Simulating Turing machines using switching map systems

Turlough Neary and Damien Woods

TASS Research Group,
Department of Computer Science,
National University of Ireland, Maynooth, Ireland.
Email: `tneary@cs.may.ie`, `dwoods@cs.may.ie`
URL: `http://www.cs.may.ie/TASS/`

Communicated by: J. Paul Gibson

Date: December 21, 2003

Technical report: NUIM-CS-2003-TR-11

Key words: model of computation, unconventional model of computation, switching map system, baker's map, computability, Chomsky hierarchy, Turing machine, generalized shift.

Abstract

This report describes research carried out during a final year B.Sc. research project. The research is based on the work of Moore and Sato. A function converts Turing machines into switching baker's maps. The theory of switching baker's maps and the converter function are both explained and adapted. A switching baker's map simulator is implemented and used to recognise a number of languages in the Chomsky hierarchy. The computation path of the switching baker's map simulator is graphically represented and output patterns are examined.

1 Introduction

In 1998 Moore showed [4] that a single Turing machine (TM) [7] tape head action could be simulated by a single baker's map (BM) application. This was an observation from previous work where Moore showed the equivalence of TMs and generalized shifts [3]. In 2000 Sato and Ikegami showed that a switching map systems with only two BMs can simulate TMs, by showing how to convert an arbitrary TM into a switching map system [6]. In [6] the BMs for this switching map system are given (without any formal derivation). A number of experiments relevant to TMs and switching map systems were also carried out. The aim of this report is to explain and derive the relevant BMs, to implement a switching map system to simulate any TM, and to carry-out experiments on this switching map system that may tell us something about TMs and problem solving.

1.1 Baker's Maps

A BM is a two dimensional dynamical system (see [5]). The generalised BM is defined as a transformation on the unit square $B([0,1] \times [1,0] \to [0,1] \times [1,0])$ and is given by the following equation [5]

$$B(x,y) = \begin{cases} (\lambda_a x, y/\alpha) & \text{if } y < \alpha \\ (1-\lambda_b+\lambda_b x, (y-\alpha)/\beta) & \text{if } y \geq \alpha \ . \end{cases} \tag{1}$$

Where $\beta = 1 - \alpha$, $\lambda_a + \lambda_b \leq 1$, and x and y are coordinates on the unit square.

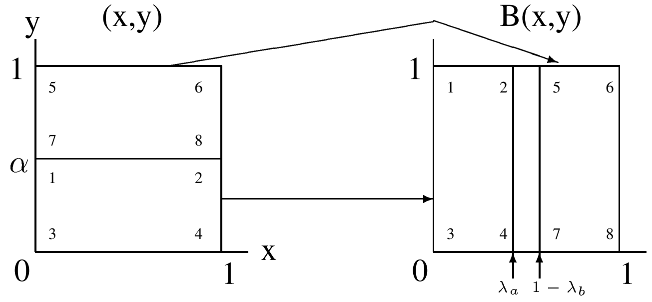


Figure 1: BM acting on the unit square.

From Eq. (1) and Fig. 1 we see that the unit square is mapped from two horizontal stripes onto two vertical stripes. The α value above defines how the domain is divided (i.e. $y < \alpha$ and $y \geq \alpha$). The λ values partition the space that is mapped to (i.e. the range is $x < \lambda_a$ and $x \geq 1 - \lambda_b$). The two stripes that are mapped to do not overlap. The BM need not be area preserving. If $\lambda_a + \lambda_b = 1$ the BM is area preserving and if $\lambda_a + \lambda_b < 1$ it is dissipative. In Figs. 1 and 2 we see an example of a dissipative BM; as we continue with each successive mapping the space dissipates. For more on BMs see [5].

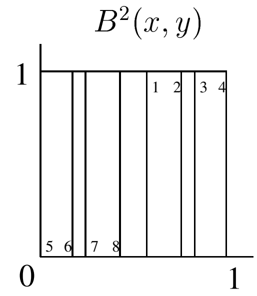


Figure 2: Result after two applications of the mapping given in Fig. 1.

2 Switching map systems and TMs

In this section the BMs given in [6] are developed. The BMs developed here have some differences from those given by [6].

2.1 Representing TM tape contents as a pair of coordinates

Take a TM that has a binary tape alphabet. The contents of the tape at any time can be represented as a coordinate on the unit square [4]. The contents stored on the tape to the left of the tape head are represented by the x-coordinate and the contents to the right of, and including, the tape head are represented by the y-coordinate.

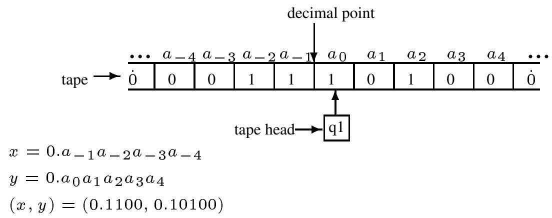


Figure 3: Representation of the tape contents of a TM as a pair of coordinates.

The decimal point is considered to be positioned immediately to the left of the tape head. If we look at the format of the coordinates on the TM tape in Fig. 3 we see that the x-coordinate is in reverse on the tape. If we shift the tape head to the left or right the values of the x- and y-coordinates may change.

2.2 Simulating a right shift

In Fig. 4 we have the result of a right shift operation, carried out on the TM configuration in Fig. 3. The new configuration of our TM represented as a coordinate on the unit square is $(0.11100, 0.0100)$. If we examine the change in the x- and y-coordinates we see that the decimal point has moved one place to the right on the y-coordinate, and one place to the left on the x-coordinate. After the right shift operation the most significant bit (MSB) of the y-coordinate (which in this case is $a_0 = 1$) becomes the MSB of the x-coordinate.

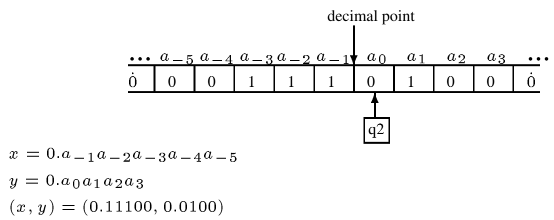


Figure 4: x- and y-coordinates after a right shift on the TM configuration given in Fig. 3.

In binary, if the decimal point is moved one place to the right it corresponds to multiplying by 2 and moving one place to the left corresponds to dividing by 2. So it would seem that a right shift (Fig. 4) on a TM tape is equivalent to a right shift on the y-coordinate and a left shift on the x-coordinate. To get this result we divide our x-coordinate by 2 and multiply our y-coordinate by 2, so we get

$$(x, y) \to (x/2, 2y)$$

However, this equation is not sufficient to cover all possibilities of the right shift operation. If it is tried on the configuration in Fig. 3 we do not get the same result as a right shift.

$$\begin{array}{ll} x = 0.1100 & x/2 = 0.01100 \\ y = 0.10100 & 2y = 1.0100 \end{array}$$

The y-coordinate is too large as we are using coordinates on the unit square (i.e. $0 \leq y < 1$) to represent the TM tape contents. If we are carrying out a right shift operation, and the MSB of the y-coordinate is 1, then the MSB must be removed (by subtracting 1). This 1 also has an effect on the x-coordinate. The MSB of the y-coordinate becomes the MSB of the x-coordinate (see Figs. 3 and 4, the bit in cell a_0 moves one place to the left with respect to the tape head and now becomes bit a_{-1}). If the MSB of y is 0 the equation above gives the correct value for the x-coordinate since dividing x by 2 will simply put a 0 in the MSB position of the

x-coordinate. If however the MSB of the y-coordinate is 1 we need to place this 1 in the MSB position of the x-coordinate in place of the 0. This is done by adding 0.1 to the x-coordinate. We now have two possibilities to cover when simulating a right shift using our coordinate system, the MSB of the y-coordinate can either be 0 or 1. If the MSB of y is 0 then $y < 1/2$ and if it is 1 then $y \geq 1/2$. So if we incorporate these changes in our equation above we have

$$(x, y) \to (x/2, 2y) \ \text{ if } \ y < 1/2$$
$$(x, y) \to (x/2 + 0.1, 2y - 1) \ \text{ if } \ y \geq 1/2$$

If we compare these values to our BM in Eq. (1) we see that $\lambda_a = 1/2, \alpha = 1/2, \beta = 1/2$ and $\lambda_b = 1/2$. The value for λ_b is 1/2 because our BM must be area preserving (BM is area preserving if $\lambda_a + \lambda_b = 1$) so that any possible tape contents of a TM with a binary tape alphabet is representable. We get our equation for a BM that simulates a right shift

$$B^{RS}(x, y) = \begin{cases} (x/2, 2y) & \text{if } y < 1/2 \\ ((x+1)/2, 2y - 1) & \text{if } y \geq 1/2 \ . \end{cases} \tag{2}$$

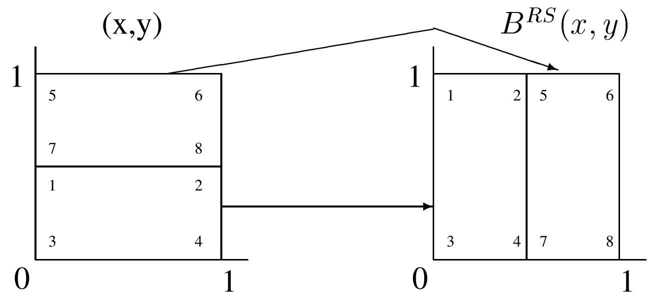


Figure 5: B^{RS} acting on the unit square.

2.3 Simulating a right shift with bit flip

If we wish to simulate a right shift and bit flip we need to modify Eq. (2). In Fig. 6 the y-coordinate remains the same as in Fig. 4 this is because the bit being flipped at position a_0 in Fig. 3 has become a_{-1} in Fig. 6. So this bit flip will not affect the y-coordinate. The x-coordinate however will be affected, the reverse of what we have seen in the right shift will be true for the right shift with bit flip (i.e. if the MSB of the y-coordinate is 0 before the shift then the MSB of the x-coordinate will be 1 after the shift and if the MSB of the y-coordinate is 1 before the shift

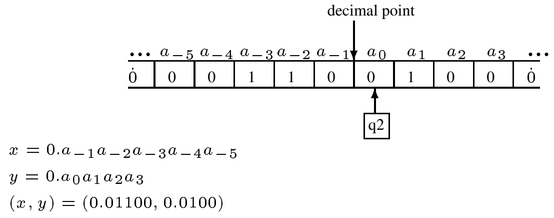


Figure 6: Result of a right shift bit flip operation on the TM configuration in Fig. 3.

then the MSB of the x-coordinate will be 0 after the shift). So we have the right shift with bit flip BM

$$B^{RSBF}(x, y) = \begin{cases} ((x+1)/2, 2y) & \text{if } y < 1/2 \\ (x/2, 2y-1) & \text{if } y \geq 1/2 \end{cases} . \tag{3}$$

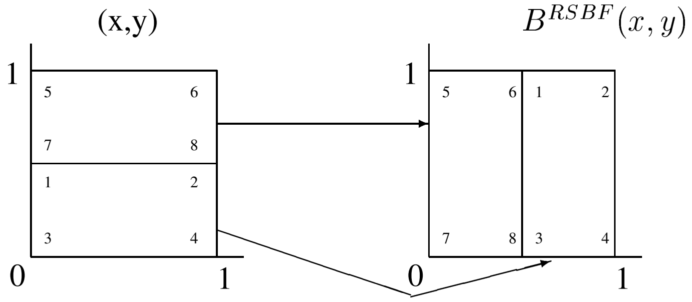


Figure 7: B^{RSBF} acting on the unit square.

2.4 Simulating a left shift

If a left shift on the TM configuration in Fig. 3 is carried out its result can be represented by $(x, y) = (0.100, 0.110100)$. We see that the decimal point moves to the left on the y-coordinate (equivalent to dividing y by 2) and moves to the right on the x-coordinate (equivalent to multiplying x by 2). This gives

$$(x, y) \to (2x, y/2) \ .$$

As before, this equation is not sufficient to represent a left shift operation since the MSB of x is 1 (a_{-1} in Fig. 3 becomes a_0). If the MSB of x is 0 before the shift this equation would be sufficient but if it is 1 we need to place this 1 in the MSB position of the y-coordinate. As we have seen before if the MSB of x is 0 then our $x < 1/2$ and if the MSB of x is 1 the $x \geq 1/2$. So we have

$$(x, y) \to (2x, y/2) \text{ if } x < 1/2$$
$$(x, y) \to (2x - 1, (y + 1)/2) \text{ if } x \geq 1/2 \ .$$

This gives the third BM equation

$$B^{LS}(x, y) = \begin{cases} (2x, y/2) & \text{if } x < 1/2 \\ (2x - 1, (y + 1)/2) & \text{if } x \geq 1/2 \ . \end{cases} \tag{4}$$

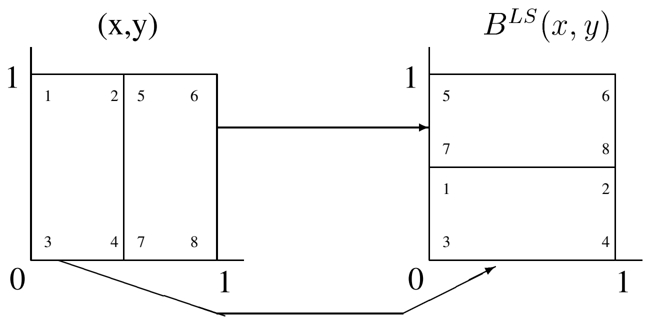


Figure 8: B^{LS} acting on the unit square.

It can be seen that the domain is divided differently for B^{LS} than for the first two BM equations (Eq. 2 and 3) that were derived. This is because we test the x value instead of the y value to see where the coordinates will map. The BMs carry out operations on both coordinates, but depending on the direction of the shift we must test the value of either one of the coordinates. This all depends on the bit that is travelling across the tape head boundary (i.e. our decimal point) to the other coordinate.

2.5 Switching map systems

A switching map (SM) system [6] consists of one or more maps (e.g. BMs, horseshoe maps). A SM system involves switching between the different mappings of the system. A single mapping (computation step) on a SM system may involve the composition of a number of different maps. Using the two BMs, B^{RS} and B^{RSBF}, here is an example of a mapping on the unit square by some (SM) system; $(x, y) \to (B^{RS}(B^{RSBF}(x, y)))$. The SM system also uses states to decide which map sequence comes next. For more on SM systems see [6].

2.5.1 Map sequences

It can be shown that for each TM there exists a SM system consisting of just two BMs that simulates the computation of the TM, step by step [6]. We will use the

BMs B^{LS} and B^{RSBF} (given by Eqs. (4) and (3) respectively), to construct our SM system. Examine the possible operations of a TM (right shift, right shift bit flip, left shift, left shift bit flip, no shift, no shift bit flip). We already have two of the six possible operations: left shift, B^{LS} (seq. 1) and right shift bit flip, B^{RSBF} (seq. 2). We can simulate the right shift operation by using the map sequence $B^{RSBF}(B^{LS}(B^{RSBF}(x,y)))$ (seq. 3). The first map simulates a right shift bit flip. The second map then simulates a left shift on this result. If we think of this in terms of a TM the tape head is back in its original position, except that the bit under the tape head (a_0) has been flipped. When the third map (B^{RSBF}) is applied, a_0 is flipped back to its original value and the right shift is simulated. A left shift bit flip can be simulated by using the map sequence $B^{LS}(B^{LS}(B^{RSBF}(x,y)))$ (seq. 4). As with seq. 3, after the first two maps have been applied the tape head can be thought of as being in its original position except that the bit a_0 is flipped. When the left shift map (B^{LS}) is then applied, we have simulated a left shift bit flip. For no shift bit flip, the sequence $B^{LS}(B^{RSBF}(x,y))$ (seq. 5) is used. This sequence is the same as applying the first two maps in seq. 3, so is the same as a no shift with the bit a_0 having been flipped. The sequence for no shift is the identity sequence (x,y) (seq. 6), it leaves the coordinates unchanged. So now each possible TM operation is simulated by one of these 6 map sequences.

2.6 Formal definition of a switching baker's map

We now give our definition of a switching baker's map (SBM). This definition has some notational differences from the definition in [6]. A SBM is a 6-tuple $(S,\ C,\ F,\ M,\ q_0,\ q_{-accept})$ where

1. S is the set of states,
2. C is the space, $C = [0,1] \times [0,1] \subset \mathbb{R} \times \mathbb{R}$,
3. $F : S \times C \to S \times C$ is the switching function,
4. M is the set of map sequences given above,
5. $q_0 \in S$ is the start state,
6. $q_{-accept} \in S$ is the accept state.

The purpose of the states in a SM system is to determine which map sequence is to be applied next. The SBM simulates a computation step of a TM using the switching function.

2.6.1 The switching function

The switching function maps from a state/space pair to a state/space pair, by one of a number of switching rules. We use a worked example to explain, this will lead to a general template for a switching rule. A switching rule contains the current state, the value of the current coordinates, the map sequence and the next state. Let $S = \{s_0, s_1, s_2, \ldots, s_{z-1}\}$ and $M = \{m_0, m_1, m_2, \ldots, m_{t-1}\}$ where S is the set of states of a SBM and M is the set of map sequences (e.g. seqs. 1–6 defined in Sect. 2.5.1) of the SBM. The SBM switching rules have the format

$$(s_i, a_0, m_k, s_j)$$

where $s_i, s_j \in S$ and $m_k \in M$ and a_0 is the MSB of the y-coordinate.

Like a TM, a SBM instance will have an initial configuration. The configuration of a SBM will be given by the (x, y) coordinate and the current state (analogous to the TM tape contents and state). The configuration will change, analogous to that of a TM, until the SBM halts (if it halts). The configuration of the SBM chnages ovedr time, starting from some initial state s_0 and continuing until some final state s_{n-1}, as follows

$$\begin{aligned}
F : (s_0, (x_0, y_0)) &\rightarrow (s_1, (x_1, y_1)) \\
F : (s_1, (x_1, y_1)) &\rightarrow (s_2, (x_2, y_2)) \\
&\vdots \\
F : (s_{n-2}, (x_{n-2}, y_{n-2})) &\rightarrow (s_{n-1}, (x_{n-1}, y_{n-1}))
\end{aligned}$$

In general the last state in this sequence may or may not be an accepting state. At each configuration the switching function F determines the next configuration by choosing a switching rule based on the current state and the current coordinate value. The current MSB of the y-coordinate (a_0) is the only bit of the current coordinate that is actually used to decide which switching rule (if one exists) is chosen. The direct relationship can be seen between the operation of a TM and a SBM as it is the read symbol of the TM (which is in the corresponding position to a_0, see Fig. 3) that decides which transition rule is applied next. Each map sequence simulates a single TM operation, which gives a linear time simulation of the TM by the SBM.

2.7 Converting a TM table of behaviour into a SBM table of behaviour

To convert some TM to a SBM we must convert the set of transition rules that describe the operation of the TM to a set of switching rules that describe the operation of a SBM. Take the transition rule below

$$\text{(initial state, read symbol, write symbol, shift direction, destination state)}$$
$$(s_i, 1, 0, R, s_j) \ .$$

This transition rule right shifts, bit flips, and changes from state s_i to state s_j. To convert this to a switching rule we use the map sequence for right shift bit flip, B^{RSBF} (seq. 2 from Sect. 2.5.1). There is no need to change the state names, we can use the same states for the switching rule. As seen earlier in Fig. 3, the bit a_0 will be the same as the symbol under the tape head, in this case it is 1. So our conversion looks as follows

$$(s_i, 1, 0, R, s_j) \to (s_i, 1, B^{RSBF}(x, y), s_j) \ .$$

In Fig. 9 we take all the possible transition rules from a fixed state s_i to a fixed state s_j, using a tape alphabet of $\{0, 1\}$ and the possible head movements of $\{$L, R, N$\}$, and give their representation as switching rules.

$$\begin{array}{l}
\Phi(s_i, 1, 0, R, s_j) \to (s_i, 1, B^{RSBF}(x, y), s_j) \\
\Phi(s_i, 1, 1, R, s_j) \to (s_i, 1, B^{RSBF}(B^{LS}(B^{RSBF}(x, y))), s_j) \\
\Phi(s_i, 0, 0, R, s_j) \to (s_i, 0, B^{RSBF}(B^{LS}(B^{RSBF}(x, y))), s_j) \\
\Phi(s_i, 0, 1, R, s_j) \to (s_i, 0, B^{RSBF}(x, y), s_j) \\
\Phi(s_i, 1, 0, L, s_j) \to (s_i, 1, B^{LS}(B^{LS}(B^{RSBF}(x, y))), s_j) \\
\Phi(s_i, 1, 1, L, s_j) \to (s_i, 1, B^{LS}(x, y), s_j) \\
\Phi(s_i, 0, 0, L, s_j) \to (s_i, 0, B^{LS}(x, y), s_j) \\
\Phi(s_i, 0, 1, L, s_j) \to (s_i, 0, B^{LS}(B^{LS}(B^{RSBF}(x, y))), s_j) \\
\Phi(s_i, 1, 0, N, s_j) \to (s_i, 1, B^{LS}(B^{RSBF}(x, y)), s_j) \\
\Phi(s_i, 1, 1, N, s_j) \to (s_i, 1, (x, y), s_j) \\
\Phi(s_i, 0, 0, N, s_j) \to (s_i, 0, (x, y), s_j) \\
\Phi(s_i, 0, 1, N, s_j) \to (s_i, 0, B^{LS}(B^{RSBF}(x, y)), s_j)
\end{array}$$

Figure 9: The function Φ maps each TM transition rule to a unique SBM rule.

Using Φ (see Fig. 9, taken from [6]) it is possible to convert any table of behaviour of a binary TM into a set of switching rules for a SBM. As we have seen earlier, the state names map directly across (from transition rule to switching rule) without being changed, the read symbol also maps directly across and is interpreted as a numerical digit (i.e. a_0). The only thing left to do when applying Φ is to find the correct map sequence. As seen in Sect. 2.5.1 there are only 6 possible map sequences that can be chosen. To find the correct map sequence two things in our transition rule need to be checked, first is there a bit flip (i.e. is the read symbol the same as the write symbol), second what is the direction of the shift. The Φ function will check both and pick the correct map sequence for the corresponding switching rule. We have now shown that any TM can be converted into a SBM.

2.8 TM and SBM equivalence

The following is an explanation of a property of Sato's [6] Φ function. We have explained how Φ maps from any binary TM table of behaviour to a SBM table of behaviour. If we restrict SBMs to use only finitely described rationals[1] and only the 6 map sequences defined in this paper (see Sect. 2.5.1), then Φ is bijective. Hence, given such a restricted SBM we can use Φ^{-1} to get a TM that simulates the SBM. In practice, it is simply a matter of examining the map sequence and the value a_0, and converting these into a shift direction and write symbol, for each switching rule. Also, the read symbol is given by a_0 and the state names need not be changed. Hence, the SBM (using finitely described rationals and only these six map sequences) and the TM are equivalent models, in terms of computational power.

In fact any SBM that is defined on a rational space and uses map sequences that are definable solely in terms of finite sequences of the two maps (Eqs. 3 and 4) is equivalent in terms of computational power to the TM model. Any finite composition of these two maps can be simulated by a finite number of computational steps on a TM. Using such map sequences in the SBM will have an effect on the computational complexity of the TM computation (i.e. a time and/or space increase), but not on computational power. In fact, the time complexity of the TM will be O(n), where n is the time complexity of the SBM.

2.9 Generalized Shifts

Generalized shifts (GSs) were introduced by [2, 3] as a generalisation of shift maps such as the BM and horseshoe map. A GS is a mapping from a bi-infinite sequence a to a bi-infinite sequence $\psi(a)$. As the name suggests, the mapping involves a shift of the sequence. This shift occurs with respect to a decimal point which is at a given location in the sequence. In this light we can compare a GS sequence to a TM tape with head (in that a TM has a bi-infinite tape with the tape head separating both sides of the sequence). The mapping may also change the symbols in a finite subsequence of the sequence. Let the $A^{\mathbb{Z}}$ be set of all bi-infinite sequences over a finite alphabet Σ. The operation of the GS is determined by two

[1] In this paper we are assuming that TMs have a single input tape, and at each computation step the tape contains a finite length word from $\{0,1\}^*$, followed immediately by an infinite string of 0s. Obviously there is a straightforward bijective mapping between such TM tapes and standard TMs tapes that make use of an explicit blank symbol. In terms of computability we wish to adhere strictly to the standard TM model, and hence we say the SBM space is rational. When we say that the coordinates must be finitely describable rationals we mean that they are of the form $(x, y) = (0.x'000\ldots, 0.y'000\ldots)$ where $x', y' \in \{0,1\}^*$, and x', y' are both interpreted as numerical strings.

functions F and G. $F : A^{\mathbb{Z}} \to \mathbb{Z}$ maps to some integer value that specifies the direction and magnitude of the shift. $G : A^{\mathbb{Z}} \to \{A \cup \phi\}^{\mathbb{Z}}$ maps to a finite sequence over Σ that is both appended and prepended by an infinite sequence of null (ϕ) symbols. G specifies an overwriting of the symbols in a finite subsequence of the sequence a. Both of these functions are dependent on a finite subsequence known as the domain of dependence (DoD) . The locations in sequence a that are modified by G are known as the domain of effect (DoE).

A GS ψ (Moore [3] calls it Φ) is then

$$\psi : a \to \sigma^{F(a)}(a \oplus G(a)) \ ,$$

where $\sigma^{F(a)}$ shifts the sequence by the integer value $F(a)$, and $a \oplus G(a)$ overwrites the DoE of a using $G(a)$ in the following way

$$(a \oplus G(a))_i = \begin{cases} G(a)_i & \text{if } G(a)_i \neq \phi \\ a_i & \text{if } G(a)_i = \phi \ . \end{cases}$$

Here $i \in \mathbb{Z}$ and as usual w_i is the i^{th} symbol in w. Moore [3] showed that GSs simulate TMs. A GS's behaviour is defined by its G and its F functions. For a given TM instance the next configuration is dependent on the current state and the current read symbol. When simulating a TM, a GS must (under some suitable encoding) examine both the current state and the read symbol. This is achieved by incorporating both into the DoD. The number of bits needed for the DoD depends on the alphabets of the TM and GS (but for simplicity lets say that both are using binary). If, for example, our TM has 8 states the DoD would read 4 locations in the sequence (3 bits for the state as there are 2^3 possible states, and 1 bit for the read symbol). The location of the DoE will vary depending on the shift direction, since $\sigma^{F(A)}$ is applied after the G function and we require our state and read symbol to occupy the same locations on the tape after each shift. It is clear that any TM can be simulated by a GS. It is also possible to go the other direction and convert a GS (that is defined on finite sequences) into a TM. Each step of the GS (i.e. applying $a \oplus G$ and then σ^F) can be simulated by a finite number of TM steps. The number of states for the TM will depend on the size of the DoD. For more on GSs and their relationship to TMs see [3].

3 Results

A number of different TMs were implemented. The TMs recognised languages from different levels of the Chomsky hierarchy [1]. Table 1 details the different

Language level	Name	Language definition
Regular	isEven	$L = \{w \mid w \in \{1,0\}^* \text{ \& } w_{\lvert w \rvert} = 0\}$
Regular	Fibonnaci	$L = \{w \mid w \in \{1,0\}^* \text{ \& } w_i = 1 \Rightarrow w_{i+1} = 0\}$
Context free	$a^n b^n$	$L = \{w \mid w = 1^i 0^i \text{ \& } i \in \mathbb{N}\}$
Context free	wcw^R	$L = \{wcw^R \mid w \in \{1,0\}^*$ & $w^R =$ the reverse of w & c is a symbol$\}$
Context free	Dyck	$L = \{w \mid w \in \{1,0\}^*$ & for each 0 there must be a unique 1 and this 1 must occur before the 0 & $\#1\text{s} = \#0\text{s}\}$
Context sensitive	wcw	$L = \{wcw \mid w \in \{1,0\}^*$ & c is a symbol$\}$
Context sensitive	prime	$L = \{w \mid w \in \{1\}^* \text{ \& } \frac{\lvert w \rvert}{i} \notin \mathbb{N} \text{ \& } 1 < i < \lvert w \rvert\}$
Recursively Enumerable†	threeX (or 3x+1)	$L = \{x \mid x \in \{1\}^* \text{ \& } f^n(x) = 1 \text{ \& } n \in \mathbb{N}\}$, where $f(x) = \begin{cases} 3\lvert x \rvert + 1, & \text{if } x \text{ is odd} \\ \frac{\lvert x \rvert}{2}, & \text{if } x \text{ is even} \end{cases}$

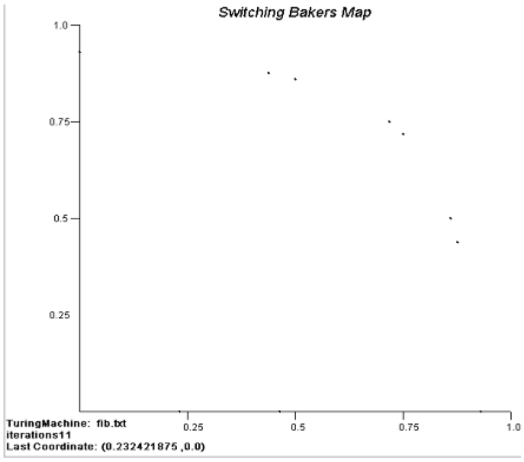
Table 1: Languages for which TMs were written and their respective levels in the Chomsky hierarchy, w_j is the j^{th} symbol in w. †Conway has shown that a generalisation of the 3x+1 problem is undecidable, it is unknown whether the standard version of the problem that we use is decidable or not.

languages that were examined. Each TM was run with a number of different inputs. Each TM, on each input, was also simulated by a Java SBM simulator. The output of the SBM simulator was then graphed.

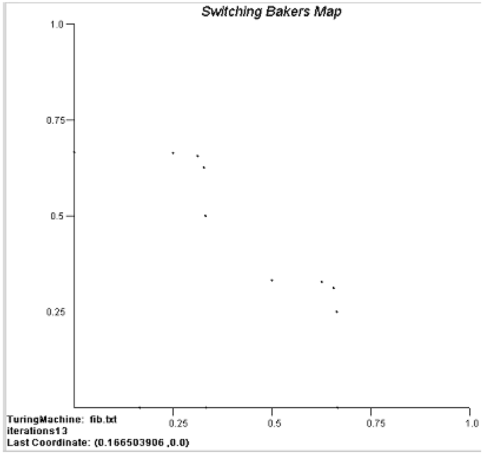
3.1 Graphed results

This section contains a selection of graphical outputs from our Java SBM simulator. In the lower left hand corner of each graph is the name of the simulated TM, the number of iterations, and the last coordinates when the SBM simulator finished its computation. The language recognised by each TM and the encodings for each TM are given before the relevant graphs, $\bar{z}$ denotes the encoding of z and b is the blank symbol. All final states are accepting states unless specified otherwise.

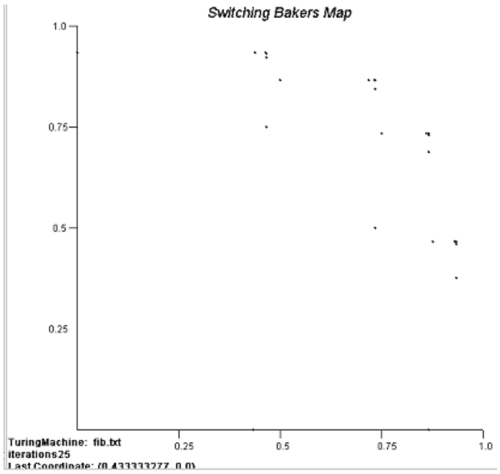
Graphs: 1 to 4. Language: Fibonnaci. Encoding: $\bar{1} = 11$, $\bar{0} = 10$ and $\bar{b} = 00$.



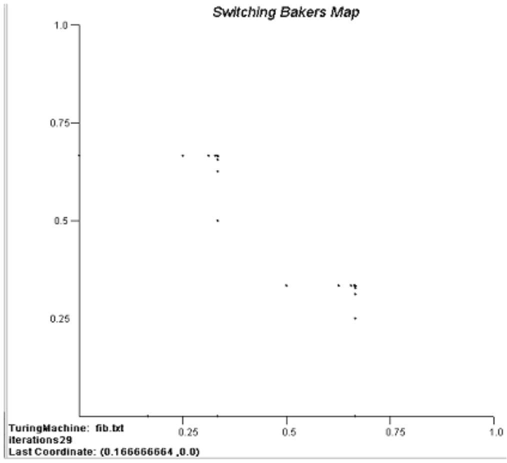
Graph 1. Output for the input representing 1010.



Graph 2. Output for the input representing 0^5.

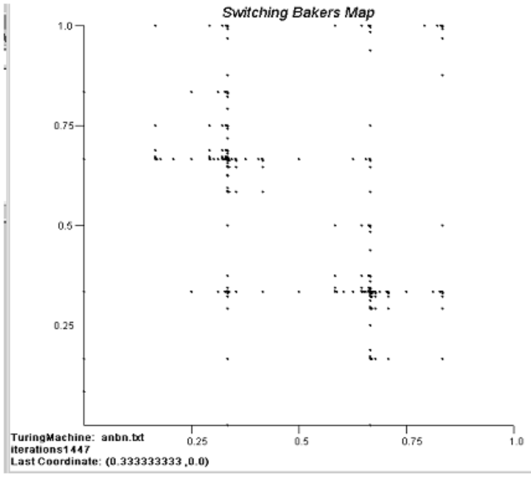


Graph 3. Output for the input representing 10101010101.



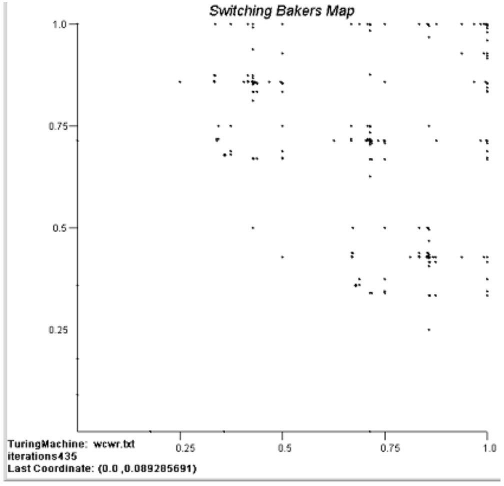
Graph 4. Output for the input representing 0^{13}.

Graph: 5. Language: $a^n b^n$. Encoding: $\bar{1} = 11$, $\bar{0} = 10$, $\bar{x} = 01$ and $\bar{b} = 00$.

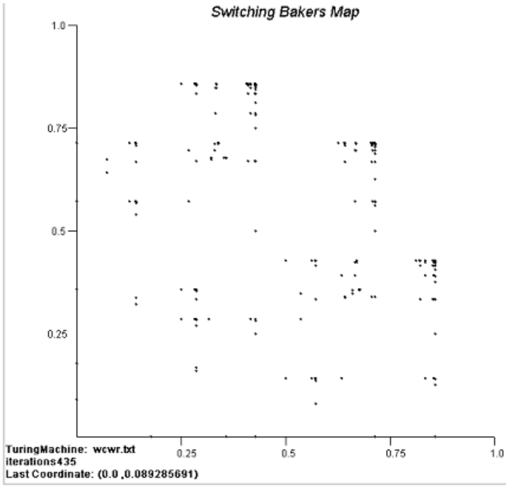


Graph 5. Output for the input representing $0^{18}1^{18}$.

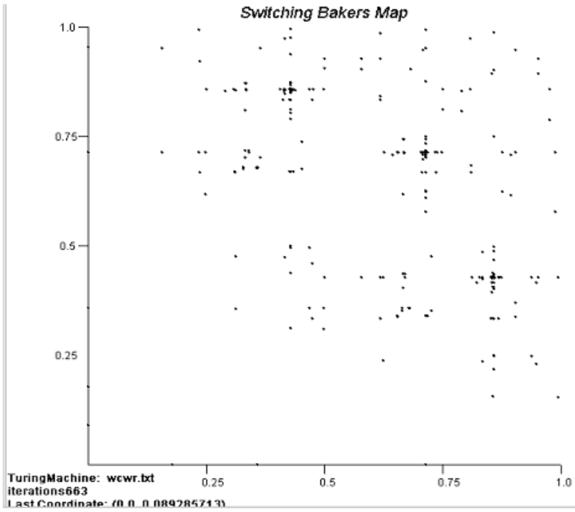
Graphs: 6 to 8. Language: wcw^r. Encoding: $\bar{1} = 111, \bar{0} = 100, \bar{c} = 010, \bar{x} = 101$ and $\bar{b} = 000$.



Graph 6. Output for the input representing 1^7c1^7.

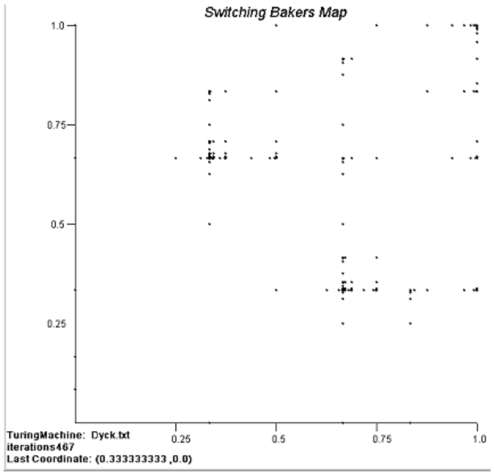


Graph 7. Output for the input representing 0^7c0^7.

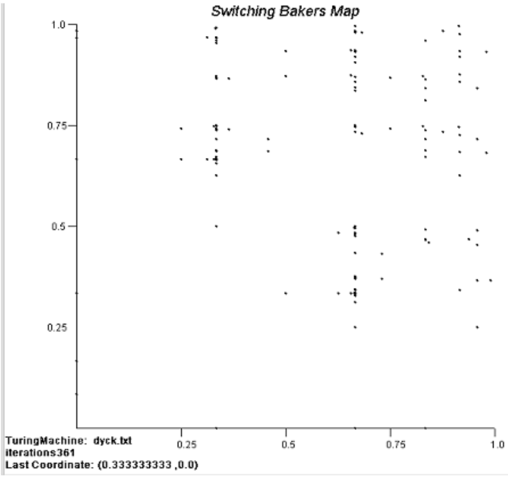


Graph 8. Output for the input representing 101100101c101001101.

Graphs: 9 and 10. Language: Dyck. Encoding: $\bar{1} = 11$, $\bar{0} = 10$, $\bar{x} = 01$ and $\bar{b} = 00$.

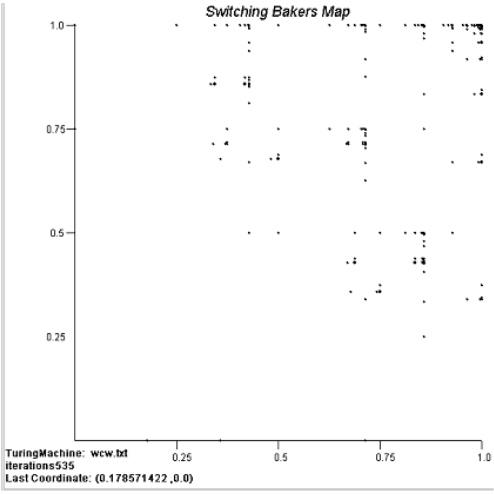


Graph 9. Output for the input representing $1^{10}0^{10}$.

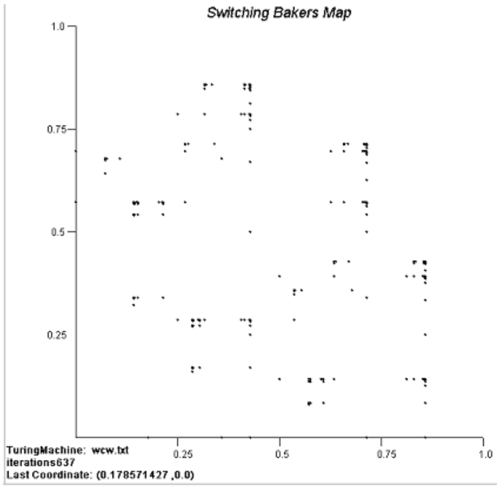


Graph 10. Output for the input representing 11010110010010101101 00.

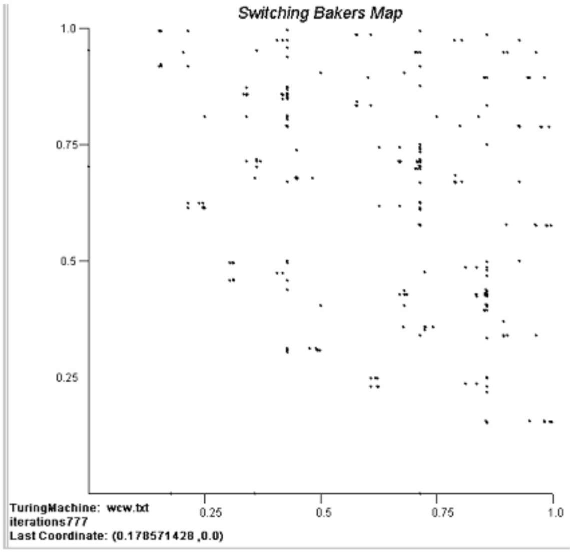
Graphs: 11 to 13. Language: wcw. Encoding: $\bar{0} = 100$, $\bar{1} = 111$, $\bar{x} = 101$, $\bar{c} = 010$ and $\bar{b} = 000$.



Graph 11. Output for the input representing 1^8c1^8.

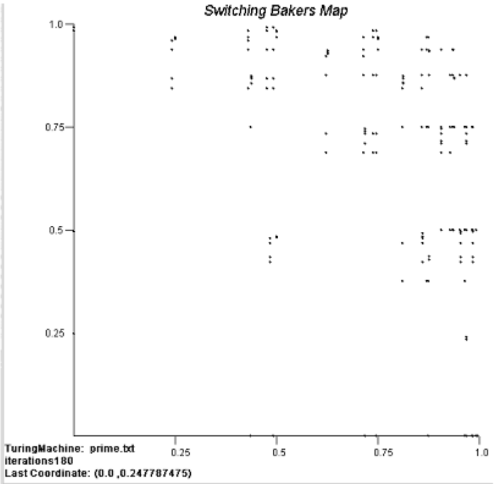


Graph 12. Output for the input representing 0^9c0^9.

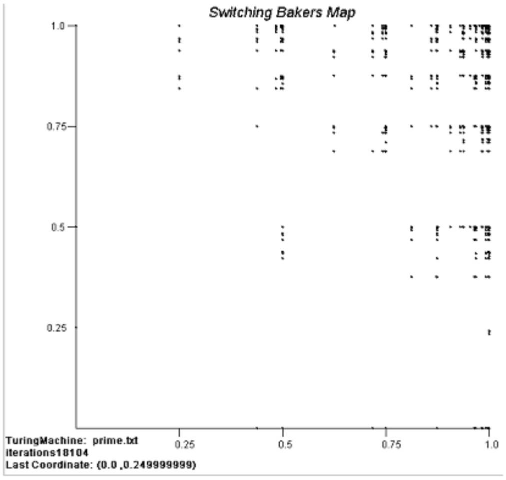


Graph 13. Output for the input representing 1010011001c1010011001.

Graphs: 14 and 15. Language: prime. Encoding: $\bar{1} = 11$, $\bar{x} = 10$, $\bar{y} = 01$ and $\bar{b} = 00$.

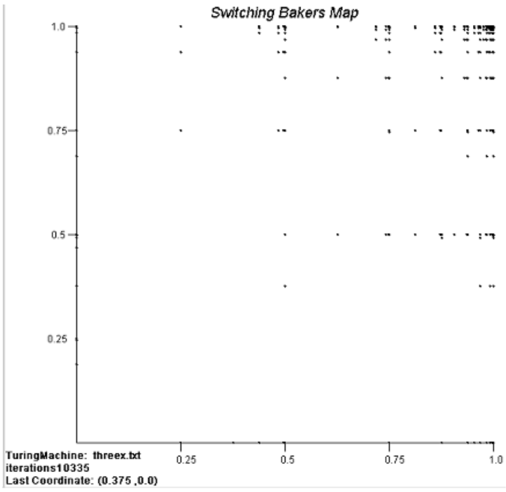


Graph 14. Output for the input representing 1^3.

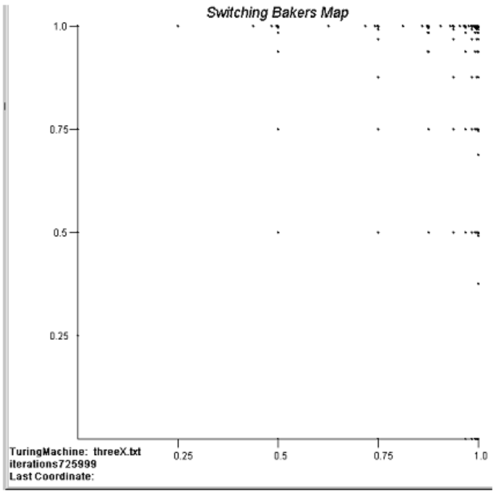


Graph 15. Output for the input representing 1^{17}.

Graphs: 16 and 17. Language: threeX. Encoding: $\bar{1} = 11$, $\bar{x} = 10$ and $\bar{b} = 00$.



Graph 16. Output for the input representing 1^{17}.



Graph 17. Output for the input representing 1^{54}. This computation did not halt.

3.2 Examination of results

In the first four graphs we see output from a Fibonnaci language recogniser. Since this language is regular the tape head of the simulated TM is constantly moving right across the input (similar to the action of a finite state automata, meaning that no bits need be flipped or read twice). Hence, in the SBM the value of the x-coordinate will increase and the value of the y-coordinate will decrease over time. In general, if the number of 0s in the input increases then the coordinates will have smaller values. For smaller inputs there seems to be no patterns, this is because there is not enough iterations of the SBM and hence not enough points plotted for a clear pattern to emerge. For larger inputs there are sharp corners (see graph 4) where the coordinates change from larger y values to smaller y values and from smaller x values to larger x values. This is because with large inputs when the tape head is in the centre of the input there is a slowing down of the change in magnitude of both the x- and y-coordinates. Any regular language recogniser will begin with $x = 0$ and finish with $y = 0$ (assuming the word is accepted). Even if the input contains many 0s the coordinates will still converge on sharp corners before the computation finishes. If we look at graph 4 we see two well defined corners at $(0.\dot{6}, 0.\dot{3})$ and $(0.\dot{3}, 0.\dot{6})$. Due to the characteristic property of the Fibonnaci language and the encoding used, the following properties hold for the input to graph 4: (a) if the tape head is over a 0 then the MSB of x is 1 and the MSB of y is 0, (b) if the tape head is over a 1 then the MSB of x is 0 and the MSB of y is 1. So in terms of our SBM we see that throughout the computation the coordinates bounce between these two corners. The graphs for only one regular language are given since examining these four graphs also explains behaviour exhibited in the other regular language graphs. The only difference between the Fibonnaci language and the isEven language is that the isEven language can accept long strings of consecutive 1s provided they end with a 0. This means that in such cases we have graphs with just one sharp corner (i.e. converging at point $(1, 1)$).

In the remaining graphs (5–17) the coordinates seem to be covering areas of the unit square that are not covered in the regular language graphs. Again on giving larger inputs the coordinates seem to converge on sharp corners. Most of the TMs will just change 1 or 2 symbols with each scan of the input. Again we see the positions of the coordinates are affected by the inputs. Some of this behaviour can be easily explained. If we look at graphs 6 and 7 we see that again there are points of convergence but graph 6 covers higher areas of the unit square than graph 7. This is because the input for graph 6 contains more 1s than the input for graph 7. We can see that the two graphs have some common areas of convergence (i.e. $(0.4, 0.85)$, $(0.7, 0.7)$, $(0.85, 0.4)$) this is because as the computation progresses they will tend towards the same values (tape contents). Again we see these common areas of convergence in graph 8. In graphs 15–17 the top right corner of the unit square contains the majority of the coordinates plotted. This is because the threeX and prime TMs both use unary encodings to represent input so computation will mainly involve scanning through long strings of 1s. In graph 14 there are no sharp corners, this is simply because the TM tape contents remains too short for the patterns seen in graph 15 to emerge.

In many of the graphs a number of patterns comprised of a series of square shapes appear. In graphs 5, 9, 16 and 17 we see more distinctive patterns. They seem to have symmetry around the line $x - y = 0$. All these graphs share a common feature, that is their SBM coordinates all contained a large unbroken string of 1s for a period of time during their computation. As mentioned earlier, when simulating the scanning of a TM over a long string of 1s we get sharp corners. These TMs scan over their tape contents a number of times altering the contents at a very slow rate, usually just one or two bit flips for each scan. If we look at what happens when we scan over a string of 1s on our TM and see what this looks like in terms of our SBM graphs the patterns make more sense. Initially x is 0 and as the number of 1s in the input increases the y value approaches 1. If we examine the x-coordinate's progression we see that with each right shift on the string of 1s it gets larger. For simplicity we take the y-coordinate to be 1 until x itself gets very close to 1. We end up with a progression as follows $(0, 1) \rightarrow (0.5, 1) \rightarrow (0.75, 1) \rightarrow (0.875, 1) \rightarrow (0.9375, 1) \rightarrow \ldots \rightarrow (1, 0.9375) \rightarrow (1, 0.875) \rightarrow (1, 0.75) \rightarrow (1, 0.5) \rightarrow (1, 0)$. It can be easily seen that this traces out a right angle with a convergence (corner) at the point (1, 1). When x is increasing the change in the x-coordinate value over time where $t > 0$ is given by $x_{t+1} = x_t + \frac{x_t}{2}$, where $t = 0$ is the first timestep. This means the increase in the x-coordinates value is halved with each iteration (the opposite is true for y decreasing, $y_{t+1} = y_t - 2(y_{t-1} - y_t)$). The sequence of coordinates described above will be reversed when scanning in the left direction over the tape again.

If a very small portion of the input is changed during a scan of the input we can see, from the sequence above, why the symmetry about the line $x - y = 0$ occurs. However in most cases 0s will appear at different places in the coordinates during a computation. Over many iterations and changing of 1s to 0s, this gives rise to many series of points, each series seemingly converging to a corner. This gives the impression of right angles, which in turn gives the impression of squares of decreasing size (decreasing by multiples of 2, see graphs 4, 9, 16 and 17). In many cases we have a single 0 in the center of a string of 1s when we move towards it and then away from it we will get symmetric points on either side of the line $x - y = 0$ similar to what we have seen above (e.g. graphs 16 and 17).

Notice that graphs 5 and 9 use the same TM tape encoding and that the input is of a similar structure (graph 5's input is a string of 10s followed by a string of 11s and graph 9's input is a string of 11s followed by a string of 10s). We have seen that scanning a string of 10s leads to sharp corners around 2 points. We have also seen how symmetry occurs about the line $x - y = 0$ when scanning through a symmetric string that contains a large number of 1s. This symmetry can also be considered true of long substrings that are symmetric, as the influence of the rest of the string is greatly reduced when we are in the centre part of a long substring. Looking at the encodings for graphs 5 and 9 we see that as the computation continues our encodings of 1s and 0s will be replaced with xs. So we have two different inputs tending towards similar tape contents (string of xs). In terms of SBM output, this leads to other symmetries evolving about the line $x - y = 0$ that are similar in the two graphs.

3.3 Conclusions

A number of patterns emerged that were common to many of the languages from different levels of the Chomsky hierarchy. It would seem that these patterns do not give any new information into the computational process of the TM recognisers of these languages. The appearance of specific graphical properties is explainable in terms of the encoding and dynamic TM and tape contents over time. For instance symmetry as mentioned above will always play a role since TMs of languages that are not regular will almost certainly scan their input strings more than once [4]. The encoding will also play an important part in what patterns emerge. If we take graphs for the threeX (graphs 16 and 17) and prime (graph 15) TMs we see that their SBM outputs cover similar areas of the unit square. Both of these SBMs take unary words as input and have similar computation cycles. The TMs for the Dyck and $a^n b^n$ languages also used identical encodings and when the input was similar the patterns that emerged were similar (graphs 5 and 9).

So can these graphs tell us anything about TM computation? From above, by looking at the graphs it may be possible to distinguish which graphs are of TMs that have similar encodings and take similar input. They may also be used to classify languages in new ways by looking at the links between emerging patterns and computational aspects of these languages. We could also plot the spatial coordinates against time and see how the computation cycle changes for non-regular languages (e.g. we would see the slower movement of the tape head away from its initial position due to scanning back and forth). One point to note when looking at the threeX language is that it may our may not halt on certain inputs. If it does not halt then there are a number of possibilities. One possibility is that it enters a looping state (i.e. a finite set of elements which does not include the members of $L = \{w|w \in \{1\}^* \ \& \ |w| = 2^i \ \& \ i \in \mathbb{N}\}$). The second is that it keeps getting bigger in which case as we scan back and forth and the coordinates approach $(1, 1)$, the length of the string of 1s tends to infinity. This would be spatially represented as an ever increasing number of progressively smaller squares. However, since the problem is undecidable, the SBM may unpredictably switch out of this behaviour at any time. This poses the question: given an undecidable problem P, can we use information from SBM outputs to classify whether certain instances (or subsets) of P will terminate?

A SBM graph does not include all elements of a simulated TM's dynamic configuration. The graphs give the tape contents and tape head's position, but TM state information is not included. We could use the GS mentioned in Sect. 2.9 and plot the left and right parts of the bi-infinite sequence as coordinates, similar to the method used for the SBM. This would graphically represent the dynamics of a TM's computation more accurately. How would the output look? In the GS the space is divided into a series of rectangles (the DoD divides the unit square into a series rectangles) because the states are incorporated into the DoD this means each rectangle will correspond to a particular state (and read symbol). Using the TMs implemented in this study, the computation continues until an accepting state is reached or the TM halts. An accepting state would correspond to a rectangle that had no mappings to other rectangles (states). Hence in an output graph a rectangular section of the unit square would contain either zero or one point. In general, plotting state and tape contents will result in more complex plots. Patterns that are different, but analogous, to those in our SBM simulations will form. However, incorporating state information may lead to interesting dynamical classifications of TM computations.

Acknowledgments

We wish to acknowledge Dr. J. Paul Gibson for his helpful comments on a draft of this report.

References

[1] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 1979.

[2] Cristopher Moore. Undecidability and unpredictability in dynamical systems. *Physical Review Letters*, 64(20):2354–2357, May 1990.

[3] Cristopher Moore. Generalized shifts: undecidability and unpredictability in dynamical systems. *Nonlinearity*, 4:199–230, 1991.

[4] Cristopher Moore. Finite-dimensional analog computers: Flows, maps, and recurrent neural networks. In *Proceedings of the 1st International Conference on Unconventional Models of Computation*, Berlin, January 1998. Springer-Verlag.

[5] Edward Ott. *Chaos in dynamical systems*. Cambridge University Press, Cambridge, 1993.

[6] Yuzuru Sato and Takashi Ikegami. Computation with switching map systems. *Journal of Universal Computer Science*, 6(9):881–889, September 2000.

[7] Micheal Sipser. *Introduction to the theory of computation*. PWS, Boston, 1997.