not b\rangle$
$\langle \sigma'_b\rangle = 0^{12}10^{z-13}$	z	40	52	$\langle \sigma_b\rangle$
$\langle \sigma'_b\rangle = 0^{12}10^{z-13}$	z	$z+40$	$z+52$	$\langle \not\phi_b\rangle$
$\langle \sigma'_d\rangle = 0^{13}10^{z-14}$	z	40	53	ϵ
$\langle \sigma'_d\rangle = 0^{13}10^{z-14}$	z	$z+40$	$z+53$	ϵ

Table 4.3.5: (Stage 2. Mark half of the encoded tape symbols). Each encoded object omitted from this summary table indexes a copy of itself.

$$
\begin{aligned}
&\boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle 1, q_{i,L}\rangle\langle \phi_2\rangle\langle \sigma_b\rangle \not\mu\mu\not\mu\mu\langle \phi_b\rangle\langle \sigma_1\rangle \\
\vdash^{34z+10}\ &\alpha_0, \ldots, \boldsymbol{\alpha_{10}}, \ldots, \alpha_{2z-1} && \langle 2, q_{i,L}\rangle\langle \phi_2\rangle\langle \sigma'_b\rangle \not\mu\not\mu\not\mu\mu'\langle \phi_b\rangle\langle \sigma'_1\rangle \\
\vdash^{54z-10}\ &\boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle 1, q_{i,L}\rangle\langle \phi_2\rangle\langle \phi_b\rangle \not\mu\not\mu\not\mu\mu\langle \phi_b\rangle\langle \sigma_1\rangle \\
\vdash^{17z}\ &\alpha_0, \ldots, \boldsymbol{\alpha_z}, \ldots, \alpha_{2z-1} && \langle 1', q_{i,L}\rangle\langle \phi_2\rangle\langle \phi_b\rangle \not\mu\not\mu\not\mu\not\mu\langle \phi_b\rangle\langle \sigma_1\rangle \\
\vdash^{2z+10}\ &\alpha_0, \ldots, \boldsymbol{\alpha_{z+10}}, \ldots, \alpha_{2z-1} && \langle \phi_2\rangle\langle \phi_b\rangle \not\mu\not\mu\not\mu\not\mu\langle \phi_b\rangle\langle \sigma_1\rangle\langle 3, q_{i,L}\rangle \\
\vdash^{16z}\ &\alpha_0, \ldots, \boldsymbol{\alpha_{z+10}}, \ldots, \alpha_{2z-1} && \langle 3, q_{i,L}\rangle\langle \phi_2\rangle\langle \phi_b\rangle \not\mu\not\mu\not\mu\not\mu\langle \phi_b\rangle\langle \sigma_1\rangle
\end{aligned}
$$

The configuration immediately above is the first configuration of Stage 3.

When must the counter value s' be doubled?

If the simulated tape head is reading $\langle \sigma_b\rangle$ (the encoding of the infinite sequence of blank symbols to the left and right of the tape contents) then the tape length is increasing. If this is the case, and s is a power of 2, then we must double the counter value in order to satisfy Equation (4.3.3). Figure 4.3.2 illustrates this process. The doubling occurs in Stage 3. However, the tape length test happens in Stage 1 using Table 4.3.1 as follows.

Suppose that s (the length of the tape is $s-2$) is not a power of 2 and thus $s < s'$. Then, on some iteration, Table 4.3.5 reads an odd number, strictly greater than 1, of unmarked encoded tape symbols. If this occurs then $\langle 1, q_{i,L}\rangle$ indexes the appendant $\langle 1', q_{i,L,s<s'}\rangle$. To see this, notice that in Stage 2, following execution of Table 4.3.2, the tape symbols a, b are

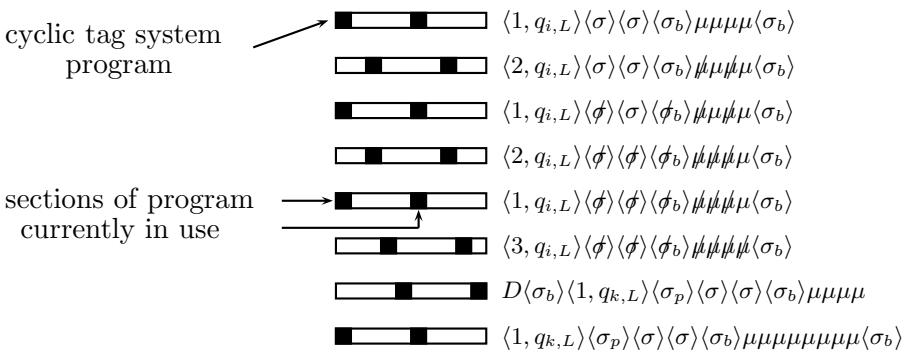


Figure 4.3.2: Cyclic tag system simulation of transition rule $(q_i, \sigma_b, \sigma_p, L, q_k)$. The beginning of the simulation is the same as the simulation in Figure 4.3.1. However this simulation differs as the tape length is increasing and so the counter value (number of μ symbols) must be doubled to maintain the equality in Equation (4.3.3).

encoded as $\langle a'\rangle, \langle b'\rangle$ and the infinite sequence of blank symbols to the left and right is encoded as $\langle\sigma'_b\rangle$ where $|\langle\sigma'_b\rangle| = |\langle a'\rangle| = |\langle b'\rangle| = z$. If C_M reads an odd number of these during execution of Table 4.3.5 then the initial marker index is at z. Suppose otherwise that s is not a power of 2. Then $\langle 1, q_{i,L}\rangle$ will always index the appendant $\langle 1', q_{i,L}\rangle$ during some iteration of Stage 1. In summary, if $\langle 1', q_{i,L,s<s'}\rangle$ is not appended before Stage 3 begins then the number of $\langle\sigma\rangle$ objects in the dataword is a power of 2 and $s = s'$.

Stage 3. Complete simulation of transition rule

In this stage an appendant α_y is indexed, based on the value of the encoded current state and encoded read symbol using Table 4.3.6. The printing of appendant α_y simulates the encoded write symbol, encoded next state, and the tape head movement.

Using Table 4.3.6 we read the encoded current state $\langle 3, q_{i,L}\rangle$ or $\langle 3, q_{i,L,s<s'}\rangle$ after which the initial marker index is $60i + 60$ or $60i + 70$ respectively. The encoded read symbol was already isolated and uniquely retains its original value of $\langle a\rangle$, $\langle b\rangle$ or $\langle\sigma_b\rangle$; this value points at the appendant α_y (rows 3 to 6). All other (non-isolated) encoded tape symbols are of the form $\langle\not\sigma\rangle$, $\langle\not b\rangle$ or $\langle\not\sigma_b\rangle$ and they point to the appendants $\langle a\rangle$, $\langle b\rangle$ or $\langle\sigma_b\rangle$ respectively.

The transitions rule $(q_i, \sigma_j, \sigma_p, L, q_k)$ is encoded by each of the appendants given in rows 3 to 6 of Table 4.3.6. The simulated tape head is *not* reading $\langle\sigma_b\rangle$ and hence the appendant appended is of the form $\langle 1, q_{k,L}\rangle\langle\sigma_p\rangle$.

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$\langle 3, q_{i,L}\rangle = 0^{60i+44}10^{z+5}$	$z+60i+50$	$z+10$	$z+60i+54$	$0^{2z-60i-60}$
$\langle 3, q_{i,L,s<s'}\rangle = 0^{60i+49}10^{z+10}$	$z+60i+60$	$z+10$	$z+60i+59$	$0^{2z-60i-70}$
$\langle a\rangle = 010^{2z-2}$	$2z$	$60i+v$	$60i+v+1$	$\langle 1, q_{k,L}\rangle\langle\sigma_p\rangle$
$\langle b\rangle = 0^2 10^{2z-3}$	$2z$	$60i+v$	$60i+v+2$	$\langle 1, q_{k,L}\rangle\langle\sigma_p\rangle$
$\langle\sigma_b\rangle = 0^3 10^{2z-4}$	$2z$	$60i+60$	$60i+63$	$\langle\sigma_b\rangle D\langle 1, q_{k,L}\rangle\langle\sigma_p\rangle$
$\langle\sigma_b\rangle = 0^3 10^{2z-4}$	$2z$	$60i+70$	$60i+73$	$\langle\sigma_b\rangle\langle 1, q_{k,L}\rangle\langle\sigma_p\rangle$
$\langle\not{a}\rangle = 0^4 10^{2z-5}$	$2z$	$60i+v$	$60i+v+4$	$\langle a\rangle$
$\langle\not{b}\rangle = 0^5 10^{2z-6}$	$2z$	$60i+v$	$60i+v+5$	$\langle b\rangle$
$\langle\not{\sigma}_b\rangle = 0^6 10^{2z-7}$	$2z$	$60i+v$	$60i+v+6$	$\langle\sigma_b\rangle$
$\not{\mu} = 0^9 10^{2z-10}$	$2z$	$60i+v$	$60i+v+9$	μ

Table 4.3.6: (Stage 3. Simulate transition rule $(q_i, \sigma_j, \sigma_p, L, q_k)$). This table prints the encoded write symbol σ_p and establishes the new encoded current state $\langle 1, q_{k,L}\rangle$. If the counter does not need to be doubled this table completes simulation of the transition rule. Also, $v \in \{60, 70\}$.

Continuing the simulation we get:

$$\begin{aligned}
& \alpha_0, \ldots, \boldsymbol{\alpha_{z+10}}, \ldots, \alpha_{2z-1} && \langle 3, q_{i,L}\rangle\langle\not{\sigma}_2\rangle\langle\not{\sigma}_b\rangle\not{\mu}\not{\mu}\not{\mu}\not{\mu}\not{\mu}\langle\not{\sigma}_b\rangle\langle\sigma_1\rangle \\
\vdash^{z+60i+50} & \alpha_0, \ldots, \boldsymbol{\alpha_{60i+60}}, \ldots, \alpha_{2z-1} && \langle\not{\sigma}_2\rangle\langle\not{\sigma}_b\rangle\not{\mu}\not{\mu}\not{\mu}\not{\mu}\not{\mu}\langle\not{\sigma}_b\rangle\langle\sigma_1\rangle 0^{2z-60i-60} \\
\vdash^{14z} & \alpha_0, \ldots, \boldsymbol{\alpha_{60i+60}}, \ldots, \alpha_{2z-1} && \langle\sigma_1\rangle 0^{2z-60i-60}\langle\sigma_2\rangle\langle\sigma_b\rangle\mu\mu\mu\mu\langle\sigma_b\rangle \\
\vdash^{2z} & \alpha_0, \ldots, \boldsymbol{\alpha_{60i+60}}, \ldots, \alpha_{2z-1} && 0^{2z-60i-60}\langle\sigma_2\rangle\langle\sigma_b\rangle\mu\mu\mu\mu\langle\sigma_b\rangle\langle 1, q_{k,L}\rangle\langle\sigma_p\rangle \\
\vdash^{12z-60i-60} & \boldsymbol{\alpha_0}, \ldots, \alpha_{2z-1} && \langle 1, q_{k,L}\rangle\langle\sigma_p\rangle\langle\sigma_2\rangle\langle\sigma_b\rangle\not{\mu}\mu'\not{\mu}\mu'\langle\sigma_b\rangle
\end{aligned}$$

The transition rule simulation is now complete. The appendant marker of C_M's program is at appendant α_0. The encoded write symbol is written, the new encoded state $\langle 1, q_{k,L}\rangle$ is established and the tape head movement is simulated. Although the counter has already been halved this does not alter the algorithm control flow during simulation of the next transition rule.

We have given a sequence of configurations that explicitly simulate the application of a left moving transition rule of M. We used arbitrary initial and next states $q_i, q_k \in Q$, and arbitrary tape symbols $\sigma_j, \sigma_p \in \{a, b\}$.

The transition rule simulation is specific in the sense that the length of the tape data is fixed and the encoded read symbol is not of the form $\langle\sigma_b\rangle$. If the simulated tape head is reading $\langle\sigma_b\rangle$ (it has moved beyond the edge of the encoded tape contents) the tape length is increasing. If this is the case and s is a power of 2, then we must double the counter value in order to satisfy Equation (4.3.3). This means the appendant in row 5 of Table 4.3.6 will be appended. The D in this appendant causes Table 4.3.7 to execute doubling

encoded object	encoded object length	initial marker index	index y of appendant	appendant α_y
$D = 0^{69}10^{2z}$	$2z + 70$	0	69	0^{2z-70}
$\langle 1, q_{k,L}\rangle = 0^{60k+50}10^{2z-60k-51}$	$2z$	70	$60k + 120$	$\langle 1, q_{k,L}\rangle$
$\langle a\rangle = 010^{2z-2}$	$2z$	70	71	$\langle a\rangle$
$\langle b\rangle = 0^2 10^{2z-3}$	$2z$	70	72	$\langle b\rangle$
$\langle \sigma_b\rangle = 0^3 10^{2z-4}$	$2z$	70	73	$\langle \sigma_b\rangle$
$\mu = 0^7 10^{z-8}$	z	70	77	$\mu\mu$
$\mu = 0^7 10^{z-8}$	z	$z + 70$	$z + 77$	$\mu\mu$
$\mu' = 0^8 10^{2z-9}$	$2z$	70	78	$\mu'\mu'$
$\not\mu = 0^9 10^{2z-10}$	$2z$	70	79	$\not\mu\not\mu$

Table 4.3.7: (Stage 3. Double counter). Each μ, μ' and $\not\mu$ indexes the appendants $\mu\mu$, $\mu'\mu'$ and $\not\mu\not\mu$ respectively.

the counter's value. Doubling the counter results in a single additional pass of the dataword. This process is illustrated in the last two configurations of Figure 4.3.2.

The computation of C_M remains similar when s is not a power of 2. If s is not a power of 2, and thus $s < s'$, then C_M enters Stage 3 via $\langle 3, q_{i,L,s<s'}\rangle$ instead of $\langle 3, q_{i,L}\rangle$.

As mentioned earlier the case for a right moving transition rule is covered in previous work [NW06a]. The execution of a right moving transition rule is similar to the execution of a left moving transition rule with the exception that we do not execute Tables 4.3.3 and 4.3.4.

We have shown how C_M simulates an arbitrary transition rule of M. To simulate halting C_M enters a repeating sequence of configurations. The halt state $q_{|Q|}$ is encoded in the normal way as $\langle 1, q_{|Q|,L}\rangle = 0^{60|Q|+50}10^{2z-60|Q|-51}$. We define the appendant at index $60|Q| + 50$ to be $\langle 1, q_{|Q|,L}\rangle$. Therefore $\langle 1, q_{|Q|,L}\rangle$ indexes a copy of itself. Also after $\langle 1, q_{|Q|,L}\rangle$ is read, each encoded tape symbol indexes a copy of itself. This causes C_M to enter a repeating sequence of configurations.

We have given a cyclic tag system C_M which simulates Turing machine M. The cyclic tag system algorithm, including the extra table entries that are required to simulate right move transition rules, has been subject to extensive computer testing. Note that simulating right moving transitions rules requires no more time and space than simulating left moving transition rules. Hence in giving the complexity analysis we can assume all transition rules are left moving.

Space analysis

At time t there are $O(t)$ encoded objects (state and symbols) in the dataword of C_M; each of length $O(|Q|)$. Thus C_M uses $O(|Q|t)$ space.

Time analysis

Simulating a transition rule involves 3 stages. Each stage executes in $O(|Q|t)$ steps. To simulate a single transition rule the counter is halved $O(\log t)$ times, (i.e. Stages 1 and 2 are executed $O(\log t)$ times) and Stage 3 is executed once. Thus $O(|Q|t \log t)$ time is sufficient to simulate a transition rule and $O(|Q|t^2 \log t)$ time is sufficient to simulate the computation of M.

This completes the proof of Theorem 4.3.2. □

A consequence of Theorem 4.3.2 is that Rule 110 simulates Turing machines in polynomial time. The weak machines in [Coo04, Wol02] simulate Rule 110 in quadratic time, which in turn (using Cook's construction with the 2-tag systems in [CM64]) simulates Turing machines in exponential time. We have improved this time bound to polynomial.

Corollary 4.3.3 *The small weakly universal Turing machines in [Coo04] simulate Turing machines in $O(t^4 \log^2 t)$ polynomial time.*

4.4 P-completeness of Rule 110

Finally we show that the reduction from the GENERIC MACHINE SIMULATION PROBLEM (GMSP) [GHR95] to RULE 110 PREDICTION is computable by a logspace transducer Turing machine. The GMSP is stated as: given a word w_M, an encoding $\langle M \rangle$ of a single-tape Turing machine M, and an integer t in unary, does M accept w_M within t steps?

Lemma 4.4.1 *The GMSP is logspace reducible to* RULE 110 PREDICTION.

Proof. We reduce the simulation problem for M to the analogous problem for cyclic tag systems. In the proof of Theorem 4.3.2 we showed how to construct C_M. We encode C_M as a word $\langle C_M \rangle$. The value z used in the proof of Theorem 4.3.2 is linear in $|Q|$, the number of states of M. There are $2z$ appendants, each of length $O(|Q|)$, giving an encoded program length of $O(|Q|^2)$. From Equation (4.3.2) the input $\langle w_M \rangle$ to C_M is of length $O(|Q||w_M|)$. Thus the encoded appendants and input are logspace constructable.

To show that a logspace transducer Turing machine generates a Rule 110 instance from $\langle C_M \rangle \# \langle w_M \rangle \#^t$ we examine Cook's Rule 110 simulation of cyclic tag systems [Coo04]. The input is written directly as the states of $O(|\langle w_M \rangle|)$ contiguous cells beginning at, say, cell p_0. On the left of the input

a constant word (representing Cook's 'ossifiers') is repeated $O(t)$ times. On the right the cyclic tag system program (list of appendants and 'leaders') is written $O(t)$ times. □

Since we already know that RULE 110 PREDICTION is in P, the proof of Theorem 4.1.2 is complete.

4.5 Discusion

We have proved that the prediction problem of Rule 110, one of the simplest cellular automata, is P-complete. In fact, we have established the stronger result of improving the simulation time of Turing machines by Rule 110 from exponential [Coo04, Wol02] to quadratic time. What about improving the simulation of Turing machines by Rule 110 to linear time? This result could be achieved by giving a direct simulation of Turing machines using Rule 110 or proving Rule 110 intrinsically universal [Oll02]. Another alternative would be to prove that cyclic tag systems simulate Turing machines in linear time. This however may not be possible. The simulation techniques of Turing machines using different variants of tag systems (bi-tag systems, 2-tag systems and cyclic tag systems) involve reading the entire input to simulate a single transition rule. Such a simulation technique implies a polynomial time simulation. It may not be possible to significantly improve on this technique.

The time efficiency of cyclic tag systems, established in this chapter, has already found applications. We have shown that 2-tag systems simulate cyclic tag systems [WN06a] and thus the small universal Turing machines of Minsky and Rogozhin et al. are efficient polynomial time simulators of Turing machines. The semi-weakly universal Turing machines in [WN07b, WNb] efficiently simulate cyclic tag systems, therefore these semi-weak machines are efficiently universal.

5

Tag systems

5.1 Introduction

In this chapter we give results on tag systems. We present a new form of tag system which we call bi-tag systems. Bi-tag systems were introduced to allow for the construction of small universal Turing machines. Prior to this all of the smallest universal Turing machines simulated Turing machines via 2-tag systems. A computation step of a bi-tag system is similar to that of a 2-tag system so many of the techniques used by small 2-tag simulators can be adapted to give small bi-tag simulators. In fact some of the smallest known universal Turing machines simulate bi-tag systems. We will encounter these machines in Chapter 6. In Section 5.2 we prove that bi-tag systems simulate Turing machine efficiently in polynomial time. This result has previously appeared in [NW05a].

In this chapter we also prove that 2-tag systems efficiently simulate Turing machines. We give a proof outline that 2-tag systems simulate Turing machines in polynomial time. As an immediate corollary of this result, we find that the small universal Turing machines of Minsky and Rogozhin et al. simulate Turing machines in polynomial time, an exponential improvement.

In the early 1960s, Cocke and Minsky [CM64] showed that 2-tag systems simulate Turing machines, but in an exponentially slow fashion. Minsky [Min62a] constructed a small 7-state, 4-symbol universal Turing machine that simulates 2-tag systems in polynomial time. Later, Rogozhin [Rog96] and others [Bai01, KR02, Rob91] used Minsky's simulation technique to find small universal Turing machines for a number of different state-symbol pairs. All of these small machines efficiently simulate 2-tag systems. However, since Cocke and Minsky's 2-tag simulation of Turing machines is exponentially slow it has remained an open problem as to whether these universal Turing machines can be made to run in polynomial time.

In Section 5.3 we give a brief outline of a proof that 2-tag systems efficiently simulate cyclic tag systems in polynomial time. This leads us to our second main result of this chapter, which is given by Theorem 5.1.1

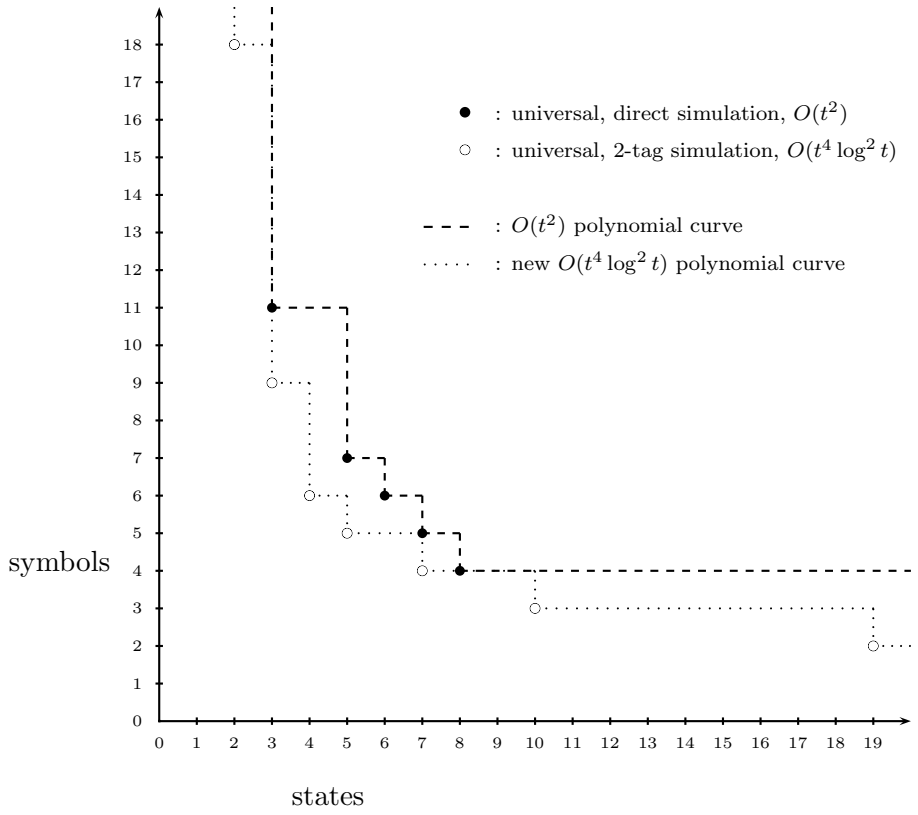


Figure 5.1.1: State-symbol plot of small universal Turing machines. The plot shows the improvement in the polynomial time curve following Corollary 5.1.2.

Theorem 5.1.1 *Let M be a deterministic Turing machine with a single tape that computes in time t, then there is a 2-tag system T_M that simulates the computation of M in time $O(t^2 \log t)$.*

This immediately gives the following interesting result.

Corollary 5.1.2 *The small universal Turing machines of Minsky, Rogozhin and others [Bai01, KR02, Rob91, Min62a, Rog96] are $O(t^4 \log^2 t)$ polynomial time simulators of Turing machines.*

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

Early examples of time efficient small universal Turing machines were found by Ikeno and Watanabe [Ike58, Wat61]. Prior to our result the small-

est known polynomial time universal Turing machines were given in [NW06c] (also to be found in Chapter 3). However, these time efficient machines are not as small as those of Rogozhin et al., hence our result represents a significant size improvement when considering small polynomial time universal Turing machines. This improvement is illustrated in Figure 5.1.1.

In Section 5.3 we prove that 2-tag systems simulate Turing machines in $O(t^2 \log t)$ polynomial time and give some applications of this result. The result we present here improves on the time overhead of the version presented in [WN06a, WN06b].

5.2 Bi-tag systems simulate Turing machines

5.2.1 Clockwise Turing machines simulate Turing machines

A clockwise Turing machine is a Turing machine that has a single tape, which is circular, and whose tape head moves only in a clockwise direction. The operation of clockwise Turing machines is quite similar to that of Kudlek and Rogozhin's [KR01b] Post machines.

Definition 5.2.1 (clockwise Turing machine [NW05a]) *A clockwise Turing machine is a tuple $C = (Q, \Sigma, f, q_1, q_{|Q|})$. Q and Σ are the finite sets of states and tape symbols, respectively. $q_1 \in Q$ is the start state and $q_{|Q|} \in Q$ is the halt state. The transition function $f : Q \times \Sigma \to \{\Sigma \cup \Sigma\Sigma\} \times Q$ is undefined on state $q_{|Q|}$ and is defined for all $q \in Q$, where $q \neq q_{|Q|}$.*

We write f as a list of clockwise transition rules. Each clockwise transition rule is a quadruple $t = (q_x, \sigma_1, v, q_y)$, with initial state q_x, read symbol σ_1, write value $v \in \{\Sigma \cup \Sigma\Sigma\}$ and next state q_y. A clockwise transition rule is executed as follows: If the write value v is from Σ then the tape cell containing the read symbol is overwritten by v, if v is from $\Sigma\Sigma$ then the cell containing the read symbol becomes 2 cells each of which contain one symbol from v. The machine's state becomes q_y and the tape head moves clockwise by one tape cell. Clockwise Turing machines are deterministic.

Lemma 5.2.2 *Let M be a deterministic Turing machine with a single tape that computes in time t, then there is a clockwise Turing machine C_M that simulates the computation of M in time $O(t^2)$ and space $O(t)$.*

Proof. Let N be a Turing machine that has the following restrictions: (i) the blank symbol σ_1 does not appear as input to N, (ii) N may read the blank symbol but is not permitted to write it to the tape, (iii) N has exactly one final state. Due to the restrictions placed on N we know that when N reads a blank symbol it is either at the left or right end of its tape contents. We wish to simulate $M = (\{q_1, \ldots, q_{|Q|}\}, \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}, \sigma_1, f, q_1, H\})$ a Turing machine without these restrictions. M is converted to a restricted

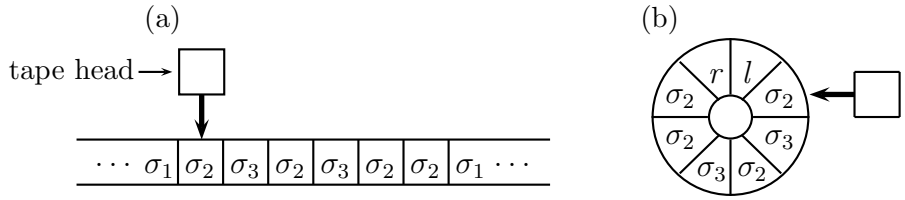


Figure 5.2.1: (a) Example Turing machine tape contents. The Turing machine's blank symbol is σ_1. (b) clockwise Turing machine encoding of the Turing machine tape contents in (a), the symbols r and l encode the infinite sequence of blank symbols to the right and left of M's encoded tape contents, respectively.

Turing machine N that requires one extra state and one extra symbol, and executes one extra computation step. We define Turing machine $N = (\{q_1, \ldots, q_{|Q|+1}\}, \{\sigma_1, \ldots, \sigma_{|\Sigma|+1}\}, \sigma_1, f_N, q_1, \{q_{|Q|+1}\})$. We construct a clockwise Turing machine $C_M = (Q_C, \Sigma_C, f_C, q_1, q_{|Q|+1})$ that simulates M via N, where Q_C, Σ_C, f_C are defined below.

$$\Sigma_C = \{\sigma_2, \ldots, \sigma_{|\Sigma|+1}, r, l, \gamma\}$$

The symbol γ is a special marker symbol and symbols r and l encode the infinite sequence of blank symbols to the right and left of N's encoded tape contents respectively (see Figure 5.2.1).

$$\begin{aligned} Q_C = \{&q_1, q_{1,2}, \ldots, q_{1,|\Sigma|+1}, q_{1,r}, q_{1,r'}, q_{1,l}, \\ &q_2, q_{2,2}, \ldots, q_{2,|\Sigma|+1}, q_{2,r}, q_{2,r'}, q_{2,l}, \\ &\vdots \\ &q_{|Q|}, q_{|Q|,2}, \ldots, q_{|Q|,|\Sigma|+1}, q_{|Q|,r}, q_{|Q|,r'}, q_{|Q|,l}, q_{|Q|+1}\} \end{aligned}$$

We can think of right moves of N's tape head as clockwise moves. Here we give right move transition rules followed by the clockwise transition rules that simulate them.

$$q_x, \sigma_k, \sigma_j, R, q_y \quad : \quad q_x, \sigma_k, \sigma_j, q_y \tag{5.2.1}$$

$$q_x, \sigma_1, \sigma_j, R, q_y \quad : \quad q_x, l, l\sigma_j, q_y \tag{5.2.2}$$

where $\sigma_k, \sigma_j \neq \sigma_1$. The clockwise transition rule in Equation (5.2.2) simulates N printing the write symbol σ_j over the blank symbol immediately to the left of its tape contents. The clockwise transition rule's write value $l\sigma_j \in \Sigma\Sigma$ also preserves l; the symbol that encodes the infinite sequence of blank symbols to the left of the tape contents.

The remaining right moving case is when N's tape head is over the blank symbol immediately to the right of its tape contents. In such a case C_M's

tape head is initially over r, and then immediately after simulation of the transition rule, C_M's tape head is again over r. Immediately below are the clockwise transition rules that simulate this case.

$$q_x, \sigma_1, \sigma_j, R, q_y \quad : \quad \begin{array}{ll} q_x, r, \sigma_j r, q_{y,r'} & (*) \\ q_{y,r'}, \kappa, \kappa, q_{y,r'} & (**) \end{array}$$

where $\kappa \in \Sigma_C - \{\gamma, r\}$. The clockwise transition rule (*) prints N's encoded write symbol σ_j and sends C_M's control into state $q_{y,r'}$. State $q_{y,r'}$ moves C_M's tape head to the cell containing r. This completes the simulation of the transition rule.

Left moving transition rules are more difficult to simulate as C_M's tape head moves only clockwise. C_M begins by marking the current location of the tape head with the symbol γ. C_M now moves each symbol clockwise by one cell. When C_M's tape head reads γ the left move is complete. This process moves the tape head anti-clockwise relative to the tape contents, thus simulating a left move. Immediately below is given the clockwise transition rules that mark the tape head's location with the symbol γ.

$$\begin{array}{lcl} q_x, \sigma_1, \sigma_j, L, q_y & : & q_x, l, l\gamma, q_{y,j} \\ q_x, \sigma_1, \sigma_j, L, q_y & : & q_x, r, \gamma\sigma_j, q_{y,r} \\ q_x, \sigma_k, \sigma_j, L, q_y & : & q_x, \sigma_k, \gamma, q_{y,j} \end{array}$$

The clockwise transition rules that move each symbol clockwise by one cell are of the form:

$$q_{y,\upsilon}, \rho, \upsilon, q_{y,\rho}$$

where $\upsilon, \rho \in \Sigma_C - \{\gamma\}$. When C_M's tape head reads γ then C_M is in a state of the form $q_{y,\rho}$ and the unique clockwise transition rule defined by the state-symbol pair $(q_{y,\rho}, \gamma)$ will begin simulation of the next transition rule. This transition rule is of the form $(q_y, \sigma_1, \sigma_k, D, q_z)$ if $\rho = r, l$ and of the form $(q_y, \rho, \sigma_k, D, q_z)$ if $\rho \neq r, l$.

Input to N is encoded for C_M by a finite state transducer. Given this encoded input C_M simulates the sequence of t transition rules in N's computation and halts in state $q_{|Q|+1}$ the encoding of N's halt state $q_{|Q|+1}$. C_M uses space of $O(t)$. A single computation step of N is simulated in $O(t)$ steps of C_M. Thus the computation time of C_M is $O(t^2)$. □

5.2.2 Bi-tag systems simulate clockwise Turing machines

In this section we present the bi-tag system, our new variant on the tag system, and prove that it simulates Turing machines via clockwise Turing machines. The operation of a bi-tag system is similar to that of a standard tag system [Min67]. Bi-tag systems are essentially 1-tag systems (and so they read and delete one symbol per timestep), augmented with additional context sensitive rules that read, and delete, two symbols per timestep.

Definition 5.2.3 (Bi-tag system) *A bi-tag system is a tuple* (A, E, e_h, P). *Here* A *and* E *are disjoint finite sets of symbols and* $e_h \in E$ *is the halt symbol.* P *is the finite set of productions. Each production is of one of the following 3 forms:*

$$P(a) = a, \quad P(e, a) \in AE, \quad P(e, a) \in AAE,$$

where $a \in A$, $e \in E$, *and* P *is defined on all elements of* $\{A \cup ((E - \{e_h\}) \times A)\}$ *and undefined on all elements of* $\{e_h\} \times A$. *Bi-tag systems are deterministic.*

A configuration of a bi-tag system is a word of the form $w = A^*(AE \cup EA)A^*$. We call w the dataword.

Definition 5.2.4 (BTS computation step) *A production is applied in one of two ways:*

(i) if $s = as'$ *then* $as' \vdash s'P(a)$,

(ii) if $s = eas'$ *then* $eas' \vdash s'P(e, a)$.

A bi-tag system computation is a finite sequence of computation steps that are consecutively applied to an initial dataword. If e_h is the leftmost symbol in the dataword then the computation halts.

Example 5.2.1 *(Bi-tag system computation.)* Let bi-tag system $B_1 = (\{a_0, a_1\}, \{e_0, e_1, e_2\}, e_2, P)$ where the set $P = \{a_0 \to a_0,\ a_1 \to a_1,\ e_0a_0 \to a_1e_0,\ e_0a_1 \to a_1e_2,\ e_1a_0 \to a_0e_0,\ e_1a_1 \to a_1e_2\}$. Given the word $a_1e_0a_0$, the computation of B_1 proceeds as follows:

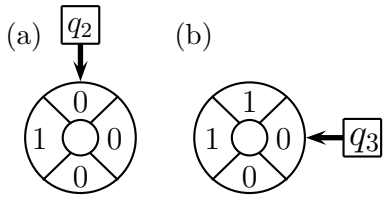
$$a_1e_0a_0 \vdash e_0a_0a_1 \vdash a_1a_1e_0 \vdash a_1e_0a_1 \vdash e_0a_1a_1 \vdash a_1a_1e_2 \vdash a_1e_2a_1 \vdash e_2a_1a_1$$

The computation halts because the halt symbol e_2 has become the leftmost symbol.

Lemma 5.2.5 *Let* C *be a clockwise Turing machine that runs in time* t, *then there is a bi-tag system* B_C *that simulates the computation of* C *in time* $O(t^2)$ *and space* $O(t)$.

Before giving the proof of Lemma 5.2.5 we explain the proof idea. Each A symbol of B_C encodes a symbol of C's tape alphabet. Each E symbol of B_C encodes a state of C. The location of the E symbol in the dataword represents the location of C's tape head, as illustrated in Figure 5.2.2.

Each clockwise transition rule of C is simulated in the following way. The change of state, symbol and tape head position is simulated by executing a P production over the $E \times A$ pair that encodes the current state and read symbol (see Figure 5.2.2(c)). A production is then applied to each symbol in the dataword. This moves the new $E \times A$ pair to the left of the dataword, in order to prepare for the simulation of the next clockwise transition rule.



(c) $e_2 0\ 0\ 0\ 1$ apply $e_2 0 \to 1 e_3$
$0\ 0\ 1\ 1\ e_3$ apply $0 \to 0$
$0\ 1\ 1\ e_3 0$ apply $0 \to 0$
$1\ 1\ e_3 0\ 0$ apply $1 \to 1$
$1\ e_3 0\ 0\ 1$ apply $1 \to 1$
(d) $e_3 0\ 0\ 1\ 1$ simulation complete

Figure 5.2.2: Bi-tag system simulating the clockwise transition rule $(q_2, 0, 1, q_3)$. The clockwise Turing machine states q_2 and q_3 are encoded as e_2 and e_3 respectively. The e symbols also mark the location of the simulated tape head. (a) Configuration of the clockwise Turing machine before execution of the clockwise transition rule. (b) Configuration of the clockwise Turing machine after execution of the clockwise transition rule. (c) bi-tag system encoding of configuration in (a). (d) Bi-tag system encoding of configuration in (b).

Proof. Let clockwise Turing machine C=$(\{q_1, \ldots, q_{|Q|}\}, \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}, f, q_1, q_{|Q|})$. We construct a bi-tag system B_C that simulates C's computation.

$$B_C = (A_C,\ E_C,\ e_{|Q|},\ P_C)$$

where A_C, E_C, P_C are defined below.

$$A_C = \{a_1, \ldots, a_{|\Sigma|}\}$$

C's tape symbols $\sigma_1, \ldots, \sigma_{|\Sigma|}$ are encoded as $a_1, \ldots, a_{|\Sigma|}$ respectively.

$$E_C = \{e_1, \ldots, e_{|Q|}\}$$

C's states $q_1, \ldots, q_{|Q|}$ are encoded as $e_1, \ldots, e_{|Q|}$ respectively and the encoded halt state $e_{|Q|}$ is the halt symbol of B_C.

$$P_C = \{a_1 \to a_1, \ldots, a_{|\Sigma|} \to a_{|\Sigma|}\} \cup P'_C$$

P'_C is the set of productions defined on $(E - \{e_{|Q|}\}) \times A$. There is one production in P'_C for each clockwise transition rule in C. Clockwise transition rules fall into two categories, those that write a single symbol from Σ and those that write a pair of symbols from $\Sigma\Sigma$. The two possible clockwise transition rules, and their encodings as productions, are as follows

$$\begin{array}{rcl} (q_x, \sigma_i, \sigma_j, q_y) & : & e_x a_i \to a_j e_y \\ (q_x, \sigma_i, \sigma_j \sigma_k, q_y) & : & e_x a_i \to a_j a_k e_y \end{array}$$

We have constructed a bi-tag system B_C that simulates C. B_C uses $O(t)$ space. To simulate a computation step of C, a production is applied

to each symbol in the dataword that encodes the current configuration of C. This takes $O(t)$ steps and yields a new dataword that encodes the next configuration of C's computation. In this way B_C simulates t steps of C's computation in time $O(t^2)$. The simulation halts when the halt symbol $e_{|Q|}$ that encodes the halt state of C becomes the leftmost symbol in the dataword. □

Given a single tape deterministic Turing machine M that runs in time t, we conclude from Lemmata 5.2.2 and 5.2.5 that M is simulated by a bi-tag system in time $O(t^4)$. However this overhead is easily improved to $O(t^3)$ as the next theorem shows.

Theorem 5.2.6 *Let M be a deterministic Turing machine with a single tape that computes in time t, then there is a bi-tag system B_M that simulates the computation of M in time $O(t^3)$ and space $O(t)$.*

Proof. From Lemmata 5.2.2 and 5.2.5 a bi-tag system simulates the computation of M via a clockwise Turing machine C_M. From Lemma 5.2.2 C_M simulates M in time $O(t^2)$. However, C_M uses $O(t)$ space and hence B_M uses $O(t)$ space. B_M applies $O(t)$ productions to simulate a clockwise transition rule of C_M. Thus B_M executes $O(t^2)$ clockwise transition rules to simulate M via C_M in time $O(t^3)$. □

5.3 Time complexity of 2-tag systems and small universal Turing machines

Tag systems where introducted by Post [Pos36] and proved universal by Minsky [Min61]. Later Cocke and Minsky [CM64] proved 2-tag systems universal.

Definition 5.3.1 (2-tag system) *A tag system consists of a finite alphabet of symbols Σ, a finite set of rules $R : \Sigma \to \Sigma^*$ 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. The computation of a 2-tag system acts on a dataword $w = \sigma_1\sigma_2 \dots \sigma_l$. 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 $\sigma_1 \to \sigma_{l+1} \dots \sigma_{l+c}$, then the word $\sigma_{l+1} \dots \sigma_{l+c}$ is appended. A dataword (configuration) w_2 is obtained from w_1 via a single computation step as follows:

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

where $\sigma_1 \to \sigma_{l+1} \dots \sigma_{l+c} \in R$. A 2-tag system completes its computation if (i) $|w| < \beta$, or (ii) there is no rule for the leftmost symbol.

In an earlier version of this work [WN06a] the efficiency of 2-tag systems was proved by the following chain of simulations

$$\text{Turing machine} \mapsto \text{Cyclic tag system} \mapsto \text{2-tag system} \tag{5.3.1}$$

In [NW06a] cyclic tag systems are proved efficient polynomial time simulators of Turing machines and in [WN06a] it is proved that 2-tag systems are efficient simulators of cyclic tag systems. Hence from the sequence of simulations in Equation (5.3.1) we get the Theorem 5.3.2.

Theorem 5.3.2 ([WN06a]) *Let M be a deterministic Turing machine with a single tape that computes in time t, then there is a 2-tag system T_M that simulates the computation of M in time $O(t^4 \log^2 t)$.*

In the sequel we eliminate the cyclic tag system in Equation (5.3.1) to give the following simulation:

$$\text{Turing machine} \mapsto \text{2-tag system} \tag{5.3.2}$$

This gives the $O(t^2 \log t)$ overhead in Theorem 5.1.1 whose proof is outlined below. The new proof outlined here uses ideas from the proof given in Section 4.3, that cyclic tag systems simulate Turing machines efficiently. The proof is also similar to the previous proof given in [WN06a] and makes use of many of the techniques therein. Hence the techniques used in [WN06a] and Section 4.3 may be applied to the outline given below in order to construct a detailed simulation.

Proof outline of Theorem 5.1.1

The bulk of the proof is concerned with simulating an arbitrary right moving transition rule $(q_y, \sigma_1, \sigma_j, R, q_z)$ of M in time $O(t \log t)$. The following equation encodes a Turing machine configuration as a 2-tag dataword:

$$\langle q_y, \underline{\sigma_1}\, \sigma_2\, \sigma_3\, \cdots\, \sigma_n \rangle \;=\; \underset{y}{x_1}\underset{y}{x_1}\, \underset{y}{x_2}\underset{y}{x_2}\, \underset{y}{x_3}\underset{y}{x_3}\, \cdots\, \underset{y}{x_n}\underset{y}{x_n}\, (aa)^{2^{\lceil \log_2(n) \rceil}} \tag{5.3.3}$$

where the current state of the Turing machine is q_y, the read symbol is σ_1 and the tape contents is $\sigma_1\sigma_2\sigma_3 \ldots \sigma_n$. Each $x_i x_i$ pair of 2-tag system symbols encoded the Turing machine symbol σ_i. Note that we decorate x_i symbols with underscripts and slashes. The 2-tag word $(aa)^{2^{\lceil \log_2(n) \rceil}}$ is used as a counter. Our 2-tag system algorithm has three stages given below.

Stages 1 and 2 (isolate encoded read symbol)

Stages 1 and 2 isolate the encoded read symbol $x_1 x_1$ which is the leftmost encoded Turing machine symbol in Equation (5.3.3). These stages make use of the tape-length counter $(aa)^{2^{\lceil \log_2(n) \rceil}}$. In Stage 1 every aa pair at an odd

numbered position is marked by changing it to $\not{a}\not{a}$ and then in Stage 2 every xx pair at an even numbered position is marked by changing it to $\not{x}\not{x}$. This process is iterated until all aa pairs are marked ($1 + \log_2 n$ iterations), this gives:

$$\underset{y}{x}_1\underset{y}{x}_1 \underset{0}{\not{x}}_2\underset{0}{\not{x}}_2 \underset{0}{\not{x}}_3\underset{0}{\not{x}}_3 \ \ldots\ \underset{0}{\not{x}}_n\underset{0}{\not{x}}_n \, (\not{a}\not{a})^{2^{\lceil \log_2(n) \rceil}}$$

In the configuration above the encoded read symbol has been isolated as it is the only unmarked encoded tape symbol.

Stages 3 (simulate transition rule)

Note in the initial configuration each encoded 2-tag symbol also carries the current state information q_y. Thus the encoded read symbol that was isolated during Stages 1 and 2 contains all the information required to print the encoded write symbol, establish the new encoded state, and simulate the right move. The rule $\underset{y}{x}_1 \rightarrow \underset{0}{\not{x}}_j\underset{0}{\not{x}}_j \, s^{2^z}$ encodes the write symbol σ_j and the current state q_z.

This rule is applied to give:

$$\underset{0}{\not{x}}_2\underset{0}{\not{x}}_2 \underset{0}{\not{x}}_3\underset{0}{\not{x}}_3 \ \ldots\ \underset{0}{\not{x}}_n\underset{0}{\not{x}}_n \, (\not{a}\not{a})^{2^{\lceil \log_2(n) \rceil}} \underset{0}{\not{x}}_j\underset{0}{\not{x}}_j \, s^{2^z}$$

The slashes are removed to give:

$$s^{2^z} \underset{0}{x}_2\underset{0}{x}_2 \underset{0}{x}_3\underset{0}{x}_3 \ \ldots\ \underset{0}{x}_n\underset{0}{x}_n \, aa^{2^{\lceil \log_2(n) \rceil}} \underset{0}{x}_j\underset{0}{x}_j$$

The dataword is read and the number of s symbols is halved to give:

$$s^{2^{z-1}} \underset{1}{x}_2\underset{1}{x}_2 \underset{1}{x}_3\underset{1}{x}_3 \ \ldots\ \underset{1}{x}_n\underset{1}{x}_n \, aa^{2^{\lceil \log_2(n) \rceil}} \underset{1}{x}_j\underset{1}{x}_j$$

Note that when the dataword is read each dataword symbol $\underset{i}{x}_v$ becomes $\underset{1+i}{x}_v$ this process is repeated, thus halving the number of s symbols with each iteration. This continues until only a single s symbol remains after z iterations. At this point each $\underset{i}{x}_v$ symbol in the dataword is of the form $\underset{z}{x}_v$ and contains the state information. Finally this s symbol is deleted from the dataword to give:

$$\underset{z}{x}_2\underset{z}{x}_2 \underset{z}{x}_3\underset{z}{x}_3 \ \ldots\ \underset{z}{x}_n\underset{z}{x}_n \, aa^{2^{\lceil \log_2(n) \rceil}} \underset{z}{x}_j\underset{z}{x}_j$$

The simulation of transition rule $(q_y, \sigma_1, \sigma_j, R, q_z)$ is now complete. A left moving transition rule may be simulated in a similar manner with some minor changes. Following simulation of a left moving transition rule the encoded read symbol will be the rightmost (instead of the leftmost) encoded read symbol. So in the initial dataword (i.e. Equation (5.3.3)) the encoded read symbol would be $\underset{y}{x}_n\underset{y}{x}_n$ instead of $\underset{y}{x}_1\underset{y}{x}_1$. Recall that Stage 2 marks

encoded tape symbols at even numbered positions. In order to make sure this rightmost encoded symbol $\underset{y}{x_n}\underset{y}{x_n}$ is not marked we must make sure it remains in an odd numbered position during Stage 2. Thus, if the encoded read symbol is at an even numbered position a dummy symbol is appended to place the encoded read symbol at an odd numbered position. A similar technique is used during Stage 2 in the proof of Theorem 4.3.2. Finally, if necessary, the counter value $aa^{2^{\lceil \log_2(n) \rceil}}$ is updated at the end of Stage 3, in a manner similar to that given in Lemma 3 of [WN06a]. This completes our proof outline.

Using our proof outline, we show that the 2-tag systems simulate Turing machines in $O(t^2 \log t)$ giving the result in Theorem 5.1.1. At timestep t the space used by M is bounded by t. This is encoded as $O(t)$ symbols in the dataword of T_M. Thus $O(t)$ space is sufficient to simulate M. Simulating a transition rule involves 3 stages. Each Stage executes in $O(t)$ steps. To simulate a single transition rule the counter is halved $O(\log t)$ times, (i.e. Stages 1 and 2 are executed $O(\log t)$ times) and Stage 3 is executed once. Thus $O(t \log t)$ time is sufficient to simulate a transition rule and $O(t^2 \log t)$ time is sufficient to simulate the computation of M.

5.3.1 Applications of efficient 2-tag systems

There are numerous applications of Theorem 5.1.1 besides the improved simulation time of the small universal Turing machines given in Corollary 5.1.2. Following our result, many biologically inspired models of computation now simulate Turing machines in polynomial time instead of exponential time. Some of these models include neural networks, H-systems and P-systems. For example, Siegelmann and Margenstern [SM99] give a neural network that uses only nine high-order neurons to simulate 2-tag systems. Taking each synchronous update of the nine neurons as a single parallel timestep, their neural network simulates 2-tag systems in linear time. They note that "tag systems suffer a significant slow-down ... and thus our result proves only Turing universality and should not be interpreted complexity-wise as a Turing equivalent." Our work shows that their neural network is in fact efficiently universal. Rogozhin and Verlan [RV06] give a tissue P-system with eight rules that simulates 2-tag systems in linear time, and thus we have improved its simulation time overhead from exponential to polynomial. This system uses splicing rules (from H-systems) with membranes (from P-systems) and is non-deterministic. Harju and Margenstern [HM05] gave an extended H-system with 280 rules that generates recursively enumerable sets using Rogozhin's 7-state, 4-symbol universal Turing machine. Using our result from 2-tag systems, the time efficiency of their construction is improved from exponential to polynomial, with a possible small constant increase in the number of rules. The efficiency of Hooper's [Hoo69] small 2-tape universal Turing machine is also improved from exponential to poly-

nomial. The technique of simulation via 2-tag systems is at the core of many of the universality proofs in Margenstern's survey [Mar00]. Our work exponentially improves the time overheads in these simulations, such as Lindgren and Nordahl's cellular automata [LN90], Margenstern's non-erasing Turing machines [Mar93, Mar95a], and Robinson's tiling [Rob71].

5.4 Discussion

We have seen from the previous section that our 2-tag system result has many applications. Bi-tag systems have applications in finding small universal Turing machine as we will see in Chapter 6. Also, bi-tag systems may be useful as an alternative to 2-tag systems to give universality results for other simple models.

As future work, it would be interesting to see if the time efficiency of 2-tag systems and bi-tag systems may be improved further. The different forms of tag systems including cyclic tag systems seem restricted in how they work on their dataword. Each symbol is read and the result is appended to the end of the dataword and thus, the entire configuration must be read before the appended result can be read. This is similar to placing the result at the bottom of a queue. A Turing machine seems less restricted in this way as it can repeatedly work on a section of its tape before moving to some other location. It remains to be seen if the time efficiency of these systems may be further improved.

6

Four small universal Turing machines

6.1 Introduction

In this chapter we present (standard) universal Turing machines with state-symbol pairs of $(5, 5)$, $(6, 4)$, $(9, 3)$ and $(15, 2)$. These are the smallest known universal machines with 5, 4, 3, and 2 symbols, respectively. Our machines simulate Turing machines via bi-tag systems, a form of tag system which we defined in Chapter 5. Each machine is plotted as a triangle in Figure 6.1.1.

The earliest small universal Turing machines simulated Turing machines directly [Ike58, Wat61]. Subsequently, the technique of indirect simulation, via 2-tag systems, was applied by Minsky [Min62a]. In 1962 Minsky [Min62a] constructed a 7-state, 4-symbol universal Turing machine that simulates Turing machines via 2-tag systems [CM64]. Minsky's technique of 2-tag simulation was extended by Rogozhin [Rog96, Rog98] to construct small universal Turing machines with state-symbol pairs of $(22, 2)$, $(10, 3)$, $(7, 4)$, $(5, 5)$, $(4, 6)$, $(3, 10)$ and $(2, 18)$. Subsequently, some of these machines were reduced in size to give machines with state-symbol pairs of $(3, 9)$ [KR02], $(19, 2)$ [Bai01] and $(7, 4)$ [Bai01]. The current smallest 2-tag simulators of Rogozhin et al. are plotted as hollow circles in Figure 6.1.1.

Our 5-symbol machine uses the same number of instructions (22) as the smallest known universal Turing machine (Rogozhin's 6-symbol machine [Rog96]). Also, our 5-symbol machine has less instructions than Rogozhin's 5-symbol machine. Since Minsky [Min62a] constructed his 7-state, 4-symbol machine, a number of authors [Bai01, Rob91, Rog96] have decreased the number of transition rules used for 4-symbol machines. However, the 6-state, 4-symbol machine we present here is the first reduction in the number of states.

The halting problem has been proved decidable for the following state-symbol pairs: $(2, 2)$ [Kud96, Pav73], $(3, 2)$ [Pav78], $(2, 3)$ (Pavlotskaya, unpublished), $(1, n)$ [Her68c], and $(n, 1)$ (trivial) for $n \geqslant 1$. These results induce the decidable halting problem curve given in Figure 6.1.1. Also, these decidability results imply that a universal Turing machine, that simulates any Turing machine M and halts if and only if M halts, is not possible

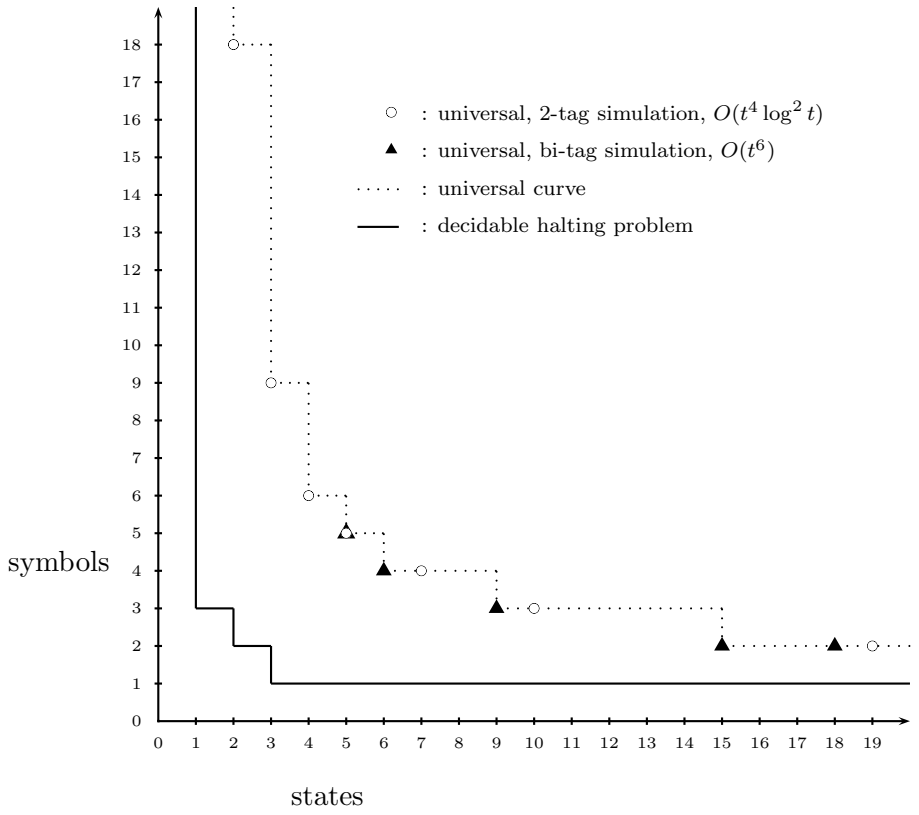


Figure 6.1.1: State-symbol plot of small universal Turing machines. Each of our new universal Turing machines is represented by a solid triangle.

for these state-symbol pairs. Hence these results give lower bounds on the size of universal machines of this type. The decidable halting problem curve in Figure 6.1.1 could be considered a non-universality curve in this sense. Following our results there are 39 state-symbol pairs that remain open.

Corollary 5.1.2, states that the universal Turing machines of Minsky and Rogozhin et al. simulate Turing machines in $O(t^4 \log^2 t)$ time. Our universal Turing machines in this chapter simulate bi-tag systems with a quadratic polynomial increase in time. Hence from Theorem 5.2.6 these universal Turing machines simulate Turing machines efficiently in $O(t^6(n))$ time.

Note that prior to constructing our 15-state, 2-symbol machine we constructed an 18-state, 2-symbol machine [NW07a] which is also plotted as a triangle in Figure 6.1.1. The results in the chapter appeared in [Nea06, NW07a, NW].

6.2 Universal Turing machines

In this section we give the input encoding to our universal Turing machines. Following this we give each machine and describe its operation by explaining how it simulates bi-tag systems. Bi-tag systems are defined in Section 5.2.2 of Chapter 5. Let $B = (A, E, e_h, P)$ be a bi-tag system where $A = \{a_1, \ldots, a_q\}$ and $E = \{e_1, \ldots, e_h\}$. The encoding of B as a word is denoted $\langle B\rangle$. The encodings of symbols $a \in A$ and $e \in E$ are denoted $\langle a\rangle$ and $\langle e\rangle$, respectively. The encodings of productions $P(a)$ and $P(e, a)$ are denoted as $\langle P(a)\rangle$ and $\langle P(e, a)\rangle$, respectively.

Definition 6.2.1 *The encoding of a configuration of B is of the form*

$$\ldots ccc\langle B\rangle S^* G(\langle A\rangle N)^* \Big(\langle A\rangle N\langle E\rangle \cup \langle E\rangle\langle A\rangle N\Big)(\langle A\rangle N)^* Dccc\ldots \tag{6.2.1}$$

where $\langle B\rangle$ is given by Equation (6.2.2)*, S and G are given by Table 6.2.1, and $(\langle A\rangle N)^*\Big(\langle A\rangle N\langle E\rangle \cup \langle E\rangle\langle A\rangle N\Big)(\langle A\rangle N)^* D$ encodes B's dataword via Table 6.2.1.*

$$\begin{aligned}\langle B\rangle =& H\langle P(e_{h-1}, a_q)\rangle V\langle P(e_{h-1}, a_{q-1})\rangle \ldots V\langle P(e_{h-1}, a_1)\rangle \\ & \vdots \\ & V\langle P(e_1, a_q)\rangle V\langle P(e_1, a_{q-1})\rangle \ldots V\langle P(e_1, a_1)\rangle \\ & V^2\langle P(a_q)\rangle V^2\langle P(a_{q-1})\rangle \ldots V^2\langle P(a_1)\rangle V^3\end{aligned} \tag{6.2.2}$$

where $\langle P\rangle$ for Turing machines $U_{9,3}$, $U_{5,5}$, $U_{6,4}$, and $U_{15,2}$ is given by Equations (6.2.3)*,* (6.2.4)*,* (6.2.5)*, and* (6.2.7)*, respectively, and V and H are given by Table 6.2.1. In Equation* (6.2.1) *the position of the tape head is over the rightmost symbol of G for $U_{15,2}$ and is over the symbol immediately to the right of $\langle B\rangle S^*$ for each of the other Turing machines. The initial state of each machine is u_1 and the blank symbol is c.*

	$\langle a_i\rangle$	$\langle e_j\rangle$	$\langle e_h\rangle$	S	G	N	D	V	H
$U_{5,5}$	b^{4i-1}	b^{4jq}	$b^{4hq+3}\delta$	d^2	ϵ	δ	ϵ	δ	$cd\delta$
$U_{6,4}$	b^{8i-5}	b^{8jq}	$b^{8q(h+1)+5}\delta$	g^2	ϵ	δ	b	δ	Eq (6.2.6)
$U_{9,3}$	b^{4i-1}	b^{4jq}	b^{4hq}	c^2	ϵ	δ	ϵ	δcc	$bccbc$
$U_{15,2}$	$(cb)^{8i-5}$	$(cb)^{8jq}$	$(cb)^{8hq+2}bb$	$(cc)^2$	bc	bb	ϵ	cb	$bbcc$

Table 6.2.1: Symbol values for Equations (6.2.1) and (6.2.2). The value of H for $U_{6,4}$ is given by Equation (6.2.6) in Section 6.2.4.

6.2.1 Universal Turing machine algorithm overview

Each of our universal Turing machines uses the same basic simulation algorithm. Here we give a brief description of the algorithm by explaining

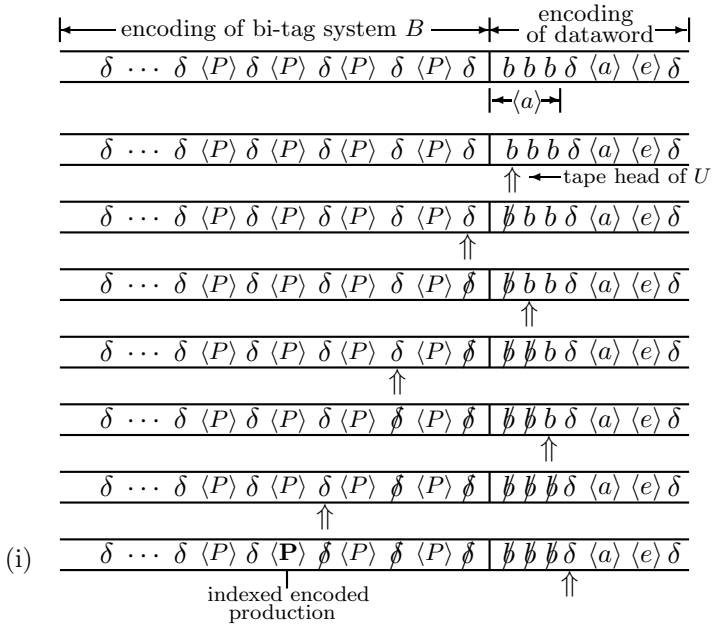


Figure 6.2.1: Indexing of an encoded production during simulation of a production of B. The encoded production $\langle P\rangle$, to be executed, is indexed by reading the leftmost encoded symbol $\langle a\rangle$ in the encoded dataword and marking off δ symbols in the encoding of B.

how our machines locate and simulate a production. The encoded production to be simulated is located using a unary indexing method as illustrated in Figure 6.2.1. The encoded production, $\langle P(a_i)\rangle$ or $\langle P(e_j, a_i)\rangle$ in Equation (6.2.2), is indexed (pointed to) by the number of symbols contained in the leftmost encoded symbol or pair of symbols in the encoded dataword (Equation (6.2.1)). For illustration purposes we assume that we are using $U_{9,3}$. If the leftmost encoded symbol is $\langle a_i\rangle = b^{4i-1}$ (Table 6.2.1) then the value $4i-1$ is used to index $\langle P(a_i)\rangle$. If the leftmost encoded symbol is $\langle e_j\rangle = b^{4jq}$, and $\langle a_i\rangle = b^{4i-1}$ is adjacent, then the value $4jq + 4i - 1$ is used to index $\langle P(e_j, a_i)\rangle$. The number of b symbols in the encoded symbol, or pair of encoded symbols, is equal to the number of δc^* words between the leftmost encoded symbol and the encoded production to be simulated. To locate this production, $U_{9,3}$ simply changes each δc^* to δb^*, for each b in the leftmost encoded symbol or pair of encoded symbols. This process continues until the δ that separates two encoded symbols in the dataword is read. Note from Equation (6.2.1) that there is no δ marker between each $\langle E\rangle$ and the $\langle A\rangle$ to its right, thus allowing $\langle e_j\rangle\langle a_i\rangle$ to be read together during indexing. After indexing, our machines print the indexed production immediately to the right of the encoded dataword as shown in Figure 6.2.2.

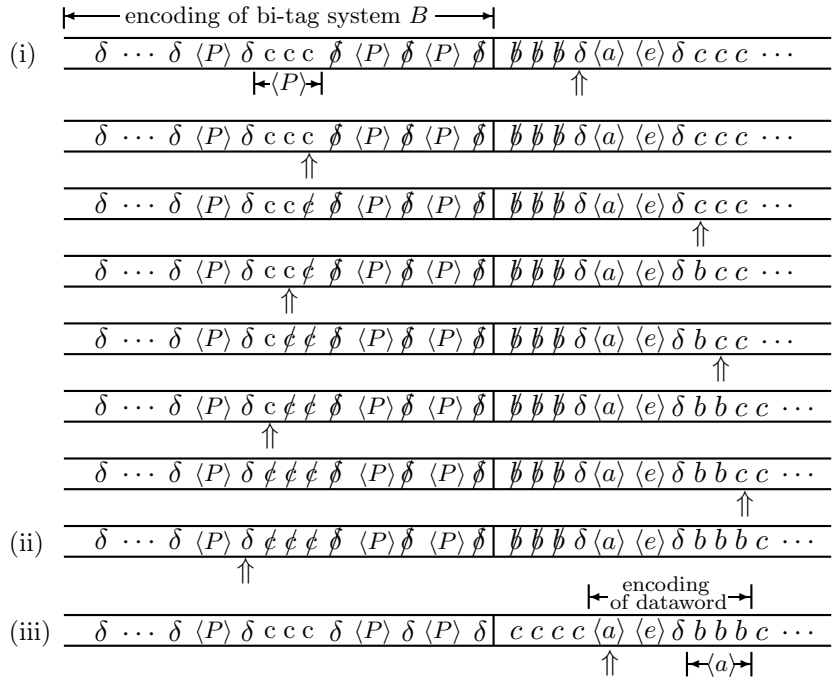


Figure 6.2.2: Printing of an encoded production during simulation of a production of B. Over a number of timesteps, the encoded production $\langle P \rangle$, indexed in configuration (i) of Figure 6.2.1, is printed to the right of the encoded dataword.

After the indexed production has been read, then $\langle B \rangle$, the encoding of B, is restored to its original value as illustrated in configurations (ii) and (iii) of Figure 6.2.2. This completes the simulation of the production. Extensive computer testing has been carried out on each of our universal Turing machines.

6.2.2 $U_{9,3}$

The following equation is used with Definition 6.2.1 to encode bi-tag system configurations for $U_{9,3}$.

$$\langle P \rangle = \begin{cases} \delta\delta cc\delta c^{8i} & \text{if } P(a_i) \\ \delta cc\delta\delta c^{8mq+2}\delta c^{8k} & \text{if } P(e_j, a_i) = a_k e_m \\ \delta\delta c^{8mq+2}\delta c^{8k}\delta c^{8v} & \text{if } P(e_j, a_i) = a_v a_k e_m \end{cases} \tag{6.2.3}$$

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9
c	bRu_1	cLu_3	cLu_3	bLu_9	cRu_6	bLu_4	δLu_4	cRu_7	bLu_5
b	cLu_2	cLu_2	bLu_4	bLu_4		bRu_6	bRu_7	cRu_9	cRu_8
δ	δRu_3	δLu_2	δRu_1	δLu_4	δLu_8	δRu_6	δRu_7	δRu_8	cRu_1

Table 6.2.2: Table of behaviour for $U_{9,3}$.

Example 6.2.1 ($U_{9,3}$ simulating the execution of the production $P(a_1)$) This example is presented using three cycles. The tape head of $U_{9,3}$ is given by an underline. The current state of $U_{9,3}$ is given to the left in bold. The dataword $a_1e_ja_i$ is encoded via Equation (6.2.1) and Table 6.2.1 as $bbb\delta b^{4jq}b^{4i-1}\delta$ and $P(a_1)$ is encoded via Equation (6.2.3) as $\langle P(a_1)\rangle = \delta\delta cc\delta c^8$. From Equation (6.2.1) we get the initial configuration:

$$\boldsymbol{u_1},\ \ldots\langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta cc\delta cc\delta cc\underline{b}bb\delta b^{4jq}b^{4i-1}\delta ccc\ldots$$

Cycle 1 (Index next production). In Cycle 1 (Table 6.2.3), $U_{9,3}$ reads the leftmost encoded symbol and locates the next encoded production to execute (see Figure 6.2.1). $U_{9,3}$ scans right until it reads b in state u_1. Then $U_{9,3}$ scans left in states u_2 and u_3 until it reads the subword δc^*. This subword is changed to δb^* as $U_{9,3}$ scans right in states u_1 and u_3. The process is repeated until $U_{9,3}$ reads b in state u_3. This indicates that we have finished reading the leftmost encoded symbol, or pair of encoded symbols, and that the encoded production to be executed has been indexed. This signals the end of Cycle 1 and the beginning of Cycle 2.

	u_1	u_2	u_3
c	bRu_1	cLu_3	cLu_3
b	cLu_2	cLu_2	bLu_4
δ	δRu_3	δLu_2	δRu_1

Table 6.2.3: Cycle 1 of $U_{9,3}$.

	u_4	u_5	u_6	u_7	u_8	u_9
c	bLu_9	cRu_6	bLu_4	δLu_4	cRu_7	bLu_5
b	bLu_4		bRu_6	bRu_7		
δ	δLu_4	δLu_8	δRu_6	δRu_7	δRu_8	

Table 6.2.4: Cycle 2 of $U_{9,3}$.

$$\begin{aligned}
\vdash\quad & \boldsymbol{u_2},\ \ldots\langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta cc\delta cc\delta c\underline{c}cbb\delta b^{4jq}b^{4i-1}\delta ccc\ldots\\
\vdash^2\quad & \boldsymbol{u_3},\ \ldots\langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta cc\delta cc\underline{\delta}cccbb\delta b^{4jq}b^{4i-1}\delta ccc\ldots\\
\vdash^4\quad & \boldsymbol{u_1},\ \ldots\langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta cc\delta cc\delta bbb\underline{b}b\delta b^{4jq}b^{4i-1}\delta ccc\ldots\\
\vdash^{44}\quad & \boldsymbol{u_1},\ \ldots\langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta bb\delta bb\delta bbbbb\underline{\delta}b^{4jq}b^{4i-1}\delta ccc\ldots\\
\vdash^2\quad & \boldsymbol{u_4},\ \ldots\langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8\delta bb\delta bb\delta bbbbb\underline{\delta}b^{4jq}b^{4i-1}\delta ccc\ldots
\end{aligned}$$

In the configuration immediately above the encoded production $\langle P(a_1)\rangle$ has been indexed and we have entered Cycle 2.

Cycle 2 (Print production). Cycle 2 (Table 6.2.4) prints the encoded production, that was indexed in Cycle 1, immediately to the right of the

encoded dataword (see Figure 6.2.2). $U_{9,3}$ scans left in state u_4 and records the next symbol of the encoded production to be printed. If $U_{9,3}$ reads the subword ccc it enters state u_6, scans right, and prints b at the right end of the encoded dataword. A single b is printed for each cc pair that does not have δ immediately to its left. If $U_{9,3}$ reads the subword $c\delta cc$ it scans right in state u_7 and prints δ at the right end of the encoded dataword. This process is repeated until the end of the encoded production is detected by reading the subword $\delta\delta cc$ which causes $U_{9,3}$ to enter Cycle 3.

$$\begin{aligned}
&\vdash^{13} && \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^6 c\underline{c}(\delta bb)^3 bbb\delta b^{4jq}b^{4i-1}\delta ccc\ldots\\
&\vdash^{3} && \boldsymbol{u_6}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^6 \underline{b}\underline{b}(\delta bb)^3 bbb\delta b^{4jq}b^{4i-1}\delta ccc\ldots\\
&\vdash^{4(jq+i)+15} && \boldsymbol{u_6}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^6 bb(\delta bb)^3 bbb\delta b^{4jq}b^{4i-1}\delta \underline{c}ccc\ldots\\
&\vdash && \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^6 bb(\delta bb)^3 bbb\delta b^{4jq}b^{4i-1}\underline{\delta}bccc\ldots
\end{aligned}$$

In the configuration immediately above the first symbol of the encoded production $\langle P(a_1)\rangle$ has been printed. Following the printing of the final symbol of the encoded production we get:

$$\begin{aligned}
&\vdash^{*} && \boldsymbol{u_4}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta c\underline{c}\delta b^8(\delta bb)^3 bbb\delta b^{4jq}b^{4i-1}\delta b^3\delta ccc\ldots\\
&\vdash^{3} && \boldsymbol{u_8}, \ldots \langle P(a_2)\rangle(\delta cc)^2\underline{\delta}\delta bb\delta b^8(\delta bb)^3 bbb\delta b^{4jq}b^{4i-1}\delta b^3\delta ccc\ldots
\end{aligned}$$

In the configuration immediately above we have finished printing the encoded production $\langle P(a_1)\rangle$ to the right of the dataword and we have entered Cycle 3.

Cycle 3 (Restore tape). Cycle 3 (Table 6.2.5) restores $\langle B\rangle$ to its original value (see configurations (ii) and (iii) of Figure 6.2.2). The tape head of $U_{9,3}$ scans right switching between states u_8 and u_9 changing b symbols to c symbols. This continues until $U_{9,3}$ reads the δ marking the leftmost end of the dataword in u_9. Note from Equations (6.2.2) and (6.2.3) that there is an even number of b symbols between each pair of δ symbols in $\langle B\rangle$ hence each δ symbol in $\langle B\rangle$ will be read in state u_8. Each a_i symbol in the dataword is encoded by an odd number of b symbols ($\langle a_i\rangle = b^{4i-1}$) and hence the first δ symbol in the dataword will be read in state u_9. This δ symbol marks the left end of the new dataword and causes $U_{9,3}$ to enter state u_1 thus completing Cycle 3 and the production simulation.

	u_8	u_9
b	cRu_9	cRu_8
δ	δRu_8	cRu_1

Table 6.2.5: Cycle 3 of $U_{9,3}$.

$$\begin{aligned}
&\vdash^{25} && \boldsymbol{u_9}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8(\delta cc)^3 ccc\underline{\delta}b^{4jq}b^{4i-1}\delta b^3\delta ccc\ldots\\
&\vdash && \boldsymbol{u_1}, \ldots \langle P(a_2)\rangle(\delta cc)^2\delta\delta cc\delta c^8(\delta cc)^3 cccc\underline{b}b^{4jq-1}b^{4i-1}\delta b^3\delta ccc\ldots
\end{aligned}$$

In the configuration immediately above our example simulation of production $P(a_1)$ is complete.

Theorem 6.2.2 *Given a bi-tag system B that runs in time t the computation of B is simulated by $U_{9,3}$ in time $O(t^2)$.*

Proof. In order to prove the correctness of $U_{9,3}$ we prove that $U_{9,3}$ simulates any possible $P(a)$ or $P(e,a)$ production of an arbitrary bi-tag system and, that $U_{9,3}$ also simulates halting when the encoded halt symbol $\langle e_h \rangle$ is encountered. In Example 6.2.1 $U_{9,3}$ simulates $P(a_1)$ for an arbitrary bi-tag system where a_1 is the leftmost symbol in a fixed dataword. This example easily generalises to any production $P(a_i)$ where a_i is the leftmost symbol in an arbitrary dataword. When some $e \in E$ is the leftmost symbol in the dataword then some production $P(e,a)$ must be executed. The simulation of $P(a_1)$ in Example 6.2.1 is also used to verify the simulation of $P(e,a)$. Note from Equation (6.2.1) that there is no δ marker between each $\langle e_j \rangle$ and the adjacent $\langle a_i \rangle$ to its right, thus $\langle e_j \rangle$ and $\langle a_i \rangle$ are read together during Cycle 1. Using the encoding in Definition 6.2.1, the number of b symbols in $\langle e_j \rangle \langle a_i \rangle$ indexes $\langle P(e,a) \rangle$. Thus, the indexing of $\langle P(e,a) \rangle$ is carried out in the same manner as the indexing of $\langle P(a) \rangle$. The printing of production $\langle P(e,a) \rangle$ during Cycle 2 and the subsequent restoring of $\langle B \rangle$ during Cycle 3 proceed in the same manner as with $P(a_1)$.

If the encoded halt symbol $\langle e_h \rangle = b^{4hq}$ is the leftmost symbol in the encoded dataword, and $\langle a_i \rangle = b^{4-i}$ is adjacent, this is encoded via Definition 6.2.1 as follows:

$$\boldsymbol{u_1},\ bccbc\langle P(e_{h-1},a_q)\rangle \ldots \langle P(a_1)\rangle(\delta cc)^3(cc)^*\underline{b}b^{4hq-1}b^{4i-1}\delta(\langle A\rangle\delta)^*ccc\ldots$$

During Cycle 1, immediately after reading the $(4hq+3)^{\text{th}}$ b symbol in the dataword, $U_{9,3}$ scans left in u_2 and we get the following:

$$\begin{aligned}
&\vdash^* \ \boldsymbol{u_2},\ bccb\underline{c}\langle P(e_{h-1},a_q)\rangle \ldots \langle P(a_1)\rangle(\delta cc)^3(cc)^*c^{4hq+3}b^{4i-4}\delta(\langle A\rangle\delta)^*ccc\ldots\\
&\vdash^4 \ \boldsymbol{u_5},\ \underline{b}bbbc\langle P(e_{h-1},a_q)\rangle \ldots \langle P(a_1)\rangle(\delta cc)^3(cc)^*c^{4hq+3}b^{4i-4}\delta(\langle A\rangle\delta)^*ccc\ldots
\end{aligned}$$

There is no transition rule in Table 6.2.2 for the case 'when in u_5 read b', hence the computation halts. □

The proof of correctness given for $U_{9,3}$ can be applied to the remaining machines in a straightforward way, so we do not restate it.

6.2.3 $U_{5,5}$

The following equation is used with Definition 6.2.1 to encode bi-tag system configurations for $U_{5,5}$.

$$\langle P\rangle = \begin{cases} \delta\delta d^{16i-6} & \text{if } P(a_i) \\ \delta\delta d^{16mq}\delta d^{16k-6} & \text{if } P(e_j,a_i) = a_k e_m \\ \delta\delta d^{16hq+14}\delta d^{16k-6} & \text{if } P(e_j,a_i) = a_k e_h \\ \delta d^{16mq}\delta d^{16k-2}\delta d^{16v-6} & \text{if } P(e_j,a_i) = a_v a_k e_m \\ \delta d^{16hq+14}\delta d^{16k-2}\delta d^{16v-6} & \text{if } P(e_j,a_i) = a_v a_k e_h \end{cases} \tag{6.2.4}$$

where $e_m \neq e_h$.

	u_1	u_2	u_3	u_4	u_5
g	bLu_1	gRu_1	bLu_3		
b	gLu_1	gRu_2	dRu_5	gRu_4	dRu_3
δ	cRu_2	cRu_2	δRu_3	cRu_4	dRu_1
c	δLu_1	bLu_3	δLu_3	δLu_3	
d	bLu_1	gRu_2	bLu_5	bLu_2	bLu_4

Table 6.2.6: Table of behaviour for $U_{5,5}$.

The dataword $a_1e_ja_i$ is encoded via Equation (6.2.1) and Table 6.2.1 as $bbb\delta b^{4jq}b^{4i-1}\delta$, and $P(a_1)$ is encoded via Equation (6.2.4) as $\langle P(a_1)\rangle = \delta\delta d^{10}$. From Equation (6.2.1) we get the initial configuration:

$$\boldsymbol{u_1}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}\delta\delta\delta\underline{b}bb\delta b^{4jq}b^{4i-1}\delta ccc\ldots$$

Cycle 1 (Index next production). In Cycle 1 (Table 6.2.7) when $U_{5,5}$ reads b in state u_1, it changes it to g and scans left until it reads δ. This δ is changed to c and $U_{5,5}$ then enters state u_2 and scans right until it reads g which causes it to re-enter state u_1. This process is repeated until $U_{5,5}$ reads the δ that separates a pair of encoded symbols in the encoded dataword. This signals the end of Cycle 1 and the beginning of Cycle 2.

	u_1	u_2
g	bLu_1	gRu_1
b	gLu_1	gRu_2
δ	cRu_2	cRu_2
c	δLu_1	
d	bLu_1	

Table 6.2.7: Cycle 1 of $U_{5,5}$.

	u_2	u_3	u_4	u_5
g		bLu_3		
b	gRu_2		gRu_4	
δ	cRu_2	δRu_3	cRu_4	
c	bLu_3	δLu_3	δLu_3	
d	gRu_2	bLu_5	bLu_2	bLu_4

Table 6.2.8: Cycle 2 of $U_{5,5}$.

$$\begin{aligned}
\vdash^{3} \quad & \boldsymbol{u_1}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}\delta\delta cg\underline{b}b\delta b^{4jq}b^{4i-1}\delta ccc\ldots \\
\vdash^{18} \quad & \boldsymbol{u_1}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}cccggg\underline{\delta}b^{4jq}b^{4i-1}\delta ccc\ldots \\
\vdash \quad & \boldsymbol{u_2}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}cccgggc\underline{b}b^{4jq-1}b^{4i-1}\delta ccc\ldots
\end{aligned}$$

Cycle 2 (Print production). Cycle 2 (Table 6.2.8) begins with $U_{5,5}$ scanning right and printing b to the right of the encoded dataword. Following this $U_{5,5}$ scans left in state u_3 and records the next symbol of the encoded production to be printed. If $U_{5,5}$ reads the subword $dddd$ it enters state u_2, scans right, and prints b at the right end of the encoded dataword. If $U_{5,5}$ reads the subword δdd it scans right in state u_4 and prints δ at the right end of the encoded dataword. This process is repeated until the end of the encoded production is detected by reading δ in state u_3, which causes $U_{5,5}$ to enter Cycle 3.

$$\begin{aligned}
\vdash^{*} \quad & \boldsymbol{u_3}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^6ddd\underline{d}\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bccc\ldots \\
\vdash^{3} \quad & \boldsymbol{u_2}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^6\underline{d}bbb\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bccc\ldots \\
\vdash^{*} \quad & \boldsymbol{u_3}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d\underline{d}b^8\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bbbccc\ldots \\
\vdash^{2} \quad & \boldsymbol{u_4}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\underline{\delta}bbb^8\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bbbccc\ldots \\
\vdash^{*} \quad & \boldsymbol{u_3}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\underline{\delta}\delta bbb^8\delta\delta\delta bbb\delta b^{4jq}b^{4i-1}\delta bbb\delta ccc\ldots
\end{aligned}$$

Cycle 3 (Restore tape). In Cycle 3 (Table 6.2.9) the tape head of $U_{5,5}$ scans right switching between states u_3 and u_5 changing b symbols to d symbols. This continues until $U_{5,5}$ reads the δ marking the leftmost end of the encoded dataword in u_5. Note from Equations (6.2.4) and (6.2.2) that there is an even number of d symbols between each pair of δ symbols in $\langle B\rangle$ hence each δ symbol in $\langle B\rangle$ will be read in state u_3. Each a_i symbol in the dataword is encoded by an odd number of symbols ($\langle a_i\rangle = b^{4i-1}$) and hence the first δ symbol in the dataword will be read in state u_5. This causes $U_{5,5}$ to enter state u_1 thus completing Cycle 3 and the production simulation.

	u_3	u_5
b	dRu_5	dRu_3
δ	δRu_3	dRu_1

Table 6.2.9: Cycle 3 of $U_{5,5}$.

$$\vdash^{19} \quad \boldsymbol{u_1}, \ldots \delta^2\langle P(a_2)\rangle\delta^2\delta\delta d^{10}\delta\delta\delta dddd\underline{b}b^{4jq-1}b^{4i-1}\delta bbb\delta ccc\ldots$$

Halting for $\mathbf{U_{5,5}}$. If the encoded halt symbol $\langle e_h\rangle = b^{4hq+3}\delta$ is the leftmost symbol in the encoded dataword then this is encoded via Definition 6.2.1 as follows:

$$\boldsymbol{u_1},\ cd\delta\langle P(e_{h-1}, a_q)\rangle\delta\ldots\delta^2\langle P(a_1)\rangle\delta^3(dd)^*\underline{b}b^{4hq+2}\delta(\langle A\rangle\delta)^*ccc\ldots$$

The computation continues as before until $U_{5,5}$ enters Cycle 2 and scans left in u_3. Immediately after $U_{5,5}$ reads the leftmost d during this leftward scan we get:

$$\vdash \quad \boldsymbol{u_5}, \underline{c}b\delta\langle P(e_{h-1}, a_q)\rangle'\delta \dots \delta^2\langle P(a_1)\rangle'\delta^3(dd)^* b^{4hq+3}\delta(\langle A\rangle\delta)^* bccc\dots$$

In the configuration above, $\langle P\rangle'$ denotes the word in which all the d symbols in $\langle P\rangle$ are changed to b symbols. There is no transition rule in Table 6.2.6 for the case 'when in u_5 read c' hence the computation halts.

6.2.4 $U_{6,4}$

The following equation is used with Definition 6.2.1 to encode bi-tag system configurations for $U_{6,4}$.

$$\langle P\rangle = \begin{cases} \delta^5 g^{12i-10}\delta & \text{if } P(a_i) \\ \delta^4 g^{12mq}\delta\delta g^{12k-10}\delta & \text{if } P(e_j, a_i) = a_k e_m \\ \delta^4 g^{12q(h+1)+8}\delta\delta g^{12k-10}\delta & \text{if } P(e_j, a_i) = a_k e_h \\ \delta^2 g^{12mq}\delta\delta g^{12hq+12k-4}\delta\delta g^{12v-10}\delta & \text{if } P(e_j, a_i) = a_v a_k e_m \\ \delta^2 g^{12q(h+1)+8}\delta\delta g^{12hq+12k-4}\delta\delta g^{12v-10}\delta & \text{if } P(e_j, a_i) = a_v a_k e_h \end{cases} \tag{6.2.5}$$

where $e_m \neq e_h$.

	u_1	u_2	u_3	u_4	u_5	u_6
g	bLu_1	gRu_1	bLu_3	bRu_2	bLu_6	bLu_4
b	gLu_1	gRu_2	bLu_5	gRu_4	gRu_6	gRu_5
δ	cRu_2	cRu_2	δLu_5	cRu_4	δRu_5	gRu_1
c	δLu_1	gRu_5	δLu_3	cRu_5	bLu_3	

Table 6.2.10: Table of behaviour for $U_{6,4}$.

The dataword $a_1e_ja_i$ is encoded via Equation (6.2.1) and Table (6.2.1) as $bbb\delta b^{8jq}b^{8i-5}\delta b$. From Equation (6.2.1) we get the initial configuration:

$$\boldsymbol{u_1}, \dots \delta^2\langle P(a_2)\rangle\delta^2\langle P(a_1)\rangle\delta\delta\delta\underline{b}bb\delta b^{8jq}b^{8i-5}\delta bccc\dots$$

Cycle 1 (Index next production). In Cycle 1 (Table 6.2.11) when $U_{6,4}$ reads b in state u_1 it scans left until it reads δ. This δ is changed to c and $U_{6,4}$ then enters state u_2 and scans right until it reads g which causes it to re-enter state u_1. This process is repeated until $U_{6,4}$ reads the δ that separates a pair of encoded symbols in the encoded dataword. This signals the end of Cycle 1 and the beginning of Cycle 2.

Cycle 2 (Print production). Cycle 2 (Table 6.2.12) begins with $U_{6,4}$ scanning right and printing bb to the right of the encoded dataword. Following this, $U_{6,4}$ scans left in state u_3 and records the next symbol of the

	u_1	u_2
g	bLu_1	gRu_1
b	gLu_1	gRu_2
δ	cRu_2	cRu_2
c	δLu_1	

Table 6.2.11: Cycle 1 of $U_{6,4}$.

	u_2	u_3	u_4	u_5	u_6
g		bLu_3	bRu_2	bLu_6	bLu_4
b	gRu_2	bLu_5	gRu_4		
δ	cRu_2	δLu_5	cRu_4	δRu_5	
c	gRu_5	δLu_3	cRu_5	bLu_3	

Table 6.2.12: Cycle 2 of $U_{6,4}$.

encoded production to be printed. If $U_{6,4}$ reads the subword $ggg\delta$ or $gggb$ it enters state u_2, scans right, and prints bb at the right end of the encoded dataword. If $U_{6,4}$ reads the subword δggb it scans right in state u_4 and prints δb at the right end of the encoded dataword. This process is repeated until the end of the encoded production is detected by reading δ in state u_5, which causes $U_{6,4}$ to enter Cycle 3.

Cycle 3 (Restore tape). In Cycle 3 (Table 6.2.13) the tape head of $U_{6,4}$ scans right switching between states u_5 and u_6, changing b symbols to g symbols. This continues until $U_{6,4}$ reads the δ marking the leftmost end of the encoded dataword in u_6. Note from Equations (6.2.5) and (6.2.2) that there is an even number of g symbols between each pair of δ symbols in $\langle B\rangle$, hence each δ symbol in $\langle B\rangle$ is read in state u_5. Each a_i symbol in the dataword is encoded by an odd number of symbols ($\langle a_i\rangle = b^{8i-5}$) and hence the first δ symbol in the dataword is read in state u_6. This causes $U_{6,4}$ to enter state u_1, thus completing Cycle 3 and the production simulation.

	u_5	u_6
b	gRu_6	gRu_5
δ	δRu_5	gRu_1

Table 6.2.13: Cycle 3 of $U_{6,4}$.

Special case for $U_{6,4}$. If we are simulating a production of the form $P(e,a) = a_v a_k e_m$ we have a special case. Note from the fourth row of Equation 6.2.5 and Cycle 2 that the simulation of $P(e,a) = a_v a_k e_m$ for $U_{6,4}$ results in the word $b^{8v-5}\delta b^{8hq+8k-3}\delta b^{8mq}b$ being printed to the right of the encoded dataword. From Table 6.2.1 it is clear that a_k is not encoded in this word in its usual from. However when $U_{6,4}$ reads the subword $b^{8hq+8k-3}\delta$ it indexes $\langle P(a_k)\rangle$ in H which results in $\langle a_k\rangle$ being printed to the dataword. To see this, note that the value of H from Equation (6.2.2) for $U_{6,4}$ is as follows:

$$H = cgbV^2\langle P(a_q)\rangle V^2\langle P(a_{q-1})\rangle \dots V^2\langle P(a_1)\rangle V^3 \tag{6.2.6}$$

The halting condition for $U_{6,4}$ occurs in a similar manner to that of $U_{5,5}$. Halting occurs during the first scan left in Cycle 2 when $U_{6,4}$ reads c in state

u_6 at the left end of H (note from Table 6.2.10 that there is no transition rule for state-symbol pair (u_6, c)).

6.2.5 $U_{15,2}$

The following equation is used with Definition 6.2.1 to encode bi-tag system configurations for $U_{15,2}$.

$$\langle P \rangle = \begin{cases} (cb)^4(cccb)^2(cc)^{8i-5} & \text{if } P(a_i) \\ (cb)^5(cc)^{8mq}(cccb)^2(cc)^{8k-5} & \text{if } P(e_j, a_i) = a_k e_m \\ (cb)^3(cccb)^2(cc)^{8hq+2}(cccb)^2(cc)^{8k-5} & \text{if } P(e_j, a_i) = a_k e_h \\ (cb)^3(cc)^{8mq}(cccb)^2(cc)^{8k-5}(cccb)^2(cc)^{8v-5} & \text{if } P(e_j, a_i) = a_v a_k e_m \\ cb(cccb)^2(cc)^{8hq+2}(cccb)^2(cc)^{8k-5}(cccb)^2(cc)^{8v-5} & \\ & \text{if } P(e_j, a_i) = a_v a_k e_h \end{cases} \tag{6.2.7}$$

where $e_m \neq e_h$.

$U_{15,2}$	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
c	cRu_2	bRu_3	cLu_7	cLu_6	bRu_1	bLu_4	cLu_8	bLu_9
b	bRu_1	bRu_1	cLu_5	bLu_5	bLu_4	bLu_4	bLu_7	bLu_7
$U_{15,2}$	u_9	u_{10}	u_{11}	u_{12}	u_{13}	u_{14}	u_{15}	
c	cRu_1	bLu_{11}	cRu_{12}	cRu_{13}	cLu_2	cLu_3	cRu_{14}	
b	bLu_{10}		bRu_{14}	bRu_{12}	bRu_{12}	cRu_{15}	bRu_{14}	

Table 6.2.14: Table of behaviour for $U_{15,2}$.

Example 6.2.2 ($U_{15,2}$ simulating the execution of the production $P(a_1)$) The example dataword $a_1 e_j a_i$ is encoded via Equation (6.2.1) and Table (6.2.1) as $cbcbcbbb(cb)^{8jq}(cb)^{8i-5}bb$ and $P(a_1)$ is encoded via Equation 6.2.7 as $\langle P(a_1) \rangle = (cb)^4(cccb)^2(cc)^3$. Thus from Equation (6.2.1) we get the following initial configuration

$$\boldsymbol{u_1}, \ldots \langle P(a_2) \rangle (cb)^6(cccb)^2(cc)^3 cb\ cb\ cb\ b\underline{c}\ cb\ cb\ cb\ bb(cb)^{8jq+8i-5}bb\ c \ldots$$

In this example we explain how $U_{15,2}$ operates by considering how it treats pairs of symbols during each cycle. Thus, the extra whitespace between each pair of symbols is to improve readability and help illustrate our explanation of $U_{15,2}$'s operation.

Cycle 1 (Index next production). In Cycle 1 (Table 6.2.15) $U_{15,2}$ scans right in states u_1, u_2 and u_3 until it reads the subword ccb which it changes

to *cbc*. Following this, it scans left in states u_4, u_5 and u_6 until it reads the subword *cb*. This *cb* is changed to *bb* and $U_{15,2}$ re-enters state u_1 and scans right. This process is repeated until $U_{15,2}$ has finished reading the encoded read symbol $\langle a_i \rangle$ or symbols $\langle e_j \rangle$ and $\langle a_i \rangle$. This occurs when the subword *ccb* no longer appears to the right of the tape head and signals the end of Cycle 1 and the beginning of Cycle 2.

$U_{15,2}$	u_1	u_2	u_3	u_4	u_5	u_6
c	cRu_2	bRu_3	cLu_7	cLu_6	bRu_1	bLu_4
b	bRu_1	bRu_1	cLu_5	bLu_5	bLu_4	bLu_4

Table 6.2.15: Cycle 1 of $U_{15,2}$.

$$
\begin{aligned}
\vdash^{3} \quad & \boldsymbol{u_5}, \ldots \langle P \rangle (cb)^6 (cccb)^2 (cc)^3 cb\ cb\ cb\ bc\ \underline{b}c\ cb\ cb\ bb (cb)^{8jq+8i-5} bbc \ldots \\
\vdash^{4} \quad & \boldsymbol{u_5}, \ldots \langle P \rangle (cb)^6 (cccb)^2 (cc)^3 cb\ cb\ \underline{c}b\ bc\ bc\ cb\ cb\ bb (cb)^{8jq+8i-5} bbc \ldots \\
\vdash^{4} \quad & \boldsymbol{u_2}, \ldots \langle P \rangle (cb)^6 (cccb)^2 (cc)^3 cb\ cb\ bb\ bc\ \underline{b}c\ cb\ cb\ bb (cb)^{8jq+8i-5} bbc \ldots \\
\vdash^{20} \quad & \boldsymbol{u_2}, \ldots \langle P \rangle (cb)^6 (cccb)^2 (cc)^3 cb\ bb\ bb\ bc\ bc\ \underline{b}c\ cb\ bb (cb)^{8jq+8i-5} bbc \ldots \\
\vdash^{28} \quad & \boldsymbol{u_2}, \ldots \langle P \rangle (cb)^6 (cccb)^2 (cc)^3 bb\ bb\ bb\ bc\ bc\ bc\ \underline{b}c\ bb (cb)^{8jq+8i-5} bbc \ldots
\end{aligned}
$$

Note that in the configuration immediately above each *cb* subword in the encoded read symbol $\langle a_1 \rangle = cbcbcb$ has been changed to the subword *bc*. Note also that the subword *ccb* which causes a scan to the left in u_4, u_5, and u_6 no longer appears in the configuration to the right of the tape head. This causes $U_{15,2}$ to enter Cycle 2.

Cycle 2 (Print production). Cycle 2 (Table 6.2.16) begins with $U_{15,2}$ scanning right and printing *cb* to the right of the encoded dataword. Following this $U_{15,2}$ scans left in states u_7, u_8, u_9, u_{10} and u_{11} and records the next symbol of the encoded production to be printed. If, during a scan left, $U_{15,2}$ reads the subword *ccc* then it scans right in states u_1 and u_2 and changes the *cc* immediately to the right of the encoded dataword to *cb*. If, during a scan left, $U_{15,2}$ reads the subword *ccbcc* it scans right in states u_{12} and u_{13} and changes the first *c* to the right of the encoded dataword to a *b*. This process is repeated until the end of the encoded production is detected by reading the subword *bcbcc* during the scan left. This causes $U_{15,2}$ to enter Cycle 3.

$U_{15,2}$	u_1	u_2	u_3	u_7	u_8
c	cRu_2	bRu_3	cLu_7	cLu_8	bLu_9
b	bRu_1	bRu_1	cLu_5	bLu_7	bLu_7
$U_{15,2}$	u_9	u_{10}	u_{11}	u_{12}	u_{13}
c	cRu_1	bLu_{11}	cRu_{12}	cRu_{13}	cLu_2
b	bLu_{10}		bRu_{14}	bRu_{12}	bRu_{12}

Table 6.2.16: Cycle 2 of $U_{15,2}$.

$$\begin{aligned}
\vdash^* \quad & \boldsymbol{u_1}, \ldots \langle P\rangle (cb)^6 (cccb)^2 cc\ cc\ cc (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb\ \underline{c}c\ cc\ cc \ldots \\
\vdash^3 \quad & \boldsymbol{u_7}, \ldots \langle P\rangle (cb)^6 (cccb)^2 cc\ cc\ cc (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb\ c\underline{b}\ cc\ cc \ldots \\
\vdash^* \quad & \boldsymbol{u_7}, \ldots \langle P\rangle (cb)^6 (cccb)^2 cc\ cc\ c\underline{c} (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb\ cb\ cc\ cc \ldots \\
\vdash^3 \quad & \boldsymbol{u_1}, \ldots \langle P\rangle (cb)^6 (cccb)^2 cc\ cc\ \underline{b}c (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb\ cb\ cc\ cc \ldots \\
\vdash^* \quad & \boldsymbol{u_7}, \ldots \langle P\rangle (cb)^6 (cccb)^2 cc\ c\underline{c}\ bc (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb\ cb\ cb\ cc \ldots
\end{aligned}$$

Each time the subword ccc is read during a scan left in states u_7, u_8, and u_9, $U_{15,2}$ scans right and prints cb at the right of the encoded dataword. Thus we get:

$$\begin{aligned}
\vdash^* \quad & \boldsymbol{u_7}, \ldots \langle P\rangle (cb)^6 cc\ cb\ cc\ cb\ c\underline{c}\ bc\ bc (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb (cb)^3\ cc\ cc \ldots \\
\vdash^5 \quad & \boldsymbol{u_{12}}, \ldots \langle P\rangle (cb)^6 cc\ cb\ cc\ \underline{b}b\ bc\ bc\ bc (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb (cb)^3\ cc\ cc \ldots \\
\vdash^* \quad & \boldsymbol{u_7}, \ldots \langle P\rangle (cb)^6 cc\ cb\ c\underline{c}\ bb (bc)^3 (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb (cb)^3\ bc\ cc \ldots
\end{aligned}$$

Each time the subword $ccbcc$ is read during a scan left in states u_7, u_8, u_9, u_{10}, and u_{11} $U_{15,2}$ scans right and prints b at the right of the encoded dataword. Thus we get:

$$\begin{aligned}
\vdash^* \quad & \boldsymbol{u_7}, \ldots \langle P\rangle (cb)^4 cb\ cb\ c\underline{c}\ bb\ bc\ bb (bc)^3 (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb (cb)^3 bb\ c \ldots \\
\vdash^5 \quad & \boldsymbol{u_{14}}, \ldots \langle P\rangle (cb)^4 cb\ \underline{b}b\ bc\ bb\ bc\ bb (bc)^3 (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb (cb)^3 bb\ c \ldots
\end{aligned}$$

When the subword $bcbcc$ is read during a scan left in states u_7, u_8, u_9, u_{10}, and u_{11} Cycle 2 is complete and Cycle 3 is entered. Thus in the configuration immediately above $U_{15,2}$ has entered Cycle 3.

Cycle 3 (Restore tape). In Cycle 3 (Table 6.2.17) the tape head of $U_{15,2}$ scans right in states u_{14} and u_{15} changing each bc to cc and each bb to cb. This continues until $U_{15,2}$ reads a c in state u_{14}. This c marks the leftmost end of the dataword. Note that during Cycles 1 and 2 each cc in $\langle B\rangle$ and each cb in the encoded read symbol are changed to the subwords bc. Also during Cycles 1 and 2, each cb subword in $\langle B\rangle$ is changed to the subword

bb. Thus c will not be read in u_{14} until we encounter the subword cb at the left end of the next encoded symbol to be read in the dataword.

$$\begin{aligned}
&\vdash^9 \quad \boldsymbol{u_{15}}, \ldots \langle P\rangle (cb)^4 cb\, cb\, cc\, cb\, cc\, c\underline{b} (bc)^3 (bb)^3 (bc)^4 bb (cb)^{8jq+8i-5} bb (cb)^3 bb\, c \ldots \\
&\vdash^* \quad \boldsymbol{u_{14}}, \ldots \langle P\rangle (cb)^6 (cccb)^2 (cc)^3 (cb)^3 (cc)^4\ cb\, \underline{c}b (cb)^{8jq+8i-6} bb (cb)^3 bb\, c \ldots \\
&\vdash^3 \quad \boldsymbol{u_1}, \ldots \langle P\rangle (cb)^6 (cccb)^2 (cc)^3 (cb)^3 (cc)^4\ b\underline{c}\, cb (cb)^{8jq+8i-6} bb (cb)^3 bb\, c \ldots
\end{aligned}$$

In the configuration immediately above the example simulation of production $\langle P(a_1)\rangle$ is complete. The encoded symbol $\langle a_1\rangle = (cb)^3$ has been appended onto the right end of the dataword, the encoded tag system $\langle B\rangle$ has been restored to its original value and $U_{15,2}$ is ready to read the encoded symbols $\langle e_j\rangle$ and $\langle a_i\rangle$.

$U_{15,2}$	u_3	u_5	u_{14}	u_{15}
c		bRu_1	cLu_3	cRu_{14}
b	cLu_5		cRu_{15}	bRu_{14}

Table 6.2.17: Cycle 3 of $U_{15,2}$.

Halting for $\mathbf{U_{15,2}}$. If the encoded halt symbol $\langle e_h\rangle = (cb)^{8hq+2}$ is the leftmost symbol in the encoded dataword then this is encoded via Definition 6.2.1 as follows:

$$\boldsymbol{u_1},\ bb\, cc \langle P(e_{h-1}, a_q)\rangle \ldots \langle P(a_1)\rangle (cb)^3 (cc)^* b\underline{c}\, (cb)^{8hq+2} bb (\langle A\rangle bb)^* cc\, c \ldots$$

The computation continues as before until $U_{15,2}$ enters Cycle 2 and scans left in u_7, u_8, and u_9. This scan ends with the following configuration:

$$\vdash^* \quad \boldsymbol{u_{10}},\ \underline{b}b\, bc \langle P(e_{h-1}, a_q)\rangle' \ldots \langle P(a_1)\rangle' (bb)^3 (bc)^* (bc)^{8hq+2} bb (\langle A\rangle bb)^* cb\, c \ldots$$

In the configuration above, $\langle P\rangle'$ denotes the word in which each cc and cb subword in $\langle P\rangle$ is changed to the subword bc and bb, respectively. There is no transition rule in Table 6.2.14 for the case 'when in u_{10} read c' hence the computation halts.

6.3 Discussion

We noted in the introduction to this chapter that 39 state-symbol pairs remain open, as can be seen in Figure 6.1.1. It is not known which pairs have a decidable halting problem and which pairs contain standard universal Turing machines.

Following Minsky's [Min62a] 7-state, 4 symbol universal machine all of the smallest universal Turing machines (including our bi-tag simulators)

have used a similar algorithm. Since Rogozhin [Rog79, Rog82] established the universal curve there have been incremental reductions in the size of many of his machines. However the smallest of Rogozhin's machines, the 6-symbol machine, has not been improved upon since it was first presented almost 30 years ago. In order to significantly reduce the space between the decidable halting problem curve and the universality curve we suspect that a radically new approach must be taken. Below we give three methods to aid in the search for smaller universal Turing machines.

The first approach is to look for some universal systems other that 2-tag or bi-tag systems that would require less instructions to simulate. Cyclic tag systems (see Chapter 4) may be used to give smaller machines. However the operation of cyclic tag systems is similar to that of tag systems so this may not give much of an improvement. Perhaps a simple universal cellular automaton could be simulated. The cellular automaton Rule 110 has given rise to very small weakly universal Turing machines (see Chapter 7). Perhaps a sufficiently simple universal cellular automaton could be found that allows us to construct small Turing machines that are universal, rather than only weakly universal.

Another approach is to simplify some existing universal model in order to make it easier to simulate. As an interesting example we will briefly consider small semi-weak machines. Watanabe [Wat61] gave a small semi-weakly universal Turing machine with 6 states and 5 symbols that simulates Turing machines directly. Later, Watanabe [Wat72] gave a small semi-weakly universal Turing machine with 5 states and 4 symbols that simulates restricted Turing machines. Watanabe noted that Turing machines with a binary $\{0, 1\}$ tape alphabet where the tape head always moves right on a 1 and left on a 0 are universal. Because of this restriction, Watanabe's encoded table of behaviour for each Turing machine had no need to include information about the direction of movement of the tape head. This in turn simplified the problem of simulating Turing machines.

A third approach is to find an encoding that allows many different operations to be carried out by the same group of instructions. In Chapter 3 we gave small universal Turing machines that simulate Turing machine directly. The encoding used by these machines allowed each set of transition rules to serve more than one purpose. A single set of transition rules reads both the encoded current state and the encoded read symbol. Another set of transition rules, prints the encoded write symbol, moves the simulated tape head, and establishes the new encoded current state. Combining steps in this way has reduced the number of transition rules needed by our universal machines in Chapter 3.

The number of state-symbol pairs that remain open could also be reduced by searching for decidability results. Decidability results provide useful lower bounds for small universal Turing machines. To date, there has been no improvement on Pavlotskaya's [Pav78] 1978 proof that the halting problem

is decidable for 3-state, 2-symbol machines. Pavlotskaya's proof is quite long and complex and improving on this result may well be difficult. As the state-symbol product increases, the number of possible machines increases exponentially. Thus it seems that a new approach needs to be taken. To find new lower bounds one possible method is to prove that some non-universal system simulates all of the Turing machines for a given state-symbol pair.

7

Small weakly universal Turing machines

7.1 Introduction

In this chapter we present small universal Turing machines with state-symbol pairs of $(6, 2)$, $(3, 3)$ and $(2, 4)$. These machines are weakly universal, which means that they have an infinitely repeated word to the left of their input and another to the right. They simulate Rule 110 and are currently the smallest known weakly universal Turing machines.

We recall that, beginning in the early sixties Minsky and Watanabe engaged in a vigorous competition to see who could come up with the smallest universal Turing machine [Min60a, Min62a, Wat60, Wat61, Wat72]. In 1961 Watanabe [Wat61] gave a 6-state, 5-symbol universal Turing machine, the first weakly universal machine. In 1962, Minsky [Min62a] found a small 7-state, 4-symbol universal Turing machine. Not to be out-done, Watanabe improved on his earlier machine to give 5-state, 4-symbol and 7-state, 3-symbol weakly universal machines [Wat72, Noz69]. Some of the earliest small universal machines are given in Table 1.1.1.

The 7-state universal Turing machine of Minsky has received much attention. Minsky's machine simulates Turing machines via 2-tag systems, which were proved universal by Cocke and Minsky [CM64]. The technique of simulating 2-tag systems, pioneered by Minsky, was extended by Rogozhin [Rog82] to give the (then) smallest known universal Turing machines for a number of state-symbol pairs. These 2-tag simulators were subsequently reduced in size by Rogozhin [Rog96], Kudlek and Rogozhin [KR02], and Baiocchi [Bai01]. In Chapter 6 we gave small universal machines that simulate Turing machines via a new variant of tag systems called bi-tag systems. In Figure 7.1.1 each of the smallest 2-tag simulators are plotted as hollow circles and each of the smallest bi-tag simulators are plotted as solid triangles. These (standard) machines induce a universal curve.

The small weak machines of Watanabe have received little attention. In particular the 5-state and 7-state machines seem little known and are largely ignored in the literature. It is worth noting that unlike other weak Turing machines the weak machines of Watanabe are proved universal using the

technique of direct simulation of Turing machines. His machines are the most time-efficient of the small weak machines: Watanabe's machines are the smallest weak machines that simulate with a $O(t^2)$ time overhead.

We often refer to Watanabe's machines as being semi-weak. Semi-weak machines are a restriction of weak machines: they have an infinitely repeated word to one side of their input, and on the other side they have a (standard) infinitely repeated blank symbol. Recently, Woods and Neary [WN07b, WNb] have given semi-weakly universal machines that simulate cyclic tag systems. These machines have state-symbol pairs of $(2, 14)$, $(3, 7)$ and $(4, 5)$. In Figure 7.1.1 the semi-weakly universal machines of Watanabe are plotted as hollow diamonds and those of Woods and Neary are plotted as solid diamonds.

Cook and Eppstein [Coo04], and Wolfram [Wol02] recently gave weakly universal Turing machines, smaller than Watanabe's semi-weak machines, that simulate the universal cellular automata Rule 110. These machines have state-symbol pairs of $(7, 2)$, $(4, 3)$, $(3, 4)$, $(2, 5)$ and are plotted as hollow squares in Figure 7.1.1. (Note that David Eppstein constructed the $(7, 2)$ machine to be found in [Coo04].)

The weakly universal Turing machines we give here simulate (single tape, deterministic) Turing machines in time $O(t^4 \log^2 t)$, via Rule 110. These machines are plotted as solid squares in Figure 7.1.1 and induce a weakly universal curve. The weak machines we present in this chapter have previously appeared in [NW07b].

Over the years, small universal programs were given for a number of variants on the standard model. By generalising the model we often find smaller universal programs. Weakness has not been the only generalisation on the standard model in the search for small universal Turing machines. Other generalisations on the standard model are to be found in Section 1.1.3.

7.2 Rule 110

Rule 110 is a very simple (2-state, nearest neighbour, one-dimensional) cellular automaton. It is composed of a sequence of cells $\ldots p_{-1}p_0p_1 \ldots$ where each cell has a binary state $p_i \in \{0, 1\}$. At timestep $s + 1$, the value $p_{i,s+1} = F(p_{i-1,s}, p_{i,s}, p_{i+1,s})$ of the cell at position i is given by the synchronous local update function F

$$\begin{aligned} F(0,0,0) &= 0 & F(1,0,0) &= 0 \\ F(0,0,1) &= 1 & F(1,0,1) &= 1 \\ F(0,1,0) &= 1 & F(1,1,0) &= 1 \\ F(0,1,1) &= 1 & F(1,1,1) &= 0 \end{aligned} \tag{7.2.1}$$

Rule 110 was proven universal by Cook [Coo04] (a sketch of Cook's proof also appears in [Wol02]). In Chapter 4 we proved that Rule 110 simulates

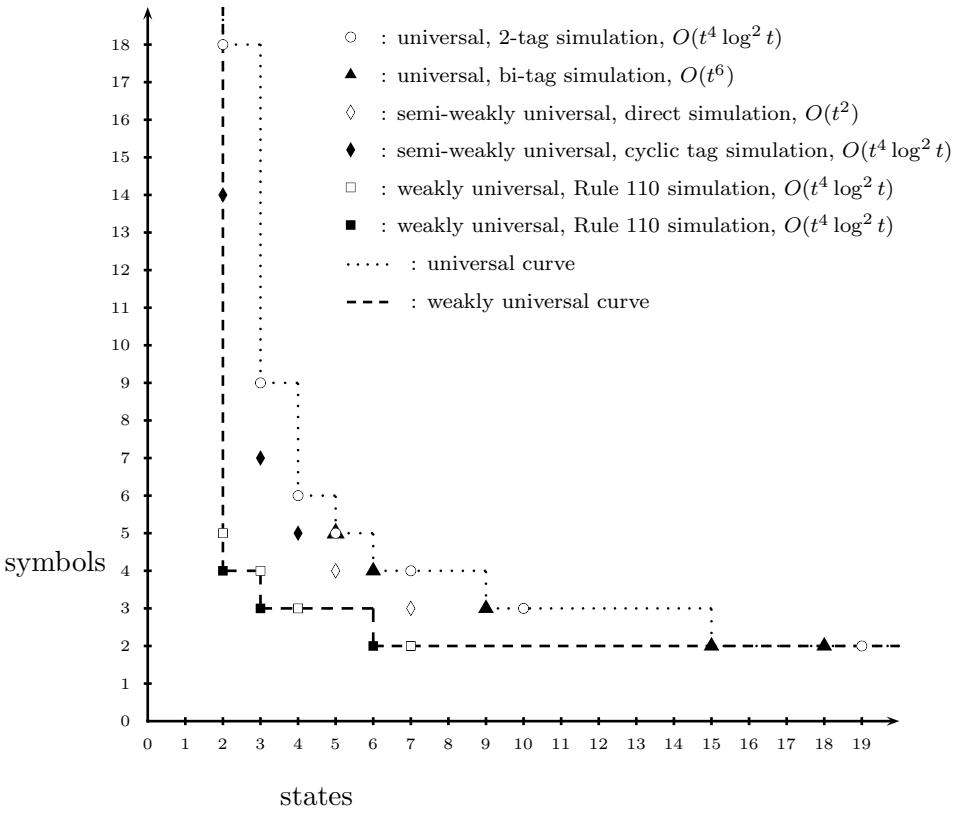


Figure 7.1.1: State-symbol plot of small weakly universal Turing machines. Each new weakly universal Turing machines is plotted as a solid square.

Turing machines efficiently in polynomial time. Rule 110 simulates cyclic tag systems in linear time. Thus from Theorem 4.3.2 Rule 110 simulates Turing machines in $O(t^2 \log t)$ time. The weak machines in this chapter, and in [Coo04, Wol02], simulate Rule 110 with a quadratic polynomial increase in time and hence simulate Turing machines in time $O(t^4 \log^2 t)$. It is worth noting that the prediction problem [GHR95] for these machines is thus P-complete, and this is also the case when we restrict to bounded initial conditions.

7.3 Three small weakly universal Turing machines

The following observation is one of the reasons for the improvement in size over previous [Coo04, Wol02] weak machines, and gives some insight into the simulation algorithm that we use. Notice from Equation (7.2.1) that the value of the update function F, with the exception of $F(0,1,1)$ and $F(1,1,1)$, may be determined using only the rightmost two states. Each

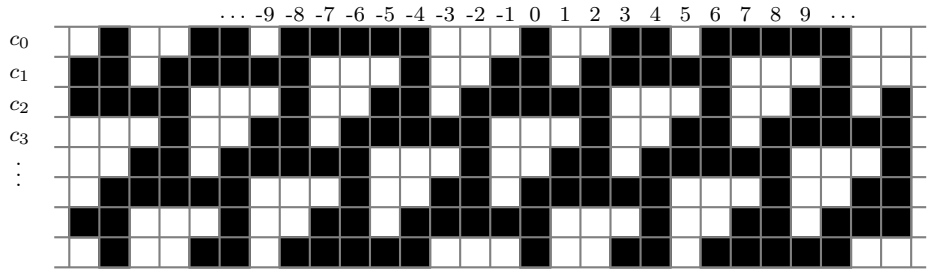


Figure 7.2.1: Seven consecutive timesteps of Rule 110. These seven timesteps are taken from the background ether that is used in the proof [Coo04] of universality of Rule 110. Each black or each white square represents, a Rule 110 cell containing, state 1 or 0 respectively. Each cell is identified by the index given above it. To the left of each row of cells there is a configuration label that identifies that row.

of our universal Turing machines exploit this fact as follows. The machines scan from right to left, and in six of the eight cases they need only remember the cell immediately to the right of the current cell i in order to compute the update for i. Thus for these six cases we need only store a single cell value, rather than two values. The remaining two cases are simulated as follows. If two consecutive encoded states with value 1 are read, it is assumed that there is another encoded 1 to the left and the update $F(1,1,1) = 0$ is simulated. If our assumption proves false (we instead read an encoded 0), then our machine returns to the wrongly updated cell and simulates the update $F(0,1,1) = 1$.

Before giving our three small Rule 110 simulators, we give some further background explanation. Rule 110 simulates Turing machines via cyclic tag systems. A Rule 110 instance that simulates a cyclic tag system computation is of the following form (for more details see [Coo04]). The input to the cyclic tag system is encoded in a contiguous finite number of Rule 110 cells. On the left of the input a fixed constant word (representing the 'ossifiers') is repeated infinitely many times. On the right, another fixed constant word (representing the cyclic tag system program/appendants, and the 'leaders') is repeated infinitely many times. Both of these repeated words are independent of the input.

Our weakly universal machines operate by traversing a finite amount of the tape from right to left and then from left to right. This simulates a single timestep of Rule 110 over a finite part of the encoded infinite Rule 110 instance. With each simulated timestep the length of a traversal increases. To ensure that each traversal is of finite length, the left blank word l and the right blank word r of each of our weak machines must have a special form. These words contain special subwords or symbols that terminate

each traversal, causing the tape head to turn. When the head is turning it overwrites any symbols that caused a turn. Thus the number of cells that are being updated increases monotonically over time. This technique simulates Rule 110 properly if the initial condition is set up so that within each repeated blank word, the subword between each successive turn point is shifted one timestep forward in time.

In the sequel we describe the computation of our three machines by showing a simulation of the update on the ether in Figure 7.2.1. In the next paragraph below, we outline why this example is in fact general enough to prove universality. First, we must define blank words that are suitable for this example. The left blank word l, on the Turing machine tape, encodes the Rule 110 sequence 0001. In the initial configuration as we move left each subsequent sequence 0001 is one timestep further ahead. To see this note from Figure 7.2.1 that 0001 occupies, cells -7 to -4 in configuration c_1, cells -11 to -8 in c_2, cells -15 to -12 in c_3, etc. Similarly, the right blank word r encodes the Rule 110 sequence 110011. Looking at the initial configuration, as we move right from cell 0, in the first blank word the first four cells 1100 are shifted two timesteps ahead, and the next two cells 11 are shifted a further one timestep. To see this note from Figure 7.2.1 that 1100 occupies cells 1 to 4 in c_2 and 11 occupies cells 5 and 6 in c_3. In each subsequent sequence the first four cells 1100 are shifted only one timestep ahead and the last two cells 11 are shifted one further timestep. In each row the ether in Figure 7.2.1 repeats every 14 cells and if the number of timesteps s between two rows is $s \equiv 0 \mod 7$ then the two rows are identical. The periodic nature of the ether, in both time and space, allows us to construct such blank words.

It should be noted that the machines we present here, and those in [Coo04], require suitable blank words to simulate a Rule 110 instance directly. If no suitable blank words can be found (i.e. if it is not possible to construct the specific subwords that we use to terminate traversals in the encoding) then it may be the case that the particular instance can not be simulated directly. However, in the sequel our machines simulate the background ether that is used in the universality proof of Rule 110 [Coo04]. The gliders used by Cook [Coo04] that move through this ether are periodic in time and space. Thus, we can construct blank words that included these gliders and place the subwords that terminate traversals in the ether. By this reasoning, our example is sufficiently general to prove that our machines simulate Turing machines via Rule 110 and we do not give a full (and possibly tedious) proof of correctness. For $U_{3,3}$ we explicitly simulate three updates from Figure 7.2.1, which is general enough so that an update [Equation (7.2.1)] on each of the eight possible three state combinations is simulated. We give shorter examples for the machines $U_{2,4}$ and $U_{6,2}$ as they use the same simulation algorithm as $U_{3,3}$.

The machines we present here do not halt. Cook [Coo04] shows how a

special glider may be produced during the simulation of a Turing machine by Rule 110. This glider may be used to simulate halting as the encoding can be such that it is generated by Rule 110 if and only if the simulated machine halts. The glider would be encoded on the tape of our machines as a unique, constant word.

7.3.1 $U_{3,3}$

We begin by describing an *initial* configuration of $U_{3,3}$. To the left of, and including, the tape head position, the Rule 110 state 0 is encoded by 0, and the Rule 110 state 1 is encoded by either 1 or b. The word $1b0$ is used to terminate a left traversal. (Note an exception: the 1 in the subword $1b0$ encodes the Rule 110 state 0.) To the right of the tape head position, the Rule 110 state 0 is encoded by 1, and the Rule 110 state 1 is encoded by 0 or b. The tape symbol 0 is used to terminate a right traversal. The left and right blank words, described in paragraph 4 of Section 7.3, are encoded as $0\,0\,1\,b$ and $0\,b\,1\,1\,0\,b$ respectively.

	u_1	u_2	u_3
0	$1Lu_1$	$0Ru_1$	bLu_1
1	bLu_2	$1Lu_2$	$0Ru_3$
b	bLu_3		$1Ru_3$

Table 7.3.1: Table of behaviour for $U_{3,3}$.

We give an example of $U_{3,3}$ simulating the three successive Rule 110 timesteps $c_0 \vdash c_1 \vdash c_2 \vdash c_3$ given in Figure 7.2.1. In the below configurations the current state of $U_{3,3}$ is highlighted in bold, to the left of its tape contents. The tape head position of $U_{3,3}$ is given by an underline and the start state is u_1. The configuration immediately below encodes c_0 from Figure 7.2.1 with the tape head over cell index 0.

$$
\begin{aligned}
&\boldsymbol{u_1},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,0\,\underline{1} \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots \\
\vdash\ &\boldsymbol{u_2},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,\underline{0}\,b \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots \\
\vdash\ &\boldsymbol{u_1},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,0\,\underline{b} \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots \\
\vdash\ &\boldsymbol{u_3},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,\underline{0}\,b \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots \\
\vdash\ &\boldsymbol{u_1},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,\underline{0}\,b\,b \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots \\
\vdash^2\ &\boldsymbol{u_1},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,1\,\underline{b} \quad 1\,1\,b\,b \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots \\
\vdash\ &\boldsymbol{u_3},\ \ldots\ 0\,0\,1\,b \quad 0\,0\,1\,b \quad 0\,0\,\underline{1}\,b \quad 1\,1\,b\,b \quad 0\,b\,1\,1\,0\,b \quad 0\,b\,1\,1\,0\,b\ \ldots
\end{aligned}
$$

When the tape head reads the subword $1b0$ the left traversal is complete

and the right traversal begins.

$$\begin{array}{rlllllll}
\vdash^6 & \boldsymbol{u_3}, \ldots & 001b & 001b & 0001 & 0011 & \underline{0}b110b & 0b110b \ldots \\
\vdash & \boldsymbol{u_1}, \ldots & 001b & 001b & \mathbf{0001} & \mathbf{001\underline{1}} & bb110b & 0b110b \ldots
\end{array}$$

Immediately after the tape head reads a 0, during a right traversal, the simulation of timestep $c_0 \vdash c_1$ is complete. To see this, compare the part of the Turing machine tape in bold with cells -7 to 0 of configuration c_1 in Figure 7.2.1. We continue our simulation to give timestep $c_1 \vdash c_2$.

$$\begin{array}{rlllllll}
\vdash & \boldsymbol{u_2}, \ldots & 001b & 001b & 0001 & 00\underline{1}b & bb110b & 0b110b \ldots \\
\vdash & \boldsymbol{u_2}, \ldots & 001b & 001b & 0001 & 0\underline{0}1b & bb110b & 0b110b \ldots \\
\vdash & \boldsymbol{u_1}, \ldots & 001b & 001b & 0001 & 00\underline{1}b & bb110b & 0b110b \ldots \\
\vdash & \boldsymbol{u_2}, \ldots & 001b & 001b & 0001 & 0\underline{0}bb & bb110b & 0b110b \ldots \\
\vdash^3 & \boldsymbol{u_1}, \ldots & 001b & 001b & 0001 & \underline{0}bbb & bb110b & 0b110b \ldots \\
\vdash^2 & \boldsymbol{u_2}, \ldots & 001b & 001b & 00\underline{0}b & 1bbb & bb110b & 0b110b \ldots \\
\vdash^3 & \boldsymbol{u_1}, \ldots & 001b & 001b & 0\underline{0}bb & 1bbb & bb110b & 0b110b \ldots \\
\vdash^3 & \boldsymbol{u_3}, \ldots & 001b & 00\underline{1}b & 11bb & 1bbb & bb110b & 0b110b \ldots \\
\vdash^{15} & \boldsymbol{u_1}, \ldots & 001b & \mathbf{0001} & \mathbf{0011} & \mathbf{0111} & \mathbf{1100}\underline{0}bb & 0b110b \ldots
\end{array}$$

The simulation of timestep $c_1 \vdash c_2$ is complete. To see this, compare the part of the Turing machine tape in bold with cells -11 to 4 of configuration c_2 in Figure 7.2.1. We continue our simulation to give timestep $c_2 \vdash c_3$.

$$\begin{array}{rlllllll}
\vdash^3 & \boldsymbol{u_2}, \ldots & 001b & 0001 & 0011 & 0111 & \underline{1}b11bb & 0b110b \ldots \\
\vdash^4 & \boldsymbol{u_2}, \ldots & 001b & 0001 & 0011 & \underline{0}111 & 1b11bb & 0b110b \ldots \\
\vdash^5 & \boldsymbol{u_1}, \ldots & 001b & 0001 & 001\underline{1} & bb11 & 1b11bb & 0b110b \ldots \\
\vdash^2 & \boldsymbol{u_2}, \ldots & 001b & 0001 & 0\underline{0}1b & bb11 & 1b11bb & 0b110b \ldots \\
\vdash^5 & \boldsymbol{u_1}, \ldots & 001b & 0001 & \underline{0}bbb & bb11 & 1b11bb & 0b110b \ldots \\
\vdash^2 & \boldsymbol{u_2}, \ldots & 001b & 00\underline{0}b & 1bbb & bb11 & 1b11bb & 0b110b \ldots \\
\vdash^6 & \boldsymbol{u_3}, \ldots & 00\underline{1}b & 11bb & 1bbb & bb11 & 1b11bb & 0b110b \ldots \\
\vdash^{21} & \boldsymbol{u_1}, \ldots & \mathbf{0001} & \mathbf{0011} & \mathbf{0111} & \mathbf{1100} & \mathbf{01001\underline{1}} & bb110b \ldots
\end{array}$$

The simulation of timestep $c_2 \vdash c_3$ is complete. To see this, compare the part of the Turing machine tape in bold with cells -15 to 6 of configuration c_3 in Figure 7.2.1.

7.3.2 $U_{2,4}$

We begin by describing an *initial* configuration of $U_{2,4}$. To the left of, and including, the tape head position, the Rule 110 state 0 is encoded by either 0

or $\emptyset$ and the Rule 110 state 1 is encoded by either 1 or $\not 1$. The word $\emptyset 1$ is used to terminate a left traversal. To the right of the tape head position, the Rule 110 state 0 is encoded by $\emptyset$ and the Rule 110 state 1 is encoded by $\not 1$ or 0. The tape symbol 0 is used to terminate a right traversal. The left and right blank words, from paragraph 4 of Section 7.3, are encoded as $00\emptyset 1$ and $0\not 1\emptyset\emptyset 0\not 1$ respectively.

	u_1	u_2
0	$\emptyset L u_1$	$\not 1 R u_1$
1	$\not 1 L u_2$	$\emptyset L u_2$
$\emptyset$	$\not 1 L u_1$	$0 R u_2$
$\not 1$	$\not 1 L u_1$	$1 R u_2$

Table 7.3.2: Table of behaviour for $U_{2,4}$.

By way of example we give $U_{2,4}$ simulating the two successive Rule 110 timesteps $c_0 \vdash c_1 \vdash c_2$ given in Figure 7.2.1. The configuration immediately below encodes c_0 from Figure 7.2.1 with the tape head over cell index 0.

$$
\begin{array}{rlllllll}
 & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 00\emptyset 1 & 000\underline{1} & 0\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash^6 & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 00\emptyset\underline{1} & \emptyset\emptyset\not 1\not 1 & 0\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash & \boldsymbol{u_2}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 00\underline{\emptyset}\not 1 & \emptyset\emptyset\not 1\not 1 & 0\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots
\end{array}
$$

When the tape head reads the subword $\emptyset 1$ the left traversal is complete and the right traversal begins.

$$
\begin{array}{rlllllll}
\vdash^6 & \boldsymbol{u_2}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 0001 & 0011 & \underline{0}\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & 00\emptyset 1 & \mathbf{0001} & \mathbf{0011} & \not 1\underline{\not 1}\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots
\end{array}
$$

Immediately after the tape head reads a 0, during a right traversal, the simulation of timestep $c_0 \vdash c_1$ is complete. To see this, compare the part of the Turing machine tape in bold with cells -7 to 0 of configuration c_1 in Figure 7.2.1. We continue our simulation to give timestep $c_1 \vdash c_2$.

$$
\begin{array}{rlllllll}
\vdash^2 & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 0001 & 001\underline{1} & \not 1\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash^2 & \boldsymbol{u_2}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 0001 & 00\underline{\emptyset}\not 1 & \not 1\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 0001 & 0\not 1\underline{\emptyset}\not 1 & \not 1\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash^4 & \boldsymbol{u_2}, \ldots & 00\emptyset 1 & 00\emptyset 1 & 00\underline{0}\not 1 & \emptyset\not 1\not 1\not 1 & \not 1\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash^5 & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & 00\emptyset\underline{1} & \emptyset\emptyset\not 1\not 1 & \emptyset\not 1\not 1\not 1 & \not 1\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash & \boldsymbol{u_2}, \ldots & 00\emptyset 1 & 00\underline{\emptyset}\not 1 & \emptyset\emptyset\not 1\not 1 & \emptyset\not 1\not 1\not 1 & \not 1\not 1\emptyset\emptyset 0\not 1 & 0\not 1\emptyset\emptyset 0\not 1 \ldots \\
\vdash^{15} & \boldsymbol{u_1}, \ldots & 00\emptyset 1 & \mathbf{0001} & \mathbf{0011} & \mathbf{0111} & \mathbf{1100}\not 1\underline{\not 1} & 0\not 1\emptyset\emptyset 0\not 1 \ldots
\end{array}
$$

The simulation of timestep $c_1 \vdash c_2$ is complete. To see this, compare the part of the Turing machine tape in bold with cells -11 to 4 of configuration c_2 in Figure 7.2.1.

7.3.3 $U_{6,2}$

We begin by describing an *initial* configuration of $U_{6,2}$. To the left of, and including, the tape head position, the Rule 110 state 0 is encoded by the word 00 and the Rule 110 state 1 is encoded by the word 11. The word 010100 is used to terminate a left traversal and encodes the sequence of Rule 110 states 010. To the right of the tape head position the Rule 110 state 0 is encoded by the word 00 and the Rule 110 state 1 is encoded by either of the words 01 or 10. The word 10 is used to terminate a right traversal. The left and right blank words, from paragraph 4 of Section 7.3, are encoded as $0\,0\,0\,0\,0\,1\,0\,1$ and $1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1$ respectively.

	u_1	u_2	u_3	u_4	u_5	u_6
0	$0Lu_1$	$0Lu_6$	$0Ru_2$	$1Ru_5$	$1Lu_4$	$1Lu_1$
1	$1Lu_2$	$0Lu_3$	$1Lu_3$	$0Ru_6$	$1Ru_4$	$0Ru_4$

Table 7.3.3: Table of behaviour for $U_{6,2}$.

To illustrate the operation of $U_{6,2}$ we simulate the Rule 110 timestep $c_0 \vdash c_1$ given in Figure 7.2.1. The configuration immediately below encodes c_0 from Figure 7.2.1 with the tape head over cell index 0.

$$
\begin{aligned}
& \boldsymbol{u_1},\ \ldots 0\,0\,0\,0\,0\,1\,0\,1 && 0\,0\,0\,0\,0\,0\,1\,\underline{1} && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_2},\ \ldots 0\,0\,0\,0\,0\,1\,0\,1 && 0\,0\,0\,0\,0\,0\,\underline{1}\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_3},\ \ldots 0\,0\,0\,0\,0\,1\,0\,1 && 0\,0\,0\,0\,0\,\underline{0}\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_2},\ \ldots 0\,0\,0\,0\,0\,1\,0\,1 && 0\,0\,0\,0\,0\,0\,\underline{0}\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_6},\ \ldots 0\,0\,0\,0\,0\,1\,0\,1 && 0\,0\,0\,0\,0\,\underline{0}\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_1},\ \ldots 0\,0\,0\,0\,0\,1\,0\,1 && 0\,0\,0\,0\,\underline{0}\,1\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash^5\ & \boldsymbol{u_1},\ \ldots 0\,0\,0\,0\,0\,1\,0\,\underline{1} && 0\,0\,0\,0\,0\,1\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_2},\ \ldots 0\,0\,0\,0\,0\,1\,\underline{0}\,1 && 0\,0\,0\,0\,0\,1\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_6},\ \ldots 0\,0\,0\,0\,0\,\underline{1}\,0\,1 && 0\,0\,0\,0\,0\,1\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots \\
\vdash\ & \boldsymbol{u_4},\ \ldots 0\,0\,0\,0\,0\,0\,\underline{0}\,1 && 0\,0\,0\,0\,0\,1\,0\,1 && 1\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,1\ \ldots
\end{aligned}
$$

When the tape head reads the subword $1\,0\,1\,0\,0$ the left traversal is complete and the right traversal begins.

$\vdash \boldsymbol{u_5}, \ldots 00000011 \quad 00000101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_4}, \ldots 00000011 \quad 00000101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_5}, \ldots 00000011 \quad 10000101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_4}, \ldots 00000011 \quad 11000101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_6}, \ldots 00000011 \quad 01000101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_4}, \ldots 00000011 \quad 00000101 \quad 100100001001\ldots$
$\vdash^4 \boldsymbol{u_4}, \ldots 00000011 \quad 00000101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_5}, \ldots 00000011 \quad 00001101 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_4}, \ldots 00000011 \quad 00001101 \quad 100100001001\ldots$
$\vdash^2 \boldsymbol{u_4}, \ldots 00000011 \quad 00001111 \quad 100100001001\ldots$
$\vdash \boldsymbol{u_6}, \ldots 00000011 \quad 00001111 \quad 000100001001\ldots$
$\vdash \boldsymbol{u_1}, \ldots \mathbf{00000011} \quad \mathbf{00001111} \quad 010100001001\ldots$

Immediately after the tape head reads a 10, during a right traversal, the simulation of timestep $c_0 \vdash c_1$ is complete. To see this, compare the part of the Turing machine tape in bold (recall 0 and 1 are encoded as 00 and 11 respectively) with cells -7 to 0 of configuration c_1 in Figure 7.2.1.

7.4 Discussion

The pursuit to find the smallest possible universal Turing machine must also involve the search for lower bounds, finding the largest set of Turing machines that are in some sense non-universal. One approach is to settle the decidability of the halting problem, but this approach is not suitable for the non-halting machines we have presented.

It is known that the halting problem is decidable for (standard) Turing machines with the following state-symbol pairs $(2,2)$ [Kud96, Pav73], $(3,2)$ [Pav78], $(2,3)$ (claimed by Pavlotskaya [Pav73]), $(1,n)$ [Her68c] and $(n,1)$ (trivial), where $n \geqslant 1$. Then, these decidability results imply that a universal Turing machine, that simulates any Turing machine M and halts if and only if M halts, is not possible for these state-symbol pairs. Hence these results give lower bounds on the size of universal machines of this type. While it is trivial to prove that the halting problem is decidable for (possibly halting) weak machines with state-symbol pairs of the form $(n,1)$, it is not known whether the above decidability results generalise to (possibly halting) weak Turing machines.

The weakly universal machines presented in this chapter, and those in [Coo04], do not halt. Hence the non-universality results discussed in the previous paragraph would have to be generalised to *non-halting weak* machines to give lower bounds. This may prove difficult for two reasons. The

first issue is that, intuitively speaking, weakness gives quite an advantage. For instance, the program of a universal machine may be encoded in one of the infinitely repeated blank words of the weak machine. The second issue is related to the problem of defining a computation. Informally, a computation could be defined as a sequence of configurations that ends with a special terminal configuration. For non-halting machines, there are many ways to define a terminal configuration. Given a definition of terminal configuration we may prove that the terminal configuration problem (will a machine ever enter a terminal configuration?) is decidable for a machine or set of machines. However this result may not hold as a proof of non-universality if we subsequently alter our definition of terminal configuration.

It is trivial that no weakly universal Turing machines exist for the state-symbol pair $(n, 1)$ even when we consider machines with no halting condition. We also believe that such relevant decidability results for the state-symbol pair $(2, 2)$ may also be given. If this is true, then the problem for 2-state and 3-state weakly universal machines only remains open for $(2, 3)$ and $(3, 2)$ respectively.

Margenstern [Mar00] and Baiocchi [Bai98] constructed small machines that simulate iterations of the Collatz $(3x + 1)$ problem. The size of the smallest known weakly universal machines are quite close to the possible minimum size for weakly universal machines. Implementing the Collatz problem on weak machines could be an interesting way of exploring the little space remaining between these machines and the state-symbol pairs where weak universality is not possible.

8

Conclusion

Our results reduce the size and improve the time efficiency of many of the simplest known models of computation.

We presented new polynomial time (standard) universal Turing machines with state-symbol pairs of $(5, 5)$, $(6, 4)$, $(9, 3)$ and $(15, 2)$. These machines simulate our new variant of tag system, the bi-tag system, and are the smallest known universal Turing machines with 5, 4, 3 and 2-symbols respectively. Our 5-symbol machine uses the same number of instructions (22) as the smallest known universal Turing machine (Rogozhin's 6-symbol machine [Rog96]).

We have shown 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 on a forty-year old result. We also introduced a new form of tag system which we call a bi-tag system. We prove bi-tag systems are universal by showing they efficiently simulate Turing machines in polynomial time.

We have shown that cyclic tag systems are polynomial simulators of Turing machines. This result is used to prove that Rule 110, one of the most intuitively simple cellular automata, is P-complete to predict. In fact, we proved that Rule 110 is a polynomial simulator of Turing machines. As a corollary of this result, we find that the smallest weakly universal Turing machines simulate Turing machines efficiently in polynomial time. All of these results represent an exponential improvement.

We have given the smallest efficient universal Turing machines that simulate Turing machines in $O(t^2)$ time. They are also the smallest known machines where direct simulation of Turing machines is the technique used to establish their universality. This result also gives a new algorithm for small universal Turing machines.

We presented the smallest known weakly universal Turing machines. These machines have state-symbol pairs of $(6, 2)$, $(3, 3)$ and $(2, 4)$. The 3-state and 2-state machines are very close to the minimum possible size for weakly universal machines with 3 and 2 states, respectively.

We have given new results for many different simple universal models including Turing machines, weak Turing machines, tag systems and cellular

automata. These models have applications for a variety of other results. For instance the results surveyed in Section 5.3.1 rely on simulations of 2-tag systems. Many of these results also rely on simulation of small universal Turing machines. It is possible that the (much smaller) weakly universal machines from Chapter 7 may also be of use in some of these simulations to give further improvements. In particular our weak machines may be used with the results of Lindgren and Nordahl [LN90] to give simple universal cellular automata that use periodic backgrounds. We have already seen an application of Rule 110 in constructing small efficient weakly universal machines in Chapter 7 and similar applications of cyclic tag systems in constructing efficient semi-weakly universal machines in [WN07b, WNb]. More recently, our new 4-symbol universal Turing machine from Chapter 6 was used to give a small universal spiking neural P system [Nea08a]. No doubt, other interesting applications are out there waiting to be discovered.

8.1 Future work

8.1.1 Standard Turing machines and program size

Perhaps one of the most obvious aims is to further reduce the size of the smallest (standard) universal Turing machines. As noted in Section 6.3 we suspect that some radically new techniques are needed to significantly reduce the size of the smallest known machines.

The search for the smallest universal Turing machine goes hand-in-hand with the search for decidability results. Decidability results provide useful lower bounds for small universal Turing machines. In Section 6.1 we saw that the universality/non-universality question still remains open for 39 state-symbol pairs. There has been no improvement on Pavlotskaya's [Pav78] result that the halting problem is decidable for 3-state, 2-symbol machines, which was given in 1978. Pavlotskaya's proof is quite long and complex. As the state-symbol product increases, the number of possible machines increases exponentially, thus it seems that a new approach needs to be taken. To find new lower bounds one possible method is to prove that some non-universal system simulates all of the standard Turing machines for a given state-symbol pair.

8.1.2 Weak Turing machines and program size

It would be interesting to give decidability results (lower bounds) for non-halting weak Turing machines. To date, no relevant decidability results have been given for such machines. Such results would provide lower bounds relevant to our weakly universal machines in Chapter 7. We discussed the problem of giving these lower bounds in Section 7.4. It is trivial that no weakly universal Turing machine exists for the state-symbol pair $(n, 1)$ even

when we consider machines with no halting condition. We also conjecture that relevant decidability results for the state-symbol pair $(2, 2)$ may also be given. If this is true, the problem for 2-state and 3-state weakly universal machines only remains open for $(2, 3)$ and $(3, 2)$ respectively.

Margenstern [Mar00] and Baiocchi [Bai98] constructed small machines that simulate iterations of the Collatz $(3x + 1)$ problem. It would be interesting to construct weak machines to simulate this problem. The size of the smallest known weakly universal machines are quite close to the possible minimum size for weakly universal machines. Implementing the Collatz problem on weak machines could be an interesting way of exploring the little space remaining between these machines and the state-symbol pairs where weak universality is not possible.

8.1.3 Time efficiency of simple universal models

It would be interesting to study the time complexity of small multitape Turing machines. For instance, what about simple universal Turing machines with more than one tape that simulate Turing machines in linear time? We noted in Section 3.6 that the algorithm we suggested for linear time universal Turing machines may require many transition rules. Constructing a small 2-tape linear time simulator could perhaps be more easily achieved: one tape could contain the simulated tape contents and tape head position and the other tape could contain the encoded table of behaviour to search for the next instruction. This would avoid scanning back and forth through the tape contents to simulate a transition rule.

It would also be interesting to see if the time efficiency of 2-tag systems, bi-tag systems and cyclic tag systems may be improved further.

8.1.4 The not halting problem

In Section 2.2 we discuss a number of different definitions of universal Turing machine. A number of authors have given definitions for universal Turing machines that do not require the machine to halt. Our small weakly universal Turing machines in Chapter 7 are non-halting. What about proving relevant non-universality results? Such non-universality results are not achievable by proving the halting problem decidable. Before we attempt such an endeavor we must agree on the assumptions for the definition of universal Turing machine. We allow the following generalisation of Definition 2.1.7, instead of the computation ending with a single terminal configuration, the computation ends with a finite configuration sequence called the terminal configuration sequence. The output of the simulated Turing machine is retrieved by applying the recursive decoding function to this sequence. There are many ways to define a terminal configuration sequence. Some examples of possible terminal configuration sequences are:

- The configurations containing a given sequence of states.
- A sequence containing two identical configurations.
- A single configuration containing a constant word.

Given a definition of a terminal configuration sequence we may prove that the terminal sequence problem (will a machine ever execute a terminal configuration sequence) is decidable. This gives non-universal lower bounds that are relevant to universal machines that end their computation with such a sequence. However, this result may not hold as a proof of non-universality if we subsequently alter our definition of terminal configuration sequence. One more general approach is to prove that the terminal sequence problem for all possible terminal sequences, of a machine or set of machines, is decidable.

Definition 2.1.7, augmented with the notion of terminal configuration sequences, is a generalisation on the definitions of Davis and Priese, which were discussed in Section 2.2. We propose this definition as an attempt to include as many reasonable models as possible.

8.1.5 Some final questions

In Figure 1.3.1 the smallest (standard) universal machines are currently not symmetric about the line where states equals symbols. Are the smallest possible universal Turing machines symmetric about the states equals symbols line?

Following the work in this thesis, all of the smallest known weakly, semi-weakly, and standard universal Turing machines now simulate Turing machines efficiently in polynomial time. Are all of the smallest possible universal Turing machines efficient polynomial time simulators?

Currently the smallest weakly universal Turing machines are significantly smaller than the smallest standard universal Turing machines. The 3-state, 3-symbol weakly universal machine, given in Chapter 7, sits in the armchair that is the current decidability curve for standard machines. Is this *weakly* universal machine smaller than the smallest possible *standard* universal Turing machine?

Finally, there is no need to restate Shannon's famous question here!

Notation

M	single tape deterministic Turing machine
q_x	an internal state of M
t	the running time of M
U	a universal Turing machine
$U_{m,n}$	a universal Turing machine with m state and n symbols
u_x	an internal state of $U_{m,n}$
$\vdash$	a single computation step
$\vdash^s$	a sequence of s computation steps
$\vdash^*$	a sequence of $\geqslant 0$ computation steps
$\mathbb{N}$	non-negative integers
Σ^*	set of all words of length $\geqslant 0$ over alphabet $\Sigma = \{\sigma_1, \sigma_1, \ldots, \sigma_n\}$
w	a word over some alphabet Σ
$\|w\|$	the length of the word w
ϵ	the empty word, $\|\epsilon\| = 0$
ω	cardinality of $\mathbb{N}$
$\langle X \rangle$	the encoding of object X as a word
$A - B$	The set containing all the elements of set A that are not elements of set B
$A \times B$	The set of ordered pairs (a, b) such that $a \in A$ and $b \in B$

Bibliography

[Aar02] Scott Aaronson. Book review: A new kind of science. *Quantum Information and Computation*, 2(5):410–423, August 2002.

[AC87] Jürgen Albert and Karel Culik II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Systems*, 1(1):1–16, 1987.

[AF67] Stål Aanderaa and Patrick C. Fischer. The solvability of the halting problem for 2-state Post machines. *Journal of the Association for Computing Machinery (ACM)*, 14(4):677–682, 1967.

[AFR06] Artiom Alhazov, Rudolf Freund, and Yurii Rogozhin. Computational power of symport/antiport: history, advances and open problems. In Rudolf Freund, Georg Lojka, Marion Oswald, and Gheorghe Păun, editors, *Sixth international Workshop on Membrane Computing (2005)*, volume 3850 of *LNCS*, pages 1–30, Vienna, July 2006. Springer.

[AKR02] Artiom Alhazov, Manfred Kudlek, and Yurii Rogozhin. Nine universal circular Post machines. *Computer Science Journal of Moldova*, 10(3):247–262, 2002.

[AR06] Artiom Alhazov and Yurii Rogozhin. Towards a characterization of P systems with minimal symport/antiport and two membranes. In Hendrik Jan Hoogenboom, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Seventh international Workshop on Membrane Computing*, volume 4361 of *LNCS*, pages 135–153, Leiden, The Netherlands, July 2006. Springer.

[Bai98] Claudio Baiocchi. 3n+1, UTM e tag-system. Technical Report Pubblicazione 98/38, Dipartimento di Matematico, Università di Roma, 1998. (In Italian).

[Bai01] Claudio Baiocchi. Three small universal Turing machines. In Maurice Margenstern and Yurii Rogozhin, editors, *Machines, Computations, and Universality (MCU)*, volume 2055 of *LNCS*, pages 1–10, Chişinău, Moldova, May 2001. Springer.

[Ban70] Edwin R. Banks. Universality in cellular automata. In *Conference Record of the Eleventh Annual Symposium on Switching an Automata Theory (FOCS)*, pages 194–215, Santa Monica, California, 1970. IEEE.

[BCG82] Elwyn Berlekamp, John Conway, and Richard Guy. *Winning ways for your mathematical plays, vol 2: Games in particular*. Academic Press, New York, 1982.

[Ben73] Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, November 1973.

[Bra83] Allen Brady. The determination of the value of Rado's noncomputable function $\Sigma(k)$ for four-state Turing machines. *Mathematics of Computation*, 40(162):647–665, April 1983.

[CM63] John Cocke and Marvin Minsky. Universality of tag systems with $P = 2$, April 1963. AIM-52, A.I. memo 52, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts.

[CM64] John Cocke and Marvin Minsky. Universality of tag systems with $P = 2$. *Journal of the Association for Computing Machinery (ACM)*, 11(1):15–20, 1964.

[Cod68] Edgar F. Codd. *Cellular Automata*. Academic Press, New York, 1968.

[Coo66] Stephen Cook. The solvability of the derivability problem for one-normal systems. *Journal of the Association for Computing Machinery (ACM)*, 13(2):233–225, 1966.

[Coo04] Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[CVMVV07] Erzsébet Csuhaj-Varjú, Maurice Margenstern, György Vaszil, and Sergey Verlan. On small universal antiport P systems. *Theoretical Computer Science*, 372(2-3):152–164, 2007.

[Dav56] Martin Davis. A note on universal Turing machines. *Automata Studies, Annals of Mathematics Studies*, 34:167–175, 1956.

[Dav57] Martin Davis. The definition of universal Turing machine. *Proceedings of the American Mathematical Society*, 8(6):1125–1126, dec 1957.

[Dav58] Martin Davis. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.

[Dav65] Martin Davis. *The undecidable: basic papers on undecidable propositions, unsolvable problems and computable functions*. Raven Press, New York, 1965.

[De Mol07] Liesbeth De Mol. Study of the limits of solvability in tag systems. In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality (MCU)*, volume 4664 of *LNCS*, pages 170–181, Orléans, France, September 2007. Springer.

[DK89] Volker Diekert and Manfred Kudlek. Small deterministic Turing machines. *Papers on Automata and Languages, Department of Mathematics, Karl Marx University of Economics, Budapest*, 1988-4:77–87, 1989.

[DKB04] Jean-Charles Delvenne, Petr Kurka, and Vincent Blondel. Computational universality in symbolic dynamical systems. In Maurice Margenstern, editor, *Machines, Computations, and Universality (MCU)*, volume 3354 of *LNCS*, pages 104–115, Saint Petersburg, Russia, September 2004. Springer.

[Fis65] Patrick C. Fischer. On formalisms for Turing machines. *Journal of the Association for Computing Machinery (ACM)*, 12(4):570–580, 1965.

[FMKY00] Claudio Ferretti, Giancarlo Mauri, Satoshi Kobayashi, and Takashi Yokomori. On the universality of Post and splicing systems. *Theoretical Computer Science*, 231(2):157–170, January 2000.

[FO06] Rudolf Freund and Marion Oswald. Small universal antiport P systems and universal multiset grammars. In Carmen Graciani Díaz, Gheorghe Păun, Alvaro Romero-Jiménez, and Fernando Sancho-Caparrini, editors, *Fourth Brainstorming Week on Membrane Computing (Volume II)*, pages 51–64, Sevilla, Spain, 2006.

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to parallel computation:* P *-completeness theory*. Oxford University Press, Oxford, 1995.

[Hea87] Tom Head. Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviours. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

[Her66a] Gabor T Hermann. The halting problem of generalized one state Turing machines. Master's thesis, University of California, Berkeley, 1966.

[Her66b] Gabor T Hermann. On the impossibility of a one state universal Turing machines. Technical report, Electronics Research Laboratory, University of California, Berkeley, 1966.

[Her68a] Gabor T Hermann. The halting problem of one state Turing machines with n-dimensional tape. *Zeitshrift fur Mathematische Logik-und Grundlagen der Matematik*, 14(2):185–191, 1968.

[Her68b] Gabor T Hermann. Simulation of one abstract computing machine by another. *Communications of the Association for Computing Machinery (ACM)*, 11(12):802 and 813, 1968.

[Her68c] Gabor T Hermann. The uniform halting problem for generalized one state Turing machines. In *Proceedings, Ninth Annual Symposium on Switching and Automata Theory (FOCS)*, pages 368–372, Schenectady, New York, October 1968. IEEE Computer Society Press.

[HM03] Fracine Herrmann and Maurice Margenstern. A universal cellular automaton in the hyperbolic plane. *Theoretical Computer Science*, 296(2):327–364, 2003.

[HM05] Tero Harju and Maurice Margenstern. Splicing systems for universal Turing machines. volume 3384 of *LNCS*, pages 149–158, Milan, Italy, 2005. Springer, Heiedlberg.

[Hoo63] Philip Hooper. Some small, multitape universal Turing machines. Technical report, Computation Labortory, Harvard University, Cambridge, Massachusetts, 1963.

[Hoo69] Philip Hooper. Some small, multitape universal Turing machines. *Information Sciences*, 1(2):205–215, 1969.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, Massachusetts, 1979.

[IIM07] Katsunobu Iami, Chuzo Iwanmoto, and Kenichi Morita. A five-state von Neumann neighbor universal hyperbolic cellular automaton. *Journal of Cellular Automata*, 1(4):275–297, 2007.

[Ike58] N Ikeno. A 6-symbol 10-state universal Turing machine. In *Proceedings, Institute of Electrical Communications Tokyo*, 1958.

[Ind95] Piotr Indyk. Optimal simulation of automata by neural nets. In Ernst Mayr and Claude Puech, editors, *Twelfth Annual Symposium on Theoretical Aspects of Computer Science*, volume 900, pages 337–348, 1995.

[IPY06] Mihai Ionescu, Gheorghe Păun, and T Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, 2006.

[KB77] Hans Kleine-Büning. *Über probleme bei homogener Parkettierung von $\mathbb{Z} \times \mathbb{Z}$ durch Mealy-automaten bei normierter verwendung.* PhD thesis, Institut für Mathematische Logik, Münster, 1977.

[KBO77] Hans Kleine-Büning and Thomas Ottmann. Kleine universelle mehrdimensionale Turingmaschinen. *Elektronische Informationsverarbeitung und Kybernetik*, 13(4-5):179–201, 1977.

[KCG94] Pascal Koiran, Michel Cosnard, and Max H. Garzon. Computability with low dimensional dynamical systems. *Theoretical Computer Science*, 132(1-2):113–128, 1994.

[Kor96] Ivan Korec. Small universal register machines. *Theoretical Computer Science*, 168(2):267–301, 1996.

[KR01a] Manfred Kudlek and Yurii Rogozhin. New small universal circular Post machines. In Rusins Freivalds, editor, *Fundamentals of Computation Theory (FCT)*, volume 2138 of *LNCS*, pages 217–227, Riga, Latvia, August 2001. Springer.

[KR01b] Manfred Kudlek and Yurii Rogozhin. Small universal circular Post machines. *Computer Science Journal of Moldova*, 9(1):34–52, 2001.

[KR02] Manfred Kudlek and Yurii Rogozhin. A universal Turing machine with 3 states and 9 symbols. In Werner Kuich, Grzegorz Rozenberg, and Arto Salomaa, editors, *Developments in Language Theory (DLT) 2001*, volume 2295 of *LNCS*, pages 311–318, Vienna, May 2002. Springer.

[KR03] Manfred Kudlek and Yurii Rogozhin. Small universal Turing and circular Post machines. *Pure Mathematics and Applications*, 13(1-2):197–210, 2003.

[Kry67] Y Kryukov. Turing machines with two symbols and three states. *Algebra i Logka*, 16(3):54–60, 1967.

[Kry71] Y Kryukov. Turing machines with three states and two symbols and with one state and n symbols. *Kibernetika*, 1:12–13, 1971.

[KS96] Joe Kilian and Hava Siegelmann. The dynamic universality of sigmoidal neural networks. *Information and Computation*, 128(1):48–56, 1996.

[Kud96] Manfred Kudlek. Small deterministic Turing machines. *Theoretical Computer Science*, 168(2):241–255, 1996.

[LN90] Kristian Lindgren and Mats G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.

[LP07] Gregory Lafitte and Christophe Papazian. The fabric of small Turing machines. In S. Barry Cooper, Thomas Kent, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World (CIE 2007)*, Universitá Degli Studi di Siena, Dipartimento di Scienze Matematiche ed Informariche, Technical Report number 487, pages 219–227, Siena, Italy, June 2007. Springer.

[LR65] Shen Lin and Tibor Rado. Computer studies of Turing machine problems. *Journal of the Association for Computing Machinery (ACM)*, 12(2):196–212, 1965.

[Mar92] Maurice Margenstern. Sur la frontière entre machines de Turing á arrêt décidable et machines de Turing universelles. Technical Report 92-83, LITP Institut Blaise Pascal, 1992.

[Mar93] Maurice Margenstern. Non-erasing Turing machines: A frontier between a decidable halting problem and universality. In Zoltán Ésik, editor, *FCT*, volume 710 of *LNCS*, pages 375–385, Szeged, Hungry, August 1993. Springer, Heidelberg.

[Mar94] Maurice Margenstern. Une machine de Turing universelle sur {0,1}, non-effaçante et à trois instructions gauches. Technical Report 94-08, LITP Institut Blaise Pascal, 1994.

[Mar95a] Maurice Margenstern. Non-erasing Turing machines: A new frontier between a decidable halting problem and universality. In Ricardo A. Baeza-Yates, Patricio V. Poblete, and Eric Goles, editors, *LATIN*, volume 911 of *LNCS*, pages 386–397, Valparaíso, Chile, 1995. Springer, Heidelberg.

[Mar95b] Maurice Margenstern. Une machine de Turing universelle non-effaçante à exactement trois instructions gauches. *CRAS, Paris*, 320(I):101–106, 1995.

[Mar97a] Maurice Margenstern. Decidability and undecidability of the halting problem on Turing machines, a survey. In Sergei Adian and Anil Nerode, editors, *Logical Foundations of Computer Science (LFCS)*, volume 1234 of *LNCS*, pages 226–236, Yaroslav, Russia, July 1997. Springer.

[Mar97b] Maurice Margenstern. The laterality problem for non-erasing Turing machines on {0,1} is completely solved. *Theoretical Informatics and Applications*, 31(2):159–204, 1997.

[Mar98] Maurice Margenstern. Frontier between decidability and undecidability: a survey. In Maurice Margenstern, editor, *Machines, Computations, and Universality (MCU) volume 1*, pages 141–177, France, May 1998. IUT, Metz.

[Mar00] Maurice Margenstern. Frontier between decidability and undecidability: a survey. *Theoretical Computer Science*, 231(2):217–251, 2000.

[Mar01] Maurice Margenstern. On quasi-unilateral universal Turing machines. *Theoretical Computer Science*, 257(1–2):153–166, 2001.

[Mar06] Maurice Margenstern. An algorithm for building intrinsically universal cellular automata in hyperbolic spaces. In Hamid R. Arabnia and Mark Murgin, editors, *Proceedings of the 2006 International Conference on Foundation of Computer Science*, pages 3–9, Las Vegas, Nevada, June 2006. CSREA Press.

[MB90] Heiner Marxen and Jürgen Butrock. Attacking the Busy Beaver 5. *Bulletin of the European Association for Theoretical Computer Science*, 40:247–251, 1990.

[MF05] Daniel Miller and Edward Fredkin. Two-state, reversible, universal cellular automata in three dimensions. In *2nd Conference on Computing Frontiers*, pages 45–51, Ischia, Italy, 2005. ACM Press.

[MH89] Kenichi Morita and Masateru Harao. Computation universality of one-dimensional reversible (injective) cellular automata. *The Transactions of the IEICE Japan*, E72(6):758–762, 1989.

[Mic93] Pascal Michel. Busy beaver competition and Collatz-like problems. *Archive Mathematical Logic*, 32(5):351–367, 1993.

[Mic04] Pascal Michel. Small Turing machines and the generalized busy beaver competition. *Theoretical Computer Science*, 326:45–56, 2004.

[Min60a] Marvin Minsky. A 6-symbol 7-state universal Turing machines. Technical Report 54-G-027, Licoln Laboratory, MIT, Cambridge, Massachusetts, August 1960.

[Min60b] Marvin Minsky. Recursive unsolvability of Post's tag problem. Technical Report 54 G-023, Licoln Laboratory, MIT, Cambridge, Massachusetts, June 1960.

[Min61] Marvin Minsky. Recursive unsolvability of Post's problem of tag and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

[Min62a] Marvin 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. American Mathematical Society.

[Min62b] Marvin Minsky. Universality of (p=2) tag systems and a 4 symbol 7 state universal Turing machine, 1962. AIM-33, A.I. memo 33, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, Massachusetts.

[Min67] Marvin Minsky. *Computation, finite and infinite machines.* Prentice-Hall, Englewood Cliffs, New Jersey, 1967.

[Moo52] Edgar F. Moore. A simplified universal Turing machine. In *ACM national meeting*, pages 50–54, Toronto, Canada, 1952. ACM Press.

[Moo97a] Cristopher Moore. Majority-vote cellular automata, Ising dynamics, and P-completeness. *Journal of Statistical Physics*, 88(3-4):795–805, 1997.

[Moo97b] Cristopher Moore. Quasi-linear cellular automata. *Physica D*, 103:100–132, 1997.

[Moo98] Cristopher Moore. Predicting non-linear cellular automata quickly by decomposing them into linear ones. *Physica D*, 111:27–41, 1998.

[Mor96] Kenichi Morita. Universality of a reversible two-counter machine. *Theoretical Computer Science*, 168(2):303–320, November 1996.

[MP43] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP95a] Maurice Margenstern and Liudmila Pavlotskaya. Deux machines de Turing universelles á au plus deux instructions gauches. Technical Report I, CRAS, Paris, 1995.

[MP95b] Maurice Margenstern and Liudmila Pavlotskaya. Vers ue nouvelle approche de l'universalité concernant les machines de Turing. Technical Report 95-58, LITP Institut Blaise Pascal, 1995.

[MP03] Maurice Margenstern and Liudmila Pavlotskaya. On the optimal number of instructions for universality of Turing machines connected with a finite automaton. *International Journal of Algebra and Computation*, 13(2):133–202, 2003.

[MR02] Maurice Margenstern and Yurii Rogozhin. A universal time-varying distributed H system of degree 1. In Natasa Jonoska and Nadrain C. Seeman, editors, *DNA-7(2001)*, volume 2340 of *LNCS*, pages 371–380, Tampa, Florida, 2002. Springer.

[MSG89] Kenichi Morita, Akihiko Shirasaki, and Yoshifumi Gono. A 1-tape 2-symbol reversible Turing machine. *The Transactions of the IEICE Japan*, E72(3):223–228, 1989.

[MY07] Kenichi Morita and Yoshikazu Yamaguchi. A universal reversible Turing machine. In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality (MCU)*, volume 4664 of *LNCS*, pages 90–98, Orléans, France, September 2007. Springer.

[Nea06] Turlough Neary. Small polynomial time universal Turing machines. In Ted Hurley, A. Seda, et al., editors, 4^{th} *Irish Conference on the Mathematical Foundations of Computer Science and Information Technology(MFCSIT)*, pages 325–329, Cork, Ireland, August 2006.

[Nea08a] Turlough Neary. On the computational complexity of spiking neural P systems. In *Unconventional Computation, 7th International Conference, UC 2008*, volume 5204 of *LNCS*, pages 189–205, Vienna, August 2008. Springer.

[Nea08b] Turlough Neary. A small universal spiking neural P system. In *International Workshop on Computing with Biomolecules*, pages 65–74, Vienna, August 2008. Austrian Computer Society.

[Noz69] A. Nozaki. On the notion of universality of Turing machine. *Kybernetika Academia Praha*, 5(1):29–43, 1969.

[NPRP06] Hitesh Nagda, Andrei Păun, and Alfonso Rodríguez-Patón. P systems with symport/antiport and time. In Hendrik Jan Hoogenboom, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Seventh international Workshop on Membrane Computing*, volume 4361 of *LNCS*, pages 463–476, Leiden, The Netherlands, 2006.

[NW] Turlough Neary and Damien Woods. Four small universal Turing machines. *Fundamenta Informaticae*. (Accepted).

[NW05a] Turlough Neary and Damien Woods. A small fast universal Turing machine. Technical Report NUIM-CS-TR-2005-12, Department of Computer Science, National university of Ireland, Maynooth, 2005.

[NW05b] Turlough Neary and Damien Woods. Small fast universal Turing machines. Technical Report NUIM-CS-TR-2005-11, Department of Computer Science, National university of Ireland, Maynooth, 2005.

[NW06a] Turlough Neary and Damien Woods. P-completeness of cellular automaton Rule 110. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *International Colloquium on Automata Languages and Programing 2006, (ICALP) Part I*, volume 4051 of *LNCS*, pages 132–143, Venice, July 2006. Springer.

[NW06b] Turlough Neary and Damien Woods. P-completeness of cellular automaton Rule 110. Technical Report 04/2006, Boole Centre for Research in Informatics, University College Cork, Ireland, 2006.

[NW06c] Turlough Neary and Damien Woods. Small fast universal Turing machines. *Theoretical Computer Science*, 362(1–3):171–195, October 2006.

[NW07a] Turlough Neary and Damien Woods. Four small universal Turing machines. In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality (MCU)*, volume 4664 of *LNCS*, pages 242–254, Orléans, France, September 2007. Springer.

[NW07b] Turlough Neary and Damien Woods. Small weakly universal Turing machines. Technical Report arXiv:0707.4489v1 [cs.CC], July 2007.

[Oll02] Nicolas Ollinger. The quest for small universal cellular automata. In Peter Widmayer et al., editor, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2380 of *LNCS*, pages 318–329, Malaga, Spain, July 2002. Springer.

[Ott75a] Thomas Ottmann. Eine universelle Turingmaschine mit zweidimensionalem band. *Elektronische Informationsverarbeitung und Kybernetik*, 11(1-2):27–38, 1975. (In German).

[Ott75b] Thomas Ottmann. Einfache universelle mehrdimensionale Turingmaschinen. Habilitationsschrift, Karlsruhe, 1975.

[Pap95] Christos Papadimitriou. *Computational complexity*. Addison-Wesley, 1995.

[Pău00] Gheorghe Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[Pău02] Gheorghe Păun. *Membrane Computing: An Introduction*. Springer, 2002.

[Pav73] Liudmila Pavlotskaya. Solvability of the halting problem for certain classes of Turing machines. *Mathematical Notes (Springer)*, 13(6):537–541, June 1973. (Translated from Matematicheskie Zametki, Vol. 13, No. 6, pages 899–909, June, 1973.).

[Pav75] Liudmila Pavlotskaya. O minimal'nom chisle razlichnykh kodov vershin v grafe universal'noj mashiny T'juringa. *Disketnyj analiz, Sbornik trudov instituta matematiki SO AN SSSR*, 27:52–60, 1975. (On the minimal number of distinct codes for the vertices of the graph of a universal Turing machine. In Russian).

[Pav78] Liudmila Pavlotskaya. Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. *Problemi Kiber-*

netiki, 27:91–118, 1978. (Sufficient conditions for the halting problem decidability of Turing machines. In Russian).

[Pav96] Liudmila Pavlotskaya. On machines, universal by extensions. *Theoretical Computer Science*, 168(2):257–266, 1996.

[Pav02] Liudmila Pavlotskaya. Turing machines connected to the undecidability of the halting problem. *Mathematical Notes*, 71(5):667–675, 2002.

[Pol87] J Pollack. *On connectionist models of natural language processing*. PhD thesis, Computer Science Department, University of Illinois, Urbana, 1987.

[Pos36] Emil Post. Finite combinatory processes. formulation I. *The Journal of Symbolic Logic*, 1(3):103–105, sep 1936. (Also to be found in [Dav65] pages 289–291).

[Pos43] Emil Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathmatics*, 65(2):197–215, April 1943.

[Pos47] Emil Post. Recursive unsolvability of a problem of Thue. *The Journal of Symbolic Logic*, 12(1):1–11, March 1947. (Also to be found in [Dav65] pages 293–303).

[Pos65] Emil Post. Absolutely unsolvable problems and relatively undecidable propositions - account of an anticipation. In Martin Davis, editor, *The undecidable: basic papers on undecidable propositions, unsolvable problems and computable functions*, pages 340–406. Raven Press, New York, 1965.

[PP07] Andrei Păun and Gheorghe Păun. Small universal spiking neural P systems. *BioSystems*, 90(1):48–60, 2007.

[Pri79] Lutz Priese. Towards a precise characterization of the complexity of universal and non-universal Turing machines. *Siam journal of Computing*, 8(4):508–523, November 1979.

[PRS98] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA Computing: New Computing Paradigms*. Springer, 1998.

[Rad62] Tibor Rado. On non-computable functions. *The Bell Systems Technical Journal*, 41(3):877–884, 1962.

[Ric06] Gaétan Richard. A particular universal cellular automaton. HAL research report (oai:hal.archives-ouvertes.fr:hal-00095821_v1), September 2006.

[Rob71] Raphael Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.

[Rob91] Raphael Robinson. Minsky's small universal Turing machine. *International Journal of Mathematics*, 2(5):551–562, 1991.

[Rog79] Yurii Rogozhin. Sem' universal'nykh mashin T'juringa. In *Fifth all union conference on Mathematical Logic, Akad. Naul SSSR. Otdel. Inst. Mat., Novosibirsk*, page 27, 1979. (Seven universal Turing machines. In Russian).

[Rog82] Yurii Rogozhin. Sem' universal'nykh mashin T'juringa. *Systems and theoretical programming, Matematicheskie Issledovanija*, 69:76–90, 1982. (Seven universal Turing machines. In Russian).

[Rog92] Yurii Rogozhin. Universal'naja mashina T'juringa s 10 sostojanijami i 3 simvolami. *Izvestiya Akademii Nauk Respubliki Moldova, Matematika*, 4(10):80–82, 1992. (A universal Turing machine with 10 states and 3 symbols. In Russian).

[Rog93] Yurii Rogozhin. About Shannon's problem for Turing machines. *Computer Science Journal of Moldova*, 1(3):108–111, 1993.

[Rog96] Yurii Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996.

[Rog98] Yurii Rogozhin. A universal Turing machine with 22 states and 2 symbols. *Romanian Journal of Information Science and Technology*, 1(3):259–265, 1998. (In Russian).

[RV06] Yurii Rogozhin and Sergey Verlan. On the rule complexity of universal tissue P systems. In Rudolf Freund, Georg Lojka, Marion Oswald, and Gheorghe Păun, editors, *Sixth international Workshop on Membrane Computing(2005)*, volume 3850 of *LNCS*, pages 356–362, Vienna, July 2006. Springer, Heidelberg.

[Sha56] Claude Elwood Shannon. A universal Turing machine with two internal states. *Automata Studies, Annals of Mathematics Studies*, 34:157–165, 1956.

[Sie98] Hava Siegelmann. *Neural networks and analog computation: beyond the Turing limit*. Birkhauser, Boston, 1998.

[Sip97] Micheal Sipser. *Introduction to the theory of computation*. PWS, Boston, 1997.

[SM99] Halva Siegelmann and Maurice Margenstern. Nine switch-affine neurons suffice for Turing universality. *Neural Networks*, 12(4-5):593–600, 1999.

[Smi71] Alvy Smith III. Simple computation-universal cellular spaces. *Journal of the Association for Computing Machinery (ACM)*, 18(3):339–353, July 1971.

[SS91] Hava Siegelmann and Eduardo Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[SS95] Hava Siegelmann and Eduardo Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

[Sut03] K Sutner. Almost periodic configurations on linear cellular automata. *Fundamenta Informaticae*, 58(3-4):223–240, 2003.

[Tak58] H Takahashi. Keisankikai II. *Iwanami, Tokyo*, 1958. (Computing machine II. In Japanese.).

[Tof77] Tommaso Toffoli. Computation and construction universality of reversible cellular automata. *Journal of Computer and Systems Science*, 15(2):213–231, 1977.

[Tur37] Alan Turing. On computable numbers with application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1937. (Also to be found in [Dav65] pages 116–154).

[vEB90] Peter van Emde Boas. Machine models and simulations. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume A*, pages 1–66. Elsevier, 1990.

[vN66] John von Neumann. *The theory of self-reproducing automata*. University of Illinois Press, Urbana, 1966.

[Wag73] Klaus Wagner. Universelle Turingmaschinen mit n-dimensionale band. *Elektronische Informationsverarbeitung und Kybernetik*, 9(7-8):423–431, 1973.

[Wan57] Hao Wang. A variant to Turing's theory of computing machines. *Journal of the Association for Computing Machinery (ACM)*, 4(1):63–92, January 1957.

[Wan63] Hao Wang. Tag systems and lag systems. *Mathematical Annals*, 152(4):65–74, 1963.

[Wat60] Shigeru Watanabe. On a minimal universal Turing machine. Technical report, MCB Report, Tokyo, August 1960.

[Wat61] Shigeru Watanabe. 5-symbol 8-state and 5-symbol 6-state universal Turing machines. *Journal of the Association for Computing Machinery (ACM)*, 8(4):476–483, 1961.

[Wat72] Shigeru Watanabe. Four-symbol five-state universal Turing machine. *Information Processing Society of Japan Magazine*, 13(9):588–592, September 1972. (In Japanese).

[WNa] Damien Woods and Turlough Neary. The complexity of small universal Turing machines: A survey. *Theoretical Computer Science*. (Accepted).

[WNb] Damien Woods and Turlough Neary. Small semi-weakly universal Turing machines. *Fundamenta Informaticae*. (Accepted).

[WN06a] Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 439–446, Berkeley, California, October 2006. IEEE.

[WN06b] Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal Turing machines. Technical Report arXiv:cs/0612089v1 [cs.CC], December 2006.

[WN06c] Damien Woods and Turlough Neary. Remarks on the computational complexity of small universal Turing machines. In Ted Hurley, A. Seda, et al., editors, *Fourth Irish Conference on the Mathematical Foundations of Computer Science and Information Technology(MFCSIT)*, pages 334–338, Cork, Ireland, August 2006.

[WN07a] Damien Woods and Turlough Neary. The complexity of small universal Turing machines. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World, Third Conference on Computability in Europe, CIE 2007*, volume 4497 of *LNCS*, pages 791–799, Siena, Italy, June 2007. Springer.

[WN07b] Damien Woods and Turlough Neary. Small semi-weakly universal Turing machines. In Jérôme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations, and Universality (MCU)*, volume 4664 of *LNCS*, pages 303–315, Orléans, France, September 2007. Springer.

[Wol83] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983.

[Wol02] Stephen Wolfram. *A new kind of science.* Wolfram Media, 2002.

[ZZP] Xingyi Zhang, Xiangxiang Zeng, and Linqiang Pan. Smaller universal spiking neural P systems. In submission.